Hardware Implementation of Double precision Floating-Point Reciprocator on FPGA

Naga valli.Vegesna
Dept of ECE
Shri Vishnu Engineering College for women
E-mail:valli.vegesna@gmail.com

V.Srinivasa Rao
Dept of ECE
Shri Vishnu Engineering College for women
E-mail:vemu1974@gmail.com

***Abstract*—Floating point division is generally regarded as a low frequency, high latency operation in typical floating point applications. So due to this not much development had taken place in this field. But nowadays floating point divider has become most important in many modern applications. Most of the previous implementations required larger area and latencies. In this paper we have presented an efficient FPGA implementation of a reciprocator for double-precision floating point numbers. The method is based on the use of small look-up tables and partial block multipliers. The modules occupy less area and less latency.**

***Keywords:* Double precision, Floating-point arithmetic, reciprocator, partial block-multipliers, FPGA.**

I. INTRODUCTION

Floating point arithmetic like addition, multiplication, division and square root etc are mostly used in modern applications. Mainly in scientific and signal processing applications. The greater dynamic range and lack of need to scale the numbers makes development of algorithms much easier. Implementing of arithmetic operations for floating point numbers in hardware is very difficult. Among the operations division is most difficult to implement in hardware. So to make floating point division simple we are going for binomial expansion method.

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 for faster and better implementations.

There are many algorithms to implement division. Many algorithms were developed for division which includes subtractive method, functional iterations, Digit recurrence method, seed architecture which uses multipliers and algorithms for faster computation of division like high radix algorithm. But these algorithms uses huge look up tables along with wider multipliers which affects the area and performance.

Our approach focuses on finding the reciprocal. It is based on the well known binomial-expansion, contains small look-up table, and uses partial block-multipliers, resulting in Less area, less delay. We are taking only normalized numbers. All the exceptional cases are detected, and indicated as invalid input/output. When compared with other methods binomial expansion methods have efficient hardware.

We are using Xilinx ISE synthesis tool, ModelSim 6.4c simulation tool, and FPGA as our platform.

II. APPROACH

The format of a floating-point number is as follows in fig:1 for Single Precision and Double Precision:

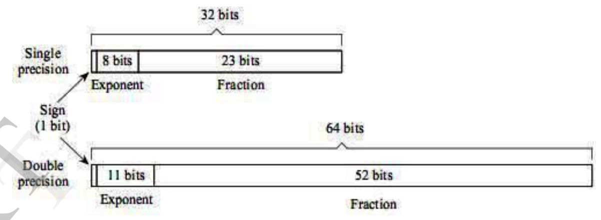


Fig:1 format for single and double precision

In this paper, we do not discuss the exponent manipulation as it is a standard process. The benefits of our implementation are in the computation of the inverse of the mantissa.

Let y be the inverse of the mantissa *a*. Then,

$$\frac{1}{X} = 1.a$$, *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

$$X=1/(a_1+a_2)$$

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

$$=a_1^{-1}-a_2^{-2}.a_2+a_1^{-3}.a_2^{2}-a_1^{-4}a_3 \text{---------} (1)$$

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

$$a_1^{-1} = 0.\overbrace{xxxxxxxx}^{\text{full significant bits}}$$

$$a_1^{-2}.a_2 = 0.\overbrace{00\cdots00}^{m-\text{zero bits}}\overbrace{xx\cdots xx}^{\text{significant bits}}$$

$$a_1^{-3}.a_2^{2} = 0.\overbrace{00\cdots00}^{2m-\text{zero bits}}\overbrace{xx\cdots xx}^{\text{significant bits}}$$

$$a_1^{-4}.a_2^{3} = 0.\overbrace{00\cdots00}^{3m-\text{zero bits}}\overbrace{xx\cdots xx}^{\text{significant bits}}$$

$\cdots$ *and so on*

. *Where m is the number of bits of a1.*

If we go for higher terms, contribution to main results decreases. Thus, depending upon our precision number of terms can be taken from equation (1) for calculating inverse, based on value of *m*.

For our 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 we have taken the first three terms, while for the case of double-precision 7 terms have been taken. The value of *m* we have chosen is 8 for both cases. These values were selected based on available FPGA. We have simplified the desired terms in such a way so that we can use less hardware with low latency and good accuracy.

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

$$Y=a_1^{-1}-a_1^{-2}a_2+a_1^{-3}a_2^{2}$$

$$y = a_1^{-1}-a_1^{-1}[(a_1^{-1}a_2-a_1^{-2}a_2^{2})\,(1+a_1^{-2}a_2^{2}+a_1^{-4}a_2^{4})]\text{-------}(2)$$

The above equation can be little more simplified but it affects 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

An algorithm for single precision and double precision floating point reciprocal is implemented using Binomial Expansion method which contains small look up tables, and partial block multipliers, resulting in less area, less delay. We have shown the implementations for single precision and double precision separately as different issues arise in each case.

First we convert the required decimal number in to IEEE 754 floating point number using IEEE 754 decimal to floating point converter and mantissa is inverted.

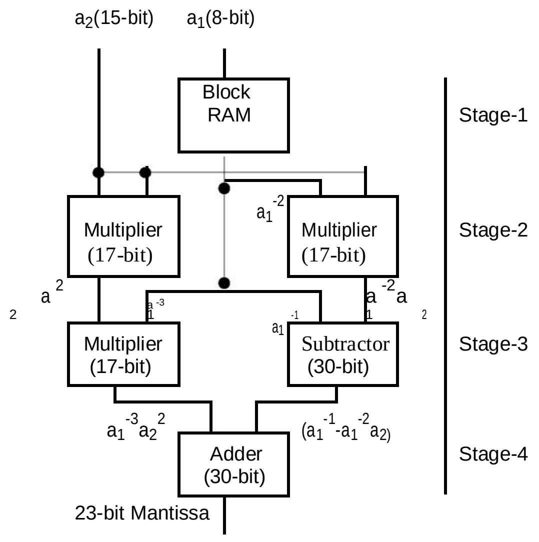


Fig. 2. Architecture for single-precision floating-point reciprocator

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 float 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 (LSB's)) 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. In both cases we will use only a single BRAM on FPGA, so we prefer the first approach.

The architecture has latency of four, though we can include the BRAM access in the first stage with a slight loss in maximum operating frequency. By using pipelined multiplier we can approximately double the overall frequency. We have shown the result with the latency four. Our aim here is to only show the use of less necessary hardware. We can do pipelining in the given architecture very easily.

B. *Double-precision Floating-point*

The architecture of double-precision floating-point reciprocator is shown in Fig. 2. It also includes a single 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. Here we have a huge saving on block-memory compared to other methods discussed later.

There are three type of multiplier (based on Xilinx MULT18x18 block) that have been used. Second, third and

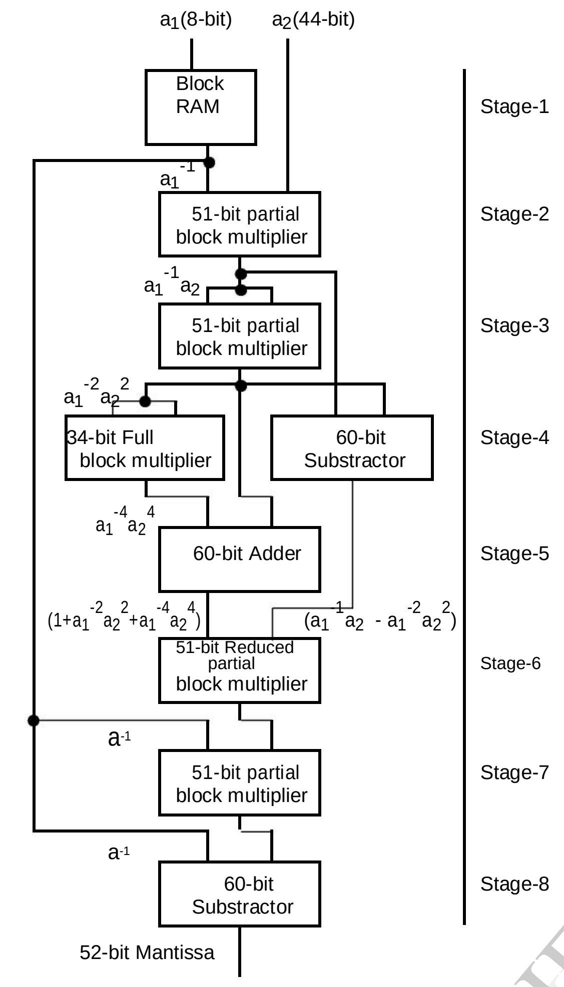


Fig. 3. Architecture for double-precision floating-point reciprocator

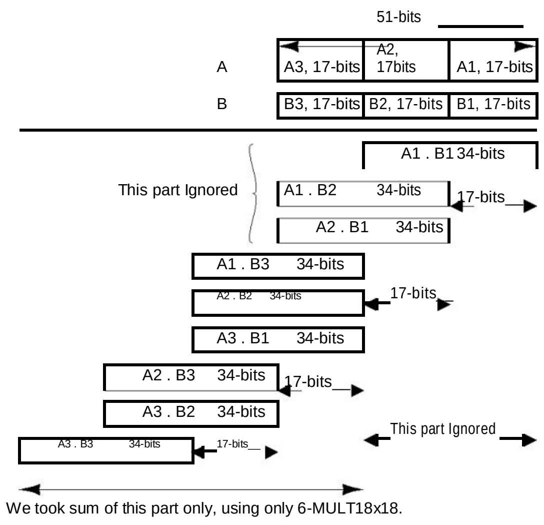


Fig. 4. Partial 51-bit multiplier for stages 2,3 and 7

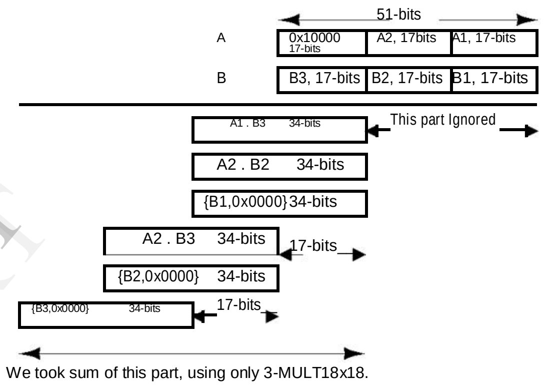


Fig. 5. Partial 51-bit multiplier for stage 6

seventh stage has 51-bit partial multiplier, which is shown in Fig. 3. It uses only six-MULT18x18 block instead of nine, to produce more than 52-bit (MSB) of correct result, which is all that we need. Stage six is also a 51-bit partial multiplier, but due to it's specific input nature (17-bits of first input is 0x10000 in hex), it contains only three-MULT18x18 block Fig. 4. The fourth stage multiplier is a 34-bit full multiplier, but instead of using IP-core for it we have designed it using four MULT18x18 block (shown in Fig. 5) which is taking less (about 2/3) glue logic and is faster than the IP-core available from Xilinx. Overall latency of module is eight, which we can increase further using pipelining as discussed in the case of single precision, for better performance.

IV. RESULTS

Hardware utilization and performance of both the single-precision and double-precision is shown in Table-I. Since our implementation neglects some of the lower order bits in the computation, it is important to estimate the impact of this on the overall accuracy of results. For the error performance 5-

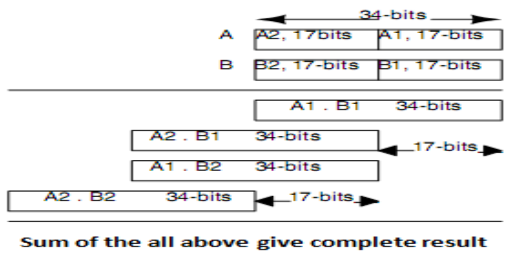


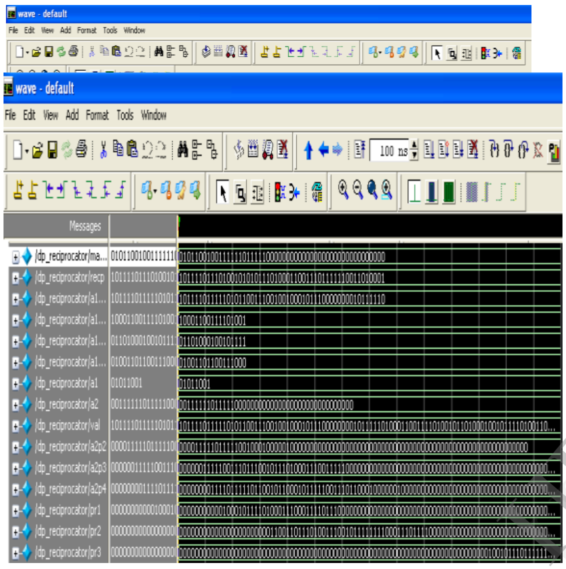
Fig. 6. 34-bit block multiplier for stage 4

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 last place), while in the case of double precision numbers, it was 1 ulp. The error we got is without rounding.

V.CONCLUSION

We have implemented an efficient reciprocal unit on FPGA for both single and double precision floating-point numbers. The method uses the idea of neglecting higher order terms in the partial block multiplication to reduce the number of multipliers. At the same time, the look-up table requirements are kept to a minimum, and are the least reported in the literature for double precision implementation. Initial latency for our module is also less (4 for single and 8 for double-precision), that too with promising frequency, which we can improve by pipelining them very easily. The error performance is also within acceptable range (1-ulp for double-precision).

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.



```
/dp_reciprocator/ma...    0101100100111111  0101100100111111011111000000000000000000000000000000
/dp_reciprocator/recp     1011110111010010  1011110111010010101011101000110011101111110011010001
/dp_reciprocator/a1...    1011110111110101  1011110111110101100111001001000101110000000010111110
/dp_reciprocator/a1...    1000110011110100  10001100111101001
/dp_reciprocator/a1...    0110100010010111  01101000100101111
/dp_reciprocator/a1...    0100110110011100  01001101100111000
/dp_reciprocator/a1       01011001          01011001
/dp_reciprocator/a2       0011111101111110  0011111101111110000000000000000000000000000
/dp_reciprocator/val      1011110111110101  10111101111101011001110010010001011100000000101111101000110011110100101101000100101111010011 0...
/dp_reciprocator/a2p2     0000111110111110  0000111110111110010001000001000000000000000000000000000000000000000000000000000000000000
/dp_reciprocator/a2p3     0000001111100111  00000011111001110111001011101000111001111100000000000000000000000000000000000000000000000...
/dp_reciprocator/a2p4     0000000011110111  00000000111101111101100101100010111110011101100010000001000000000000000000000000000000000...
/dp_reciprocator/pr1      0000000000010001  00000000000100010111101000110001110111000000000000000000000000000000000000000000000000000...
/dp_reciprocator/pr2      0000000000000000  00000000000000000000000110011011101001110010111111110001101110000000000000000000000000000...
/dp_reciprocator/pr3      0000000000000000  00000000000000000000000000000000000000000000000000000000000000000000000000000100101110111111...
```

SINGLE PRECISION:
DOUBLE PRECISION

Resource Comparison:

Method	Single-precision		Double-precision	
	MULT18x18	BRAM	MULT18x18	BRAM
2-NR	6	1	20	28
[2]	14	2	36	2
[5][10]	12	1	48	29
[6][13]	4	12	16	impractical
[7]	7	1	10	30
[9]	8	1	32	50
[12]	-	impractical	-	impractical
Proposed Method	3	1	25	1

VI.REFERENCES

[1] K. Scott Hemmert and Keith D. Underwood "Floating Point Divider Design for FPGAs", IEEE Transaction on very large scale integration systems,vol. 15, No. 1, pp. 115-118,Jan 2007.

[2] Mohamed anane, Hamid Bessalah ,Mohamed Issad, Nadjia Anane and Hassen Salhi "Higher radix and redundancy factor for floating point SRT Division", IEEE Transaction on very large scale integration systems, vol. 16, no. 16, pp. 122-128,June 2008.

[3] Anuja Jayraj Thakkar and Abdel Ejnioui " Pipelining of Double Precision Floating Point Divider and Square Root Operations Proceedings of the 44th annual southeast regional conference, March 2006.

[4] Anuja Jayraj Thakkar and Abdel Ejnioui " Design and Implementation of Double Precision Floating Point Divider And Square Root Operations On FPGAs ,"IEEE Conference on field programmable technology,2006

[5]P. Hung, H. Fahmy, O. Mencer, M. J. Flynn, "Fast division algorithm with a small look-up table", *33th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, USA., Vol-2, Pages 1465-1468, Oct-1999.

[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]Xiaojun Wang, B. E. Nelson, "Tradeoffs ofdesigning floating-point division and square root on Virtex FPGAs", *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2003)*, Pages 195- 203, Apr-2003.

[8] J. Hopf, "A parameterizable HandelC divider generator for FPGAs with embedded hardware multipliers", *IEEE International Conference on Field-Programmable Technology*, Pages 355-358, Dec-2004.

[9] W. F. Wong, Member, IEEE, and E. Goto, "Fast Hardware-Based Algorithms for Elementary Function Computations Using Rectangular Multipliers", *IEEE Transactions on Computers*, Issue 3, VOL. 43,March-1994.

[10] P. Montuschi, L. Ciminiera, A. Giustina, "Division unit with Newton-Raphson approximation and digit-by-digit refinement of the quotient",*IEEE Proceedings - Computers and Digital Techniques*, Issue 6, Vol. 141,
Pages 317 - 324, Nov-1994.

[11] W. F. Wong, Member, IEEE, and E. Goto, "Fast Evaluation of the Elementary Functions in Single Precision", *IEEE Transactions on Computers*, Issue 3, Vol. 44, Pages 453-457, March-1995.

[12] 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.