A scaling-less Newton-Raphson pipelined implementation for a fixed-point inverse square root operator

Erwan Libessart, Matthieu Arzel, Cyril Lahuec, Francesco Andriulli

A scaling-less Newton-Raphson pipelined implementation for a fixed-point inverse square root operator

Erwan Libessart*, Matthieu Arzel*, Cyril Lahuec* and Francesco Andriulli†
*Electronics Department, †Microwave Department
IMT Atlantique, Brest, France
Email: firstname.lastname@imt-atlantique.fr

***Abstract*—The inverse square root is a common operation in digital signal processing architectures, in particular when matrix inversions are required. The Newton-Raphson algorithm is usually used, either in floating or in fixed-point formats. With the former format, the well-known Fast inverse square root computation is based on a 32-bit integer constant, which is allowed by the standardized format of the mantissa. For the fixed-point format, there are many possibilities, which usually force a design with scaling of the input in order to respect a pre-determined work range. Having the input in a known range makes it possible to compute a first approximation with coefficients stored in memory. In this paper, a novel generic architecture which does not require scaling is proposed. This design is totally pipelined, ROM-less and can be directly used in any architecture. The implementation is optimized to reach the maximum clock frequency offered by the DSP cells of Xilinx FPGA. This frequency is higher than the one available by using memory blocks.**

I. Introduction

Signal processing requires many mathematical operators to perform various algorithms. These operators often need to be hard implemented on chip to speed up execution time. Many are difficult to implement especially those requiring calculating the inverse such as the inverse of a square root. This operation is useful for various scientific fields where matrix inversion is required, such as telecommunications [1] or EEG systems [2]. A simple method to compute the inverse square root is to calculate the reciprocal of the input, and then the square root of the result. Both operations can be implemented easily with the CORDIC algorithm [3]. There are some issues with this method, for example necessary resources or long latency, when the size of the input increases. A more effective way is the Newton-Raphson method. It is an iterative algorithm in which the convergence speed is determined by the precision of the first approximation. In floating-point format, the well-known Fast inverse square root is based on a 32-bit integer constant which is used to determine this approximation [4], [5]. In fixed-point format, it is mandatory to have the input in a certain working range, for example $[0.25, 1[$, $[0.5, 1]$ or $[1, 2[$ [6]–[8]. The use of this method also requires re-scaling the output. The computation and the storage of coefficients are often adapted to each use-case in order to have a high-precision first approximation. This means that additional work is necessary to integrate this operator as an IP core in a whole signal processing architecture.

In this paper, a method for implementing the Newton-Raphson algorithm without scaling the data is proposed. So a generic IP core adapted for any design can be proposed. This is possible thanks to the computation of an inverse square root's first approximation, which allows meeting the condition of convergence, whatever the value of the algorithm's input. Moreover, the proposed method does not require any memory block to store coefficients and can be fully pipelined, which permits high-frequency computing. Such a design allows to reach higher frequency than one which uses RAM blocks because of inherent characteristics of FPGA DSP cells and memory blocks.
This paper is organized as follows. Section II presents the Newton-Raphson algorithm and its typical use. The technique that allows to avoid the scaling in described in Section III. Then, the implementation of the whole Newton-Raphson algorithm is described in Section IV. Section V presents the results of the scaling-less Newton-Raphson FPGA implementation. Finally, Section VI concludes the paper.

II. Related work

The Newton-Raphson method is an iterative algorithm which can be used to compute the inverse square root of a number a in which each iteration doubles the number of bits of precision. x_n, the final estimation of $\frac{1}{\sqrt{a}}$, is obtained after n iterations of this equation:

$$x_{i+1} = \frac{x_i}{2}(3 - ax_i^2) \tag{1}$$

where x_0 is the first approximation of $\frac{1}{\sqrt{a}}$, and is another input of the algorithm. This method is usable for both formats of representation: fixed-point and floating-point. The usual approach is to consider that a belongs to a predefined interval as $[1, 2[$, which is the range of the floating-point's mantissa or $[0.5, 1[$, $[0, 1[$ [7], [9]. With a belonging to a predetermined range, it is possible to compute the first approximation with great precision by using polynomial method. So this first step requires the use of a memory block in order to store the different coefficients. With this method, the obtained

precision for the first approximation decreases the number of necessary Newton-Raphson iterations. This strategy requires scaling at the input and output of the algorithm. So this method requires pre- and post-processing in order to manage the scaling which brings additional resources and constraints, and consequently additional integration time.

In this work, an alternative method for fixed-point representation without any scaling of the data, whatever the initial value, is proposed. So the scaling-less Newton-Raphson architecture is "ready to use" and does not need any additional element. Moreover, it allows to reach higher frequency than an architecture which stores coefficients.

III. Choice of the first approximation

The main idea is based on the condition that has to be respected in order to be sure that the algorithm converges:

$$0 < a \times x_0^2 < 3. \tag{2}$$

The strategy presented here consists in adapting the computation of x_0 to a, whereas in the literature a is adapted to the method of x_0's computation, by being in a predefined range. To the authors' knowledge, no other technique to avoid the scaling can be found in the literature.
It is important to note that the closer to 1 the product is, the faster the convergence is. In the following, a is assumed to be in $uQn.n$ format which means that it is an unsigned number with n integer bits and n fractional bits. If the input does not fit this condition, it is simple to concatenate some zeros in order to have the desired symmetry. The output $\frac{1}{\sqrt{a}}$ is represented in the same format.

Let n be the number of integer bits. a is represented in base 2 as:

$$a = a_{n-1}a_{n-2}\dots a_0 a_{-1}\dots a_{-n}. \tag{3}$$

Then let j be the index of the leading one of a. So a respects the following inequality:

$$2^j \leq a < 2^{j+1}. \tag{4}$$

Then a value for x_0 which respects the condition of convergence can be deduced:

$$x_0 = 2^{-\lfloor \frac{(j+1)}{2} \rfloor}. \tag{5}$$

This implies:

$$x_0^2 = 2^{-2\lfloor \frac{(j+1)}{2} \rfloor}. \tag{6}$$

If j is odd:

$$x_0^2 = 2^{-(j+1)}, \tag{7}$$

and then:

$$0.5 \leq a \times x_0^2 < 1. \tag{8}$$

Else if j is even:

$$x_0^2 = 2^{-j}, \tag{9}$$

and then:

$$1 \leq a \times x_0^2 < 2. \tag{10}$$

From (8) and (10):

$$0.5 \leq a \times x_0^2 < 2. \tag{11}$$

Equation (11) implies that x_0 determined by (6) allows meeting the condition of convergence. But the precision of this first approximation can be improved in the case where j is even. For this case, x_0 can be determined as:

$$x_0 = 2^{-\frac{j}{2}+1} + 2^{-\frac{j}{2}+2}. \tag{12}$$

Then:

$$x_0^2 = 0.5625 \times 2^{-j}, \tag{13}$$

and the product with a gives:

$$0.5625 \leq a \times x_0^2 < 1.125, \tag{14}$$

which is more precise then the result in (10).

Finally, x_0, the first approximation of $\frac{1}{\sqrt{a}}$, is given by:

$$x_0 = 2^{-\frac{j+1}{2}} \text{ if } j \text{ is odd} \tag{15}$$

$$x_0 = 2^{-(\frac{j}{2}+1)} + 2^{-(\frac{j}{2}+2)} \text{ if } j \text{ is even} \tag{16}$$

where j is the index of the leading one of a.

And therefore: $0.5 \leq a \times x_0^2 < 1.125$.

IV. The inverse square root architecture

Figure 1 shows the scaling-less Newton-Raphson architecture for the inverse square root operator. The FA block is the first approximation block described in Section III. This block is composed of two parts. The first one is an LOD (Leading One Detector) computation. The method used for the LOD computation consists in the following steps:

1) Take the bit-reversal of a named BR(a)
2) Calculate the two's complement of BR(a)
3) Apply a bitwise AND between this two's complement and BR(a)

The result of this bitwise operation gives exactly the value $2^{-(j+1)}$ which is the bit-reversal of $LOD(a)$. The demonstration of this theory is quite simple. From (4) a is represented like this:

$$a = 0\dots 01 a_{j-1}\dots a_{-n}. \tag{17}$$

So BR(a) is equal to:

$$BR(a) = a_{-n}\dots a_{j-1}10\dots 0. \tag{18}$$

And for its two's complement $TC(BR(a))$:

$$TC(BR(a)) = \overline{a_{-n}}\dots\overline{a_{j-1}}10\dots 0. \tag{19}$$

And then the bitwise AND operation gives the bit-reversal of $LOD(a)$ as below.

$$a_{-n}\dots a_{j-1}10\dots 0 \quad \& \quad \overline{a_{-n}}\dots\overline{a_{j-1}}10\dots 0 = 2^{-(j+1)}. \tag{20}$$

With this method, the critical path is located in the two's complement computation. In order to reduce this path, this

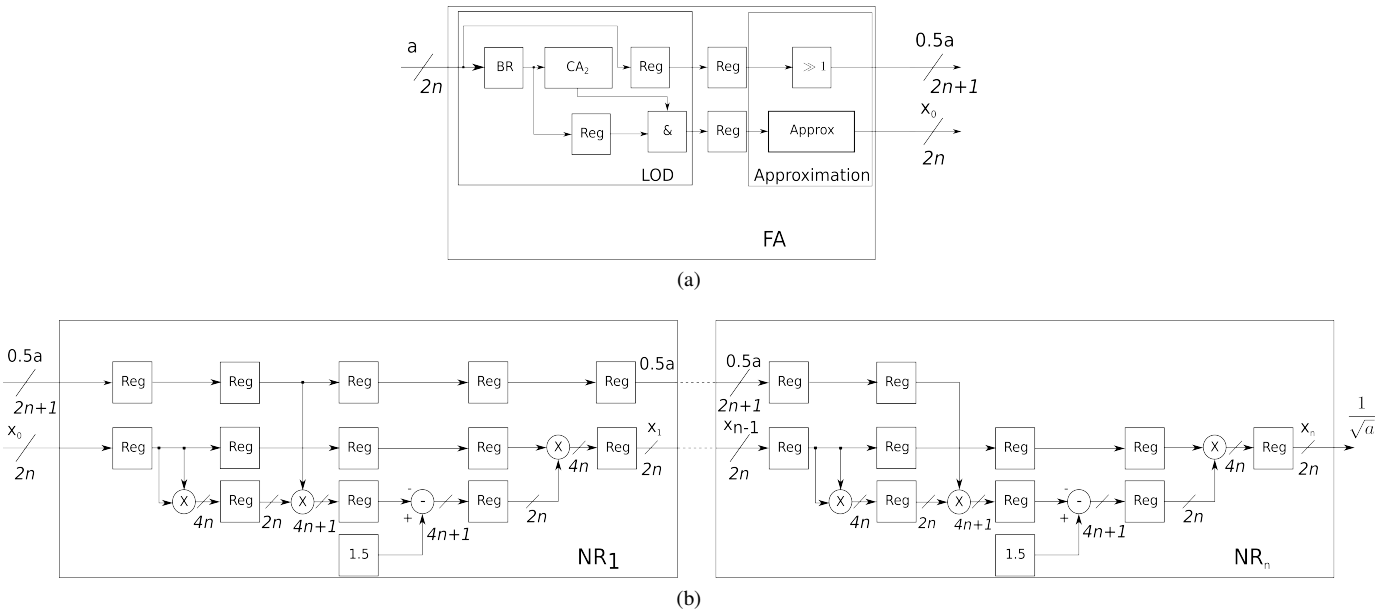

Fig. 1. Architecture for the inverse square root operator (a) First approximation (b) Newton-Raphson iterations

step is realized in two clock cycles. As a reminder, the two's complement of $BR(a)$ is given by:

$$TC(BR(a)) = \overline{a_{-n}}\,\overline{a_{-n+1}}\ldots\overline{a_{n-2}}\,\overline{a_{n-1}} + 1. \quad (21)$$

This binary addition is split in two half additions. During the first cycle, the operation which is done is:

$$ADD_{firsthalf} = c\,\overline{a_0}\ldots\overline{a_{n-2}}\,\overline{a_{n-1}} + 1, \quad (22)$$

where c is the binary carry for this addition. The other bits of $BR(a)$ are simply delayed.

During the second clock cycle, the other half addition is computed:

$$ADD_{secondhalf} = \overline{a_{-n}}\ldots\overline{a_{-1}} + c. \quad (23)$$

Both half results are then concatenated in order to have the required $TC(BR(a))$.

This method for the LOD computation does not depend on the size of the input so it is possible to implement it in a generic way with a Hardware Description Language (HDL).

The Approximation block in Figure 1 has two functions. First, it transmits $\frac{a}{2}$ to the first Newton-Raphson block. Then, the $LOD(a)$ is used to compute the first approximation defined by (15) and (16). As a reminder, $LOD(a)$ is a $uQn.n$ data which copies only the leading one bit of a. The equations can be interpreted as, for all index i:

$$\text{if } LOD(a)_{2i-1} = 1 \text{ then } x_{0\_i} = 1, \quad (24)$$

$$\text{if } LOD(a)_{2i} = 1 \text{ then } x_{0\_(i+1)} = 1 \text{ and } x_{0\_(i+2)} = 1. \quad (25)$$

(24) and (25) gives the logical equation to generate x_0, which is implemented in the $Approx$ block presented in Figure 1:

$$x_{0_i} = LOD(a)_{-2i-1} + LOD(a)_{-2i-2} + LOD(a)_{-2i-4}. \quad (26)$$

So each bit of x_0 is generated by a 3-input OR. This operation is only computed for the bits of x_0 indexed between $\frac{-n-3}{2}$ and $\frac{n-1}{2}$. The FA block transmits the value of $\frac{a}{2}$ and delivers x_0 in $2n$ bits in three clock cycles. Its implementation is totally generic and can be adapted to any input sizes.

A Newton-Raphson iteration block (NR_i) in Figure 1 computes the i^{th} evaluation x_i (1). Each of these blocks contains 3 multipliers and is totally pipelined. Bus reductions are required during the process. In fact, the different arithmetic operations generate useless bits for the required precision so these can be ignored. All the blocks are identical, except for the last one, which has not to transmit the value of a. The ouput of the operator is in $uQn.n$ format, so it provides an accuracy of 2^{-n}.

V. Implementation Results

This section is composed of two parts. First, the operator presented in this paper is compared to the solution of [8] on a Virtex-6 target. Then, the results will be discussed in terms of maximum available frequency.

A. Comparison on Virtex-6 FPGA

Table I presents the implementation results of the solution presented in this paper in comparison with [8]. Those results are a 16-bit inverse square root. In [8], the format used is signed $Q1.14$. So the effective computation is on 15 bits. This predefined format allows to compute a first approximation based on memory and address generator.

Only one Newton-Raphson iteration is required thanks to the precision provided by this memory. The architecture presented in Figure 1 requires 3 iterations. This explains that it requires slightly more LUTs, registers and DSP cells and it needs more clock cycles. In [8]'s work, the critical path is located in the first approximation part and limits the maximum frequency at 351.9 MHz. Comparatively, the architecture presented in Figure 1 allows to reach a maximum frequency of 638.6 MHz. So the proposed operator in this paper requires more resources but delivers the benefits of being size generic, reaching higher computation frequency and being easily integrated as an IP core.

TABLE I
FPGA IMPLEMENTATION RESULTS ON VIRTEX-6

	Proposed design	[8]
Registers	258	160
LUTs	203	160
DSP48E1s	9	4
BRAMs	0	1
Clock cycles	34	5
Max frequency (MHz)	638.6	351.9

B. Discussion on the results

Table II shows the implementation results of the scaling-less inverse square root operator on the Virtex-7 690T FPGA. In terms of resources, it is very similar to the results presented in Table I. The main difference concerns the achieved frequency, which increases from 638.6 MHz to 740 MHz. The latter frequency is the highest frequency that can be achieved by the DSP cells of the Virtex-7 FPGA family [10], [11].

It is true that the architecture presented in [8] can be fully pipelined in order to increase the frequency of the design. The critical path will be located in the Block RAM. For example, on a Virtex-7 FPGA target a Block RAM limits the architecture at 601 MHz [11]. This difference between the maximum frequency of DSP cells and Block RAM also applies the newer generation of Xilinx FPGA. In fact, for the Virtex Ultrascale, the maximum frequency of the Block RAM (660 MHz) is lower than the DSP cell one (741 MHz) [12]. So, in addition to offering flexibility and ease of use, the operator presented in this paper is better than memory-based architectures for high throughput applications.

TABLE II
IMPLEMENTATION RESULTS: 16-BIT INVERSE SQUARE ROOT ON VIRTEX-7 690T

Registers	261
LUTs	221
DSP48E1s	9
BRAMS	0
Clock cycles	34
Max frequency (MHz)	740

VI. CONCLUSION

This paper presents a scaling-less fixed-point Newton-Raphson implementation for an inverse square root operator. It is possible to have a design in which no scaling is obligatory by adapting the first approximation, which allows to implement a "ready to use" IP core. This adaptation is made with a Leading One Detector operator, followed by a simple logical operation. The proposed design is size generic in order to be more flexible and save resources. The architecture is totally pipelined, which allows to reach the maximum clock frequency offered by FPGA DSP cells. This frequency is higher than the one allowed by memory blocks, which are not used in the design. Thus this is an architecture which can be used directly and easily in every digital signal processing architecture regardless of the representation format of the input. Moreover, the presented work is available as an open-source project [13].

REFERENCES

[1] C. Mahapatra, S. Mahboob, V. C. M. Leung, and T. Stouraitis, "Fast Inverse Square Root Based Matrix Inverse for MIMO-LTE Systems," in *2012 International Conference on Control Engineering and Communication Technology (ICCECT)*, Dec. 2012, pp. 321–324.

[2] K. J. Huang, W. Y. Shih, J. C. Chang, C. W. Feng, and W. C. Fang, "A pipeline VLSI design of fast singular value decomposition processor for real-time EEG system based on on-line recursive independent component analysis," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Jul. 2013, pp. 1944–1947.

[3] R. Andraka, "A survey of cordic algorithms for fpga based computers," in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, ser. FPGA '98. New York, NY, USA: ACM, 1998, pp. 191–200. [Online]. Available: http://doi.acm.org/10.1145/275107.275139

[4] "Fast inverse square root," Oct. 2016, page Version ID: 747006619. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Fast_inverse_square_root&oldid=747006619

[5] J. L. V. M. Stanislaus and T. Mohsenin, "High performance compressive sensing reconstruction hardware with QRD process," in *2012 IEEE International Symposium on Circuits and Systems*, May 2012, pp. 29–32.

[6] M. Allie and R. Lyons, "A root of less evil [digital signal processing]," *IEEE Signal Processing Magazine*, vol. 22, no. 2, pp. 93–96, Mar. 2005.

[7] H. C. Neto and M. P. Vestias, "Very low resource table-based FPGA evaluation of elementary functions," in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Dec. 2013, pp. 1–6.

[8] G. R. Prabhu, B. Johnson, and J. S. Rani, "FPGA Based Scalable Fixed Point QRD Core Using Dynamic Partial Reconfiguration," in *2015 28th International Conference on VLSI Design*, Jan. 2015, pp. 345–350.

[9] S. Niu, S. Aslan, and J. Saniie, "FPGA based architectures for high performance adaptive FIR filter systems," in *2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, May 2013, pp. 1662–1665.

[10] Xilinx. (2016 (Accessed: 2017-02-10)) 7 series dsp48e1 slice-user guide. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

[11] ——. (2016 (Accessed: 2017-02-10)) Virtex-7 t and xt fpgas data sheet: Dc and ac switching characteristics. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds183_Virtex_7_Data_Sheet.pdf

[12] ——. (2016 (Accessed: 2017-02-10)) Virtex ultrascale fpgas data sheet: Dc and ac switching characteristics. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds893-virtex-ultrascale-data-sheet.pdf

[13] *Removed for review.*