Microprocessors and Microsystems 39 (2015) 74–82

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Fast bit-accurate reciprocal square root

L. Pizano-Escalante [a,1], R. Parra-Michel [a,1], J. Vázquez Castillo [b,*], O. Longoria-Gandara [c,2]

[a] Department of Electrical Engineering, Communications Section, CINVESTAV-IPN, Av. del Bosque 1145, col. El Bajío, 45019 Zapopan, Jalisco, Mexico
[b] Science Division and Engineering, University of Quintana Roo, Boulevard Bahia s/n esq. Ignacio Comonfort, Chetumal, Quintana Roo, Mexico
[c] Department of Electronics, Systems and IT, ITESO, Jalisco, Mexico

ARTICLE INFO

Article history:
Available online 7 February 2015

Keywords:
Fixed-point arithmetic
Newton–Raphson
Polynomial approximation
Reciprocal square root

ABSTRACT

The reciprocal square root (RSR) is an operation extensively used in signal processing algorithms, where it is necessary the design of RSR architectures in fixed-point (FxP) representation for using in mobile devices. Currently, RSR implementations are mainly focused on floating point format, which requires long execution time and large area resources. In this paper, an algorithm for designing FxP RSR architectures is proposed, which achieves bit-accurate results in two clock cycles of execution time. The proposed algorithm is based on the Newton–Raphson method, where the seed is provided through piecewise polynomial approximation. A comparison between this RSR proposal and the straightforward approach shows that the proposed algorithm achieves an approximately 10-fold gain in execution time, which allows a speed-up of signal processing algorithms.

1. Introduction

The hardware implementation of the reciprocal square root (RSR) operation is an essential block in several digital signal processing (DSP) designs, such as matrix decomposition [1], normalization of vectors [2] or image rendering [3]. In general, RSR hardware implementations have been focused on floating point (FP) schemes as presented in [4,5]. However, it is known that FP architectures require a long execution time and large area resources [6,7], and they are mainly used for high demanding processing like scientific computations [8].

On the other hand, current and future multimedia applications with computationally intensive algorithms that are implemented in embedded systems, like mobile devices and wireless systems, have power-optimization problems with hard real time and area constraints. Therefore, fixed-point (FxP) hardware implementations are preferred for these kind of devices due to the fact that FxP architectures offer a shorter execution time to satisfy real-time requirements, less area resources and power consumption [9,7,6]. Furthermore, FxP hardware implementations provide good numerical accuracy in the system under development without introducing significant performance losses and reliability (e.g. in the bit error rate of wireless communication system [6] or image quality in multimedia co-processor for rendering [10]).

In spite of the advantages of FxP architectures there are a few works related to efficient FxP implementation of the RSR operation. The straightforward FxP RSR implementation consists of a concatenated approach, where a square root (SR) operation is first performed and then a divider is employed. In this context, the research in the open literature has been oriented to the efficient implementation of the separate SR and divider modules; see for instance: [11–14] for FxP SR and [15–17] for FxP division.

This paper proposes a combined architecture for FxP RSR computation. The approach is based on the Newton–Raphson (NR) method, producing bit-accurate results in 16-bit precision in only two clock cycles, greatly improving previous results. It is well-known that NR has quadratic convergence and its performance is highly dependent on a good initial approximation [18]. This problem has been solved in this paper by using a piecewise polynomial approximation for providing the NR seed.

This paper is organized as follows: Section 2 presents an analysis of a bit-accurate RSR implementation based on the straightforward divided approach, in order to determine its real performance and constraints. Section 3 introduces the proposed algorithm, while Section 4 discusses its architecture in detail. Section 5 explains the test methodology. Section 6 presents the implementation results and performance comparison with the traditional approach for both bit-accurate and bit-inaccurate cases; and finally, Section 7 presents the conclusions.

* Corresponding author. Tel.: +52 9838350300; fax: +52 9838329656.
E-mail addresses: jpizano@gdl.cinvestav.mx (L. Pizano-Escalante), rparra@gdl.cinvestav.mx (R. Parra-Michel), jvazquez@uqroo.edu.mx (J. Vázquez Castillo), olongoria@iteso.mx (O. Longoria-Gandara).
[1] Tel.: +52 33 3777 3600x1020; fax: +52 333777360.
[2] Tel.: +52 3336693598; fax: +52 3336693511.

http://dx.doi.org/10.1016/j.micpro.2015.01.008
0141-9331/

2. Bit-accurate RSR based on concatenation of square root and division operators

2.1. Importance of bit accuracy in fixed-point algorithms

Real applications oriented to mobile devices rely on FxP arithmetic [6,19]. Therefore, it is very important to assure that mathematical operations are optimally implemented. As this concept is hard to manage, it is normal to talk about optimality of a digital design in terms of a cost function that depends on several factors: (a) accuracy of the given result, (b) area resources, (c) clock cycles required, (d) maximum clock frequency attainable, and (e) power consumption. While the last four metrics are well-known and widely used for comparing architectures' performance, the accuracy metric is not usually discussed in detail. Therefore, the following discussion could serve to clarify this concept.

An important issue related with FxP implementations is to determine how much FxP format affects the numerical accuracy of the algorithm being implemented. Numerical errors or an incorrect selection in the FxP format will degrade the system performance. The standard way to analyze the numerical accuracy of a FxP architecture is through simulations using the well-known DSP techniques proposed in [20,7]. These analysis takes as figure of merit the signal-to-quantization-noise ratio (SQNR), which is defined in this context as the logarithm of the ratio between the power of the signal, *Ps*, and the power of the error, *Pe*, as follows:

$$\mathrm{SQNR} = 10\log_{10}\left\{\frac{Ps}{Pe}\right\}. \tag{1}$$

Assuming ergodicity, *Ps* and *Pe* can be calculated with

$$Ps = \frac{1}{N}\sum_{i=1}^{N} Vfp_i^2,$$

$$Pe = \frac{1}{N}\sum_{i=1}^{N} (Vfp_i - Vxp_i)^2, \tag{2}$$

where Vfp_i and Vxp_i are the FP value and FxP value, respectively, of the ith result and N is the number of experiments. SQNR value is application dependent, thus it is not possible to know in advance either the minimum number of bits in the FxP representation of the numerical values, or the minimum value of the SQNR required to guarantee system performance.

On the other hand, a more demanding accuracy metric can be defined as being bit-accurate. It means that the algorithm is capable of providing the exact k bits obtained by rounding the FP result. This requirement arises in any deterministic math processing where the expected result should provide a minimum SQNR for any single value (depending on the bit width), and not an average SQNR. This approach, while requiring more resources, provides the maximum attainable SQNR for a given precision. The aim of this paper is to provide a bit-accurate RSR. However, as this accuracy metric has not been addressed extensively in the open literature, it is presented in the following subsection.

2.2. Background on SR and divider

Square root and division operations can be implemented with a wide variety of algorithms [21]. These algorithms produce different architectures and they are adequate for different applications. In FxP applications, high-throughput architectures with moderate hardware resources are required. One approach for meeting this requirement consists of having architectures with a simple structure that runs at high clock frequency, and the high throughput is achieved by pipelining the algorithm. For this approach, the FxP SR algorithm proposed in [11] is the simplest and most cited option. For the division operation, on the other hand, the coordinate rotation digital computer (CORDIC) is the approach that is most often referenced [15]. The FxP SR approach in [11], with a word length input of k bits, will produce a result with a length of k bits after k iterations. CORDICs with a word length input of k bits also produce a result of k bits after k iterations.

High-radix approaches produce more than one bit per iteration at the cost of increasing the amount of hardware required. In addition, the quotient digit selection function becomes more complicated and the time for each iteration also increases [21]; as a consequence, the frequency operation decreases. In the case of the pipelined implementation, the amount of hardware required is significantly increased. In order to compare the proposed algorithm with the straightforward approach, the architecture presented in [11] has been selected for the implementation of the SR algorithm while the CORDIC for the division algorithm, hereafter referred to as concatenated RSR (CRSR). This choice is based on the fact that this approach represents the extreme case of a very fast and simple architecture with few iterations. However, as the results provided by the original works are not bit-accurate, these algorithms have to be analyzed in this context. The analysis is presented in the following paragraphs.

2.3. A bit-accurate CRSR

The original SR implementation in [11] has been modified in this paper to produce a final result of k bits while being bit-accurate. To achieve this modification, the iterations must be increased to $k+1$ and the least significant bit is used for rounding the result. In this way, the algorithm in [11] can provide k accurate bits after $k+2$ iterations.

Even though CORDICs are discussed extensively in the literature, the important characteristic of their inaccuracy is not addressed. Thus, CORDICs are employed with an input and internal word length of k bits, which provides a result of k bits after k iterations [15]. However, as it has been previously commented, this result is not bit-accurate and the error in the result is not reduced significantly when the number of iterations is increased [21]. In order to produce a bit-accurate result with CORDICs in the division mode for an input word with 16 bits, both the internal word length and the number of iterations have to be increased. This study concludes that performing the division operation with 16 bits of precision using CORDICs takes 32 iterations and requires 32 bits of internal word length.

In addition to these modifications, when the requirement is to produce an accurate 16-bit RSR, the number of bits passed to the CORDIC from the SR module must increase to 28. According to the performance commented above, the SR performs 28 iterations to provide 28 accurate bits for the CORDIC, and this block then performs 32 iterations. The final count shows that the CRSR implementation requires 60 clock cycles to produce a 16-bit-accurate result. Fig. 1 shows the basic hardware modules of the CRSR architecture.

In Fig. 1 the input data is first processed in the SR block, which is made up of two barrel shifters and one adder/subtracter (**ADD**/**SUB**), and whose operation is described in detail in [11]. The algorithm is iterated $26+1$ times, in order to provide the extra bit required for rounding operations, which are carried out in the 28th clock cycle. The result of the first block is then passed to the divider block. This block implements the operation Num/Den employing the CORDIC algorithm in vectoring mode, where Num = 0x0800 and Den corresponds to the result given by the SR block. The CORDIC is made up of two sub-blocks. Sub-block 1 (**SB1**) implements the rotation of the data, while the sign S of the rotation (the most significant bit of the rotated data) is used to