A multi-cycle fixed point square root module for FPGAs

Fernando Martin del Campo[1], Alicia Morales-Reyes[1], Roberto Perez-Andrade[3], Rene Cumplido[1], Aldo G. Orozco-Lugo[2] and Claudia Feregrino[1]

1 INAOE, Computer Science Department, Puebla, Mexico
2 CINVESTAV IPN, Electrical Engineering Department, D. F., Mexico
3 CINVESTAV IPN, Information Technology Laboratory, Tamps., Mexico
[martin,a.morales]@ccc.inaoep.mx, jrperez@tamps.cinvestav.mx, aorozco@cinvestav.mx, [rcumplido,cferegrino]@ccc.inaoep.mx

Abstract: This paper presents a module that solves the square root by obtaining a number of more significant bits from a look-up table as an approximate root. A set of possible roots are then appended and squared for comparison to the original radicand, finely tuning the calculation. The module stops as soon as it finds an exact root, therefore not all entries take the same number of cycles, reducing the number of iterations required for full resolution. The proposed FPGA module overcomes a Xilinx's logiCORE IP in terms of resources utilization and in several cases latency due to its flexible structure configuration.
Keywords: Square root, non-restoring algorithm, FPGA
Classification: Electron devices, circuits, and systems

References

[1] K. Piromsopa, C. Aporntewan, and P. Chogsatitvataa, "An FPGA implementation of a fixed-point square root operation", Proc. Int. Symp. on Communications and Information Technology, Thailand, pp. 587-589, 2001.

[2] Yamin Li and Wanming Chu, "A new non-restoring square root algorithm and its VLSI implementations", Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors, Austin, USA, pp. 538-544, October, 1996.

[3] Yamin Li and Wanming Chu, "Parallel-Array implementations of a non-restoring square root algorithm", Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors, Washington, USA, pp. 690-695, October, 1997.

[4] Tole Sutikno, "An efficient implementation of the non restoring square root algorithm in gate level", Int. Journal of Computer Theory and Engineering, Vol. 3, no. 1, pp. 46-51, February, 2011.

[5] S. Imtiaz, M.M. Ahmed and Z.G. Sotirios, "Novel Pipelined Architecture for Efficient Evaluation of the Square Root Using a Modified Non-Restoring Algorithm", *Journal of Signal Processing and Systems*, Vol. 67, pp. 157-166, September, 2010.

[6] Xilixn, Inc., "LogiCORE IP CORDIC v4.0", Prod. specification DS249, March, 2011.

[7] S. Samavi, A. Sadrabadi and A. Fanian, "Modular array structure for non-restoring square root circuit", *Journal of Systems Architecture*, Vol.

54, pp. 957-966, April, 2008.

[8] S. Lachowicz and H-J. Pfleiderer, "Fast Evaluation of the Square Root and Other Nonlinear Functions in FPGA", IEEE International Symposium on Electronic Design, Test & Applications - DELTA, pp. 474-477, 2008.

[9] F. Dinechin, M. Joldes, B. Pasca and G. Revy, "Multiplicative Square Root Algorithms for FPGAs", IEEE International Conference on Field Programmable Logic and Applications, pp. 574-577, 2010.

[10] T. Kwon and J. Draper, "Floating-point division and square root using a Taylor-series expansion algorithm", Microelectronics Journal, Elsevier, Vol. 40, pp. 1601-1605, 2009.

1 Introduction

The square root operation is common and widely used in several areas such as image and signal processing, statistics, communications and design of scientific engines. Calculating the square root is a complicated task thus several techniques have been proposed to implement this operation in hardware. Many of them use digit recurrence algorithmic approaches which were suitable at a time when hardware devices did not provide embedded modules such as multipliers or RAM blocks. These modules are now available on reconfigurable fabrics easing the implementation of multiplicative square root techniques.

In [1], a fixed-point square root module implemented on an Altera FLEX-10K20RC240 FPGA is reported. Its maximum operation frequency is 21.36 MHz using 13% logic cells for a 48-bit radicand. This architecture only uses an adder/subtractor and three registers performing one and two bits left shift and storage for results.

In [2] a non-restoring square root algorithm is proposed. It focuses on the partial remainder instead of calculating one bit's root every iteration reducing significantly the total number of iterations. Two VLSI approaches were implemented: a fully pipelined design which accepts a square root instruction every clock cycle and a low-cost implementation using an adder/subtractor and three registers. A systolic modification of [2] is presented in [3]. The adder/subtractor unit is replaced by a systolic array using carry-save adders. This approach reduces the calculation in 1 and 2 cycles for 16 and 32-bit respectively, but increasing in 8 cycles for 64-bit. The approach significantly reduces resources requirements.

In [4] a modified non-restoring square root algorithm is presented. Its main difference is to use a subtract operation and append a binary value 01 instead of appending 11 and add to the developed root. Results for 32 and 64-bit input are characterized with a significant improvement in resource utilization of more than 50% achieved by an Altera APEX-20KE FPGA. In [5] another modified non-restoring technique is introduced using fix point representation for low cost implementation. Its pipeline approach avoids a write after read hazard by feeding register using different lines in the same

clock cycle. Results showed reduction in hardware usage, shorter latencies and lower power consumption. Samavi et al. presented a modular array based on add/subtract units using the non-restoring square root algorithm [7]. Several elements are removed without accuracy loss achieving significant reduction in the required area. The structure allows arbitrary input width. Results showed that latency and hardware resources usage are reduced.

Multiplicative square root techniques have also been proposed considering current hardware capabilities. In [8], a non-constant LUT based technique is presented. It consists on an initial approximation using a square function which allows a reduction in the LUT size. More LUT locations are needed to store non-linear points but less points describe linear segments of the square curve. Multipliers are used to calculate the increments iteratively. This approach is defined for 24-bit radicand only.

In [9], a survey on square root techniques and a polynomial based multiplicative square root approach are presented. A comparison of a pipelined digit recurrence square root for single and double precision radicand slightly outperformed the Xilinx LogiCore in terms of hardware resources. Attention is paid to the rounding problem which is computationally expensive. This multiplicative approach based on polynomial approximations results in a theoretical latency of 25-27 cycles with maximum accuracy for 64-bit radicand. In [10], a single precision square root operation is added to a multiply/divide unit. The method takes advantage of the Taylor-series expansion used in division to calculate the square root. A comparison among algorithmic approaches implemented on different GPPs shows a latency of 12 cycles achieved by this technique.

In this article, a combination of a non-restoring square root algorithm and a multiplicative technique is proposed. The approach takes advantage of current reconfigurable hardware capabilities such as embedded multipliers. To evaluate the proposed approach, a Xilinx LogiCore for square root calculation is used as reference. This highly efficient core provides a good measure of the strengths of the proposed module.

2   Square Root Module

The algorithmic approach for the square root calculation is a combination of a multiplicative technique and the non-restoring square root algorithm [1]. In Algorithm 1 the pseudo code is provided. Initially, the approximate root is obtained from a look-up table using as index a number of the radicand's most significant bits (lines 3-4). The approximate root is appended to a set of possible roots that are squared and compared to the original radicand (lines 7-11). A comparison tree evaluates the remainders to determine the minimal (lines 12-20). If the minimal remainder equals zero, the exact square root was obtained and the algorithm exits (line 22). On the contrary, the square root with minimal remainder is updated as the new approximate root (line 24). In every iteration, the approximate root grows by a number of bits, until reaching the least significant bit.

Algorithm 1 Square root algorithm

1: **procedure** SQR(X_{2n}) $\triangleright$ X radicand, $2n$ input length

2:     $(Q_n, ..., Q_0) \leftarrow 0;$ $\triangleright$ Q_n square root

3:     $\left(Q_n, ..., Q_{\left(n-\frac{m}{2}\right)}, 0, ..., 0\right) \leftarrow$ $\triangleright$ Approx. root for m-bit of X' MSB

4:         $ROM\left[\left(X_{2n}, ..., X_{(2n-m)}\right)\right];$

5:     $r = \frac{2n-m}{p};$ $\triangleright$ r blocks of p-bit pending for square root calculation

6:    **while** $r \neq 0$ **do**

7:       $Q'_0 \leftarrow$ $\triangleright$ Square root options by appending 2^p solutions

8:        $\left(Q_n, ..., Q_{\left(n-\frac{m}{2}\right)}, 0_p, ..., 0_0, 0, ..., 0\right);$ ...

9:       ... $Q'_p \leftarrow \left(Q_n, ..., Q_{\left(n-\frac{m}{2}\right)}, 1_p, ..., 1_0, 0, ..., 0\right);$

10:       $R_0 \leftarrow \left(X - (Q'_0)^2\right);$ ... $\triangleright$ 2^p remainders after subtraction from X

11:       ... $R_p \leftarrow \left(X - \left(Q'_p\right)^2\right);$

12:       $q \leftarrow 2^p >> 1;$ $\triangleright$ Comparison tree levels

13:       **while** $q \neq 0$ **do** $\triangleright$ Determine minimal remainder

14:         **if** $R_q < R_{q-1}$ **then**

15:           $R'q \leftarrow R_q;$

16:         **else**

17:           $R'q \leftarrow R_{q-1};$

18:         **end if**

19:         $q \leftarrow q >> 1;$

20:       **end while**

21:       **if** $R'q = 0$ **then**

22:         Break; $\triangleright$ Exact square root had been calculated

23:       **else**

24:         $Q \leftarrow Q'_q;$ $\triangleright$ Solution with minimal remainder is assigned

25:         $r \leftarrow r - 1;$ $\triangleright$ r block(s) pending for root calculation

26:       **end if**

27:    **end while**

28: **end procedure**

The new square root module operates in a similar way to a non-restoring algorithm implementation with two main differences:

- A number of the root's more significant bits are obtained from a look-up table. This initial root approximation is then finely tuned. For example, the square root of 5 is ≈ 2.236. The look-up table would give the value of 2, and only the decimal part of the result has to be calculated. This increases drastically the speed of the coprocessor using very little FPGA area.

- Every iteration calculates, in parallel, a block of the root's bits whose size varies according to the radicand size and not only one bit as the non-restoring original technique does. A block chart of the proposed system is shown in Figure **1**.
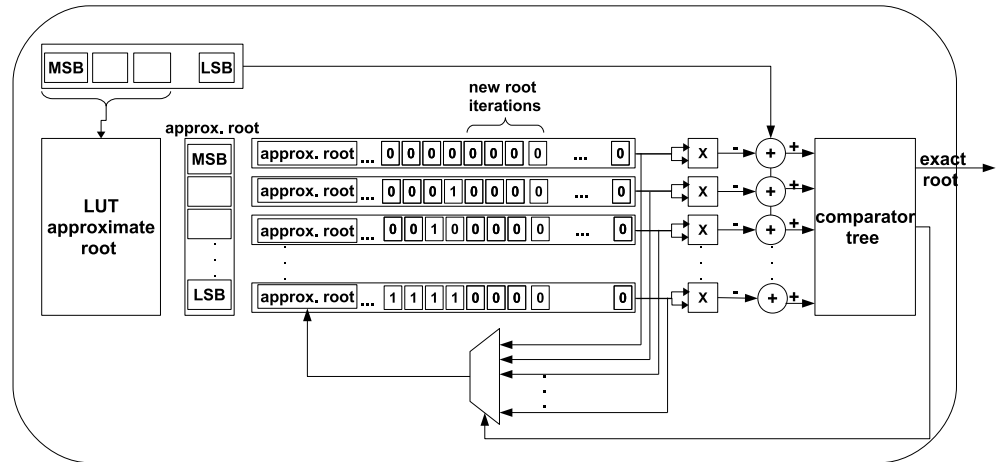
Fig. 1. Square root module

It is important to consider the bit length of the look-up table memory. As more bits are added to this structure, the number of iterations to calculate the square root decreases. However, as this length grows, the memory size increases significantly, to the point that it is impossible to implement using FPGA's memory blocks and even external memories available, like the SDRAM. Defining the size of the look-up table involves a trade-off that is particularly important in FPGA architectures.

Another advantage of the coprocessor is that it stops once an exact root is found, so not all entries take the same number of cycles to be calculated. For example, if the root of 14.0625 is computed, the process stops after the exact root is calculated (3.75), even if the original number is represented by 64-bit requiring usually 6 iterations for full resolution or 5 iterations for maximum error of $\approx 7.15 \times 10^{-7}$.

3   Results and Discussion

In Table **I**, results obtained with characterization data for a Xilinx's FPGA device Virtex-5 XC5VSX35T are presented. For comparison, the first row in the table shows performance and resource utilization for the Xilinx's logiCORE IP CORDIC v4.0 configured to perform the square root calculation [6]. This Xilinx's IP core is selected due to being optimized for FPGA fabrics and therefore a fair reference for performance comparison. This module can be configured for 16, 32 and 48-bit as maximum input width. For the proposed architecture a 64-bit input width case is also evaluated.

For each input data size the number of radicand's most significant bits whose root is obtained from a LUT (and therefore the LUT's size) is changed. Having larger ROM sizes while maintaining the iterative block size increases the resource utilization but not significantly. The number of multipliers used in the comparison tree varies according to the size of the iterative block defined by 2-bit for cases reported in Table **I**. The number of cycles required to produce a result is also reduced while increasing the ROM size and maintaining the same iterative block size. This is a consequence of requiring less

	Input width	LUT-FF pairs	Max.Freq. (MHz)	Cycles
logiCORE, IP CORDIC v4.0 [6]	16	461	339	16
	32	1550	283	32
	48	2860	241	48
	64	–	–	–
4-bit ROM 2-bit iterative block	16	325	197	22
	32	609	197	50
	48	857	138	78
	64	1139	90	106
8-bit ROM 2-bit iterative block	16	328	197	15
	32	618	197	43
	48	870	138	71
	64	1144	90	99
12-bit ROM 2-bit iterative block	16	399	197	8
	32	679	197	36
	48	936	138	64
	64	1196	90	92

Table I. Results comparison with characterization data for a Xilinx's FPGA device Virtex-5 XC5VSX35T

iterations due to the number of bits in the initial approximate root obtained from the LUT. A more aggressive approach is to significantly increase the size of both, the LUT memory and the iterative block, that would reflect in an increase in the use of hardware resources with a reduction in the number of cycles.

For 16, 32 and 48-bit input width the proposed approach presents a significant reduction in resources utilization, when using 4, 8 and 12-bit ROM array with 2-bit iterative block size. For 16-bit input width, a reduction of 1 and 8 clock cycles is achieved by using 8 and 12-bit ROM in comparison to the Xilinx's logiCORE IP CORDIC v4.0. Moreover, a minimum latency of $4.06 \times^{-8}$ sec. is also achieved by the proposed non-restoring square root algorithm. In the next section, the main conclusions of this study are presented.

4   Conclusions

In this article, a flexible approach to the square root non-restoring algorithm is presented. Two main differences are introduced, the use of a ROM array to obtain an approximate root of the radicand's MSB and an iterative block containing possible root options. ROM and iterative block sizes should be defined. Having variable size in these elements reflects on the overall module's performance. The Xilinx's logiCORE IP Cordic v4.0 is used for performance reference. This IP core is designed to exploit all features available in modern FPGAs providing a good measure of the strengths of the new proposed approach. In terms of maximum clock frequency the reference module achieves

higher frequencies. However, resources utilization by the proposed module improves when the iterative block size is small together with a reduction in the number of cycles required to calculate the square root. The proposed approach requires 4 multipliers for an iterative block size of 2 which is 2% of the total number of available DSP48E slices in the characterized device. Thus, it does not represent a significant overuse of those resources. Therefore, the proposed approach is a competitive option that reduces in less than half the area required and achieves similar or smaller latencies. Moreover, it is a compact option suitable for integration into larger architectures designs with significant configuration flexibility to adapt to specific applications as required.