A 4-MHz parameterized Logarithm-Square Root IP-Core

Nelson Campos, Roberto Costa, Elton Costa, Gutemberg Junior and Elmar Melcher
Federal University of Campina Grande - UFCG
Center of Electrical Engineering and Informatics - CEEI
(nelson.campos, roberto.costa, elton.costa)@ee.ufcg.edu.br
gutemberg@dee.ufcg.edu.br
elmar@dsc.ufcg.edu.br

***Abstract*—Logarithms and square root are non-elementary operations frequently used in digital signal processing. In this work, implementation and design of an IP-Core to compute square root and multibase logarithm is presented. The design is parameterized in fixed point notation achieving a low arithmetic error even when irrational numbers are being calculated. The module was synthesized in ASIC using FSC0G_D_GENERIC_CORE from UMC and in FPGA occupying 518 logic elements and two DSP blocks for multiplication.**

***Keywords*—Digital Signal Processing, FPGA, Logarithm, Square Root, VLSI.**

I. Introduction

The logarithm operation is widely used in digital signal processing applications such as speech recognition [4] [10] and 3D graphics [5]. The square root operation is also present in many modern applications such as Cholesky decomposition, LU factorization and in the solution of quadratic equations [11]. This operation requires expensive computational resources and high energy consumption, and its efficient implementation may be accomplished with the use of dedicated hardware. This paper demonstrates the hardware implementation of an arithmetic module called SQRTLOG comprising four modules: $sqrt$, log_2 , log_{10} and log_e. First, the paper describes some common algorithms to compute the square root and the binary logarithm. Next, the paper explains the proposed architecture of the SQRTLOG and finally discusses the results of the synthesized hardware in ASIC and FPGA.

II. Square Root Computation Methods

There are many ways to compute the square root operations for VLSI and FPGA implementations. Three algorithms will be discussed here.

A. The Newton-Raphson Algorithm

The Newton-Raphson method is widely used in order to calculate $Y = \sqrt{X}$, by approximation. The reciprocal square root using this method may be computed iteratively according to equation 1, where q_i is the approximate value of $1/\sqrt{X}$. After n iterations the square root of X is $Y \approx q_n X$ [6].

$$q_{i+1} = q_i(3 - xq_i^2)/2 \tag{1}$$

B. The Restoring Algorithm

The restoring algorithm will calculate square root and its remainder value in iterative process. Consider that $S = \sqrt{D}$ and $D = S^2 + R$, where D is the radicand, S is the square root result, and R is the remainder. The restoring algorithm will guess the value of S and R iteratively. If there is a wrong guess of S, the previous value is restored[9].

C. The Non-Restoring Algorithm

The non-restoring algorithm is similar to the restoring one and it will calculate square root and its remainder value in iterative process. However, unlike the restoring algorithm, in case of a wrong guess of S, it does not change the bits of S more than once [9].

III. Logarithm Computation Methods

Some techniques have been used in hardware implementations of Logarithms.

A. Look-Up Table

According to the implementation described in [10] any number can be represented in the form of $a = N \times 2^p$, where p is integer and $N \in [0.5, 1]$. Taking the binary logarihtm of a, the equation 2 is obtained. Since p is the most significant bit of a, and $log_2(N)$ can be found using a LUT, the computation of the $log_e(a)$ can be done mutliplying $log_2(a)$ by a constant as shown in equation (2).

$$log_2(a) = p + log_2(N) \tag{2}$$

B. Cordic Method

The CORDIC is a simple and efficient algorithm to calculate hyperbolic and trigonometric functions setting one bit per iteration [2][7][8]. The logarithm computation may be accomplished using the hyperbolic expression shown in equation 3.

$$ln(a) = 2 \times tanh^{-1}\frac{(a-1)}{(a+1)} \tag{3}$$

C. Floor-Shift Method

The method presented in [7] computes the binary logarithm through the following steps:

- first calculate $log_2(x)$ by searching for the leading one to get the integer part of $log_2(x)$

- then calculte the fractional part of the logarithm with equation 4:

$$log_2(m) = \frac{x - 2^{\lfloor log_2(x) \rfloor}}{2^{\lfloor log_2(x) \rfloor}} \quad (4)$$

IV. PROPOSED ARCHITECTURE

This section discusses the proposed architecture of the SQRTLOG, which is divided in two parts: the *sqrt* module and the *logarithm* module. The *sqrt* module computes the square root of $N = (a_1 + a_2 + a_3 + .. + a_n)^2$ guessing one bit at a time according to equations 5 to 7, where a_m is the $m-th$ bit being guessed.

$$X_m = X_{m-1} - Y_m \quad (5)$$

$$Y_m = (2P_{m-1} + a_m)a_m \quad (6)$$

$$P_m = \sum_{i=0}^{m} a_i \quad (7)$$

The steps to compute the square root algorithm can be seen in the flowchart depicted in Figure 1.

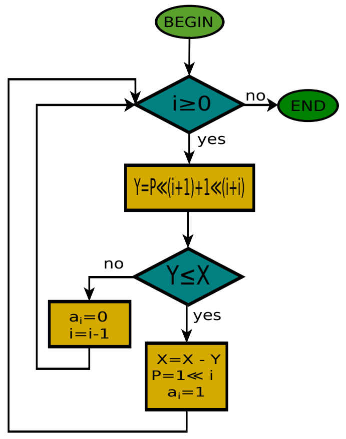

Fig. 1. Square Root Algotihm Flowchart

Similarly, the algorithm to compute the binary logarithm proposed by [12] was adapted to a finite state machine for hardware implementation setting one bit per iteration according to equation 8. The algorithm referenced by [12] takes the logarithm of x (where $x \in [0,1]$) and the finite state machine scales a number out of this range according to equation 9. To compute logarithms in base 10 and natural, SQRTLOG simply takes the binary logarithm output multiplying its result by $1/log_2(10)$ and $1/log_e(10)$, respectively.

$$y_i = \begin{cases} 1, & \text{if } x_i^2 \geq 2 \Rightarrow x_i = \frac{x_i}{2} \\ 0, & \text{if } x_i^2 < 2 \end{cases} \quad (8)$$

$$x = 2^k \times p,\ 0 \leq x \leq 1 \Rightarrow log_2(x) = k + log_2(p) \quad (9)$$

The algorihtm flowchart to compute the binary logarithm can be seen in Figure 2.

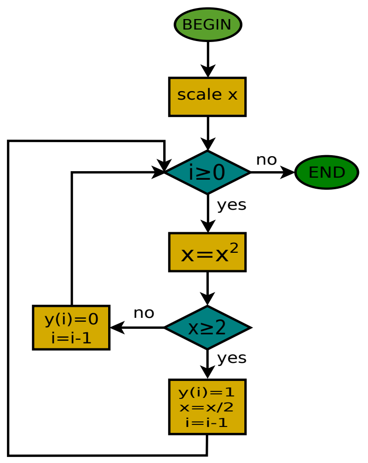

Fig. 2. Binary Logarithm Algotihm Flowchart

The square root and logarithm modules are interconnected using an interface Valid/Ready present in AMBA AXI Protocol [1] and its architecture is described in Figure 3. The signals of its interface are described in Table I and depicted in Figure 4.

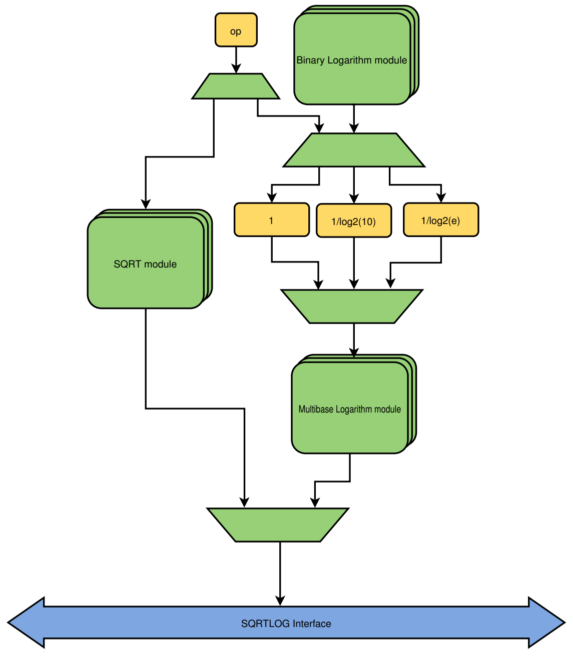

Fig. 3. The SQRTLOG architecture

TABLE I
SQRTLOG: DESCRIPTION OF THE INTERFACE SIGNALS

Signal name	Port type	Size in bits	Signal description
clock	input	1	clock signal
reset	input	1	reset signal
op	input	2	operation code that selects one computation (*sqrt*, log_2, log_{10} or log_e)
data_in	input	N_BITS	input number generated by the source to stimulate the SQRTLOG
data_out	output	N_BITS	output number computed by the SQRTLOG
iReady	output	1	handshake signal that indicates that the destination is ready to receive the number
oReady	input	1	handshake signal that indicates that the SQRTLOG is ready to receive the number
iValid	input	1	handshake signal that indicates that the source is ready to send the number
oValid	output	1	handshake signal that indicates that the LOG is ready to send the number
done	output	1	indicates that the computation is done

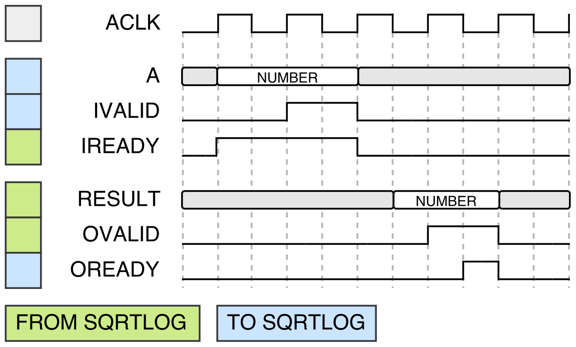

Fig. 4. SQRTLOG: Interface signals

V. RESULTS

This section discusses the synthesis results of the SQRTLOG in ASIC and in FPGA.

A. ASIC Synthesis

The module was simulated using dc_shell from Synopsys [3] with the standard cell library FSC0G_D_GENERIC_CORE from UMC [13] with $0.13\mu m$ CMOS technology using a $4MHz$ clock. During the experiment, the number of ports and cells are plotted as a function of the number of bits of N (see Figure 5).

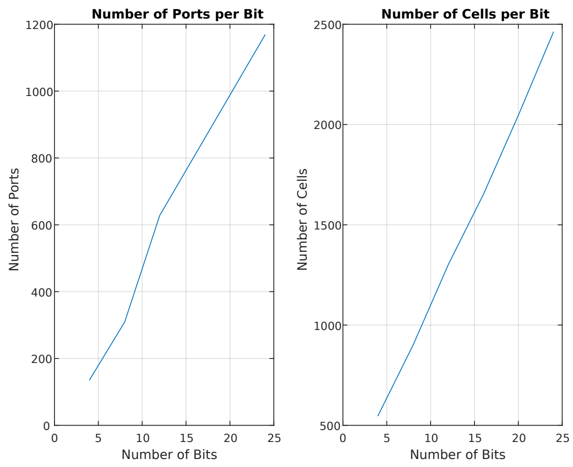

Fig. 5. SQRTLOG: Number of ports and cells vs NBITS

The area and the power consumption as a function of the number of bits of the SQRTLOG are presented in Figure 6.

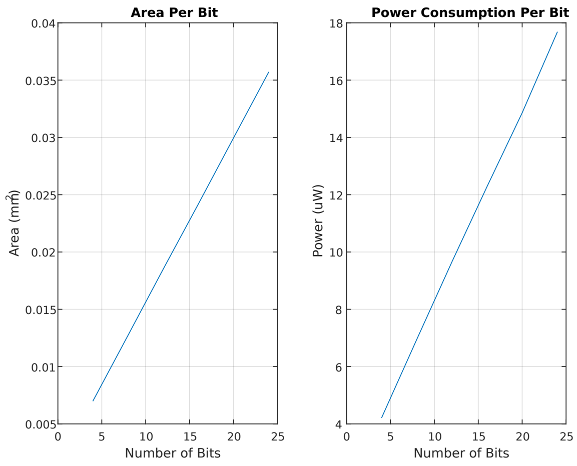

Fig. 6. SQRTLOG: Area and Power vs NBITS

The arithmetical percentage error of the module was analyzed to compute $\sqrt{\pi}$ and $ln(\pi)$ and the results can be seen in Figure 7.

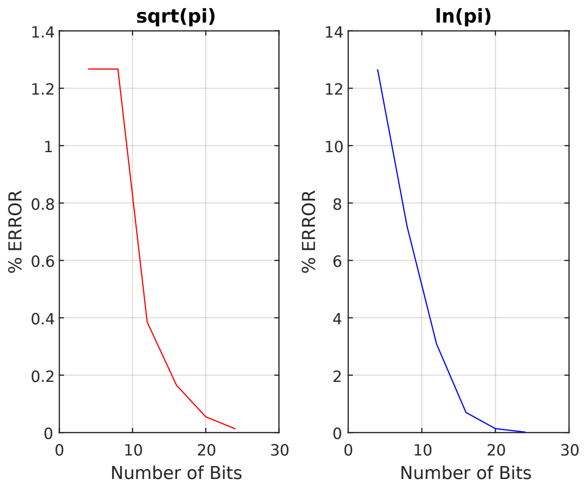

Fig. 7. Arithmetic error vs NBITS

The module was parameterized in fixed point representation with $QI.F$, with I integer bits and F fractional bits and $NBITS = I + F$. The experiment setup measured the module with $NBITS$, ranging from 4 to 24 bits, with a 4-bit step for each measurement. Table II shows that when $NBITS = 4$, it presents the lower area and power consumption. However, the Higher arithmetic error is obtained. For $NBITS = 24$, the arithmetic error decreases significantly, increasing power and area. For $NBITS = 16$, there is a good trade-off between power and area vs % error, as can be seen on Table II.

TABLE II
ASIC SYNTHESIS RESULTS

NBITS	POWER	AREA	%error sqrt(pi)	%error ln(pi)
4	$4.21\mu W$	$7010\mu m^2$	1.267	12.643
16	$9.65\mu W$	$18513\mu m^2$	0.165	0.700
24	$17.67\mu W$	$35683\mu m^2$	0.0134	0.017

B. FPGA Synthesis

The design was also synthesized (with $NBITS = 20$) in the altera DE1-SoC board using 2% of its logic utilization. The synthesis report was generated using the Quartus Prime from Altera and the results can be seen in Table III.

Using the Netlist Viewer from Quartus the circuit of the Figure 8 was obtained. Due to the square part of the logarithm algorithm, two multipliers were inferred from a total of 87 of the FPGA device.

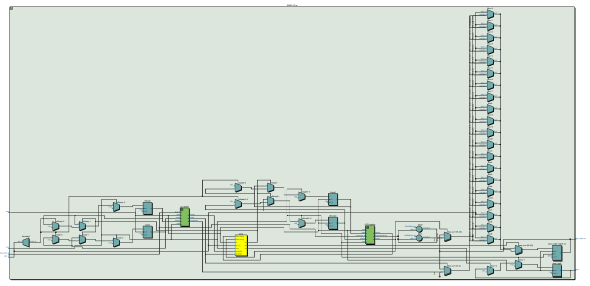

Fig. 8. Quartus Netlist of the SQRTLOG

TABLE III
FPGA SYNTHESIS RESULTS

FPGA: Cyclone V Device: 5CSEMA5F31C6	
Logic utilization (in ALMs)	518 / 32,070 (2 %)
Total registers	432
Total DSP Blocks	2 / 87 (2 %)

VI. CONCLUSIONS

This paper introduces some methods to compute the square root and logarithms in hardware implementation. It is proposed the architecture of a module called SQRTLOG, which basically computes four operations: $sqrt$, log_2 , log_{10} and log_e. Results are available for both implementations, ASIC and FPGA. It is shown that the SQRTLOG presents a good trade-off between arithmetic error and the number of bits of its input, providing a fast and accurate algorithm for digital signal processing and scientific applications.

ACKNOWLEDGMENT

The authors would like to thank the PEM (Projects for Excellence on Microelectronics) initiative for the financial support.

REFERENCES

[1] AXI AMBA. Protocol specification. *ARM, June*, 2003.
[2] Liu Bangqiang, He Ling, and Yan Xiao. Base-n logarithm implementation on fpga for the data with random decimal point positions. In *Signal Processing and its Applications (CSPA), 2013 IEEE 9th International Colloquium on*, pages 17–20. IEEE, 2013.
[3] Design Compiler. Synopsys inc, 2016.
[4] Aidong Deng, Li Zhao, and Yan Zhao. Recognition of acoustic emission signal based on mae and propagation theory. In *Management and Service Science, 2009. MASS'09. International Conference on*, pages 1–4. IEEE, 2009.
[5] Hyejung Kim, B-G Nam, J-H Sohn, J-H Woo, and H-J Yoo. A 231-mhz, 2.18-mw 32-bit logarithmic arithmetic unit for fixed-point 3-d graphics system. *IEEE journal of solid-state circuits*, 41(11):2373–2381, 2006.
[6] Yamin Li and Wanming Chu. Parallel-array implementations of a non-restoring square root algorithm. In *Computer Design: VLSI in Computers and Processors, 1997. ICCD'97. Proceedings., 1997 IEEE International Conference on*, pages 690–695. IEEE, 1997.
[7] AM Mansour, AM El-Sawy, MS Aziz, and AT Sayed. A new hardware implementation of base 2 logarithm for fpga. *International Journal of Signal Processing Systems*, 3(2):171–181, 2015.
[8] Pramod K Meher, Javier Valls, Tso-Bing Juang, K Sridharan, and Koushik Maharatna. 50 years of cordic: Algorithms, architectures, and applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(9):1893–1907, 2009.
[9] Rachmad Vidya Wicaksana Putra. A novel fixed-point square root algorithm and its digital hardware design. In *ICT for Smart Society (ICISS), 2013 International Conference on*, pages 1–4. IEEE, 2013.
[10] VB Saambhavi, SSSP Rao, and P Rajalakshmi. Design of feature extraction circuit for speech recognition applications. In *TENCON 2012-2012 IEEE Region 10 Conference*, pages 1–5. IEEE, 2012.
[11] Shashank Suresh, Spiridon F Beldianu, and Sotirios G Ziavras. Fpga and asic square root designs for high performance and power efficiency. In *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, pages 269–272. IEEE, 2013.
[12] C Turner. A fast binary logarithm algorithm. *IEEE Signal Processing Mag*, 27(5):124–140, 2010.
[13] UMC. United Microelectronics Corporation. www.umc.com.