Implementation of Babylonian Square Root Computation Algorithm with VHDL

Abdulkadir SADAY[1], Ilker Ali OZKAN[2]

[1]Selcuk University, Konya/Turkey, asaday@selcuk.edu.tr
[2]Faculty of Technology, Selcuk University, Konya/Turkey, ilkerozkan@selcuk.edu.tr

***Abstract* - One of the most important calculation operation in many systems is the calculation of the square root of numbers. In digital systems and digital signal processing there are some different algorithms to calculate the square root of numbers. Designing of the square root calculator depends on the algorithm or method for the programming format. The Babylonian method was examined and used as a different approach in this study. The digital square root calculator designed is mainly based on the Babylonian method and design was implemented with VHDL language.**

***Keywords* – VHDL, square root, Babylonian method, FPGA**

I. INTRODUCTION

Square root is a necessary and frequently used mathematical expression in applications such as image processing, sound processing, scientific calculation. Numerous computational algorithms such as mathematical predictive method and restoring method have been developed for square root calculation. In general, algorithms have been developed in order to perform the calculation process quickly and to obtain an accurate result [1, 2].

In many applications, the Newton method is adopted for the square root calculation. In this method, the approximate value of the square root is calculated by iterations. In order to obtain the iteration equation, Newton method can be used which approximately result is obtained in iteration equation. In each iteration, multiplication, addition and subtraction operations are performed and this causes process density. In order to accelerate the multiplication in the calculation, the parallel multiplier is used by partial production and then an aggregator is used to obtain the result of the production. Since multipliers require a very large number of gates, it is impractical to use multiple multipliers to achieve exact square root calculations [3].

In many of the science and engineering applications, square root calculation has an important place. In applications, the result of an operation is not always expected to be an integer. For this reason, square root calculations with decimal numbers are studied. On FPGA, operations with decimal numbers are provided by the decimal points libraries called as floating-point and fixed-point libraries. Hasnat et al [4] presented square root and inverse square root calculation by using the Quake's algorithm. In their studies, they applied square root calculation using single precision floating point library.

The current commercial DSP and embedded processors do not provide special complex number calculation units. Instead, they use a number of real basic computational processes at the software level. This parsed form of calculation results in reduced computational performance due to process complexity. The software unit called Cordic is an advanced library that allows you to handle process with complex and trigonometric numbers. Cordic provides to done complex arithmetic operations with only summing and shifting operation [5].

In their study, Leeser and Wang designed the square root calculation with the floating-point library of variable precision. Their calculator is based on the computational algorithm called Taylor Series in the literature. In their calculator, a small lookup table and small multipliers are used to obtain the first few terms of the Taylor series. The methodology they offer is in IEEE standard format and is flexible so that different formats can be obtained from their method. They stated that their methods support zero and nearest rounding property [6].

In addition to speed, it has an important place that the field used in calculation operations. In the literature, there are studies in which the minimum field is used within the square root calculation algorithms. This algorithm is based on the formula called Dwandwa Yoga. In their study, Kachhwal and Rout presented square root calculators based on the old Indian mathematical formula [7]. Their calculator generates 16-bit floating-point output versus 24-bit floating-point input. They argued that the method presented by them had the lowest area usage.

In this study, square root calculator was performed based on Babylon method. While calculating perfect-square numbers is easy in mathematics, the calculation of the roots of non-perfect square numbers are very difficult. In the Babylon method, the square root is calculated by a numerical repeating process. The first algorithms for the square root calculation process are known to be found by Babylonians [8].

II. MATERIAL AND METHOD

In this study, the Babylonian method, called the simple repeating algorithm, was used to calculate the square root of numbers. This algorithm is based on the repetition of simple operations until the result is reached regardless of whether the number is a perfect-square number.

In the Babylonian method, it is stated that the constant number that the series of numbers generated by the iteration algorithm for any initial value approaches is the result of the

square root of the number to be rooted [8]. The square root value of a number starts with an estimate of any initial value. This randomly selected value is the initial value of the repetition rule, and each new value obtained as a result of the operation is the new initial value.

In the Babylonian method, the calculation in the repetition process is made by using Equation 1.

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \quad (1)$$

A set of numbers is obtained by applying the repetition rule with any initial value in the equation. The number approached and repeated as a result of the sequence of numbers is the square root of the desired number.

If it is wanted to calculate the square root of the number 12 as an example of calculating the square root with the Babylon method, first any number should be selected for the initial value. With this initial value, the repetition rule must be applied and the number that is repeating in the number array must be found. The result can be expressed as a repetitive number. The square root calculation of the number 12 depending on the algorithm is given in Figure 1.

$$x_0 = 2$$
$$x_1 = \frac{1}{2}\left(x_0 + \frac{a}{x_0}\right) = \frac{1}{2}\left(2 + \frac{12}{2}\right) = 4{,}00$$
$$x_2 = \frac{1}{2}\left(x_1 + \frac{a}{x_1}\right) = \frac{1}{2}\left(4 + \frac{12}{4}\right) = 3{,}50$$
$$x_3 = \frac{1}{2}\left(x_2 + \frac{a}{x_2}\right) = \frac{1}{2}\left(3{,}50 + \frac{12}{3{,}50}\right) = 3{,}46..$$
$$x_4 = \frac{1}{2}\left(x_3 + \frac{a}{x_3}\right) = \frac{1}{2}\left(3{,}46 + \frac{12}{3{,}46}\right) = 3{,}46..$$

Figure 1 – The square root calculation of 12 by the Babylon method

As shown in Figure 1, a random number is assigned for x0 and the repetition rule is calculated based on this initial value. At each step, the result was calculated using the new initial value. The result is repeated for 4 steps as shown in the number array. In this case, the repetitive number is the square root of the number given at the entry. When this process is done with the calculator, it is concluded that the square root of the number 12 is 3.46.

Compiled libraries as standard can be used in the calculation of arithmetic operations with FPGA. Floating-point and fixed-point libraries are standard libraries that allow operations with decimal numbers. Fixed-point is frequently used for simple decimal numbers, while floating-point is used for more complex decimal numbers. Due to its structure fixed-point is an easier and faster library to implement and calculate. It is frequently used in applications in which result can be predicted and which requires speed [9].

In the literature, many algorithms such as Rough estimation method, exponential identification, Taylor series, Newton Raphson method and digit-by- digit method are used in the studies. There are many applied methods of square root calculation. They can be grouped into two classes as predictive and digit-by-digit methods. The digit-by-digit calculation method is generally divided into two groups, restoring and non-restoring. In this method, the computational speed is lower than other methods [9, 10].

The implementation of the algorithm that returns the approximate square root result by the Babylon method on FPGA is uncomplicated because that the method contains simple mathematical expressions. Therefore, the use of a fixed-point library is sufficient to implement this algorithm and to obtain results. The use of fixed-point provides the advantage of quick calculation for applications requiring a square root calculator [11].

In the fixed-point library, the definitions are made in the form of sfixed (signed) and ufixed (unsigned) numbers. In defining the decimal number by VHDL, the exact part and the decimal part of the number are expressed separately. The definition is made by specifying a positive integer of the desired length for the full part of the number and negative integers for the decimal part. The expression of a sample number with the fixed-point library is can be defined as $x = sfixed(10\ downto - 8)$.

The definition shown in Figure 2 indicates that the full part of the x number is 11-bit and the decimal part is 8-bit. As the value defined for the full part increases, the magnitude of the number that can be expressed increases, as the value defined for the decimal part increases, the decimal precision of the number to be expressed increases. In the calculation of the numbers expressed by the fixed-point, multipliers 2^0, 2^1, 2^2.. are used for the full part and 2^{-1}, 2^{-2}, 2^{-3}.. for the decimal part. If the number is negative, the first bit indicates the sign and the calculation is made as 1's complement to the number.

In the implementation of the Babylon method with VHDL, the repetition rule is provided by a simple loop. The initial estimate value is calculated by dividing the number by the largest and smallest multiplier. In the calculator, 4 variables are used as number, initial values, temporary value and output. The square root of the value defined by the number variable is kept at the temporary value until the output value is repeated. In the repetition where the output value is equal to the temporary value, the square root of the number is found.

III. RESULTS

FPGAs are programmed with programming languages called Verilog and VHDL [7]. In this study, the square root calculation process based on the approached Babylon method is emphasized and the square root calculator software is realized.

The realized square root calculator was run on Nexys 4 DDR FPGA development board of Digilent company. As input, the 16-bit switch on the development board is used and the result calculated on the output is expressed with 16-bit led. In order to indicate the completion of the calculation process, a status changing led is assigned to the result and a change in status is provided after the calculation.

The flowchart of the calculation of the Babylon method using FPGA with VHDL programming language is given in Figure 2.

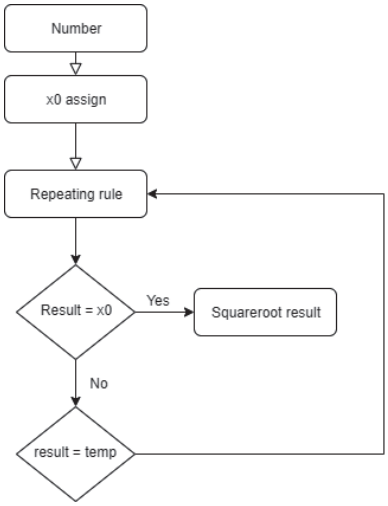

Figure 2 – Babylonian Algorithm flow chart

The process flow shown in Figure 2 starts with the assignment of the number to the input. With the assignment of the number, a random number is assigned to the initial value x_0. This assignment is performed in the random number generation block generated by the VHDL and is valued according to the multipliers of the input number. After the initial value is assigned, the repeating rule loop given in Equation 1 is executed. The value calculated during the loop is assigned to the temporary variable and held in it until the next calculation. If the temporary variable is equal to the calculated result, the square root calculator block terminates by returning the result. If the result is not equal to the transient variable, the repeating rule is reapplied, and this loop continues until it is equal.

The result obtained after the loop gives the square root of the entered number. In integer operations, the result gives the value closest to the actual value. The rounding of the number up or down varies depending on the programmer's choice during programming. In this study, square root calculator was realized using fixed-point library. Thus, even if the square root value of the entered number is a decimal number, it can be expressed.

Several examples of numbers calculated with the performed VHDL software are shown in Figure 3.

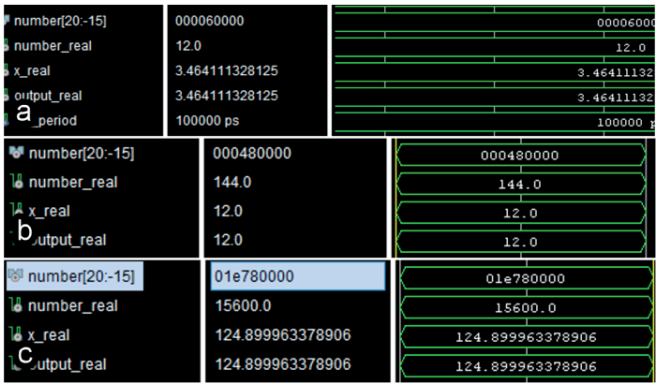

Figure 3 – Square root calculations of different numbers with FPGA (a) 12 (b) 144 (c) 15600

The square root value 3.46 of the number 12 calculated in Figure 1, is the same as the algorithm result performed on FPGAs. As can be seen in Figure 4, the square root calculator with the Babylon method works on FPGA in a way that yields successful results in decimal numbers.

Calculations were tested at different clock frequencies of the processor with different values. For 10us, 1us and 10ns clock pulses, the results were checked for each rising edge. According to the results obtained, the algorithm works at a speed that can respond in a clock pulse. It was observed that the calculation time did not change by keeping the numbers to be calculated at very high values and assigning the predictive initial variable at far values. However, depending on the application that use the square root calculator and the complexity of the numbers, it is possible to obtain the calculation result at different time intervals.

As a result of this study, a square root calculator based on the Babylonian method, which can calculate with integer and decimal numbers and which the calculation result can be obtained as a decimal number has been realized by using FPGA with VHDL programming language.

REFERENCES

1. Jidin, A.Z. and T. Sutikno, *FPGA Implementation ofLow-Area Square Root Calculator.* Telkomnika, 2015. **13**(4): p. 1145.
2. Nanhe, A., et al., *Implementation of fixed and floating point square root using nonrestoring algorithm on FPGA.* International Journal of Computer and Electrical Engineering, 2013. **5**(5): p. 533.
3. Li, Y. and W. Chu. *A new non-restoring square root algorithm and its VLSI implementations*. in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors*. 1996. IEEE.
4. Hasnat, A., et al. *A fast FPGA based architecture for computation of square root and inverse square root*. in *2017 Devices for Integrated Circuit (DevIC)*. 2017. IEEE.
5. Yang, B., D. Wang, and L. Liu. *Complex division and square-root using CORDIC*. in *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*. 2012. IEEE.
6. Leeser, M. and X. Wang, *Variable precision floating point division and square root*. 2005, NORTHEASTERN UNIV BOSTON MA DEPT OF ELECTRICAL AND COMPUTER ENGINEERING.
7. Kachhwal, P. and B.C. Rout. *Novel square root algorithm and its FPGA implementation*. in *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014)*. 2014. IEEE.
8. Flannery, D., *The square root of 2: A dialogue concerning a number and a sequence*. 2006: Springer Science & Business Media.
9. Sutikno, T., *An efficient implementation of the non restoring square root algorithm in gate level.* International journal of computer theory and engineering, 2011. **3**(1): p. 46.
10. Zhu, H., Z. Lei, and F.P. Chin, *An improved square-root algorithm for BLAST.* IEEE Signal Processing Letters, 2004. **11**(9): p. 772-775.
11. Li, Y. and M. Leeser, *HML: an innovative hardware description language and its translation to VHDL*. 1995: IEEE.