NON RESTORING ALGORITHM FOR SQUARE ROOT

B.Sneha Priya[1]., A.Pravalika[2]., G.Akshitha[3]., L.Akshitha Reddy[4] ., B.Sreeja[5]

1 Assistant Professor, Department of ECE., Malla Reddy College of Engineering for Women., Maisammaguda., Medchal., TS, India (✉ budhasnehapriya@gmail.com)

2, 3, 4, 5 B.Tech ECE, (19RG1A04C5, 19RG1A04D7, 19RG1A04F3, 19RG1A04C9), Malla Reddy College of Engineering for Women., Maisammaguda., Medchal., TS, India

Abstract

This article recommends a method for effectively using the FPGA in a fully pipelined design by abstracting the algorithm at the gate level using VHDL. Controlled subtract-multiplex (CSM) is presented as a novel fundamental building component. The suggested approach has the same basic idea as the standard non-restoring algorithm, but it avoids the usage of the add operation and appends 01 to the result instead. The suggested method has been used effectively to develop FPGA and provides a useful hardware resource.

Key words

square root, Field Programmable Gate Array (FPGA), non-restoring, Gate Level

INTRODUCTION

Multimedia, data processing and control, and even digital signal processing (DSP) techniques all rely heavily on the square root function [1-6]. This issue arises often in computational number theory and is a traditional one for which an accurate solution is difficult to get [7-8]. Rough estimation, the Babylonian method, the exponential identity, the Taylor-series expansion algorithm, the Newton-Raphson method, the Sweeney-Robertson-Tocher redundant method, the Sweeney-Robertson-Tocher non redundant method, and the sequential algorithm (the digit-by-digit method) are just some of the many square root algorithms that have been studied, developed, and implemented. The square root operation in the aforementioned techniques is performed in early processors using software, which results in lengthy delays [6]. Hardware implementation of the square root function gained favour [6] as technology advanced to the point that it became feasible to combine massive circuits on a single chip, and as the need for quicker computational execution time grew.

Unfortunately, the square root computation is not simple to implement on field programmable array (FPGA) technology due to the complexity of the square root algorithms [1, 3, 5, 10]. Some square root algorithms have been designed for use with FPGA hardware. There are essentially two types of them. Estimation techniques, such as the CORDIC, DeLugish, and Chen methods, and the Newton-Raphson approach fall under the first group, whereas the second is known as the digit-by-digit method. Finally, it's important to differentiate between restoring and non-restoring algorithms when classifying additional digit-by-digit methods. There is a severe restriction on the restoring algorithm's regular flow during the restoring stage. This is primarily why it is no longer utilised, despite having been the pioneering approach that inspired all the others [11]. The non-restoring approach, which does not restore the remainder, requires less hardware resources to implement and is thus easier to implement in hardware. It's optimised for FPGA implementation and works well with IEEE rounding standards [1-3, 6]. Multiple approaches or designs have been developed to implement the non-restoring digit-by-digit square root method on FPGA hardware. A non-restoring approach, completely pipelined and iterative, and requiring neither multipliers nor multiplexors was presented by Yamin and Waning [1-2, 9]. As fundamental components, they presented the carry save adder (CSA) and the carry propagate adder (CPA).

A SYSTEM FOR COMPUTING BY DIGITS

Each digit of the square root is located in a sequence in which one digit is created at each iteration in the digit-by-digit computation technique [2, 6, 13]. There are a number of benefits to this method, including the fact that the algorithm can be used with any number base, the fact that once the root has been found, all of its digits are correct and won't need to be changed, and the fact that if the square root needs to grow, the process will end once the last digit has been found (of course the process depends on number base). There are two main types of algorithms in this category; those that restore data and those that don't [6, 7]. The restoring algorithm's process is consisting of the steps of taking the current remainder, shifting it by one, adding one, and removing the resulting square root. The 1 in 01 is a new guess bit, while the 0 represents a multiplication by 2. If the resultant remaining is positive, the newly created root bit is 1, and if it is

Copyright @ 2022 Authors

negative, the remainder must be restored by adding the amount just deducted. In contrast, if the outcome of a subtraction operation is negative, the non-restoring algorithm will not restore the subtraction. Instead, it takes the current root and adds 11 to it before moving on to the next iteration. If the excess occurs during addition, the following cycle will revert to subtraction [15]. The restoring and non-restoring algorithms in Figure 1 (a) and (b) use the binary square root of 01011101 (corresponding with 93 decimal) as an example.

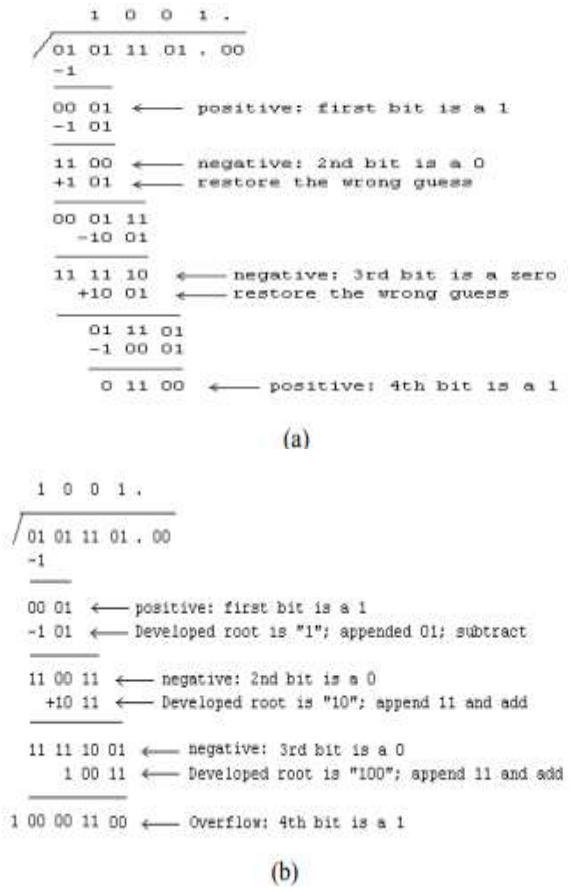


Figure 1. The example of digit-by-digit calculation to solve square root: (a) restoring algorithm; (b) non-restoring algorithm

CONCEIVED SQUARE-ROOT ALGORITHM

Figure 2 depicts a little deviation from the typical non-restoring digit-by-digit technique presented in Figure 1(b) that may be used to achieve easier implementation and quicker computation. To make this adjustment, the add operation and append 11 have been removed, leaving just the subtract operation and append 01 in use.

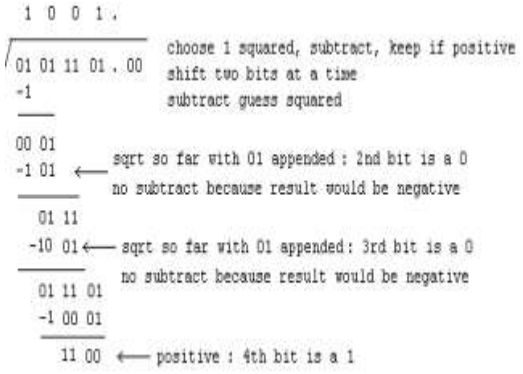


Figure 2. The example of using modified non-restoring digit-by-digit calculation algorithm to solve square root

By removing unnecessary components, Samavi et al. [6] have enhanced the performance of the basic non-restoring digit-by-digit square root circuit. The non-restoring, low-area circuit that they use. But its fundamental building blocks remain the addition and subtraction of numbers with a constant digit of 01 or 11. (still refer to Figure 1 b). In this work, we provide a less complex option that requires just a subtraction operation and the addition of 01. As a result, the subtract-multiplex serves as the fundamental building component (refer to Figure 2). Illustration of the suggested algorithm's core concept.

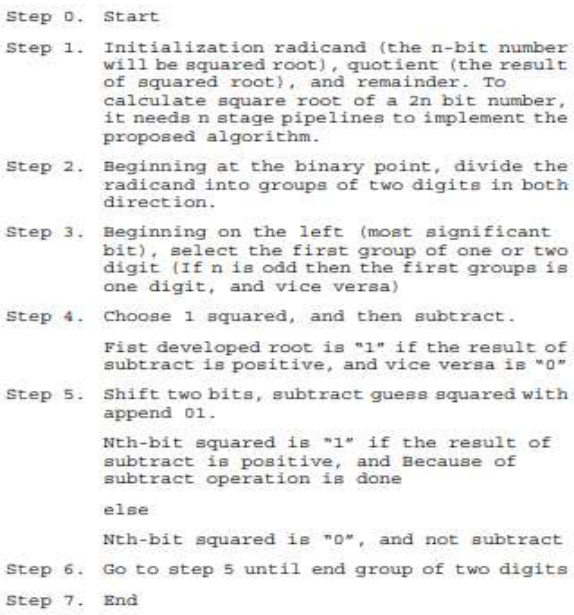


Figure 3. The principle of proposed algorithm to solve square root

A simple hardware implementation of the proposed non-restoring digit-by-digit algorithm for unsigned 6-bit square root by an array structure is shown in Figure 4. The radicand is P (P5,P4,P3,P2,P1,P0), U (U2,U1,U0) as quotient and R (R4,R3,R2,R1,R0) as

Copyright @ 2022 Authors

remainder. It can be shown that the implementation needs 3 stage pipelines. The

basic building blocks of the array are blocks called as controlled subtract-multiplex (CSM). Figure 5 present the details of a CSM. Input of the building block is x,y,b and u, and as an output is bo(borrow) and d result). If u=0, then d<=x-y-b else d<=x

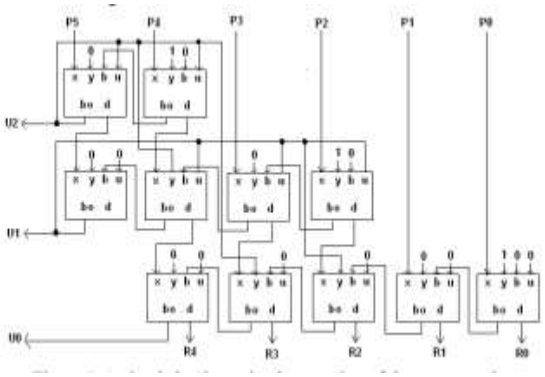


Figure 4. A simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned 6-bit square root

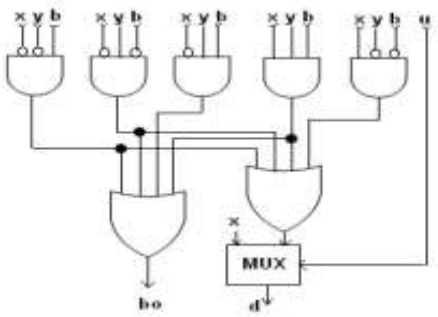


Figure 5. Internal structure of a CSM block

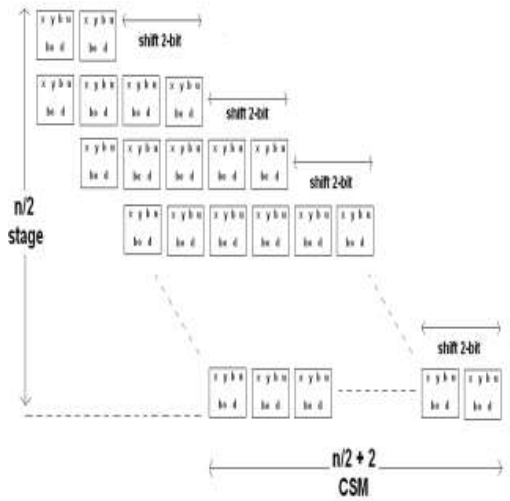


Figure 6. A simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned n-bit square root

The generalization of simple implementation of the non-restoring digit-by-digit algorithm for unsigned n-bit square root by an array structure is shown in Figure 6. Each row (stage) of the circuit in Figure 6 executes one-iteration of the non-restoring digit-by-digit square root algorithm, where it only uses subtracts operation and appends 01. To optimize hardware resource utilization of the implementation above, specialized entities can be created as building block components. It will eliminate circuitry that is not needed. As example, the implementation in Figure 6 for unsigned 6-bit square root can be optimized become as shown in Figure 7. The specialized entities A, B, C, D and E are minimized CSM when input ybu=100, yu=00, u=0, yu=10, and y=0 respectively, and the remainder is ignored. The generalization of optimized simple implementation of the non-restoring digit-by-digit algorithm for unsigned n-bit square root is shown in Figure 8.

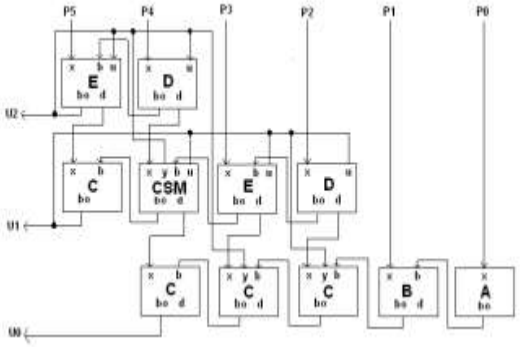


Figure 7. Optimized simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned 6-bit square root

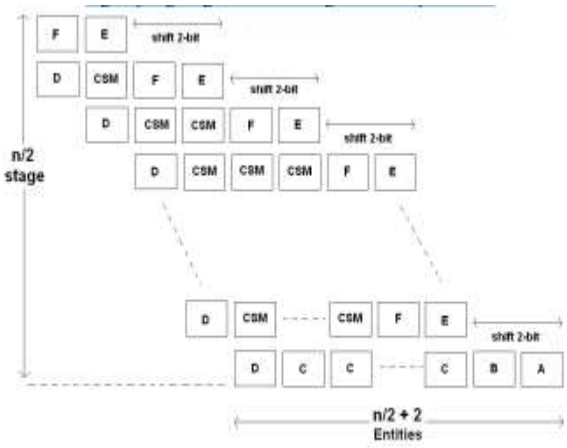


figure 8. Optimized simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned n-bit square root

THE IMPLEMENTATION OF THE NON-RESTORING SQUARE ROOT ALGORITHM IN GATE LEVEL

The implementation of the proposed non restoring square root algorithm in gate level approach is conducted in VHDL language. The VHDL source

Copyright @ 2022 Authors

codes for modules A, B, C, D, E, F and CSM is shown in Figure 9, 10, 11, 12, 13, 14 and 15 respectively

```
-- modul A
library IEEE;
use IEEE.std_logic_1164.all;

entity A is
  port ( x  : in  std_logic;
         bo : out std_logic);
end S0b;

architecture circuits of A is
begin  -- circuits of S0b
  bo <= not x;
end;
```

Figure 9. Source code for module A

```
-- module B
library IEEE;
use IEEE.std_logic_1164.all;

entity B is
  port ( x  : in  std_logic;
```

Figure 10. Source code for module B

```
         b  : in  std_logic;
         bo : out std_logic);
end S1b;

architecture circuits of B is
begin
  bo <= (not x) and b;
end;
```

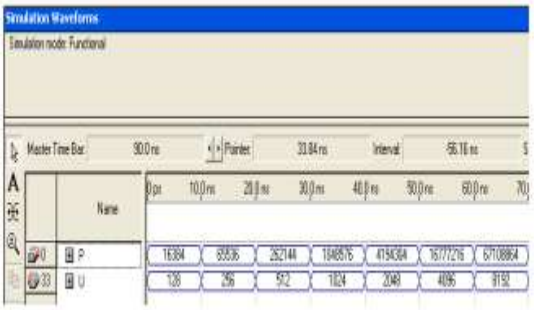
Figure 11. Source code for module C

```
-- module C
library IEEE;
use IEEE.std_logic_1164.all;

entity C is
  port ( x  : in  std_logic;
         y  : in  std_logic;
         b  : in  std_logic;
         bo : out std_logic);
end;

architecture circuits of C is
  signal t011, t111, t010, t001 : std_logi
begin
  t011 <= (not x) and y and b;
  t111 <= x and y and b;
  t010 <= (not x) and y and (not b);
  t001 <= (not x) and (not y) and b;
  bo   <= t011 or t111 or t010 or t001;
end;
```

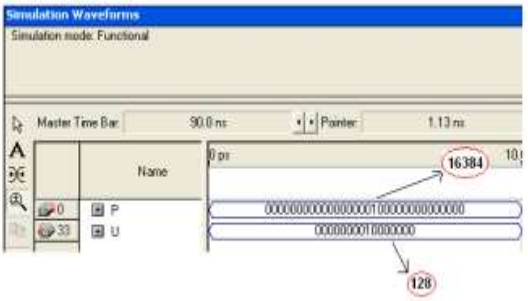
Figure 12. Source code for module D

RESULTS AND ANALYSIS

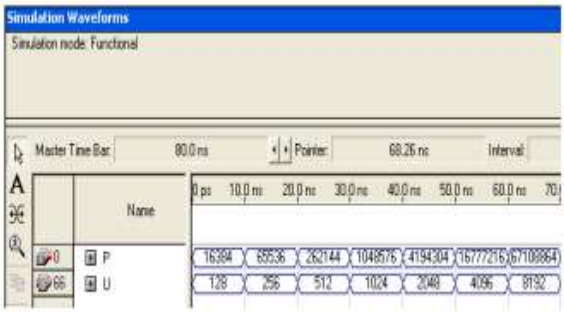
An efficient, low-overhead hardware implementation of the non-restoring digit-by-digit technique for square root was described before. Here, we provide the above-mentioned method's simulated results for 32-bit and 64-bit square root on the Altera APEX 20KE FPGA in Figure 18. P is the radicand and U is the quotient in this hypothetical situation. The findings confirmed the success of the deployment. As shown in the compilation report, 256 and 1023 logic elements (LE) are required to implement 32-bit and 64-bit square root utilising the optimised simple hardware implementation approach of the non-restoring digit-by-digit algorithm on an Altera FPGA APEX 20KE. Table 1 displays a comparison of the outcomes achieved using various implementation methods. This table compares the use of LEs and LCs, based on the research found in sources [6] and [16]. It has shown to be quite useful in terms of lowering the amount of hardware resources needed to perform a given task. This is because, as seen in Figure 8, we have adopted a completely pipelined design and simplified CSM.



(a)

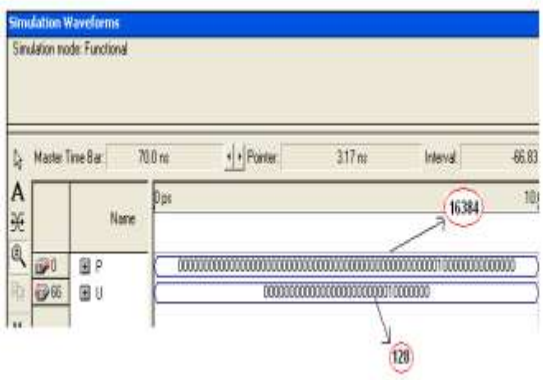


(b)



(c)

Copyright @ 2022 Authors



(d)

Figure 18. Simulation result of n-bit square root using optimized simple hardware implementation method of the non-restoring digit-by-digit algorithm: (a) 32-bit in decimal display, (b) 32-bit in binary display, (c) 64-bit in decimal display, (d) 64-bit in binary display.

TABLE 1. THE COMPARISON OF LOGIC ELEMENT USAGE

No	Method	LEs Usage	
		32-bit square root	64-bit square root
1	Classical-NR	1008	4092
2	Reduced-Area-NR	632	2464
3	Modular-NR	624	2468
4	Simple-X-Module	648	2488
5	Proposed	**256**	**1023**

Based on [16], for Altera APEX 20KE & Xilinx Virtex-E, 1 LC = 1 LE, and 1 CLB = 4 LE

TABLE 2. THE LIST OF LC/LE USAGE USING VARIOUS ALTERA FPGA FAMILIES

No	FPGA Families	LEs Usage	
		32-bit square root	64-bit square root
1	FLEX10K	256	1024
2	ACEX1K	256	1024
3	Cyclone	256	1025
4	Cyclone II	256	1025
5	Cyclone III	256	1025
6	Stratix	256	1025
7	Stratix GX	256	1025
8	APEX20KE	256	1023
9	Arria GX	185	736
10	Stratix II	185	736
11	Stratix III	185	736

Many different FPGA families are tested in an effort to implement the suggested method. The tally of LC/LE applications is shown in Table 2. The "hardware resource" (LE) size of an implemented circuit is proportional to the number of LE used. It was shown that the suggested strategy makes the best use of available hardware. This is understandable given that it has a completely pipelined design and only makes use of the subtract operation and append 01 (instead of the add operation and append 11). The findings show that the suggested method is simple to implement, requires less resources, and can be scaled to address more complex square root problems in FPGA implementation.

CONCLUSION

This work offered an adaptation of the standard non-restoring digit-by-digit computation approach for use with fully pipelined FPGA hardware. The suggested technique relies on a two-bit shifter and a subtractor-multiplexer, with the subtract operation and append 01 replacing the add operation and append 11. FPGA-based unsigned 32-bit and 64-bit binary square root has been successfully implemented using the suggested technique. Comparing the suggested technique to previous studies, the findings reveal that it makes the best use of hardware resources. To address the challenging square root issue in FPGA implementation, the technique may be simply scaled to a bigger number.

REFERENCES

[1] L. Yamin and C. Wanming, "Implementation of Single Precision Floating Point Square Root on FPGAs," in IEEE Symposium on FPGA for Custom Computing Machines, Napa, California, USA, 1997, pp. 226-232.

[2] L. Yamin and C. Wanming, "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, 1997, pp. 690-695.

[3] K. Piromsopa, et al., "An FPGA Implementation of a fixed-point square root operation," presented at the Int. Symp. on Communications and Information Technology (ISCIT 2001), ChiangMai, Thailand, 2001.

[4] D. R. Llamocca-Obregon, "A Core Design to Obtain Square Root Based on a Non-Restoring Algorithm," presented at the IBERCHIPS Workhsop, Salvador Bahia, Brazil, 2005.

[5] XiaojunWang, "Variable Precision Floating-Point Divide and Square Root for Efficient FPGA Implementation of Image and Signal Processing Algorithms," Doctor of Philosophy, Electrical and Computer Engineering, Northeastern University, Boston, Massachusetts, 2007.

[6] S. Samavi, et al., "Modular array structure for non-restoring square root circuit," Journal of Systems Architecture, vol. 54, pp. 957-966, 2008.

[7] H. Dong-Guk, et al., "Improved Computation of Square Roots in Specific Finite Fields," Computers, IEEE Transactions on, vol. 58, pp. 188-196, 2009.

[8] S. Lachowicz and H. J. Pfleiderer, "Fast Evaluation of the Square Root and Other Nonlinear Functions in FPGA," in Electronic Design, Test and Applications, 2008. DELTA 2008. 4th IEEE International Symposium on, 2008, pp. 474-477.

[9] W. Chu; and Y. Li;, "Cost/Performance Tradeoff of n-Select Square Root Implementations," in 5th Australasian Computer Architecture Conference (ACAC 2000), Canberra, ACT 2000, pp. 9-16.

Copyright @ 2022 Authors

[10] J. Xiaoliang, "Implementation of Square Root Arithmetic Based on FPGA," Modern Electronics Technique, vol. 30, 2007.

[11] P. Montuschi and M. Mezzalama, "Survey of square rooting algorithms," in Computers and Digital Techniques, IEE Proceedings E, Italy, 1990, pp. 31 - 40.

[12] A. J. Thakkar and A. Ejnioui, "Design and implementation of double precision floating point division and square root on FPGAs," in Aerospace Conference, 2006 IEEE, 2006, p. 7 pp.

[13] W. Xiumin, et al., "A New Algorithm for Designing Square Root Calculators Based on FPGA with Pipeline Technology," in Hybrid Intelligent Systems, 2009. HIS '09. Ninth International Conference on, 2009, pp. 99-102.

[14] G. Renxi, et al., "Hardware Implementation of a High Speed Floating Point Multiplier Based on FPGA," in 4th International Conference on Computer Science & Education, Nanning, Guangxi, P.R.China, 2009.

[15] S. Dattalo. (2000, March 17, 2010). Square Root Theory. Available: http://www.dattalo.com/technical/theory/sqrt.html

[16] March 30, 2010). Comparing Altera APEX 20KE & Xilinx Virtex-E Logic Densities. Available: http://www.altera.com/products/devices/apex/features/apx-compdensi ty.html

Copyright @ 2022 Authors