A Novel Fixed-Point Square Root Algorithm and Its Digital Hardware Design

Conference Paper · June 2013

DOI: 10.1109/ICTSS.2013.6588110

1 author:

Rachmad Vidya W. Putra
TU Wien
85 PUBLICATIONS **742** CITATIONS

SEE PROFILE

A Novel Fixed-Point Square Root Algorithm and Its Digital Hardware Design

Rachmad Vidya Wicaksana Putra

Microelectronics Center - IC Design Laboratory
Institut Teknologi Bandung
Jalan Tamansari 126, Bandung, West Java, INDONESIA
rachmadvidyawp@design.paume.itb.ac.id, rachavidyawp@gmail.com

***Abstract*— Square root operation is one of the basic important operation in digital signal processing. It will calculate the square root value from the given input. This operation is known hard to implement in digital hardware because of the complexity of its algorithm. There were many researches related to this topic to obtain the optimum design between area consumption and speed. Regarding this condition, we propose an alternative square root algorithm which is based on two approaches, digital binary input decomposition and iterative calculation. Its fixed-point digital hardware implementation is very simple, low complexity, and resource-efficient. It doesn't need any correction adjustments and directly produces accurate value of square root result and remainder in (N/2)+1 clock cycles, which N represents the wordlength of input. This design has been synthesized for FPGA target board Altera Cyclone II EP2C35F672C6 and produced good results in resource consumption and speed.**

Keywords*— *Novel square root algorithm; iterative calculation; fixed-point; simple; low complexity; resource-efficient

1. INTRODUCTION

In digital hardware, we know there are several basic calculations: addition, subtraction, multiplication, division, and square root. From those several calculations, square root is one of the most difficult operation to implement in digital hardware because of the complexity of its algorithm [1].

Regarding this problem, there were several methods developed. In the previous researches, there were three main methods developed for square root algorithms: SRT-redundant method, non-redundant method, and Newton-Raphson method [2]. Paper [3] was the example of non-redundant method which proposed new non-restoring algorithm. Paper [4] proposed four alternative designs of low-cost fully pipelined square root architecture based on a non-restoring remainder algorithm. Paper [5] discussed how to optimize the design implementation of non-restoring square root algorithm for FPGA. Paper [6] proposed about an efficient hardware implementation of non-restoring based square root algorithm in gate level. Besides those researches, there was a proposal of VHDL coding simplification for non-restoring square root algorithm [7]. Different from previous works, paper [8] proposed multi-cycle square root design.

In this paper, we propose a novel fixed-point square root algorithm which is different from previous square root algorithm. Principally, this algorithm uses binary input representation and iterative calculation process. Binary input representation is used to build binary decomposition method for calculation. Iterative process is used to accommodate subtraction processes in order to get square root and remainder value. This algorithm can give the exact result of square root and remainder without any additional adjustments and correction calculations. Those approaches lead its hardware implementation has small resource consumption and small critical path delay.

This paper is organized in a number of sections. First, the introduction section which briefly introduces several past researches about square root algorithm and our research background. It is followed by explanation of several previous works which used binary input decomposition approach. The next two sections are explanation of our proposed algorithm and its hardware implementation. Here, we explain our design and its implementation in detail. It is followed by simulation and synthesis results. Finally, this section is followed by conclusion and references, respectively.

2. PREVIOUS WORKS

There are two methods developed using binary input decomposition method : restoring and non-restoring algorithm.

A. Restoring Algorithm

Restoring algorithm is one of the most popular method in square root algorithm. Principally, restoring algorithm will calculate square root and remainder value in iterative process. Assume that we have D as input data (radicand), Q as square root result (quotient), and R as remainder. Restoring algorithm will calculate and update the value of Q and R iteratively. The algorithm will guess the operand value for subtraction or addition process to the current R. If there was a wrong guess, it has to restore previous value. So, it will modify each bit of Q twice [3]. The illustration example can be found in [6].

B. Non-Restoring Algorithm

Besides restoring algorithm, there is non-restoring algorithm. It is also one of the most popular method in square root algorithm. This algorithm is almost similar to restoring algorithm. It will calculate square root and remainder value in iterative process. Assume that we have D as input data (radicand), Q as square root result (quotient), and R as remainder. Non-restoring algorithm will calculate and update

the value of Q and R iteratively. The algorithm will guess the operand value for subtraction or addition process to the current R. If there was a wrong guess, it doesn't have to restore previous value. So, it will only modify each bit of Q once [3]. The illustration example can be found in [6].

3. Proposed Algorithm

The equation for square root calculation can be written as:

$$D = Q^2 + R \tag{1}$$

From (1), we get three variables *D, Q,* and *R*. *D* is an input variable of square root calculation. It is usually called as radicand. Result of square root calculation is represented as *Q*. It is usually called as quotient. The last, remainder value is represented as *R*. It is a big advantage if the algorithm can calculate and give the exact value of square root *Q* and remainder *R* at once altogether. The design will be simple, low complexity, and resource-efficient.

In order to achieve those targets, we propose a new square root algorithm which is based on iterative process and binary input decomposition method. Here is the square root algorithm:

1. Start
2. Prepare input data D (radicand), remainder R, square root Q (quotient), partial factor F, and bit-index i.
3. Initalize radicand with input data value, R=0, Q=0, F=0, i=n; n is MSB bit-index of D.
4. If radicand has odd digits
   Expand radicand with a bit of "0" as MSB.
   Then, go to step 5.
   Else (even digits)
   Go to step 5.
5. Divide the radicand into sub-groups which each sub-group consists of 2 digits starting from integer LSB.
6. Start the calculation from MSB sub-group to LSB sub-group. Treat current sub-group as current partial remainder.
   $R_t = D_t[i:i-1]$; t is time index indicator.
7. Do a comparison whether current partial remainder is bigger or equal than current partial factor $((F_t<<1)|1)$.
   If yes
   Update Q; $Q_{t+1} = (Q_t<<1)|1$;
   Update F; $F_{t+1} = ((F_t+F_t[0])<<1)|1$;
   Else
   Update Q; $Q_{t+1} = (Q_t<<1)|0$;
   Update F; $F_{t+1} = ((F_t+F_t[0])<<1)|0$;
8. Do subtraction to partial remainder by the result value of factor multiplication; Then append the subtraction result with next sub-group data of D in the LSB position of partial remainder, in order to update R.
   $R_{t+1} = ((R_t-(F_t*F_t[0]))<<2)|D[i-2:i-3]$;
9. Update the current indexes for next use.
   t+1 is changed into t;
   i-2 is changed into i;
   i-3 is changed into i-1;
10. If the process is not over
    Jump to step 7 and loop the process.
    Else (process is over)
    Latest Q value is final square root value.
    Latest R value is final remainder value.
11. End

Those are eleven steps of proposed algorithm. From this algorithm, we can illustrate step by step how the algorithm is actually working. Illustration of calculation process in this algorithm can be seen in Fig. 1.

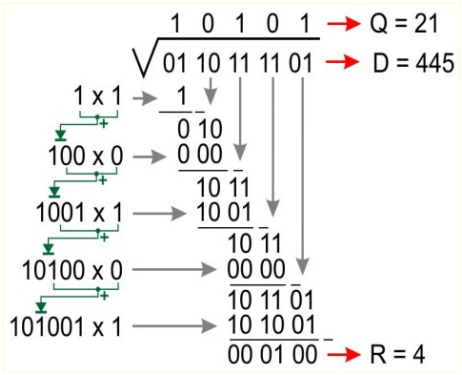

Fig 1. Illustration of proposed square root calculation

Fig. 1 shows us the example of calculation process step by step. For the example, we have input D = 445. Binary representation of D is 110111101_2. First, we have to initialize the value of *R, Q,* and *F* as "0". Then, we will divide the radicand data. Because the digits of radicand are odd, so it has to be added with "0" in MSB becomes "01 10 11 11 01".

Calculation is started with first MSB sub-group "01" as current remainder. Then, comparison is conducted to examine whether we will use "0" or "1" as new bit-facor to the current partial factor. The comparison is whether "01" is bigger or equal than $((F_t << 1) \mid 1)$. Obviously, "01" is equal to "01", then it is a true. Hence, "1" is selected as new bit-factor and the current partial factor is "1". Then, subtraction is conducted to get the partial remainder. Value of "01" will be subtracted by "1"x"1". The subtraction result will be appended with next sub-group of radicand in order to form the next partial remainder. These processes will be continued until the last sub-group of radicand has been used and produced the last remainder and last bit of quotient. Result of the calculation from Fig. 1 are "21_{10}" as quotient and "4_{10}" as remainder.

4. Digital Hardware Design

Digital hardware implementation of this algorithm is simple because we can capture several advantages there. First, we only need single comparator and two traditional calculation

modules: addition and subtraction. Second, although there are multiplication processes, but we don't need any multiplication module. We can substitute the multiplication module with traditional logical gate. It is because of the multiplication processes only needs "0" and "1" as multiplier. Hence, it can be easily substituted by a multiplexor. Then, the rest operations are only about shifting, logical *OR* operation, radicand data accessing, and delay control. The design architecture of our proposed square root algorithm is shown in Fig. 2.

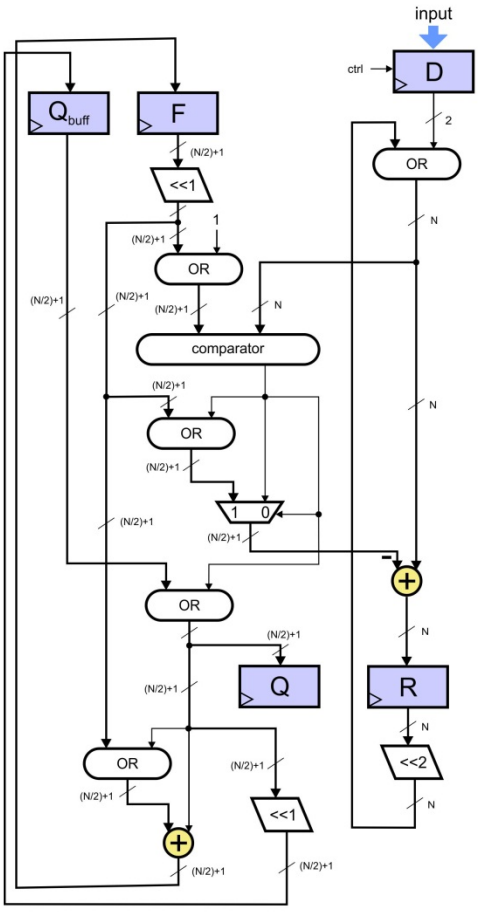

Fig 2. Architecture of proposed square root algorithm

This proposed architecture is iterative version architecture, because our goal is the resource-efficient design. It doesn't need any adjustments and directly produces the accurate value of square root and remainder. Fig. 2 shows us about the architecture components. There are 5 registers, 1 adder, 1 subtractor, 3 shifters, 5 logical *OR* modules, 1 comparator, and 1 multiplexor. Letter *N* in this illustration means the bit wordlength of radicand input. Proposed architecture is simple, low complexity, and resource-efficient. It is easy to implement in digital hardware. This *N*-bit radicand input architecture needs *(N/2)+1* clock cycles. Hence, if the radicand has 32-bit, then this design will needs *(N/2)+1 = (32/2)+1 = 17* clock cycles to complete the calculation.

5. RESULTS

A. Simulation

Functional simulation results of our proposed design are shown in Fig. 3 and Fig. 4. From the simulation results, 32-bit radicand input architecture will need 17 clock cycles and 64-bit radicand input architecture will need 33 clock cycles. From those figures, radicand is 877595_{10}. The calculations (32-bit and 64-bit designs) give 936_{10} as quotient and 1499_{10} as remainder. Those two calculations are correct.

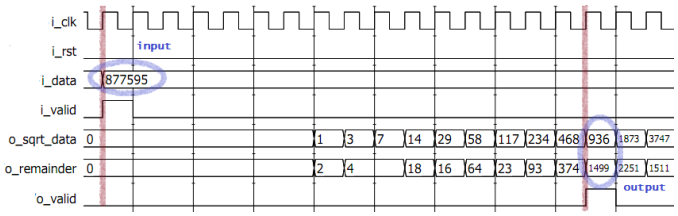

Fig 3. Simulation for 32-bit radicand input

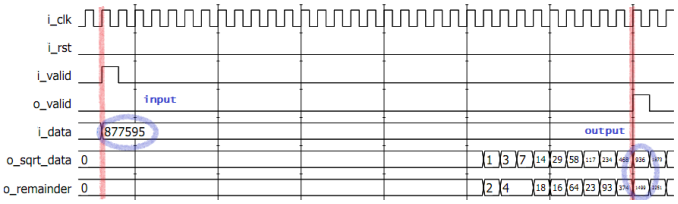

Fig 4. Simulation for 64-bit radicand input

B. Synthesis

This proposed designs have been synthesized in Altera Quartus II with FPGA target board Cyclone II EP2C35F672C6. Synthesis summaries of 32-bit and 64-bit radicand input architecture are shown in Fig. 5 and Fig. 6 respectively. Synthesis summaries show that our proposed architecture is resource-efficient.

The designs consume small resources. The 32-bit design consumes 147 logic elements, 132 combinational fuctions, and 88 dedicated logic registers, while the 64-bit design consumes 286 logic elements, 254 combinational functions, and 170 dedicated logic registers. For timing analysis, the 32-bit design can reach 109.89MHz (delay 9.1 ns) clock speed and the 64-bit design can reach 77.59MHz (delay 12.889 ns) clock speed. The critical path delay of this proposed architecture is found at the process from input buffer block to remainder buffer block.

Quartus II Version	9.0 Build 132 02/25/2009 SJ Full Version
Revision Name	qs_project_optz
Top-level Entity Name	rch_sqroot_top_optz
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	147 / 33,216 (< 1 %)
Total combinational functions	132 / 33,216 (< 1 %)
Dedicated logic registers	88 / 33,216 (< 1 %)
Total registers	88
Total pins	69 / 475 (15 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Fig 5. Synthesis summary for 32-bit radicand input

Quartus II Version	9.0 Build 132 02/25/2009 SJ Full Version
Revision Name	qs_project_optz_64b
Top-level Entity Name	rch_sqroot_top_optz_64b
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	286 / 33,216 (< 1 %)
Total combinational functions	254 / 33,216 (< 1 %)
Dedicated logic registers	170 / 33,216 (< 1 %)
Total registers	170
Total pins	133 / 475 (28 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Fig 6. Synthesis summary for 64-bit radicand input

6. CONCLUSION

A new square root algorithm is presented in this paper. The algorithm can produces accurate value of square root and remainder without any adjustments or additional correction. Moreover, this algorithm is suitable with VLSI design because it can be easily impelented in digital hardware architecture. Our proposed architecture focuses on resource-efficient, hence the design is in iterative version. This *N*-bit wordlength input architecture needs *(N/2)+1* clock cycles. Design synthesis summaries show that the 32-bit design needs 147 logic elements, 132 combinational fuctions, and 88 dedicated logic registers while the 64-bit design needs 286 logic elements, 254 combinational functions, and 170 dedicated logic registers. For timing analysis, the 32-bit design has 109.89MHz (9.1 ns) clock speed and the 64-bit design has 77.59MHz (12.889 ns) clock speed.

REFERENCES

[1] Y. Li, W. Chu, "Implementation of single precision floating point square root on FPGAs," The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 226-232, April 1997.

[2] Y. Li, W. Chu, "Parallel-array implementations of a non-restoring square root algorithm," Proceedings International Conference on Computer Design : VLSI in Computers and Processors, pp. 690-695, October 1997.

[3] Y. Li, W. Chu, "A new non-restoring square root algorithm and its VLSI implementations," Proceedings International Conference on Computer Design : VLSI in Computers and Processors, pp. 538-544, October 1996.

[4] Y. Li, W. Chu, "Cost/performance tradeoff of n-select square root implementations," Proceedings 5th Australasian Computer Architecture Conference, vol.22, pp. 9-16, February 2000.

[5] T. Sutikno, "An optimized square root algorithm for implementation in FPGA hardware," Telkomnika vol.8, pp. 1-8, April 2010.

[6] T. Sutikno, "An efficient implementation of the non restoring square root algorithm in gate level," International Journal of Computer Theory and Engineering, vol.3, pp.46-51, February 2011.

[7] T. Sutikno, A. Z. Jidin, A. Jidin, N. R. N. Idris, "Simplified VHDL coding of modified non-restoring square root calculator," International Journal of Reconfigurable and Embedded Systems, vol.1, pp.37-42, March 2012.

[8] F. M. del Campo, A. Morales-Reyes, R. Perez-Andrade, R. Cumplido, A. G. Oroczo-Lugo, and C. Feregrino, "A multi-cycle fixed point square root module for FPGAs," ICEIE Electronics Express, vol.9, pp. 971-977, June 2012.