International Journal of Electrical and Computer Engineering (IJECE)
Vol. 4, No. 4, August 2014, pp. 548~556
ISSN: 2088-8708

Strategies for FPGA Implementation of Non-Restoring Square Root Algorithm

Tole Sutikno[1], Aiman Zakwan Jidin[2], Auzani Jidin[3], Nik Rumzi Nik Idris[4]
[1]Universitas Ahmad Dahlan (UAD), Yogyakarta, Indonesia
[2,3] Universiti Teknikal Malaysia Melaka (UTeM), Melaka, Malaysia
[4]Universiti Teknologi Malaysia (UTM), Johor Bahru, Malaysia

Article Info

Article history:

Received Jun 2, 2014
Revised Jul 10, 2014
Accepted Jul 26, 2014

Keyword:

FPGA
Non-restoring algorithm
Pipelined architecture
Square root calculation

ABSTRACT

This paper presents three strategies to implement non restoring square root algorithm based on FPGA. A new basic building block is called controlled subtract-multiplex (CSM) is introduced in first strategy which use gate level abstraction. The main principle of the method is similar with conventional non-restoring algorithm, but it only uses subtract operation and append 01, while add operation and append 11 is not used. Second strategy presents the first strategy in register transfer level (RTL) abstraction. In third strategy, a modification for the implementation of conventional non-restoring algorithm is presented which also use RTL abstraction. The all above strategies is implemented in VHDL programming and adopt fully pipelined architecture. The strategies have conducted to implement successfully in FPGA hardware, and each of the strategies is offer an efficient in hardware resource. In generally, the third strategy is superior.

Corresponding Author:

Tole Sutikno
Departement of Electrical Engineering
Universitas Ahmad Dahlan
Kampus 3, Jln. Prof. Soepomo, Janturan, Umbul Harjo, Yogyakarta 55164, Indonesia
Email: tole@ee.uad.ac.id

1. INTRODUCTION

Square root calculation is one of the most useful and vital operation in computer graphics and scientific calculation applications, such as digital signal processing (DSP) algorithms, math coprocessor, data processing and control, and even multimedia applications [1-6]. It is a classical problem in computational number theory and often encountered, which is a hard task to get an exact result [7, 8].

Some square root calculation approach has have been studied, such as Rough estimation, Babylonian method, exponential identity, Taylor-series expansion algorithm, Newton-Raphson method, Sweeney Robertson Tocher redundant and non redundant method, restoring and non-restoring algorithm (digit-by-digit method) [1-9]. However, the early processors carry out the square root operation of the algorithms above by software means, which have long delays for its completion [6].

With the rapid advancement of technology which is possible to integrate large circuits on a single chip and increase in demand for faster computational execution time, hardware realize of square root became more attractive [6]. Unfortunately because of the complexity of the square root algorithms, the square root calculation is not easy to implement on field programmable array (FPGA) technology [1, 3, 5, 10].

There are some algorithms of square root which are implemented on FPGA. They are generally grouped into two distinct categories. In first category is called estimation methods, such as Rough estimation and Newton-Raphson method (and also its derivations: CORDIC, DeLugish's and Chen's), and in second category is called digit-by-digit method. The restoring algorithm has a big limitation at restoring step in the regular flow. Primarily for this reason, although initially having led the way for all the other methods, it has declined in importance and nowadays it is no longer used [11]. The non restoring algorithm does not restore the remainder, which can be implemented with fewest hardware resource. It is most suitable for FPGA implementation and allows for IEEE standard rounding to be readily implemented [1-3, 6].

There are many strategies or architectures have conducted to implement the non restoring digit-by-digit square root algorithm in FPGA hardware. Yamin and Wanming [1, 2, 9] have introduced a non restoring algorithm with fully pipelined and iterative version that requires neither multipliers nor multiplexors. They introduced the carry save adder (CSA) and carry propagate adder (CPA) as basic building blocks. Although the algorithms in [1, 2] have a speed processing, they consumes too many hardware resource, while the algorithms in [9] although it cost less resource, but it has low speed. The similar architectures above have introduced by Xiaoliang [10], Thakkar [12] and Xiumin et al [13]. In the other study, Samawi et al [6] have introduced controlled add-sub (CAS) as basic building blocks. The effort is done to reduce hardware consumed, with moderate delay. The other architecture also has proposed is fully combinational architecture [4]. However, the FPGA is very suitable for adoption of the fully pipelined architecture because of the characteristics of its structure. Hence, the very little or even needless extra cost, if the pipeline technology is implemented in FPGA [14].

This paper presents three strategies to implement non restoring square root algorithm based on FPGA which adopt fully pipelined architecture. The first strategy use gate level abstraction which introduce CSM as a basic building block. The main principle of the first method is only uses subtract operation and append 01, while add operation and append 11 is not used. Second strategy presents the first strategy in register transfer level (RTL) abstraction, and in third strategy, a modification for the implementation of conventional non-restoring algorithm is presented which also use RTL abstraction. In the three strategies will needs fewer pipeline stages compared with the proposed algorithm in [12]. Next, the performance of developed systems will be compared to Samawi et al [6].

2. DIGIT-BY-DIGIT CALCULATION METHOD

In digit-by-digit calculation method, each digit of the square root is found in a sequence where only one digit of the square root is generated at each iteration [2, 6, 13]. It has several advantages, such as: every digit of the root found is known to be correct and it will not has to be changed later; if the square root has to be expanded, it will be terminated after the last digit is found; and the algorithm works for any number base (of course the process depends on number base).

In general, this method can be divided in two classes, i.e. restoring and non restoring digit-by-digit algorithm [6]. In restoring algorithm, the procedure is composed by taking the square root obtained so far, appending 01 to it and subtracting it, properly shifted, from the current remainder. The 0 in 01 corresponds to multiplying by 2; the 1 is a new guess bit. The new root bit developed is 1, if the resulting remainder is positive, else it is 0, which the remainder must be restored by adding the quantity just subtracted. It is different from the non restoring algorithm where the subtraction is not restored if the result is negative. Instead, it appends 11 to the root developed so far and on the next iteration it performs an addition. If the addition causes an overflow, then on the next iteration it has to go back to the subtraction mode [15]. Figure 1 (a) and (b) gives an example on how take the binary square root of 01011101 (equivalent with 93 decimal) for restoring and non restoring algorithm respectively.

The conventional method is shown in Figure 1(a) whereas the modification is shown in Figure 1(b). In this modification, only subtract operation with append 01 is used; add operation and append 11 is not used. This paper adopts this modification to implement unsigned 32 and 64-bit binary square root based on FPGA.

3. THE PROPOSED STRATEGIES FOR FPGA IMPLEMENTATION OF NON-RESTORING SQUARE ROOT ALGORITHM

2.1. First Strategy

The first strategy offers a simple alternative solution. Samavi, et al [6] has improved classical non-restoring digit-by-digit square root circuit by eliminate redundant blocks which still based on constant digit of 01 or 11 and add-subtract as the main building block. The first strategy offers a simple strategy while only uses subtract operation and appends 01. The strategy is implemented by VHDL programming in gate level abstraction.

A hardware implementation of the non-restoring digit-by-digit algorithm for unsigned 6-bit square root by an array structure is shown in Figure 2. The radicand is P (P5,P4,P3,P2,P1,P0), U (U2,U1,U0) as quotient and R (R4,R3,R2,R1,R0) as remainder.

```
        1  0  0  1 .
      ___________________
     /01 01 11 01 . 00
      -1
      ----
      00 01  <----  positive: first bit is a 1
      -1 01
      -----
      11 00  <----  negative: 2nd bit is a 0
      +1 01  <----  restore the wrong guess
      --------
      00 01 11
        -10 01
      --------
      11 11 10   <----  negative: 3rd bit is a zero
        +10 01   <----  restore the wrong guess
      ----------
         01 11 01
         -1 00 01
         --------
          0 11 00   <----  positive: 4th bit is a 1
```

(a)

```
    1  0  0  1 .
  ___________________
 / 01 01 11 01 . 00
   -1
   ----
   00 01  <---- positive: first bit is a 1
   -1 01  <---- Developed root is "1"; appended 01; subtract
   -------
   11 00 11  <---- negative: 2nd bit is a 0
     +10 11  <---- Developed root is "10"; append 11 and add
   ----------
   11 11 10 01  <---- negative: 3rd bit is a 0
       1 00 11  <---- Developed root is "100"; append 11 and add
   -----------
 1 00 00 11 00  <---- Overflow: 4th bit is a 1
```

(b)

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

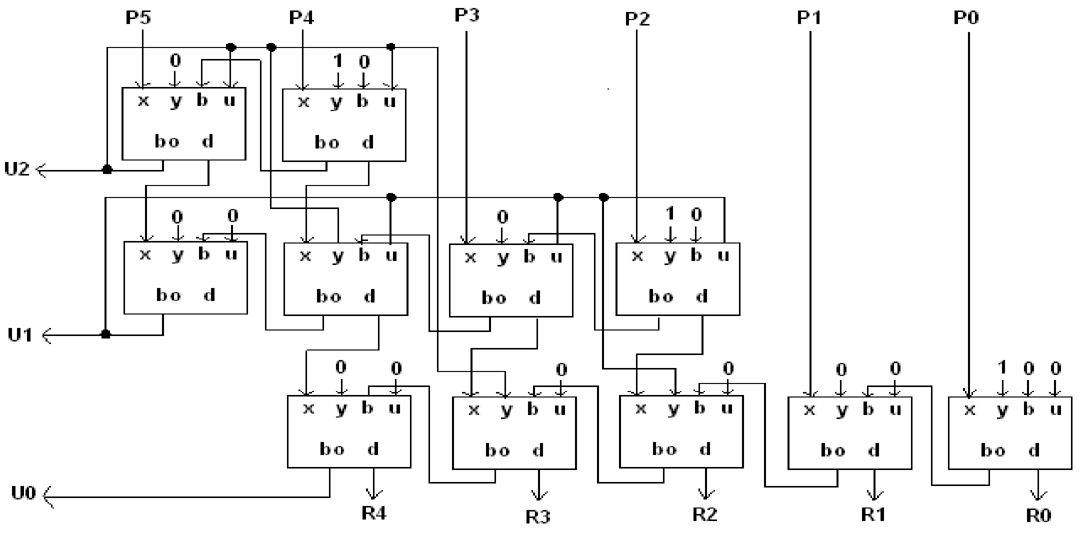


Figure 2. A simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned 6-bit square root

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 3 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.

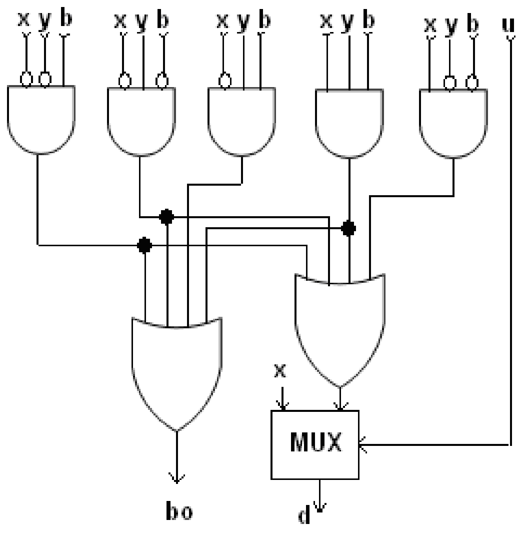


Figure 3. Internal structure of a CSM block

In the first strategy, 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, to optimize the implementation of unsigned 6-bit square root can be optimized become as shown in Figure 4. 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.

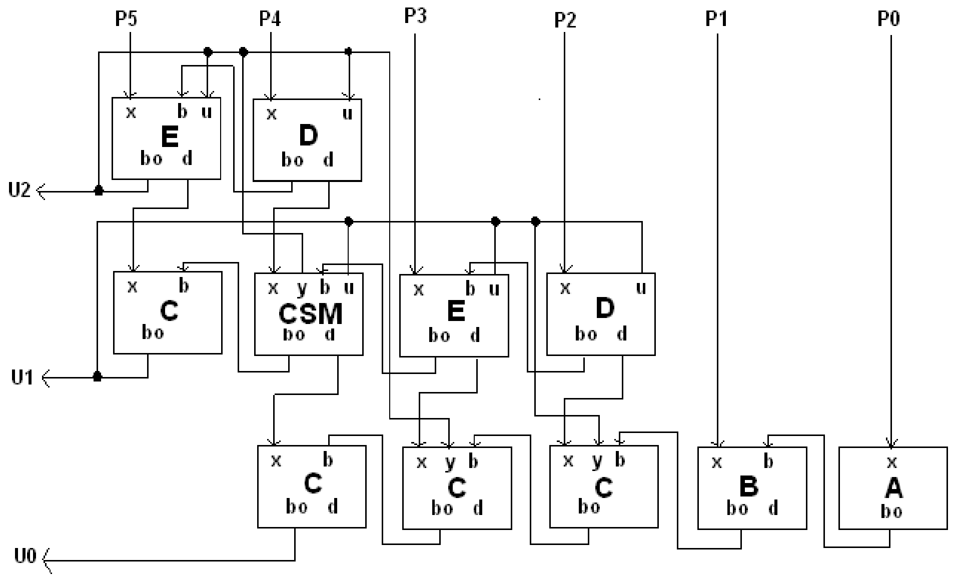


Figure 4. Optimized simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned 6-bit square root

The generalization of the non-restoring digit-by-digit algorithm for unsigned n-bit square root is shown in Figure 5.

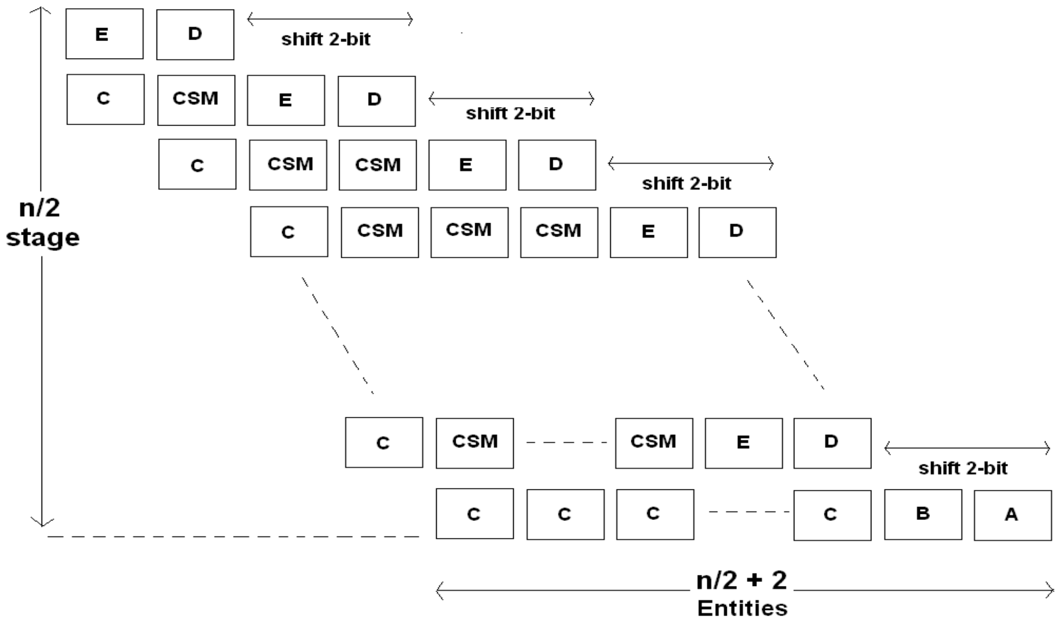


Figure 5. Optimized simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned n-bit square root

2.2. Second Strategy

The second strategy also offers a simple alternative solution as the first strategy that it only uses subtracts operation and appends 01. But, the second strategy presents the first strategy in register transfer level (RTL) abstraction. The principle of the second proposed algorithm can be described as follow:

```
Step 0.   Start
Step 1.   Initialization radicand (the n-bit number will be squared root),
          quotient (the result of squared root), and remainder. To calculate
          square root of a 2n bit number, it needs n stage pipelines to
          implement the proposed algorithm.
Step 2.   Beginning at the binary point, divide the radicand into groups of two
          digits in both direction.
Step 3.   Beginning on the left (most significant bit), select the first group
          of one or two digit (If n is odd then the first groups is one digit,
          and vice versa)
Step 4.   Choose 1 squared, and then subtract.
          Fist developed root is “1” if the result of subtract is positive, and
          vice versa is “0”
Step 5.   Shift two bits, subtract guess squared with append 01.
          Nth-bit squared is “1” if the result of subtract is positive, and
          Because of subtract operation is done
          else
          Nth-bit squared is “0”, and not subtract
Step 6.   Go to step 5 until end group of two digits
Step 7.   End
```

2.3. Third Strategy

The third strategy is the modification of Samawi et al [6]. The strategy is also implemented in register transfer level (RTL) abstraction as the second strategy. The basic modification of the third strategy can be described as follow:

$r_o = -1$ (n/2 + 2 bit)

$q_o = 0$ (n/2 + 1 bit)

the radicand $D = D_{n-1}D_{n-2}...D_1D_0$

```
For $i = 0$ to $\left(\frac{n}{2} - 1\right)$ do:
If $r_i \geq 0$ then
            $r_{i+1} = \left[4r_i + D_{(n-2i)-1}D_{(n-2i)-2}\right] - \left[4q + 1\right]$
      else
            $r_{i+1} = \left[4r_i + D_{(n-2i)-1}D_{(n-2i)-2}\right] + \left[4q + 3\right]$

If $r_{i+1} \geq 0$ then
            $q_{i+2} = 2q_i + 1$
      else
            $q_{i+2} = 2q_i$
```

The final result of the square root is equal to q_n(n/2-1) downto 0, coded in n/2 bits.

4. RESULTS AND ANALYSIS

In the previous sections, the three hardware implementation strategies of the non-restoring digit-by-digit algorithm for square root were explained. In this section, simulation results of 32-bit and 64-bit square root based on Altera APEX 20KE FPGA using the above method are presented, as shown in Figure 4. In this simulation, P is radicand and U is quotient. The results showed that the implementation has succeeded and worked properly.

Based on compilation report, to implement 32-bit and 64-bit square root using three above strategies using Altera FPGA APEX 20KE are needed 256 and 1023 logic element (LE) respectively, for the first strategy. The comparison of results obtained from different implementation method is shown in Table 1. This comparison of LE or logic cell (LC) usage is listed based on references [6] and [16]. It has shown a fantastic value for reducing of hardware resource consumed. This is due adoption fully pipelined architecture and also simplification of CSM as shown in Figure 4.

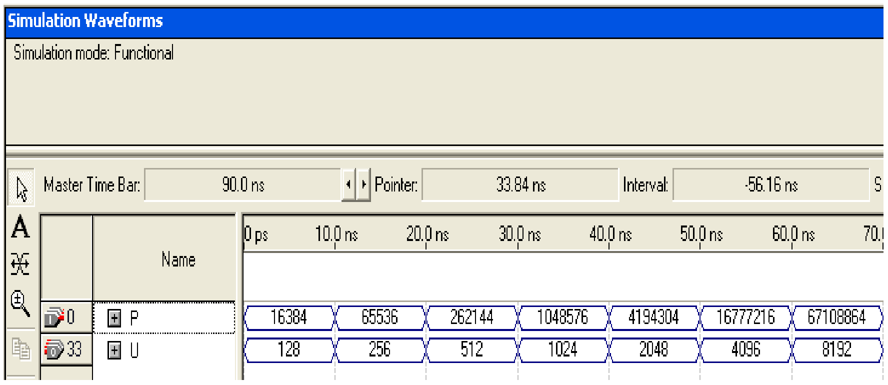
Table 1. The comparison of logic element usage

No	Method	LEs Usage		Abstraction level
		32-bit square root	64-bit square root	
1	Classical-NR	1008	4092	N/A
2	Reduced-Area-NR	632	2464	N/A
3	Modular-NR	624	2468	N/A
4	Simple-X-Module	648	2488	N/A
5	The first proposed strategy	**256**	**1023**	**gate**
6	The second proposed strategy	**340**	**1365**	**RTL**
7	The third proposed strategy	**264**	**1061**	**RTL**

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

The second and third strategies consume LE bigger than the first strategy. Nevertheless, the second and third strategies are more flexible because the strategies use the RTL abstraction level. If it is needed to set the number bits of the radicand which will be determined its square root, it can be done easily without hard modification of VHDL source while in the first strategy, it must rebuild the VHDL source. By considering the abstraction and the amount of LE consume, the study recommends the use of the third strategy.

Simulation results of 32-bit and 64-bit square root based on Altera APEX 20KE FPGA using the proposed method is presented, as shown in Figure 6. In this simulation, P is radicand and U is quotient. The results showed that the implementation is successful and worked properly.



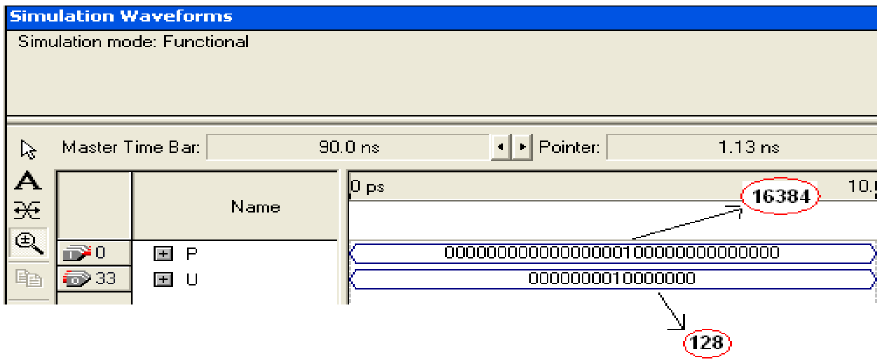
Simulation Waveforms

Simulation mode: Functional

Master Time Bar: 90.0 ns Pointer: 33.84 ns Interval: -56.16 ns

Name	0 ps	10.0 ns	20.0 ns	30.0 ns	40.0 ns	50.0 ns	60.0 ns
0 P	16384	65536	262144	1048576	4194304	16777216	67108864
33 U	128	256	512	1024	2048	4096	8192

(a)



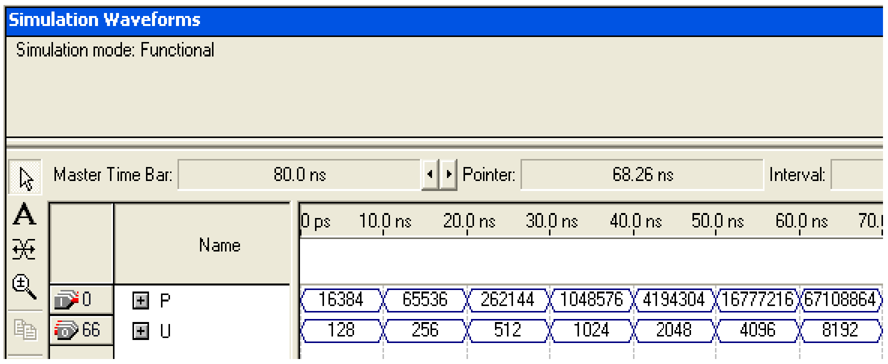
Simulation Waveforms

Simulation mode: Functional

Master Time Bar: 90.0 ns Pointer: 1.13 ns

Name	0 ps
0 P	00000000000000000100000000000000
33 U	0000000010000000

(b)



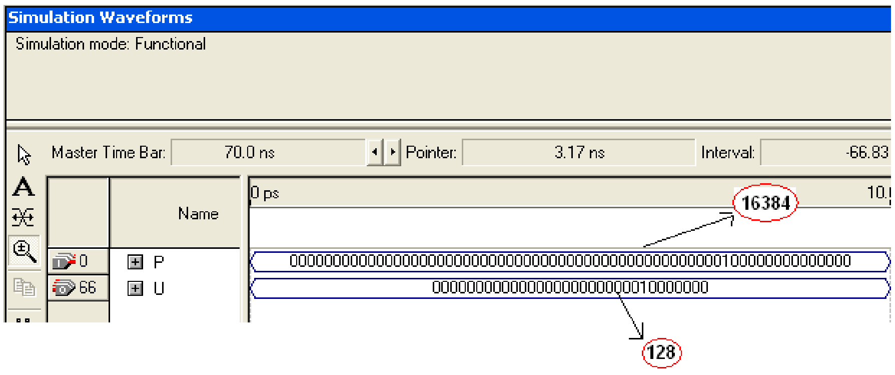
Simulation Waveforms

Simulation mode: Functional

Master Time Bar: 80.0 ns Pointer: 68.26 ns Interval:

Name	0 ps	10.0 ns	20.0 ns	30.0 ns	40.0 ns	50.0 ns	60.0 ns
0 P	16384	65536	262144	1048576	4194304	16777216	67108864
66 U	128	256	512	1024	2048	4096	8192

(c)



Simulation Waveforms

Simulation mode: Functional

Master Time Bar: 70.0 ns Pointer: 3.17 ns Interval: -66.83

Name	0 ps
0 P	0000000000000000000000000000000000000000000000000100000000000000
66 U	00000000000000000000000010000000

(d)

Figure 6. 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

Based on Figure 5, actually the first strategy can be expanded for larger number to solve complicated square root problem in FPGA implementation. Unfortunately, the proposed method is only appropriate for gate level abstraction and is not powerful for RTL or behaviour level abstraction. The second and third methods are better choice for solving square root problem for larger number. We don't need re-written the VHDL source codes for different numbers. The methods will have many advantages over the method proposed by Samawi *et al* [6], and the third method is best choice for hardware resource saving.

5. CONCLUSION

This paper has presented three strategies of the FPGA implementation of non restoring square root algorithm. In first strategy, a CSM as basic building block use gate level abstraction has introduced. The principle of the strategy is similar with conventional non-restoring algorithm, but it only uses subtract operation and append 01, while add operation and append 11 is not used. Second strategy has presented the first strategy form in register transfer level (RTL) abstraction, and in third strategy, a modification for the implementation of conventional non-restoring algorithm has presented which also use RTL abstraction. The all above strategies have implemented in VHDL programming and adopt fully pipelined architecture. The strategies have conducted to implement successfully in FPGA hardware, and each of the strategies is offer an efficient in hardware resource. In generally, the third strategy is superior because it do not need hard modification to set the number bits of the radicand.

References

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

[2] Yamin, L. 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.

[3] Piromsopa, K., C. Aporntewan, and P. Chongstitvatana, An FPGA Implementation of a fixed-point square root operation, in Int. Symp. on Communications and Information Technology (ISCIT 2001)2001: ChiangMai, Thailand.

[4] Llamocca-Obregon, D.R., A Core Design to Obtain Square Root Based on a Non-Restoring Algorithm, in IBERCHIPS Workhsop2005: Salvador Bahia, Brazil. p. 1-5.

[5] XiaojunWang, Variable Precision Floating-Point Divide and Square Root for Efficient FPGA Implementation of Image and Signal Processing Algorithms, in Electrical and Computer Engineering2007, Northeastern University: Boston, Massachusetts. p. 119.

[6] Samavi, S., A. Sadrabadi, and A. Fanian, *Modular array structure for non-restoring square root circuit.* Journal of Systems Architecture, 2008. **54**(10): p. 957-966.

[7] Dong-Guk, H., C. Dooho, and K. Howon, *Improved Computation of Square Roots in Specific Finite Fields.* Computers, IEEE Transactions on, 2009. **58**(2): p. 188-196.

[8] Lachowicz, S. 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.

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

[10] Xiaoliang, J., *Implementation of Square Root Arithmetic Based on FPGA.* Modern Electronics Technique, 2007. **30**(14).

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

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

[13] Xiumin, W., 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.

[14] Renxi, G., et al. Hardware Implementation of a High Speed Floating Point Multiplier Based on FPGA. in 4th International Conference on Computer Science & Education. 2009. Nanning, Guangxi, P.R.China: 1902-1906.

[15] Dattalo, S. *Square Root Theory*. Technical Stuff 2000 March 10, 2010 March 17, 2010]; Available from: http://www.dattalo.com/technical/theory/sqrt.html.
[16] *Comparing Altera APEX 20KE & Xilinx Virtex-E Logic Densities*. [cited 2010 March 30, 2010]; Available from: http://www.altera.com/products/devices/apex/features/apx-compdensity.html.