A SIMPLE STRATEGY TO SOLVE COMPLICATED SQUARE ROOT PROBLEM IN DTC FOR FPGA IMPLEMENTATION

TOLE SUTIKNO
AIMAN ZAKWAN JIDIN
NIK RUMZI NIK IDRIS
AUZANI JIDIN

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

A Simple Strategy to Solve Complicated Square Root Problem in DTC for FPGA Implementation

Tole Sutikno[1], Aiman Zakwan Jidin[2], Nik Rumzi Nik Idris[3], Auzani Jidin[4]
[1]Electrical Engineering Dept., Ahmad Dahlan University, Yogyakarta, Indonesia.
[2]Ecole Supérieure d'Ingénieur en Electronique et Electrotechnique Paris, Paris, France.
[3]Energy Conversion Dept., Universiti Teknologi Malaysia, Johor Bahru, Malaysia.
[4]Power Elect. & Drives Dept., Universiti Teknikal Malaysia Melaka, Melaka, Malaysia.
Email: thsutikno@ieee.org, jidina@esiee.fr, nikrumzi@ieee.org, auzani@ieee.org

Abstract—**This paper presents a digit-by-digit calculation method to calculate the complicated square root problem faced in implementing the direct torque control (DTC) of induction motor drives using Field Programmable Gate Arrays (FPGA). The main principle of the proposed method is based on a two-bit shifter and a subtractor-multiplexor operation that gives simpler implementation and faster calculation. The proposed strategy was successfully implemented on an FPGA device using unsigned 32 bit and 64-bit binary square root. The strategy can be easily expanded to larger number.**

Keywords—**Digit-by-Digit Calculation; FPGA; Square Root**

I. INTRODUCTION

It is well-known that the direct torque control method (DTC) for AC motors has simple structure and excellent performance such as fast torque response, no requirements for PWM pulse generation and coordinate transformation, no position encoder and does not need current regulators [1-7]. The DTC algorithm is usually implemented based on serial calculations using Microcontroller or Digital Signal Processing (DSP) [8-11]. These are truly software-based platform which may be not fast enough for some applications. For instance, in DTC drive implementation, the stator flux and electric torque have to be estimated in real-time. For hysteresis-based DTC with digital implementation, it is necessary to have several samplings as the torque error moves from lower to upper (or vice versa) bands in order to avoid large torque ripple. In space vector modulation -based (SVM-based) implementation, other than estimating the torque and flux, the processor has to implement the SVM algorithm as well. These all translate to the need of faster and hence more expensive processors. As an alternative solution to provide a faster calculation for real time flux and torque estimations is to use FPGA based system [12-14]. However, the main drawback in using FPGA in performing the estimation is the complexity. One problem which has been identified in DTC induction motor drive system based on FPGA implementatio is the complicated square root calculations [15-17].

There are many algorithms which have been proposed to solve the square root calculation problem, such as using Rough estimation [18], Babylonian method [19], exponential identity [20], Taylor-Series Expansion Algorithm [21], Newton-Raphson method [22-24], and sequential algorithm (digit-by-digit calculation method) [25-29]. However, these proposed methods are not particularly found in electric drive system applications where square root calculation is normally required. Specifically, in this paper, calculation of square root for the direct torque control drive will be addressed.

This paper proposes digit-by-digit calculation method as a simple strategy to solve complicated square root calculation implemented using FPGA. The proposed implementation strategy is slightly different from the strategies proposed in [25-29], as will be discussed later. An optimization is also done by eliminating circuitry that is not needed. The method is proposed as part of the development of the controller used in the DTC drive system which is fully based on FPGA.

II. 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 [27-29]. 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 [29]. 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 [30]. 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.

In this paper, a simple modification to the conventional non-restoring digit-by-digit algorithm is made in order to give a simpler implementation and faster calculation. 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.

III. PROPOSED SQUARE ROOT ALGORITHM

Samavi, et al. [29] has improved classical non-restoring digit-by-digit square root circuit by eliminating the redundant blocks. Their circuit is referred to as the reduced area non restoring circuit. However, it is still based on constant digit of 01 or 11 and add-subtract as the main building block (Figure 1 b). This paper offers a simple alternative solution that only uses subtract operation and appends 01. Consequently, the subtract-multiplex is used as the main building block (refer to Figure 2). The principle of the proposed algorithm can be described as shown in Figure 3.

```
      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

```
  1  0  0  1 .
 ________________      choose 1 squared, subtract, keep if positive
/01 01 11 01 . 00      shift two bits at a time
 -1                    subtract guess squared
 ___
 00 01
 -1 01   <----     sqrt so far with 01 appended : 2nd bit is a 0
 _______           no subtract because result would be negative
    01 11
  -10 01 <---- sqrt so far with 01 appended: 3rd bit is a 0
 ________      no subtract because result would be negative
    01 11 01
    -1 00 01
    ________
       11 00   <---- positive : 4th bit is a 1
```

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

```
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
```

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

A simple hardware implementation of the 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 remainder. It can be shown that the implementation needs 3 stage pipelines. The main 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.

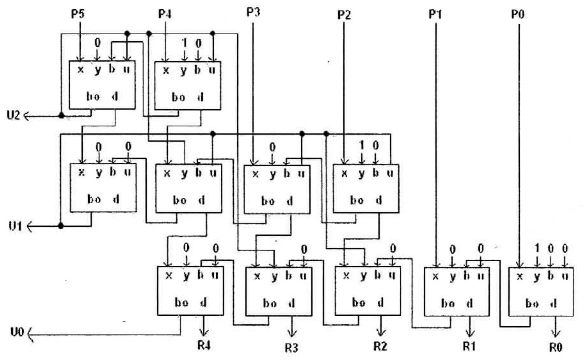


Figure 4. A simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned 6-bit square root

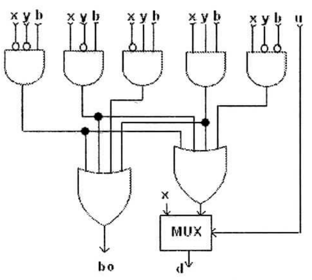


Figure 5. Internal structure of a CSM block

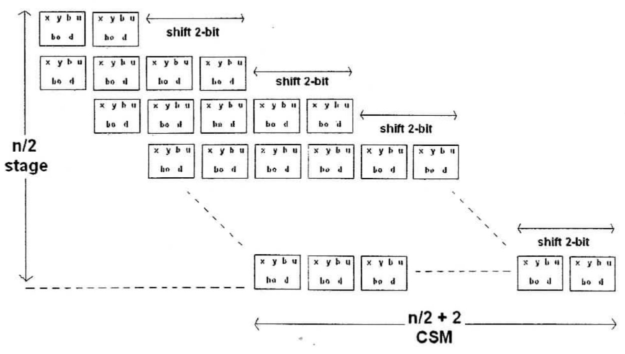


Figure 6. A simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned n-bit square root

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, specialized entities can be created as building block components. It will eliminate circuitry that is not needed. As an example, the implementation in Figure 6 for unsigned 6-bit square root can be optimized to become as shown in Figure 7 (in this case, the remainder is ignored, because in the DTC drive, it is not required). 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.

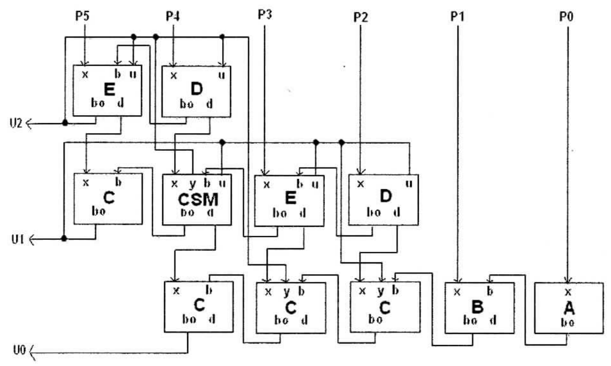


Figure 7. Optimized simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned 6-bit square root

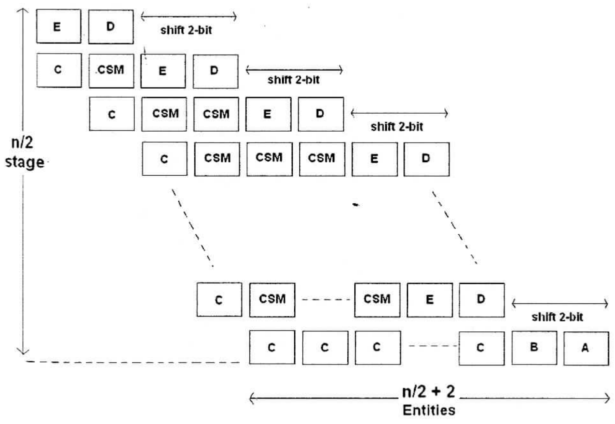


Figure 8. Optimized simple hardware implementation of the non-restoring digit-by-digit algorithm for unsigned n-bit square root

IV. The Calculation of Stator Flux Linkage in DTC of AC Machine Drive

The most important and difficult task in DTC of AC machine drive in the d-q stationary reference frame is to accurately calculate the stator flux linkage [4-5, 7]. This is because the performance of the DTC drive is highly dependent on the control of the amplitude and phase of the stator flux linkage, φ_s and thus require an accurate estimation of these quantities. If the calculation error is such that the estimated angle δ exceeds the limit value, the torque may fall with increasing δ and thus can lead to instability.

The d-q axes components of the stator flux linkage, $\varphi_D(k)$ and $\varphi_Q(k)$, at the *k-th* sampling instant can be calculated as follows:

$$\varphi_D(k) = \varphi_D\big|_{k-1} + (v_D\big|_{k-1} - R\bar{i}_D)T_s \qquad (1)$$

$$\varphi_Q(k) = \varphi_Q\big|_{k-1} + (v_Q\big|_{k-1} - R\bar{i}_Q)T_s \qquad (2)$$

$$\varphi_s(k) = \sqrt{\varphi_D^2(k) + \varphi_Q^2(k)} \qquad (3)$$

In this paper, equation (3) is addressed as complicated square root problem in DTC for FPGA implementation [15-17]. In Figure 9 which shows the torque and flux estimator design based on FPGA, the need for the 64-bit square root calculation is specifically shown. To solve this problem, optimized simple hardware implementation method of the non-restoring digit-by-digit algorithm is proposed.

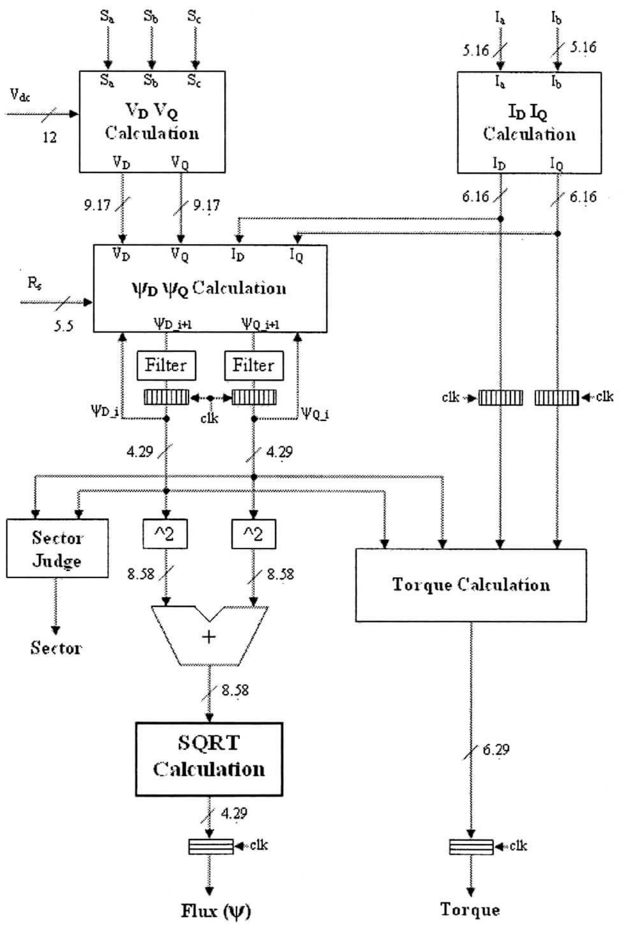
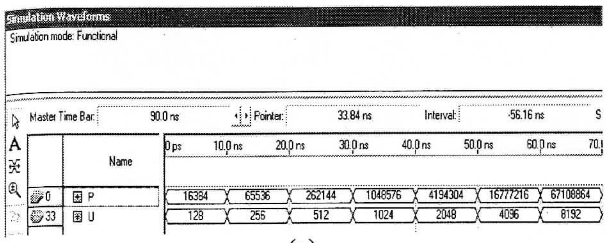


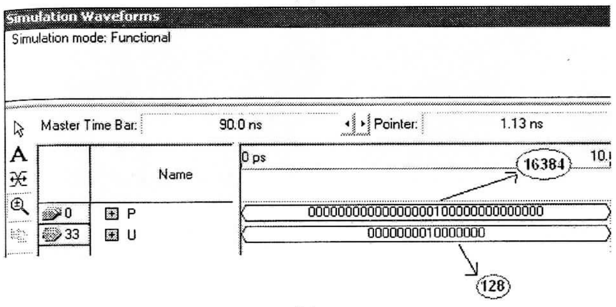
Figure 9. Block diagram of torque and flux estimator design based on FPGA in fixed point term

V. Results And Analysis

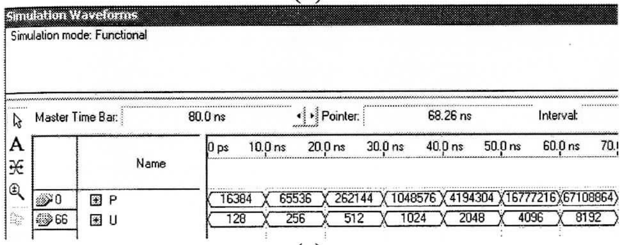
In the previous sections, optimized simple hardware implementation method of the non-restoring digit-by-digit algorithm for square root and the difficult task in DTC to calculate square root were explained. In this section, simulation results of 32-bit and 64-bit square root based on Altera APEX 20KE FPGA using the proposed method is presented, as shown in Figure 10. In this simulation, P is radicand and U is quotient. The results showed that the implementation is successful and worked properly.



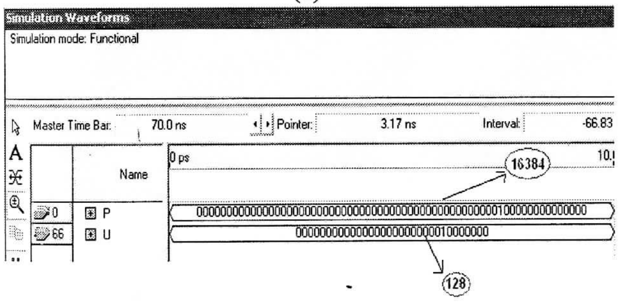
(a)



(b)



(c)



(d)

Figure 10. 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 the compilation report, in order to implement 32-bit and 64-bit square root using the optimized simple hardware implementation method of the non-restoring digit-by-digit algorithm, 256 and 1023 logic element (LE) are needed respectively. The comparison of results obtained from different implementation methods is shown in Table 1. The comparison of LE or logic cell (LC) usage is listed based on references [29] and [31]. The number of employed LE indicates the size of hardware resource required to implement the circuit. The table clearly showed that the proposed method is most efficient method in terms of the usage of hardware resource. This is particularly true as it only uses subtract operation and append 01 with no add operation.

Based on Figure 8, the strategy can be easy 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.

VI. Conclusion

This paper presents a modification of conventional non restoring digit-by-digit calculation method for implementation in FPGA hardware, mainly to solve complicated square root in DTC drive algorithm. The main principle of the proposed method is the two-bit shifter and subtractor-multiplexor operations, and only uses subtract operation and append 01, without add operation and append 11. The proposed strategy has been successfully implemented on an FPGA device based on unsigned 32-bit and 64-bit binary square root. The results have shown that the proposed method is the most efficient in terms of hardware resource usability compared to other methods. The strategy can also be easily expanded to solve larger numbers for complicated square root problem in FPGA-based implementation system.

Acknowledgment

The authors wish to acknowledge the Ministry of Higher Education (MOHE) of Malaysia for sponsoring the research under the FRGS scheme.

TABLE I.
The Comparison of Logic Element Usage

No	Method	LE 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 [31], for Altera APEX 20KE & Xilinx Virtex-E, 1 LC = 1 LE, and 1 CLB = 4 LE

References

[1] I. Takahashi and T. Noguchi, "A New Quick-Response and High-Efficiency Control Strategy of an Induction Motor," *IEEE Transactions on Industry Applications,* vol. Vol.IA-22, No.5, pp. 820-827, Sept/Oct 1986.

[2] M. Depenbrock, "Direct self control (DSC) of inverter-fed induction machine," *IEEE Trans. on Power Electronics,* vol. 3 (4), pp. 420–429, 1988.

[3] T. G. Habetler, *et al.*, "Direct torque control of induction machines using space vector modulation," *Industry Applications, IEEE Transactions on,* vol. 28, pp. 1045-1053, 1992.

[4] L. Zhong, *et al.*, "Analysis of direct torque control in permanent magnet synchronous motor drives," *Ieee Transactions on Power Electronics,* vol. 12, pp. 528-536, May 1997.

[5] C. French and P. Acarnley, "Direct torque control of permanent magnet drives," *Industry Applications, IEEE Transactions on,* vol. 32, pp. 1080-1088, 1996.

[6] L. Yong, *et al.*, "Direct torque control of brushless DC drives with reduced torque ripple," *Industry Applications, IEEE Transactions on,* vol. 41, pp. 599-608, 2005.

[7] L. Yong, *et al.*, "Commutation-Torque-Ripple Minimization in Direct-Torque-Controlled PM Brushless DC Drives," *Industry Applications, IEEE Transactions on,* vol. 43, pp. 1012-1021, 2007.

[8] B. K. Bose and P. M. Szczesny, "A microcomputer-based control and simulation of an advanced IPM synchronous machine drive system for electric vehicle propulsion," *Industrial Electronics, IEEE Transactions on,* vol. 35, pp. 547-559, 1988.

[9] L. Lianbing, *et al.*, "A high-performance direct torque control based on DSP in permanent magnet synchronous motor drive," in *Intelligent Control and Automation, 2002. Proceedings of the 4th World Congress on*, 2002, pp. 1622-1625 vol.2.

[10] L. Weijie, "Implementation of Direct Torque Control for Permanent Magnet Synchronous Motor with Space Vector Modulation Based on DSP," in *Signal Processing, 2006 8th International Conference on*, 2006.

[11] S. M. A. Cruz, *et al.*, "DSP implementation of the multiple reference frames theory for the diagnosis of stator faults in a DTC induction motor drive," *Energy Conversion, IEEE Transactions on,* vol. 20, pp. 329-335, 2005.

[12] E. Monmasson and M. N. Cirstea, "FPGA Design Methodology for Industrial Control Systems: A Review," *Industrial Electronics, IEEE Transactions on,* vol. 54, pp. 1824-1842, 2007.

[13] C. T. Kowalski, *et al.*, "FPGA Implementation of DTC Control Method for the Induction Motor Drive," in *EUROCON, 2007. The International Conference on Computer as a Tool*, 2007, pp. 1916-1921.

[14] V. D. Colli, *et al.*, "Design of a System-on-Chip PMSM Drive Sensorless Control," in *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, 2007, pp. 2386-2391.

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

[16] 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.

[17] 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.

[18] M. D. Ercegovac, "On Digit-by-Digit Methods for Computing Certain Functions," in *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*, 2007, pp. 338-342.

[19] O. Kosheleva, "Babylonian method of computing the square root: Justifications based on fuzzy techniques and on computational complexity," in *Fuzzy Information Processing Society, 2009. NAFIPS 2009. Annual Meeting of the North American*, 2009, pp. 1-6.

[20] W. B. Ligon, III, *et al.*, "Implementation and analysis of numerical components for reconfigurable computing," in *Aerospace Conference, 1999. Proceedings. 1999 IEEE*, 1999, pp. 325-335 vol.2.

[21] K. Taek-Jun, *et al.*, "Floating-point division and square root implementation using a Taylor-series expansion algorithm," in *Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference on*, 2008, pp. 702-705.

[22] H. Kabuo, *et al.*, "Accurate rounding scheme for the Newton-Raphson method using redundant binary representation," *Computers, IEEE Transactions on,* vol. 43, pp. 43-51, 1994.

[23] M. Allie and R. Lyons, "A root of less evil [digital signal processing]," *Signal Processing Magazine, IEEE,* vol. 22, pp. 93-96, 2005.

[24] W. Liang-Kai and M. J. Schulte, "Decimal floating-point square root using Newton-Raphson iteration," in *Application-Specific Systems, Architecture Processors, 2005. ASAP 2005. 16th IEEE International Conference on*, 2005, pp. 309-315.

[25] V. Tchoumatchenko, *et al.*, "A FPGA based square-root coprocessor," in *EUROMICRO 96. 'Beyond 2000: Hardware and Software Design Strategies'., Proceedings of the 22nd EUROMICRO Conference*, 1996, pp. 520-525.

[26] N. Takagi and K. Takagi, "A VLSI Algorithm for Integer Square-Rooting," in *Intelligent Signal Processing and Communications, 2006. ISPACS '06. International Symposium on*, 2006, pp. 626-629.

[27] 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.

[28] 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.

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

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

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