Hardware Implementation of Single Iterated Multiplicative Inverse Square Root

Jun Luo[1, 2], Qijun Huang[2], Hongwei Luo[1], Yue Zhi[1], Xiaoqiang Wang[1]
[1]*China Electronic Product Reliability and Environmental Testing Research Institute, No. 110, Dongguanzhuang Road, Guangzhou, Guangdong, China*
[2]*Department of Physics and Technology, Wuhan University, No. 16, Luojia Shan, Wuhan, Hubei, China*
huangqj@whu.edu.cn

***Abstract*—Inverse square root has played an important role in Cholesky decomposition, which devoted to hardware efficient compressed sensing. However, the performance is usually limited by the trade-off between throughput and precision. This paper presents hardware implementation of fixed-point single iterated multiplicative inverse square root. Multiple piecewise linear approximation in softly nonlinear range is used to compute the initial value. Single iterated Newton-Raphson method is employed to obtain high precision. Multiple constants multiplication technique is proposed to achieve high throughput. The combination of these techniques yields high performance in terms of throughput and precision. It obtains more than 70 % of throughput improvement and almost 100 × higher precision over the inverse square root Intellectual Property (IP) from Altera. In addition, Cholesky decomposition has been presented to validate the proposed architecture, which shows that 42 % of throughput improvement is achieved compared with the IP.**

***Index Terms*—Digital circuits; fixed-point arithmetic; piecewise linear approximation; hardware.**

I. INTRODUCTION

Arithmetic element functions (reciprocal, square root and inverse square root) are playing very important roles in digital signal processing, multimedia and scientific computing. So far, most of the researches are focused on reciprocal [1] and square root [2]–[4]. Other reports are concentrated on inverse square root [5]–[9], which plays a significant role not only in vector normalization, least squares lattice filters, Cholesky decomposition and Givens rotation, but also in 3D graphics application and compressed imaging [10]. Several algorithms have been developed to compute inverse square root. Traditionally, table method has been used to approximate inverse square root. Direct look up table, interpolation table [11] and bipartite table [12] have been investigated. The drawback of table approximation is that the memory size is grown exponentially with the increasing precision. To overcome this issue, convergence algorithms are developed after the initial seed (table techniques). Subtractive and multiplicative techniques are the main kinds of convergence algorithms. Coordinate rotation digital computer (CORDIC) [1] and digit recurrence algorithm [13] are subtractive techniques. They have low complexity for their simple operations (additions and subtracts), but they have long latency and linear convergence. Multiplicative technique provides quadratic convergence. Polynomial approximation [14], series expansion [15], Goldschmidt algorithm [16] and Newton-Raphson (NR) method [3], [5] are multiplicative techniques. Goldschmidt algorithm has short latency whereas it requires large amounts of memory and area. First-order NR method is an attractive approach for its quadratic convergence and moderate complexity. It can be derived from the Taylor series expansion. The bit accuracy in multiplicative technique doubles per iteration. However, in order to reduce the total iterations, it is important to get a high accurate initial seed.

Some algorithms have been developed based on multiplicative technique previously. An efficient initial approximation was proposed in [3], although it concentrated on division and square root, it can lead to a solution of inverse square root as well. In [6], a simple hardware architecture was presented to approximate the initial of reciprocal and square root reciprocal. It used linear approximation with specific coefficients and a lookup table. However, the precision was relatively low. A high speed single precision floating point inverse square root was proposed in [7], using special squaring unit and truncated multiplier. However, the read only memory (ROM) used in [7] requires much memory bits for a high accurate initial seed. In order to reduce complexity, linear approximation with no multipliers was presented in [5] to solve inverse square root. Whereas, it's accuracy was depending on the number of items by the function expansion.

In this paper, hardware of single iterated multiplicative inverse square root is implemented. To reduce complexity, the softly nonlinear function ($y = 1/\sqrt{c}, 1 \le c < 2$) is used to avoid the highly nonlinear function ($y = 1/\sqrt{x}, 0 < x \le 1$). Different from one-piecewise linear approximation used in [6], a three-piecewise linear approximation is presented to compute the initial seed in this paper. The multiple piecewise linear approximation leads to a high precision initial seed. In addition, single iterated NR method is employed to obtain a high precise output. Multiple constants multiplication (MCM) technique is proposed to achieve a multiplier-less

Manuscript received 8 September, 2016; accepted 5 March, 2017.
This research was funded by grants (No. 61204096 and No. J1210061) from the National Natural Science Foundation of China, and by a grant (2042014kf0238) from the Fundamental Research Funds for the Central Universities.

and memory-free approach, which leads to a high throughput for the square root reciprocal. The combination of these techniques leads to good performance trade-off for most applications. Experiment results show competitive throughput and precision improvement over Intellectual Property (IP) from Altera.

II. PROPOSED INVERSE SQUARE ROOT

A. Precision Analysis of Different Approximating Methods

In the inverse square root approximation, errors (E_{total}) can be caused by the approximated methods and the truncation processes in hardware implementation. Shown in (1), E_{method} and $E_{truncat}$ denote the error caused by approximation algorithm and truncation, respectively. The errors derived from width representation of a data and shifts in an algorithm are included in the truncation in terms of bits. These errors are related to specific hardware framework. In order to evaluate the performance of different approximating algorithms, error of E_{method} is analysed in this section

$$E_{total} = E_{method} + E_{truncat}. \tag{1}$$

Absolute error comparison of different approximating methods has been shown in Fig. 1, in which M1 is the method of single NR combined with eight-piecewise linear approximation in highly non-linear range. The method of single NR with table approximation is presented in M2 and M3, where a large table size ($2^{10} \times 10$ bits) is used in M2 and a small table size ($2^7 \times 7$ bits) is used in M3. The method of different piecewise linear approximation in softly non-linear range with single NR has been presented in M4 and M5, where there are three-piecewise segments in M4 and four-piecewise segments in M5. The mean absolute errors of the different methods are also denoted in Fig. 1. It can be seen that the proposed M4 provides relatively fewer error than M3, which indicates that three-piecewise linear approximation is more effective (higher precision) than table approximation using the size of $2^7 \times 7$ bits. It has an average error of 10^{-5} in M4. It can be found that M1 has a slightly better precision than M4 on the average because it uses more coefficients in approximation. M2 gives the best precision among all the methods on average, which implies more memory bits with NR method can improve the precision effectively. Compared with M1, M5 shows a slightly better precision at the same number of approximation coefficients (eight even and eight odd coefficients in M5, sixteen coefficients in M1). This indicates that the softly non-linear approach outperforms the highly non-linear method for linear approximation. The precision gain obtained by M2 and M5 is based on higher complexity (more memory bits in M2 and more approximating coefficients in M5) compared to M3 and M4, respectively. To find a good performance trade-off, the proposed M4 is a competitive option for high precision and high throughput applications.

B. Proposed Approximating Method and Hardware Architecture

Initial approximation and NR method are combined in this paper to yield a competitive method in terms of throughput and precision. A softly non-linear function $y = 1/\sqrt{c}$, $1 \le c < 2$ is presented to solve $y = 1/\sqrt{x}, 0 < x \le 1$ so as to get the initial seed. The variable x is represented in (2), where α is the number of leading zero, and w is the width of c. In addition, $c_{w-1} = 1$. Thus, inverse square root can be derived in (3) based on the parity of number α (even and odd). Integer k is a positive number. Number of leading zero (α) can be detected using shifts and additions so that no more extra complexity will be consumed. Three-piecewise linear approximation (belongs to the interpolation table category) is used to compute the seed (y_0) in (4), where a and b indicate the multiplier and addition coefficient, respectively. The coefficients obtained through piecewise fitting are shown in Table I.

TABLE I. FITTING COEFFICIENTS.

Coefficient	Segment	Even	Odd
a	1	-0.4081	-0.5771
	2	-0.2875	-0.4066
	3	-0.2080	-0.2942
b	1	1.4038	1.9852
	2	1.2485	1.7656
	3	1.1209	1.5852

$$x = \underbrace{0.0...0}_{\alpha} c_{w-1}...c_0, \tag{2}$$

$$y = \frac{1}{\sqrt{x}} = \frac{1}{\sqrt{2^{-\alpha} c}} = 2^{\alpha/2} \frac{1}{\sqrt{c}} = \begin{cases} 2^k \dfrac{1}{\sqrt{c}}, & \alpha = 2k, \\ 2^k \dfrac{\sqrt{2}}{\sqrt{c}}, & \alpha = 2k+1, \end{cases} \tag{3}$$

$$y_0 = 2^k (ax + b). \tag{4}$$

After the initial seed is obtained, single iterated NR method is applied to get the required precision. NR method can be derived by function f and its first derivation f' in (5) and (6), respectively. Being substituted, NR algorithm can be expressed by (7), where y_0 is the initial seed:

$$f(y) = 1/y^2 - x, \tag{5}$$

$$y_{n+1} = y_n - f(y_n)/f'(y_n), \tag{6}$$

$$y_{n+1} = y_n (3 - x y_n^2)/2. \tag{7}$$

The proposed architecture is illustrated in Fig. 2, in which y is the inverse square root of input x. $P0$ is the three-piecewise linear approximation of $1/\sqrt{c}$ and $\sqrt{2}/\sqrt{c}$ for even and odd, respectively. The left shifting by k bits in $P0\_shift$ is supposed to accomplish (3). Three multipliers are needed in the NR method. In order to reduce the complexity, the additional multiplier in the initial approximation process is eliminated by MCM technique (also known as common sub-expression elimination (CSE)). The principle of MCM is to reduce the complexity by finding common items in expressions, and the multiplication is reduced by additions and shifts. The multiplication (ax) is reduced by Spiral Project [17]. In the single precision floating point standard, there are 23 bits of fraction. To be applied to a wide application, 28 bits of fixed-point is chosen in this paper (x is scaled by 2^{26}).

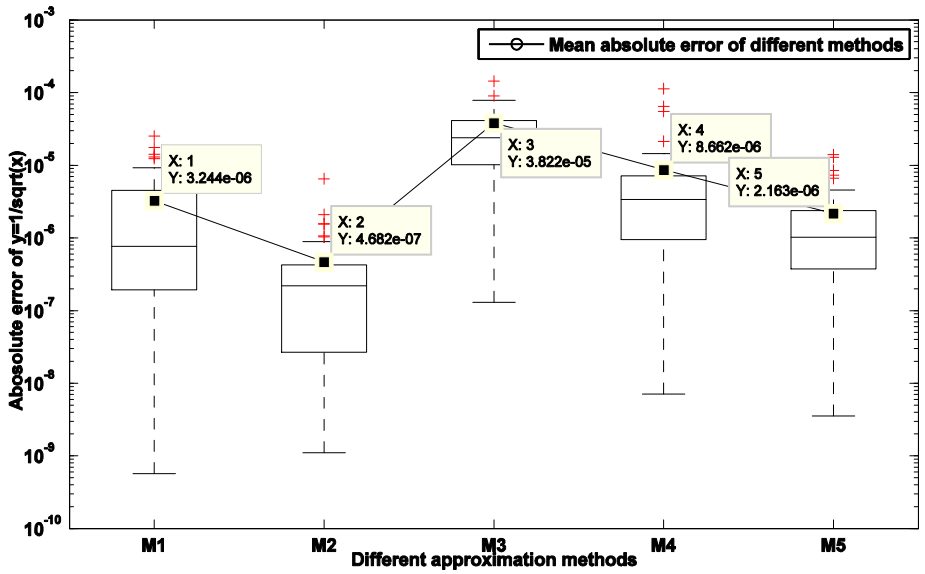

Fig. 1. Absolute error (E_{method}) comparison of different approximating methods.

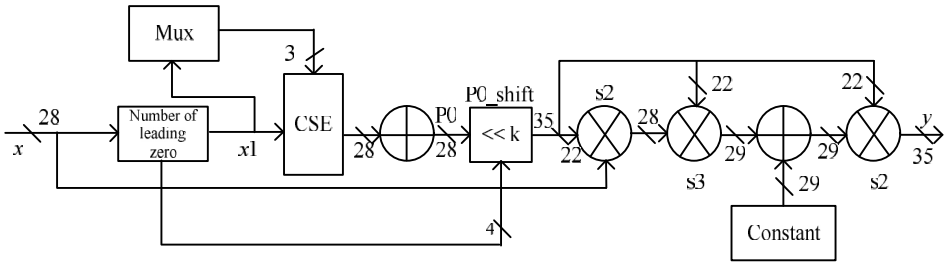

Fig. 2. Proposed hardware architecture of inverse square root.

III. RESULTS AND COMPARISON

A. Implementation

To evaluate the performance of different methods, hardware architectures of inverse square root have been implemented using field programmable gate array (FPGA). Functional verifications are completed by Modelsim SE 6.2b. Synthesized results are obtained by Quartus II. Performance comparison ofdifferent approximating methods (shown in Fig. 1) in calculating $z/\sqrt{x}$ has been illustrated in Table II in terms of complexity and throughput. The complexity is measured by the logic element (*LE*), registers (*Reg*), memory bits (*Mem*), and 9-bits multipliers (*Mult*). The target device is Cyclone II EP2C35F672C6. Throughput is measured using the synthesized maximum frequency by million samples per second (*Msps*). Initial cycles (*Delayed Cycles*) are the number of clocks to compute the first output valid data. The anchor IP core (a divider plus a square root) is chosen from Quartus II 10.1 (Altera), with 20 pipelines in IP1 and 40 pipelines in IP2. The proposed hardware architecture (M4) is full pipelined. The hardware architecture of M2 and M3 can be found in previous work [10]. The hardware architecture of M11 is the same as M1 except that multiplication item (ax in (4)) is eliminated using additions and shifts by the technique of MCM. The multipliers in M4 and M5 are also eliminated by MCM so that no memory bits are consumed.

It can be seen that more than 70 % throughput is achieved in M4 compared to IP1 at the cost of additional logic resources (8.2 %) and multipliers. Compared with IP2, there is still throughput improvement (15 %) in M4, but the resource usage is lower and the initial latency is shorter. Compared with the earlier developed direct linear approximating methods (M1 and M11), nearly 45 % throughput is obtained by M4. It should be pointed out that M5 outperforms M11 in both throughput and complexity (49 % of throughput improvement and 9 % of complexity reduction). This indicates that softly non-linear approximating outperforms highly non-linear method in complexity (Table II) and precision (Fig. 1). The direct table approximation combined with single NR method (M2 and M3 in Table II) consumes much memory bits. It can be seen from Fig. 1 and Table II that almost 100× higher precision can be obtained in M2 at the cost of 10× more memory bits compared to M3, which implies that a sufficient precision gain is achieved by the combination approach. If a very high precision is required, more NR iterations and more memory consumptions will be needed. Compared to direct table approximation methods (M2 and M3), the softly linear approximation methods (M4 and M5) achieve relatively comparable throughput, whereas releasing the burden of memory and decreasing the latency.

To validate the precision of different implemented methods, simulated data from different hardware architectures has been used to analyse the absolute error, as shown in Fig. 3. Compared to the IP, all the hardware architectures (from M1 to M5) have a better precision, and about 100× higher precision improvement has been obtained in the proposed M4. Additionally, M11 has the same precision as M1 because they use the same approximating method. Taken into account the factors of throughput,

complexity and precision, the proposed M4 offers a competitive option for most applications. It requires only single NR iteration and achieves an average precision of 10^{-5}. As a result, the proposed hardware architecture shows high throughput and high precision improvement over IP from Altera at the cost of almost the same logic element resources.

B. Application in Cholesky Decomposition

Cholesky decomposition is the key process in matrix factorization, and inverse square root is the critical path in Cholesky factoring. High throughput hardware architecture of inverse square root can accelerate Cholesky decomposition.

The algorithm of column-oriented Cholesky decomposition is illustrated in Fig. 4, where N is the order of matrix A and a is the element of matrix A. Variables i, j, k are circulating elements. In order to avoid the long latency delay of division in *PE*2, inverse square root and a multiplier are implemented to replace *PE*1 and *PE*2 in Fig. 4. Different hardware architectures in previous section are implemented to evaluate the performance of Cholesky decomposition.

Hardware architecture of Cholesky decomposition is illustrated in Fig. 5, where the module of *Inverse Square Root* is implemented by the method of M1, M3, M4 and IP, respectively. Parallel multipliers are proposed to process the data so as to accelerate the updating task. Three-port random access memory (RAM) is used to store the input and output matrix data. The module of *Control logic* is served as the status controller.

To demonstrate the efficiency of the proposed inverse square root, comparison results of Cholesky decomposition using different inverse square root approximating methods are shown in Table III. Target device is Stratix II EP2S130F1508C3 for its many input/output pins. The throughput (T) is determined by (8), where S denotes the input samples per cycle, and $f_{\max}$ is the maximum frequency (F_{max} in Table III).

TABLE II. PERFORMANCE COMPARISON IN COMPUTING $z/\sqrt{x}$.

Method	*LE*	*Reg*	*Mem*	*Mult*	*Throughput/Msps*	*Delayed Cycles*
IP1	1132	693	88	0	80.54	21
IP2	1434	1202	286	0	115.58	41
M1	753	425	368	36	133.9	22
M11	1631	950	0	32	136.76	20
M2	1035	775	10240	20	137.7	22
M3	925	719	896	16	139.12	22
M4	1225	682	0	32	137.53	19
M5	1487	838	0	32	141.6	19

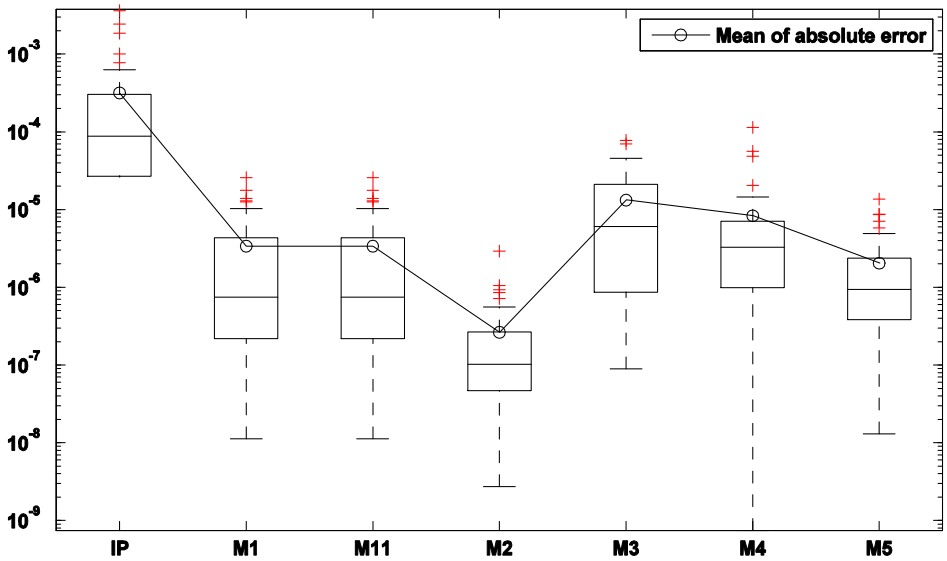

Fig. 3. Absolute error (E_{total}) analysis of different implemented hardware architecture.

TABLE III. NINE-ORDER CHOLESKY DECOMPOSITION OF DIFFERENT APPROXIMATING METHODS.

Method	*ALUT*	*Reg*	*Mem*	*Mult*	F_{max}	*Throughput*
M0 [18]	4123	3870	88	16	124.8	11.29
M1	4866	4098	368	168	95.75	7.69
M11	4664	4173	0	160	187.44	15.06
M3	4275	3980	896	152	175.25	13.64
M4	4530	3924	0	160	193.46	16.06

Variable t_u represents the microsecond time consumed in solving each 9×9 matrix at the speed of 100 MHz

$$T = S \times f_{\max} / (100 \times t_u). \quad (8)$$

```
repeat k from 1 το N
    α(k,k) = square root(a(k,k));    PE1
    repeat i from k + 1 το N
        α(i,k) = a(i,k) / a(k,k);    PE2
    end
    repeat j from k + 1 το N
        repeat i from j το N
            α(i,j) = α(i,j) − α(i,j) × α(j,k);
        end
    end
end
```

Fig. 4. Algorithm of column-oriented Cholesky decomposition.

In Table III, M0 is the anchor Cholesky decomposition based on IP [18]. M1, M11, M3 and M4 denote the Cholesky decomposition using corresponding method in Table II. Compared with M0, it can be seen that there are throughput improvements in M11, M3 and M4. M0 outperforms M1 because more pipelines are used in the IP (M0). It can be seen that the proposed M4 shows the best throughput performance in Table III. It achieves nearly 42 % throughput improvement over previous work [18]. The throughput improvement in M4 is obtained at the cost of a high complexity (9.8 % of more Adaptive Look-up Tables (ALUTs) and 9× of more multipliers). As a result, a high throughput inverse square root is achieved in this paper and no memory bits are required by the utilization of MCM technique.

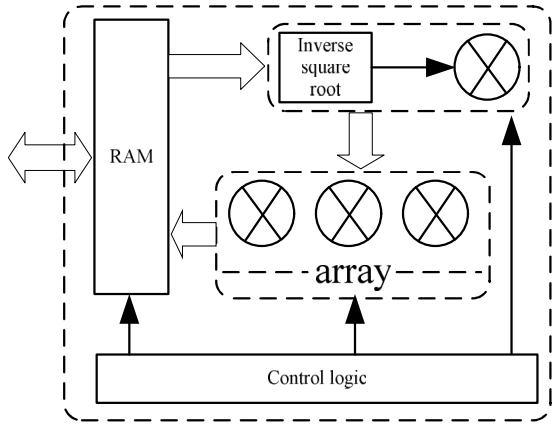

Fig. 5. Hardware architecture of Cholesky decomposition.

C. Comparison

As for the comparison to other reports, it will make sense from the point of some performance metrics. The proposed hardware architecture achieves higher precision than [14] as more segments will bring higher precision (indicated by M4 and M5). In [15], single precision floating point is used. It uses a similar approach as M2 and M3 in the mantissa. The single precision floating point representation is not the best choice to specific application. In addition, much memory will be consumed to get a high precision. Area efficiency is focused in [13] with no multipliers are used. It can reach a precision of 10^{-3} at four segments while 10^{-5} of precision can be obtained in this paper at the same segments. Compared to [18], the proposed architecture leads to 42 % of throughput improvement when applied in Cholesky decomposition.

IV. CONCLUSIONS

Single iterated multiplicative inverse square root has been implemented in this paper. The proposed method combines the techniques of multiple piecewise linear approximation in softly nonlinear range, single iterated NR method and MCM technique to achieve high precision and high throughput. The implemented hardware architecture reduces the complexity by eliminating multiplier with additions and shifts. As a result, the proposed method offers a good performance trade-off that yields high throughput, high precision with relatively low complexity. Experiment results indicate that the proposed inverse square root hardware architecture is a competitive solution suitable for matrix factoring, and being integrated into larger architecture designs such as Cholesky decomposition and compressed imaging with significant performance improvements.

REFERENCES

[1] D. Wang, P. Ren, L. Liu, "A high-throughput fixed-point complex divider for FPGAs", *IEICE Electron. Express*, vol. 10, no. 4, pp. 1–8, 2013. [Online]. Available: http://doi.org/10.1587/elex.10.20120879

[2] F. Martin del Campo, A. Morales-Reyes, R. Perez-Andrade, R. Cumplido, A. G. Orozco-Lugo, C. Feregrino, "A multi-cycle fixed point square root module for FPGAs", *IEICE Electron. Express*, vol. 9, no. 11, pp. 971–977, 2012. [Online]. Available: http://doi.org/10.1587/elex.9.971

[3] M. Ito, N. Takagi, S. Yajima, "Efficient initial approximation for multiplicative division and square root by a multiplication with operand modification", *IEEE Trans. Comput.*, vol. 46, no. 4, pp. 495–498, 1997. [Online]. Available: http://doi.org/10.1109/12.588066

[4] F. Auger, Zhen Lou, B. Feuvrie, Feng Li, "Multiplier-free divide, square root, and log algorithms", *IEEE Signal Process. Mag.*, vol. 28, no. 4, pp. 122–126, 2011. [Online]. Available: http://doi.org/10.1109/MSP.2011.941101

[5] P. Salmela, A. Burian, T. Jarvinen, A. Happonen, J. H. Takala, "Low-complexity inverse square root approximation for basebank matrix operation", *ISRN Signal Processing*, vol. 2011, 2011. [Online]. Available: http://dx.doi.org/10.5402/2011/615934

[6] M. Ercegovac, J. Muller, A. Tisserand, "Simple seed architectures for reciprocal and square root reciprocal", *Conf. Rec. Asilomar Conf. Signals Syst. Comput. Arith.*, Pacific Grove, USA, 2005, pp. 1167–1171. [Online]. Available: http://dx.doi.org/10.1109/ACSSC.2005.1599944

[7] M. J. Schulte, K. E. Wires, "High-speed inverse square roots", in *Proc. 14th Symp. Comput. Arith.*, Adelaide, 1999, pp.124–131. [Online]. Available: http://dx.doi.org/10.1109/ARITH.1999.762837

[8] L. V. Moroz, C. J. Walczyk, A. Hrynchyshyn, V. Holimath, J. L. Cieslinski, "Fast calculation of inverse square root with the use of magic constant − analytical approach", *arXiv preprint arXiv:* 1603.04483, 2016.

[9] N. Thuning, L. Barring, "Hardware architectures for the inverse square root and the inverse functions using harmonized parabolic synthesis", M.S. thesis, Dept. Elect. Info. Tech., Lund Univ., Lund, Sweden, 2016.

[10] J. Luo, Q. Huang, S. Chang, H. Wang, "Hardware efficient architecture for compressed imaging", *IEIEC Electron. Express*, vol. 11, no. 14, pp. 1–12, 2014. [Online]. Available: http://doi.org/10.1587/elex.11.20140562

[11] D. Das Sarma, D. W. Matula, "Faithful interpolation in reciprocal tables", in *Proc. 13th Symp. Comput. Arith.*, Asilomar, 1997, pp. 82–91. [Online]. Available: http://doi.org/10.1109/ARITH.1997.614882

[12] D. S. Debjit, D. W. Matula, "Faithful bipartite ROM reciprocal tables", in *Proc. 12th Symp. Comput. Arith.*, Bath, 1995, pp. 17–28.

[13] M. D. Ercegovac, T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Boston: Kluwer Academic, 1994.

[14] J. A. Pineiro, J. D. Bruguera, "High-speed double-precision computation of reciprocal, division, square root, and inverse square root", *IEEE Trans. Comput.*, vol. 51, no. 12, pp. 1377–1388, 2002. [Online]. Available: http://doi.org/10.1109/TC.2002.1146704

[15] M. D. Ercegovac, T. Lang, J. Muller, A. Tisserand, "Reciprocation, square root, inverse square root, and some elementary functions using small multipliers", *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 628–637, 2000. [Online]. Available: http://doi.org/10.1109/12.863031

[16] M. D. Ercegovac, L. Imbert, D. W. Matula, J. Muller, G. Wei, "Improving Goldschmidt division, square root, and square root reciprocal", *IEEE Trans. Comput*., vol. 49, no. 7, pp. 759–763, 2000. [Online]. Available: http://doi.org/10.1109/12.863046

[17] Y. Voronenko, M. Puschel, "Multiplierless multiple constant multiplication", *ACM Trans. Algorithms*, vol. 3, no. 2, pp. 1–37, 2007. [Online]. Available: http://doi.org/10.1145/1240233.1240234

[18] J. Luo, Q. Huang, S. Chang, X. Song, Y. Shang, "High throughput Cholesky decomposition based on FPGA", in *Proc. 6th Int. Congr. Image Signal Process.*, Hangzhou, 2013, pp. 1649–1653. [Online]. Available: http://doi.org/10.1109/CISP.2013.6743941