Hardware Efficient Reciprocal Using Second Order Harmonized Parabolic Synthesis and Squaring Shrunk Method

Jun Luo, Hongwei Luo, Yue Zhi, Xiaoqiang Wang, Hongfeng Lv, Ming Dang
China Electronic Product Reliability and Environmental Testing Research Institute,
No. 110, Dongguanzhuang Road, Guangzhou, Guangdong, China
kyea168@126.com

***Abstract*—In applications as in wireless communication, computer graphics and digital signal processing, a massive of complex matrix operations is often performed. Reciprocal is computed in large quantities in these matrix operations. To obtain high performance, efficient algorithm and hardware architecture are important in terms of low cost, low computation time and high precision. Second order first sub-function and squaring shrunk method have been proposed to build efficient hardware architecture for reciprocal using field programmable gate array. Second order first sub-function in harmonized parabolic synthesis is presented to improve the approximating precision and decrease the memory usage at the cost of additional multipliers. To further reduce the complexity, squaring shrunk method is proposed to decrease the expensive cost of multipliers. The combination of these techniques yields good performance trade-off. Precision simulation and hardware implementation result has shown that hardware reciprocal of high precision, low memory and low multiplier usage has been obtained compared to traditional first order first sub-function harmonized parabolic synthesis method.**

***Index Terms*—Digital integrated circuits; Parallel architectures; High performance computing; Field programmable gate arrays.**

I. INTRODUCTION

Arithmetic element functions are playing very important roles in wireless communication, computer graphics and digital signal processing, reciprocal is one of these functions which are frequently computed in matrix operations [1]. Because of the characteristic of high throughput and low latency, hardware implementation has become a main approach in computing acceleration. To approximate reciprocal, a lot of algorithms have been developed so far. Look up table [2], [3], Newton-Rapshon (NR) method [4], [5], Coordinate Rotation Digital Computer (CORDIC) [6], series expansion [7], and *et al*. are of these algorithms developed to compute reciprocal. The key problem stated among these algorithms is how to find the best trade-off among precision, convergence speed, throughput, latency and cost. Look up table is easy to implement, but the memory size will be huge when a high precision is required. Newton-Raphson method is a multiplicative method, it has the property of quadratic convergence, but it consumes additional multipliers. CORDIC is a subtractive technique, which is hardware friendly, but it occupies a long latency especially at a high precision. Series expansion is also multiplicative based, which is a polynomial approximation method. It has the advantages of fast convergence, but the mass multipliers used in hardware are expensive.

In order to design efficient hardware architecture for reciprocal, parabolic synthesis method has been presented [8]. It uses the product of series of sub-functions to approximate the original reciprocal function, and it indicates performance improvement over CORDIC and Newton-Raphson method [9]. Parabolic synthesis method has shown a wide application in arbitrary function, e.g., sine and cosine function [10], [11], logarithmic and exponential function [12], and roots, inverse and inverse roots function [13]. To further release the hardware burden and improve the computing precision of reciprocal, harmonized parabolic synthesis (HPS) [14] and non-linear interpolation [15] have been proposed recently. For example, a simple first order first sub-function with non-linear interpolation second order second sub-function in harmonized parabolic synthesis for reciprocal has been analysed in [14] and [16], it shows performance improvement over Newton-Raphson method.

To construct a hardware efficient implementation of reciprocal with high precision and low memory and multiplier usage, second order first sub-function and squaring shrunk method have been proposed in this paper. The proposed method inherits the harmonized parabolic synthesis method with non-linear interpolation in second sub-function as shown in [14] and [16], and constructs second order first sub-function to obtain a high precision. The proposed second order first sub-function reduces the memory usage by employing the symmetric property of the first help function. To overcome the additional consumed expensive multipliers in second order first sub-function, squaring shrunk method is employed in both the first and the second sub-function. It reduces the hardware complexity by decreasing the number of used multipliers as well as shrinking the multipliers to squares. The proposed harmonized parabolic synthesis with second order first sub-function and non-linear interpolation in second sub-function combined with squaring shrunk method has obtained performance improvement for hardware architecture of reciprocal in terms of precision and complexity (memory bits and multipliers). It shows a competitive solution for reciprocal in application of field

Manuscript received 19 November, 2017; accepted 8 March, 2018.

programmable gate array (FPGA) and acceleration of complex algorithms.

II. RECIPROCAL APPROXIMATION METHOD

A. Newton-Raphson Method

Newton-Raphson method has a long history [4], it is proposed to solve the non-linear problem by using of linear equations. NR method for computing reciprocal ($z = 1/v$) can be expressed by (1), where t represents the number of iterations

$$z_{t+1} = 2z_t - v \times z_t^2. \tag{1}$$

In order to obtain a fast convergence, look-up table (LUT) and NR method are combined [17]. LUT is used to acquire a relatively precise initial value, and NR method is employed to get a high precision of reciprocal approximating. Let the size of the LUT is $2^h \times d$ bits, the binary representation of the input $(1 < v < 2)$ and output $(0.5 < z < 1)$ can be expressed by (2) and (3), respectively. As a result, the table entry of reciprocal can by denoted by (4), where $ii = 0,1,2,...,2^h - 1$, and $LUT(ii)$ is the pre-computed initial value to be stored by LUT, and $addr(ii)$ is the access address of the LUT, and $\lfloor \ \rfloor_{d+1}$ represents keeping $d+1$ bits when fixing to zero:

$$v: \ 1.\underbrace{bbb...b}_{hbits}bb, \tag{2}$$

$$z: \ 0.1\underbrace{bbb...b}_{dbits}bb, \tag{3}$$

$$LUT(ii) = (\left\lfloor \frac{1}{1 + addr(ii) \times 2^{-h} + 2^{-h}} \right\rfloor_{d+1} - \frac{1}{2}) \times 2^{d+1}. \tag{4}$$

The hardware architecture of NR method is illustrated in Fig. 1, it uses table size of $2^{10} \times 10$ bits, and consumes two additions and two multipliers. The input data binary is composed of 1 bit of integer and 15 bits of fraction.

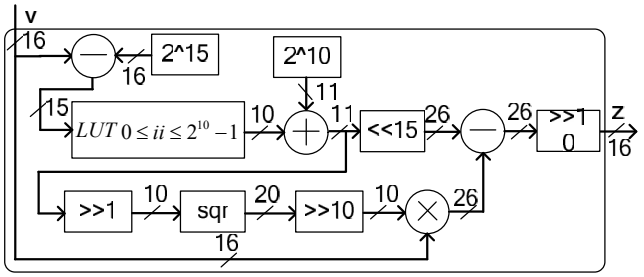

Fig. 1. Hardware architecture of NR method with single iteration and initial value approximation based on LUT.

B. Harmonized Parabolic Synthesis Method

The normally form of reciprocal computation can be shown in (5), it is the mantissa of the floating-point number [14]. In the harmonized parabolic synthesis method, the input and output variable should be normalized to the range of [0, 1]. Thus, a pre-processing step is needed, as shown in (6), which normalizes the input variable (x). As to normalizes the output variable (y), the target computing function can be expressed in (7). Finally, the original reciprocal function (5) can be computed through (8), which is an after-processing step. As the pre- and after-processing step is easy to be implemented in hardware by addition and shifting, the target function (7) is mainly focused, computed and analysed in this paper

$$z = \frac{1}{v}, \tag{5}$$

where $1 \le v < 2, \ 0.5 < z \le 1$.

$$x = v - 1, \tag{6}$$

$$y = \frac{2}{1+x} - 1. \tag{7}$$

where $0 \le x < 1, 0 < y \le 1$.

$$z = \frac{1}{2}(y+1). \tag{8}$$

Parabolic synthesis is a newly developed method in approximating unary functions, which is first presented by Erik in [8]. It uses the product of a series of second order sub-functions, defined as $s_1(x)$, $s_2(x)$, **...** , $s_\infty(x)$ to approximate the original function, defined as $f_{org}(x)$, shown in (9)

$$f_{org}(x) = s_1(x) \times s_2(x) \times ... \times s_\infty(x), \tag{9}$$

where $1 \le x < 1, 1 \le f < 1$. In parabolic synthesis, the number of sub-functions affects the approximating accuracy. To construct hardware friendly architecture, harmonized parabolic synthesis (HPS) method has been developed [18], which uses two sub-functions, as defined in (10). HPS simplifies the approximating process, which leads to hardware complexity reduction. To avoid precision loss, it often combines with non-linear interpolation method, which will be introduced in section C

$$f_{org}(x) = s_1(x) \times s_2(x), \tag{10}$$

where $1 \le x < 1, 1 \le f < 1$. In parabolic synthesis, the first help function, $f_1(x)$, is defined as the quotient of the original function and the first sub-function, as shown in (11). The first help function is used in HPS to estimate the second sub-function

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} = s_2(x) \times ... \times s_\infty(x). \tag{11}$$

C. First Order First Sub-Function Based Harmonized Parabolic Synthesis Method (FOFS-HPS)

In the HPS method, in order to implement reciprocal with high accuracy and low complexity, constructing sufficient sub-functions has become the key. To simplify the hardware implementation, simple first order first sub-function has been

presented in [14] and [16] for approximating reciprocal. However, it has the drawback of low convergence and low accuracy. To overcome this issue, non-linear interpolation has been utilized in the second sub-function in HPS, which can compensate the precision loss. The first order first sub-function based HPS method (FOFS-HPS) is shown in (12), where i represents the i^{th} interval (total number of intervals is $I = 2^n$) divided in the first help function, and x_w is defined in (13) which can be implemented by shifting in hardware:

$$\begin{cases} s_1(x) = 1 - x, \\ s_2(x) = l_{2,i} + j_{2,i} \times x_w - c_{2,i} \times {x_w}^2, \end{cases} \tag{12}$$

$$x_w = fract(2^n \times x), \tag{13}$$

where $l_{2,i}$, $j_{2,i}$ and $c_{2,i}$ are constant coefficients, and they are defined by (14), (15) and (16), respectively. $x_{start,i}$ and $x_{end,i}$ denotes the starting and ending point of the i^{th} interval, separately. $k_{2,i}$ is the slope of the i^{th} interval, as shown in (17). The employment of non-linear interpolation can approximate the second sub-function more precisely through second order multiple piecewise approximation:

$$l_{2,i} = f_1(x_{start,i}), \tag{14}$$

$$j_{2,i} = k_{2,i} + c_{2,i}, \tag{15}$$

$$c_{2,i} = 4 \times (f_1(\frac{x_{start,i} + x_{end,i}}{2}) - l_{2,i} - 0.5 \times k_{2,i}), \tag{16}$$

$$k_{2,i} = f_1(x_{start,i}) - f_1(x_{end,i}). \tag{17}$$

The hardware architecture of FOFS-HPS method is illustrated in Fig. 2, in which 16 intervals $(n = 4)$ is used as an example. Input and output variables are 15 bits and 16 bits, respectively. It costs three multipliers, one square and three additions. Among which, only an addition is consumed in the first sub-function, leading to a low complexity implementation.

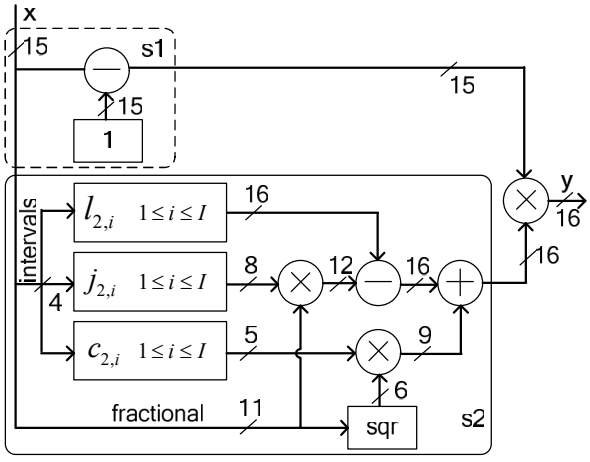

Fig. 2. Hardware architecture of first order first sub-function with harmonized parabolic synthesis (FOFS-HPS).

D.Proposed Method Using Second Order First Sub-Function and Squaring Shrunk in Harmonized Parabolic Synthesis

The limitation of traditional FOFS-HPS method is that the low accuracy in the first sub-function may result in more intervals in the second sub-function so as to obtain a relatively high precision. To overcome this issue, HPS with second order first sub-function (SOFS-HPS) has been proposed in this paper. As shown in (18), a second order approximation in the first sub-function has been constructed, leading to a precise approximation of $s_1(x)$. In (18), i represents the i^{th} interval divided in the first help function, and x_w, $l_{2,i}$, $j_{2,i}$ and $c_{2,i}$ are the same as in (14), (15), (16) and (17), respectively:

$$\begin{cases} s_1(x) = 1 - \frac{1}{2} x \times (3 - x), \\ s_2(x) = l_{2,i} + j_{2,i} \times x_w - c_{2,i} \times {x_w}^2. \end{cases} \tag{18}$$

Because of the using of SOFS, the first help function of HPS can exploit the symmetric property which can be utilized to save the memory bits, as illustrated in Fig. 3. It can be found that the first help function using SOFS has the property of symmetric, which can lead to nearly half of memory bits saving in hardware compared to the FOFS method.

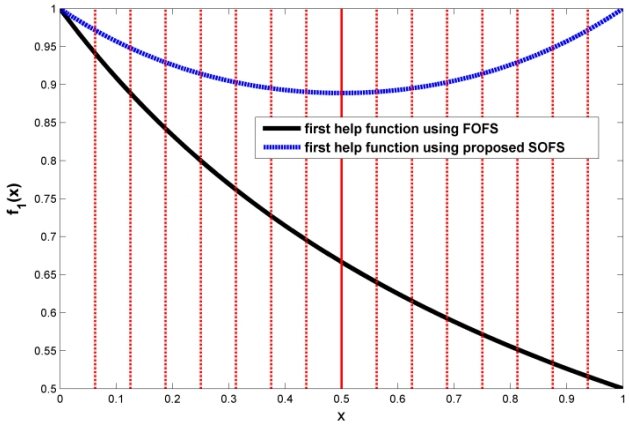

Fig. 3. The first help function using conventional first order first sub-function (FOFS) and proposed second order first sub-function (SOFS).

Hardware architecture of proposed SOFS-HPS is shown in Fig. 4, where $n = 4$ and the input and output width is the same as in Fig. 2. It can be seen from (17) and (18) that the main difference of approximating reciprocal relies on the first sub-function. High accuracy is obtained by using of second order approximation. Hardware architectures (from Fig. 2 and Fig. 4) indicate that the proposed SOFS-HPS method can reduce nearly half of the memory bits at the cost of an additional multiplier and addition.

To further release the hardware complexity burden of the proposed SOFS-HPS method, squaring shrunk method has been employed. The fundamental of the proposed HPS with squaring shrunk method relies on the fact that squarer has lower complexity than multiplier. The complexity analysis is based on the following assumption and hypothesis.

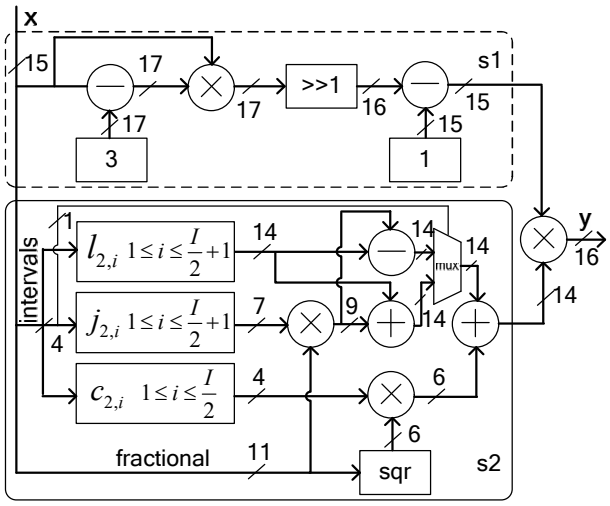

Fig. 4. Hardware architecture of proposed second order first sub-function with harmonized parabolic synthesis (SOFS-HPS).

Assumption 1

Define the following macros (multiplier (M_{mul}), squarer (M_{sqr}) and adder (M_{add})) used in approximating reciprocal, and these macros have the properties in (19) and (20):

$$T_{mul} > T_{sqr} >> T_{add}, \tag{19}$$

$$C_{mul} >> C_{sqr} >>> C_{add}, \tag{20}$$

In the assumption, T represents the latency of the macro, and C denotes the complexity of the macro. Obviously, the multiplier is the most complexity and slowest macro among three.

Hypothesis 1

To evaluate the target second polynomial function $\Psi(x)$, where a_2, a_1, a_0 are constant coefficients

$$\Psi(x) = a_2 x^2 + a_1 x + a_0. \tag{21}$$

It is advantage (considering complexity and latency) to use the equivalent squaring function, where p, m and k are constant coefficients

$$\Psi(x) = p(x+m)^2 + k. \tag{22}$$

The squaring function in (22) needs one addition, one squarer followed by another multiplication and addition. However, the target second polynomial function in (21) requires one squarer, two multipliers and two adders. The complexity and latency comparison is shown in Table I. It can be seen that the advantage of the function in (22) is obvious in terms of hardware cost where two multipliers are shrunk to a squarer while the latency is the same as in (21).

The proposed SOFS-HPS using squaring shrunk method in approximating reciprocal is shown in (23), where $p_{2,i}$, $m_{2,i}$ and $k_{2,i}$ are defined by (24). It can be seen that two adders and one squarer are needed in $s_1(x)$ and the coefficients symmetric property can be inherited in the second sub-function $s_2(x)$:

$$\begin{cases} s_1(x) = \frac{1}{2}(x-\frac{3}{2})^2 - \frac{1}{8}, \\ s_2(x) = p_{2,i}(x_w + m_{2,i})^2 + k_{2,i}, \end{cases} \tag{23}$$

$$\begin{cases} p_{2,i} = -c_{2,i}, \\ m_{2,i} = -\dfrac{j_{2,i}}{2c_{2,i}}, \\ k_{2,i} = l_{2,i} + \dfrac{j_{2,i}^2}{4c_{2,i}}. \end{cases} \tag{24}$$

TABLE I. COMPLEXITY AND LATENCY COMPARISON.

Category		Function in (21)	Function in (22)
Latency		$T_{mul} + T_{sqr} + 2T_{add}$	$T_{mul} + T_{sqr} + 2T_{add}$
Complexity	Number of multipliers	2	1
	Number of squares	1	1
	Number of adders	2	2

As multipliers are shrunk to squares, the hardware complexity of the proposed SOFS-HPS method is reduced a lot. Number of resources usage is compared in Table II. Compared to FOFS-HPS and proposed SOFS-HPS method, the proposed SOFS-HPS with squaring shrunk method consumes the least multipliers at the cost of one additional squarer without latency and precision loss. It also saves the memory bits comparing to FOFS-HPS method.

TABLE II. COMPLEXITY COMPARISON OF DIFFERENT METHODS.

Complexity	FOFS-HPS	Proposed SOFS-HPS	Proposed SOFS-HPS with squaring shrunk
Number of multipliers	3	4	1
Number of squares	1	1	2
Number of adders	3	5	3
Number of coefficients being stored (memory)	$3I$	$\frac{3}{2}I + 2$	$2I + 1$

The hardware architecture of the proposed SOFS-HPS with squaring shrunk method is shown in Fig. 5, where $n = 4$ is illustrated. The proposed method has the same approximating accuracy as SOFS-HPS, which devotes a high precision. By utilizing second order first sub-function, non-linear interpolation in the second sub-function, and squaring shrunk method, hardware implementation of the proposed SOFS-HPS with squaring shrunk method for reciprocal yields good trade-off between precision and complexity.

The proposed SOFS-HPS with squaring shrunk method has advantages in considering the precision and complexity of implementing reciprocal in hardware. On one side, the proposed method improves the approximating precision by using of second order first sub-function at the cost of additional resources. It reduces the memory bits usage as well by exploiting the symmetric property. On the other side, squaring shrunk method is proposed in this paper to reduce

the multiplier consumption. Thus, a good trade-off between approximating precision and hardware complexity has been obtained which leads to hardware efficient reciprocal implementation.

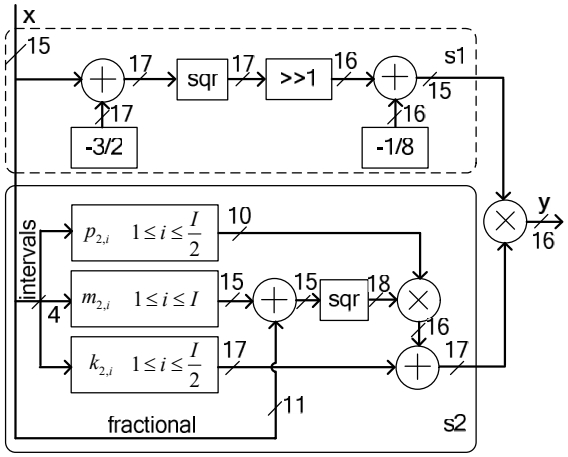

Fig. 5. Hardware architecture of the proposed SOFS-HPS with squaring shrunk method.

III. ANALYSIS, RESULT AND COMPARISON

A. Precision Analysis

To evaluate the precision of different approximating method, error (err) and mean error (err_mean) are used as the precision metric as shown in (25) and (26), where $z_{r,org}$ and $z_{r,est}$ are defined as the $r'th$ accurate and approximate reciprocal result, respectively. Number R denotes the total number of computed points within the input range $(v \in (1,2))$, $R = 100$ is chosen to analyse the error level in this paper:

$$err = \left| z_{r,est} - z_{r,org} \right|, \tag{25}$$

$$err\_mean = \frac{1}{R} \sum_{r=1}^{R} \left(\left| z_{r,est} - z_{r,org} \right| \right). \tag{26}$$

Figure 6 illustrates the error of LUT based NR method, FOFS-HPS method and the proposed SOFS-HPS with squaring shrunk method. The non-linear interpolation intervals are 16 (n = 4). To be clarified, the proposed SOFS-HPS with squaring shrunk method has the same error distribution as SOFS-HPS method. Because they are using the same approximating equations in software even though their hardware architectures are different. It can be seen that the proposed method has a relatively higher precision than FOFS-HPS method, and it shows comparable precision as NR method with single iteration and table size of $2^{10} \times 10$ bits. It is also shown that NR method with two iterations and table size of $2^7 \times 7$ bits has the highest precision among them.

Figure 7 displays the average error of different approximating methods. It can be found that the proposed method yields an average precision promotion by 98 % compared to FOFS-HPS method considering different interpolation intervals. More intervals will lead to lower average error. The proposed SOFS-HPS with 64 non-linear interpolation intervals in the second sub-function method can reach an average error magnitude of 10^{-8}. It can also be seen that NR method has a faster convergence with the increasing of iteration. Nevertheless, HPS method has a lower convergence with the increasing of intervals.

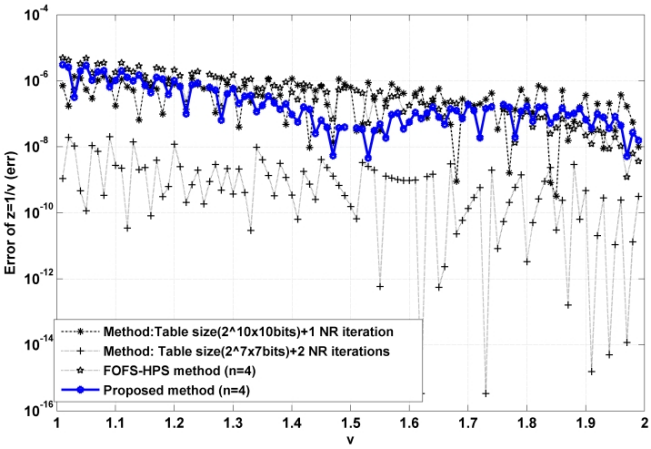

Fig. 6. Error of the proposed, FOFS-HPS and NR method in the input variable range (1, 2) when intervals are 16 $(n = 4)$.

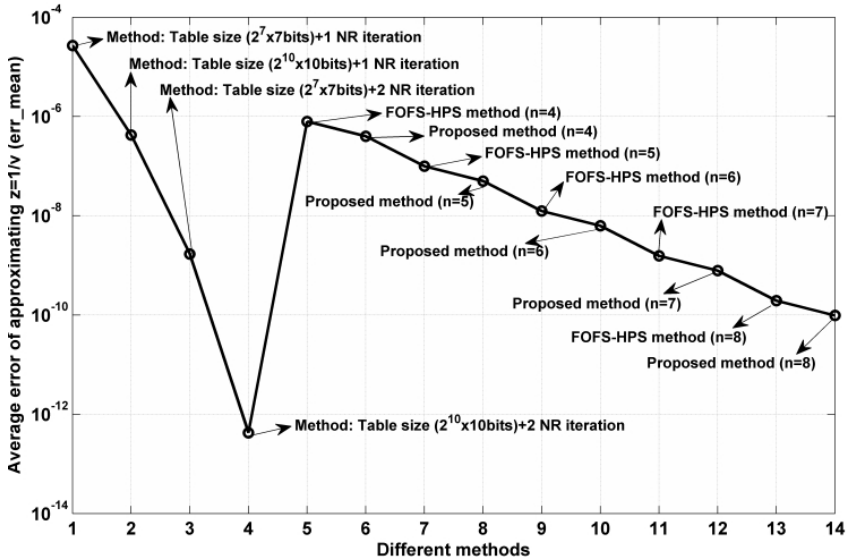

Fig. 7. Average Error of the proposed and FOFS-HPS method with different intervals, and NR method with different iteration and table size.

It should be pointed out that the error analysis in Fig. 6 and Fig. 7 is based on software simulation result. In hardware architecture, there will be precision loss considering the size of the memory and the width of the input, output and internal signal. This will cause additional truncated error which is often related to specific application.

B. Hardware Precision Analysis Based on Table Size

In hardware architecture, the truncated error caused by the truncation of different coefficients ($l_{2,i}$, $j_{2,i}$, $c_{2,i}$, $p_{2,i}$, $m_{2,i}$ and $k_{2,i}$) can influence the precision of hardware.

Let the value l_H, j_H, c_H, p_H, m_H and k_H denotes the implemented hardware value (determined by the width of table size) of $l_{2,i}$, $j_{2,i}$, $c_{2,i}$, $p_{2,i}$, $m_{2,i}$ and $k_{2,i}$, respectively. Let the value l_ε, j_ε, c_ε, p_ε, m_ε and k_ε denotes the truncated error caused by the limited table width of $l_{2,i}$, $j_{2,i}$, $c_{2,i}$, $p_{2,i}$, $m_{2,i}$ and $k_{2,i}$, respectively. Then, the following function is met:

$$\begin{cases} l_{2,i} = l_H + l_\varepsilon, \\ j_{2,i} = j_H + j_\varepsilon, \\ c_{2,i} = c_H + c_\varepsilon, \\ p_{2,i} = p_H + p_\varepsilon, \\ m_{2,i} = m_H + m_\varepsilon, \\ k_{2,i} = k_H + k_\varepsilon. \end{cases} \quad (27)$$

In the FOFS-HPS and proposed SOFS-HPS method, the second sub-function can be expressed as

$$\begin{aligned} s_2(x) &= l_{2,i} + j_{2,i}x_w - c_{2,i}{x_w}^2 = \\ &= l_H + j_H x_w - c_H {x_w}^2 + \underbrace{l_\varepsilon + j_\varepsilon x_w - c_\varepsilon {x_w}^2}_{truncated \quad error}. \end{aligned} \quad (28)$$

It can be seen from (28) that increasing the size of the table (enlarge β_H will decrease β_ε, $\forall \beta \in \{l_{2,i}, j_{2,i}, c_{2,i}\}$) can directly decrease the truncated error.

In the proposed SOFS-HPS with squaring shrunk method, the second sub-function can be expressed as

$$\begin{aligned} s_2(x) &= p_{2,i}(x_w + m_{2,i})^2 + k_{2,i} = \\ &= (p_H + p_\varepsilon)(x_w + m_H + m_\varepsilon)^2 + k_H + k_\varepsilon = \\ &= p_H(x_w + m_H)^2 + 2p_H(x_w + m_H)m_\varepsilon + p_H m_\varepsilon^2 + \\ &+ p_\varepsilon(x_w + m_H)^2 + 2p_\varepsilon(x_w + m_H)m_\varepsilon + \\ &+ p_\varepsilon m_\varepsilon^2 + k_H + k_\varepsilon \approx \\ &\approx p_H(x_w + m_H)^2 + k_H + \\ &+ \underbrace{2p_H(x_w + m_H)m_\varepsilon + p_\varepsilon(x_w + m_H)^2 + k_\varepsilon}_{truncated \quad error}. \end{aligned} \quad (29)$$

High order of truncated error has been omitted in (29), where only first order of error has been concerned to simplify the analysis. It can be found in (29) that the table size of k_H has a linear proportion to decrease the truncated error while k_ε has the minimum influence on the precision. The other two variables have a conversed relationship on the truncated error, where p_ε has a more obvious influence on the precision than m_ε. It can be seen that the proposed method in (29) can effectively reduce the multipliers in hardware, but more memory bits will be consumed to store the more accurate coefficients so as to obtain the same precision as in (28).

Average error based on different table size of the proposed hardware architecture has been shown in Fig. 8, where there are 16 intervals and the table width of coefficient $m_{2,i}$ is 11 bits. It can be seen that the table width of $p_{2,i}$ plays more important role in affecting the average error than table width of $k_{2,i}$. It is consistent with the analysis of the truncated error in (29).

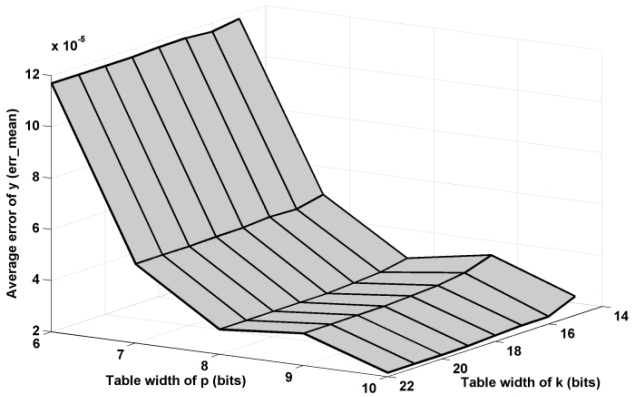

Fig. 8. Average Error of the proposed (SOFS-HPS with squaring shrunk method) hardware architecture $(n = 4)$ with different table width of coefficient $p_{2,i}$ and $k_{2,i}$ (table width of $m_{2,i}$ = 11 bits).

C. Implementation Result of the Proposed SOFS-HPS Method

HPS with 16, 32 and 64 intervals on the second sub-function is implemented. FOFS-HPS and SOFS-HPS method are compared using FPGA. The implemented FOFS-HPS architecture is the same as in [16]. The target device is Cyclone V 5CSEMA5F31C6, and the verification platform is DE1-SoC. Quartus Prime 16.0 and ModelSim 6.2b is chosen as the synthesis and simulation tool. The width of input is 15 bits and the output is 16 bits, implementation of multiplier uses intellectual property (IP) provided by Quartus.

Implementation results are shown in Table III. All implementations are using full pipelined structures, which lead to computed data refreshed every clock (the value of throughput equals maximum speed). It can be seen that the proposed method reduces the memory bits by 53 %, and improves the precision by 32 % on the average. The improved precision and memory bit has a direct proportion to the intervals. Although extra adaptive logic modules (ALMs) are consumed in the proposed architecture because of the additional usage of addition and multiplexer, it can be found that the proposed SOFS-HPS method is more effective when more intervals are used. Above all, the proposed SOFS-HPS method is a hardware-efficient architecture by taking advantage of symmetric property in the first help function. It brings performance improvement in terms of precision and

memory bits at the cost of two adders and one multiplier over FOFS-HPS method.

TABLE III. IMPLEMENTATION COMPARISON OF DIFFERENT HPS METHOD.

Intervals	Method	ALMs	Mem. (bits)	DSP	Freq. (MHz)	Mean Error $\times 10^{-6}$
n = 4	FOFS-HPS	90	576	3	285.71	8.489
	Proposed SOFS-HPS	120	252	4	284.01	4.601
n = 5	FOFS-HPS	95	960	2	267.02	8.123
	Proposed SOFS-HPS	126	442	3	290.95	5.326
n = 6	FOFS-HPS	52	1728	3	290.28	8.650
	Proposed SOFS-HPS	114	825	3	288.68	7.146

D. Implementation Result of the Proposed SOFS-HPS with Squaring Shrunk Method

In order to validate the hardware efficiency of the proposed SOFS-HPS with squaring shrunk method, implementation of NR method with single iteration and table size of $2^7 \times 7$ bits (M1), NR method with single iteration and table size of $2^{10} \times 10$ bits (M2), traditional FOFS-HPS method (M3) [14], proposed SOFS-HPS method (M4) and SOFS-HPS with squaring shrunk method (M5) based on 16 intervals $(n = 4)$ have been implemented and compared using FPGA. The verification platform and software tools are the same as in section C. Stratix IV EP4SGX230FF35C4 FPGA is chosen as a comparison target device as well, where adaptive look-up table (ALUT) is the basic element of resources. The implemented hardware architecture is shown in Fig. 1, Fig. 2, Fig. 4 and Fig. 5, respectively. Full pipelined structures are implemented either.

TABLE IV. IMPLEMENTATION RESULT OF DIFFERENT APPROXIMATING METHODS.

Method	FPGA	ALMs/ ALUTs	Mem. (bits)	DSP	Freq. (MHz)	Mean Error $\times 10^{-5}$
M1	Cyclone V	58	896	1	271.96	1.01
	Stratix IV	59	896	2	284.82	
M2	Cyclone V	76	10240	1	269.98	0.39
	Stratix IV	96	10240	2	292.91	
M3	Cyclone V	88	464	3	282.73	1.33
	Stratix IV	176	464	5	311.43	
M4	Cyclone V	128	189	3	278.86	1.32
	Stratix IV	236	189	6	298.69	
M5	Cyclone V	207	216	2	277.09	1.24
	Stratix IV	395	216	4	310.37	

Implementation result of different approximating method in computing $z = 1/v$ is shown in Table IV, where 16 intervals are considered. It can be seen that the proposed SOFS-HPS with squaring shrunk method consumes the least number of DSP blocks and moderate memory bits (Mem.) at the cost of extra resources compared to FOFS-HPS and SOFS-HPS method. The extra ALMs/ALUTs which are considered less expensive than DSP is resulted from the squaring shrunk approach. Although NR method reaches a low complexity in terms of logic element and DSP blocks, it is due to the mass usage of memory bits. Nevertheless, the proposed SOFS-HPS with squaring shrunk method can lead to relatively low memory consumption.

Error of the implemented hardware for different method has been shown in Fig. 9, where the standard deviation of M1, M2, M3, M4 and M5 is 1.07×10^{-5}, 2.82×10^{-6}, 1.22×10^{-5}, 1.33×10^{-5} and 6.58×10^{-6}, respectively. It can be seen from Table IV and Fig. 9 that HPS method has outperformed NR method in terms of speed and memory usage at the cost of extra logic elements and DSP blocks. It also shows that the proposed M5 method has the most robust error distribution among them.

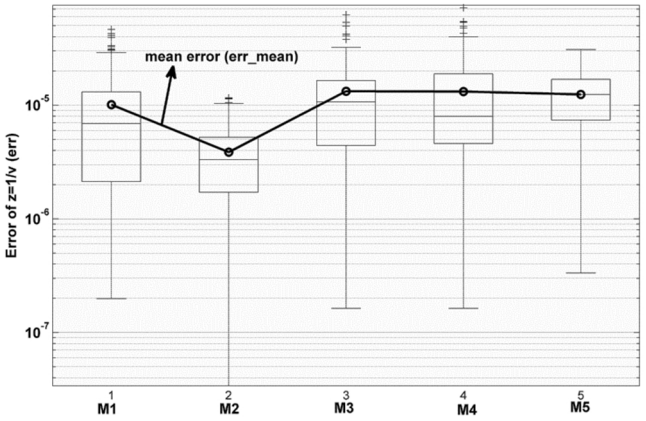

Fig. 9. Hardware error of different method when input variable range is (1, 2).

From mathematical analysis, precision simulation and hardware implementation, it can be found that the proposed SOFS-HPS with squaring shrunk method devotes a hardware-efficient architecture by exploiting second order first sub-function, symmetric property and squaring shrunk technology. It obtains a more robust error distribution compared to existing FOFS-HPS. It also eliminates the expensive multipliers without precision loss. Although more logic elements are consumed to obtain the same precision as FOFS-HPS, it can be inferred that (from Fig. 8) the influence of table width for different methods will be negligible when the accuracy is improved. The proposed method can be easily implemented in low cost FPGAs for acceleration purpose or be integrated into complex circuit design where memory and multiplier is difficult or expensive to implement.

IV. CONCLUSIONS

Second order first sub-function of harmonized parabolic synthesis with squaring shrunk method has been proposed in this paper to approximate reciprocal. The use of high order first sub-function can benefit the accuracy of reciprocal, and reduce the memory usage by exploiting symmetric coefficients. Squaring shrunk method can be utilized to

release the burden of multipliers. These techniques have been combined to yield an efficient hardware reciprocal, which has been shown performance improvement over traditional harmonized parabolic synthesis in terms of precision, memory and multiplier. Mathematical derivation, simulation and hardware implementation have been presented to demonstrate the efficiency of the proposed method, which shows a promising solution suitable for hardware acceleration and FPGA design for wireless communication, matrix inversion, image processing, and *et al*.

REFERENCES

[1] 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. DOI: 10.1109/CISP.2013.6743941.

[2] D. S. Debjit, D. W. Matula, "Faithful bipartite ROM reciprocal tables", in *Proc. 12th Symp. Comput. Arith.*, Bath, 1995, pp. 17–28. DOI: 10.1109/ARITH.1995.465381.

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

[4] T. J. Ypma, "Historical development of the Newton-Raphson method", *SIAM Rev.*, vol. 37, no. 4, pp. 531–551, 1995. DOI: 10.1137/1037125.

[5] J. Luo, Q. Huang, H. Luo, Y. Zhi, X. Wang, "Hardware implementation of single iterated multiplicative inverse square root", *Elektronika ir Elektrotechnika*, vol. 23, no. 4, pp. 18–23, 2017. DOI: 10.5755/j01.eie.23.4.18717.

[6] 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. DOI: 10.1587/elex.10.20120879.

[7] 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. DOI: 10.1109/12.863031.

[8] E. Hertz, P. Nilsson, "A methodology for parabolic synthesis of unary functions for hardware implementation", in *Proc. 2nd Int. Conf. on Signals, Circuits & Systems*, Tunisia, 2008, pp. 30–35. DOI: 10.1109/ICSCS.2008.4746866.

[9] E. Hertz, P. Nilsson, *A methodology for parabolic synthesis*. In-Tech, Vienna, 2010.

[10] E. Hertz, P. Nilsson, "Parabolic synthesis methodology implemented on the sine function", in *Int. Symp. on Circuits & Systems*, Taipei, 2009, pp. 253–256. DOI: 10.1109/ISCAS.2009. 5117733.

[11] A. M. Hashmi, "Parabolic synthesis and non-linear interpolation", M.S. thesis, Dept. Elect. Info. Tech., Lund Univ., Lund, Sweden, 2015.

[12] P. Pouyan, E. Hertz, P. Nilsson, "A VLSI implementation of logarithmic and exponential functions using a novel parabolic synthesis methodology compared to the CORDIC algorithm", *in Euro. Conf. On Circuit Theory and Design*, Linkoping, 2011, pp. 709–712. DOI: 10.1109/ECCTD.2011.6043642.

[13] E. Hertz, "Methodologies for approximation of unary functions and their implemention in hardware", Ph.D, thesis, Halmstad Univ., Halmstad, Sweden, 2016.

[14] S. Savas, E. Hertz, T. Nordstrom, Z. Ul-Abdin, "Efficient single-precision floating-point division using harmonized parabolic synthesis", *IEEE Comput. Soc. Annu. Symp. on VLSI*, Bochum, 2017, pp. 110–115. DOI: 10.1109/ISVLSI. 2017.28.

[15] E. Hertz, B. Svensson, P. Nilsson, "Combining the parabolic synthesis methodology with second-degree interpolation", *Microprocessors and Microsystems,* vol. 42, pp. 142–155, 2016. DOI: 10.1016/j.micpro.2016.01.015.

[16] J. Chen, J. Shi, "Hardware implementation of number inversion function", M.S. thesis, Dept. Elect. Info. Tech., Lund Univ., Lund, Sweden, 2016.

[17] 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. DOI: 10.1587/elex.11. 20140562.

[18] N. Thuning, T. 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.