J Sign Process Syst (2018) 90:1623–1637
https://doi.org/10.1007/s11265-017-1300-4

The Harmonized Parabolic Synthesis Methodology for Hardware Efficient Function Generation with Full Error Control

Erik Hertz[1] · **Jingou Lai**[2] · **Bertil Svensson**[1] · **Peter Nilsson**[2]

Received: 23 November 2016 / Revised: 4 October 2017 / Accepted: 15 October 2017 / Published online: 26 October 2017

Abstract The Harmonized Parabolic Synthesis methodology is a further development of the Parabolic Synthesis methodology for approximation of unary functions such as trigonometric functions, logarithms and the square root with moderate accuracy for ASIC implementation. These functions are extensively used in computer graphics, communication systems and many other application areas. For these high-speed applications, software solutions are not sufficient, and a hardware implementation is therefore needed. The Harmonized Parabolic Synthesis methodology has two outstanding advantages: it is parallel, thus reducing the execution time, and it is based on low complexity operations, thus being simple to implement in hardware. A difference compared to other approximation methodologies is that it is a multiplicative, and not additive, methodology. Compared to the Parabolic Synthesis methodologies it is possible to significantly enhance the performance in terms of reducing chip area, computation delay and power consumption. Furthermore, it increases the possibility to tailor the characteristics of the error, improving conditions for subsequent calculations. To evaluate the methodology, the fractional part of the logarithm is implemented and its performance is compared to the Parabolic Synthesis methodology. The comparison is made with 15-bit resolution. The design implemented using the proposed methodology performs 3× better than the Parabolic Synthesis implementation in terms of throughput, while consuming 90% less energy. The chip area is 70% smaller than for the Parabolic Synthesis methodology. In summary, the new technology further increases the advantages of Parabolic Synthesis.

Keywords Approximation · Parabolic synthesis · Unary functions · Elementary functions · Second-degree interpolation · Arithmetic computation · Look-up table · VLSI

1 Introduction

Computation of elementary functions, such as trigonometric functions, logarithms and the square root, as well as binary functions, such as division, are numerous in a multitude of applications. A trend in wireless communication systems is handheld applications with very high data rates and moderate accuracy in the computations for ASIC implementation. With such applications follows also requirements on small chip area and low power consumption. Such an emerging application is wireless communication systems with multiple antennas on the transmitter and receiver, known as Multiple-Input Multiple-Output (MIMO). The great interest in MIMO falls back on its ability to cost-effectively improve transmission performance. To accomplish the high data rates these systems are performing an extensive amount of computation. An essential part of the computation is spent on matrix inversions, which are often executed as QR decompositions in which very high throughput is needed. An example of the required levels of performance for a QR decomposition is described in [1] for an Ordered Successive Interference Cancellation (OSIC) detector. The computation performance needed in the OSIC detector is 1.56 million inversions of complex-valued 4 × 4 channel matrices per second. Since each inversion uses 12 computations of trigonometric functions, 12 × 1,560,000 = 18.72 million computations per second of the trigonometric functions sine and

✉ Erik Hertz
erik.hertz@hh.se

[1] Centre for Research on Embedded Systems, Halmstad University, Halmstad, Sweden

[2] Department of Electrical and Information Technology, Lund University, Lund, Sweden

cosine have to be executed. For these numerically intensive real-time applications, software routines - although they are capable of providing extremely accurate results - are too slow. Therefore, for the future of high-speed wireless communication, there is a significant interest for new development regarding hardware implementations of function generators.

In ASIC implementations of function generators, the control of the error (i.e., of its distribution and characteristics) is a neglected area. Full control of these parameters is indeed necessary in order to reach optimal performance in terms of small chip area, short critical path delay and low power consumption. Optimizing the word lengths and achieving a normally distributed error in the result from an approximation will enable reduced chip area, critical path delay and power consumption – not only in the circuitry of the approximation itself but, even more so, in the rest of the algorithm, which uses the approximation as input. The main reason for this is that it will reduce the effects of the accumulated error in the following parts, which otherwise would have led to a necessary increase in word lengths in the computations to handle this effect.

Hardware computation of elementary functions can be performed by employing many different algorithms [2, 3], such as table-based methods, polynomial and rational approximations and functional iteration methods. Table-based methods remain manageable for low precision computations as long as the input operand is up to 12–16 bits, corresponding to table sizes of 4 K–64 K words. The size of the table grows exponentially with the word length and becomes unacceptably large when operating with higher precision on an ASIC. An alternative way of making approximations is based on polynomials. Since polynomials only involve additions, subtractions and multiplications, using them is a natural way to approximate elementary functions. There are a number of polynomial schemes available for polynomial approximations, such as Taylor, Maclaurin, Legendre, Chebyshev, Jacobi, and Laguerre. For a given precision, the chosen polynomial scheme affects the number of terms included and thus the computational complexity. Two strategies are available in developing an approximation, one to minimize the average error, called least squares approximation, and one to minimize the worst case error, called least maximum approximation [2]. An example of when least squares approximation is favorable is when the approximation is used in a series of computations. On the other hand, least maximum approximation is favorable when it is important that the maximum error to the function to be approximated is kept small. An example of when least maximum approximation is favorable is when the error from the approximation has to be within a limit from the true function value. An advantage of polynomials is that they are table-less, but their drawback is that they impose large computational complexities and delays [2]. A reduction in computational complexity - and to some extent also in delay - can be accomplished by combining table-based methods with polynomial based ones [2].

For implementation of elementary functions in hardware, the approximation methodology of sum of bit-products [4] can be beneficial since it can give an area efficient implementation with a high throughput at a reasonable accuracy.

The commonly used COordinate Rotation DIgital Computer (CORDIC) algorithm [5, 6] is an iterative algorithm. The benefit of the algorithm is that approximations of basic elementary functions only require small look-up tables, simple shifts and addition operations. However, since it is an iterative method, it is inherently slow and therefore insufficient for very high performance applications. As shown in [[7], p. 125], CORDIC requires several extra bits of accuracy in order to cope with the error in the approximation.

The Parabolic Synthesis methodology [8–11] is founded on a multiplicative synthesis of factors, each of them a second-order function. The more factors that are used, the higher the accuracy of the approximation is. In fact, the most fundamental difference in the Parabolic Synthesis methodology compared to many other approximation methodologies, like polynomial and CORDIC, is that it is a multiplicative and not an additive methodology. With the introduction of the Parabolic Synthesis methodology, the following improvements were accomplished compared to CORDIC. First, due to a highly parallel architecture, a significant reduction of the propagation delay was achieved, which also led to a significant reduction of the power consumption. Second, the Parabolic Synthesis methodology allows full control of the characteristics and distribution of the error, which opens an opportunity to design with shorter word lengths and thereby gain in area, speed and power. As reported in [12], a further improvement of the Parabolic Synthesis methodology was developed by combining it with second-degree interpolation [3, 13, 14].

When developing a multiplicative approximation based on such a combination of the methodologies, the first factor is computed using the Parabolic Synthesis methodology. It is a rough approximation of the target function. The second factor is computed using a Second-Degree Interpolation methodology where the number of intervals in the interpolation decides the accuracy of the approximation. The approximation can be looked upon as a synthesis of two second-degree polynomials with a different variable in each polynomial. In the first factor, the variable is over the total interval, whereas in the second factor the variable is over a sub interval. As a summary, the approximation is computed as a polynomial with two variables and with the total degree of 2 + 2 = 4. An attractive feature with the Parabolic Synthesis methodology combined with Second-Degree Interpolation [12] is that it enables a rough internal error adjustment as part of the approximation. The error compensation can be performed in order to improve the distribution of the error to suit the properties required of the approximation. Compared to the Parabolic Synthesis methodology the Parabolic Synthesis methodology Combined with Second-Degree Interpolation accomplishes

the following [12]: A reduction in chip area because the number of second-order functions is reduced to two; a reduction of the critical path delays because the number of second-order functions is reduced to two; a reduction in power consumption because of reduced chip area and critical path delay. And last, the improved possibilities to tailor the distribution of the error.

This paper proposes an extension of the Parabolic Synthesis methodology Combined with Second-Degree Interpolation. This extension is named the Harmonized Parabolic Synthesis methodology because the two parts in this new methodology are developed as a unity in order to improve and optimize performance such as chip area, critical path delay and power consumption. The new methodology allows the exclusion of up to two additions and two multiplications in the algorithm, which is a considerable improvement. Another advantage with the Harmonized Parabolic Synthesis methodology is that it does not have the limitations of the Parabolic Synthesis methodology Combined with Second-Degree Interpolation when it comes to implementing roots, division and inverse roots.

The feasibility of Parabolic Synthesis has been verified by implementing a vast range of unary functions, as shown in [10]. With the Harmonized Parabolic Synthesis the boundary conditions for the functions to be approximated are relaxed, which implies that more extreme functions also can be carried out.

A dilemma that appears when comparing different approximation methods is that such comparisons need to be done in the same context in order to be conclusive. The Parabolic Synthesis methodologies allow full control of the characteristics and distribution of the error, something that is very seldom the case in traditional approximation methods. Using accuracy as the only comparison criterion is not enough; other characteristics of the error, most notably its distribution, are equally important. An illustrative example is the comparison in [12] between an implementation based on one of the Parabolic Synthesis methodologies and an implementation performed with the CORDIC method (which, by far, is the most commonly used approximation method today). It is shown that the implementation performed using Parabolic Synthesis combined with second-degree interpolation outperforms the CORDIC implementation in all aspects. In addition, as shown in [7, p. 125], CORDIC requires several extra bits of accuracy in order to cope with the error in the approximation; this will further degrade the performance of the CORDIC implementation.

The remaining part of this paper is organized as follows: Section 2 describes the Harmonized Parabolic Synthesis methodology; Section 3 describes the general structure of the hardware architecture resulting from the methodology; Section 4 proposes a general strategy for truncation as well as optimization, which, if combined, can have a positive impact on the characteristic of the error; Section 5 presents the implementation of the logarithm in Harmonized Parabolic Synthesis and the distribution of its error; Section 6 gives a comparison of implementations performed in the methodologies Parabolic Synthesis and Harmonized Parabolic Synthesis. The comparison is made with respect to chip area, critical path delay and power consumption; and Section 7 closes the paper with conclusions.

2 Harmonized Parabolic Synthesis

The Harmonized Parabolic Synthesis methodology is founded on two factors, both in the form of second-order parabolic functions, called the first sub-function, $s_1(x)$, and the second sub-function, $s_2(x)$. When recombined, as shown in (1), they equal the original function $f_{org}(x)$.

$$f_{org}(x) = s_1(x) \cdot s_2(x) \tag{1}$$

In (1) the first sub-function, $s_1(x)$, is a second-order parabolic function, which in conjunction with the second sub-function, $s_2(x)$, develops an approximation of the original function, $f_{org}(x)$. The second sub-function is a second-degree interpolation [3, 13, 14] specifically shown in [12], where the number of intervals in the interpolation decides the final accuracy of the approximation and allows the distribution of the error to be tailored. When developing an approximation with the proposed methodology, an empirical design methodology is the only feasible approach since the sub-functions are designed as a complete unity to fulfill the design criteria.

2.1 Requirements on the Original Function

To facilitate the development it is required that the function to be approximated is being normalized. When normalized the function has to satisfy the requirement that the values are in the interval $0 \le x < 1.0$ on the x-axis and $0 \le y < 1.0$ on the y-axis as well as have the starting point at (0,0), as illustrated in Fig. 1. The normalization of the function to be approximated creates the original function, $f_{org}(x)$.

Finally, the original function, $f_{org}(x)$, divided by the first sub-function, $s_1(x)$, must have a limit value when x goes towards 0. Otherwise the function to be developed, the second sub-function, $s_2(x)$, has no starting point when $x = 0$ and therefore cannot be developed.

Compared to Parabolic Synthesis, the number of criteria on the function to perform an approximation on has decreased from three to one. The two criteria that have been excluded have instead been transformed into recommendations. Thereby, the number of functions that can be approximated with the methodology is increased. The third criterion which restricts the limit of $f_{org}(x)$ divided by $s_1(x)$ when x goes towards 0 is eliminated because the Harmonized Parabolic Synthesis methodology can handle these limits.

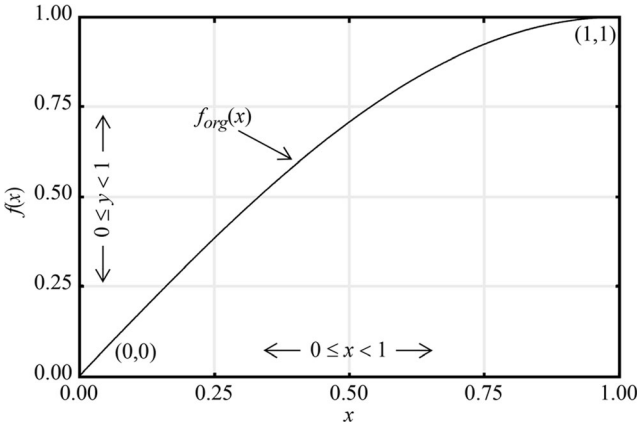

Figure 1 The conditions for the original function.

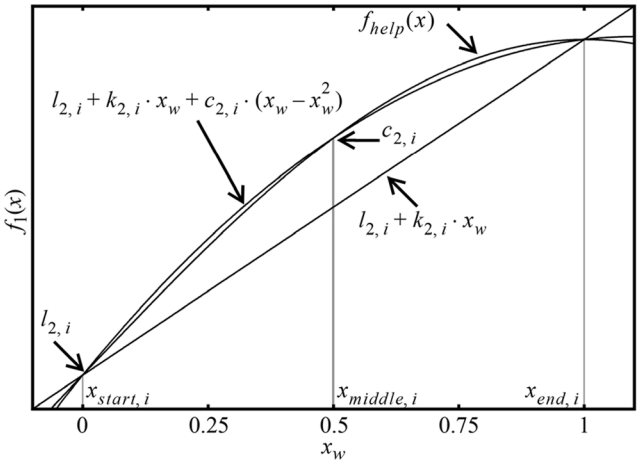

Figure 2 Description of the development of the second-degree interpolation.

2.2 The First Sub-Function

The first sub-function, $s_1(x)$, is a second-order parabolic function as shown in (2).

$$s_1(x) = l_1 + k_1 \cdot x + c_1 \cdot \left(x - x^2\right) \tag{2}$$

It can be seen that the two leftmost terms form a linear function with a constant l_1 and a gradient k_1, while the rightmost term is the nonlinear part. As described above, the starting point for the first sub-function, $s_1(x)$, is in (0,0). This gives the start value, l_1, to be 0. Furthermore, the gradient, k_1, is 1 since the function starts in (0,0) and ends in (1,1). The first sub-function, $s_1(x)$, can therefore be rewritten according to (3).

$$s_1(x) = x + c_1 \cdot \left(x - x^2\right) \tag{3}$$

2.3 The Second Sub-Function

The second sub-function, $s_2(x)$, is based on splitting the function in intervals to perform an interpolation in each of them. The procedure when developing the second sub-function is to perform a division of the original function, $f_{org}(x)$, with the first sub-function, $s_1(x)$. This division generates the help function, $f_{help}(x)$, as shown in (4).

$$f_{help}(x) = \frac{f_{org}(x)}{s_1(x)} \tag{4}$$

From the help function, $f_{help}(x)$, the second sub-function, $s_2(x)$, is developed as a second-degree interpolation, as shown in (5) and Fig. 2, where the number of intervals in the interpolation decides the order of the accuracy of the approximation.

$$s_2(x) = l_{2,i} + k_{2,i} \cdot x_w + c_{2,i} \cdot \left(x_w - x_w^2\right) \tag{5}$$

To simplify the hardware representation of the interval, i, in hardware, the number of equal-range intervals in the second sub-function, I, is chosen as 2 to the power of w, where w is a natural number, as shown in (6).

$$I = 2^w \tag{6}$$

To simplify the normalization of the interval of x_w, it is selected as an exponential scaling by 2 of x where the integer part is removed. The normalization of x is therefore made by multiplying x with 2^w. Furthermore, the integer part is dropped, which gives x_w as a fractional part of x, as shown in (7).

$$x_w = frac(2^w \cdot x) \tag{7}$$

In hardware the normalization, x_w, is simply performed as a truncation of the w most significant bits of x. This truncation performs normalization to the interval, as shown in Fig. 2. The dropped integer part from the normalization is used to decode the interval in which the second sub-function is performed and is therefore synonymous with the index i in the sub-function, as shown in Fig. 2.

The index i is defined as the number of the interval of the interpolation, starting with 0 and ending with $I - 1$. In (5), $l_{2,i}$ is the starting point of an interval of the interpolation. This is computed by inserting the value of x for the starting point of the interval, $x_{start,i}$, of the help function, $f_{help}(x)$, (4) as shown in (8) and Fig. 2.

$$l_{2,i} = f_{help}\left(x_{start,i}\right) \tag{8}$$

Eq. (8) does not apply on the start value of the first interval, which has to be calculated as the limit according to (9). Since the x^2 term in (9) goes faster towards 0 than the x term, it can be excluded when calculating the limit, as shown in (9).

$$l_{2,0} = \lim_{x\to 0} \frac{f_{org}(x)}{x + c_1 \cdot (x - x^2)} = \lim_{x\to 0} \frac{f_{org}(x)}{(1 + c_1)\cdot x} \tag{9}$$

In (5), $k_{2,i}$ is the gradient for an interpolation interval i. The gradient $k_{2,i}$ for an interval is computed as the end point value of the help function, $f_{help}(x_{end,i})$, subtracted with the start point value of the help function, $f_{help}(x_{start,i})$. Since the interval is normalized to 1 the denominator when computing the gradient, $k_{2,i}$, is set to 1, and therefore no division is needed, as shown in (10) and Fig. 2.

$$k_{2,i} = f_{help}(x_{end,i}) - f_{help}(x_{start,i}) \tag{10}$$

In (5) the coefficient, $c_{2,i}$, in an interval, i, of the second sub-function, $s_2(x)$, is pre-computed so that the second sub-function in an interval, $s_{2,i}(x_w)$, cuts the help function, $f_{help}(x)$, in the middle of the interval when $x_w = 0.5$. This satisfies the point $x_{middle,i}$ for the help function, $f_{help}(x)$, as shown in (11) and Fig. 2.

$$c_{2,i} = 4 \cdot \left(f_{help}(x_{middle,i}) - l_{2,i} - k_{2,i} \cdot 0.5 \right) \tag{11}$$

The sub-function in (5) can be simplified according to (12).

$$s_2(x) = l_{2,i} + j_{2,i} \cdot x_w - c_{2,i} \cdot x_w^2 \tag{12}$$

In (12), $j_{2,i}$ is pre-determined according to (13).

$$j_{2,i} = k_{2,i} + c_{2,i} \tag{13}$$

In Eq. (14), it is shown how the sub-function, $s_{2,i}(x)$, is divided into partial functions.

$$s_2(x) = \begin{cases} s_{2,0}(x_w), & 0 \le x < \frac{1}{2^w} \\ s_{2,1}(x_w), & \frac{1}{2^w} \le x < \frac{2}{2^w} \\ \dots & \dots \\ s_{2,I-1}(x_w), & \frac{I-1}{2^w} \le x < 1 \end{cases} \tag{14}$$

Note that, in (14), x has been changed to x_w. The change is because the intervals for the partial sub-functions, $s_{2,i}(x)$, in (14) have equal ranges.

2.4 Simultaneous Development of the two Sub-Functions

The foundation of the Harmonized Parabolic Synthesis methodology is to approach the development with a more holistic view. This is expressed in that the development of the two sub-functions is made simultaneously. When developing an approximation of an original function, $f_{org}(x)$, the first and second sub-function are looked upon as one device. While, in the Parabolic Synthesis methodology, the first sub-function, $s_1(x)$, was developed to have as good conformity as possible with $f_{org}(x)$, the objective of the Harmonized Parabolic Synthesis methodology is rather to develop the first sub-function in such a way that the product of the two sub-functions gives a good conformity to the original function. This includes that the distribution of the error is to be favorable and the hardware implementation as simple as possible. The approach when developing the first sub-function, $s_1(x)$, can, in contrast to the Parabolic Synthesis methodology, not be based on independent analytical calculations since it is dependent on the performance of the second sub-function, $s_2(x)$. Therefore, the coefficient c_1 in the first sub-function, $s_1(x)$, has to be determined by, for different values of the coefficients, calculating the maximum absolute error, $f_{error}(x)$, between the approximation and the original function according to (15).

$$f_{error}(x) = \left| s_1(x) \cdot s_2(x) - f_{org}(x) \right| \tag{15}$$

When developing the second sub-function it is dependent on the first sub-function as shown in (4), since the second sub-function is developed from the help function, $f_{help}(x)$. As shown in (3), only the coefficient c_1 has to be determined when developing the first sub-function. To perform the calculation of the absolute error, $f_{error}(x)$, the second sub-function, $s_2(x)$, has therefore to be made dependent on the coefficient c_1 in the first sub-function, $s_1(x)$, as shown in (3) to (5) and (8) to (11). The calculation is interesting only as an indication of how the absolute error, $f_{error}(x)$, depends on the coefficient c_1. When choosing the coefficient c_1 it has to be made with regard to both the behavior of the error of the approximation and the efficiency of the hardware implementation. The number of intervals in the second sub-function, $s_2(x)$, needs to be increased to achieve the intended accuracy; this has also to be taken into account when performing the calculation of the error. As an example, Fig. 3 shows the bit accuracy for the sine function, shown when using 1, 2, 4 and 8 intervals in the second sub-function, $s_2(x)$, with different values of the coefficient, c_1. In our example implementation of the fractional part of the logarithm later in this paper, further details of how these calculations are done are given.

Based on Fig. 3, values of the coefficient c_1 are chosen to allow an efficient implementation of the hardware. As shown in (3), c_1 is fed into a multiplier, why choosing a value that is a

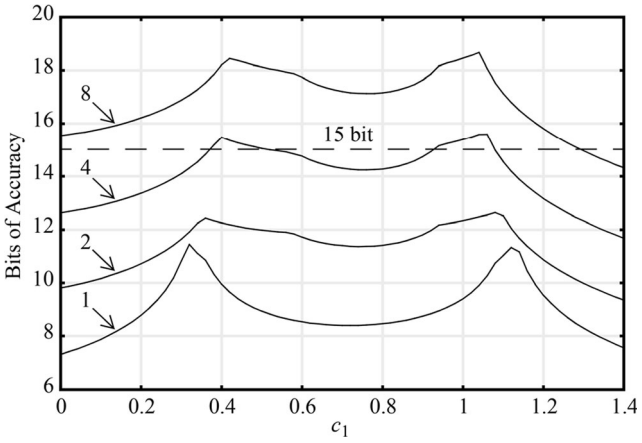

Figure 3 The bit accuracy depending on c_1 for 1, 2, 4 and 8 intervals in the second sub-function, $s_2(x)$.

power of two is desirable. As an example, shown in Fig. 3, the desired accuracy is 15 bit. To accomplish this, at least four intervals are necessary. As shown in Fig. 3, only $c_1 = 1.0$ for four intervals is interesting since it is a power of two. If increasing the number of intervals to eight, the coefficient c_1 can be chosen to zero. This is interesting since this will exclude the multiplication in (3). The behavior of the bit accuracy in Fig. 3 results from the approximation using different values of the coefficient c_1. From this, the number of intervals and the value on the coefficient c_1 used in the design can be selected.

3 Hardware Architecture

The hardware architecture resulting from the methodology can be divided into three parts as shown in Fig. 4, following a principle described by Tang [15]. In the preprocessing part, the incoming operand is transformed to fit the processing part, in which the approximation is performed. In the postprocessing part, the result is transformed to the desired output format.

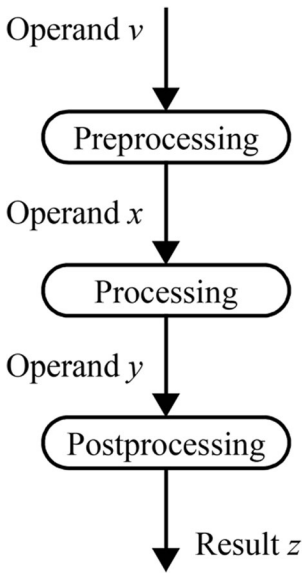

Figure 4 The three parts of the hardware architecture.

In most cases, preprocessing of the operand means normalization, but the transformation of the operand can also be more comprehensive such as converting a fixed-point number into a floating-point number. If the approximation is implemented as a block in a larger system, the preprocessing part can be integrated in the previous blocks, in which case the preprocessing part can be reduced or even excluded. For the postprocessing, similar conditions apply.

In the processing part, the approximation of the original function, $f_{org}(x)$, is directly computed in the way now described.

The architecture synthesized using Harmonized Parabolic Synthesis implements two sub-functions computed in parallel. The first sub-function is implemented as a second-order parabolic function and the second sub-function is implemented using the Second-Degree Interpolation methodology. Figure 5 shows the architecture, where the two sub-functions are implemented by the upper and lower half, respectively, and then combined via a multiplication.

It is worth noting that this, in fact, is a generic architecture, which can be used for approximating several different functions. The set of parameters defines the function.

In the first sub-function, the result of the $(x-x^2)$ part is multiplied with c_1 and then added to x. In the proposed methodology, the coefficient c_1 is chosen as described in conjunction with Fig. 3, to reduce the hardware consumption. This implies that the algorithm of the first sub-function, $s_1(x)$, can be simplified, which also reduces the complexity of the implementation. The second sub-function is implemented as a second-degree

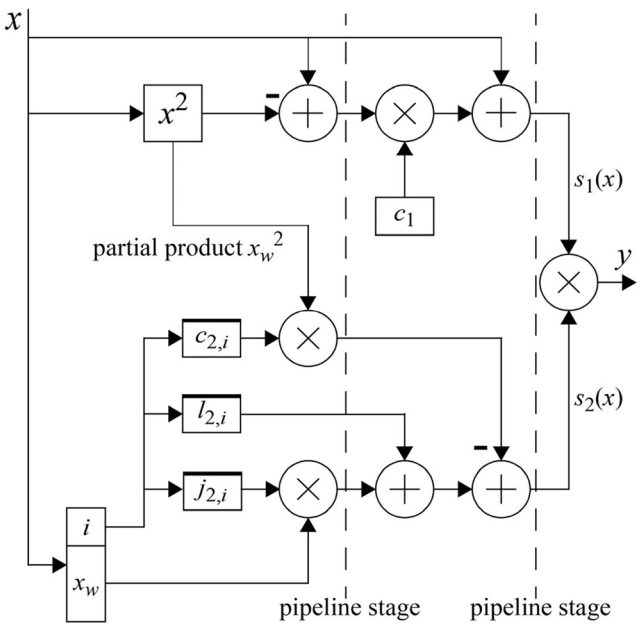

Figure 5 The hardware architecture of the design based on the Harmonized Parabolic Synthesis containing two sub-functions and with pipeline stages.

interpolation, consisting of a look-up table containing, for each interval i, the coefficients $j_{2,i}$ from (13). The coefficient $j_{2,i}$ is multiplied with x_w, which is the remaining part of x when the index part i is removed. The value of x_w is also the normalized value of x for the interval. After the multiplication, the start value of the interval $l_{2,i}$ is added. The third branch of the second sub-function consists of a look-up table containing the coefficients $c_{2,i}$ from (11) for each interval i. The coefficient $c_{2,i}$ for interval i is multiplied with ${x_w}^2$, which is the partial products of x^2 for interval i. To simplify this computation, a special squaring unit to compute x^2 has been developed; this unit is described in [8, 10]. The benefit of the squarer is that it computes all the partial products to x^2 in the same hardware and that it halves the chip area and computation time compared to a multiplier. The result of the multiplication between $c_{2,i}$ and ${x_w}^2$ is subtracted from the result of the previous addition. After the values of the first and second sub-functions are computed, they are multiplied with each other.

The architecture is suitable for pipelining in order to increase the throughput. An example of where pipeline stages can be introduced is shown in Fig. 5. Introduction of pipeline stages in this case is decided by the computation time of the rightmost multiplication, since this is the largest multiplier. To decrease the computation time further the next step will be to also insert pipeline stages in the multipliers. The introduction of data pipelining stages in the architectures will only have little effect on the size of the hardware, since the data paths are few. However, registers consume power, thus power consumption will increase.

4 Optimizing

The purpose with the optimization of the design is to reduce the chip area, the critical path delay and the power consumption, but also to gain full control over the error in the result of an approximation. The main factor that affects the parameters (chip area, critical path delay, and power consumption) is the data word lengths used in the computations. The main factor behind the need to increase the word lengths is that the errors accumulate through the computations. The characteristics and distribution of the error of the result of the approximation affect the word lengths in the computations in the remaining design. Thus it is important in the approximation to keep the word length of the result as short as possible – but not shorter – and see to that the error distribution is favorable for the remaining part of the algorithm. This can only be achieved if the designer gains full control over the error in the approximation and is able to tailor the characteristics of the error so that they improve the conditions for the subsequent calculations. Unlike many other approximation methodologies, the Parabolic Synthesis methodologies support such tailoring of the characteristics and distribution of the error. In other approximation methodologies, such as CORDIC, the tailoring of the characteristics and distribution of the error is not supported to any significant extent. The effects of the disadvantageous error performance of the CORDIC algorithm are commonly mitigated by increasing the accuracy, as shown in the analysis of error and datapath precision of CORDIC that is presented in [7, p. 125]. Of course, increasing the accuracy is synonymous with longer word lengths in the design, which has devastating effects on the hardware efficiency of the implementation.

Finding the optimal set of coefficients and word lengths is, in this stage of the development of the methodologies, an empirical, iterative procedure that is performed manually using MatLab. After deciding the initial coefficients in the first and second sub-functions, the next step is to start the optimization of the architecture. The method for the optimization is to decide the word lengths used in the architecture and then optimize the coefficients. This optimization is performed in an iterative trial and error manner and the evaluation of different coefficient values should be performed in parallel with the evaluation of the word lengths, since the truncation error effects influence the performance of calculations in the design. The strategy is to adjust coefficients and word lengths in the design for best accuracy and distribution of the error. The procedure, when done manually, takes roughly two hours; it can certainly be automated in the future – or at least supported by appropriate tools. In the text below, the optimization strategy will be illustrated using the sine function, since this function is commonly used and has a simple implementation, thus making the steps of the optimization easy to follow. The target accuracy for the implementations is chosen to be around 15 bit.

In practice, the simulation of the approximation is performed with a bit-accurate C model and the performance of the approximation is analyzed in MatLab.

4.1 Truncation and Optimization

Truncation always results in a negative offset of the error compared to the non-truncated value, as illustrated in Fig. 6. In the figure, the gray curve shows the error before the truncation and the black is the error after truncation.

Figure 6 shows the errors in both curves winding through the eight intervals. The winding of the curves is caused by the rightmost term in (5), which is the nonlinear part. This term in each interval in the second sub-function, $s_2(x)$, causes a single winding of the curve. To counteract the negative offset of the error caused by truncation, the coefficients in the second sub-function, $s_2(x)$, can be adjusted. A favorable way to do this is to seek a solution in which the error has a distribution that is asymptotically normal. An advantage of an asymptotically normal error distribution is that it is centered around zero and the most error values are close to zero. This distribution implies that the number range is optimally utilized which has

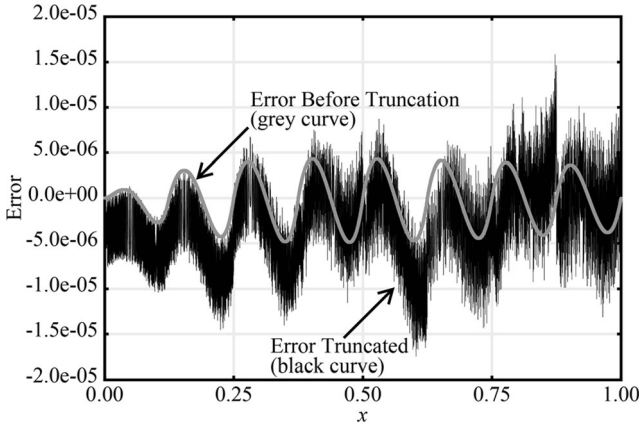

Figure 6 The error before and after truncation of the approximated value.

positive effects on the architecture, in reduced size of hardware and in the subsequent calculations to reduce their error. In Fig. 7 the gray curve shows the error before the truncation of the word lengths and the optimization of the coefficients, and the black is the error after truncation and optimization of the coefficients.

When comparing the two figures it can be seen that the error after truncation and optimization, i.e., the one in Fig. 7, is more evenly distributed around zero than the error in the one without optimization. This shows that it is mainly by adjusting the coefficients that the distribution of the error is determined.

4.2 The Characteristic Metrics

The error resulting from an approximation can be characterized in several ways [16], the most important being the following:

- The maximum absolute error denotes the largest error possible using the approximation.
- The mean error denotes the average error over the approximation's sample space.

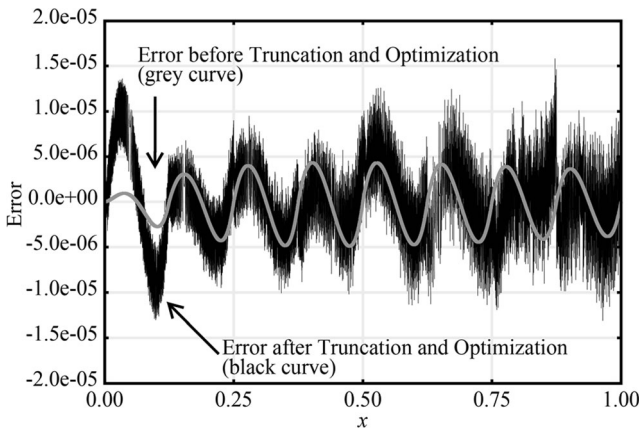

Figure 7 The error before and after truncation and optimization.

- The median error denotes the skewness in the error distribution, if there is any.
- The standard deviation is a measure of the variation from the mean error.
- The Root Mean Square (RMS) error is a measure of the magnitude of a continuously varying quantity of the error.

Furthermore, the evenness of the error distribution is determined by comparing the standard deviation with the RMS error. If the standard deviation and the RMS error are equal, it indicates that the error distribution is symmetric around zero. This is a preferred error distribution. An advantage of the methodology presented in this paper is that it obtains a very good possibility for developing a symmetrically distributed error around zero. To illustrate the distribution of the error a histogram is used, as shown in Fig. 8. Figure 8 shows the distribution of the error after truncation and optimization that was shown in Fig. 7. As shown, the error distribution is asymptotically normal and with a mean value near zero.

5 Implementation of the Fractional Part of the Logarithm

As an illustration of the Harmonized Parabolic Synthesis methodology, an implementation is presented of an algorithm that performs an approximation of the logarithm function. It is performed on a simple, nonstandard floating-point number format where the mantissa is in the range from 1.0 to 2.0. The approximation is only performed on the mantissa, since the exponential part of the input value is directly used as the integer part of the result and scaling the mantissa, [17, 18].

The implementation is in hardware using a 14 bit input. The target accuracy is 15 bits and the target distribution of the error is the normal distribution.

The performance of the approximation is analyzed as described in Section 4.

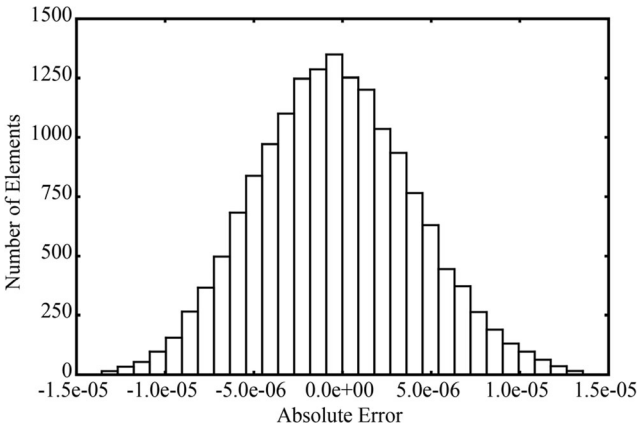

Figure 8 The error distribution after truncation and optimization of the approximation shown in Fig. 7.

5.1 Preprocessing

As described in Section 2.1, to facilitate the hardware implementation of the approximation, a normalization to satisfy that the values are in the interval $0 \le x < 1.0$ on the x-axis and $0 \le y < 1.0$ on the y-axis has to be performed. The result of the binary logarithm satisfies that the values are in the interval $0 \le y < 1.0$. As shown in Fig. 9, the operand v is within $1 \le v < 2$. To satisfy that the incoming operand x is within the interval $0 \le x < 1$, a 1 has to be added to the operand as shown in Fig. 9 and (16).

$$v = 1 + x \tag{16}$$

To normalize the $f(v) = \log_2(v)$ function, v is substituted according to (17), which gives the original function, $f_{org}(x)$, shown in (17).

$$f_{org}(x) = \log_2(1 + x) \tag{17}$$

Figure 9 shows the function, $f(v)$, together with the normalized function, $f_{org}(x)$.

5.2 Processing

For the processing part, the sub-functions are developed according to the description in Section 2.4. Developing the coefficient c_1 is made with two aspects in mind: the error distribution of the approximation, and the simplicity of the hardware implementation. The number of intervals that the second sub-function, $s_2(x)$, needs to be divided into, to achieve the intended accuracy, has also to be taken into account when performing the calculation of the error. Figure 10 shows the resulting minimum number of bits of accuracy for the logarithm function when using 1, 2, 4 and 8 intervals in the second sub-function, $s_2(x)$.

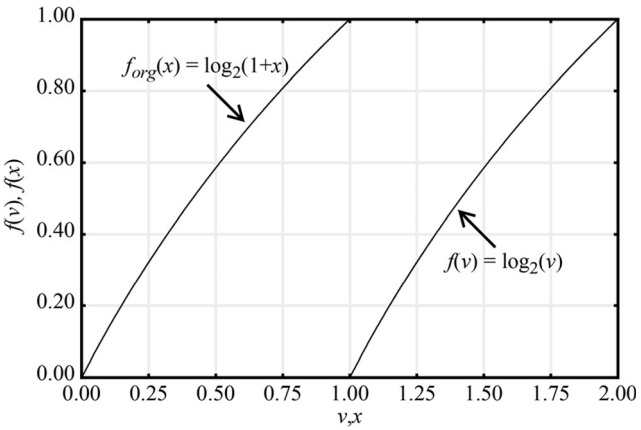

Figure 9 The function $f(v)$ before normalization and the normalized function, $f_{org}(x)$.

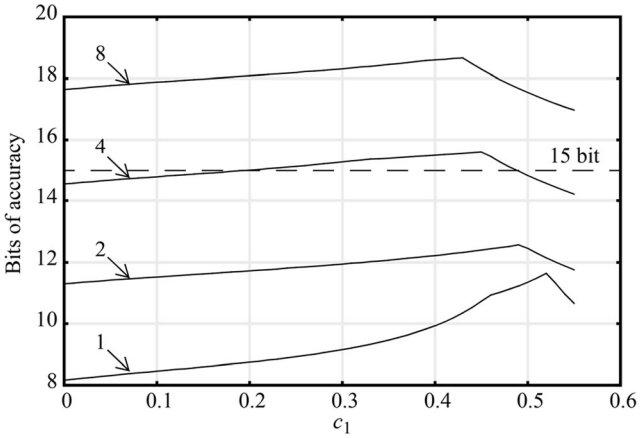

Figure 10 The bit accuracy depending on c_1 for 1, 2, 4 and 8 intervals in the second sub-function, $s_2(x)$, of the logarithm.

The needed accuracy for the implementation was set to 15 bits and the distribution of the error shall be similar to a normal distribution. Figure 10 shows that using 4 or 8 intervals will fulfill the accuracy demand with some specific ranges of coefficient, c_1. The requirement to have a distribution of the error that is similar to the normal distribution implies that using 8 intervals would be advantageous since this will give greater margin when developing the second sub-function, $s_2(x)$. Figure 10 shows that using 8 intervals in the second sub-function will allow the coefficient c_1 to be 0, which in turn will imply that the hardware in the first sub-function, $s_1(x)$, is reduced, as shown in (18).

$$s_1(x) = x + c_1 \cdot (x - x^2) = x + 0 \cdot (x - x^2) = x \tag{18}$$

Choosing the coefficient c_1 to be 0 will always be beneficial in terms of reduced chip area. With increasing number of intervals, the chip area and the length of the critical path will decrease to a certain point. First, chip area and later the critical path. A disadvantage with increasing the interval is that the distribution of the error will go towards the rectangular distribution since the second sub-function will end up as one look-up table. The choice of 8 intervals is made to illustrate the methodology, not to demonstrate the optimal solution.

The help function, $f_{help}(x)$, is computed as shown in (19).

$$f_{help}(x) = \frac{f_{org}(x)}{s_1(x)} = \frac{\log_2(1 + x)}{x} \tag{19}$$

Figure 11 shows the help function, $f_{help}(x)$.

From the help function, $f_{help}(x)$, an initial second sub-function, $s_2(x)$, is developed according to the description in Section 2.3 using 8 intervals.

When the initial second sub-function, $s_2(x)$, is developed, the next task is to achieve a distribution of the error similar to the normal distribution, which is performed as described in Section 4.

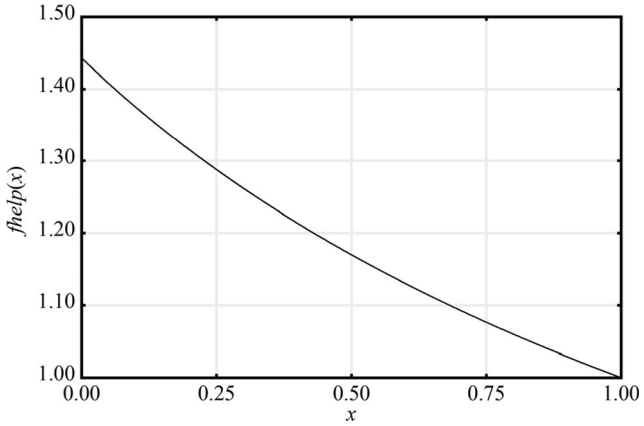

Figure 11 The help function, $f_{help}(x)$, for the approximation of the logarithm.

Tables 1, 2 and 3 show the developed coefficients in (12) for 8 intervals.

It can be seen from Table 2 that the three most significant bits after the binary point in the binary representation of the coefficients are zeros. As a result of this, the word length of the coefficients can be reduced from 15 bits to 12 bits.

It can be seen from Table 3 that the seven most significant bits after the binary point in the binary representation of the coefficients are zeros. As a result of this, the word length of the coefficients can be reduced from 16 bits to 9 bits.

When developing the algorithm for the approximation, there is an interaction between the word lengths and the values of the coefficients in the architecture. The word lengths of the coefficients and the data paths in the design are shown in Fig. 12.

In the process of truncating words in the design and optimizing the coefficients, the size and distribution of the error is analyzed and adjusted. Figure 13 shows the error of the final implementation compared to the error before the truncation of the word lengths in the design and optimization of the coefficients.

It can be seen that the error is fairly equal over the interval.

Figure 14 shows the absolute error of the implementation. It shows that the implemented approximation well meets the requirement of an accuracy of about 15 bits.

When optimizing the coefficients, the intention has been to achieve a normal distribution of the error. Such a distribution

Table 1 shows the coefficients for the initial value, $l_{2,i}$, for each interval, i, in (12).

Table 1: Coefficients $l_2 i$

Coefficient	Value
$l_{2,0}$	$1.44268798828125000_{dec}$
	$1.01110001010101000_{bin}$
$l_{2,1}$	$1.35937988818359375_{dec}$
	$1.01011100000000011_{bin}$
$l_{2,2}$	$1.28771209716796875_{dec}$
	$1.01001001101001111_{bin}$
$l_{2,3}$	$1.22514343261718750_{dec}$
	$1.00111001101000110_{bin}$
$l_{2,4}$	$1.16991424560546875_{dec}$
	$1.00101011011111111_{bin}$
$l_{2,5}$	$1.12069702148437500_{dec}$
	$1.00011110111001100_{bin}$
$l_{2,6}$	$1.07646942138671875_{dec}$
	$1.00010011100100111_{bin}$
$l_{2,7}$	$1.03644561767578125_{dec}$
	$1.00001001010101001_{bin}$

Table 2 shows the coefficients for the gradient, $j_{2,i}$, for each interval, i, in (12).

Table 2 Coefficient $j_{2,i}$

Coefficients	Value
$j_{2,0}$	-0.089294433593750_{dec}
	-0.000101101101110_{bin}
$j_{2,1}$	-0.076629638671875_{dec}
	-0.000100111001111_{bin}
$j_{2,2}$	-0.066589355468750_{dec}
	-0.000100010000110_{bin}
$j_{2,3}$	-0.058471679687500_{dec}
	-0.000011101111100_{bin}
$j_{2,4}$	-0.051849365234375_{dec}
	-0.000011010100011_{bin}
$j_{2,5}$	-0.046447753906250_{dec}
	-0.000010111110010_{bin}
$j_{2,6}$	-0.041900634765625_{dec}
	-0.000010101011101_{bin}
$j_{2,7}$	-0.038085937500000_{dec}
	-0.000010011100000_{bin}

Table 3 shows the coefficients for the parabolic part, $c_{2,i}$, for each interval, i, in (12).

Table 3 Coefficient $c_{2,i}$

Coefficient	Value
$c_{2,0}$	$-0.0060424804687500_{dec}$
	$-0.0000000110001100_{bin}$
$c_{2,1}$	$-0.0049438476562500_{dec}$
	$-0.0000000101000100_{bin}$
$c_{2,2}$	$-0.0040435791015625_{dec}$
	$-0.0000000100001001_{bin}$
$c_{2,3}$	$-0.0032501220703125_{dec}$
	$-0.0000000011010101_{bin}$
$c_{2,4}$	$-0.0026397705078125_{dec}$
	$-0.0000000010101101_{bin}$
$c_{2,5}$	$-0.0022277832031250_{dec}$
	$-0.0000000010010010_{bin}$
$c_{2,6}$	$-0.0018920898437500_{dec}$
	$-0.0000000001111100_{bin}$
$c_{2,7}$	$-0.0016479492187500_{dec}$
	$-0.0000000001101100_{bin}$

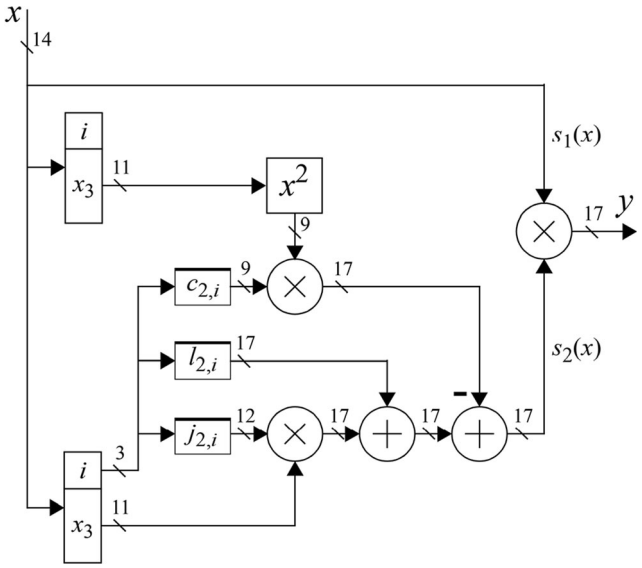

Figure 12 The hardware architecture of the implementation of the logarithm, with the word lengths.

is beneficial in the calculations in most algorithms. Figure 15 shows the distribution of the error in the final, optimized implementation.

The distribution in Fig. 15 shows a very high resemblance with the normal distribution.

The conclusion from Table 4 is that the maximum absolute error corresponds to an accuracy of more than 15 bits. The mean error is very small, less than 25 bits, and the median confirms that the error distribution is not skewed. When comparing the standard deviation value with the root mean square value, it can be seen that the values are nearly identical, which confirms that the error of the approximation is symmetrically distributed.

5.3 Postprocessing

No postprocessing is needed since the result from the processing part is in the right format.

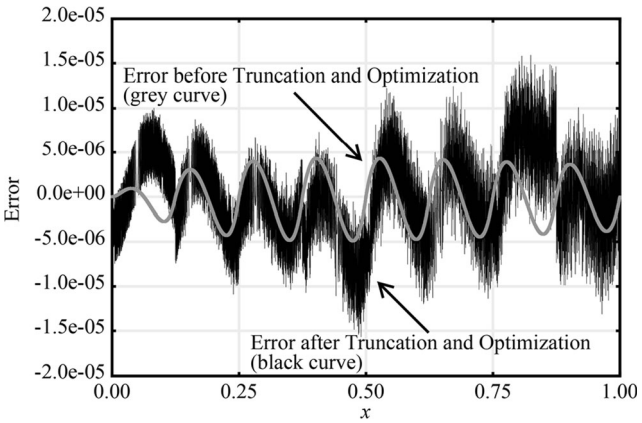

Figure 13 The error of the implementation of the logarithm before and after truncation and coefficient optimization.

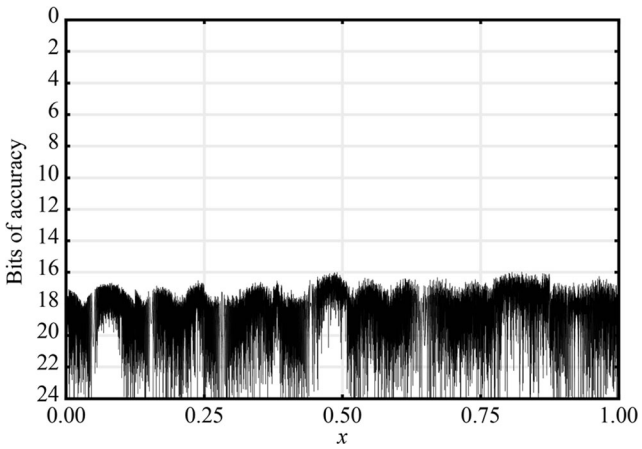

Figure 14 The absolute error in bits of the approximation of the logarithm.

6 Comparing Methodologies

This section compares implementations of the fractional part of the logarithm using two different methodologies; on the one hand the Parabolic Synthesis methodology, where the accuracy is decided with the number of sub-functions, and on the other hand, the Harmonized Parabolic Synthesis methodology, where the accuracy is decided with the number of intervals. The comparison will be performed in terms of chip area, critical path delay and power consumption. Both implementations are performed with the same bit accuracy and so that the error is asymptotically normally distributed.

The implementations use identical setups of the design tools and identical cell libraries. The implementations are made with the purpose of comparison, not to maximize performance; hence, potentially performance-increasing features like pipelining are not used.

The implementations are realized as ASICs with a 65 nm Standard-V_T (1.2 V) technology. Synopsys Design Compiler [19], Synopsys Primetime [20] and Mentor Graphics Modelsim [21] are used for all implementations. The estimation of the power consumption is done with the help of default switching activity.

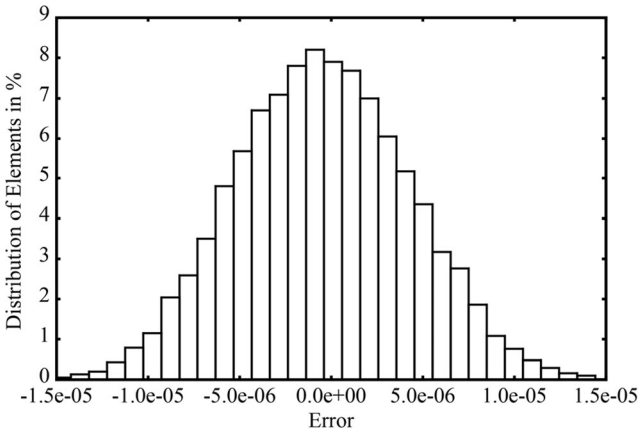

Figure 15 The error distribution of the implemented approximation.

Table 4 shows the error statistics of the implementation.

Table 4: Error statistics

	Value
Max absolute error	0.000015897615
Mean error	0.000000019483
Median error	−0.000000025005
Standard deviation of error	0.000004737796
Root Mean Square error	0.000004737692

When comparing different approximation methodologies, it should be noted that, to enable the same performance of the algorithm which they are part of, different methodologies may require different precisions. Approximation methodologies should therefore always be compared in the same context. In this case the methodologies have both been designed to have the same accuracy and an error that has a normal distributed around zero. Since they both have about the same characteristic of the error they can be compared without actually putting them in the same context. In paper [12] two implementations using the Parabolic Synthesis methodologies are compared with an implementation using the CORDIC methodology. A problem with this comparison is that the error distribution for the CORDIC methodology has major drawbacks compared to the one of the Parabolic Synthesis methodologies. As shown in [7, p. 125], CORDIC requires several extra bits of accuracy in order to cope with the error in the approximation. This illustrates that, when comparing different implementations, it is essential to use the same context. A poor error distribution of an approximation can corrupt the performance of the context that it is a part of.

The comparison is performed for an approximation of the binary logarithm of a mantissa in the range from 1 through 2. The required accuracy for the implementation is set to 15 bits.

The results of the implementation carried out with the Parabolic Synthesis are based on [11] after a redesign without bonding pads. The results for chip area, path delay, and energy consumption per sample are shown in the upper entry of Table 5. In [11], four sub-functions were used in the architecture. Unfortunately, in the implementation performed with Parabolic Synthesis in [11], no fully performed optimization of the word lengths in the look-up tables containing the coefficients was performed; therefore slightly longer word lengths than needed were used in this part of the design. This yields an impact on the chip area and, to some extent, on the critical path delay of the design. Hence the parameters such as chip area, critical path delay and power consumption are somewhat larger than they need to be. For the implementation using Harmonized Parabolic Synthesis [22], eight intervals are used in the second sub-function. In this case, optimization of the word lengths in the design was fully performed. The results of the implementation are shown in the lower entry of Table 5.

Table 5 Comparison of chip area, critical path delay and energy consumption per sample.

Methodology	Implemented Function	Chip Area	Critical Path Delay	Energy Consumption per Sample
Parabolic Synthesis	$\log_2(x)$	100%	100%	100%
		16,258 μm^2	21 ns	0.065 nW
Harmonized Parabolic Synthesis	$\log_2(x)$	30%	33%	9%
		4865 μm^2	7 ns	0.0061 nW

The conclusion from Table 5 is that the Harmonized Parabolic Synthesis is clearly superior to Parabolic Synthesis. The chip area is reduced with 70%, mainly because Parabolic Synthesis requires four sub-functions whereas Harmonized Parabolic Synthesis only needs two. To some extent, also the not fully performed optimization of the Parabolic Synthesis implementation affects its performance. With full optimization, the chip area for the Parabolic Synthesis is estimated to be reduced with roughly 10%. In computation speed, the Harmonized Parabolic Synthesis is 3 times faster. This is primarily due to that the implementation performed with Parabolic Synthesis in the critical path has more and larger multipliers. The smaller chip area and the shorter critical path of the Harmonized Parabolic Synthesis methodology results in an 11 times lower energy consumption. Comparison between Paraboloic Synthesis and the well-spread CORDIC methodology was presented in [12]. Combining this with Table 5 gives the conclusion that the Harmonized Parabolic Synthesis provides the following performance compared to CORDIC: Chip area, 39% of the one for CORDIC; critical path delay: 8% of the CORDIC delay; and Energy consumption: 4% of the one for CORDIC. Again it should be mentioned that the Harmonized Parabolic Synthesis methodology has a more favorable error distribution.

7 Conclusion

This paper has described the Harmonized Parabolic Synthesis methodology for developing hardware approximations of unary functions, such as trigonometric functions, logarithm functions, exponential functions and square root. The methodology is mainly intended for hardware implementation with moderate accuracy in computation intensive applications within, e.g., computer graphics, digital signal processing, communication systems, robotics, astrophysics and fluid physics, as well as in many other application areas. An emerging computation intensive application that serves as a strong motivating example is wireless communications systems with multiple antennas on the transmitter and receiver, known as Multiple-Input Multiple-Output (MIMO). An essential part of the computation in these systems is performed when

computing matrix inversions, which are often executed as QR decompositions in which very high throughput is needed. The QR decomposition algorithm mainly requires approximations of trigonometric functions, roots and inverses. For these computation demands, the Harmonized Parabolic Synthesis with its benefits in terms of small chip area, short critical path and low energy consumption is most favorable.

When going from Parabolic Synthesis to Harmonized Parabolic Synthesis, the major attractiveness lies in how increasing the accuracy is done. In Parabolic Synthesis the way to increase accuracy of the approximation is to introduce more sub-functions. In Harmonized Parabolic Synthesis, which only uses two sub-functions, it is instead done by increasing the number of intervals in the second sub-function. The effect of this is that, with increasing accuracy, the resulting chip area, critical path delay, and energy consumption will show a much slower increase than for Parabolic Synthesis. The Harmonized Parabolic Synthesis has also more extensive opportunities to handle truncation effects and to achieve a desired distribution of the error. The use of second-degree interpolation in the second sub-function also has the positive effect that, compared, to Parabolic Synthesis, extraordinary functions such as the third root and inverse third root also can be approximated.

The architecture based on Harmonized Parabolic Synthesis is, as the one based on Parabolic Synthesis, very suitable for pipelining. Further, both architectures can easily be reused for approximations of other unary functions by simply changing the set of coefficients.

When analyzing the hardware architecture for implementation of the logarithm, it was found that Harmonized Parabolic Synthesis reduced the complexity of the architecture significantly, compared to Parabolic Synthesis.

When analyzing the characteristics of the error for the implementation of the logarithm, it was found that the Harmonized Parabolic Synthesis has a mean error that is near zero and a median error that is zero or near zero. This implies that only a minimum of skewness is present in the error distribution and that the difference between the standard deviation and the root mean square value is very small, which in turn guarantees that the error is evenly distributed.

To get an indication of the implementation performance of the Harmonized Parabolic Synthesis, it has been compared to an implementation carried out with the Parabolic Synthesis. The comparison shows that an approximation based on the Harmonized Parabolic Synthesis methodology results in a chip area that is smaller than with Parabolic Synthesis. The throughput is also higher, and the energy consumption per sample is almost ten times lower.

As noted in Section 3, the resulting architecture when using the Harmonized Parabolic Synthesis methodology is in fact generic. That is, the choice of function is given by the (small) table of coefficients. In earlier papers, it was shown that Parabolic Synthesis can be used with good results on several functions (e.g., the sine function [9] and the exponential function [11]). Since the Harmonized version of Parabolic Synthesis, presented in the current paper, can be applied to any functions (even without the limitations of earlier methods) it can be concluded that it is an efficient method for a wide class of functions.

References

1. Luethi, P., Burg, A., Haene, S., Perels, D., Felber, N., Fichtner, W., (2007) VLSI implementation of a high-speed iterative sorted MMSE QR decomposition, *Proc. of the International Symposium on Circuits and* Systems (ISCAS 2007), New Orleans, Louisiana, USA, May 27–20, pp. 1421–1424.
2. Muller, J.-M. (2006) *Elementary Functions: Algorithm Implementation,* second ed. Birkhauser, ISBN 0–8176–4372-9, Birkhauser Boston, c/o Springer Science+Business Media Inc., 233 Spring Street, New York, NY 10013, USA.
3. Parhami, B., (2000) *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press Inc., ISBN 0–19–512583-5, 198 Madison Avenue, New York, NY 10016.
4. Johansson, K., Gustafsson, O, and Wanhammar, L., (2005) Approximation of elementary functions using a weighted sum of bit-products, *IEEE International Symposium on Circuits and Systems (ISCAS),* IEEE, pp.795–798, Aug. 28 – Sept. 2.
5. Andraka, R. (1998) A survey of CORDIC algorithms for FPGA based computers, *Proc. of the 1998 ACM/SIGDA Sixth Inter. Symp. on Field Programmable Gate Array* (FPGA'98), pp. 191–200, ISBN: 0–89791–978-5, Monterey, CA, ACM Inc.
6. Volder, J. E. (1959). The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computers, EC-8*(3), 330–334.
7. Omondi, A. (2015) *Computer-Hardware Evaluation of Mathematical Functions*," Imperial Collage Press, ISBN 978–1–78326-860-3, Imperial Collage Press, 57 Shelton Street, Covent Garden, London WC2H 9HE, UK.
8. Hertz, E, Nilsson, P. (2008) A methodology for parabolic synthesis of unary functions for hardware implementation, in *Proceedings of the 2008* IEEE Conference on Signals, Circuits and System, pp 1–6, Hammamet, Tunisia, November 7–9.
9. Hertz, E., Nilsson, P. (2009) Parabolic Synthesis Methodology Implemented on the Sine Function, *Proc. of the 2009 International Symposium on Circuits and Systems (ISCAS'09)*, Taipei, Taiwan, May 24–27.
10. Hertz, E., Nilsson, P. (2010) A methodology for parabolic synthesis, a book chapter in *VLSI, In-Tech, ISBN 978–3–902613-50-9*, pp. 199–220, Vienna, Austria.
11. Pouyan, P., Hertz, E., Nilsson, P. (2011) A VLSI implementation of logarithmic and exponential functions using a novel parabolic synthesis methodology compared to the CORDIC algorithm, in *Proc. of the European Conference on Circuit Theory and Design (ECCTD 2011)*, Linköping, Sweden.
12. Hertz, E., Svensson, B., Nilsson, P. (2016) Combining the parabolic synthesis methodology with second-degree interpolation, *Microprocessors and Microsystems,*, 42, 142–155.

13. Beiu, V., Betowski, D.J., Wu, P.-S. (2003) Over 100 dBc ROM-less piecewise non-linear direct digital frequency synthesizers, Proc. Intl. Midwest Symp. Circ. & Syst. (MWSCAS'03), Cairo, Egypt, Dec. 27–30, 760–763.
14. Sodagar, A.M., Lahiji, G.R. (2000) Parabolic approximation: a new method for phase-to-amplitude conversion in sine-output direct digital frequency synthesizers, Proc. Intl. Symp. Circ. & Sys. (ISCAS'00), 1, 515–518.
15. Tang, P.T.P. (1991) Table-lookup algorithms for elementary functions and their error analysis, *Proc. of the 10th IEEE Symposium on Computer Arithmetic,* pp. 232–236, ISBN: 0–8186–9151-4, Grenoble, France.
16. Giunta, AA., Watson, L. T. (1998) A comparison of approximation modeling techniques, *American Institute of Aeronautics and Astronautics*, AIAA-98-4758, Blacsburg.
17. Combet, M., Van Zonneveld, H., & Verbeek, L. (1965). Computation of the base two logarithm of binary numbers. *IEEE Trans. Electronic Computers, EC-14*, 863–867.
18. Mitchell, J. N. (1962). Computer multiplication and division using binary logarithms. *IRE Trans. Electronic Computers*, 512–517.
19. Synopsys's, Design Compiler: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DCGraphical/Pages/default.aspx.
20. Synopsys's, PrimeTime: http://www.synopsys.com/Tools/Implementation/SignOff/PrimeTime/Pages/default.aspx.
21. Mentor Graphics http://www.mentor.com/products/fv/modelsim/.
22. Lai, J. 2013) *Hardware Implementation of the Logarithm Function*, Master of Science Thesis at the Department of Electrical and Information Technology, Faculty of Engineering, Lund University, Sweden, September 2. http://www.eit.lth.se/sprapport.php?uid=751.

Erik Hertz received his MS degree in Electrical Engineering in 1983 and his Licentiate degree in 2011, both at the Faculty of Engineering, Lund University, Sweden. In 2016, he received his PhD degree in Computer Science and Engineering at Halmstad University, Sweden. Since 1989, he has done research at the Swedish Defense Research Agency, Linköping, Sweden, Ericsson Mobile Platforms AB, Lund, Sweden and Cargine AB, Ängelholm, Sweden, connected to Koenigsegg Automotive AB. His research has been in a wide range of areas such as radar, radio, control engineering, sensors, microwave, ASIC and solid state physics. His main interest is in algorithm development for hardware implementation. In 2012, he joined the Centre for Research on Embedded Systems (CERES), Halmstad University, Halmstad, Sweden.

Jingou Lai received his MSc degree on System on Chip Design at Lund University, Sweden in 2013. His master thesis was on the Implementation of Logarithm using Harmonized Parabolic Synthesis. He received the BSc degree in Electrical Engineering from University of Gävle, Sweden, in 2011. His bachelor thesis was on Wireless Communication and Digital Signal Processing. In summer 2012, he worked as an intern on MCU Logic Design & Verification in Appotech, China. In 2013, he worked as a Product Validation Engineer on imagine sensor in Galaxy core, China. Since April 2014, he has been working as an R&D in Cadence Design System, China.

Bertil Svensson received his MS degree in electrical engineering in 1970 and PhD degree in computer engineering in 1983, both from Lund University, Sweden. He was a member of the faculty of Halmstad University, Sweden, from 1983, served as acting professor at Luleå University of Technology, Sweden, from 1989 and was professor of computer systems engineering at Chalmers University of Technology, Sweden, from 1991. Since 1998, he has been professor of computer systems engineering at Halmstad University, where he has been the dean of the school of information science, computer and electrical engineering and the initiator and director of several research centres. Until recently, he was the director of CERES, the centre for research on embedded systems and of Research for Innovation, a strategic, university over-arching research program. His research interests are parallel computer architecture, embedded parallel computing and cooperating embedded systems, including wireless and intelligent systems. He is a member of the IEEE and the ACM.

Peter Nilsson passed away in February 2016. Peter was since July 2008, a full professor in electronic design. In February 1988, Peter reached his Master of Science degree in Electrical Engineering and in May 1996, his degree Doctor of Philosophy in Engineering was awarded, both at Faculty of Engineering, Lund University. Since 1996 Peter has supervised seven PhD students to the completion of their doctoral degree, and had two on the way. He has supervised and/or examined 156 students at the Master's thesis level, both nationally and internationally. Peter authored or co-authored 126 peer-reviewed journal and conference papers. He was the program manager as well as the person authorized to sign for the "Socware Research & Education" program, a five-year national 15 million USD program. He was an Associate Editor for IEEE Transactions on Circuits and Systems I, 2004–05, and was a member of the VLSI Systems and Applications Technical Committee in IEEE Circuits and Systems Society. He was awarded the IEEE Senior Member grade. He was also a Technical Program Committee (TPC) member of the IEEE International Solid State Circuits Conference (ISSCC). Peter's main interest was implementation of digital ASICs, and then often with a focus on wireless communication.