10

A Methodology for Parabolic Synthesis

Erik Hertz and Peter Nilsson
Department of Electrical and Information Technology, Lund University
Sweden

1. Introduction

In relatively recent research of the history of science interpolation theory, in particular of mathematical astronomy, revealed rudimentary solutions of interpolation problems date back to early antiquity (Meijering, 2002). Examples of interpolation techniques originally conceived by ancient Babylonian as well as early-medieval Chinese, Indian, and Arabic astronomers and mathematicians can be linked to the classical interpolation techniques developed in Western countries from the 17th until the 19th century. The available historical material has not yet given a reason to suspect that the earliest known contributors to classical interpolation theory were influenced in any way by mentioned ancient and medieval Eastern works. For the classical interpolation theory it is justified to say that there is no single person who did so much for this field as Newton. Therefore, Newton deserves the credit for having put classical interpolation theory on a foundation. In the course of the 18th and 19th century Newton's theories were further studied by many others, including Stirling, Gauss, Waring, Euler, Lagrange, Bessel, Laplace, and Everett. Whereas the developments until the end of 19th century had been impressive, the developments in the past century have been explosive. Another important development from the late 1800s is the rise of approximation theory. In 1885, Weierstrass justified the use of approximations by establishing the so-called approximation theorem, which states that every continuous function on a closed interval can be approximated uniformly to any prescribed accuracy by a polynomial. In the 20th century two major extensions of classical interpolation theory is introduced: firstly the concept of the cardinal function, mainly due to E. T. Whittaker, but also studied before him by Borel and others, and eventually leading to the sampling theorem for band limited functions as found in the works of J. M. Whittaker, Kotel'nikov, Shannon, and several others, and secondly the concept of oscillatory interpolation, researched by many and eventually resulting in Schoenberg's theory of mathematical splines.

The parabolic synthesis methodology
Unary functions, such as trigonometric functions, logarithms as well as square root and division functions are extensively used in computer graphics, digital signal processing, communication systems, robotics, astrophysics, fluid physics, etc. For these high-speed applications, software solutions are in many cases not sufficient and a hardware implementation is therefore needed. Implementing a numerical function $f(x)$, by a single

look-up table (Tang, 1991) is simple and fast which is strait forward for low-precision computations of $f(x)$, i.e., when x only has a few bits. However, when performing high-precision computations a single look-up table implementation is impractical due to the huge table size and the long execution time.
Approximations only using polynomials have the advantage of being ROM-less, but they can impose large computational complexities and delays (Muller, 2006). By introducing table based methods to the polynomials methods the computational complexity can be reduced and the delays can also be decreased to some extent (Muller, 2006).
The CORDIC (COordinate Rotation DIgital Computer) algorithm (Volder, 1959) (Andrata, 1998) has been used for these applications since it is faster than a software approach. CORDIC is an iterative method and therefore slow which makes the method insufficient for this kind of applications.
The proposed methodology of parabolic synthesis (Hertz & Nilsson, 2008) develops functions that perform an approximation of original functions in hardware. The architecture of the processing part of the methodology is using parallelism to reduce the execution time. For the development of approximations of functions a parabolic synthesis methodology has been applied. Only low complexity operations that are simple to implement in hardware are used

2. Methodology

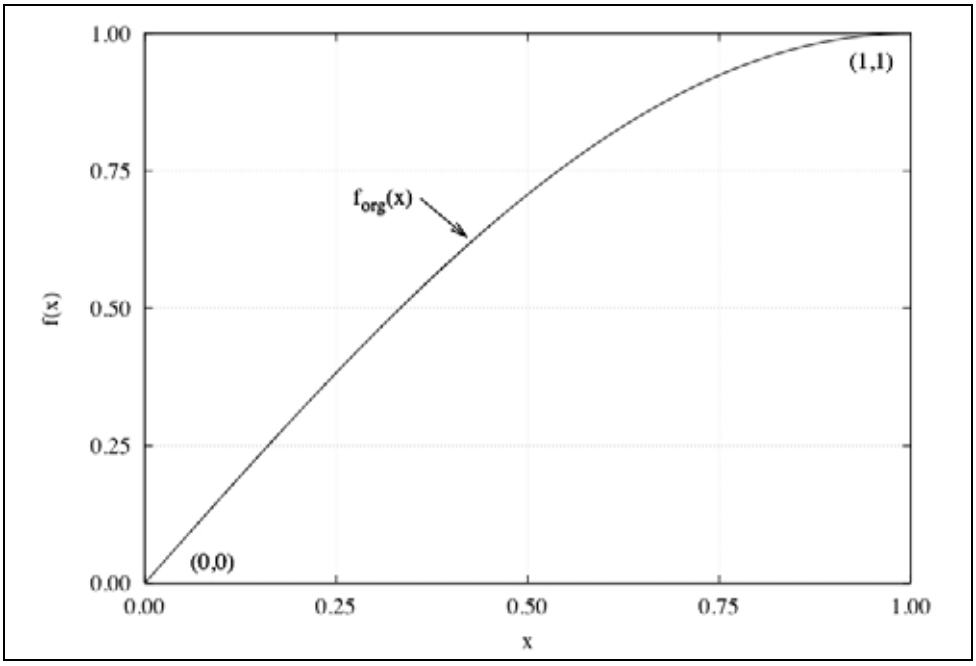

Fig. 1. Example of normalized function, in this case $\sin\left(\frac{\pi \cdot x}{2}\right)$.

The methodology is developed for implementing approximations of unary functions in hardware. The approximation part is of course the important part of this work but there are sometimes two other steps that are necessary, a preprocessing normalization and postprocessing transformation as described by (P.T.P. Tang, 1991) (Muller, 2006). The computation is therefore divided into three steps, normalizing, approximation and transforming.

2.1 Normalizing

The purpose with the normalization is to facilitate the hardware implementation by limiting the numerical range.

The normalization has to satisfy that the values are in the interval $0 \le x < 1$ on the x-axis and $0 \le y < 1$ on the y-axis. The coordinates of the starting point shall be (0,0). Furthermore, the ending point shall have coordinates smaller than (1,1) and the function must be strictly concave or strictly convex through the interval. An example of such a function, called an original function $f_{org}(x)$, is shown in Fig. 1.

2.2 Developing the Hardware Architecture

When developing a hardware architecture that approximates an original function, only low complexity operations are used. Operations such as shifts, additions and multiplications are efficient to implement in hardware and therefore searched for. The downscaling of the semiconductor technologies and the development of efficient multiplier architectures has made the multiplication operation efficient in both size and execution, time when implemented in hardware. The multiplier is therefore commonly used in this methodology when developing the hardware.

As in Fourier analysis (Fourier, 1822) the proposed methodology is based on decomposition of basic functions. The proposed methodology is not, as in Fourier analysis, a decomposition method in terms of sinusoidal functions but in second order parabolic functions. Second order parabolic functions are used since they can be implemented using low complexity operations. The proposed methodology also differs from the Fourier synthesis process since the proposed methodology is using multiplications in the recombination process and not additions as in the Fourier case.

The proposed methodology is founded on terms of second ordered parabolic functions called sub-functions $s_n(x)$, that when recombined, as shown in (1), obtains to the original function $f_{org}(x)$. When developing the approximate function, the accuracy depends on the number of sub-functions used.

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

The procedure when developing sub-functions is to divide the original function $f_{org}(x)$, with the first sub-function $s_1(x)$. This division generates the first function $f_1(x)$, as shown in (2).

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} \tag{2}$$

The first sub-function $s_1(x)$, will be chosen to be feasible for hardware, according to the methodology described in (4). In the same manner the following functions $f_n(x)$, are generated, as shown in (3).

$$f_{n+1}(x) = \frac{f_n(x)}{s_{n+1}(x)} \tag{3}$$

The purpose with the normalization is to facilitate the hardware implementation by limiting the numerical range.

2.3 Methodology for developing sub-functions

The methodology for developing sub-functions is founded on decomposition of the original function $f_{org}(x)$, in terms of second order parabolic functions for the interval $0 \leq x < 1.0$ and the sub intervals within the interval. The second order parabolic function is chosen as decomposition function since the structure is reasonable simple to implement in hardware i.e. only low complexity operations such as additions and multiplications are used.

First sub-function

The first sub-function $s_1(x)$, is developed by dividing the original function $f_{org}(x)$, with x as an approximation.

As shown in Fig. 2 there are two possible results after dividing the original function with x, one where $f(x)>1$ and one where $f(x)<1$.

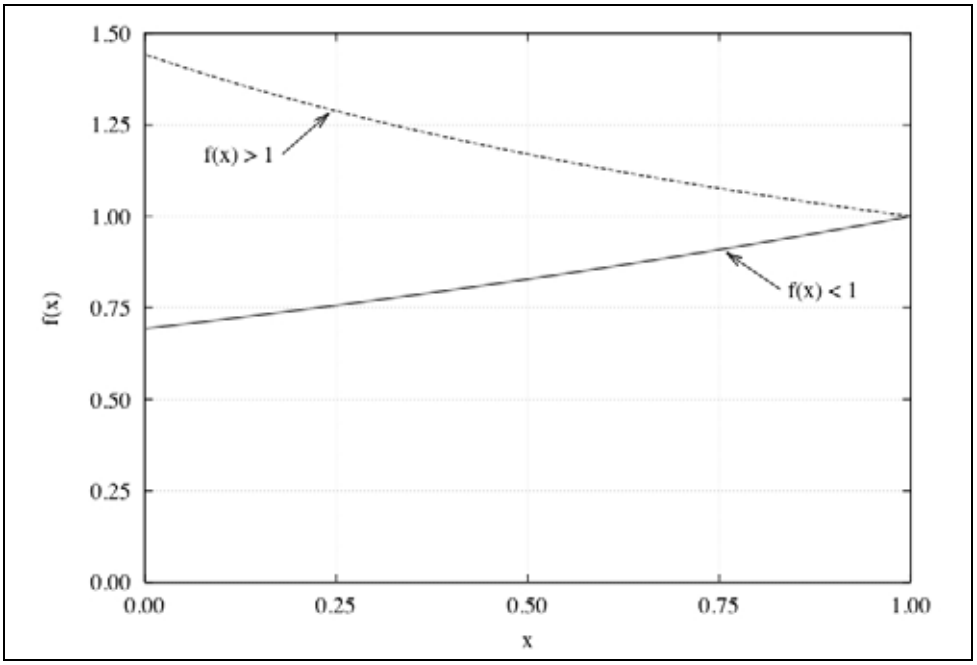

Fig. 2. Two possible results after dividing an original function with x.

The first sub-function $s_1(x)$, is according to (4). To approximate these functions $1+(c_1 \cdot (1-x))$ is used. The first sub-function $s_1(x)$, is given by a multiplication of x and $1+(c_1 \cdot (1-x))$ which results is a second order parabolic function according to (4).

$$s_1(x) = x \cdot (1 + (c_1 \cdot (1-x))) = x + (c_1 \cdot (x - x^2)) \tag{4}$$

In (4) the coefficient c_1 is determined as the limit from the division of the original function with x and subtracted with 1, according to (5).

$$c_1 = \lim_{x \to 0} \frac{f_{org}(x)}{x} - 1 \tag{5}$$

Second sub-function

The first function $f_1(x)$, is calculated according to (2) and the result of this operation is a function which appearance is similar to a parabolic function, as shown in Fig. 3.

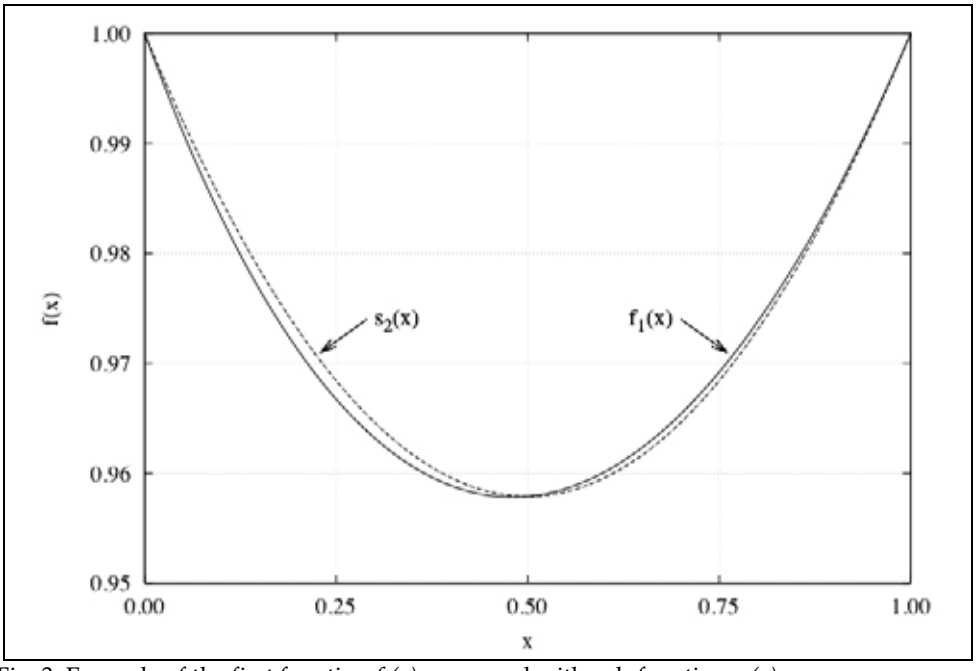

Fig. 3. Example of the first function $f_1(x)$ compared with sub-function $s_2(x)$.

The second sub-function $s_2(x)$, is chosen according to the methodology as a second order parabolic function, see (6).

$$s_2(x) = 1 + (c_2 \cdot (x - x^2)) \tag{6}$$

In (6) the coefficient c_2, is chosen to satisfy that the quotient between the function $f_1(x)$, and the second sub-function $s_2(x)$, is equal to 1 when x is equal to 0.5, see (7).

$$c_2 = 4 \cdot \left(f_1\left(\frac{1}{2}\right) - 1 \right) \tag{7}$$

Thereby the second function $f_2(x)$, will get a shape of a lying **S**, as shown in Fig. 4.

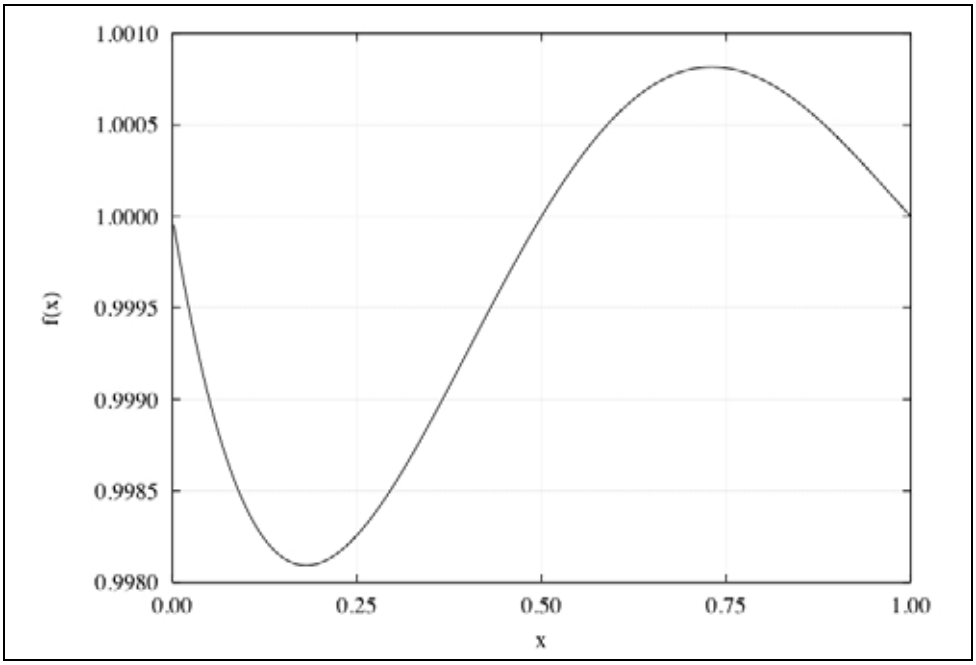

Fig. 4. Example of the second function $f_2(x)$, shaped like a lying **S**.

When developing the third sub-function $s_3(x)$, the function is to be split into two parabolic functions where the first function is restricted by function $f_2(x)$ to be in the interval $0 \le x < 0.5$ and the second function is thus restricted to the interval $0.5 \le x < 1.0$. By splitting the function we get strictly convex and concave functions in each interval. The intervals can be chosen differently but that will lead to a more complex hardware, as shown in section 3.

Sub-functions when $n > 2$

For functions $f_n(x)$ when $n > 2$, the function is characterized by the form of one or more **S** shaped functions. When developing the higher order sub-functions, each **S** shaped function is divided into two parabolic functions. For each sub interval, a parabolic sub-function is developed as an approximation of the function $f_n(x)$ in the sub interval. To show which sub

interval the partial functions is valid for, the subscript index is increased with the index m, which gives the following appearance of the partial function $f_{n,m}(x)$.

$$f_n(x) = \begin{cases} f_{n,0}(x), & 0 \le x < \frac{1}{2^{n-1}} \\ f_{n,1}(x), & \frac{1}{2^{n-1}} \le x < \frac{2}{2^{n-1}} \\ \cdots & \\ f_{n,2^{n-1}-1}(x), & \frac{2^{n-1}-1}{2^{n-1}} \le x < 1 \end{cases} \tag{8}$$

In equation (8) it is shown how the function $f_n(x)$, is divided into partial functions $f_{n,m}(x)$, when $n > 2$.

As shown in (8), the number of partial functions is doubled for each order of $n > 1$ i.e. the number of partial functions is 2^{n-1}. From these partial functions, the corresponding sub-functions are developed. Analogous to the function $f_n(x)$, also the sub-function $s_{n+1}(x)$, will have partial sub-functions $s_{n+1,m}(x)$. In equation (9) it is shown how the sub-function $s_n(x)$, is divided into partial functions when $n > 2$.

$$s_n(x) = \begin{cases} s_{n,0}(x), & 0 \le x < \frac{1}{2^{n-2}} \\ s_{n,1}(x), & \frac{1}{2^{n-2}} \le x < \frac{2}{2^{n-2}} \\ \cdots & \\ s_{n,2^{n-2}-1}(x), & \frac{2^{n-2}-1}{2^{n-2}} \le x < 1 \end{cases} \tag{9}$$

Note that in (9), the partial functions to the sub-functions; x has been changed to x_n. The change to x_n is normalization to the corresponding interval, which simplifies the hardware implementation of the parabolic function. To simplify the normalization of the interval of x_n it is selected as an exponentiation by 2 of x where the integer part is removed. The normalization of x is therefore done by multiplying x with 2^{n-2}, which in hardware is $n-2$ left shifts and the integer part is dropped, which gives x_n as a fractional part ($frac(\ )$) of x, as shown in (10).

$$x_n = frac\left(2^{n-2} \cdot x\right) \tag{10}$$

As in the second sub-function $s_2(x)$, the second order parabolic function is used as an approximation of the interval of the function $f_{n-1}(x)$, as shown in (11).

$$s_{n,m}(x_n) = 1 + \left(c_{n,m} \cdot \left(x_n - x_n^2\right)\right) \tag{11}$$

Where the coefficients $c_{n,m}$ is computed according to (12).

$$c_{n,m} = 4 \cdot \left(f_{n-1,m} \left(\frac{2 \cdot (m+1) - 1}{2^{n-1}} \right) - 1 \right) \tag{12}$$

After the approximation part the result is transformed into its desired form.

3. Hardware Implementation

For the hardware implementation two's complement representation (Parhami, 2000) is used. The implementation is divided into three hardware parts, preprocessing, processing, and postprocessing as shown in Fig. 5, which was introduced by (P.T.P. Tang, 1991), (Muller, 2006).

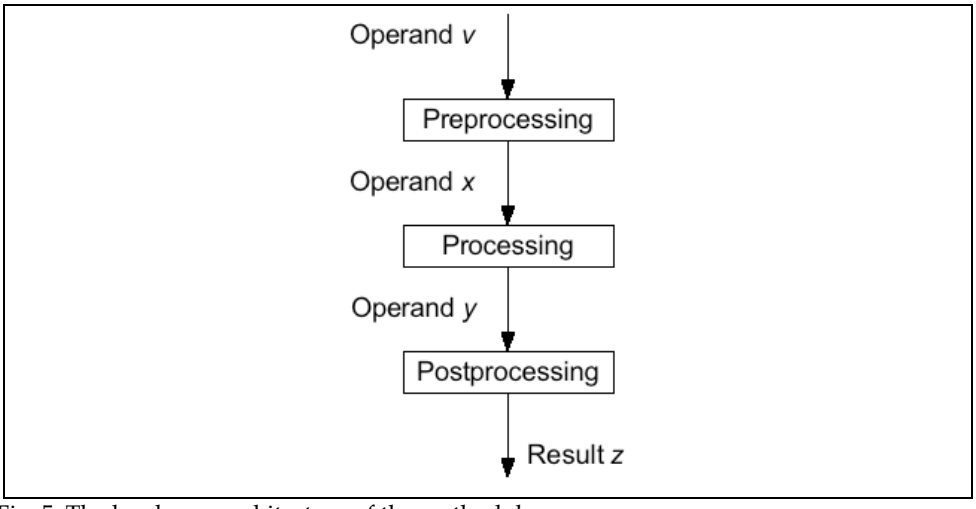

Fig. 5. The hardware architecture of the methodology.

3.1 Preprocessing

In this part the incoming operand v is normalized to prepare the input to the processing part, according to section 2.1.

If the approximation is implemented as a block in a system the preprocessing part can be taken into consideration in the previous blocks, which implies that the preprocessing part can be excluded.

3.2 Processing

In the processing part the approximation of the original function is directly computed in either iterative or parallel hardware architecture.

The three equations (4), (6) and (11) has the same structure which gives that the approximation can be implemented as an iterative architecture as shown in Fig. 6.

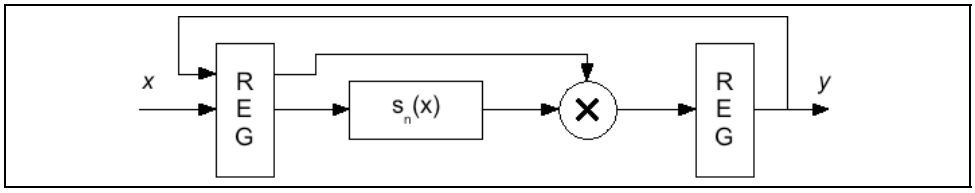

Fig. 6. The principle of an iterative hardware architecture.

The benefit of the iterative architecture is the small chip area whereas the disadvantage is longer computation time.
The advantages with parallel hardware architecture are that it gives a short critical path and fast computation to the prize of a larger chip area. The principle of the parallel hardware architecture for four sub-functions is shown in Fig. 7.

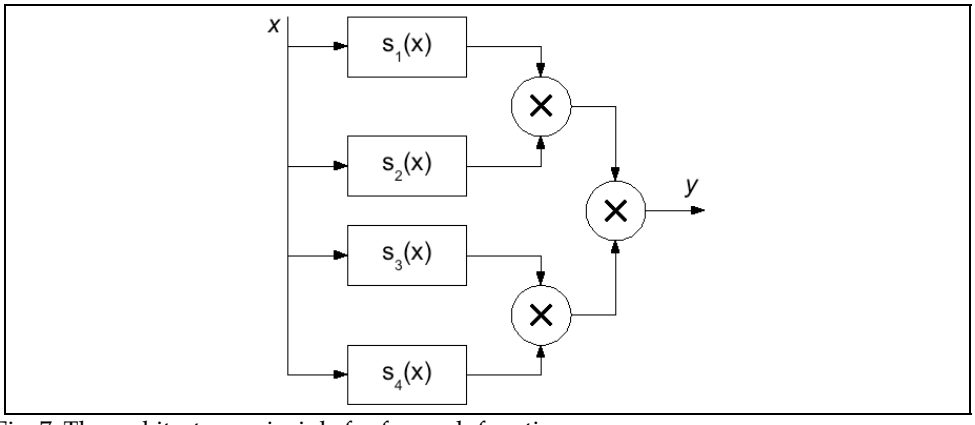

Fig. 7. The architecture principle for four sub-functions.

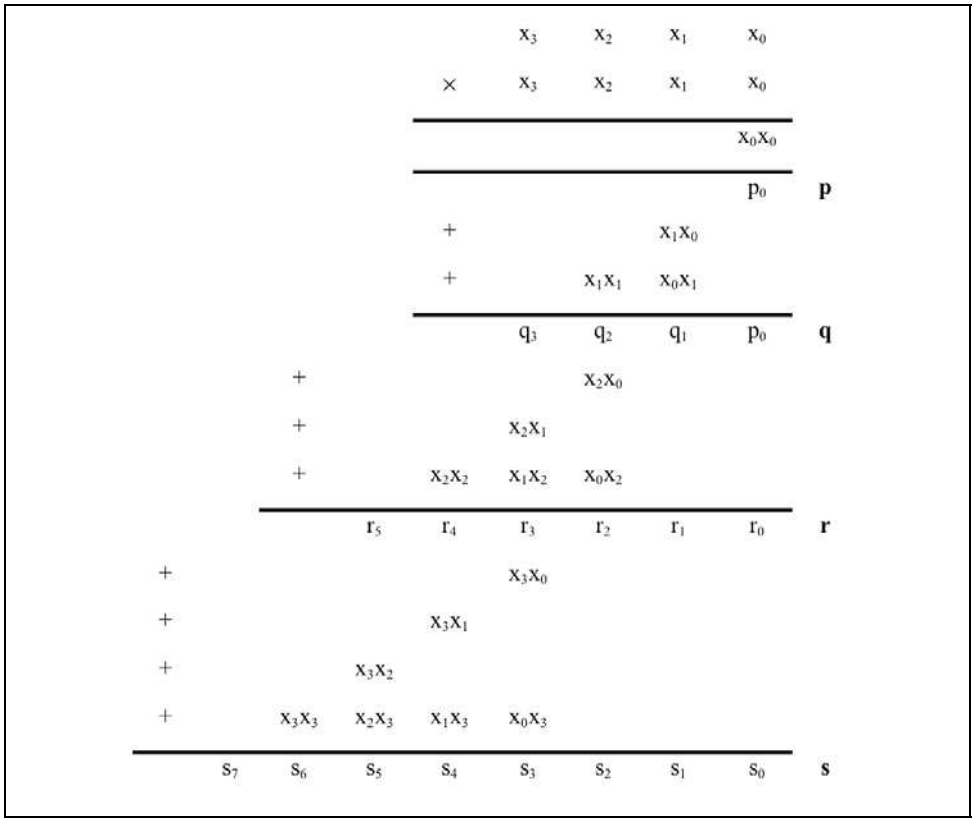

Fig. 8. Squaring algorithm for the partial products x_n^2.

To increase the throughput even more, pipeline stages can be implemented in the parallel hardware architecture.

In the sub-functions (4), (6) and (11) x^2 and x_n^2 are reoccurring operations. Since the square operation x_n^2, in the parallel hardware architecture is a partial result of x^2 a unique squarer has been developed. In Fig. 8 the algorithm that performs the squaring and delivers partial product of x_n^2 is described.

The squaring algorithm for the partial products x_n^2 can be simplified as shown in Fig. 9.

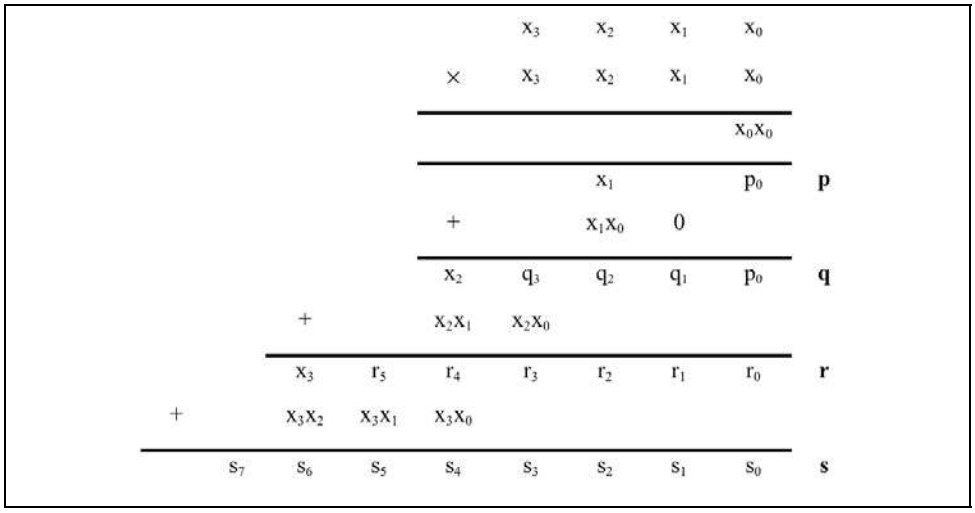

Fig. 9. Simplified squaring algorithm for the partial products x_n^2.

In Fig. 8 and Fig. 9, the squaring algorithm that performs the partial products x_n^2, shown. The first partial product p, is the squaring of the least significant bit in x. The second partial product q, is the squaring of the two least significant bits in x. The partial product r, is the result of the squaring of the three least significant bits in x and s is the result of the squaring of x. The squaring operation is performed with unsigned numbers. When analyzing the squarer in Fig. 8 and Fig. 9, it was found that the resemblance to a bit-serial squarer (Ienne & Viredaz, 1994) (Pekmestz et al., 2001) is large. By introducing registers in the design of the bit-serial squarer the partial results of x_n^2 is easily extracted. The squaring algorithm can thus be simplified to one addition only when computing each partial product.

From (4), (6) and (11) it is found that only the coefficients values differentiate when implementing different unary functions. This implies that different unary functions can be realized in the same hardware in the processing part, just by using different sets of coefficients.

Since the methodology is calculating an approximation of the original function the error to the desired precision can be both positive and negative. Especially, if the value of the approximation is less than the desired precision, the word length can have to be increased compared with the word length needed to accomplish the desired precision. If the order of the last used sub-function is $n > 1$, an improvement of the precision can be done by optimizing one or more coefficients c_2 in (7) or $c_{n,m}$ in (12). The optimization of the coefficients will minimize the error in the last used sub-function and thereby it can reduce the word length needed to accomplish the desired accuracy. Computer simulations perform such coefficient optimization numerically.

3.3 Postprocessing

The postprocessing part transforms the value to the output result z. If the approximation is implemented as a block in a system the postprocessing part can be taken into consideration in the following blocks, which implies that the postprocessing part can be excluded.

4. Implementation of the sine function

An implementation of the function *sin(v)*, using the proposed methodology (Hertz & Nilsson, 2009) is described in this section as an example.

4.1 Preprocessing

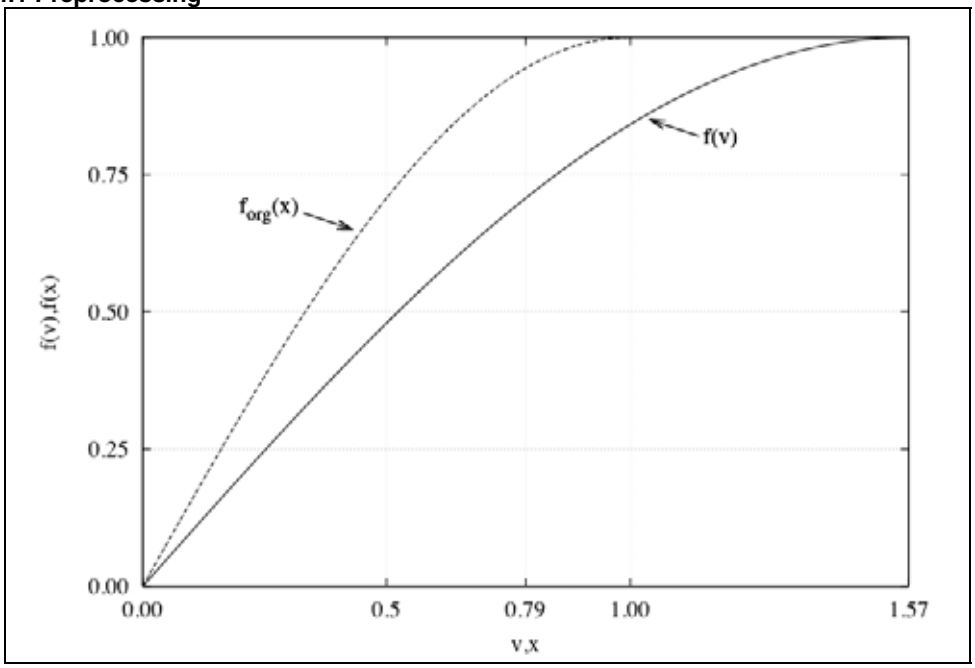

Fig. 10. The function *f(v)* before normalization and the original function $f_{org}(x)$.

To satisfy that the values of the incoming operand x is in the interval $0 \leq x < 1$ a $\pi/2$ is multiplied with the operand as shown in (13).

$$v = \frac{\pi}{2} \cdot x \tag{13}$$

To normalize the $f(v)=sin(v)$ function v is substituted with x which gives the original function $f_{org}(x)$ (14).

$$f_{org}(x) = \sin\left(\frac{\pi}{2} \cdot x\right) \tag{14}$$

In Fig. 10 the *f(v)* function is shown together with the original function $f_{org}(x)$.

4.2 Processing

For the processing part, sub-functions are developed according to the proposed methodology. For the first sub-function $s_1(x)$, the coefficient c_1 is defined according to (5). The determined value of the coefficient is shown in (15).

$$s_1(x) = x + \left(\left(\frac{\pi}{2} - 1 \right) \cdot \left(x - x^2 \right) \right) \tag{15}$$

The first function $f_1(x)$, is computed as shown in (16).

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} \tag{16}$$

To develop the second sub-function $s_2(x)$, the coefficient c_2 is defined according to (7). The determined value of the coefficient is shown in (17).

$$s_2(x) = 1 + \left(\left(0.400858 \right) \cdot \left(x - x^2 \right) \right) \tag{17}$$

The second function $f_2(x)$, is computed as shown in (18).

$$f_2(x) = \frac{f_1(x)}{s_2(x)} \tag{18}$$

To develop the third sub-functions $s_3(x)$, the second function $f_2(x)$, is divided into its two partial functions as shown in (8). The third order of sub-functions is thereby divided into two sub-functions, where $s_{3,0}(x_3)$ is restricted to the interval $0 \le x < 0.5$ and $s_{3,1}(x_3)$ is restricted to the interval $0.5 \le x < 1.0$ according to (9). A normalization of x to x_3 is done to simplify in the implementation in hardware, which is described in (10).
For each sub-function, the corresponding coefficients $c_{3,0}$ and $c_{3,1}$ is determined. These coefficients are determined according to (12) where higher order sub-functions can be developed. The determined values of the coefficients are shown in (19).

$$\begin{aligned} s_{3,0}(x_3) &= 1 + \left(-0.0122452 \cdot \left(x_3 - x_3^2 \right) \right) \quad 0 \le x < 0.5 \\ s_{3,1}(x_3) &= 1 + \left(0.0105947 \cdot \left(x_3 - x_3^2 \right) \right) \quad 0.5 \le x < 1 \end{aligned} \tag{19}$$

The third function $f_3(x)$, is computed as shown in (20).

$$f_3(x) = \frac{f_2(x)}{s_3(x)} \tag{20}$$

To develop the fourth sub-functions $s_4(x)$, the third function $f_3(x)$, is divided into its four partial functions as shown in (8). The fourth order of sub-functions is thereby divided into four sub-functions, where $s_{4,0}(x_4)$ is restricted to the interval $0 \le x < 0.25$, $s_{4,1}(x_4)$ is restricted

to the interval $0.25 \le x < 0.5$, $s_{4,2}(x_4)$ is restricted to the interval $0.5 \le x < 0.75$ and $s_{4,3}(x_4)$ is restricted to the interval $0.75 \le x < 1.0$ according to (9). A normalization of x to x_4 is done to simplify the hardware implementation, which is described in (10).
For each sub-function, the corresponding coefficients $c_{4,0}$, $c_{4,1}$, $c_{4,2}$ and $c_{4,3}$ is determined. These coefficients are determined according to (12) which accomplish that higher order of sub-functions can be developed. The determined values of the coefficients are shown in (21).

$$\begin{aligned} s_{4,0}(x_4) &= 1 + \left(-0.00223363 \cdot \left(x_4 - x_4^2\right)\right) \quad 0 \le x < 0.25 \\ s_{4,1}(x_4) &= 1 + \left(0.00192558 \cdot \left(x_4 - x_4^2\right)\right) \quad 0.25 \le x < 0.5 \\ s_{4,2}(x_4) &= 1 + \left(-0.00119216 \cdot \left(x_4 - x_4^2\right)\right) \quad 0.5 \le x < 0.75 \\ s_{4,3}(x_4) &= 1 + \left(0.00126488 \cdot \left(x_4 - x_4^2\right)\right) \quad 0.75 \le x < 1 \end{aligned} \tag{21}$$

No postprocessing is needed since the result out from the processing part has the right size.

4.3 Optimization

If no more sub-functions are to be developed the precision of the approximation can be further improved by optimization of coefficients $c_{4,0}$, $c_{4,1}$, $c_{4,2}$ and $c_{4,3}$. As shown in Fig. 12 sub-function $s_{4,3}(x)$ in the interval $0.75 \le x < 1.0$ has the largest relative error. When performing an optimization of sub-function $s_{4,3}(x)$ in the interval $0.75 \le x < 1.0$ it was found that the word length in the computations could be reduced from 17 bits to 16 bits.

4.4 Architecture

In Fig. 11, architecture of the approximation of the sine function using the proposed methodology is shown.
The x^2 block in Fig. 11 is the special designed multiplier described in Fig. 8 and Fig. 9 that delivers the partial results q, q_3 and q_4 used in the following blocks. In the x-q block, x is subtracted with the partial result q, from the x^2 block. The result r from the x-q block is then used in the two following blocks as shown in Fig. 11. In the $x+(c_1 \cdot r)$ block is $s_1(x)$ performed, in $1+(c_2 \cdot r)$ is $s_2(x)$ performed, in $1+(c_3 \cdot (x_3 - q_3))$ is $s_3(x)$ performed and in $1+(c_4 \cdot (x_4 - q_4))$ is $s_4(x)$ performed. Note, that in the blocks for sub-function $s_3(x)$ and $s_4(x)$, the individual index m is addressing the MUX that selects the coefficients in the block.

4.5 Optimization of Word Length

As shown in (19) and (21) the absolute value of the coefficients decreases in size with increasing index number of the coefficient. In similarity to the word length of the coefficients the word length of the $(x_n - q_n)$ part, shown in Fig. 11, will decrease in size with increasing index number. The decreased word length will course that the size of the multiplier used in a sub-function to decreas accordingly to the highest value bit in the coefficients and of the $(x_n - q_n)$ part. In resemblance to above the size of the multipliers computing the multiplication of the sub-functions can be analyzed. This analysis will also result in that some of the following multipliers accordingly can be decrease in size.

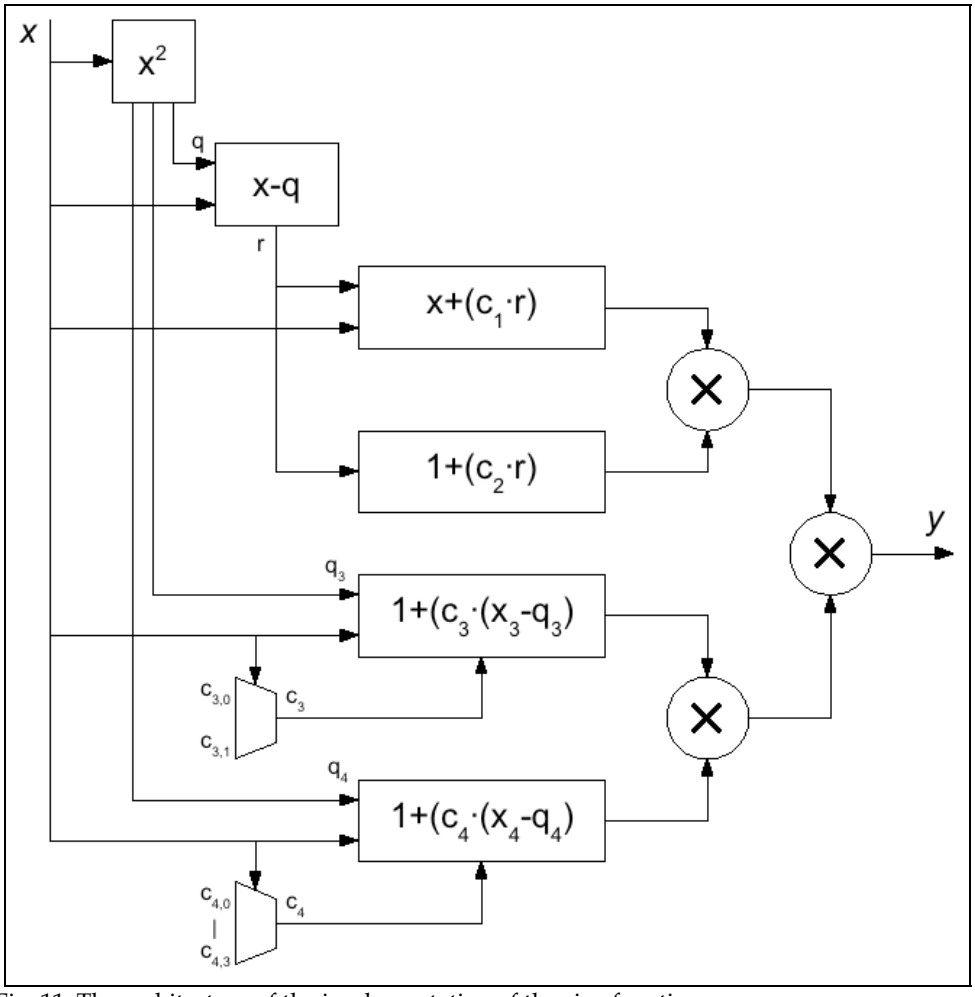

Fig. 11. The architecture of the implementation of the sine function.

4.6 Precision

In Fig. 12 the resulting precision when using one to four sub-functions is shown. Decibel scale is used to visualize the precision since the combination of binary numbers and dB works very well together. In dB scale 2 is equal to $20\log_{10}(2) = 20\cdot(0.3) = 6$ dB and since 6 dB corresponds to 1 bit, this will make it simpler to understand the result. As shown in Fig. 12, the relative error decreases with the number of used sub-functions. With 4 sub-functions we can see that we have accuracy better that 14 bits that will result in at least a latency of 14 adders in the CORDIC algorithm is used.

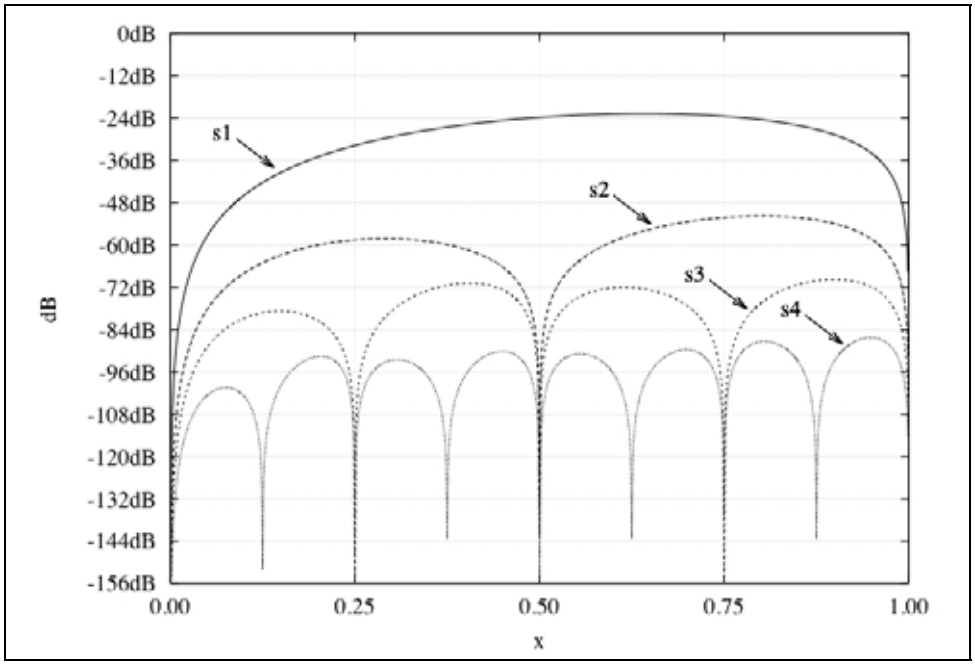

Fig. 12. Estimation of the relative error between the original function and different numbers of sub-functions.

As shown in Fig. 12, the relative error decreases with the number of sub-functions used. However, increases the delay with the number of sub-function as shown in Table 1.

Number of sub-functions	Delay
1	2 mult + 2 add
2	3 mult + 2 add
3 to 4	4 mult + 2 add
5 to 8	5 mult + 2 add

Table 1. Delay relative the number of sub-functions.

The delay of one multiplier is about two adders.

5. Comparison

The most common methods used when implementing approximation of a unary functions in hardware are look-up tables, polynomials, table-based methods with polynomials and CORDIC. Computation by table look-up is attractive since memory is much denser than random logic in VLSI realizations. However, since the size of the look-up table grows exponentially with increasing word lengths, both the table size and execution time becomes totally intolerable. Computation by polynomials is attractive since it is ROM-less. The disadvantages are that it can impose large computational complexities and delays.

Computation by table-based methods combined with polynomials is attractive since it reduces the computational complexity and decreases the delays. Since the size of the look-up tables grows with the accuracy the execution time also increases with the needed accuracy. Computation by using CORDIC is attractive since it is using an angular rotation algorithm that can be implemented with small look-up tables and hardware, which is limited to simple shifts and additions. The CORDIC algorithm is an iterative method with high latency and long delays. This makes the method insufficient for applications where short execution time is essential.
In all methods including the proposed method, it is a trade-off between complexity and memory storage. By using parallelism in the computation and parabolic synthesis in the recombination process, the proposed methodology thereby gets a short critical path, which assures fast computation.

6. Using the Methodology

It has been shown that the methodology of parabolic synthesis can directly compute the sine function but the methodology is also able to compute other trigonometric functions, logarithms as well as square root and division. In the following parts algorithms for elementary functions will be shown.
When describing the implementation of each function the different parts are shown in a table. The first row in the table shows the function to be implemented and in which interval the function is implemented. In the second row it is described how to perform the normalization of the function. The third row shows the original function to be used when developing the approximation. The last row describes how to perform transformation of the approximation into desired interval.

6.1 The Sine Function

When developing the algorithm that performs the approximation of the sine function, the normalization in the preprocessing part is performed as a substitution according to Table 2. Since the outcome of the approximation has the desired form no postprocessing is needed.

	Algorithm	Range
Function	$f(v)=\sin(v)$	$0 \le v < \frac{\pi}{2}$
Preprocessing	$x=\frac{2}{\pi}\cdot v$	$0 \le x < 1$
Processing	$y=\sin\left(\frac{\pi}{2}\cdot x\right)$	$0 \le y < 1$
Postprocessing	$z=y$	$0 \le z < 1$

Table 2. The algorithms for the sine function.

6.2 The Cosine Function

The algorithm that performs the approximation of the cosine function is founded on the algorithm that performs the approximation of the sine function. To perform the

approximation of the cosine function x is substituted with $1-x$ in the preprocessing part of the approximation for the sine function.

	Algorithm	Range
Function	$f(v)=\cos(v)$	$0 < v \le \frac{\pi}{2}$
Preprocessing	$x = 1 - \frac{2}{\pi} \cdot v$	$0 \le x < 1$
Processing	$y = \sin\left(\frac{\pi}{2} \cdot x\right)$	$0 \le y < 1$
Postprocessing	$z = y$	$0 \le z < 1$

Table 3. The algorithm for the cosine function.

6.3 The Arcsine Function

When developing the algorithm that performs the approximation of the arcsine function, the methodology has problems to perform an approximation for angels larger than $\pi/4$. Therefore, the range of the approximation has been limited according to the range of the function in Table 4. To satisfy the requirements of the methodology in the preprocessing part a substitution according to Table 4 has to be performed. To get the desired outcome the approximation is multiplied with a factor according to Table 4.

	Algorithm	Range
Function	$f(v)=\arcsin(v)$	$0 \le v < \frac{1}{\sqrt{2}}$
Preprocessing	$x = \sqrt{2} \cdot v$	$0 \le x < 1$
Processing	$y = \arcsin\left(\frac{x}{\sqrt{2}}\right)$	$0 \le y < 1$
Postprocessing	$z = \frac{\pi}{4} \cdot y$	$0 \le z < \frac{\pi}{4}$

Table 4. The algorithm for the arcsine function.

6.4 The Arccosine Function

The algorithm that performs the approximation of the arccosine function is founded on the algorithm performing the approximation of the arcsine function. The difference between the two approximations is in the transformation in the postprocessing part, as shown in Table 5.

	Algorithm	Range
Function	$f(v) = \arccos(v)$	$0 \le v < \frac{1}{\sqrt{2}}$
Preprocessing	$x = \sqrt{2} \cdot v$	$0 \le x < 1$
Processing	$y = \arcsin\left(\frac{x}{\sqrt{2}}\right)$	$0 \le y < 1$
Postprocessing	$z = \frac{\pi}{4} + \frac{\pi}{4} \cdot (1 - y)$	$\frac{\pi}{4} < z \le \frac{\pi}{2}$

Table 5. The algorithm for the arccosine function.

6.5 The Tangent Function

When developing the algorithm that performs the approximation of the tangent function the angle range is from 0 to π/4, since the tangent function is not strictly concave or convex for higher angles. To perform the normalization the preprocessing part is performed as a substitution according to Table 6. Since the outcome of the approximation has the desired form no postprocessing is needed.

	Algorithm	Range
Function	$f(v) = \tan(v)$	$0 \le v < \frac{\pi}{4}$
Preprocessing	$x = \frac{4 \cdot v}{\pi}$	$0 \le x < 1$
Processing	$y = \tan\left(\frac{\pi}{4} \cdot x\right)$	$0 \le y < 1$
Postprocessing	$z = y$	$0 \le z < 1$

Table 6. The algorithm for the tangent function.

6.6 The Arctangent Function

When developing the algorithm that performs the approximation of the arctangent function it can only be performed in the range from 0 to 1 where the function is strictly concave or convex. To get the desired outcome the approximation is in the postprocessing part multiplied with a factor according to Table 7.

	Algorithm	Range
Function	$f(v) = \arctan(v)$	$0 \le v < 1$
Preprocessing	$x = v$	$0 \le x < 1$
Processing	$y = \arctan(x) \cdot \frac{4}{\pi}$	$0 \le y < 1$
Postprocessing	$z = \frac{\pi}{4} \cdot y$	$0 \le z < \frac{\pi}{4}$

Table 7. The algorithm for the arctangent function.

6.7 The Logarithmic Function

When developing the algorithm that performs the approximation of the logarithm function with the base two, it is only perform on the mantissa of the floating-point number, since the exponent part is scaling the mantissa. For the preprocessing part a substitution according to Table 8 has to be performed to satisfy the normalization criteria's for the methodology. Since the outcome of the approximation has the desired form no postprocessing is needed.

	Algorithm	Range
Function	$f(v) = \log_2(v)$	$1 \le v < 2$
Preprocessing	$x = v - 1$	$0 \le x < 1$
Processing	$y = \log_2(1 + x)$	$0 \le y < 1$
Postprocessing	$z = y$	$0 \le z < 1$

Table 8. The algorithm for the logarithm function.

6.8 The Exponential Function

When developing the algorithm that performs the approximation of the exponential function with the base two, it is only performed on the fractional part of the logarithm since the integer part is scaling the fractional part of the logarithm. As shown in Table 9 only a one needs to be added in the postprocessing part to get the desired outcome.

	Algorithm	Range
Function	$f(v) = 2^v$	$0 \le v < 1$
Preprocessing	$x = v$	$0 \le x < 1$
Processing	$y = 2^x - 1$	$0 \le y < 1$
Postprocessing	$z = 1 + y$	$1 \le z < 2$

Table 9. The algorithm for the exponential function.

6.9 The Division Function

When developing the algorithm that performs the approximation of the division it is limited to the range according to Table 10, since the division is not strictly concave or convex outside this range. The pre- and post-processing part both needs computation when performing the approximation of the division.

	Algorithm	Range
Function	$f(v) = \frac{1}{1+v}$	$0.5 < v \le 1$
Preprocessing	$x = 2 \cdot (1 - v)$	$0 \le x < 1$
Processing	$y = \frac{6}{1 + \left(1 - \frac{x}{2}\right)} - 3$	$0 \le y < 1$
Postprocessing	$z = \frac{3+y}{6}$	$\frac{1}{2} \le z < \frac{2}{3}$

Table 10. The algorithm for the division function.

6.10 The Square Root Function

When developing the algorithm that performs the approximation of the square root function the range is limited according to Table 11. The pre- and post-processing part both needs computation when performing the approximation of the square root function.

	Algorithm	Range
Function	$f(v)=\sqrt{1+v}$	$1 \le v < 2$
Preprocessing	$x = v-1$	$0 \le x < 1$
Processing	$y=\frac{\sqrt{2+x}-\sqrt{2}}{\sqrt{3}-\sqrt{2}}$	$0 \le y < 1$
Postprocessing	$z=\sqrt{2}+y\cdot\left(\sqrt{3}-\sqrt{2}\right)$	$\sqrt{2} \le z < \sqrt{3}$

Table 11. The algorithm for the square root function.

7. Conclusions

A novel methodology for implementing approximations of unary functions such as trigonometric functions, logarithmic functions, as well as square root and division functions etc. in hardware is introduced. The architecture of the processing part automatically gives a high degree of parallelism. The methodology to develop the approximation algorithm is founded on parabolic synthesis. This combined with that the methodology is founded on operations that are simple to implement in hardware such as addition, shifts, multiplication, contributes to that the implementation in hardware is simple to perform. By using the parallelism and parabolic synthesis, one of the most important characteristics with the out coming hardware is the parallelism that gives a short critical path and fast computation. The structure of the methodology will also assure an area efficient hardware implementation. The methodology is also suitable for automatic synthesis.

8. References

B. Parhami (2000), *Computer Arithmetic*, Oxford University Press Inc., ISBN: 0-19-512583-5, 198 Madison Avenue, New York, New York 10016, USA.

E. Hertz, P. Nilsson (2008), A Methodology for Parabolic Synthesis of Unary Functions for Hardware Implementation, *Proc. of the 2nd International Conference on Signals, Circuits and Systems*, SCS08_9.pdf, pp 1-6, ISBN-13: 978-1-4244-2628-7, Hammamet, Tunisia, Nov. 2008, Inst. of Elec. and Elec. Eng. Computer Society, 445 Hoes Lane - P.O.Box 1331, Piscataway, NJ 08855-1331, United States.

E. Hertz, P. Nilsson (2009), Parabolic Synthesis Methodology Implemented on the Sine Function, *Proc. of the 2009 IEEE International Symposium on Circuits and Systems*, pp 253-256, ISBN: 978-1-4244-3828-0, Taipei, Taiwan, May. 2009.

E. Meijering (2002), A Chronology of Interpolation From Ancient Astronomy to Modern Signal and Image Processing, *Proceedings of the IEEE*, vol. 90, no. 3, March 2002, pp. 319-342, ISSN: 00189219, Institute of Electrical and Electronics Engineers Inc.

J. E. Volder (1959), The CORDIC Trigonometric Computing Technique, *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, 1959, pp. 330–334.

J. Fourier (1822), *Théorie Analytique de la Chaleur*, Paris, France.

J.-M. Muller (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.

K. Z. Pekmestzi, P. Kalivas, and N. Moshopoulos (2001), Long unsigned number systolic serial multipliers and squarers, *IEEE Circuits and Systems II:*, vol. 48, no 3, March 2001, pp. 316 -321, ISSN 1057-7130.

P. Ienne, M. A. Viredaz (1994), Bit-serial multipliers and squarers, *IEEE Transactions on Computer*, vol. 43, no12, Dec. 1994, pp. 1445 -1450, ISSN 0018-9340.

P.T.P. Tang (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, June 1991.

R. Andrata (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, Feb. 1998, ACM Inc.

VLSI

Edited by Zhongfeng Wang

ISBN 978-953-307-049-0
Hard cover, 456 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

The process of Integrated Circuits (IC) started its era of VLSI (Very Large Scale Integration) in 1970's when thousands of transistors were integrated into one single chip. Nowadays we are able to integrate more than a billion transistors on a single chip. However, the term "VLSI" is still being used, though there was some effort to coin a new term ULSI (Ultra-Large Scale Integration) for fine distinctions many years ago. VLSI technology has brought tremendous benefits to our everyday life since its occurrence. VLSI circuits are used everywhere, real applications include microprocessors in a personal computer or workstation, chips in a graphic card, digital camera or camcorder, chips in a cell phone or a portable computing device, and embedded processors in an automobile, et al. VLSI covers many phases of design and fabrication of integrated circuits. For a commercial chip design, it involves system definition, VLSI architecture design and optimization, RTL (register transfer language) coding, (pre- and post-synthesis) simulation and verification, synthesis, place and route, timing analyses and timing closure, and multi-step semiconductor device fabrication including wafer processing, die preparation, IC packaging and testing, et al. As the process technology scales down, hundreds or even thousands of millions of transistors are integrated into one single chip. Hence, more and more complicated systems can be integrated into a single chip, the so-called System-on-chip (SoC), which brings to VLSI engineers ever increasingly challenges to master techniques in various phases of VLSI design. For modern SoC design, practical applications are usually speed hungry. For instance, Ethernet standard has evolved from 10Mbps to 10Gbps. Now the specification for 100Mbps Ethernet is on the way. On the other hand, with the popularity of wireless and portable computing devices, low power consumption has become extremely critical. To meet these contradicting requirements, VLSI designers have to perform optimizations at all levels of design. This book is intended to cover a wide range of VLSI design topics. The book can be roughly partitioned into four parts. Part I is mainly focused on algorithmic level and architectural level VLSI design and optimization for image and video signal processing systems. Part II addresses VLSI design optimizations for cryptography and error correction coding. Part III discusses general SoC design techniques as well as other application-specific VLSI design optimizations. The last part will cover generic nano-scale circuit-level design techniques.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821