



Thesis for the degree of Licentiate in Engineering

Parabolic Synthesis

by

Erik Hertz

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

The Department of Electrical and Information Technology
Lund University
P.O. Box 118, S-221 00 LUND

ISBN 978-91-7473-069-2
ISSN 1654-790X
No. 28

© Erik Hertz 2011
Printed in Sweden by Tryckeriet E-huset, Lund.
January 2011

Abstract

Many consumer products, such as within the computer areas, computer graphics, digital signal processing, communication systems, robotics, navigation, astrophysics, fluid physics, etc. are searching for high computational performance as a consequence of increasingly more advanced algorithms in these applications. Until recently the down scaling of the hardware technology has been able to fulfill these higher demands from the more advanced algorithms with higher clock rates on the chips. This that the development of hardware technology performance has stagnated has moved the interest more over to implementation of algorithms in hardware. Especially within wireless communication the desire for higher transmission rates has increased the interest for algorithm implementation methodologies.

The scope of this thesis is mainly on the developed methodology of parabolic synthesis. The parabolic synthesis methodology is a methodology for implementing approximations of unary functions in hardware. The methodology is described with the criteria's that have to be fulfilled to perform an approximation on a unary function. The hardware architecture of the methodology is described and to this a special hardware that performs the squaring operation.

The outcome of the presented research is a novel methodology for implementing approximations of unary functions such as trigonometric functions, logarithmic functions, as well as square root and division functions etc. The architecture of the processing part automatically gives a high degree of parallelism. 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. The hardware architecture is characterized by a high degree of parallelism that gives a short critical path and fast computation. The structure of the methodology will also assure an area efficient hardware implementation.

Acknowledgments

Very few things in life you do by yourself and this thesis is just another example of how dependent one is of the support from a wide range of people, both near relatives, friends and colleges. Therefore, in commemoration, my deepest thanks go to my mother and father for their high beliefs in me. In memory I also want to thank my uncle Karl-Martin Linderoth for his unreserved interest in my work and his thoughtful guidance.

I also feel the greatest gratitude to my supervisor professor Peter Nilsson, who made it possible for me to restart my Ph.D. studies. In my research he always has had a devoted presence with continuously evolving and encouraging dialogs. I am also grateful for his determined search for new angles and applications for my research. To that I am over overwhelmed by his understanding and consideration knowing the difficulties with doing Ph.D. studies in your spare time.

To associate professor Clas Agnvall I also want to show my greatest appreciation for the all-out effort that he put in to reestablish me as a Ph.D. student. Finding loopholes in the rules that made it possible for me to start-up my Ph.D. studies again after many years of letting my studies lay in fallow.

My greatest appreciation goes to the administrative and technical staff at the department of Electrical and Information Technology, who has gone far beyond expectations in assisting me with numerous things in my research, travels and many other things. Especially I am very grateful for the excellent and foresighted help that I got in various ways from the administrative coordinator Pia Bruhn.

To present and past members of the Digital ASIC research group, Radio research group, many others at the department and also others at other departments I am very grateful for their help and support in all that surrounds the life of a student.

To all my friends I thank them for their support and for never giving up on our friendship even though I have for long periods been invisible.

Lund, January 28, 2011
Erik Hertz

Tallenes lyrik

En halv

er,

tænk nu hvor aparte,

to trediedele af tre kvarte.

Piet Hein

Table of Contents

Abstract	3
Acknowledgments	5
Table of Contents	9
Preface	11
1 Introduction	15
2 Hardware Approximation Methods	17
2.1 Lookup Table	17
2.1.1 Direct and Indirect Table Lookup	18
2.1.2 Lookup Table Reduction by Auxiliary Function	18
2.1.3 Interpolation	18
2.2 Polynomial Approximation	20
2.3 Piecewise Approximation	21
2.4 Sum of Bit-Product Approximation	22
2.5 CORDIC	23
3 Parabolic Synthesis	27
3.1 Normalizing	27
3.2 Developing the Hardware Architecture	28
3.3 Methodology for developing sub-functions	29
3.4 Hardware Implementation	34
3.4.1 Preprocessing	35
3.4.2 Processing	35
3.4.3 Postprocessing.....	39
4 Implementation of the sine function	41
4.1.1 Preprocessing	41
4.1.2 Processing	42
4.1.3 Optimization.....	44
4.1.4 Architecture.....	45
4.1.5 Optimization of Word Length.....	46
4.1.6 Precision.....	46
5 Using the Methodology	49
5.1.1 The Sine Function	49
5.1.2 The Cosine Function	50
5.1.3 The Arcsine Function.....	50
5.1.4 The Arccosine Function	51
5.1.5 The Tangent Function	51
5.1.6 The Arctangent Function	51

5.1.7	The Logarithmic Function	52
5.1.8	The Exponential Function.....	52
5.1.9	The Division Function	53
5.1.10	The Square Root Function.....	53
6	Results	55
7	Conclusions	57
8	Future Work.....	59
	References	61
	List of Figures.....	65
	List of Tables	67
	List of Acronyms	69

Preface

Papers contributing to this thesis

Erik Hertz and Peter Nilsson, "Parabolic Synthesis Methodology", In the proceedings of the 2010 GigaHertz Symposium, pp 35, March 9-10, 2010, Lund, Sweden.

Erik Hertz and Peter Nilsson, "A Methodology for Parabolic Synthesis", a book chapter in VLSI, pp 199-220, In-Tech, ISBN 978-953-307-049-0, Vienna, Austria, February 2010.

E. Hertz and P. Nilsson, "Parabolic Synthesis Methodology Implemented on the Sine Function", in Proceedings of the 2009 IEEE International Symposium on Circuits and Systems, pp 253-256, Taipei, May 24-27, 2009.

E. Hertz and P. Nilsson, "A Methodology for Parabolic Synthesis of Unary Function for Hardware Implementation", in Proceedings of the 2008 IEEE Conference on Signals, Circuits and System, pp 1-6, Hammamet, November 7-9, 2008.

E. Hertz and M. Torkelson, "Aritmetik anpassad för Digitala Signalbehandlingskretsar", in Proceedings for RadioVetenskap och Kommunikation 93, pp 9-12, Lund, April 5-7, 1993.

E. Hertz, "En höghastighets metod för digital aritmetik", in Proceedings of the first GigaHertz Symposium 1992, pp p2:k, Linköping, March 25-26, 1992.

E. Hertz and M. Torkelson, "Radar DSP-chip using Approximate Arithmetic, in Proceedings of the sixth NORSILC/NORCHIP seminar, pp 16.0-16.8, Copenhagen, October 23-24, 1991.

Patents contributing to this thesis

A Device for Conversion of a Binary Floating-Point Number into a Binary 2-Logarithm or the Opposite

Publication number: WO9533308

Publication date: 1995-12-07

Inventor(s): Hertz Erik [SE]

Applicant(s): Försvarets Forskningsanstalt [SE]; Hertz Erik [SE]

A Device for Conversion of a Binary Floating-Point Number into a Binary 2-Logarithm or the Opposite

Publication number: WO9414245

Publication date: 1994-06-23

Inventor(s): Hertz Erik [SE]

Applicant(s): Försvarets Forskningsanstalt [SE]; Hertz Erik [SE]

Papers not included in this thesis

Ulf Sjöström and E. Hertz, “VLSI Implementation of a Digital Array Antenna”, in Proceedings of the second GigaHertz Symposium 1994, pp ps-20, Linköping, March 22-23, 1994.

E. Hertz and M. Torkelson, “Comparison Between Two Design Concepts for a Radar DSP Chip”, in Proceedings of the 14:th Nordic Semiconductor Meeting, pp. 462-465, Aarhus, Denmark, June 17-20, 1990.

E. Hertz and M. Torkelson, “En VLSI-krets för radar applikation”, in Proceedings for RadioVetenskap och Kommunikation 90, pp 455-458, Gothenburg, April 9-11,1990.

Patents not included in this thesis

Radio Receiver

Publication number: AU2002237325

Publication date: 2002-09-24

Inventor(s): Hertz Erik [SE]; Canovas Joaquin [SE]; He Shousheng [SE]; Vicent Antonio [SE]

Applicant(s): L M Ericsson Telefon AB [SE]

Radio Receiver

Publication number: WO02073797

Publication date: 2002-09-19

Inventor(s): Hertz Erik [SE]; Canovas Joaquin [SE]; He Shousheng [SE]; Vicent Antonio [SE]

Applicant(s): L M Ericsson Telefon AB [SE]

Radio Receiver and Digital Filter Therefore

Publication number: GB2373385

Publication date: 2002-09-18

Inventor(s): Hertz Erik [SE]; Canovas Joaquin [SE]; He Shousheng [SE]; Vicent Antonio [SE]

Applicant(s): L M Ericsson Telefon AB [SE]

CHAPTER 1

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 [1]. 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 [2] were further studied by many others, including Stirling [3], Gauss [4], Waring [5], Euler [6], Lagrange [7], Bessel [8], Laplace [9] [10], and Everett [11] [12]. 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 [13] 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 [14], but also studied before him by Borel [15] and others, and eventually leading to the sampling theorem for band limited functions as found in the works of J. M. Whittaker [16] [17] [18], Kotel'nikov [19], Shannon [20], and several others, and secondly the concept of oscillatory interpolation, researched by many and eventually resulting in Schoenberg's theory [21] [22] of mathematical splines.

The parabolic synthesis methodology.

Unary functions, e.g. trigonometric functions, logarithms as well as square root and division functions are extensively used in computer graphics, digital signal processing, communication systems, robotics, navigation, 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 [23] is simple and fast which is straight 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 [24]. 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 [24].

The CORDIC (COordinate Rotation DIgital Computer) algorithm [25] [26] has been used for these applications since it is faster than a software approach. CORDIC is traditionally an iterative method and therefore slow which makes the method insufficient for this kind of applications.

This thesis proposes a methodology of parabolic synthesis [27] 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 unary functions a parabolic synthesis methodology is applied. Only low complexity operations that are simple to implement in hardware are used

CHAPTER 2

2 Hardware Approximation Methods

An unavoidable problem when implementing signal and image processing algorithms in hardware is to find realizations of elementary function approximations in hardware. With the increasing digitalization there is a growing request for hardware realization methods of approximations of elementary functions. In view of the increasing relevance it is natural that the subject of developing hardware realization methods of approximations of elementary functions gets more and more attention. This chapter will give a brief overview of existing hardware realization methods of approximations of mathematical defined elementary functions.

2.1 *Lookup Table*

Computation by lookup table is an attractive VLSI realization method because a memory simplifies the implementation of random logic in hardware [28]. Also with improving memory density, multimegabit lookup tables can become more practical in some applications. A benefit with lookup tables is that it reduces the costs of hardware development when it comes to design, validation and testing. It will also provide more flexibility for design changes and reduce the number of different building blocks or modules required when implementing arithmetic system designs.

When storing tables in read-only memories the benefit is also an improved reliability since memories are more robust than combinational logic circuits. By exchanging to read/write memories and reconfigurable peripheral logic it will facilitate simpler evaluation of different functions as well as simplifying the maintenance and repair of a design.

2.1.1 Direct and Indirect Table Lookup

The straight forward usage of lookup table is a direct table lookup. When given an m -variable function $f(x_{m-1}, x_{m-2}, \dots, x_1, x_0)$, the evaluation of f when the input values is u -bit and the desired result is v -bit then the required table construction is $2^u \times v$. The concatenated u -bit string is then used as an address into the table, with the read out v -bit value from the table and directly forwarded to the output. This arrangement gives a high degree of flexibility but is not practical in most cases. When implementing unary functions the size of the table can be manageable when the input operand is up to 12 to 16 bits, which gives a table size of 4K to 65K words. To implement binary functions, such as xy , $x \bmod y$ or x^y , in lookup tables the input operand has to be shorten to half of size of the operand size in the previous case to be manageable. This since the growth of the table size is exponential, which becomes intolerable when implementing in hardware. With increasing size the logic latency of the memory will also contribute to be a problem for many applications.

To reduce the consequences of the exponential growth of the table size preprocessing steps of the operands and postprocessing steps [23] of the values read out from the tables can be introduced and thereby perform an indirect table lookup. When implementing this hybrid scheme the elements in the pre- and postprocessing parts are both simpler and faster and may also be more cost-effective than either a pure table lookup approach or a pure logic circuit implementation based on an algorithm.

2.1.2 Lookup Table Reduction by Auxiliary Function

An approach to reduce the table size is to develop a binary function as an auxiliary unary function. With different complexity of the auxiliary function the size of the lookup table can be reduced to different extents. These auxiliary functions are used in both the pre- and postprocessing steps [23] of the lookup table. The reduction of the table size for a given resolution can be rather significant and can also allow pipelining of the design, which increases the throughput.

2.1.3 Interpolation

A simple method to reduce the size of the lookup table is to use linear interpolation when implementing a function [28]. When a function $f(x)$ is known for $x = x_{lo}$ and $x = x_{hi}$, where $x_{lo} < x_{hi}$, the values for x in the interval

$[x_{lo}, x_{hi}]$ can be computed from $f(x_{lo})$ and $f(x_{hi})$ by interpolation. The simplest method to perform an interpolation is to use linear interpolations where $f(x)$ for x in $[x_{lo}, x_{hi}]$ is computed according to (1).

$$f(x) = f(x_{lo}) + (x - x_{lo}) \cdot \frac{[f(x_{hi}) - f(x_{lo})]}{x_{hi} - x_{lo}} \quad (1)$$

When implementing linear interpolation in hardware two lookup tables are needed, one for the starting point of the interpolation $a = f(x_{lo})$, and one for the direction coefficient $b = [f(x_{hi}) - f(x_{lo})]/(x_{hi} - x_{lo})$, as shown in Figure 1 for the initial linear approximation. When choosing x_{lo} and x_{hi} it is beneficial to express them as numbers in the form of 2 to the power of z , since the addressing of the lookup tables and computation of the linear interpolation can be simplified. This will do that the addressing of the lookup tables will be in the most significant part of x and the subtraction for $\Delta x = (x - x_{lo})$ will be in the least significant part of x . This do that the hardware realization of the approximation can be simplified to $a + b \cdot \Delta x$ where a and b fetched from memories for the interval in which the computation is to be performed.

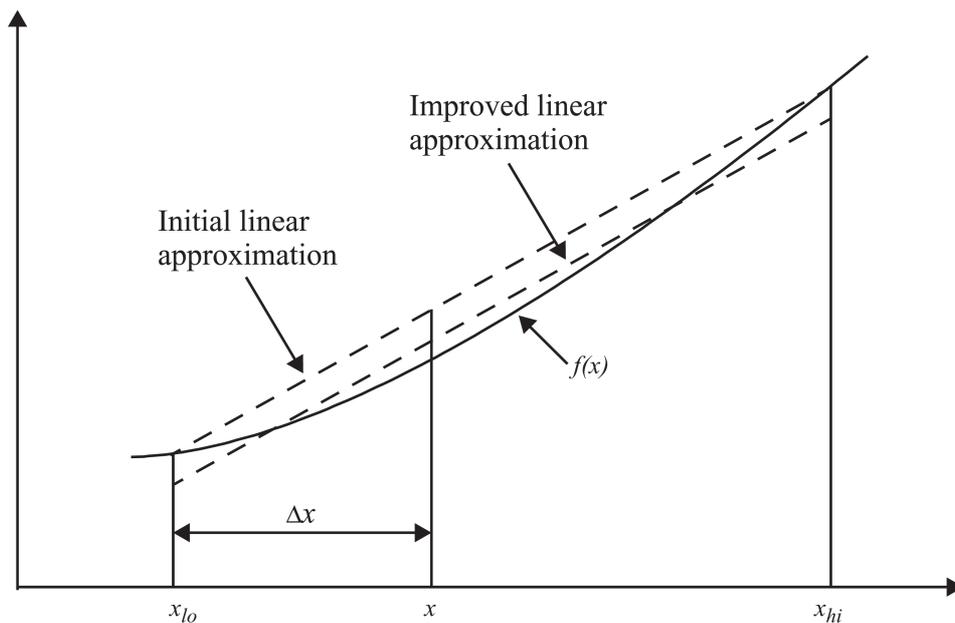


Figure 1. Linear interpolation for computing $f(x)$.

To minimize the absolute and the relative error in the worst case an improved linear approximation strategy, as shown in Figure 1, can be chosen. As shown in Figure 1 the starting point a and the direction coefficient b is chosen to minimize the error of the interpolation in the interval.

2.2 Polynomial Approximation

Since polynomials only involve additions, subtractions, multiplications and comparisons it is natural to approximate elementary functions with polynomials. To ensure the computation efficiency the multiplier is the most crucial part of the implementation. To choose a fast multiplier is therefore important for the efficiency of the computation of the polynomial approximation. For polynomial approximations there are a various number of polynomial schemes available. The chosen polynomial scheme affects the number of terms included for a given precision and thus the computational complexity.

TABLE I. APPROXIMATION SCHEMES.

Taylor Polynomial
Maclaurin Polynomial
Legendre Polynomial
Chebyshev Polynomial
Jacobi Polynomial
Laguerre Polynomial

When developing polynomial approximations the challenge is to develop an efficient approximation that conforms to the function to be approximated in the desired interval. When developing an approximation two development strategies are available, one to minimize the average error, called least squares approximations, and one to minimize the worst case error, called least maximum approximations to the function to be approximated [24]. The strategy to choose depends on the requirements of the design. When the requirements of the design is that the error of the approximation is to get the best fitting to the function to be approximated, least squares approximations is favorable. An example when least squares approximations are favorable is when the approximation is used in a series of computations. Least maximum approximations are favorable when it is important that the maximum error to the function to be approximated is important to keep small. An example when least maximum approximations

are favorable is when the error from the approximation has to be within a limit from the function to be approximated.

A list of commonly known approximation schemes is given in Table I. For additional information the reader can consult Muller's book [24].

2.3 Piecewise Approximation

Piecewise polynomial approximation [29] is a more flexible method to approximate a function $f(x)$ since the interval to perform an approximation is divided into a number of subintervals, which are implemented as piecewise approximations by polynomials of a low degree. These piecewise-polynomial approximations are called splines, and the endpoint of a subinterval is known as knots. An example of a linear spline approximation is shown in Figure 2.

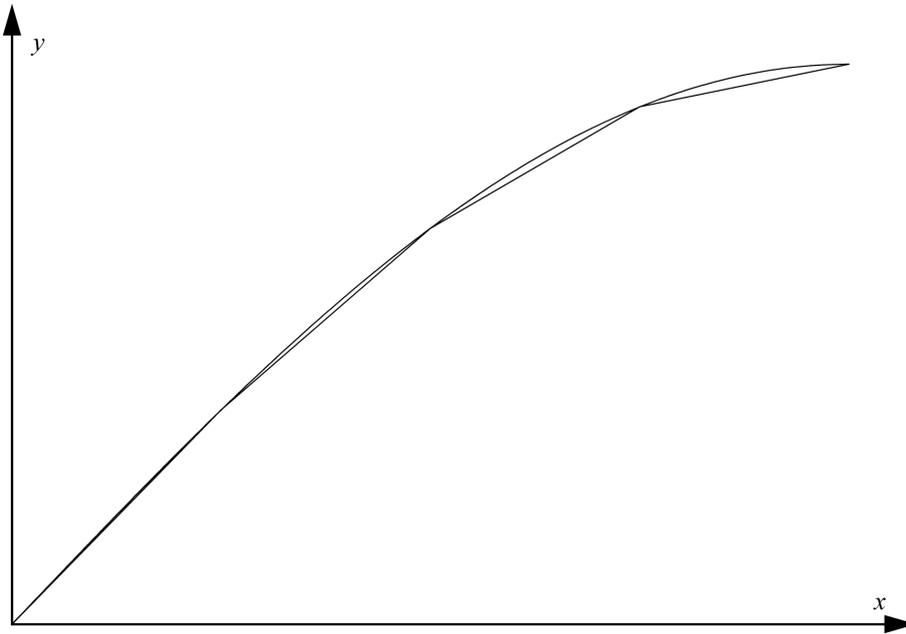


Figure 2. Example of a linear spline approximation.

The definition of a spline of degree n , $n \geq 1$, is a polynomial function of the degree n or less in each subinterval and has a prescribed degree of

smoothness. The spline is expected to be continuous and also have a continuous derivatives of the order up to k , $0 \leq k < n$.

In real-time applications performing elementary functions in dedicated hardware, software routines are often too slow and computationally intensive [30]. When using piecewise approximation methods in hardware implementations the degree of spline used is therefore often limited to first order approximations, also called linear approximations.

When developing hardware design much effort is put on reducing the delay of the computation and the chip area. To exclude multipliers is therefore interesting since excluding multipliers will reduce the execution time and the chip area of the implementation. An example of this is shown in [31] where the multipliers have been replaced with configurable shifts.

2.4 Sum of Bit-Product Approximation

For implementation of elementary functions in hardware the approximation method of sum of bit-products [32] can be beneficial since it can give an area efficient implementation with a high throughput and reasonable accuracy.

When formulating the approach, given a function, $f(X)$, where X is composed of N bits, as shown in (2).

$$X = \sum_{i=0}^{N-1} x_i 2^i \quad (2)$$

This gives the function 2^N function values, $f(x)$, where $0 \leq X \leq 2^N-1$. The functions values are then computed as a weighted sum of bit-products as shown in (3).

$$f(X) = \sum_{j=0}^{2^N-1} c_j p_j \quad (3)$$

In (Y) c_j corresponds to the weight and p_j to the bit-products there each bit-product, p_j , is composed of the bits x_j that are one in the binary representation of j .

2.5 CORDIC

The CORDIC (COordinate Rotation DIgital Calculation) algorithm is an iterative algorithm able of evaluating a numeral of elementary functions only using shifts and additions. The algorithm is therefore very appealing for hardware implementation compared to software approaches [33] [26] using multiplications and additions. The CORDIC algorithm was introduced by Volder [25] and later generalized by Walther [34].

The CORDIC algorithm approximates the sine, cosine, etc. of an angle iteratively, using only simple mathematical operations such as additions, shifts and table lookups. The algorithm is a traditionally an iterative process, consisting of micro-rotations where the initial vector is rotated by predetermined step angles. However, unrolled architectures can be beneficial for hardware as well [35] [36]. Any angle can be represented to certain accuracy by a set of predetermined angle steps.

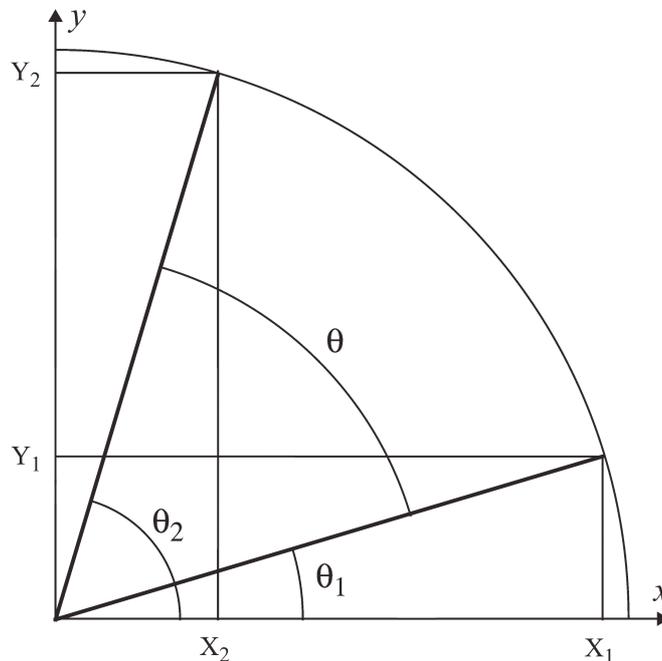


Figure 3. Vector rotation.

In Figure 3, the principle idea of CORDIC is shown. Rotation of a vector (X_1, Y_1) by the angle θ to (X_2, Y_2) is computed according to (4).

$$\begin{aligned} X_2 &= (X_1 \cdot \cos(\theta)) - (Y_1 \cdot \sin(\theta)) \\ Y_2 &= (X_1 \cdot \sin(\theta)) + (Y_1 \cdot \cos(\theta)) \end{aligned} \quad (4)$$

The functions in (4) can be rearranged according to (5).

$$\begin{aligned} X_2 &= \cos(\theta) \cdot [X_1 - (Y_1 \cdot \tan(\theta))] \\ Y_2 &= \cos(\theta) \cdot [(X_1 \cdot \tan(\theta)) + Y_1] \end{aligned} \quad (5)$$

To generalize (5) it can be further rearranged according to (6).

$$\begin{aligned} X_{i+1} &= \cos(\theta_i) \cdot [X_i - (Y_i \cdot \tan(\theta_i))] \\ Y_{i+1} &= \cos(\theta_i) \cdot [(X_i \cdot \tan(\theta_i)) + Y_i] \end{aligned} \quad (6)$$

The methodology is to compute the rotation angles θ , in steps i , where each step size is $\tan(\theta) = \pm 2^{-1}$. Through iterative rotations the vector will rotate in one or the other direction by decreasing steps until the desired angle is achieved. The iterative rotation method is based on that the rotation from θ_i to θ_{i+1} is done in carefully chosen steps with values that imply that the operations used are only shifts and additions. The values for θ_i are therefore chosen such that $\tan(\theta_i)$ is a fractional number expressed in trivial numbers i.e. numbers to the 2 power. The multiplication by $\tan(\theta_i)$ can thus be replaced by a simple right-shift operation.

If the iterations of (6) are analyzed the cosine factors K_i shown in (7), falls out as an iterative product series.

$$K_i = \prod_{i=0}^{\infty} \frac{2^i}{\sqrt{1 + 2^{2i}}} \quad (7)$$

Equation (7) is referred to as a scale factor, which represents the decrease in magnitude of the vector during the rotation process. When the number of iterations/rotations is approaching infinity the scale factor approaches the value 0.607253. The $\cos(\theta_i)$ term in (6) can therefore be replaced with the scale factor K_i , as shown in (8).

$$X_{i+1} = K_i \cdot [X_i - (Y_i \cdot d_i \cdot 2^{-i})] \quad (8)$$

$$Y_{i+1} = K_i \cdot [(X_i \cdot d_i \cdot 2^{-i}) + Y_i]$$

As previous described in the iterative process the vector will rotate in one or the other direction which do that the direction of the rotation d_i , must be introduced in (8).

To keep track of the angle that has been rotated, (9) is introduced.

$$\theta_{i+1} = \theta_i - d_i \arctan(2^{-i}) \quad (9)$$

The direction of the rotation d_i , is defined according to (10).

$$d_i = \begin{cases} -1 & \text{if } \theta_i < 0 \\ 1 & \text{otherwise} \end{cases} \quad (10)$$

Since the first rotation of the vector would be rotated 45° counterclockwise it is convenient to set the initial angle θ_0 to 0.

3 Parabolic Synthesis

Unary functions, e.g. trigonometric functions, logarithms as well as square root and division functions are extensively used in computer graphics, digital signal processing, communication systems, robotics, navigation, astrophysics, fluid physics, etc. For these high-speed applications, software solutions are in many cases not sufficient and a hardware implementation is therefore needed.

The proposed methodology of parabolic synthesis [27] 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 unary functions a parabolic synthesis methodology is applied. Only low complexity operations that are simple to implement in hardware are used.

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 necessary, a preprocessing normalization and a postprocessing transformation step as described by [23] [24]. The computation is therefore divided into three steps, normalizing, approximation and transforming.

3.1 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 $0 \leq x < 1$ on the x -axis and $0 \leq y < 1$ 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, here called an original function $f_{org}(x)$, is shown in Figure 4.

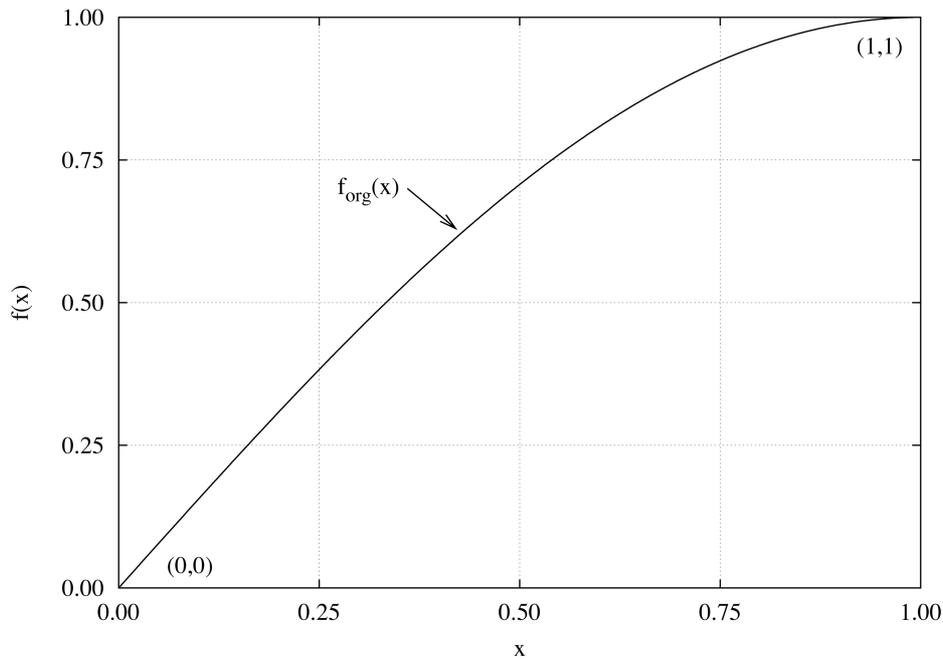


Figure 4. Example of normalized function, in this case $\sin\left(\frac{\pi \cdot x}{2}\right)$.

3.2 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 [37] 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 synthesized through 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 (11), obtains to the original function $f_{org}(x)$. When developing the approximate function, the accuracy depends on the number of sub-functions used.

$$f_{org}(x) = s_1(x) \cdot s_2(x) \cdot \dots \cdot s_{\infty}(x) \quad (11)$$

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 a parabolic looking function called the first help-function $f_1(x)$, as shown in (12).

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} \quad (12)$$

The first sub-function $s_1(x)$, will be chosen to be feasible for hardware, according to the methodology. In the same manner the following help-functions $f_n(x)$, are generated, as shown in (13).

$$f_{n+1}(x) = \frac{f_n(x)}{s_{n+1}(x)} \quad (13)$$

It is important that the sub-functions $s_n(x)$, are chosen to be feasible for hardware realization. The purpose with the normalization is to facilitate the hardware implementation by limiting the numerical range.

3.3 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 reasonably simple to implement in hardware i.e. only low complexity operations such as additions and multiplications are used.

The first sub-function

The sub-function $s_I(x)$, is developed by dividing the original function $f_{org}(x)$, with x as a first order approximation. As shown in Figure 5 there are two possible results after dividing the original function with x , one where $f(x) > 1$ and one where $f(x) < 1$.

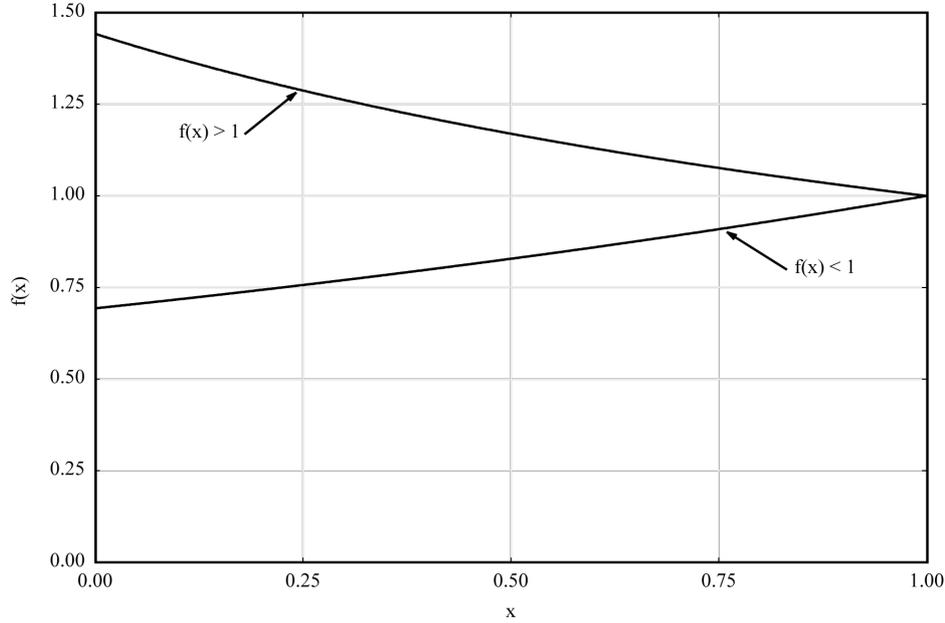


Figure 5. Two possible results after dividing an original function with x .

The first sub-function $s_I(x)$, is shown in (14). To approximate these functions the expression $1+(c_I \cdot (1-x))$ is used. The first sub-function $s_I(x)$, is given by a multiplication of x and $1+(c_I \cdot (1-x))$, which is resulting in a second order parabolic function according to (14).

$$s_I(x) = x \cdot (1 + (c_I \cdot (1 - x))) = x + (c_I \cdot (x - x^2)) \quad (14)$$

In (14) the coefficient c_I is determined as the limit from the division of the original function with x and subtracted with 1, according to (15).

$$c_I = \lim_{x \rightarrow 0} \frac{f_{org}(x)}{x} - 1 \quad (15)$$

The second sub-function

The first help-function $f_1(x)$, is calculated according to (12) and dividing two functions that are both continuous convex or concave and with the same start and ending point will result in a function with an appearance similar to a parabolic function, as shown in Figure 6.

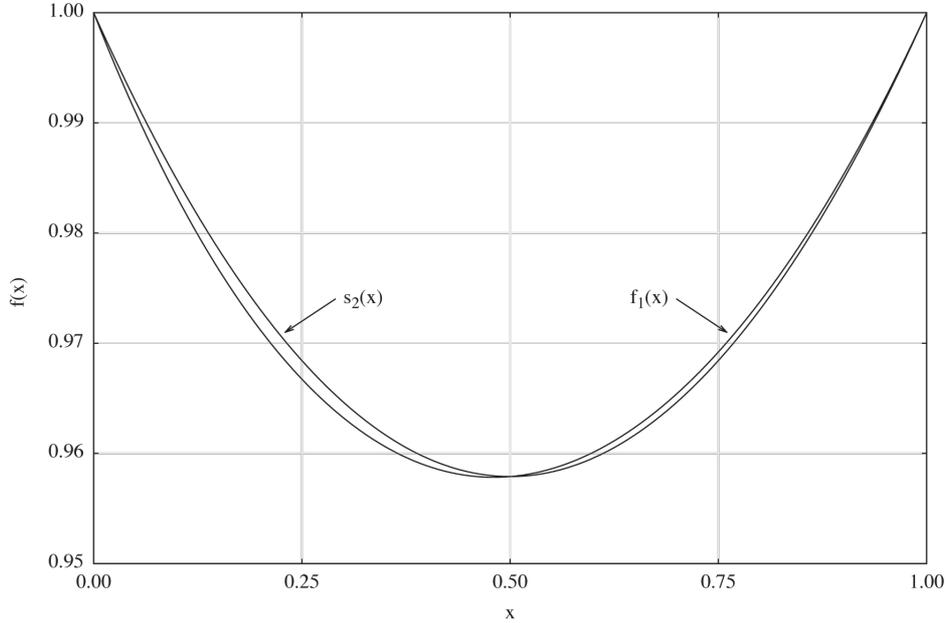


Figure 6. Example of the first function $f_1(x)$ compared with sub-function $s_2(x)$.

The second sub-function $s_2(x)$, is chosen according to the methodology as a second order parabolic function, see (16).

$$s_2(x) = 1 + (c_2 \cdot (x - x^2)) \quad (16)$$

In (16) 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 set to 0.5 in (17).

$$c_2 = 4 \cdot \left(f_1\left(\frac{1}{2}\right) - 1 \right) \quad (17)$$

When developing the second help-function $f_2(x)$, will result in that the new help-function can be divided into a pair of parabolic looking functions as shown in Figure 7, where the first interval are from $0 \leq x < 0.5$ and second interval from $0.5 \leq x < 1.0$.

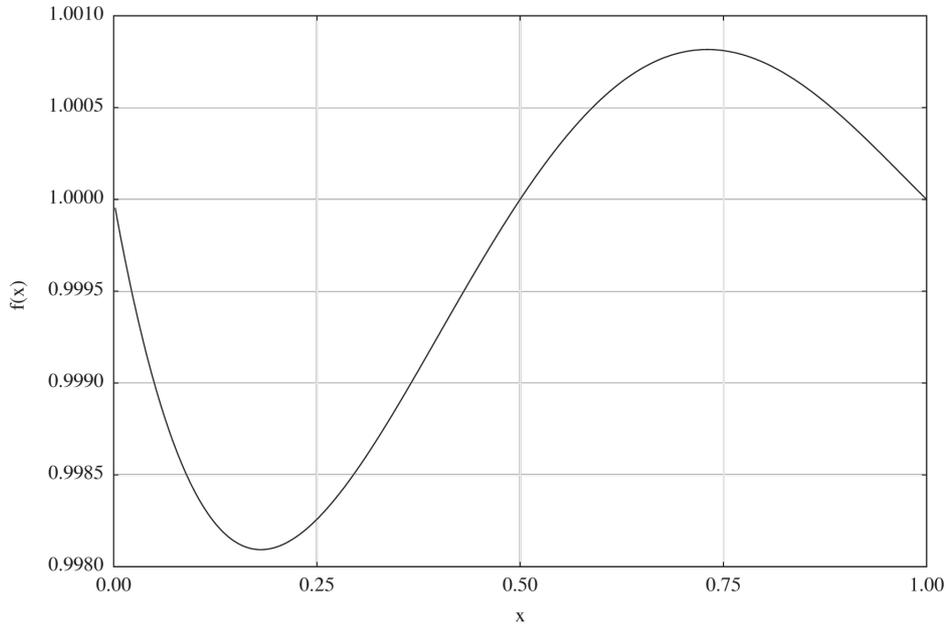


Figure 7. Example of the second function $f_2(x)$.

The size of the sub-intervals are chosen as a 2 power since the normalization of the interval can be performed as a left shift of x where the fractional part is the normalization of the two new intervals and the integer part is the addressing of the coefficients for the intervals, used in the hardware implementation as shown in section 4.1.4 Architecture.

Sub-functions when $n > 2$

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

for, the subscript index is increased with the index m , which gives the following appearance of the sub-help-function $f_{n,m}(x)$. In equation (18) it is shown how the help-function $f_n(x)$, is divided into sub-help-functions $f_{n,m}(x)$, when $n > 2$.

$$f_n(x) = \begin{cases} f_{n,0}(x), & 0 \leq x < \frac{1}{2^{n-1}} \\ f_{n,1}(x), & \frac{1}{2^{n-1}} \leq x < \frac{2}{2^{n-1}} \\ \dots \\ f_{n,2^{n-1}-1}(x), & \frac{2^{n-1}-1}{2^{n-1}} \leq x < 1 \end{cases} \quad (18)$$

As shown in (18), the numbers of sub-help-functions are doubled for each order of $n > 1$ i.e. the numbers of sub-help-functions are 2^{n-1} . From these sub-help-functions, the corresponding sub-sub-functions are developed. In analogy to the help-function $f_n(x)$, the sub-function $s_{n+1}(x)$, will have sub-sub-functions $s_{n+1,m}(x)$. In (19) it is shown how the sub-function $s_n(x)$, is divided into sub-sub-functions when $n > 2$.

$$s_n(x) = \begin{cases} s_{n,0}(x), & 0 \leq x < \frac{1}{2^{n-2}} \\ s_{n,1}(x), & \frac{1}{2^{n-2}} \leq x < \frac{2}{2^{n-2}} \\ \dots \\ s_{n,2^{n-2}-1}(x), & \frac{2^{n-2}-1}{2^{n-2}} \leq x < 1 \end{cases} \quad (19)$$

Note that in (19), the sub-sub-functions to the sub-functions; x has been changed to x_n . The change to x_n is normalized 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 ($frac(\cdot)$) of x , as shown in (20).

$$x_n = \text{frac}(2^{n-2} \cdot x) \quad (20)$$

As in the second sub-function $s_2(x)$, the second order parabolic function is used as an approximation of the interval of the help-function $f_{n-1}(x)$, as shown in (21).

$$s_{n,m}(x) = 1 + (c_{n,m} \cdot (x_n - x_n^2)) \quad (21)$$

Where the coefficients $c_{n,m}$ is computed according to (22).

$$c_{n,m} = 4 \cdot \left(f_{n-1,m} \left(\frac{2 \cdot (m+1) - 1}{2^{n-1}} \right) - 1 \right) \quad (22)$$

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

3.4 Hardware Implementation

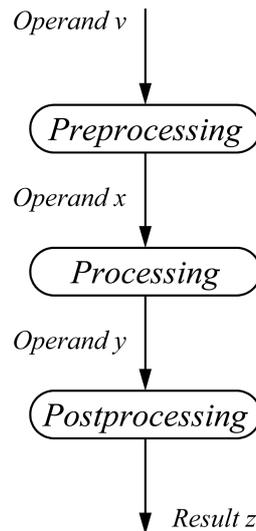


Figure 8. The hardware architecture of the methodology.

For the hardware implementation two's complement representation [28] is used. The implementation is divided into three hardware parts,

preprocessing, processing, and postprocessing as shown in Figure 8, which was introduced by P.T.P. Tang [24].

3.4.1 Preprocessing

In this part the incoming operand v is normalized to prepare the input to the processing part, according to section 3.1. If the approximation is implemented as a block in a larger system, the preprocessing part can be taken into consideration in the previous blocks, which implies that the preprocessing part can be reduced or even excluded.

3.4.2 Processing

In the processing part the approximation of the original function is directly computed in either iterative or parallel hardware architecture. The three equations (14), (16) and (21) has the same structure, which leads to that the approximation can be implemented as an iterative architecture as shown in Figure 9.

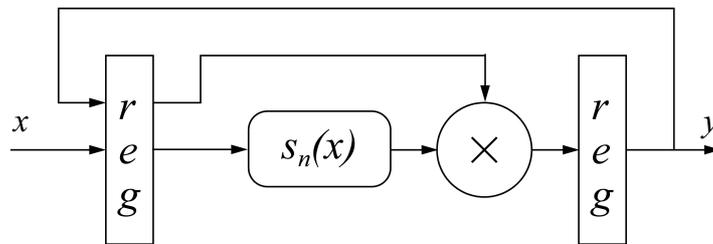


Figure 9. The principle of iterative hardware architecture.

The benefit of the iterative architecture is the small chip area whereas the disadvantage is longer computation time. The advantages with parallel hardware architectures are that they give 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 Figure 10.

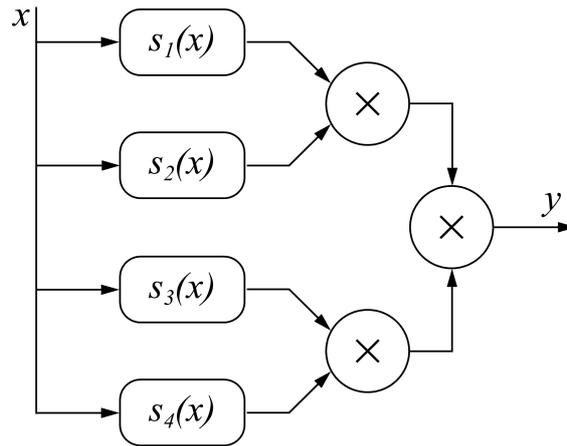


Figure 10. The architecture principle for four sub-functions.

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

In the sub-functions (14), (16) and (21) 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 Figure 11 the algorithm that performs the squaring and delivers partial product of x_n^2 is described.

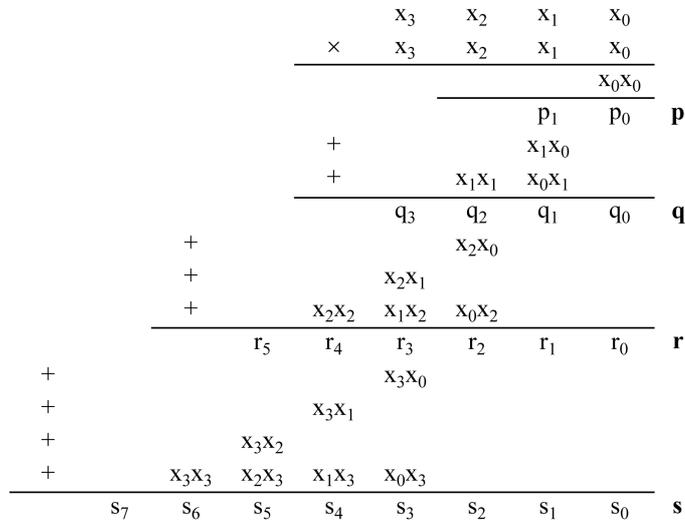


Figure 11. Squaring algorithm for the partial product x_n^2 .

$$r_3 = q_3 \cdot 2^3 + x_2x_1 \cdot 2^3 + x_1x_2 \cdot 2^3 + x_2x_0 \cdot 2^3 = q_3 \cdot 2^3 + x_2x_1 \cdot 2^4 + x_2x_0 \cdot 2^3 = q_3 \cdot 2^3 + x_2x_0 \cdot 2^3 \quad (27)$$

The result of the component r_4 in the vector r is simplified accordingly to (28).

$$r_4 = x_2x_1 \cdot 2^4 + x_2x_2 \cdot 2^4 = x_2 \cdot 2^4 + x_2x_1 \cdot 2^4 \quad (28)$$

The result of the component s_3 in the vector s is simplified accordingly to (29).

$$s_3 = r_3 \cdot 2^3 + x_3x_0 \cdot 2^3 + x_0x_3 \cdot 2^3 = r_3 \cdot 2^3 + x_3x_0 \cdot 2^4 = r_3 \cdot 2^3 \quad (29)$$

The result of the component s_4 in the vector s is simplified accordingly to (30).

$$s_4 = r_4 \cdot 2^4 + x_3x_0 \cdot 2^4 + x_3x_1 \cdot 2^4 + x_1x_3 \cdot 2^4 = r_4 \cdot 2^4 + x_3x_0 \cdot 2^4 + x_3x_1 \cdot 2^5 = r_3 \cdot 2^3 + x_3x_0 \cdot 2^4 \quad (30)$$

The result of the component s_5 in the vector s is simplified accordingly to (31).

$$s_5 = r_5 \cdot 2^5 + x_3x_1 \cdot 2^5 + x_3x_2 \cdot 2^5 + x_2x_3 \cdot 2^5 = r_5 \cdot 2^5 + x_3x_1 \cdot 2^5 + x_3x_2 \cdot 2^6 = r_5 \cdot 2^5 + x_3x_1 \cdot 2^5 \quad (31)$$

The result of the component s_6 in the vector s is simplified accordingly to (32).

$$s_6 = x_3x_2 \cdot 2^6 + x_3x_3 \cdot 2^6 = x_3 \cdot 2^6 + x_3x_2 \cdot 2^6 \quad (32)$$

In Figure 11 and Figure 12, the squaring algorithm that performs the partial products x_n^2 , is 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 Figure 11 and Figure 12, it was found that the resemblance to a bit-serial squarer [38] [39] 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 since only an addition is needed when computing each new partial product.

From (14), (16) and (21) it is found that only the coefficients values differs 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 in the desired precision can be both positive and negative. Especially, if the value of the approximation is less than the desired precision, an increased word length compared with the word length needed to accomplish the desired precision, might be necessary. 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 (17) or $c_{n,m}$ in (22). 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.

3.4.3 Postprocessing

The postprocessing part transforms the value to the output result z , as shown in Figure 8. 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 reduced or excluded. The main reason for the post processing is thus to transform the output to a feasible form for the proceeding parts in the system.

CHAPTER 4

4 Implementation of the sine function

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

4.1.1 Preprocessing

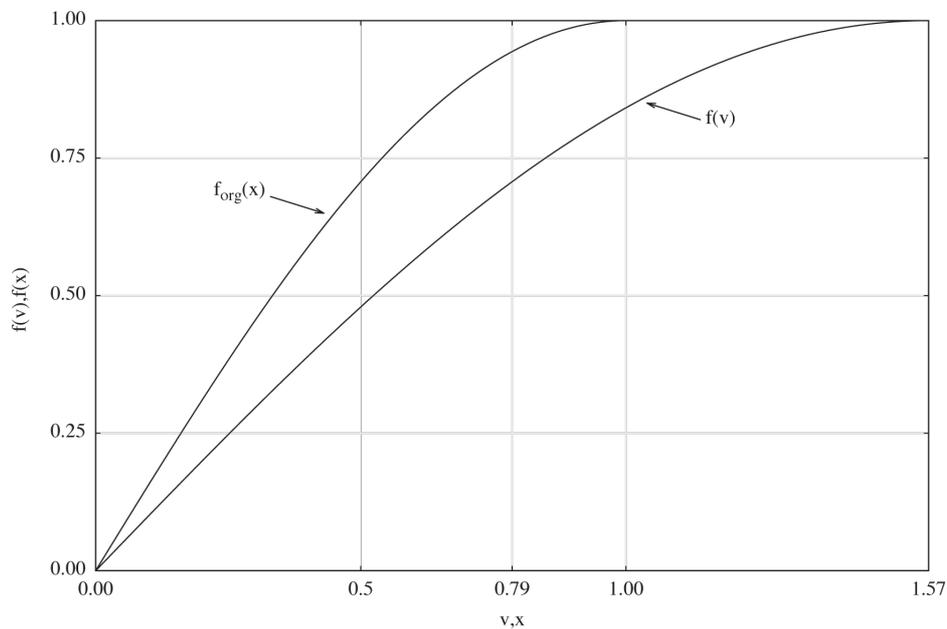


Figure 13. The function $f(v)$ before normalization and the original function $f_{org}(x)$.

To satisfy that the values of the incoming operand x is in the interval $0 \leq x < 1$, the operand x is multiplied with $\pi/2$ as shown in (33).

$$v = \frac{\pi}{2} \cdot x \quad (33)$$

To normalize the function $f(v) = \sin(v)$ v is substituted with x which gives the original function $f_{org}(x)$ (34).

$$f_{org}(x) = \sin\left(\frac{\pi}{2} \cdot x\right) \quad (34)$$

In Figure 13, the $f(v)$ function is shown together with the original function $f_{org}(x)$.

4.1.2 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 (15), here repeated for the sine function.

$$c_1 = \lim_{x \rightarrow 0} \frac{f_{org}(x)}{x} - 1 = \frac{\pi}{2} - 1 \quad (15)$$

The determined value, using the c_1 coefficient, is shown in (35).

$$s_1(x) = x + (c_1 \cdot (x - x^2)) = x + \left(\left(\frac{\pi}{2} - 1 \right) \cdot (x - x^2) \right) = 0.570796 \quad (35)$$

The first help-function $f_1(x)$, is computed as shown in (36).

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} \quad (36)$$

To develop the second sub-function $s_2(x)$, the coefficient c_2 is defined according to (17), here repeated again for the sine function

$$c_2 = 4 \cdot \left(f_1\left(\frac{1}{2}\right) - 1 \right) = 4 \cdot \left(\frac{f_{org}\left(\frac{1}{2}\right)}{s_1\left(\frac{1}{2}\right)} - 1 \right) = 0.400858 \quad (17)$$

The determined value of the second coefficient is numerically shown in (37).

$$s_1(x) = 1 + \left(c_2 \cdot (x - x^2) \right) = 1 + \left(0.400858 \cdot (x - x^2) \right) \quad (37)$$

The second help-function $f_2(x)$, is computed as shown in (38).

$$f_2(x) = \frac{f_1(x)}{s_2(x)} \quad (38)$$

To develop the third sub-functions $s_3(x)$, the second help-function $f_2(x)$, is divided into its two sub-help-functions as shown in (18). The third order of sub-functions is thereby divided into two sub-sub-functions, where $s_{3,0}(x_3)$ is restricted to the interval $0 \leq x < 0.5$ and $s_{3,1}(x_3)$ is restricted to the interval $0.5 \leq x < 1.0$ according to (19). A normalization of x to x_3 is done to simplify in the implementation in hardware, which is described in (20).

For each sub-function, the corresponding coefficients $c_{3,0}$ and $c_{3,1}$ is determined. These coefficients are determined according to (22) where higher order sub-functions can be developed. The determined values of the coefficients are shown in (39).

$$\begin{aligned} s_{3,0}(x_3) &= 1 + \left(c_{3,0} \cdot (x_3 - x_3^2) \right) = 1 + \left(-0.0122452 \cdot (x_3 - x_3^2) \right), \quad 0 \leq x < 0.5 \\ s_{3,1}(x_3) &= 1 + \left(c_{3,1} \cdot (x_3 - x_3^2) \right) = 1 + \left(0.0105947 \cdot (x_3 - x_3^2) \right), \quad 0.5 \leq x < 1 \end{aligned} \quad (39)$$

The third help-function $f_3(x)$, is computed as shown in (40).

$$f_3(x) = \frac{f_2(x)}{s_3(x)} \quad (40)$$

To develop the fourth sub-functions $s_4(x)$, the third help-function $f_3(x)$, is divided into its four sub-sub-functions as shown in (18). The fourth order of sub-functions is thereby divided into four sub-sub-functions, where $s_{4,0}(x_4)$ is restricted to the interval $0 \leq x < 0.25$, $s_{4,1}(x_4)$ is restricted to the interval $0.25 \leq x < 0.5$, $s_{4,2}(x_4)$ is restricted to the interval $0.5 \leq x < 0.75$ and $s_{4,3}(x_4)$ is restricted to the interval $0.75 \leq x < 1.0$ according to (19). A normalization of x to x_4 is done here as well, to simplify the hardware implementation, which is described in (20).

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 (22) which accomplish that higher order of sub-functions can be developed. The determined values of the coefficients are shown in (41).

$$\begin{aligned}
 s_{4,0}(x_4) &= 1 + \left(-0.00223363 \cdot (x_4 - x_4^2)\right), & 0 \leq x < 0.25 \\
 s_{4,1}(x_4) &= 1 + \left(0.00192558 \cdot (x_4 - x_4^2)\right), & 0.25 \leq x < 0.5 \\
 s_{4,2}(x_4) &= 1 + \left(-0.00119216 \cdot (x_4 - x_4^2)\right), & 0.5 \leq x < 0.75 \\
 s_{4,3}(x_4) &= 1 + \left(0.00126488 \cdot (x_4 - x_4^2)\right), & 0.75 \leq x < 1
 \end{aligned} \tag{41}$$

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

4.1.3 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 $c_{4,3}$. The precision is improved by adjusting the coefficient of the sub-sub-function with the lowest precision in a manner that the sub-sub-function reaches the minimum error of the function. The optimization of the error can favorably be performed in software. As later described by Figure 15, sub-sub-function $s_{4,3}(x)$ in the interval $0.75 \leq x < 1.0$ has the largest relative error, here in floating point but also in fixed point. When performing an optimization of sub-sub-function $s_{4,3}(x)$ in the interval $0.75 \leq x < 1.0$ it was found that the word length in the computations could be reduced from 17 to 16 bits.

4.1.4 Architecture

In Figure 14, the architecture of the approximation of the sine function using the proposed methodology is shown. The x^2 block in Figure 14, is a specially designed multiplier described in Figure 11 and Figure 12 that performs the partial result u , i.e. u_3 and u_4 , are used in the following blocks. In the $x-u$ block, x is subtracted with the partial result u , from the x^2 block. The result w from the $x-u$ block is then used in the two following blocks as shown in Figure 14. In the $x+(c_1 \cdot w)$ block $s_1(x)$ is performed, in $1+(c_2 \cdot w)$ $s_2(x)$ is performed, in $1+(c_3 \cdot (x_3-u_3))$ $s_3(x)$ is performed, and in $1+(c_4 \cdot (x_4-u_4))$ $s_4(x)$ 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, i.e. the most significant bits of x not used in x_3 and x_4 , represented by i_0 and i_1 in Figure 14, controls the MUXes.

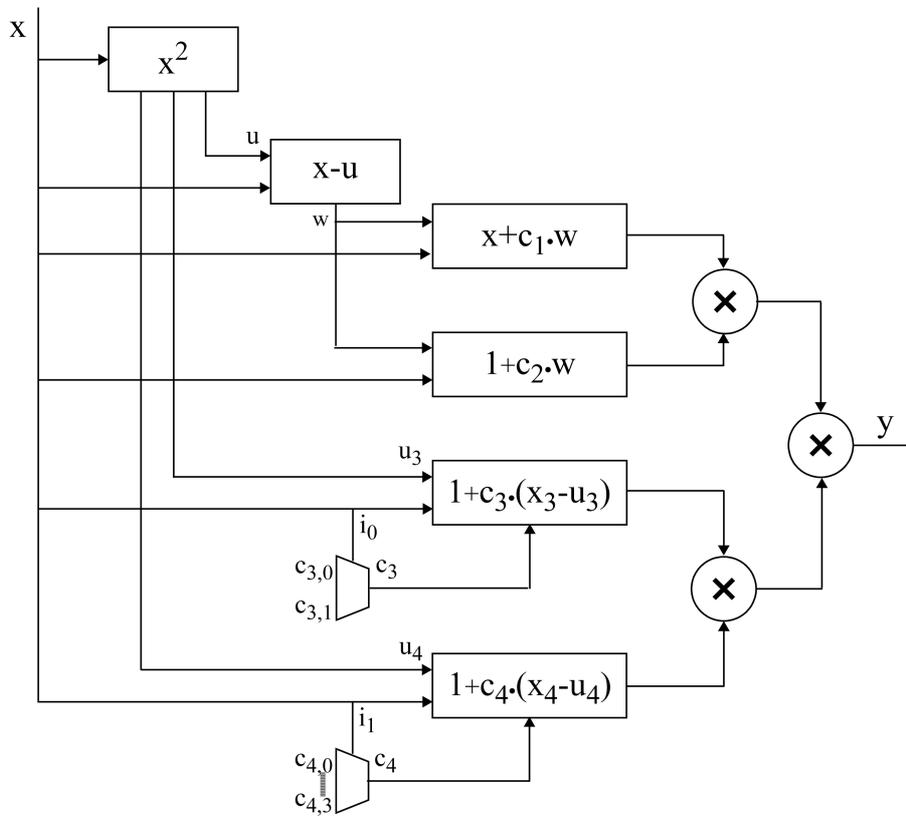


Figure 14. The architecture of the implementation of the sine function.

4.1.5 Optimization of Word Length

As shown in (39) and (41) 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 $(x_n - u_n)$ part, shown in Figure 14, 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 decrease as well, according to the highest value bit in the coefficients and of the $(x_n - u_n)$ 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.

4.1.6 Precision

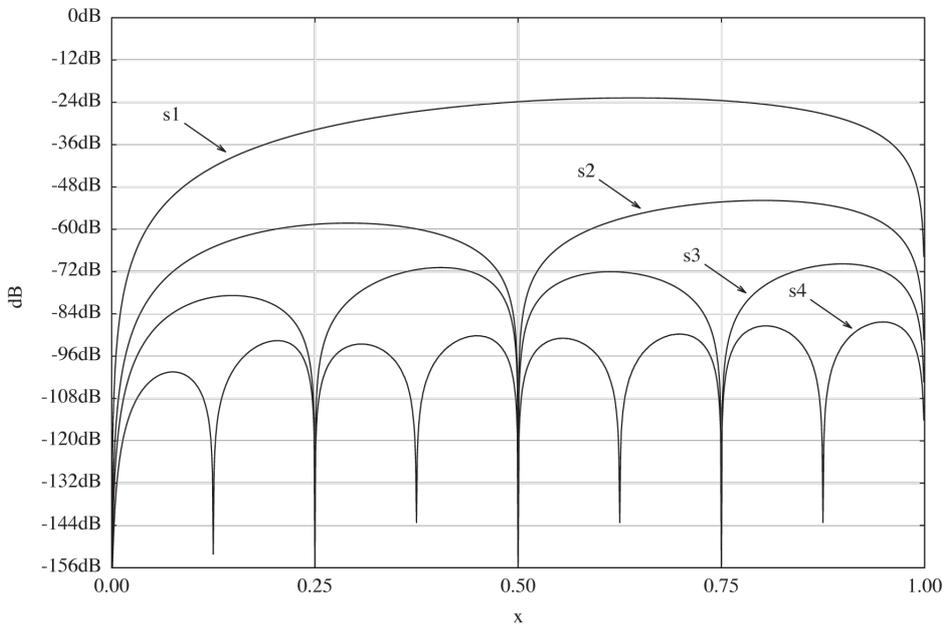


Figure 15. Estimation of the relative error between the original function and different numbers of sub-functions.

In Figure 15 the resulting precision when using one to four sub-functions is shown. A decibel scale is used to visualize the precision since the combination of binary numbers and dB works very well together. In the dB scale, 2 is equal to $20\log_{10}(2) = 20 \cdot (0.3) \approx 6$ dB and since 6 dB corresponds to 1 bit, this will make it simpler to understand the result. As shown in

Figure 15, the relative error decreases with the number of used sub-functions. With 4 sub-functions we can see that accuracy better than 14 bits will correspond to at least 14 adders in the CORDIC algorithm. A critical path of 14 adders is thus unavoidable for the same result. In the general case, one single adder cell delay is added to the critical path for each new adder. However, in the CORDIC case, the sign bit from the previous stage must be ready before next stage can start.

As shown in Figure 15, the relative error decreases with the number of sub-functions used. However, it increases the delay with the number of sub-function as shown in Table II.

TABLE II. NUMBER OF OPERATIONS IN RELATION TO THE NUMBER OF SUB-FUNCTIONS.

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

If we assume that the delay of one multiplier is about two adders, we get a delay of 10 adders compared to 14 in the CORDIC case.

CHAPTER 5

5 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 feasible to be implemented in. 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.

5.1.1 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 III. Since the outcome of the approximation has the desired form no postprocessing is needed.

TABLE III. ALGORITHM FOR THE SINE FUNCTION.

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

5.1.2 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.

TABLE IV. ALGORITHM FOR THE COSINE FUNCTION.

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

5.1.3 The Arcsine Function

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

TABLE V. ALGORITHM FOR THE ARCSINE FUNCTION.

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

5.1.4 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 VI.

TABLE VI. ALGORITHM FOR THE ARCCOSINE FUNCTION.

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

5.1.5 The Tangent Function

When developing the algorithm that performs the approximation of the tangent function the angle range is from 0 to $\pi/4$, 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 VII. Since the outcome of the approximation has the desired form no postprocessing is needed.

TABLE VII. ALGORITHM FOR THE TANGENT FUNCTION.

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

5.1.6 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 VIII.

TABLE VIII. ALGORITHM FOR THE ARCTANGENT FUNCTION.

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

5.1.7 The Logarithmic Function

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

TABLE IX. ALGORITHM FOR THE LOGARITHM FUNCTION.

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

5.1.8 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 X only a one needs to be added in the postprocessing part to get the desired outcome.

TABLE X. ALGORITHM FOR THE EXPONENTIAL FUNCTION.

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

5.1.9 The Division Function

When developing the algorithm that performs the approximation of the division it is limited to the range according to Table XI, 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.

TABLE XI. ALGORITHM FOR THE DIVISION FUNCTION.

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

5.1.10 The Square Root Function

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

TABLE XII. ALGORITHM FOR THE SQUARE ROOT FUNCTION.

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

CHAPTER 6

6 Results

The most common methods used when implementing an approximation of 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. But since the size of the look-up tables grows with the accuracy the execution time will increase with the needed accuracy. Computation by using the CORDIC algorithm 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 a 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.

CHAPTER 7

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 Future Work

The feasibility of the methodology of parabolic synthesis has not fully been investigated why different issues of the methodology remain to be investigated. For the methodology there are computational cases as wordlength optimization of the data flow in the architecture. There are hardware implementation issues regarding that with increasing order of the sub-function the weight of the coefficients will decrease. Both these issues can result in better area efficiency and computation rate for the hardware implementation.

To investigate the feasibility of the methodology it is also important to compare it with other existing methodologies for implementing approximations of unary functions.

When analyzing help functions it is found that these function are piecewise continues functions. Since a help function represent the fault to an approximation and since the fault will decrease with increasing order of sub-function it can be interesting to use the parabolic synthesis methodology as a preprocessing part to an interpolation methodology. It can be expected that this approach can increase both area efficiency and computation rate of the approximation.

References

- [1] E. Meijering (2002), A Chronology of Interpolation From Ancient Astronomy to Modern Signal and Image Processing, *Proceedings of the IEEE*, ISSN: 00189219, Institute of Electrical and Electronics Engineers Inc, vol. 90, no. 3, pp. 319-342, March 2002.
- [2] I. Newton, “Methodus differentialis,” in *The Mathematical Papers of Isaac Newton*, D. T. Whiteside, Ed. Cambridge, U.K.: Cambridge Univ. Press, vol. VIII, ch. 4, pp. 236–257, 1981.
- [3] J. Stirling, “Methodus differentialis Newtoniana illustrata,” *Philos. Trans.*, vol. 30, no. 362, pp. 1050–1070, 1719.
- [4] C. F. Gauss, “Theoria interpolationis methodo nova tractata,” in *Werke. Göttingen, Germany: Königlichen Gesellschaft der Wissenschaften*, vol. III, pp. 265–327, 1866.
- [5] E. Waring, “Problems concerning interpolations,” *Philos. Trans. R. Soc. London*, vol. 69, pp. 59–67, 1779.
- [6] L. Euler, “De eximio usu methodi interpolationum in serierum doctrina,” in *Opuscula Analytica Petropoli*, vol. 1, Academia Imperialis Scientiarum, pp. 157–210, 1783.
- [7] J. L. Lagrange, “Leçons élémentaires sur les mathématiques données a l’école normale,” in *OEuvres de Lagrange*, J.-A. Serret, Ed. Paris, France: Gauthier-Villars, vol. 7, pp. 183–287, 1877.
- [8] F. W. Bessel, “Anleitung und Tafeln die stündliche Bewegung des Mondes zu finden,” *Astronomische Nachrichten*, vol. II, no. 33, pp. 137–141, 1824.
- [9] P. S. de Laplace, “Mémoire sur les suites (1779),” in *OEuvres Complètes de Laplace*. Paris, France: Gauthier-Villars et Fils, vol. 10, pp. 1–89, 1894.
- [10] P. S. de Laplace, *Théorie Analytique des Probabilités*, 3rd ed. Paris, France: Ve. Courcier, vol. 7, 1820.
- [11] J. D. Everett, “On a central-difference interpolation formula,” *Rep. Br. Assoc. Adv. Sci.*, vol. 70, pp. 648–650, 1900.

- [12] J. D. Everett, “On a new interpolation formula,” *J. Inst. Actuar.*, vol. 35, pp. 452–458, 1901.
- [13] K. Weierstrass, “Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen reeller Argumente,” in *Sitzungsberichte der Königlich Preussischen Akademie der Wissenschaften zu Berlin*, pp. 633/789–639/805, July, 1885.
- [14] E. T. Whittaker, “On the functions which are represented by the expansions of interpolation-theory,” in *Proc. R. Soc. Edinburgh*, vol. 35, pp. 181–194, 1915.
- [15] E. Borel, “Mémoire sur les séries divergentes,” *Annales Scientifiques de l’École Normale Supérieure*, ser. 2, vol. 16, pp. 9–131, 1899.
- [16] J. M. Whittaker, “Interpolatory function theory,” in *Cambridge Tracts in Mathematics and Mathematical Physics*. Cambridge, U.K.: Cambridge Univ. Press, 1935.
- [17] J. M. Whittaker, “On the cardinal function of interpolation theory,” in *Proc. Edinburgh Math. Soc.*, vol. 1, pp. 41–46, 1927–1929.
- [18] J. M. Whittaker, “The “Fourier” theory of the cardinal function,” in *Proc. Edinburgh Math. Soc.*, vol. 1, pp. 169–176, 1927–1929.
- [19] V. A. Kotel’nikov, “On the transmission capacity of the “ether” and wire in electrocommunications,” in *Modern Sampling Theory: Mathematics and Applications*, J. J. Benedetto and P. J. S. G. Ferreira, Eds. Boston, MA: Birkhäuser, ch. 2, pp. 27–45, 2000.
- [20] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 1948.
- [21] I. J. Schoenberg, “Contributions to the problem of approximation of equidistant data by analytic functions. Part A—on the problem of smoothing or graduation. A first class of analytic approximation formulae,” *Quart. Appl. Math.*, vol. IV, no. 1, pp. 45–99, 1946.
- [22] I. J. Schoenberg, “Contributions to the problem of approximation of equidistant data by analytic functions. Part B—on the problem of osculatory interpolation. A second class of analytic approximation formulae,” *Quart. Appl. Math.*, vol. IV, no. 2, pp. 112–141, 1946.

- [23] P.T.P. Tang, "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.
- [24] J.-M. Muller, *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, 2006.
- [25] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, 1959.
- [26] R. Andrata, "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, Feb. 1998.
- [27] E. Hertz, P. Nilsson, "A Methodology for Parabolic Synthesis of Unary Functions for Hardware Implementation," *Proc. of the 2nd International Conference on Signals, Circuits and Systems*, ISBN-13: 978-1-4244-2628-7, SCS08_9.pdf, pp 1-6, Hammamet, Tunisia, Nov. 2008.
- [28] B. Parhami, *Computer Arithmetic*, Oxford University Press Inc., ISBN: 0-19-512583-5, 198 Madison Avenue, New York, New York 10016, USA, 2000.
- [29] D. F. Mayer, *An introduction to Numerical Analysis*, Cambridge University Press, ISBN 9780511076534, The Pitt Building, Trumpington Street, Cambridge, United Kingdom, 2003.
- [30] A. Pineiro, M. D. Ercegovic, and J. D. Bruguera, "Algorithm and Architecture for Logarithm, Exponential, and Powering Computation," *IEEE Trans. Computers*, vol. 53, no. 9, pp. 1085-1096, Sept. 2004.
- [31] O. Gustafsson, K. Johansson, "Multiplierless Piecewise Linear Approximation of Elementary Functions," *Proc. of the Fortieth Asilomar Conference on Signals, Systems and Computers (ACSSC'06)*, pp. 1678-1681, ISBN: 1-4244-0784-2, Pacific Grove, Canada, Oct. 29 – Nov. 1 2006.

- [32] K. Johansson, O. Gustafsson, L. Wanhammar, "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 2005.
- [33] T. H. Hu, "CORDIC-Based VLSI Architectures for Digital Signal Processing," in *IEEE Signal Processing Magazine*, pp. 16-35, July1992.
- [34] J.S. Walther, "A Unified Algorithm for Elementary Functions," *Proc. Spring. Joint Computer Conf.*, pp. 379-385, 1971.
- [35] R. Andraka, "A Survey of CORDIC Algorithms for FPGA based Computers," in *Proceedings of FPGA 98. 1998 International Symposium on Field Programmable Gate Arrays*, pp. 191-200, Monterey, CA, USA, February 22-25, 1998.
- [36] P. Nilsson, "Complexity Reductions in Unrolled CORDIC Architectures," in *Proceedings of the IEEE 14th International Conference on Electronics, Circuits and Systems (ICECS 2009)*, pp. 868-871, Hammamet, Tunisia, December 13-16, 2009.
- [37] J. Fourier, *Théorie Analytique de la Chaleur*, Paris, France, 1822.
- [38] P. Ienne, M. A. Viredaz (1994), "Bit-serial multipliers and squarers," *IEEE Transactions on Computer*, ISSN 0018-9340, vol. 43, no12, pp. 1445 -1450, Dec. 1994.
- [39] K. Z. Pekmestzi, P. Kalivas, and N. Moshopoulos (2001), "Long unsigned number systolic serial multipliers and squarers," *IEEE Circuits and Systems II*., ISSN 1057-7130, vol. 48, no 3, pp. 316 -321, March 2001.

List of Figures

Figure 1.	Linear interpolation for computing $f(x)$	19
Figure 2.	Example of a linear spline approximation.	21
Figure 3.	Vector rotation.	23
Figure 4.	Example of normalized function, in this case $\sin\left(\frac{\pi \cdot x}{2}\right)$	28
Figure 5.	Two possible results after dividing an original function with x	30
Figure 6.	Example of the first function $f_1(x)$ compared with sub-function $s_2(x)$	31
Figure 7.	Example of the second function $f_2(x)$	32
Figure 8.	The hardware architecture of the methodology.	34
Figure 9.	The principle of iterative hardware architecture.	35
Figure 10.	The architecture principle for four sub-functions.	36
Figure 11.	Squaring algorithm for the partial product x_n^2	36
Figure 12.	Simplified squaring algorithm for the partial product x_n^2	37
Figure 13.	The function $f(v)$ before normalization and the original function $f_{org}(x)$	41
Figure 14.	The architecture of the implementation of the sine function. ...	45
Figure 15.	Estimation of the relative error between the original function and different numbers of sub-functions.	46

List of Tables

TABLE I.	APPROXIMATION SCHEMES.....	20
TABLE II.	NUMBER OF OPERATIONS IN RELATION TO THE NUMBER OF SUB-FUNCTIONS.	47
TABLE III.	ALGORITHM FOR THE SINE FUNCTION.	49
TABLE IV.	ALGORITHM FOR THE COSINE FUNCTION.	50
TABLE V.	ALGORITHM FOR THE ARCSINE FUNCTION.	50
TABLE VI.	ALGORITHM FOR THE ARCCOSINE FUNCTION.	51
TABLE VII.	ALGORITHM FOR THE TANGENT FUNCTION.	51
TABLE VIII.	ALGORITHM FOR THE ARCTANGENT FUNCTION.....	52
TABLE IX.	ALGORITHM FOR THE LOGARITHM FUNCTION.	52
TABLE X.	ALGORITHM FOR THE EXPONENTIAL FUNCTION.	53
TABLE XI.	ALGORITHM FOR THE DIVISION FUNCTION.....	53
TABLE XII.	ALGORITHM FOR THE SQUARE ROOT FUNCTION.	53

List of Acronyms

ASIC	Application-Specific Integrated Circuit
CORDIC	COordinate Rotation DIgital Computer
DSP	Digital Signal Processor or Digital Signal Processing
MUX	MUltipleXer
ROM	Read-Only Memory
VLSI	Very-Large-Scale Integration