Microprocessors and Microsystems 42 (2016) 142–155

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Combining the parabolic synthesis methodology with second-degree interpolation

Erik Hertz[a,*], Bertil Svensson[a,1], Peter Nilsson[b,2]

[a] Centre for Research on Embedded Systems, Halmstad University, Halmstad, Sweden
[b] Electrical and Information Technology Department, Lund University, Lund, Sweden

ARTICLE INFO

Article history:
Received 26 June 2015
Accepted 26 January 2016
Available online 26 February 2016

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

ABSTRACT

The Parabolic Synthesis methodology is an approximation methodology for implementing unary functions, such as trigonometric functions, logarithms and square root, as well as binary functions, such as division, in hardware. Unary functions are extensively used in baseband for wireless/wireline communication, computer graphics, digital signal processing, robotics, astrophysics, fluid physics, games and many other areas. For high-speed applications, as well as in low-power systems, software solutions are not sufficient and a hardware implementation is therefore needed. The Parabolic Synthesis methodology is a way to implement functions in hardware based on low complexity operations that are simple to implement in hardware. A difference in the Parabolic Synthesis methodology compared to many other approximation methodologies is that it is a multiplicative, in contrast to additive, methodology. To further improve the performance of Parabolic Synthesis based designs, the methodology is combined with Second-Degree Interpolation. The paper shows that the methodology provides a significant reduction in chip area, computation delay and power consumption with preserved characteristics of the error. To evaluate this, the logarithmic function was implemented, as an example, using the Parabolic Synthesis methodology in comparison to the Parabolic Synthesis methodology combined with Second-Degree Interpolation. To further demonstrate the feasibility of both methodologies, they have been compared with the CORDIC methodology. The comparison is made on the implementation of the fractional part of the logarithmic function with a 15-bit resolution. The designs implemented using the Parabolic Synthesis methodology – with and without the Second-Degree Interpolation – perform 4x and 8x better, respectively, than the CORDIC implementation in terms of throughput. In terms of energy consumption, the CORDIC implementation consumes 140% and 800% more energy, respectively. The chip area is also smaller in the case when the Parabolic Synthesis methodology combined with Second-Degree Interpolation is used.

1. Introduction

Computation of elementary functions is legio in a multitude of applications, such as digital signal processing (DSP), control applications, 2D and 3D computer graphics, computer aided design (CAD), virtual reality and physical simulation. The accuracy of the functions is of course important. Software routines can be designed to provide extremely accurate results, but software is often too slow for numerically intensive and real-time applications. There is therefore a significant interest in hardware implementations of function generators.

Hardware computation of elementary functions can be performed by employing many different algorithms [1,2], such as table-based methods, polynomial and rational approximation methods, and functional iteration methods.

Table-based methods remain manageable for low precision computation when the input operand is up to 12–16 bits, corresponding to table sizes of 4–64 K words. The size of the table grows exponentially with the addressing length and becomes unacceptably large when operating with higher precision. An alternative way of making approximations is based on polynomials. Since polynomials involve only additions, subtractions and multiplications, using them is a natural way to approximate elementary functions. A number of schemes are available for polynomial approximations, such as Taylor, Maclaurin, Legendre, Chebyshev, Jacobi and Laguerre [1]. For a given precision, the chosen polynomial scheme affects the number of terms included, and thus the computational complexity. Two development strategies are available in

* Corresponding author. Tel.: +46 702890215; fax: +46 35120348.
E-mail addresses: erik.hertz@hh.se (E. Hertz), bertil.svensson@hh.se (B. Svensson), Peter.Nilsson@eit.lth.se (P. Nilsson).
[1] Tel.: +46 706480843; fax: +46 35120348
[2] Tel.: +46 705770565; fax: +46 46129948

http://dx.doi.org/10.1016/j.micpro.2016.01.015
0141-9331/

developing an approximation, one to minimize the average error, called least squares approximation, and one to minimize the worst case error, called least maximum approximation [1]. An example of when least squares approximation is favorable is when the approximation is used in a series of computations. On the other hand, least maximum approximation is favorable when it is important that the maximum error to the function to be approximated is kept small. An example of when least maximum approximation is favorable is when the error from the approximation has to be within a limit from the true function value. An advantage of polynomials is that they are table-less, but their drawback is that they impose large computational complexities and delays [1]. A reduction in computational complexity can be accomplished by combining table-based methods with polynomial based methods, and the delays can also, to some extent, be decreased [1].

For implementation of elementary functions in hardware, the sum of bit-products methodology [3] can be beneficial, since it can give an area efficient implementation with a high throughput at a reasonable accuracy.

The commonly used COordinate Rotation DIgital Computer (CORDIC) algorithm [4,5] is an iterative algorithm. The benefit of the algorithm is that the hardware for the basic elementary functions requires only a small look-up table, simple shifts and additions. The CORDIC algorithm is used in applications where aspects such as high speed, low power and low area have to be considered. However, since it is an iterative method, it is inherently somewhat slow and therefore often insufficient for very high performance applications. The Parabolic Synthesis methodology [6–9] has a parallel architecture, which significantly reduces the propagation delay and thus provides an advantage over the serial CORDIC algorithm. The reduction in propagation delay also allows a lower clock frequencies and thereby a significant reduction in power consumption.

To further reduce chip area, critical path delay and power consumption, an extension of the Parabolic Synthesis methodology has been developed, which is described in this paper. The extension is achieved by combining Parabolic Synthesis with Second-Degree Interpolation [2,10,11]. In contrast to the Parabolic Synthesis methodology, which is a synthesis of second-order functions and thus provides an accuracy that depends on the number of second-order functions, the accuracy of the combined methodology depends on the number of intervals in the Second-Degree Interpolation part. The architecture of the Parabolic Synthesis methodology is characterized by a high degree of parallelism, which ensures low execution times. Like the Parabolic Synthesis methodology, the Parabolic Synthesis methodology combined with Second-Degree Interpolation utilizes only low complexity operations, again ensuring simple implementation in hardware. The extension admits that the characteristics can to a great extent be tailored. This enables making a rough internal error compensation of the approximation, which improves the distribution of the error.

The feasibility of the Parabolic Synthesis methodology has been verified for implementation on a vast range of unary functions, as shown in [8]. The extended methodology described in this paper has the same broad applicability.

The remaining part of this paper is organized as follows: Section 2 describes the Parabolic Synthesis methodology; Section 3 describes the Parabolic Synthesis Combined with Second-Degree Interpolation; Section 4 describes the general structure of the hardware architecture of both methodologies; Section 5 proposes a general optimization strategy for both methodologies, using the sine function as an illustrative example; Section 6 describes tools for characterization of the error when approximations are made; Section 7 presents the implementation of the logarithm function in both Parabolic Synthesis and Parabolic Synthesis Combined with Second-Degree Interpolation; Section 8 gives a comparison of implementations performed in the different approximation methodologies, CORDIC, Parabolic Synthesis and Parabolic Synthesis Combined with Second-Degree Interpolation, where the comparison is made with respect to chip area, critical path delay and power consumption; and Section 9 closes the paper with conclusions.

2. Parabolic synthesis

The Parabolic Synthesis methodology [6–9] is an approximation methodology for implementing unary functions, such as trigonometric functions, logarithms and the square root, as well as binary functions, such as division, in hardware. The methodology is mainly intended for use in computation intensive applications such as computer graphics, digital signal processing, communication systems, robotics, astrophysics and fluid physics.

The methodology is founded on multiplications of sub-functions, $s_n(x)$, $n = 1,2,\ldots$, each sub-function being a second-order parabolic function. When multiplying these sub-functions, as shown in (1), the original function, $f_{org}(x)$, is obtained. Note that, in the Parabolic Synthesis methodology, the function is based on multiplication of factors, unlike most other methodologies that are based on summation of terms. This is the key to the possibility to parallelize the architecture. Note that $f_{org}(x)$ is almost always a transformation of the function to be approximated, to satisfy the Parabolic Synthesis methodology. The accuracy of an approximation with this methodology depends on the number of sub-functions used.

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

Using infinitely many factors in (1) will recreate $f_{org}(x)$. The procedure for obtaining sub-functions is to develop help functions from which sub-functions are developed. To compute the first help function, $f_1(x)$, the ratio function between the original function, $f_{org}(x)$, and the first sub-function, $s_1(x)$ is computed. This division generates the first help function, $f_1(x)$, as shown in (2).

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} = s_2(x) \cdot s_3(x) \cdot \cdots \cdot s_\infty(x) \tag{2}$$

The following help functions, $f_n(x)$, are generated in the same manner, as shown in (3).

$$f_n(x) = \frac{f_{n-1}(x)}{s_n(x)} = s_{n+1}(x) \cdot s_{n+2}(x) \cdot \cdots \cdot s_\infty(x) \tag{3}$$

2.1. First sub-function

To facilitate the development of the first sub-function, $s_1(x)$, the original function, $f_{org}(x)$, must cut the function x in (0,0) and (1,1) as shown in Fig. 1.

Thus, the function to be approximated has to be normalized and must satisfy the requirement that the values are in the interval $0 \le x < 1$ on the x-axis and $0 \le y < 1$ on the y-axis and have the starting point in (0,0). The normalization of the function to be approximated creates the original function, $f_{org}(x)$.

To satisfy the demands of the methodology, the original function, $f_{org}(x)$, must fulfill three additional criteria.

1. It must be strictly concave or convex through the interval in which it is approximated. The original function, $f_{org}(x)$, has to be strictly concave or convex through the interval since the approximation used is a second-order parabolic function.
2. The original function, $f_{org}(x)$, after it is divided by the first sub-function, $s_1(x)$, must have a limit value when x goes towards 0. If the function has no limit value, it implies that a help function, $f_1(x)$, is not defined when $x = 0$.