



<http://www.diva-portal.org>

Preprint

This is the submitted version of a paper presented at *IEEE International Symposium on Circuits and Systems (ISCAS 2009), Taipei, Taiwan, May 24-27, 2009*.

Citation for the original published paper:

Hertz, E., Nilsson, P. (2009)
Parabolic Synthesis Methodology Implemented on the Sine Function.
In: *IEEE International Symposium on Circuits and Systems, 2009. ISCAS 2009.* (pp. 253-256).
<http://dx.doi.org/10.1109/ISCAS.2009.5117733>

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-22324>

Parabolic Synthesis Methodology Implemented on the Sine Function

Erik Hertz, Member, IEEE

Department of Electrical and Information Technology
Lund University, Lund, Sweden
e-mail: Erik.Hertz@eit.lth.se

Peter Nilsson, Member, IEEE

Department of Electrical and Information Technology
Lund University, Lund, Sweden
e-mail: Peter.Nilsson@eit.lth.se

Abstract—This paper introduces a parabolic synthesis methodology for implementation of approximations of unary functions like trigonometric functions and logarithms, which are specialized for efficient hardware mapped VLSI design. The advantages with the methodology are, short critical path, fast computation and high throughput enabled by a high degree of architectural parallelism. The feasibility of the methodology is shown by developing an approximation of the sine function for implementation in hardware.

I. INTRODUCTION

Unary functions, such as trigonometric functions and logarithms, 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 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 [1]. 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 [1]. The CORDIC (COordinate Rotation DIgital Computer) algorithm [2] [3] 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.

II. METHODOLOGY

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 [1]. The computation is therefore divided into three steps, normalizing, approximation and transforming.

A. 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 convex through the interval. An example of such a function, called an original function $f_{\text{org}}(x)$, is shown in Fig. 1.

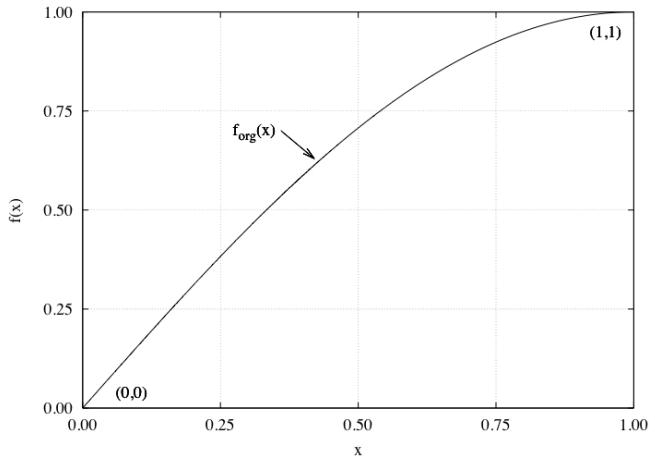


Figure 1. Example of normalized function, in this case $\sin(\pi x/2)$.

B. Developing the Hardware Architecture

When developing 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 [4] 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.

$$f_{org}(x) = s_1(x) \cdot s_2(x) \cdot \dots \cdot s_n(x) \quad (1)$$

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).

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

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).

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

C. 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 according to (4).

$$s_1(x) = x + (c_1 \cdot (x - x^2)) \quad (4)$$

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).

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

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.

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

$$s_2(x) = 1 + (c_2 \cdot (x - x^2)) \quad (6)$$

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).

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

Thereby the second function $f_2(x)$, will get a shape of 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

$0 \leq x < 0.5$ and the second function is thus restricted to the interval $0.5 \leq x < 1.0$. 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 $f_{n,m}(x)$. In equation (8) it is shown how the function $f_n(x)$, is divided into partial functions $f_{n,m}(x)$, when $n > 2$.

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

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 $n > 2$.

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

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 ($\text{frac}()$) of x , as shown in (10).

$$x_n = \text{frac}(2^{n-2} \cdot x) \quad (10)$$

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).

$$s_{n,m}(x_n) = 1 + (e_{n,m} \cdot (x_n - x_n^2)) \quad (11)$$

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

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

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

III. HARDWARE IMPLEMENTATION

For the hardware implementation two's complement representation [5] is used. The implementation is divided into three hardware parts, preprocessing, processing, and postprocessing as introduced by P.T.P. Tang [1].

A. Preprocessing

In this part the incoming operand v is normalized to prepare the input to the processing part, according to section 2A.

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.

B. Processing

In the processing part the approximation of the original function is directly computed in either iterative or parallel 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 price of a larger chip area. 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. 2 the algorithm that performs the squaring and delivers partial product of x_n^2 is described.

	x_3	x_2	x_1	x_0		
\times	x_3	x_2	x_1	x_0		
					x_0x_0	
						p_0
						p
					x_1x_0	
					x_1x_1	
					x_0x_1	
					q_3	q_2
					q_1	p_0
						q
					x_2x_0	
					x_2x_1	
					x_0x_2	
					r_5	r_4
					r_3	r_2
					r_1	r_0
						r
					x_3x_0	
					x_3x_1	
					x_3x_2	
					x_3x_3	x_2x_3
					x_2x_3	x_1x_3
					x_0x_3	
	s_7	s_6	s_5	s_4	s_3	s_2
					s_1	s_0
						s

Figure 2. Squaring algorithm for the partial products x_n^2 .

In Fig. 2 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. 2, it was found that the resemblance to a bit-serial squarer [6] [7] 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 sets of coefficients.

Since the methodology is calculating an approximation of the original function the error between the functions can be both positive and negative. Especially if the value of the approximation is less than the original function some extra bits of word length compared with the desired precision is needed to accomplish the desired precision of the approximation. 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 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.

C. 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.

IV. IMPLEMENTATION OF THE SINE FUNCTION

An implementation of $\sin(v)$, using the proposed methodology is described in this section as an example.

A. Preprocessing

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).

$$v = \frac{\pi}{2} \cdot x \quad (13)$$

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

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

B. 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 are, $c_1 = 0.570796$.

The first function $f_1(x)$, is computed as shown in (15).

$$f_1(x) = \frac{f_{org}(x)}{s_1(x)} \quad (15)$$

When developing the second sub-function $s_2(x)$, the coefficient c_2 is defined according to (7). The determined value of the coefficient are, $c_2 = 0.400857$.

The second function $f_2(x)$, is computed as shown in (16).

$$f_2(x) = \frac{f_1(x)}{s_2(x)} \quad (16)$$

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 $0 \leq x < 0.5$ and $s_{3,1}(x_3)$ is restricted to the interval $0.5 \leq x < 1.0$ 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) so that higher order of sub-functions can be developed. The determined values of the coefficients are, $c_{3,0} = -0.0122449$ and $c_{3,1} = 0.0105948$.

The third function $f_3(x)$, is computed as shown in (17).

$$f_3(x) = \frac{f_2(x)}{s_3(x)} \quad (17)$$

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 $0 \leq x < 0.25$, $s_{4,1}(x_4)$ 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 (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, $c_{4,0} = -0.00223398$, $c_{4,1} = 0.00192499$, $c_{4,2} = -0.00119209$ and $c_{4,3} = 0.00126505$.

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

C. Optimization

Only sub-function $s_{4,3}(x)$ in the interval $0.75 \leq x < 1.0$ has to be improved by optimization. By optimizing the coefficient the necessary word length to achieve the desired precision could be reduced from 17 bits to 16 bits. The optimized value of the coefficient is, $c_{4,3} = 0.0128228$.

V. COMPARISON

The most common methods used when implementing approximation of a unary functions in hardware is 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, but 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.

VI. CONCLUSIONS

A novel methodology for implementing approximations of unary functions such as trigonometric functions, logarithmic functions, etc. in hardware is introduced in this paper. 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.

REFERENCES

- [1] J.-M. Muller, Elementary Functions: Algorithm Implementation, second ed. Birkhauser, 2006.
- [2] J. E. Volder, "The CORDIC Trigonometric Computing Technique," IRE Transactions on Electronic Computers, vol. EC-8, no. 3, 1959, pp. 330–334.
- [3] 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) Feb. 1998, Monterey, CA, pp. 191-200.
- [4] E. M. Stein and R. Shakarchi, Fourier Analysis: An Introduction, Princeton University Press, Princeton, NJ, 2003.
- [5] B. Parhami, "Computer Arithmetic," New York: Oxford University Press, 2000.
- [6] P. Ienneand, M. A. Viredaz, "Bit-serial multipliers and squarers," IEEE Transactions on Computer, v.43, n.12, Dec. 1994, pp. 1445 -1450.
- [7] K.Z.. Pekmestzi, P. Kalivas, and N. Moshopoulos, "Long unsigned number systolic serial multipliers and squarers", IEEE Circuits and Systems II; vol. 48, no 3, Mar. 2001, pp. 316 -321.