Numeric Function Generators Using Decision Diagrams for Discrete Functions

Shinobu Nagayama[1] Tsutomu Sasao[2] Jon T. Butler[3]

[1] Department of Computer and Network Engineering, Hiroshima City University
[2] Department of Computer Science and Electronics, Kyushu Institute of Technology
[3] Department of Electrical and Computer Engineering, Naval Postgraduate School

Abstract

This paper introduces design methods for numeric function generators (NFGs) using decision diagrams. NFGs are hardware accelerators to compute values of numeric functions such as trigonometric, logarithmic, square root, and reciprocal functions. Most existing design methods for NFGs are intended only for a specific class of numeric functions. However, by using decision diagrams for discrete functions (i.e., word-level decision diagrams), we can systematically design fast and compact NFGs for a larger class of functions. This paper shows three design methods for NFGs using 1) multi-terminal binary decision diagrams (MTBDDs), 2) binary moment diagrams (BMDs), and 3) edge-valued binary decision diagrams (EVBDDs).

Keywords: Numeric function generators (NFGs), numeric functions, decision diagrams for discrete functions (word-level decision diagrams)

1. Introduction

Numeric functions such as trigonometric, logarithmic, square root, reciprocal, and combinations of these functions are extensively used in computer graphics, digital signal processing, communication systems, robotics, astrophysics, fluid physics, and so forth [19]. In these applications, as well as addition and multiplication, numeric functions are usually used as a basic operation.

The computation of numeric functions has been studied for more than 150 years [35], and various algorithms, such as COordinate Rotation DIgital Computer (CORDIC) [1, 33] have been proposed. Most programming languages have a library containing numeric functions (e.g., `math.h` in C) and users are often unaware of the methods for computing those functions. With the increasing use of digital systems in commercial products, there has been an increase in the need for systematic design methods for numeric function generators (NFGs) that can realize a large set of numeric functions.

Although NFGs based on iterative algorithms such as CORDIC are compact and achieve high-precision, they can be slow because their computation time is proportional to the precision (that is, the number of bits) [32]. For a function that is composed of other functions, such as the Box-Muller functions, $f_1 = \sqrt{-2\ln(g_1)}\cos(2\pi g_2)$ and $f_2 = \sqrt{-2\ln(g_1)}\sin(2\pi g_2)$, an iterative computation of each basic function, $s = \ln(g_1)$ followed by $\sqrt{-2s}$, can yield a large computation time. This is due, in part, to the time required to do an iterative computation and, in part, to the fact that one computation ($s = \ln(g_1)$) must be computed before the next ($\sqrt{-2s}$) is started. Another disadvantage of iterative algorithms is that they are applicable only to specific functions. We seek a design method that applies to a general class of functions and yields a small delay time, even if the realized function is composite.

A straightforward design method for an arbitrary function $f(x)$ is to use a single lookup table (LUT) in which the address is the binary representation of the value of x and the content of that address is the corresponding value of $f(x)$. This design method produces a very fast NFG because the value of function is obtained by only one table lookup. Thus, for low-precision computations of $f(x)$ (for example, x and $f(x)$ have 8 bits), this design method is efficient, since the size of the LUT is small. For high-precision computations, however, it is impractical due to the huge table size.

To reduce the table size, various design methods based on polynomial approximation have been proposed [3, 8, 11, 12, 16, 27–29, 31, 36]. These methods approximate the given numeric functions by piecewise polynomials and realize the polynomials with hardware. Linear or quadratic approximations offer fast and relatively high-precision computation of numeric functions. However, most existing design methods based on polynomial approximation are intended only for a specific class of functions, and thus, applying them to other functions is not always efficient. Indeed, we showed that the existing design methods are inefficient for certain numeric functions [26]. Thus, systematic design methods intended for a larger class of functions are required.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAY 2009		

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Numeric Function Generators Using Decision Diagrams for Discrete Functions	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Department of Electrical and Computer Engineering,Monterey,CA,93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

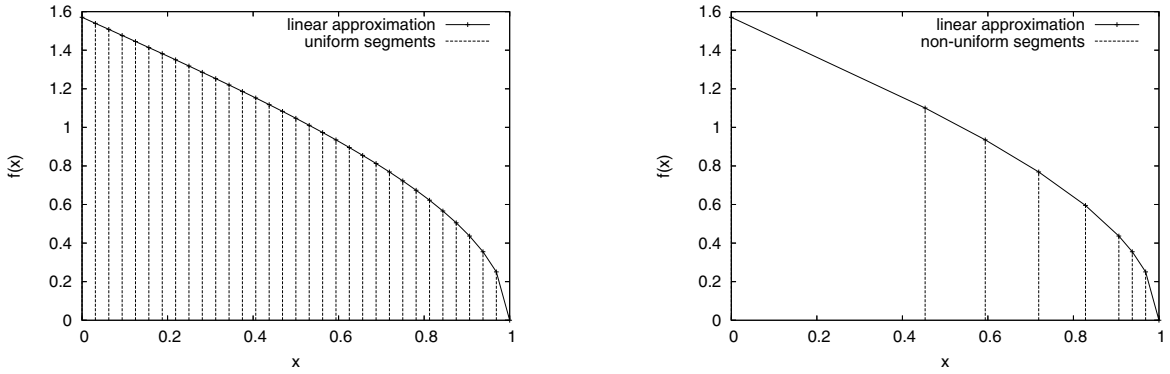
13. SUPPLEMENTARY NOTES

14. ABSTRACT

This paper introduces design methods for numeric function generators (NFGs) using decision diagrams. NFGs are hardware accelerators to compute values of numeric functions such as trigonometric, logarithmic, square root and reciprocal functions. Most existing design methods for NFGs are intended only for a speci c class of numeric functions. However, by using decision diagrams for discrete functions (i.e., word-level decision diagrams), we can systematically design fast and compact NFGs for a larger class of functions. This paper shows three design methods for NFGs using 1) multi-terminal binary decision diagrams (MTBDDs), 2) binary moment diagrams (BMDs), and 3) edge-valued binary decision diagrams (EVBDDs).

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**6**	



(a) Uniform segmentation. (b) Non-uniform segmentation.

Figure 1. Piecewise polynomial approximation of $\arccos(x)$**.**

For design of typical digital circuits, systematic methods using various decision diagrams such as binary decision diagrams (BDDs) [9,17] have been established [18,24,34,37]. Thus, we consider design methods for NFGs using decision diagrams. By using decision diagrams for discrete functions, called word-level decision diagrams, we can systematically design fast and compact NFGs for a large class of functions. This paper introduces systematic design methods for NFGs using three word-level decision diagrams: multi-terminal BDDs (MTBDDs) [4], binary moment diagrams (BMDs) [2], and edge-valued binary decision diagrams (EVBDDs) [14].

This paper is organized as follows: Section 2 introduces a design method using MTBDDs. By using an MTBDD to represent a component of an NFG, we can design a wide range of functions efficiently. Although this design method is based on a piecewise polynomial approximation, it is different from existing methods. This section also shows differences between our method and existing methods. Section 3 introduces a design method using BMDs. By considering fixed-point numeric functions as discrete (integer) functions, we can apply the arithmetic transform to them. Thereby, we can directly represent numeric functions using BMDs, and produce NFGs from the BMDs. Section 4 introduces a design method using EVBDDs. Many common fixed-point numeric functions can be converted into monotone discrete functions, and we can represent them compactly using EVBDDs. This section shows that EVBDDs are useful in producing fast and compact NFGs.

2. Design Method Using MTBDDs

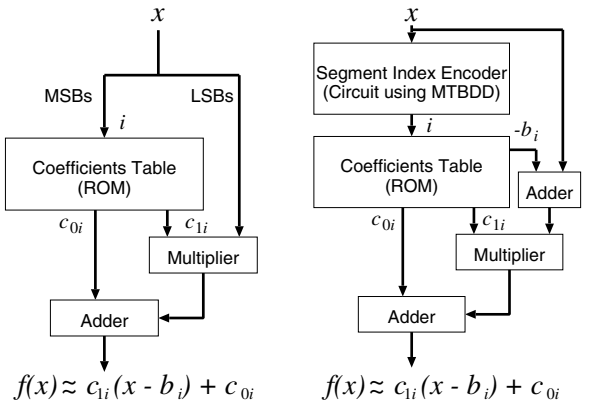
This section introduces a design method that uses an MTBDD to represent a component of an NFG. Since the design method is based on a piecewise polynomial approximation, this section first introduces the piecewise polynomial approximation.

2.1. Piecewise Polynomial Approximation

Numeric functions can be approximated by polynomial functions such as Taylor series and Chebyshev series. Since polynomial functions can be realized with multipliers and adders, any numeric functions can be realized in hardware by using polynomial approximation. However, if a single polynomial is used, a high order is usually required to achieve the desired approximation error. Therefore, many multipliers and adders are needed, and this results in a slow NFG. Instead, we use more than one polynomial, dividing the domain into (often many) segments where a lower order polynomial can achieve the given approximation error. This is achieved by *piecewise polynomial approximation*.

Piecewise polynomial approximation of a function $f(x)$ divides a domain x of the function $f(x)$ into *segments*, and approximates the function by a polynomial in each segment. Because reducing the segment size can reduce the approximation error even if polynomial order is low, this approximation method is suitable for NFG design.

Many existing methods [3,8,11,12,16,27–29,31,36] use piecewise polynomial approximation based on *uniform segmentation*, which divides a domain x into segments with the same size as the smallest segment size needed to achieve the desired accuracy as shown in Fig. 1(a). A piecewise linear polynomial approximation based on uniform segmentation can be realized with the architecture shown in Fig. 2(a). In this architecture, the most significant bits of x are used to specify a segment i where $b_i \le x < e_i$, and the least significant bits determine a point within that segment $(x-b_i)$. The coefficients table stores polynomial coefficients c_{1i} and c_{0i} for each segment, which are used to compute a linear polynomial $c_{1i}(x-b_i)+c_{0i}$. Uniform segmentation can yield too



(a) NFG based on uniform segmentation.

(b) NFG based on non-uniform segmentation.

Figure 2. Architectures for NFGs.

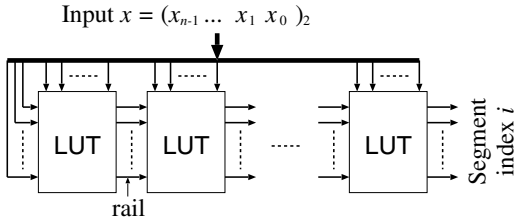


Figure 3. Architecture for segment index encoder.

many segments, depending on the functions realized [26]. Since the size of the coefficients table depends on the number of segments, for such functions, the NFGs require excessive memory size. Thus, an approximation with fewer segments is desired to produce compact NFGs for various functions.

To reduce the number of segments, we use another approximation method based on *non-uniform segmentation*. In this method, segments are chosen to be as wide as possible while still achieving the desired accuracy as shown in Fig. 1(b). Thus, the segments are likely to have different widths. In this way, non-uniform segmentation yields fewer segments than uniform segmentation, and results in a small coefficients table. However, non-uniform segmentation requires an additional circuit, called *segment index encoder*, that maps values of x into a segment i, as shown in Fig. 2(b). This is because a domain x is divided at arbitrary positions, unlike uniform segmentation. Potentially, this is a complex circuit that could cancel out the size reduction in the coefficients table. Therefore, simplifying this circuit is important.

We use MTBDDs to design the segment index encoders.

Table 1. Memory sizes (bits) needed for 24-bit precision NFGs.

Function $f(x)$	Domain	No. of segments		Memory size		Ratio
		Uniform	Non	Uniform	Non	[%]
$1/x$	[1/32,1]	507,905	28,010	25,165,824	10,911,744	43
$\sqrt{x}$	[0,1]	8,388,609	3,941	402,653,184	1,417,216	0.4
$\ln(x)$	[1/256,1]	522,241	11,761	22,544,384	5,586,944	25
$x\ln(x)$	(0,1)	2,097,152	4,535	69,206,016	2,588,672	4
$\sqrt{-\ln(x)}$	(0,1]	8,388,608	12,089	209,715,200	5,668,864	3
$\arcsin(x)$	[0,1]	8,388,609	4,415	402,653,184	2,818,048	0.7
$\tan(\pi x)$	[0,31/64]	507,905	20,770	23,592,960	8,847,360	38

Uniform: Uniform segmentation. Non: Non-uniform segmentation.

By using MTBDDs, we can compactly realize *any non-uniform segmentation* in which the memory size depends on the number of segments [26]. As a result, we can systematically design fast and compact NFGs for a wide range of numeric functions.

2.2. Design of the Segment Index Encoder

Fig. 3 shows the architecture of our segment index encoder. It consists of LUTs only. We design the segment index encoder by specifying the contents of each LUT. The segment index encoder realizes a mapping from a fixed-point representation of x into a segment index, as shown in Fig. 4(a). We consider this mapping as a discrete function shown in Fig. 4(b), and represent it using an MTBDD. By decomposing the MTBDD, as shown in Fig. 4(c), we design the segment index encoder. In this figure, the column labeled as 'r_i' in the table of each LUT denotes the rails that represent sub-functions in the MTBDD. In the MTBDD, numbers assigned to edges that cut across the horizontal lines represent the sub-functions. Since, in the MTBDD, a value of the discrete function is obtained by traversing the diagram from the root node to a terminal node, the segment index encoder obtains a segment index by accessing each LUT in a sequence.

2.3. Memory Size Needed for an NFG

Table 1 shows the number of segments for uniform and non-uniform segmentations, and memory sizes needed for the two NFGs shown in Fig. 2. Note that the memory size of NFGs based on non-uniform segmentation is the sum of memory sizes needed for the segment index encoder and the coefficients table.

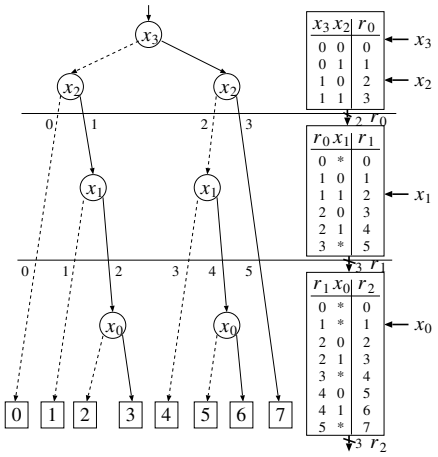
This table shows that, although NFGs based on non-uniform segmentation require a segment index encoder, they need less memory because the segment index encoder is compact. In this way, by using MTBDDs, we can compactly realize any non-uniform segmentation. As a result, we can systematically design fast and compact NFGs for a wide range of numeric functions.

Segments	Index
$0.0000 \le x < 0.2500$	0
$0.2500 \le x < 0.3750$	1
$0.3750 \le x < 0.4375$	2
$0.4375 \le x < 0.5000$	3
$0.5000 \le x < 0.6250$	4
$0.6250 \le x < 0.6875$	5
$0.6875 \le x < 0.7500$	6
$0.7500 \le x < 1.0000$	7

(a) Segments and their index.

x_3	x_2	x_1	x_0	i	x_3	x_2	x_1	x_0	i
0	0	0	0	0	1	0	0	0	4
0	0	0	1	0	1	0	0	1	4
0	0	1	0	0	1	0	1	0	5
0	0	1	1	0	1	0	1	1	6
0	1	0	0	1	1	1	0	0	7
0	1	0	1	1	1	1	0	1	7
0	1	1	0	2	1	1	1	0	7
0	1	1	1	3	1	1	1	1	7

(b) Discrete function.



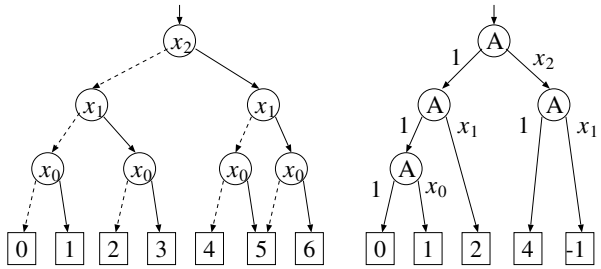
(c) Decomposition of MTBDD.

Figure 4. Design of segment index encoder.

Table 2. Number of distinct values in function vector and arithmetic spectrum.

Numeric functions	No. of distinct values		Ratio
	Function	Spectrum	[%]
$2^x - 1$	59,895	148	0.25
$\frac{1}{\sqrt{x+1}} - \frac{1}{\sqrt{2}}$	19,196	174	0.90
$\ln(x+1)$	45,427	165	0.36
$\log_2(x+1)$	59,895	160	0.27
$\sqrt{x+1} - 1$	27,147	138	0.50
$\frac{2}{x+1} - 1$	54,292	180	0.33
$\sin(x)$	55,147	141	0.25

Number of bits for function values is 16.
Domain of the functions is $0 \le x < 1$.
Ratio = Spectrum / Function $\times$ 100.



(a) MTBDD (b) BMD

Figure 5. MTBDD and BMD for 3-bit fixed-point $\sin(x)$.

3. Design Method Using BMDs

In the previous section, by *considering the segment index encoder as a discrete function*, we can use an MTBDD, resulting in a compact design. However, not only the segment index encoder but also *a numeric function itself can be considered as a discrete function*. Thus, we directly represent numeric functions using word-level decision diagrams, and produce their NFGs from the decision diagrams. Since numeric functions can be expanded into polynomial functions, such as a Taylor series, in this section, we use BMDs that can represent polynomial functions compactly [22].

While terminal nodes in an MTBDD directly represent values of a discrete function, terminal nodes in a BMD represent the arithmetic spectrum obtained by the arithmetic transform of the discrete function. And, non-terminal nodes represent the inverse arithmetic transform. That is, the BMD represents the discrete function by the arithmetic spectrum and the inverse arithmetic transform. As shown in Table 2, for many numeric functions, the number of distinct values in the arithmetic spectrum is much smaller than the number of distinct function values. Thus, BMDs represent numeric functions more compactly than MTBDDs, as the example in Fig. 5 shows. In Fig. 5(b), each node labeled by 'A' represents the arithmetic transform expansion: $f = f_0 + x_i(f_1 - f_0)$.

Each non-terminal node of a BMD is realized with an adder and an AND gate, as shown in Fig. 6. By computing the inverse arithmetic transform from the pre-computed arithmetic spectrum using the circuit in Fig. 6, we obtain values of the original discrete function. That is, by representing numeric functions using BMDs, we can design their NFGs by a systematic use of the circuit in Fig. 6. Since the

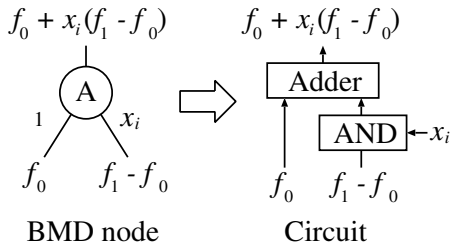


Figure 6. Realization of non-terminal node of BMD.

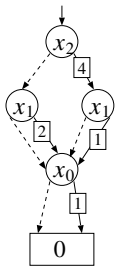


Figure 7. EVBDD for 3-bit fixed-point $\sin(x)$.

size of the designed NFGs is proportional to the number of BMD nodes, analysis of the complexity (size) of BMDs is important. It has been presented in [22, 30][1].

4. Design Method Using EVBDDs

In the previous section, we used BMDs to represent numeric functions in a design that was based on the viewpoint that *numeric functions can be expanded into polynomial functions*. In this section, we base the design on the viewpoint that *many common numeric functions can be converted into monotone functions*. In this case, we use EVBDDs [22].

Since many common numeric functions to be designed in hardware are monotone increasing or decreasing, discrete functions converted from their fixed-point representation are also monotone increasing or decreasing. Periodic functions such as $\sin(x)$ are also monotone in a domain to be designed in hardware (for example, $0 \leq x \leq \pi/2$). While an MTBDD represents function values at terminal nodes, an EVBDD represents function values by the sum of edge weights. As a result, EVBDDs represent numeric functions that are converted into monotone discrete functions more compactly than MTBDDs. Fig. 7 shows an example of an EVBDD representation of the $\sin(x)$ function.

[1] Although Stankovic and Astola [30] use arithmetic transform decision diagrams (ACDDs), their approach to the design of NFGs is similar to ours. Thus, their analysis is useful for the design of NFGs.

Table 3. Numbers of nodes in three decision diagrams for numeric functions.

Numeric functions	Number of nodes			R_{BM}	R_{EM}	R_{EB}
	MTBDD	BMD	EVBDD	[%]	[%]	[%]
$2^x - 1$	122,659	29,634	3,469	24	2.8	12
$\frac{1}{\sqrt{x+1}} - \frac{1}{\sqrt{2}}$	58,412	28,446	2,857	49	4.9	10
$\ln(x+1)$	100,880	28,442	3,187	28	3.2	11
$\log_2(x+1)$	122,542	29,553	3,465	24	2.8	12
$\sqrt{x+1} - 1$	73,406	26,149	2,383	36	3.2	9
$\frac{2}{x+1} - 1$	114,093	28,348	4,079	25	3.6	14
$\sin(x)$	115,450	22,638	2,853	20	2.5	13
Average	101,063	27,601	3,185	29	3.3	12

Number of fractional bits for function values is 16.
Domain of the functions is $0 \leq x < 1$.
R_{BM} = BMD / MTBDD $\times$ 100.
R_{EM} = EVBDD / MTBDD $\times$ 100.
R_{EB} = EVBDD / BMD $\times$ 100.

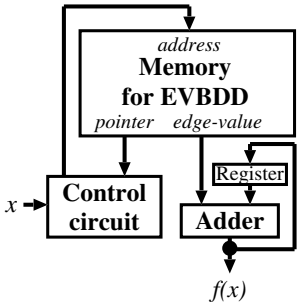


Figure 8. Architecture for NFG based on EVBDD.

Table 3 compares the numbers of nodes in MTBDDs, BMDs, and EVBDDs for seven numeric functions. BMDs represent numeric functions more compactly than MTBDDs due to the arithmetic transform. However, EVBDDs are smaller than the BMDs.

In an EVBDD, we can evaluate a discrete function by traversing the diagram and accumulating the values of traversed edges. Thus, we can systematically design NFGs for many fixed-point numeric functions using the architecture shown in Fig. 8. Since this architecture consists only of a memory to store an EVBDD, an accumulator for the edge values, and a control circuit to traverse the EVBDD, it can realize various numeric functions easier and faster than CORDIC.

5. Conclusion and Comments

This paper has introduced three design methods for numeric function generators (NFGs) using word-level decision diagrams. It showed that by using decision diagrams for discrete functions, we can systematically design fast and compact NFGs for a larger class of numeric functions.

References

[1] R. Andraka, "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, Monterey, CA, Feb. 1998.

[2] R. E. Bryant and Y-A. Chen, "Verification of arithmetic circuits with binary moment diagrams," *Design Automation Conference*, pp. 535–541, 1995.

[3] J. Cao, B. W. Y. Wei, and J. Cheng, "High-performance architectures for elementary function generation," *Proc. of the 15th IEEE Symp. on Computer Arithmetic (ARITH'01)*, Vail, Co, pp. 136–144, June 2001.

[4] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," *Proc. of 30th ACM/IEEE Design Automation Conference*, pp. 54–60, June 1993.

[5] J. Detrey and F. de Dinechin, "Table-based polynomials for fast hardware function evaluation," *16th IEEE Inter. Conf. on Application-Specific Systems, Architectures, and Processors (ASAP'05)*, pp. 328–333, 2005.

[6] J. Detrey and F. de Dinechin, "A parameterizable floating-point logarithm operator for FPGAs," *39th Asilomar Conf. on Signals, Systems and Computers,* pp. 1186–1190, 2005.

[7] J. Detrey and F. de Dinechin, "A parameterized floating-point exponential function for FPGAs," *IEEE Inter. Conf. on Field-Programmable Technology (ICFPT'05),* pp. 27–34, 2005.

[8] F. de Dinechin and A. Tisserand, "Multipartite table methods," *IEEE Trans. on Comp.*, Vol. 54, No. 3, pp. 319–330, Mar. 2005.

[9] R. Drechsler and B. Becker, *Binary Decision Diagrams: Theory and Implementation*, Kluwer Academic Publishers, 1998.

[10] ANSI/IEEE Standard 754-2008, *IEEE Standard for Floating-Point Arithmetic*, 2008.

[11] V. K. Jain, S. A. Wadekar, and L. Lin, "A universal nonlinear component and its application to WSI," *IEEE Trans. on Components, Hybrids, and Manufacturing Technology*, Vol. 16, No. 7, pp. 656–664, Nov. 1993.

[12] V. K. Jain and L. Lin, "High-speed double precision computation of nonlinear functions," *Proc. of the 12th IEEE Symp. on Computer Arithmetic (ARITH'95)*, Bath, England, pp. 107–114, July 1995.

[13] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and applications," *Multiple-Valued Logic: An International Journal*, Vol. 4, No. 1-2, pp. 9–62, 1998.

[14] Y-T. Lai and S. Sastry, "Edge-valued binary decision diagrams for multi-level hierarchical verification," *Proc. of 29th ACM/IEEE Design Automation Conference*, pp. 608–613, 1992.

[15] D.-U. Lee, W. Luk, J. Villasenor, and P. Y. K. Cheung, "Hierarchical segmentation schemes for function evaluation," *Proc. of the IEEE Conf. on Field-Programmable Technology*, Tokyo, Japan, pp. 92–99, Dec. 2003.

[16] D. M. Lewis, "Interleaved memory function interpolators with application to an accurate LNS arithmetic unit," *IEEE Trans. on Comp.*, Vol. 43, No. 8, pp. 974–982, Aug. 1994.

[17] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design: OBDD – Foundations and Applications*, Springer, 1998.

[18] D. M. Miller and M. A. Thornton, "QMDD: A decision diagram structure for reversible and quantum circuits," *36th International Symposium on Multiple-Valued Logic*, Singapore, May 17-20, 2006.

[19] J.-M. Muller, *Elementary Function: Algorithms and Implementation*, Birkhauser Boston, Inc., Secaucus, NJ, 1997.

[20] S. Nagayama and T. Sasao, "On the optimization of heterogeneous MDDs," *IEEE Trans. on CAD*, Vol. 24, No. 11, pp. 1645–1659, Nov. 2005.

[21] S. Nagayama and T. Sasao, "Representations of elementary functions using edge-valued MDDs," *37th International Symposium on Multiple-Valued Logic*, Oslo, Norway, May 13-16, 2007.

[22] S. Nagayama and T. Sasao, "Complexities of graph-based representations for elementary functions," *IEEE Trans. on Computers*, Vol. 58, No. 1, pp. 106–119, Jan. 2009.

[23] J.-A. Piñeiro, S. F. Oberman, J.-M. Muller, and J. D. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Trans. on Comp.*, Vol. 54, No. 3, pp. 304–318, Mar. 2005.

[24] T. Sasao and M. Fujita (eds.), *Representations of Discrete Functions*, Kluwer Academic Publishers 1996.

[25] T. Sasao and S. Nagayama "Representations of elementary functions using binary moment diagrams," *36th International Symposium on Multiple-Valued Logic*, Singapore, May 17-20, 2006.

[26] T. Sasao, S. Nagayama, and J. T. Butler, "Numerical function generators using LUT cascades," *IEEE Transactions on Computers*, Vol. 56, No. 6, pp. 826–838, Jun. 2007.

[27] M. J. Schulte and E. E. Swartzlander, Jr., "Hardware designs for exactly rounded elementary functions," *IEEE Trans. on Comp.*, Vol. 43, No. 8, pp. 964–973, Aug. 1994.

[28] M. J. Schulte and J. E. Stine, "Symmetric bipartite tables for accurate function approximation," *13th IEEE Symp. on Comput. Arith.*, Asilomar, CA, Vol. 48, No. 9, pp. 175–183, 1997.

[29] M. J. Schulte and J. E. Stine, "Approximating elementary functions with symmetric bipartite tables," *IEEE Trans. on Comp.*, Vol. 48, No. 8, pp. 842–847, Aug. 1999.

[30] R. Stankovic and J. Astola, "Remarks on the complexity of arithmetic representations of elementary functions for circuit design," *Workshop on Applications of the Reed-Muller Expansion in Circuit Design and Representations and Methodology of Future Computing Technology*, pp. 5–11, May 2007.

[31] J. E. Stine and M. J. Schulte, "The symmetric table addition method for accurate function approximation," *Jour. of VLSI Signal Processing*, Vol. 21, No. 2, pp. 167–177, June 1999.

[32] N. Takagi, T. Asada, and S. Yajima, "Redundant CORDIC methods with a constant scale factor for sine and cosine computation," *IEEE Trans. on Comp.*, Vol. 40, No. 9, pp. 989–995, Sept. 1991.

[33] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electronic Comput.*, Vol. EC-820, No. 3, pp. 330–334, Sept. 1959.

[34] I. Wegener, *Branching Programs and Binary Decision Diagrams: Theory and Applications*, SIAM, 2000.

[35] M. R. Williams, *History of Computing Technology*, IEEE Computer Society Press, Los Alamitos, CA, 1997.

[36] W. Wong and E. Goto, "Fast evaluation of the elementary functions in single precision," *IEEE Trans. on Comp.*, Vol. 44, No. 3, pp. 453–457, Mar. 1995.

[37] S. N. Yanushkevich, D. M. Miller, V. P. Shmerko, and R. Stankovic, *Decision Diagram Techniques for Micro- and Nanoelectronic Design*, CRC Press, Taylor & Francis Group, 2006.