2011 41st IEEE International Symposium on Multiple-Valued Logic

Numeric Function Generators Using Piecewise Arithmetic Expressions

Shinobu Nagayama* Tsutomu Sasao† Jon T. Butler‡

*Dept. of Computer and Network Eng., Hiroshima City University, Hiroshima, JAPAN
†Dept. of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka, JAPAN
‡Dept. of Electr. and Comp. Eng., Naval Postgraduate School, Monterey, CA USA

***Abstract*—This paper proposes new architectures for numeric function generators (NFGs) using piecewise arithmetic expressions. The proposed architectures are programmable, and they realize a wide range of numeric functions. To design an NFG for a given function, we partition the domain of the function into uniform segments, and transform a subfunction in each segment into an arithmetic spectrum. From this arithmetic spectrum, we derive an arithmetic expression, and realize the arithmetic expression with hardware. Since the arithmetic spectrum has many zero coefficients and repeated coefficients, by storing only distinct nonzero coefficients in a table, we can significantly reduce the table size needed to store arithmetic coefficients. Experimental results show that the table size can be reduced to only a small percent of the table size needed to store all the arithmetic coefficients. We also propose techniques to reduce table size further and to improve performance.**

***Keywords*-numeric function generators; piecewise arithmetic expressions; nonzero arithmetic coefficients; programmable architectures.**

I. Introduction

Numeric functions, such as trigonometric, logarithmic, square root, and combinations of these functions, are widely used in computer graphics, digital signal processing, communication systems, robotics, etc. [5]. In these applications, as well as addition and multiplication, numeric functions are usually used as a basic operation. Particularly, in graphics applications, about half of the total processing time is used to compute numeric functions [12]. Thus, for numerically intensive or real-time applications, hardware accelerators, called numeric function generators (NFGs), are often required. The computation of numeric functions has been studied for more than 150 years [21], and various NFGs have been proposed [2], [4], [13], [16], [17]. Many existing NFGs are based on polynomial approximations.

For design and verification of arithmetic circuits such as adders and multipliers, the arithmetic transform is often used due to its compactness [1], [3], [14], [20], [22]. However, for the design of NFGs, it is rarely used. Only a few studies on NFGs using the arithmetic transform have been reported [15], [19]. However, in both papers, different architectures are required for different numeric functions.

Although a dedicated NFG for a specific numeric function is fast, many NFGs have to be designed for a wide range of numeric functions. Since this consumes chip area and accounts for much of the design and production costs, a programmable NFG, which can compute various numeric functions at high-speed with a single architecture, is required, along with a systematic design method. To satisfy this requirement, this paper proposes new architectures and a design method for programmable NFGs using the arithmetic transform. In [15], [19], the arithmetic transform is applied to the whole of a numeric function. However, this is unsuitable for design of programmable NFGs because they require too many additions. To design an efficient NFG, we uniformly partition the domain of a given numeric function into segments, and apply the arithmetic transform to a subfunction for each segment. From the arithmetic spectrum obtained by the transform, we derive an arithmetic expression, and realize the arithmetic expression with memories and an accumulator. By changing the memory data, we can realize a wide range of numeric functions with a single architecture.

This paper is organized as follows: Section II introduces a numeric representation of a real numeric function, and the arithmetic transform. Section III presents piecewise arithmetic expressions, and architectures for NFGs based on them. Experimental results are shown in Section IV. And, Section V presents techniques to reduce memory size and to improve the performance of NFGs.

II. Preliminaries

A. Number Representation

This subsection defines a number representation and describes how to convert real functions into integer functions.

Definition 1: Let $B = \{0,1\}$, $\mathbb{Z}$ be the set of the integers, and $\mathbb{R}$ be the set of the real numbers. An n-input m-output **logic function** is a mapping: $B^n \rightarrow B^m$, a (binary-input) **integer function** is a mapping: $B^n \rightarrow \mathbb{Z}$, and a **real function** is a mapping: $\mathbb{R} \rightarrow \mathbb{R}$.

Definition 2: A value X represented by the **binary fixed-point representation** is denoted by

$$X = (x_{n\_int-1}\ x_{n\_int-2}\ \ldots\ x_1\ x_0.\ x_{-1}\ x_{-2}\ \ldots\ x_{-n\_frac})_2,$$

0195-623X/11 $26.00 © 2011 IEEE
DOI 10.1109/ISMVL.2011.32

Table I
FUNCTION TABLE FOR 3-BIT $\sin(X)$.

(a) Table for $\sin(X)$.

X	$\sin(X)$
0.000	0.000
0.125	0.125
0.250	0.247
0.375	0.366
0.500	0.479
0.625	0.585
0.750	0.682
0.875	0.768

(b) Truth table for $f_b(X)$.

X	$f_b(X)$
0.000	0.000
0.001	0.001
0.010	0.010
0.011	0.011
0.100	0.100
0.101	0.101
0.110	0.101
0.111	0.110

(c) Table for $f(X)$.

X	$f(X)$
000	0
001	1
010	2
011	3
100	4
101	5
110	5
111	6

where $x_i \in \{0,1\}$ for $-n\_frac \le i \le n\_int-1$, n_int is the number of bits for the integer part, and n_frac is the number of bits for the fractional part of X. We call

$$X = \sum_{i=-n\_frac}^{n\_int-1} 2^i x_i$$

an **n-bit fixed-point representation** in which n bits are used to represent the value, where $n = n\_int + n\_frac$. In this paper, an **n-bit function** $f(X)$ means that the input variable X has n bits.

We can convert a real function in fixed-point representation to an n-input m-output logic function. The logic function, in turn, can be converted into an integer function by considering binary vectors as integers. That is, we can convert a real function into an integer function: $B^n \rightarrow P_m$, where $P_m = \{0,1,\ldots,2^m-1\}$. In this paper, numeric functions are converted into integer functions by using a fixed-point representation, unless stated otherwise. And, for simplicity, each bit in the fixed-point representation of X is denoted by x_i; x_0 is the least significant bit.

Example 1: Table I (a) shows values of $\sin(X)$ for eight values of X. Using a 3-bit fixed-point representation, this function is converted into the logic function $f_b(X)$ in Table I (b). By representing the output vectors as integers, we have the integer function $f(X)$ in Table I (c). In this paper, the 3-bit $\sin(X)$ denotes the integer function $f(X)$ in Table I (c). (End of Example)

B. Arithmetic Transform

This subsection introduces the arithmetic transform, the arithmetic spectrum, and the arithmetic expression [18].

First, define a matrix operation and some notation.

Definition 3: Let A be an $(n \times n)$ square matrix, where

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix}.$$

Let B be an $(n \times n)$ square matrix. Then, the **Kronecker product** of A and B is the $(n^2 \times n^2)$ matrix:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \ldots & a_{1n}B \\ a_{21}B & a_{22}B & \ldots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}B & a_{n2}B & \ldots & a_{nn}B \end{bmatrix}.$$

Definition 4: Given a matrix M, the **transposed matrix** M^t is obtained by interchanging rows and columns of M. For an n-bit integer function $f(X)$, the **function-vector** F is the column vector of the function values $F = [f(00\ldots0), f(00\ldots01), \ldots, f(11\ldots1)]^t$.

We define the arithmetic transform and the arithmetic spectrum as follows:

Definition 5: The **arithmetic transform matrix** is

$$\mathcal{A}(n) = \bigotimes_{i=1}^{n} \mathcal{A}(1), \quad where \quad \mathcal{A}(1) = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix},$$

such that addition and multiplication are done in integer arithmetic. For an integer function f given by the function-vector F, the **arithmetic spectrum** $\mathcal{A}_f = [a_0, a_1, \ldots, a_{2^n-1}]^t$ is

$$\mathcal{A}_f = \mathcal{A}(n)F.$$

Each a_i in the spectrum is called an **arithmetic coefficient**.

Example 2: Consider the 1-bit adder function $f(x_1,x_2) = x_1 + x_2$. The function-vector is $F = [0,1,1,2]^t$. The arithmetic spectrum is

$$\mathcal{A}_f = \mathcal{A}(2)F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

(End of Example)

Similarly, we define the inverse arithmetic transform as follows:

Definition 6: Let $\mathcal{A}^{-1}(n)$ be the **inverse arithmetic transform matrix** defined by

$$\mathcal{A}^{-1}(n) = \bigotimes_{i=1}^{n} \mathcal{A}^{-1}(1), \qquad \mathcal{A}^{-1}(1) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

Definition 7: In a symbolic representation,

$$\mathcal{A}^{-1}(1) = \begin{bmatrix} 1 & x_i \end{bmatrix}.$$

Therefore, the **inverse arithmetic transform** is defined as

$$f = X_a \mathcal{A}_f, \qquad X_a = \bigotimes_{i=1}^{n} \begin{bmatrix} 1 & x_i \end{bmatrix}.$$

Example 3: By the inverse arithmetic transform from the arithmetic spectrum obtained in Example 2, the integer

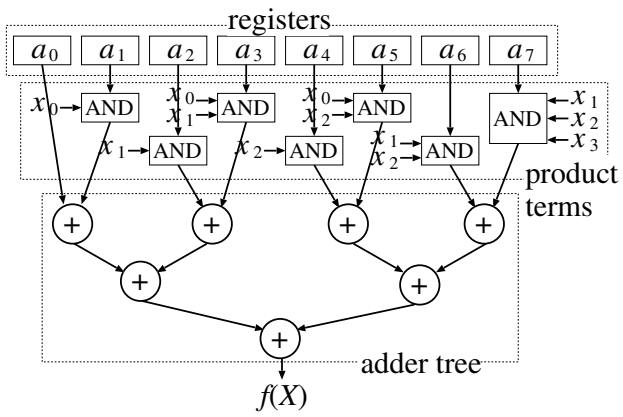


Figure 1. 3-bit programmable NFG based on the arithmetic expression.

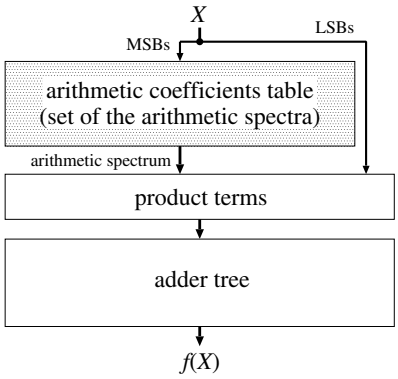


Figure 2. Programmable NFG based on the piecewise arithmetic expression.

function f is represented as follows:

$$\begin{aligned} f &= X_a \mathcal{A}_f = \begin{bmatrix} 1 & x_2 & x_1 & x_1x_2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \\ &= x_1 + x_2. \end{aligned}$$

(End of Example)

From Definitions 6 and 7, we can see that an integer function $f(X)$ can be represented by the arithmetic spectrum and the inverse arithmetic transform. That is,

Lemma 1: Using $\mathcal{A}^{-1}(1)$ and $\mathcal{A}(1)$, an integer function f is represented as follows:

$$\begin{aligned} f &= \mathcal{A}^{-1}(1)\mathcal{A}(1)F = \begin{bmatrix} 1 & x_i \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & x_i \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 - f_0 \end{bmatrix} \\ &= f_0 + x_i(f_1 - f_0), \end{aligned} \tag{1}$$

where $f_0 = f(x_i = 0)$, $f_1 = f(x_i = 1)$. (1) is the **arithmetic transform expansion** (also called A-expansion or moment decomposition [1]). The **arithmetic expression** for f is obtained by the arithmetic transform expansion. The arithmetic coefficients correspond to coefficients of the arithmetic expression for f.

III. NFGs Based on Piecewise Arithmetic Expressions

This section introduces piecewise arithmetic expressions, and presents programmable architectures for NFGs based on them.

A. Piecewise Arithmetic Expressions

Since a numeric function can be converted into an integer function using the fixed-point representation, it can be represented by the arithmetic expression:

$$a_0 + a_1x_0 + a_2x_1 + a_3x_1x_0 + \ldots + a_{2^n-1}x_{n-1}x_{n-2}\ldots x_0.$$

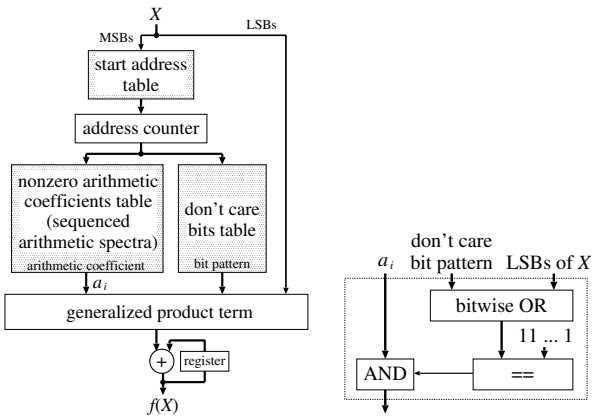
The arithmetic expression can be realized with only AND gates and adders, and thus, it is realized with a compact circuit when many arithmetic coefficients a_i are zero. Since many elementary functions, such as $\sin(x)$ and $\log(x)$, have many zero arithmetic coefficients, we can design compact NFGs for them [15], [19]. However, fixed-point representations with many bits necessarily produce the arithmetic expressions with too many product terms resulting in large and slow NFGs. In addition, a straightforward programmable implementation of the NFGs proposed in [15], [19], as shown in Fig. 1, needs too many adders ($2^n - 1$ adders).

To reduce the number of product terms (adders), we transform sub-functions into a *set of the arithmetic spectra*, instead of transforming the whole domain of a function into *the single arithmetic spectrum*, and represent the function using a *set of the arithmetic expressions*. Then, we design a programmable NFG using the set of the arithmetic expressions.

To produce a set of the arithmetic expressions, we partition the domain of a given numeric function into uniform segments, and apply the arithmetic transform to a sub-function for each segment. Hence, we call the set of arithmetic expressions a **piecewise arithmetic expression**. Note that, in the piecewise arithmetic expression, we partition the domain into segments using the most significant bits (MSBs) of X.

B. Architectures for Programmable NFGs

By realizing a set of the arithmetic spectra for the piecewise arithmetic expression with a memory (called arithmetic coefficients table) we obtain the NFG in Fig. 2. The MSBs of X select a segment (an arithmetic spectrum), and then an



(a) Overall architecture (b) Generalized product term

Figure 3. Programmable NFG based on a sequential computation.

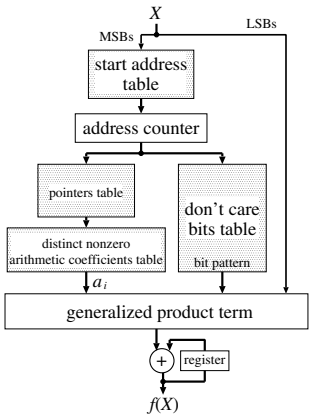


Figure 4. Improved architecture for programmable NFG.

arithmetic expression is computed using the least significant bits (LSBs) of X. This NFG requires $2^k - 1$ adders, where k is the number of the LSBs. Thus, it is more compact and faster than the NFG based on the single arithmetic expression in Fig. 1, in which $2^n - 1$ adders are required.

Unfortunately, the number of adders is still large, and this design is inefficient. Since the arithmetic spectra usually have many zero coefficients, the arithmetic coefficients table is sparse, and many unnecessary additions are performed. To perform only necessary additions, and to reduce the number of adders, we propose the architecture shown in Fig. 3. Note that for readability of the figures, the enable signal (an external input) to start the computation (reset registers), the done signal (an external output) to denote finish of the computation, and multiplexers are omitted from Fig. 3.

In this architecture, only the *nonzero* arithmetic coefficients are stored in a table. By reading out each coefficient sequentially, it computes the arithmetic expression using an accumulator. Each product term is computed with the circuit in Fig. 3(b) using a don't care bit pattern. In the don't care bit pattern, bits corresponding to input variables that do not appear in a product term are set to 1. For example, for a 5-bit function, the bit pattern for the product term x_3x_0 is 10110. The start address table stores a start address of the nonzero arithmetic coefficients table and the don't care bits table for each segment. In this architecture, the evaluation time of an arithmetic expression is proportional to the number of nonzero arithmetic coefficients. Thus, a numeric function that has many zero arithmetic coefficients can be computed at high speed.

To further reduce the number of arithmetic coefficients to be stored in a table, we omit repeated coefficients in a table. Fig. 4 shows the improved architecture. By using pointers to the distinct arithmetic coefficients instead of directly storing the coefficients, we can significantly reduce the bit width of the table if the number of distinct coefficients is small.

IV. Experimental Results

To show the efficiency of the proposed NFGs, we compare the table size and the number of additions for the three proposed NFGs. Table II shows the experimental results. In this table, the column "No. of additions" denotes the number of additions needed to compute an arithmetic expression for each segment. Since, in the NFGs in Figs. 3 and 4, the number of product terms for different arithmetic expressions are different, the average number of additions for each expression is shown.

Since the programmable NFG based on the single arithmetic expression in Fig. 1 requires $2^{16} = 65,536$ registers and $2^{16} - 1 = 65,535$ adders, the proposed NFGs based on the piecewise arithmetic expression require several orders of magnitude fewer adders, and much less storage size. As shown in Table II, for many numeric functions, the number of nonzero arithmetic coefficients and the number of distinct arithmetic coefficients are small. Thus, by storing only these, we significantly reduce size of the arithmetic coefficients table, resulting in a reduction of total size of tables.

V. Improvement Techniques for NFGs

A. Piecewise Polynomial Approximation

By using a polynomial approximation, we can reduce the number of nonzero arithmetic coefficients, and thus, the table size and the number of additions can be further reduced. This is based on the following lemma:

Lemma 2: [8] For an n-bit kth-degree polynomial function $f(X) = c_kX^k + c_{k-1}X^{k-1} + \ldots + c_0$, the number of nonzero arithmetic coefficients is at most

$$\sum_{i=0}^{k} \binom{n}{i}.$$

Table II
TABLE SIZE AND THE NUMBER OF ADDITIONS FOR 16-BIT NFGS.

Functions $f(X)$	No. of stored coefficients			Size of coefficients table			Total size of tables			No. of additions	
	NFG1 (all)	NFG2 (nonzero)	NFG3 (distinct)	NFG1 (bits)	NFG2 (bits)	NFG3 (bits)	NFG1 (bits)	NFG2 (bits)	NFG3 (bits)	NFG1	NFG2,3 (average)
2^X	65,536	35,524	445	1,048,576	568,384	7,120	1,048,576	892,196	651,093	255	138.8
e^X	65,536	36,590	587	1,048,576	585,440	9,392	1,048,576	918,846	709,872	255	142.9
$\ln(X+1)$	65,536	36,914	402	1,048,576	590,624	6,432	1,048,576	926,946	674,980	255	144.2
$\log_2(X+1)$	65,536	35,069	451	1,048,576	561,104	7,216	1,048,576	880,821	642,554	255	137.0
$1/(X+1)$	65,536	37,702	401	1,048,576	603,232	6,416	1,048,576	946,646	689,549	255	147.3
$\sqrt{X+1}$	65,536	37,316	323	1,048,576	597,056	5,168	1,048,576	936,996	681,275	255	145.8
$\sin(X)$	65,536	31,327	391	1,048,576	501,232	6,256	1,048,576	787,015	573,982	255	122.4
$\tan(X)$	65,536	33,397	548	1,048,576	534,352	8,768	1,048,576	839,021	647,955	255	130.5
$\sin^{-1}(X)$	65,536	32,059	532	1,048,576	512,944	8,512	1,048,576	805,315	622,005	255	125.2
$\tan^{-1}(X)$	65,536	32,463	401	1,048,576	519,408	6,416	1,048,576	815,415	594,590	255	126.8

NFG1: the NFG shown in Fig. 2. NFG2: the NFG shown in Fig. 3. NFG3: the NFG shown in Fig. 4.
No. of additions: the number of additions needed to compute each arithmetic expression.
The number of MSBs for uniform segmentation is 8. The domain of all functions is $0 \leq X \leq 1$.

We approximate a given numeric function using a piecewise polynomial within a desired error, and then transform the polynomial into an arithmetic expression in each segment. The piecewise arithmetic expression obtained in this way is realized with a compact NFGs.

Example 4: Consider a piecewise quadratic polynomial approximation of a 16-bit numeric function using 256 uniform segments. Then, a polynomial in each segment has 8 bits. Thus, the total number of nonzero arithmetic coefficients is at most

$$256 \times \sum_{i=0}^{2} \binom{8}{i} = 256 \times 37 = 9,472,$$

and the number of additions is only 36. (End of Example)

In this way, by using piecewise polynomial approximation, more compact and faster programmable NFGs based on the piecewise arithmetic expression can be produced.

B. Parallel Computation

The proposed NFGs based on a sequential computation in Figs. 3 and 4 produce an arithmetic coefficient one by one, and compute each product term of an arithmetic expression sequentially. Thus, they require $O(N)$ computation time, and are obviously slower than the NFG in Fig. 2 which requires $O(\log N)$ computation time, where N is the number of product terms.

On the other hand, the NFG in Fig. 2 produces all arithmetic coefficients simultaneously, and adds all terms of an arithmetic expression at once. Thus, it is faster but requires many more adders. Note that the adders have different sizes. However, in an FPGA or ASIC, this is not a problem because different size adders can be easily accommodated.

These two designs are extreme cases. By changing the number of terms to be computed in parallel, we can explore the design space taking into account a tradeoff between the number of adders and the computation time, and can produce an optimum NFG depending on applications.

VI. CONCLUSION AND COMMENTS

This paper proposes new architectures for programmable NFGs using piecewise arithmetic expressions, and design methods for them. We also propose techniques to reduce table size and to improve performance. Experimental results show that the size of the arithmetic coefficients table can be reduced to only a few percent of the table size needed to store all the arithmetic coefficients. By using the proposed NFGs, we can realize a wide range of numeric functions with a single architecture, and we can switch the functions by only changing the contents of tables.

In this paper, we used about one-half of input bits to partition the domain of a function into uniform segments. However, there could be an optimum number of bits for each function. Thus, we will study optimum segmentations as future work. We will also analyze the relation between the number of nonzero arithmetic coefficients and the characteristic of functions.

ACKNOWLEDGMENTS

This research is partly supported by the Grant in Aid for Scientific Research of the Japan Society for the Promotion of Science (JSPS), funds from Ministry of Education, Culture, Sports, Science, and Technology (MEXT) via Knowledge Cluster Project, the MEXT Grant-in-Aid for Scientific Research (C), (No. 22500050), 2010, and Hiroshima City University Grant for Special Academic Research (General Studies), (No. 0206), 2010. We would like to thank Prof. R. Stankovic for discussions that motivated this work.

REFERENCES

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

[2] 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.

[3] R. Drechsler, B. Becker, and S. Ruppertz, "K*BMDs: A new data structure for verification," *European Design & Test Conf.*, pp. 2–8, 1996.

[4] 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.

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

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

[7] 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.

[8] 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.

[9] S. Nagayama, T. Sasao, and J. T. Butler, "Floating-point numerical function generators using EVMDDs for monotone elementary functions," *39th International Symposium on Multiple-Valued Logic*, Okinawa, Japan, May, 2009.

[10] S. Nagayama, T. Sasao, and J. T. Butler, "Floating-point numerical function generators based on piecewise-split EVMDDs," *40th International Symposium on Multiple-Valued Logic*, pp.223-228, May, 2010.

[11] S. Nagayama, T. Sasao, and J. T. Butler, "A systematic design method for two-variable numeric function generators using multiple-valued decision diagrams," *IEICE Trans. on Information and Systems*, Vol. E93-D, No. 8, pp. 2059–2067, Aug. 2010.

[12] B.-G. Nam, H. Kim, and H.-J. Yoo, "Power and area-efficient unified computation of vector and elementary functions for handheld 3D graphics systems," *IEEE Transactions on Computers*, Vol. 57, No. 4, pp. 490–503, Apr. 2008.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] R. Stankovic, T. Sasao, and C. Moraga, "Spectral transform decision diagrams," Chapter 3 in [14].

[19] 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.

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

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

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