Floating-Point Numeric Function Generators Based on Piecewise-Split EVMDDs

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 proposes a new architecture for memory-based floating-point numeric function generators (NFGs). The design method uses piecewise-split edge-valued multi-valued decision diagrams (EVMDDs). To design NFGs with less memory size, we partition the domain of the floating-point function into segments, and represent the function using an EVMDD for each segment. By realizing each EVMDD with hardware, we obtain the floating-point NFG. This paper also presents an algorithm that partitions the domain by decomposing the edge-valued binary decision diagram (EVBDD) representing the whole floating-point function. Experimental results show that, for a single-precision floating-point function, our new NFG requires 40% *to* 65% *less memory than any previous one for generic function.*

1. 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. [11]. In these applications, numeric functions are usually used as a basic operation, along with addition and multiplication. For fast computation of numeric functions, various hardware circuits, called numeric function generators (NFGs), have been proposed. Fixed-point representation of numeric functions is often adopted [1, 9, 14, 16, 18, 19, 21, 22]. However, for numeric functions with a wide domain and range, a fixed-point representation requires a large number of bits. This produces large NFGs. To represent a large real value with fewer bits, floating-point representation is often used. An IEEE standard for real values exists [5]. However, floating-point representation tends to produce complex and slow NFGs. Thus, the design of floating-point NFGs is especially hard, and only design methods for some numeric functions are known [2, 3, 6, 20, 24]. Since these design methods are intended only for specific functions, different functions need different design methods and architectures. Thus, an architecture for floating-point NFGs that can realize a large set of numeric functions is required, along with a systematic design method.

Large-capacity memories are now available. Like SRAM-based FPGAs, implementation of logic circuits using memory has become practical. Thus, we focus on memory-based floating-point NFGs that can realize various numeric functions by changing data in memory. A straightforward memory-based NFG for an arbitrary floating-point function $f(x)$ is 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 NFG is very fast because the value of the function is obtained by only one table lookup. For high-precision computations, however, it requires too much memory. To design monotone floating-point numeric functions with less memory, we have proposed a memory-based NFG using an edge-valued multi-valued decision diagram (EVMDD) [15]. Unfortunately, it still requires a large memory for high-precision computations.

Therefore, this paper proposes a new architecture for memory-based NFGs that requires less memory. To design an NFG with less memory, we partition the domain of a floating-point function into segments, and represent the function using an EVMDD for each segment. By realizing each EVMDD with memory, we obtain a memory-based floating-point NFG. This paper also presents an algorithm that partitions the domain by decomposing the edge-valued binary decision diagram (EVBDD) representing the whole floating-point function.

This paper is organized as follows: Section 2 introduces a floating-point representation of a real-valued numeric function, and decision diagrams used in this paper. Section 3 presents piecewise-split EVMDDs, and an algorithm that partitions the domain of a floating-point function. Section 4 presents a new architecture for memory-based floating-point NFGs. Experimental results are shown in Section 5.

Table 1. Floating-point representation of X with a-bit exponent and b-bit significand.

Type	Exponent E	Significand D	Value of X
Zero	$(0,0,\ldots,0)_2$	$(0,0,\ldots,0)_2$	0
Subnormal no.	$(0,0,\ldots,0)_2$	$\neq(0,0,\ldots,0)_2$	$(-1)^s\times 2^{-E_s}\times 0.D$
Infinity	$(1,1,\ldots,1)_2$	$(0,0,\ldots,0)_2$	$(-1)^s\times\infty$
Not a no. (NaN)	$(1,1,\ldots,1)_2$	$\neq(0,0,\ldots,0)_2$	NaN
Normal no.	Others		$(-1)^s\times 2^{E-E_n}\times 1.D$

Bias value for subnormal numbers: $E_s = 2^{a-1}-2$

Bias value for normal numbers: $E_n = 2^{a-1}-1$

2. Preliminaries

2.1. Number Representation and Precision

This subsection defines a number representation and a precision 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: $B^n\rightarrow B^m$, an **integer function** is a mapping: $\mathbb{Z}\rightarrow\mathbb{Z}$, and a **real function** is a mapping: $\mathbb{R}\rightarrow\mathbb{R}$.*

Definition 2 *The **n-bit precision binary floating-point representation** of a number X is a binary n-tuple*

$$X=(s,\quad e_{a-1},e_{a-2},\ldots,e_0,\quad d_{b-1},d_{b-2},\ldots,d_0)_2,$$

*where $s\in B$ is the **sign bit**, $E=(e_{a-1},e_{a-2},\ldots,e_0)_2$ is the **exponent**, and $D=(d_{b-1},d_{b-2},\ldots,d_0)_2$ is the **significand**. a and b are the numbers of bits for the exponent and the significand respectively, and $n=a+b+1$. The value of X is shown in Table 1. When $|X|<2^{2-2^{a-1}}$, X is a **subnormal number**, in which the exponent E is biased by $E_s=2^{a-1}-2$, and the significand D represents only fractional bits of a fixed-point value smaller than 1. When $2^a\leq|X|$, X is **infinity**. When X cannot be defined as a number, X is represented as **not a number (NaN)**. In other cases, X is a **normal number**, in which the exponent E is biased by $E_n=2^{a-1}-1$, and the significand D represents only fractional bits of a fixed-point value that is larger than or equal to 1 and smaller than 2.*

*According to IEEE Standard 754-2008 [5], **half (16-bit) precision** has $a=5$ and $b=10$, **single (32-bit) precision** has $a=8$ and $b=23$, **double (64-bit) precision** has $a=11$ and $b=52$, and **quad (128-bit) precision** has $a=15$ and $b=112$.*

Note that by using an n-bit precision floating-point representation, we can convert a real function into an n-input n-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: $P_n\rightarrow P_n$, where $P_n=\{0,1,\ldots,2^n-1\}$. In this paper, numeric functions are converted into integer functions by using a floating-point representation, unless stated otherwise. And, for simplicity, each bit in the floating-point representation of X is denoted by x_i, where x_0 is the least significant bit.

Table 2. Tables for the 8-bit precision (3-bit exponent, 4-bit significand) floating-point $\sqrt{X}$.

(a) Table for $\sqrt{X}$.

X	$\sqrt{X}$
0.000000	0.000000
0.015625	0.125000
0.031250	0.171875
0.046875	0.218750
0.062500	0.250000
0.078125	0.281250
0.093750	0.312500
0.109375	0.328125
$\vdots$	$\vdots$

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

X	$f_b(X)$
0 000 0000	0 000 0000
0 000 0001	0 000 1000
0 000 0010	0 000 1011
0 000 0011	0 000 1110
0 000 0100	0 001 0000
0 000 0101	0 001 0010
0 000 0110	0 001 0100
0 000 0111	0 001 0101
$\vdots$	$\vdots$

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

X	$f(X)$
0	0
1	8
2	11
3	14
4	16
5	18
6	20
7	21
$\vdots$	$\vdots$

Example 1 *Table 2 (a) is the function table for $\sqrt{X}$. The 8-bit precision (3-bit exponent and 4-bit significand) floating-point representation of this function is the logic function $f_b(X)$ in Table 2 (b). By converting binary vectors into integers, we have the integer function $f(X)$ of $f_b(X)$ in Table 2 (c). That is, our 8-bit precision floating-point representation of $f(X)=\sqrt{X}$ corresponds to the integer function of Table 2 (c). (End of Example)*

2.2. Edge-Valued MDDs

This subsection defines the decision diagrams used in this paper.

Definition 3 *An **edge-valued binary decision diagram (EVBDD)** [8, 17, 23] is a variant of the BDD [4, 10] that represents an integer function. The EVBDD is obtained by repeatedly applying the expansion $f=\overline{x_i}f_0+x_i(f_1'+\alpha)$ to the integer function, where $f_1=f_1'+\alpha$, and α is the constant term of f_1. The EVBDD consists of only one terminal node representing 0 and non-terminal nodes with 1-edges having integer weights α. In an EVBDD, 0-edges always have zero weights. The incoming edge into the root node can have a non-zero weight. The output (integer) value of an EVBDD is the sum of the weights associated with the path taken from the root node to the terminal node.*

Definition 4 *For a set of n binary variables $\{X\}$, if $\{X\}=\{X_u\}\cup\{X_{u-1}\}\cup\ldots\cup\{X_1\}$, $\{X_i\}\neq\emptyset$, and $\{X_i\}\cap\{X_j\}=\emptyset$ $(i\neq j)$, then $(X_u,X_{u-1},\ldots,X_1)$ is a **partition of** X. Each*

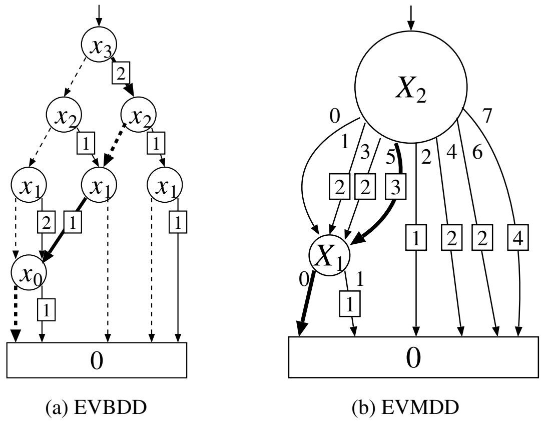


Figure 1. EVBDD and EVMDD for an integer function.

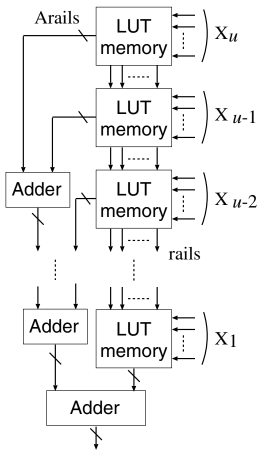


Figure 2. Architecture for NFG based on a monolithic EVMDD.

*X_i forms a **super variable**. Let $|X_i| = k_i$ and $k_u + k_{u-1} + \ldots + k_1 = n$. Then, by considering each super variable as a multi-valued variable, an integer function $f(X) : \mathbb{Z} \rightarrow \mathbb{Z}$ can be converted into a **multi-valued input integer function** $f(X_u, X_{u-1}, \ldots, X_1) : P_u \times P_{u-1} \times \ldots \times P_1 \rightarrow \mathbb{Z}$, where $P_i = \{0, 1, 2, \ldots, 2^{k_i} - 1\}$.*

Definition 5 *An **edge-valued multi-valued decision diagram (EVMDD)** [13] is an extension of the MDD [7, 12, 25], and represents a multi-valued input integer function. It consists of one terminal node representing 0 and non-terminal nodes. Edges have integer weights. Edges labeled by a logic 0 have integer 0 weight.*

Example 2 *Fig. 1 (a) and (b) show the EVBDD and the EVMDD, respectively, for the same integer function. In Fig. 1 (a), dashed lines and solid lines denote 0-edges and weighted 1-edges, respectively. In the EVMDD, the set of binary variables $\{X\}$ is partitioned into $\{X_2\} = \{x_3, x_2, x_1\}$ and $\{X_1\} = \{x_0\}$. To obtain the function value 3 for $X = (1, 0, 1, 0)_2$, we traverse the EVBDD or the EVMDD from the root node to the terminal node according to the input values, and obtain the function value as the sum of the weights for the traversed edges. Note that we traverse the EVMDD using $X_2 = 5$ and $X_1 = 0$. (End of Example)*

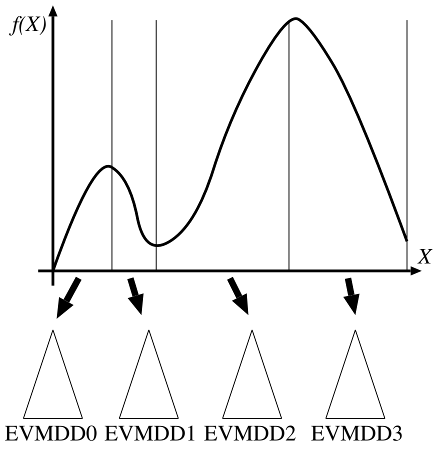


Figure 3. Example of a piecewise-split EVMDD.

3. Representation Using Piecewise-Split EVMDDs

This section introduces a piecewise-split EVMDD, and presents an algorithm that partitions the domain of a floating-point function.

3.1. Piecewise-Split EVMDDs

Since floating-point numeric functions can be converted into integer functions, they can be compactly represented using EVMDDs. Fig. 2 shows their architecture, which is the result of a decomposition of an EVMDD [15]. However, high-precision (a large number of bits) floating-point functions necessarily require large EVMDDs resulting in NFGs with large memory size. Also, they are slow. To reduce memory size, we represent a floating-point numeric function using a *set of smaller EVMDDs*, instead of using a *monolithic EVMDD*. Then, we design a floating-point NFG using the smaller EVMDDs.

In this paper, we produce a set of EVMDDs by partitioning the domain of a floating-point function into segments, and representing the function using an EVMDD for

Input:	EVBDD for floating-point numeric function $f(X)$ and threshold value t for the number of nodes.
Output:	Piecewise-split EVMDD (set of EVMDDs).

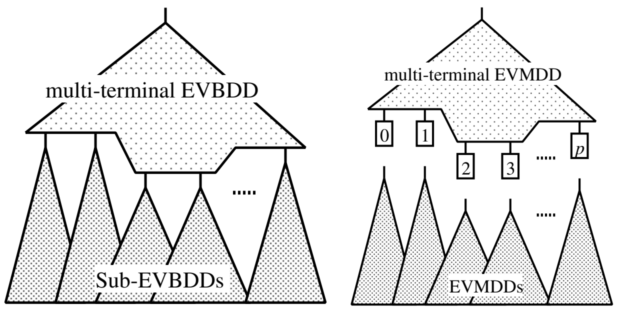
```
Step:
  1.    Compute the numbers of nodes in all the sub-EVBDDs.
  2.    Traverse the EVBDD recursively from the root node.
   2.1. Decompose the EVBDD when the number of nodes in a sub-EVBDD is smaller than or equal to $t$.
   2.2. Assign a segment number to the sub-EVBDD, which represents the numeric function in the segment.
  3.    Convert the produced sub-EVBDDs into EVMDDs.
  4.    Construct a multi-terminal EVBDD by considering each segment number as a terminal node, where the root node of the multi-terminal EVBDD is the same as the original one.
  5.    Convert the multi-terminal EVBDD into a multi-terminal EVMDD.
```

Figure 4. Segmentation algorithm using EVBDD.

each segment, as shown in Fig. 3. Hence, we call the set of EVMDDs a **piecewise-split EVMDD**. Note that, in a piecewise-split EVMDD, we need an EVMDD which selects a segment from input X, in addition to the EVMDD for each segment.

3.2. Segmentation Algorithm Using EVBDD

In a piecewise-split EVMDD, the number of EVMDDs and size of each EVMDD depend on how the domain of the function is segmented. Thus, an effective segmentation algorithm which makes the sizes of all the EVMDDs the same is desired to reduce memory size of NFG. In this paper, we propose a segmentation algorithm that partitions the domain into segments by decomposing the *EVBDD* representing the whole floating-point function. By using the EVBDD, we can simultaneously produce a segmentation of the domain and the piecewise-split EVMDD.



(a) Decomposed EVBDD (b) Piecewise-split EVMDD

Figure 5. Decomposition of EVBDD and piecewise-split EVMDD.

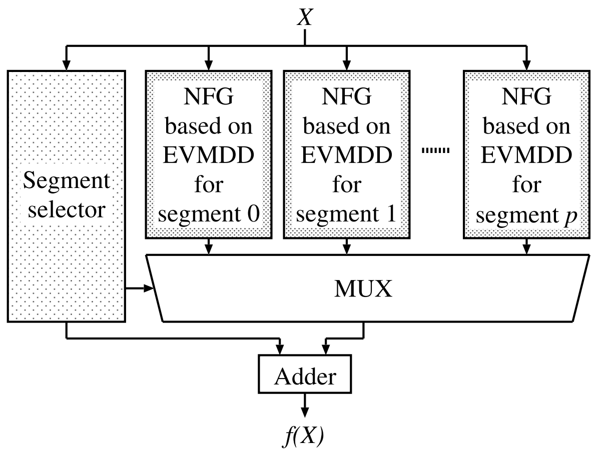

X
Segment selector
NFG based on EVMDD for segment 0
NFG based on EVMDD for segment 1
NFG based on EVMDD for segment p
MUX
Adder
f(X)

Figure 6. Architecture for NFG based on a piecewise-split EVMDD.

Fig. 4 shows the proposed segmentation algorithm. This algorithm decomposes a given EVBDD so that sizes of all sub-EVBDDs are smaller than or equal to a given threshold value, as shown in Fig. 5(a). And then, it produces a piecewise-split EVMDD as shown in Fig. 5(b). In Fig. 5(a), sub-EVBDDs whose heights are lower (i.e. in which fewer input variables are used) mean that their segments are narrower. In such segments, the floating-point function values rapidly change, and so, sizes of EVBDDs are large [15]. In Fig. 5(b), the multi-terminal EVMDD is used to select a segment, and the other EVMDDs are used to represent the floating-point function in each segment.

4. NFGs Based on Piecewise-Split EVMDDs

By realizing each EVMDD produced by the proposed segmentation algorithm using the architecture shown in Fig. 2, we obtain the NFG in Fig. 6. A value of the numeric function in each segment is computed using the least significant bits of X in parallel, and then an appropriate value is selected by the segment selector, which realizes a multi-terminal EVMDD. Since a piecewise-split EVMDD uses the most significant bits of X in parallel with function values, it is faster than the monolithic EVMDD.

In addition, the proposed NFG has the following advantages:

1. Since the proposed NFG just traverses a set of

Table 3. Memory size needed for NFGs based on a monolithic EVMDD and a piecewise-split EVMDD for single-precision (32-bit) floating-point numeric functions.

Functions $f(X)$	Memory size (Mbits)		Number of segments	Ratio (%)
	Monolithic EVMDD	Piecewise EVMDD		
$\sin^{-1}(X)$	34	12	28	35
$\ln(X)$	564	337	25	60
$1/X$	31	15	31	51
$\sqrt{X}$	34	13	23	39
$\sqrt{-\ln(X)}$	271	160	32	59

Ratio = $\frac{\text{(Piecewise EVMDD)}}{\text{(Monolithic EVMDD)}} \times 100$ (%).

EVMDDs, and computes the sum of edge weights (integers) in parallel, it requires only integer adders to compute function values of a floating-point function. That is, it requires neither the *rounding circuit* nor the *normalization circuit* (which are complex).

2. Since the NFG is a memory-based architecture, a wide range of numeric functions can be realized by changing only the data in the LUT memories.

3. Since the NFG directly realizes the function table of a floating-point function using a piecewise-split EVMDD, it is more accurate than existing NFGs using polynomial approximation [2,3,6,20,24].

4. The NFG is suitable for pipeline processing, and thus it can achieve a high throughput.

5. Experimental Results

To show the effectiveness of piecewise-split EVMDDs, we realize single-precision floating-point numeric functions using two types of NFGs, an NFG based on a monolithic EVMDD and an NFG based on a piecewise-split EVMDD. We compare memory size needed for the two types of NFGs. Table 3 shows memory size needed for the NFGs, in mega bits, and the number of segments for piecewise-split EVMDDs.

Since a single LUT that realizes a whole single-precision floating-point function requires $2^{32} \times 32 = 128$ Gbits, memory size needed for both types of NFGs is three or four orders of magnitude less than the single LUT-based NFG. And, by using piecewise-split EVMDDs, we can achieve a further reduction to 35% to 60% of memory size needed for the NFGs based on the monolithic EVMDDs. Table 3 shows that we can reduce memory size significantly with a small number of segments. Thus, the size of the multiplexer used in the NFG is also small. Further, we can generate such compact NFGs automatically.

6. Conclusion and Comments

This paper proposes a new architecture for memory-based floating-point NFGs, and a design method using piecewise-split EVMDDs. We also present an algorithm to produce efficient piecewise-split EVMDDs by decomposing the EVBDD representing a given floating-point function. Experimental results show that, for single-precision floating-point functions, our new NFGs based on piecewise-split EVMDDs require 40% to 65% less memory than ones based on monolithic EVMDDs. By using piecewise-split EVMDDs, we can automatically generate compact NFGs. Since our memory-based NFG is quite general, it can realize not only floating-point functions, but also discrete functions and even two-variable functions.

Future work includes 1) reducing memory size further, and 2) developing an optimization algorithm for decomposing an EVBDD. Our NFG requires still large memory size for FPGA implementation. We will further reduce memory size so that our single-precision floating-point NFG can be implemented with an FPGA. The proposed algorithm decomposes an EVBDD using a threshold value for the number of nodes. But, an algorithm that can find an optimum decomposition of EVBDD in terms of memory size or delay time of NFG is more practical.

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 Young Scientists (B), 200700051, 2009, and Hiroshima City University Grant for Special Academic Research (General Studies), 8108, 2009.

References

[1] 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.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.