J. of Mult.-Valued Logic & Soft Computing, Vol. 0, pp. 1–21
Reprints available directly from the publisher
Photocopying permitted by license only
©2013 Old City Publishing, Inc.
Published by license under the OCP Science imprint,
a member of the Old City Publishing Group.

Piecewise Arithmetic Expressions of Numeric Functions and Their Application to Design of Numeric Function Generators

SHINOBU NAGAYAMA[1], TSUTOMU SASAO[2] AND JON T. BUTLER[3]

[1]*Dept. of Comp. and Network Eng., Hiroshima City University, Hiroshima, JAPAN*
E-mail: s_naga@hiroshima-cu.ac.jp
[2]*Dept. of Comp. Science, Meiji Univresity, Kawasaki, JAPAN*
E-mail: sasao@cs.meiji.ac.jp
[3]*Dept. of Electr. and Comp. Eng., Naval Postgraduate School, Monterey, CA USA*
E-mail: jon_butler@msn.com

Received: September 28, 2011. Accepted: May 7, 2012.

In this paper, we propose a new representation of numeric functions using a piecewise arithmetic expression. To represent a numeric function compactly, we partition the domain of the function into uniform segments, and transform the sub-function in each segment into an arithmetic spectrum. From this arithmetic spectrum, we derive an arithmetic expression, and obtain a piecewise arithmetic expression for the function. By using the piecewise arithmetic expression, we can increase the number of zero arithmetic coefficients significantly, and represent a numeric function more compactly than using a conventional single arithmetic expression. We also present an application of the piecewise arithmetic expression to design of numeric function generators (NFGs). Since the piecewise arithmetic expression 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.

Keywords: Piecewise arithmetic expressions, nonzero arithmetic coefficients, numeric function generators (NFGs), programmable architectures.

This paper is an extended version of the paper [12].

1 INTRODUCTION

Numeric functions, such as trigonometric, logarithmic, square root, and combinations of these functions, have a wide range of applications including computer graphics, digital signal processing, communication systems, robotics, etc. [5]. These applications usually use numeric functions as a basic operation, as well as addition and multiplication. Particularly, in graphics applications for embedded systems, the computation of numeric functions accounts for about half of the total processing time [13]. 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 [22], and various NFGs have been proposed [2,4,14,17,18]. 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, 15, 21, 23]. However, for the design of NFGs, it is rarely used. Only two studies of NFGs using the arithmetic transform are known [16, 20]. However, in both papers, the NFG design is for a specific numeric function. That is, different architectures are required for different numeric functions.

Although such 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, that 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 [16, 20], the arithmetic transform is applied to the whole of a numeric function. However, this is unsuitable for the design of programmable NFGs because it requires too many additions. To design an efficient NFG, we partition the domain of a given numeric function into segments of equal widths, and apply the arithmetic transform to the sub-function for each segment. From the arithmetic spectrum obtained by the transform, we derive an arithmetic expression, and realize the arithmetic expression with memory 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 2 introduces a numeric representation of a real numeric function, and the arithmetic transform. Section 3 introduces conventional arithmetic expressions for numeric functions. Section 4 presents piecewise arithmetic expressions, and architectures for NFGs based on them. Experimental results are shown in Section 5. And, Section 6

presents techniques to reduce memory size and to improve the performance of NFGs.

2 PRELIMINARIES

2.1 Number Representation

This subsection defines functions used in this paper and a number representation, and describes how to convert real functions into other types of 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: $B^n \to B^m$, a (binary-input)* **integer-valued function** *is a mapping: $B^n \to \mathbb{Z}$, a* **real function** *is a mapping: $\mathbb{R} \to \mathbb{R}$, and a (binary-input)* **real-valued function** *is a mapping: $B^n \to \mathbb{R}$.*

Definition 2. *A* **numeric function** *is a real function built from a combination of constants, a variable, four arithmetic operations (+, -, *, /), power functions, exponential functions, logarithmic functions, trigonometric functions, and inverse trigonometric functions.*

To represent real values, this paper uses the following:

Definition 3. *A value X represented by the* **binary fixed-point representation** *is denoted by*

$$X = (x_{h-1}\ x_{h-2}\ \ldots\ x_1\ x_0.\ x_{-1}\ x_{-2}\ \ldots\ x_{-m})_2,$$

where $x_i \in \{0, 1\}$ for $h - 1 \geq i \geq -m$, h is the number of bits for the integer part, and m is the number of bits for the fractional part of X. We call

$$X = \sum_{i=-m}^{h-1} 2^i x_i$$

*an n-***bit fixed-point representation** *in which n bits are used to represent the value, where $n = h + m$. 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-valued function by considering binary vectors as integers. That is,

(a) Table for sin(X).		(b) Truth table for $f_b(X)$.		(c) Table for $f_i(X)$.		(d) Table for $f_r(X)$.	
X	sin(X)	$X = (0.x_2x_1x_0)_2$	$f_b(X)$	$x_2x_1x_0$	$f_i(X)$	$x_2x_1x_0$	$f_r(X)$
0.000	0.000	0.000	0.000	000	0	000	0.000
0.125	0.125	0.001	0.001	001	1	001	0.125
0.250	0.247	0.010	0.010	010	2	010	0.247
0.375	0.366	0.011	0.011	011	3	011	0.366
0.500	0.479	0.100	0.100	100	4	100	0.479
0.625	0.585	0.101	0.101	101	5	101	0.585
0.750	0.682	0.110	0.101	110	5	110	0.682
0.875	0.768	0.111	0.110	111	6	111	0.768

TABLE 1
Function table for 3-bit sin(X).

we can convert a real function into an integer-valued function: $B^n \to P_m$, where $P_m = \{0, 1, \ldots, 2^m - 1\}$, by using a fixed-point representation. And, similarly, by using a fixed-point representation, we can obtain a real-valued function: $B^n \to \mathbb{R}$.

For simplicity, the fixed-point representation of X is denoted by $X = (x_{n-1}x_{n-2}\ldots x_0)_2$.

Example 1. *Table 1 (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 1 (b). By representing the output vectors as integers, we have the integer-valued function* $f_i(X)$ *in Table 1 (c). And, Table 1 (d) shows a real-valued function.* ■

2.2 Rounding Error and Numeric Function Generator

Definition 4. **Error** *is the absolute difference between the exact value and an approximated value.* **Rounding error** *is an error that is caused when a real value is represented by the fixed-point representation with a finite number of fractional bits. It is no greater than* $2^{-(m+1)}$ *when m fractional bits are used to represent a real value.*

Definition 5. *A* **numeric function generator** *or NFG is a logic circuit that computes approximated values for a numeric function in the fixed-point representation.*

NFGs are usually designed so as to achieve $2^{-(m+1)}$ or 2^{-m} worst case error, where m is the number of fractional bits in the output of NFGs.

2.3 Arithmetic Transform

This subsection introduces the arithmetic transform, the arithmetic spectrum, and the arithmetic expression [19].

First, define a matrix operation.

Definition 6. *Let* $\boldsymbol{A}$ *be an* $(n \times n)$ *square matrix, where*

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

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

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

Definition 7. *Given a matrix* $\boldsymbol{M}$*, the* **transposed matrix** $\boldsymbol{M}^t$ *is obtained by interchanging rows and columns of* $\boldsymbol{M}$*. For a binary-input function* $f(X)$ *(i.e. integer-valued or real-valued function), the* **function-vector** $\boldsymbol{F}$ *is the column vector of the function values* $\boldsymbol{F} = [f(00\dots0), f(00\dots01), \dots, f(11\dots1)]^t$.

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

Definition 8. *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 a binary-input function f *given by the function-vector* $\boldsymbol{F}$*, the* **arithmetic spectrum** $\mathcal{A}_f = [a_0, a_1, \dots, a_{2^n-1}]^t$ *is*

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

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* $\boldsymbol{F} = [0, 1, 1, 2]^t$*. The arithmetic spectrum is*

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

■

Similarly, we define the inverse arithmetic transform as follows:

Definition 9. *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 10. *In a symbolic representation,*

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

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

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

Example 3. *By the inverse arithmetic transform from the arithmetic spectrum obtained in Example 2, the function* f *is represented as follows:*

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

■

From Definitions 9 and 10, we can see that a binary-input 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 $\mathcal{A}(1)$, a binary-input function f is represented as follows:*

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

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]). By applying the arithmetic expansion to f recursively, we obtain the* **arithmetic expression** *for f:*

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

where coefficients a_i of the arithmetic expression for f are the arithmetic coefficients.

Example 4. *The arithmetic spectrum of the function f_i in Table 1(c) is* $[0, 1, 2, 0, 4, 0, -1, 0]^t$*, and thus, the arithmetic expression for f_i is*

$$x_0 + 2x_1 + 4x_2 - x_2x_1.$$

The arithmetic spectrum of the function f_r in Table 1(d) is $[0.000, 0.125, 0.247, -0.006, 0.479, -0.019, -0.044, 0.026]^t$*, and the arithmetic expression for f_r is*

$$\begin{aligned} 0.125x_0 &+ 0.247x_1 - 0.006x_1x_0 + 0.479x_2 - 0.019x_2x_0 - 0.044x_2x_1 \\ &+ 0.026x_2x_1x_0. \end{aligned}$$

■

3 ARITHMETIC EXPRESSIONS FOR NUMERIC FUNCTIONS

There are two ways to obtain arithmetic expressions from given numeric functions. One way is to obtain them by applying the arithmetic transform directly to real-valued functions of given functions. The arithmetic

expressions produced by this way have real-valued arithmetic coefficients. Thus, in this paper, we call them **real-valued arithmetic expressions**.

Another way is by applying the arithmetic transform to integer-valued functions. Since this produces the arithmetic expressions with integer coefficients, we call them **integer-valued arithmetic expressions**.

3.1 Numeric Function Generators Based on Arithmetic Expressions

Integer-valued arithmetic expressions can be directly realized without any rounding error using only AND gates and adders [16, 20]. Since rounding error is caused only when a given numeric function is converted into an integer-valued function, the error of an NFG based on the integer-valued arithmetic expression is at most $2^{-(m+1)}$, where m is the number of fractional bits in the output of the NFG.

Similarly, real-valued arithmetic expressions can be realized using only AND gates and adders. But, to realize real-valued arithmetic expressions, all real-valued arithmetic coefficients have to be rounded to a finite number of fractional bits. Let l be this number, and let n be the number of input bits. Since the number of arithmetic coefficients is 2^n, the worst case error to realize a real-valued arithmetic expression is $2^n \times 2^{-(l+1)} = 2^{-(l-n+1)}$. In addition, a resulting value of the expression is also rounded to m fractional bits in the output of an NFG based on the real-valued arithmetic expression. Since this rounding can cause at most $2^{-(m+1)}$ error, the worst case error of the NFG is $2^{-(m+1)} + 2^{-(l-n+1)}$. Thus, the NFG cannot achieve a $2^{-(m+1)}$ worst case error. To achieve a 2^{-m} worst case error, $l = m + n$ bits are required for each arithmetic coefficient. That is, real-valued arithmetic expressions require more bits to represent each arithmetic coefficient but achieve a larger worst case error than integer-valued arithmetic expressions.

However, in either arithmetic expression, the size of the NFGs is strongly dependent on the number of zero arithmetic coefficients. When many arithmetic coefficients are zero, an arithmetic expression can be realized with a compact circuit because the product terms with zero coefficients in the arithmetic expression can be eliminated. Thus, in the next subsection, we consider the number of zero arithmetic coefficients.

3.2 Number of Zero Arithmetic Coefficients

The number of zero arithmetic coefficients in an arithmetic expression depends on the nonlinearity of the original numeric function. Since differentiable numeric functions can be expanded into polynomial functions using Maclaurin expansion, Taylor expansion, Chebyshev expansion, and so on, we consider polynomial functions to investigate a correlation between the number of zero arithmetic coefficients and the nonlinearity of numeric functions.

For polynomial functions, the following lemma holds:

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

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

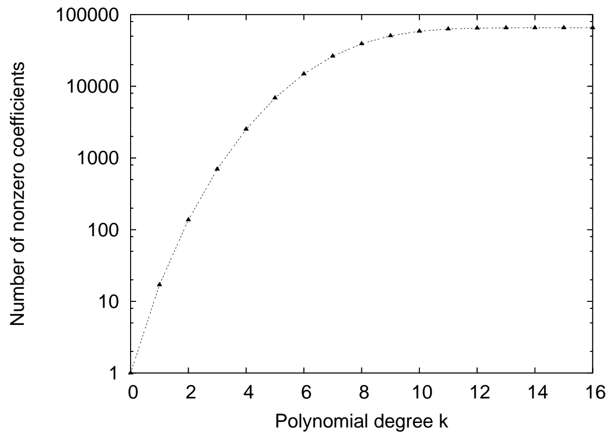
From this lemma, we can see that an upper bound on the number of nonzero arithmetic coefficients increases with k when the number of bits n is fixed, as shown in Figure 1(a). Also, as shown in Figure 1(b), in which the solid line denotes the number of nonzero arithmetic coefficients and the dashed line denotes the total number of arithmetic coefficients (i.e., 2^n), the number of *zero* coefficients increases with n when k is fixed. That is, when k is sufficiently smaller than n, the number of zero coefficients becomes larger.

Thus, a lower polynomial degree tends to produce more zero arithmetic coefficients. In other words, numeric functions that are close to linear, as shown in Figure 2(a), have more zero arithmetic coefficients. On the other hand, highly nonlinear numeric functions as shown in Figure 2(b) have fewer zero arithmetic coefficients.

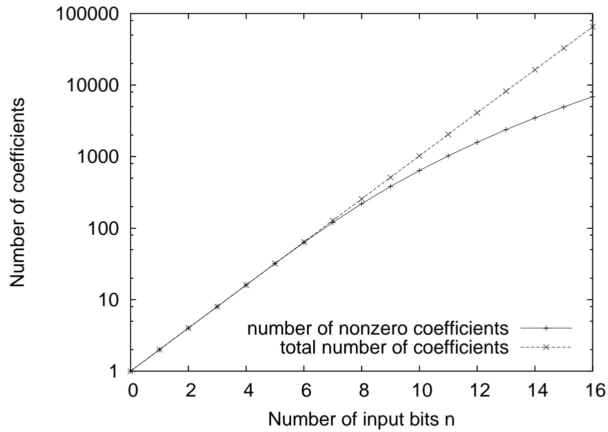
Table 2 shows the number of arithmetic coefficients for various 16-bit numeric functions. In this table, the column "Integer-valued" shows the number of arithmetic coefficients in an integer-valued arithmetic expression, and the column "Real-valued" shows the number of arithmetic coefficients in a real-valued arithmetic expression. For integer-valued arithmetic expressions, values of numeric functions are rounded to 16 fractional bits, and they can achieve 2^{-17} worst case error. On the other hand, for real-valued arithmetic expressions, arithmetic coefficients are rounded to 32 fractional bits to achieve 2^{-16} worst case error.

Table 2 shows that real-valued arithmetic expressions have more zero coefficients than integer-valued arithmetic expressions. This is because real-valued arithmetic expressions are obtained by the arithmetic transform using values of original numeric functions, and thus, the nonlinearity of numeric functions directly affects arithmetic coefficients. That is, for real-valued expressions, the number of zero arithmetic coefficients increases significantly when the original numeric functions are close to linear functions.

However, real-valued arithmetic expressions require longer bit length for each arithmetic coefficient. This increases the number of distinct nonzero arithmetic coefficients, since the probability that the same coefficients occur becomes lower.



(a) 16-bit kth-degree polynomial functions.



(b) n-bit 5th-degree polynomial functions.

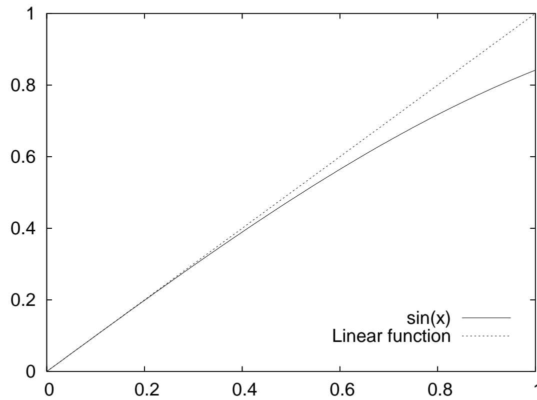
FIGURE 1
Number of nonzero coefficients for polynomial functions.

4 PIECEWISE ARITHMETIC EXPRESSIONS FOR NUMERIC FUNCTIONS

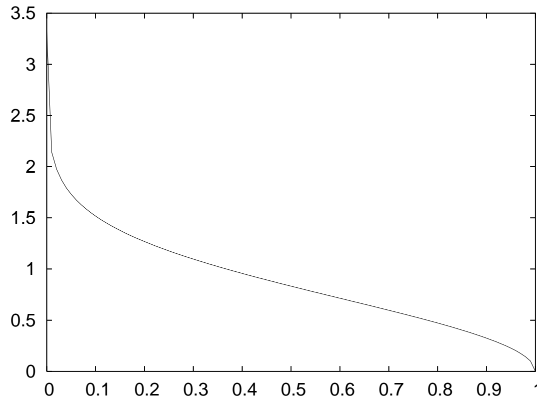
This section introduces piecewise arithmetic expressions, and presents programmable architectures for NFGs based on them.

4.1 Piecewise Arithmetic Expressions

As described in the previous section, arithmetic expressions can be realized with only AND gates and adders, and thus, they are realized with compact



(a) $\sin(X)$ $(0 \leq X < 1)$



(b) $\sqrt{-\ln(X)}$ $(0 < X < 1)$

FIGURE 2
Graphs of two numeric functions.

circuits when many arithmetic coefficients are zero. Since many standard numeric functions, such as trigonometric and square root functions, have many zero arithmetic coefficients, we can design compact NFGs for them [16, 20].

However, as shown in Table 2, highly nonlinear numeric functions, such as $\sqrt{-\ln(X)}$ and $-X\ln(X)$, have few zero coefficients. Even for numeric functions close to linear functions, fixed-point representations with many bits (i.e., high precision) necessarily yield arithmetic expressions with many product terms. In both cases, the resulting NFGs are large and slow. In addition, a straightforward programmable implementation of the NFGs proposed in [16, 20], as shown in Figure 3, needs too many adders ($2^n - 1$ adders).

Functions	Integer-valued		Real-valued	
$f(X)$	**Zero**	**Distinct**	**Zero**	**Distinct**
$\sqrt{-\ln(X)}$	0	8,235	0	20,365
$-X\ln(X)$	2,436	746	1	13,538
2^X	6,663	148	29,938	616
e^X	8,985	172	28,118	823
$\ln(X+1)$	8,800	164	31,287	1,190
$\log_2(X+1)$	6,991	157	31,195	1,278
$1/(X+1)$	9,312	186	31,270	1,557
$\sqrt{X+1}$	9,810	138	32,752	824
$\sin(X)$	20,012	140	34,718	636
$\tan(X)$	14,921	161	9,657	1,317
$\sin^{-1}(X)$	16,112	180	15,293	2,171
$\tan^{-1}(X)$	16,703	151	30,796	1,098

TABLE 2
Number of arithmetic coefficients for 16-bit numeric functions.

Zero: the number of zero coefficents.
Distinct: the number of distinct nonzero coefficients.
Domain of functions is $0 \leq X < 1$.

To reduce the number of product terms (adders), we transform subfunctions 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

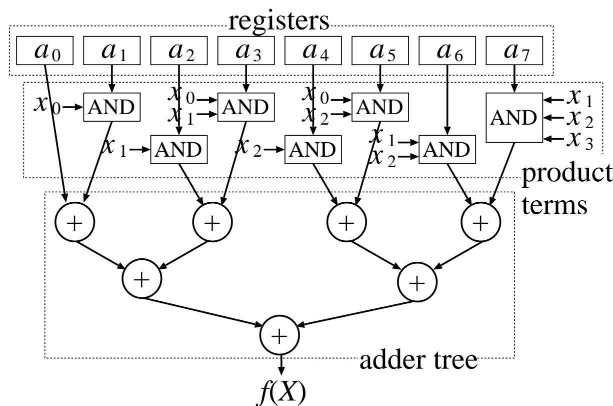


FIGURE 3
A 3-bit programmable NFG based on an arithmetic expression.

transform to the sub-function in 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. Thus, the MSBs are used to select a segment, and the remaining lower bits of X are used to compute an arithmetic expression for the segment. Other ways exist to select a segment, but this way is especially efficient.

Let k be the number of bits to compute an arithmetic expression for each segment. Then, arithmetic coefficients in a real-valued arithmetic expression for each segment are rounded to $l = m + k$ fractional bits because the number of arithmetic coefficients is 2^k. Since k is smaller than the number of input bits n, the piecewise real-valued arithmetic expression can achieve 2^{-m} worst case error with fewer bits for coefficients, compared to the single real-valued arithmetic expression.

4.2 Number of Arithmetic Coefficients in Piecewise Arithmetic Expressions

Table 3 shows the number of arithmetic coefficients in piecewise arithmetic expressions for various 16-bit numeric functions. In this table, the column "Integer-valued" shows the number of arithmetic coefficients in a piecewise integer-valued arithmetic expression, and the column "Real-valued" shows

Functions	**Integer-valued**		**Real-valued**	
$f(X)$	**Zero**	**Distinct**	**Zero**	**Distinct**
$\sqrt{-\ln(X)}$	25,318	866	39,835	3,073
$-X\ln(X)$	28,308	527	29,866	2,408
2^X	30,012	445	19,593	2,256
e^X	28,946	587	17,251	2,391
$\ln(X+1)$	28,622	402	36,212	2,164
$\log_2(X+1)$	30,467	451	33,803	2,256
$1/(X+1)$	27,834	401	33,331	2,023
$\sqrt{X+1}$	28,220	323	34,535	1,791
$\sin(X)$	34,209	391	27,866	2,060
$\tan(X)$	32,139	548	4,153	2,297
$\sin^{-1}(X)$	33,477	532	10,356	2,362
$\tan^{-1}(X)$	33,073	401	28,638	2,127

TABLE 3
Number of arithmetic coefficients in piecewise arithmetic expressions for 16-bit numeric functions.

Zero: the number of zero coefficents.
Distinct: the number of distinct nonzero coefficients.
Domain of functions is $0 \leq X < 1$.
The number of MSBs for uniform segmentation is 8.

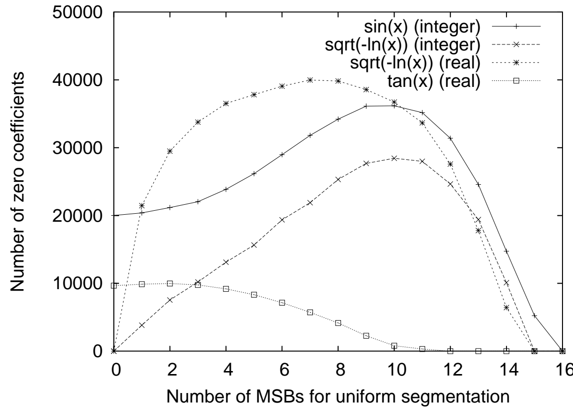
the number of arithmetic coefficients in a piecewise real-valued arithmetic expression. For piecewise integer-valued arithmetic expressions, values of numeric functions are rounded to 16 fractional bits. On the other hand, for piecewise real-valued arithmetic expressions, arithmetic coefficients are rounded to 24 fractional bits because 8 bits are used to compute an expression for each segment.

From this table, we can see that using piecewise arithmetic expressions increases the number of zero arithmetic coefficients significantly, even for highly nonlinear functions. This is because in a segmented local domain, numeric functions become close to linear functions. However, for exponential and trigonometric functions, the number of zero coefficients in piecewise real-valued arithmetic expressions is smaller than that in single real-valued arithmetic expressions.

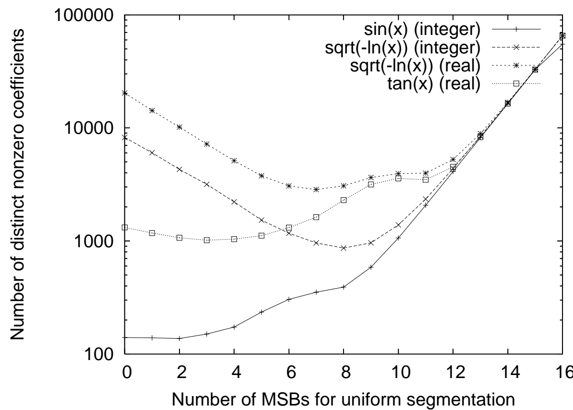
For these functions, polynomial degrees obtained by a sufficiently accurate polynomial expansion of the segmented numeric functions are not much different than the polynomial degree obtained from the original whole function. Since a real-valued arithmetic expression represents even slight curves of a numeric function in a local domain faithfully, the piecewise real-valued arithmetic expressions for these functions equivalently represent polynomial functions with the same degrees as the original functions but with fewer bits. As shown in Figure 1(b), if the polynomial degree is not changed, the number of zero coefficients tends to decrease with decreasing number of input bits. Therefore, for piecewise real-valued arithmetic expressions, the number of zero coefficients decreases since only the least significant bits (LSBs) of X are used for the expression in each segment.

In this way, increasing the number of segments can increase the number of zero coefficients, but too many segments result in fewer zero coefficients. That is, there is an optimum number of segments (i.e., an optimum number of MSBs) to produce the largest number of zero coefficients, depending on the numeric function. Figure 4(a) shows the optimum numbers for piecewise integer-valued arithmetic expressions of $\sin(X)$ and $\sqrt{-\ln(X)}$, and for real-valued expressions of $\sqrt{-\ln(X)}$ and $\tan(X)$. For piecewise integer-valued arithmetic expressions of $\sin(X)$ and $\sqrt{-\ln(X)}$, 10 MSBs are optimum. For real-valued expressions of $\sqrt{-\ln(X)}$ and $\tan(X)$, 7 MSBs and 2 MSBs are optimum, respectively.

Figure 4(b) shows the number of distinct nonzero coefficients in the four piecewise arithmetic expressions. For piecewise integer-valued arithmetic expressions of $\sin(X)$ and $\sqrt{-\ln(X)}$, 2 MSBs and 8 MSBs are optimum to produce the smallest number of distinct nonzero coefficients. For real-valued expressions of $\sqrt{-\ln(X)}$ and $\tan(X)$, 7 MSBs and 3 MSBs are optimum, respectively.



(a) Number of zero coefficients



(b) Number of distinct nonzero coefficients

FIGURE 4
Relation between the number of coefficients in piecewise arithmetic expressions and the number of MSBs.

4.3 Architectures for Programmable NFGs Based on Piecewise Arithmetic Expressions

By realizing a set of the arithmetic spectra for a piecewise arithmetic expression with a memory (called arithmetic coefficients table), we obtain the NFG in Figure 5. The MSBs of X select a segment (an arithmetic spectrum), and then an arithmetic expression is computed using the 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 an NFG based on the single arithmetic expression in Figure 3, in which $2^n - 1$ adders are required.

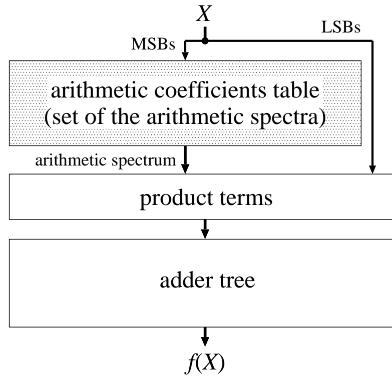


FIGURE 5
Programmable NFG based on the piecewise arithmetic expression.

Unfortunately, the number of adders is still large, and this design is inefficient. Since the arithmetic spectra usually have many zero coefficients as shown in Table 3, 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 Figure 6. Note that, for readability of the figures, the reset and enable signals for registers and the done signal (to denote completion of the computation) are omitted.

In this architecture, only the *nonzero* arithmetic coefficients are stored in a table for each segment. By reading out each coefficient sequentially, it

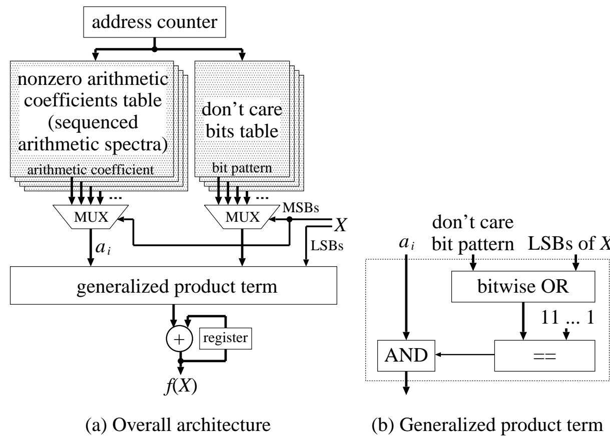


FIGURE 6
Programmable NFG based on a sequential computation.

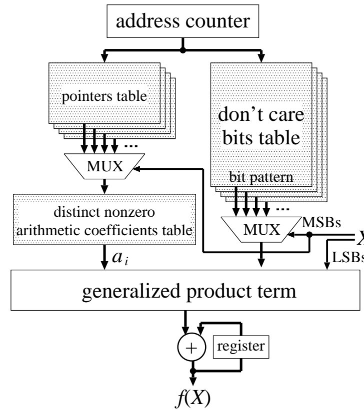


FIGURE 7
Improved architecture for programmable NFG.

computes the arithmetic expression using an accumulator. Each product term is computed with the circuit in Figure 6(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. 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. Figure 7 shows an 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.

The proposed architectures can realize both integer-valued and real-valued arithmetic expressions. However, real-valued arithmetic expressions require longer bit length for each coefficient and larger number of distinct nonzero coefficients, and thus, they are unsuitable for hardware implementation. Therefore, in the following, we focus only on integer-valued arithmetic expressions. And, the proposed architectures use a fixed number of MSBs to select a segment. As shown in the previous subsection, there is an optimum number of MSBs to minimize the number of distinct nonzero arithmetic coefficients, depending on numeric functions. However, we have to consider not only the number of arithmetic coefficients but also the size of the multiplexers with the number of additions for the proposed architectures. Therefore, in this paper, we use about one-half of the input bits for uniform segmentation to balance the size of the multiplexers with the number of additions.

Functions	No. of stored coefficients			Size of coefficients table			Total size of tables			No. of additions	
$f(X)$	NFG1 (all)	NFG2 (nonzero)	NFG3 (distinct)	NFG1 (bits)	NFG2 (bits)	NFG3 (bits)	NFG1 (bits)	NFG2 (bits)	NFG3 (bits)	NFG1	NFG2,3 (average)
$\sqrt{-\ln(X)}$	65,536	40,218	866	1,179,648	723,924	15,588	1,179,648	1,085,886	779,730	255	157.1
$-X\ln(X)$	65,536	37,228	527	983,040	558,420	7,905	983,040	893,472	715,237	255	145.4
2^X	65,536	35,524	445	1,114,112	603,908	7,565	1,114,112	923,624	646,997	255	138.8
e^X	65,536	36,590	587	1,179,648	658,620	10,566	1,179,648	987,930	705,776	255	142.9
$\ln(X+1)$	65,536	36,914	402	1,048,576	590,624	6,432	1,048,576	922,850	670,884	255	144.2
$\log_2(X+1)$	65,536	35,069	451	1,048,576	561,104	7,216	1,048,576	876,725	638,458	255	137.0
$1/(X+1)$	65,536	37,702	401	1,114,112	640,934	6,817	1,114,112	980,252	685,453	255	147.3
$\sqrt{X+1}$	65,536	37,316	323	1,114,112	634,372	5,491	1,114,112	970,216	677,179	255	145.8
$\sin(X)$	65,536	31,327	391	1,048,576	501,232	6,256	1,048,576	783,175	570,142	255	122.4
$\tan(X)$	65,536	33,397	548	1,114,112	567,749	9,316	1,114,112	868,322	643,859	255	130.5
$\sin^{-1}(X)$	65,536	32,059	532	1,114,112	545,003	9,044	1,114,112	833,534	618,165	255	125.2
$\tan^{-1}(X)$	65,536	32,463	401	1,048,576	519,408	6,416	1,048,576	811,575	590,750	255	126.8

TABLE 4
Table sizes and the number of additions for 16-bit NFGs.

NFG1: the NFG shown in Figure 5. NFG2: the NFG shown in Figure 6. NFG3: the NFG shown in Figure 7.
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 $0 \leq X < 1$.

5 EXPERIMENTAL RESULTS

To show the efficiency of the proposed NFGs, we compare the table sizes and the numbers of additions for the three proposed NFGs. Table 4 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 Figures 6 and 7, the numbers of product terms in arithmetic expressions for different segments are different, the average number of additions for each expression is shown.

Since the programmable NFG based on the single arithmetic expression in Figure 3 requires $2^{16} = 65,536$ registers and $2^{16} - 1 = 65,535$ adders, the proposed NFGs, based on the piecewise arithmetic expressions, require several orders of magnitude fewer adders, and much less storage size. As shown in Table 4, 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 tables, resulting in a reduction of total size of tables.

6 IMPROVEMENT TECHNIQUES FOR NFGS

6.1 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 Lemma 2.

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 NFG.

Example 5. *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. ■

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

6.2 Parallel Computation

The proposed NFGs based on a sequential computation in Figures 6 and 7 produce an arithmetic coefficient one by one, and compute each product term of an arithmetic expression sequentially using only one adder. Thus, they require $O(N)$ computation time, where N is the number of product terms. As shown in Table 4, they require about 120 to 160 additions on average, to compute a numeric function value, even though many product terms are reduced by zero coefficients.

On the other hand, the NFG in Figure 5 produces all arithmetic coefficients simultaneously, and adds all product terms of an arithmetic expression at once using an adder tree. Thus, it requires $O(\log N)$ computation time. In Table 4, it computes a function value in delay time for an 8-level adder tree with 255 adders. Therefore, the NFG in Figure 5 is faster but requires many more adders than the proposed NFGs. 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 product terms to be computed in parallel, we can explore the design space taking into account a trade-off between the number of adders and the computation time, and can produce an optimum NFG depending on applications.

7 CONCLUSION AND COMMENTS

This paper proposes a new representation of numeric functions using a piecewise arithmetic expression and new architectures for programmable NFGs based on the piecewise arithmetic expression. By using a piecewise arithmetic expression, we can increase the number of zero arithmetic coefficients significantly, and design a more compact NFG. Experimental results show that the size of the arithmetic coefficients table in an NFG 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.

The proposed design method for the NFGs can explore the design space taking into account a trade-off between the number of adders and the computation time by changing the number of product terms to be computed in parallel. To easily find an optimum NFG for each application, we will define a figure of merit (FoM) for NFGs in our future work.

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), 2011, and Hiroshima City University Grant for Special Academic Research (General Studies), (No. 0206), 2011. We would like to thank Prof. Radomir Stankovic and Prof. Osnat Keren for discussions that motivated this work. The comments of the reviewers were useful in improving the paper.

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.v

[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] S. Nagayama, T. Sasao, and J. T. Butler, "Numeric function generators using piecewise arithmetic expressions," *41st International Symposium on Multiple-Valued Logic*, pp. 16-21, May, 2011.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] R. Stankovic, T. Sasao, and C. Moraga, "Spectral transform decision diagrams," Chapter 3 in [15].

[20] 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.

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

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

[23] 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.