Numerical Function Generators Using Edge-Valued Binary Decision Diagrams

Shinobu Nagayama
Dept. of Computer Engineering,
Hiroshima City University,
Hiroshima 731-3194, Japan

Tsutomu Sasao
Dept. of Computer Science
and Electronics,
Kyushu Institute of Technology,
Iizuka 820-8502, Japan

Jon T. Butler
Dept. of Electrical and Computer
Engineering,
Naval Postgraduate School,
CA 93943-5121, USA

Abstract— In this paper, we introduce the edge-valued binary decision diagram (EVBDD) to reduce the memory and delay in numerical function generators (NFGs). An NFG realizes a function, such as a trigonometric, logarithmic, square root, or reciprocal function, in hardware. NFGs are important in, for example, digital signal applications, where high speed and accuracy are necessary. We use the EVBDD to produce a fast and compact segment index encoder (SIE) that is a key component in our NFG. We compare our approach with NFG designs based on multi-terminal BDD's (MTBDDs), and show that the EVBDD produces SIEs that have, on average, only 7% of the memory and 40% of the delay of those designed using MTBDDs. Therefore, our NFGs based on EVBDDs have, on average, only 38% of the memory and 59% of the delay of NFGs based on MTBDDs.

I. Introduction

There has been significant interest recently in the realization of numeric functions, like $\sin(\pi x)$, $\ln(x)$, $1/x$, and $\sqrt{x}$, by high speed logic circuits. This is, in part, due to the availability of large quantities of inexpensive, programmable logic in FPGA's, and, in part, to the development of realization methods based on polynomial approximations [3, 5, 6, 8, 16, 22, 23]. Until the last few years, the dominant approach has been to partition the domain into *uniform* segments. Within each segment a linear [23] or higher order [3, 5, 6, 8, 16, 22] approximation is used to represent the function. It has been shown [7] that linear approximation is well suited for certain 'simple' functions like 2^x, $\sin(\pi x)$, and $\cos(\pi x)$, but is inappropriate for 'complex' functions like $\sqrt{x}$ and the entropy function, $-(x\log_2(x)+(1-x)\log_2(1-x))$. For complex functions, *optimum non-uniform* segmentation produces tractable realizations [2]. In this method, segments are chosen as wide as possible, while still achieving the specified accuracy. Typically, narrow segments are used where the function changes rapidly and wide segments are used in other regions. A segment index encoder (SIE) is therefore needed to map the values in the domain to the segments. Within each segment the coefficients are the same for all points in the segment. As in uniform segmentation, a memory stores the coefficient values, which are then used to form the polynomial approximation. Potentially, the SIE designed for an optimum non-uniform segmentation is a complex circuit. To simplify the SIE, two approaches have been proposed. One is a segmentation approach [10, 11] that uses a *special (non-optimum) non-uniform* segmentation. Another one is a realization approach [20, 21] that uses an *LUT cascade* to realize the optimum non-uniform segmentation compactly.

In this paper, we use both approaches to simplify the SIE. That is, this paper proposes a new segmentation approach and a new realization approach using an edge-valued binary decision diagram (EVBDD). Our segmentation approach can also reduce the memory size and the delay time of an LUT cascade using a multi-terminal BDD (MTBDD). For both approaches, we establish a formal synthesis procedure that is easily programmed.

II. Preliminaries

A. Number Representation and Precision

Definition 1 *A value X represented by the **binary fixed-point representation** is denoted by* $X = (x_{l-1}\ x_{l-2}\ \ldots\ x_1\ x_0.\ x_{-1}\ x_{-2}\ \ldots\ x_{-m})_2$, *where* $x_i \in \{0,1\}$, *l is the number of bits for the integer part, and m is the number of bits for the fractional part. This representation is two's complement.*

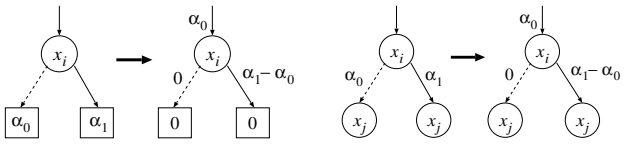
Definition 2 ***Error** is the absolute difference between the exact value and the value produced by the hardware. **Approximation error** is the error caused by a function approximation. **Rounding error** is the error caused by a binary fixed-point representation. **Acceptable error** is the maximum error that an NFG may assume. **Acceptable approximation error** is the maximum approximation error that a function approximation may assume.*

Definition 3 ***Precision** is the total number of bits for a binary fixed-point representation. Specially, **n-bit precision** specifies that n bits are used to represent the number; that is,* $n = l + m$. *We assume that an **n-bit precision NFG** has an n-bit input.*

Definition 4 ***Accuracy** is the number of bits in the fractional part of a binary fixed-point representation. **m-bit accuracy** specifies that m bits are used to represent the fractional part of the number. When the maximum error is* 2^{-m}, *the accuracy can be expressed as* 1 ***unit in the last place (ULP)**. In this paper, an **m-bit accuracy NFG** is an NFG with an m-bit fractional part of the input, an m-bit fractional part of the output, and* 1 *ULP.*

B. Edge-Valued Binary Decision Diagram

Definition 5 *A binary decision diagram (BDD) [1] is a rooted directed acyclic graph representing a logic function:* $\{0,1\}^n \rightarrow \{0,1\}$. *The BDD is obtained by repeatedly applying the Shannon expansion to the logic function. Each function, including the original function and all sub-functions resulting from applying the Shannon expansion, is represented by a non-terminal node, unless that function is a trivial function,* 0 *or* 1, *in which case, it is represented by a terminal node. Each non-terminal node has two outgoing edges,* 0*-edge and* 1*-edge, that correspond to the values of input variables. Both terminal nodes have no outgoing edges.*

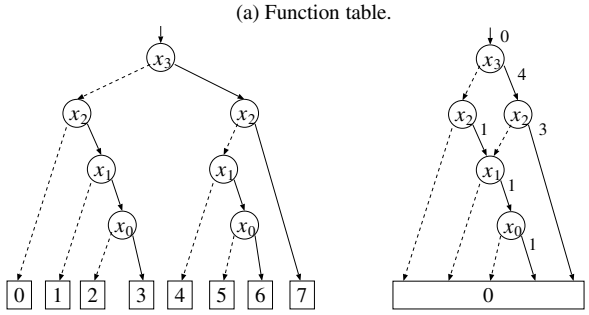


(a) Terminal case. (b) Non-terminal case.

Fig. 1. Conversion of an MTBDD node into an EVBDD node.

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

(a) Function table.



(b) MTBDD. (c) EVBDD.

Fig. 2. MTBDD and EVBDD for an integer function.

Definition 6 *A **multi-terminal BDD (MTBDD)** [4] is an extension of the BDD, and represents an integer function: $\{0,1\}^n \rightarrow Z$, where Z is a set of integers. Specifically, it is a BDD in which the terminal nodes are not restricted to* 0 *and* 1*. Rather, they are labeled by integer values. Alternatively, we can think of a BDD as a special case of an MTBDD, in which there are only two terminal nodes, labeled* 0 *and* 1*.*

Definition 7 *An **edge-valued BDD (EVBDD)** [9] is an extension of the BDD, and represents an integer function. An EVBDD consists of one terminal node representing* 0 *and non-terminal nodes with a weighted* 1*-edge, where the weight is an integer. An EVBDD is obtained by recursively applying the conversion shown in Fig. 1 to each non-terminal node in an MTBDD, where in Fig. 1, dashed lines and solid lines denote* 0*-edges and* 1*-edges, respectively. Note that, in the EVBDD,* 0*-edges (dashed lines) have weight* 0*, while the incoming edge into the root node can have some weight.*

For more detail on these BDDs, refer to [17].

Example 1 *Fig. 2(b) and (c) show an MTBDD and an EVBDD for the integer function f defined by Fig. 2(a). In Fig. 2, dashed lines and solid lines denote* 0*-edges and* 1*-edges, respectively. Note that the EVBDD has weighted* 1*-edges. In the MTBDD, terminal nodes represent function values. Thus, to evaluate the function, we traverse the MTBDD from the root node to a terminal node according to the input values, and obtain the function value (an integer) from the terminal node. On the other hand, in the EVBDD, we obtain the function value by summing the weights of the edges traversed from the root node to the terminal node. (End of Example)*

Input:	Numerical function $f(X)$, domain $[A,B)$ for X, accuracy m_{in} of X, polynomial order d, and acceptable approximation error ε_a.
Output:	Segments $[A,P_0),[P_0,P_1),\ldots,[P_{t-2},B)$.

Step:

1. For $[A,B)$, compute the maximum approximation error $\varepsilon_d(A,B)$.
2. If $\varepsilon_d(A,B) < \varepsilon_a$ or $B-A \leq 2^{-m_{in}}$, then stop.
3. Else, partition $[A,B)$ into two segments $[A,P)$ and $[P,B)$, where $P=(A+B)/2$.
4. Repeat Steps 1, 2, and 3 for each new segment recursively, until the maximum approximation errors are smaller than ε_a in all segments.

Fig. 3. Recursive segmentation algorithm for the domain.

III. Piecewise Polynomial Approximation Based on Non-uniform Segmentation

To approximate the numerical function $f(X)$ using polynomial functions, we first partition the domain for X into segments. For each segment, we approximate $f(X)$ using a polynomial function specific to that segment. In many cases, the domain is partitioned into *uniform segments*. Such methods are useful for elementary functions, such as $\sin(\pi X)$, but for some numerical functions, such as $-(X\log_2(X)+(1-X)\log_2(1-X))$, too many segments are required, resulting in large memory. To reduce the number of segments, we use a non-uniform segmentation, called *recursive segmentation*.

A. Recursive Segmentation Algorithm

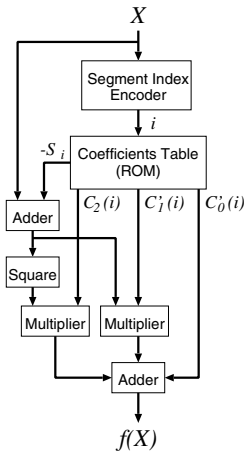
Fig. 3 shows a recursive segmentation algorithm. The inputs for this algorithm are a numerical function $f(X)$, a domain $[A,B)$ for X, an accuracy m_{in} of X, a polynomial order d, and an acceptable approximation error ε_a. Then, this algorithm produces t segments $[A,P_0),[P_0,P_1),\ldots,[P_{t-2},B)$ by recursively partitioning a segment into two equal-sized segments until achieving the acceptable approximation error ε_a in all segments. Note that this algorithm restricts the width w_i of each segment to $w_i = 2^{h_i} \times 2^{-m_{in}}$, where h_i is an integer. That is, the segmentation points P_i are restricted to values of which the least significant h_i bits are 0 (i.e., $P_i = (\ldots\ p_{-j+1}\ p_{-j}\ 00\ \ldots\ 0)_2$, where $j = m_{in} - h_i$). As shown in Fig. 3, the number of segments depends on the maximum approximation error $\varepsilon_d(A,B)$. In this paper, we use the Chebyshev approximation polynomials. For a segment $[S,E]$ of $f(X)$, the maximum approximation error of the dth-order Chebyshev approximation $\varepsilon_d(S,E)$ is given by [12]:

$$\varepsilon_d(S,E) = \frac{2(E-S)^{d+1}}{4^{d+1}(d+1)!} \max_{S\leq X\leq E} |f^{(d+1)}(X)|,$$

where $f^{(d+1)}$ is the $(d+1)$th-order derivative of f.

B. Computation of the Approximate Value

For each segment, $f(X)$ is approximated by the corresponding polynomial function $g(X,i)$. That is, the approximated value of $f(X)$ is computed by $g(X,i) = C_d(i)X^d +$



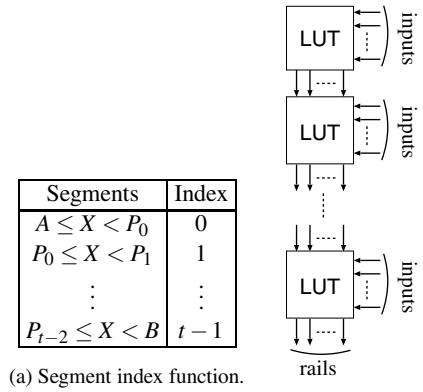
(a) Architecture for NFG.

$$
\begin{array}{rl}
X = & (\ \ldots\ x_{-j}\ \ x_{-j-1}\ \ \ldots\ \ x_{-m_{in}}\ )_2 \\
-\ S_i = & (\ \ldots\ s_{-j}\ \ 0\ \ \ldots\ \ 0\ )_2 \\
\hline
 & (\ \ldots\ 0\ \ x_{-j-1}\ \ \ldots\ \ x_{-m_{in}}\ )_2 \\
 & \Downarrow \\
X = & (\ \ldots\ x_{-j}\ \ x_{-j-1}\ \ \ldots\ \ x_{-m_{in}}\ )_2 \\
\&\ \overline{S_i} = & (\ \ldots\ \overline{s_{-j}}\ \ 1\ \ \ldots\ \ 1\ )_2 \\
\hline
 & (\ \ldots\ 0\ \ x_{-j-1}\ \ \ldots\ \ x_{-m_{in}}\ )_2
\end{array}
$$

Note: $j = m_{in} - h_i$.

(b) Computation of $X - S_i$ using AND gates.

Fig. 4. Architecture for the NFG based on 2nd-order polynomial.



(a) Segment index function.

(b) LUT cascade (MT_SIE).

(c) LUT cascade and adders (EV_SIE).

Fig. 5. Segment index encoders.

$C_{d-1}(i)X^{d-1} + \ldots + C_0(i)$, where i is a segment index assigned to each segment, and the coefficients $C_d(i), C_{d-1}(i), \ldots, C_0(i)$ are derived from the dth-order Chebyshev approximation polynomial [12].

For each segment $[S_i, E_i)$, substituting $X - S_i + S_i$ for X yields the transformation

$$g(X,i) = C_d(i)(X-S_i)^d + C'_{d-1}(i)(X-S_i)^{d-1} + \ldots + C'_0(i), \quad (1)$$

where

$$C'_j(i) = \sum_{k=0}^{d-j} \binom{j+k}{j} C_{j+k}(i) S_i^k \quad (j = 0,1,\ldots,d-1).$$

This transformation reduces the multiplier size (see Section IV-B).

IV. Architecture for NFG

Fig. 4 shows the architecture for the NFG based on a 2nd-order polynomial. As shown in Fig. 4(a), polynomials of the form (1) are realized using a *segment index encoder* (*SIE*), a coefficients table, circuits for $(X - S_i)^k$ $(k = d, d-1, \ldots, 2)$, multipliers, and adders. This architecture can realize any non-uniform segmentation. However, when recursive segmentation is used, we can realize $X - S_i$ using 2-input AND gates instead of an adder. As mentioned in the previous section, the least significant h_i bits of S_i are 0, and $X - S_i < 2^{h_i} \times 2^{-m_{in}}$. Therefore, $X - S_i$ has 1's only in the least significant h_i bits, and these 1's occur in exactly the same position as the 1's in X. Thus, as shown in Fig. 4(b), we realize $X - S_i$ using AND gates driven on one side by $\overline{S_i}$, the complement of S_i. The SIE converts X into a segment index i. It realizes the segment index function $seg\_func(X) : \{0,1\}^n \rightarrow \{0,1,\ldots,t-1\}$ shown in Fig. 5(a), where X has n bits, and t denotes the number of segments.

A. Architecture of SIE

Fig. 5(b) shows an LUT cascade [20] that realizes $seg\_func(X)$. The LUT cascade is obtained by functional decompositions using an MTBDD for $seg\_func(X)$ [18, 19], and can realize any $seg\_func(X)$, where the size depends on the number of segments. [15] has shown that the size of an LUT cascade can be reduced by reducing the number of segments. This section presents a new architecture for the SIE to reduce the size and delay time further. Fig. 5(c) shows the new architecture. To realize $seg\_func(X)$ using the SIE in Fig. 5(c), we represent $seg\_func(X)$ using an EVBDD. And then, by decomposing the EVBDD, we obtain the SIE that consists of an LUT cascade and adders. In an LUT cascade, the interconnecting lines between adjacent LUTs are called *rails*. In this case, the rails represent sub-functions in the EVBDD. And, the outputs from each LUT other than rails represent the sum of weights of edges. In this paper, we call such outputs *Arails (adder rails)*. To the best of our knowledge, this is the first design method using an EVBDD to produce the cascaded architecture.

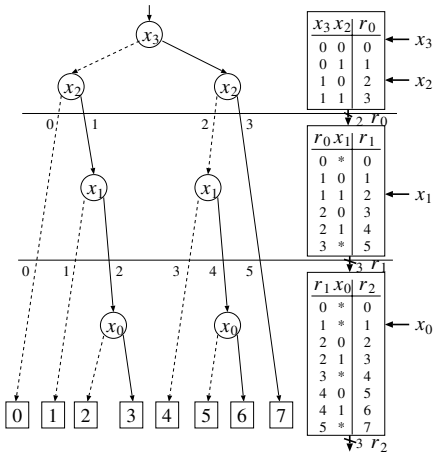
Example 2 *By decomposing the MTBDD and EVBDD in Fig. 2, we obtain the SIEs in Fig. 6. Fig. 6(a) and (b) illustrate the correspondences between each LUT and decompositions of the MTBDD and the EVBDD, respectively. In these figures, the column labeled as 'r_i' in the table of each LUT denotes the rails that represent sub-functions in BDDs. And, the column 'a_i' in Fig. 6(b) denotes the Arails that represent the sum of weights of edges. In the MTBDD, numbers assigned to edges that cut across the horizontal lines represent sub-functions. In the EVBDD, "(a_i, r_i)" assigned to edges that cut across the horizontal lines represent the sum of weights and sub-functions, respectively. The SIE in Fig. 6(a) requires a memory size of* $2^2 \times 2 + 2^3 \times 3 + 2^4 \times 3 = 80$ *bits and* 3 *levels (*3 *LUTs). On the other hand, the SIE in Fig. 6(b) requires a memory size of* $2^2 \times 4 + 2^2 \times 2 + 2^2 \times 1 = 28$ *bits and* 4 *levels (*3 *LUTs* + 1 *adder).* *(End of Example)*

This paper uses two terms: *MT_SIE* and *EV_SIE* denote the SIEs designed using an MTBDD (Fig. 5(b)) and EVBDD (Fig. 5(c)), respectively. Both the MT_SIE and the EV_SIE can realize any non-uniform segmentation. In both cases, memory size depends on the number of segments. Specifically,

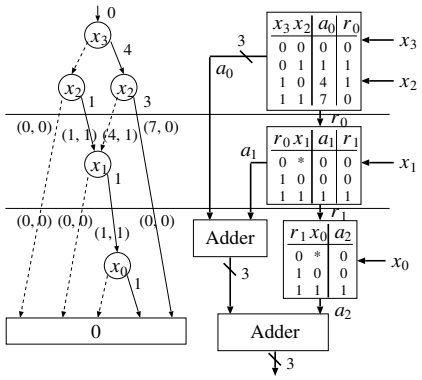
Theorem 1 *Let $seg\_func(X)$ be a segment index function with t segments. Then, there exists an EV_SIE for $seg\_func(X)$ with at most $\lceil \log_2 t \rceil$ rails and $\lceil \log_2 t \rceil$ Arails.*

The proof is omitted because of the page limitation.

The memory size and the number of levels of an EV_SIE depend on the decomposition of an EVBDD. To obtain the optimum decomposition, we use optimization algorithms for heterogeneous multi-valued decision diagrams (MDDs) [14].



(a) SIE using MTBDD (MT_SIE).



(b) SIE using EVBDD (EV_SIE).

Fig. 6. Example of SIEs.

B. Reduction of the Size of the Multiplier

Since large multipliers have large delay, it is important to reduce multiplier size. We do this in two ways; Reduce the number of bits needed to represent 1. the coefficients and 2. the variables $(X - S_i)$.

To reduce the number of bits in the coefficients, we use a *scaling method* [10]. We first shift right the coefficients. Then, we apply rounding. Then, we do the actual multiplication. And, finally, we shift left the product to compensate for the original shift right of the coefficients. This process is similar to floating point multiplication. A side effect is that rounding error is increased, since rounding occurs on a smaller value. In applying this method, we choose the largest exponent (right shift) that produces an error no greater than the given acceptable error [15]. If this yields an exponent of 0 (no right shift), in all segments, then we do not use the scaling method.

To reduce the value of the variable $X - S_i$, we make the following observation. In each segment $[S_i, E_i)$, we have $X - S_i < E_i - S_i$. Thus, reducing the segment width reduces $X - S_i$ for X near E_i. However, this also increases the number of segments, and thus the memory size. We show a segment reduction technique that does not increase memory size.

In an FPGA implementation, the coefficients table in Fig. 4 has 2^u words, where $u = \lceil \log_2 t \rceil$ and t is the number of segments. Therefore, we can increase the number of segments up to $t = 2^u$ without increasing the memory size. From Theorem 1, the size of the EV_SIE also depends on the value of u. Increasing the number of segments to $t = 2^u$ rarely increases the size of the EV_SIE. We reduce the size of segments by dividing the largest segment into two equal sized segments up to $t = 2^u$.

TABLE I
NUMBER OF SEGMENTS FOR VARIOUS SEGMENTATION METHODS.

X has 23-bit accuracy.
Acceptable approximation error: 2^{-25}

Function $f(X)$	Domain $[A,B)$	No. of uniform segs	No. of nonuni. segs	Recursive		
				No. of segs 1	No. of segs 2	Time [msec.]
e^X	$[0,1)$	128	67	103	128*	10
$\sin(\pi X)$	$[0,0.5)$	128	74	112	128*	10
$\tan(\pi X)$	$[0,0.5)$	4,194,304	4,594	5,723	8,192	1,600
$\arcsin(X)$	$[0,1)$	8,388,608	256	363	512	70
$\sqrt{X}$	$(0,1)$	8,388,607	228	322	512	30
$\sqrt{-\ln(X)}$	$(0,1)$	8,388,607	698	967	1,024	190
$X\ln(X)$	$(0,1)$	2,097,152	172	250	256	10

*Uniform segmentation is produced.
Environment: Sub Blade 2500 (Silver), UltraSPARC-IIIi 1.6GHz, 6GB memory, Solaris 9.

V. EXPERIMENTAL RESULTS

A. Number of Segments and Computation Time

Table I compares the number of segments for various segmentation methods based on 2nd-order Chebyshev approximation. In Table I, "No. of uniform segs" shows the number of uniform segments, "No. of nonuni. segs" shows the number of non-uniform segments produced by [15], and "Recursive" denotes the recursive segmentation method shown in this paper. In the column "Recursive", the sub-column "No. of segs 1" shows the number of segments produced by the segmentation algorithm shown in Section III. The sub-column "No. of segs 2" shows the number of segments produced by additionally applying the reduction method of multiplier size shown in Section IV. The sub-column "Time" shows the total CPU time, in milliseconds, for both the segmentation algorithm and the reduction method of multiplier size.

Table I shows that uniform segmentation requires excessively many segments to approximate certain functions, such as $\tan(\pi X)$. Many existing NFGs are based on uniform segmentation, and have not realized $\tan(\pi X)$ in domain $[0,0.5)$. $\tan(\pi X)$ in $[0,0.5)$ can be computed by $\sin(\pi X)/\cos(\pi X)$ or a combination of $\tan(\pi X)$ in $[0,0.25]$ and $1/\tan(\pi X')$, where $X' = 0.5 - X$. However, these require multiple NFGs for elementary functions, such as sin, cos, or the reciprocal function. On the other hand, methods based on non-uniform or recursive segmentation can compactly realize $\tan(\pi X)$ with a single NFG, since non-uniform and recursive segmentation methods require many fewer segments. For all functions in Table I, the non-uniform segmentation method [15] requires the fewest segments among the three segmentation methods. Although our recursive segmentation algorithm restricts the segmentation points, it requires only up to 2.2 times more segments than non-uniform segmentation [15]. That is, our recursive segmentation algorithm generates a segmentation appropriate to the given function, while restricting the segmentation points. For example, for e^X and $\sin(\pi X)$, our algorithm generates uniform segmentation. As shown in [15, 21], uniform segmentation is appropriate for these functions.

These results show that our recursive segmentation algorithm generates a non-uniform segmentation appropriate to the given functions quickly.

TABLE II
FPGA IMPLEMENTATION OF SIEs.

FPGA device:	Altera Stratix EP1S10F484C5 (LE: 10,570, M4K: 60, M512: 90)
Logic synthesis tool:	Altera QuartusII 5.0 (speed optimization, timing requirement of 200MHz)

Function $f(X)$	Optimum non-uniform segmentation								Recursive segmentation							
	MT_SIE				EV_SIE				MT_SIE				EV_SIE			
	Memory [bits]	LE	Level	Delay [nsec.]	Memory [bits]	LE	Level	Delay [nsec.]	Memory [bits]	LE	Level	Delay [nsec.]	Memory [bits]	LE	Level	Delay [nsec.]
e^X	26,368	73	8	27.5	23,040	119	7	24.1	0	0	0	0	0	0	0	0
$\sin(\pi X)$	26,880	71	8	27.5	23,552	114	7	24.1	0	0	0	0	0	0	0	0
$\tan(\pi X)$	1,802,240	–	5	–	179,968	207	10	36.3	1,687,552	–	5	–	15,108	142	7	24.1
$\arcsin(X)$	61,440	72	8	27.5	53,824	217	13	44.7	49,152	72	7	24.3	9,984	88	5	17.2
$\sqrt{X}$	61,440	72	8	27.5	57,408	183	11	40.3	44,544	71	7	24.1	9,216	91	5	17.2
$\sqrt{-\ln(X)}$	266,240	81	7	33.2	116,160	204	11	40.2	172,032	71	7	28.1	12,736	109	6	20.6
$X\ln(X)$	61,440	79	8	27.5	48,384	160	9	30.9	20,992	49	5	17.2	6,912	65	4	13.8

–: It cannot be mapped into the FPGA due to insufficient RAM blocks.

B. FPGA Implementation of SIEs

Table II shows that the recursive segmentation algorithm also automatically generates uniform segmentation when appropriate. Table II compares the FPGA implementation results of the MT_SIE and EV_SIE. Note that the memory size in bits and LE, the number of logic elements, are 0 for e^X and $\sin(\pi X)$ when recursive segmentation is used. This indicates that uniform segmentation was applied, and so an SIE was not needed. The result is a faster NFG for these functions. In the experiment that produced the data in Table II, we optimized the decomposition of the MTBDDs and EVBDDs by requiring the memory size of each LUT in the LUT cascade in these SIEs to be 4K bits, the same as the RAM block (M4K) of the FPGA.

Table II shows that for optimum non-uniform segmentation, the EV_SIEs have smaller memory size than the MT_SIEs. For example, for $\tan(\pi X)$, the memory size of the EV_SIE is only 10% of the memory size needed by the MT_SIE. For $\tan(\pi X)$, the memory size of MT_SIE is quite large because the number of non-uniform segments is large. From experiments with uniform segmentation we know that the NFG for $\tan(\pi X)$ using the MT_SIE requires only 1.5% of the memory size needed by the NFG based on uniform segmentation. However, this is still too large to implement with an FPGA. By using the EV_SIE, we can reduce the memory size significantly, and make the NFG implementable with an FPGA.

Our recursive segmentation can reduce both the memory size and the delay time of the MT_SIEs. Especially, for $X\ln(X)$, using an MT_SIE designed for recursive segmentation has only 34% of the memory and 63% of the delay of the MT_SIE designed for optimum non-uniform segmentation.

By using recursive segmentation and the EV_SIE, we can reduce both memory size and delay time of SIEs significantly. For all functions in Table II, both memory size and delay time of the EV_SIEs for recursive segmentation are much smaller for the MT_SIEs. In terms of the number of LEs, the EV_SIEs require only up to 1.5 times more LEs than the MT_SIEs. Therefore, designing an EV_SIE for recursive segmentation yields faster and more compact SIEs than obtained by previous methods. The design is formal and is easily programmed.

C. FPGA Implementation of NFGs

Table III compares the FPGA implementation results of our NFGs using EV_SIE (EVNFGs) with the existing NFGs using MT_SIE (MTNFGs) [15], where EVNFGs are based on recursive segmentation and MTNFGs are based on the optimum non-uniform segmentation. Both NFGs have 23-bit precision (23-bit accuracy).

From Table II and Table III, we can see that the memory size of MT_SIE accounts for more than $2/3$ of the total memory size of the MTNFG. On the other hand, by using recursive segmentation and EV_SIE, the memory size needed for the SIE can be reduced to less than $1/4$ of the total memory size of the EVNFG. Thereby, the EVNFGs require only 21% to 63% of memory size needed for the MTNFGs. For $\arcsin(X)$ and $\sqrt{X}$, as shown in Table I, our recursive segmentation requires a coefficients table that is about twice as large as needed for the optimum non-uniform segmentation. Nevertheless, by using EV_SIEs, the memory sizes of EVNFGs can be reduced to about 63% of the memory sizes of MTNFGs. Further, Table III shows that the EVNFGs require fewer LEs and levels (i.e., shorter latency) than the MTNFGs, and the delay time of EVNFGs is only about 25% to 89% of the delay time of the MTNFGs.

To compare our NFG with another existing NFG based on a segmentation approach (hierarchical segmentation) shown in [11], we implemented our 24-bit precision NFG for $X\ln(X)$ using the Xilinx Virtex-II FPGA (XC2V4000-6) and the Synplify Premier 8.5. Memory size and delay time of the NFG described in [11] are $40,446$ bits and 103.7 nsec.. On the other hand, memory size and delay time of our EVNFG based on recursive segmentation are $30,976$ bits (77%) and 63.3 nsec. (61%).

From these results, we can see that our NFGs using recursive segmentation and the EV_SIE can realize a wide range of functions faster and more compactly than existing NFGs.

VI. CONCLUSION AND COMMENTS

We have presented an architecture and a synthesis method for fast and compact NFGs for trigonometric, logarithmic, square root, reciprocal, and combinations of these functions. Our NFG partitions a given domain of the function into non-uniform segments using recursive segmentation, and approximates the given function by a polynomial function for each segment. By using an EVBDD to realize the recursive segmentation, we can implement fast and compact NFGs for a wide range of functions. Experimental results showed that: 1) By using recursive segmentation, we reduced memory size and delay time needed for the MT_SIE, and produced MT_SIEs that have, on average, only 49% of the memory and 53% of the delay of MT_SIEs for the optimum non-uniform segmentation. 2) By using EVBDD to realize recursive segmentation, we further reduced memory size and delay time needed for the SIE. Our SIEs using the EVBDDs require, on average, only 7% of the memory and 40% of the delay of MT_SIEs for the opti-

TABLE III
FPGA IMPLEMENTATION OF 23-BIT PRECISION (23-BIT ACCURACY) NFGS.

FPGA device:	Altera Stratix EP1S60F1020C5 (LE: 57,120, DSP: 144, M4K: 292, M512: 574)
Logic synthesis tool:	Altera QuartusII 5.0 (speed optimization, timing requirement of 200MHz)

Function $f(X)$	MTNFG based on optimum nonuni.					EVNFG based on recursive				
	Memory [bits]	LE	DSP	Level	Delay [nsec.]	Memory [bits]	LE	DSP	Level	Delay [nsec.]
e^X	39,040	689	10	13	99.6	8,064	432	10	3	25.1
$\sin(\pi X)$	36,864	635	10	13	99.1	7,936	395	10	3	28.3
$\tan(\pi X)$	2,867,200	–	16	11	–	973,572	1,059	16	12	92.3
$\arcsin(X)$	84,736	1,301	16	14	107.3	53,504	937	16	10	80.3
$\sqrt{X}$	83,712	1,041	16	14	116.5	52,224	962	16	10	85.5
$\sqrt{-\ln(X)}$	357,376	950	16	13	99.8	103,872	972	16	11	88.3
$X\ln(X)$	83,200	988	16	14	116.0	29,696	997	16	9	70.3

–: It cannot be mapped into the FPGA due to insufficient RAM blocks.

mum non-uniform segmentation. And therefore, 3) our NFGs require, on average, only 38% of the memory and 59% of the delay needed by the existing NFGs based on MT_SIE and the optimum non-uniform segmentation.

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 Kitakyushu innovative cluster project, a contract with the National Security Agency, the MEXT Grant-in-Aid for Young Scientists (B), 18700048, 2006, and Hiroshima City University Grant for Special Academic Research (General Studies), 6101, 2006.

REFERENCES

[1] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677–691, Aug. 1986.

[2] A. Cantoni, "Optimal curve fitting with piecewise linear functions," *IEEE Trans. on Comp.*, Vol. 20, No. 1, pp. 59–67, Jan. 1971.

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

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

[5] D. Defour, F. de Dinechin, and J.-M. Muller, "A new scheme for table-based evaluation of functions," *36th Asilomar Conference on Signals, Systems, and Computers,*, Pacific Grove, California, pp. 1608–1613, Nov. 2002.

[6] 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.

[7] C. L. Frenzen, T. Sasao, and J. T. Butler, "The tradeoff between memory size and approximation error in numerical function generators based on lookup tables," preprint.

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

[9] Y-T. Lai, M. Pedram, and S. B. Vrudhula, "EVBDD-based algorithms for linear integer programming, spectral transformation and functional decomposition," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Vol. 13, No. 8, pp. 959–975, Aug. 1994.

[10] D.-U. Lee, W. Luk, J. Villasenor, and P. Y. K. Cheung, "Non-uniform segmentation for hardware function evaluation," *Proc. Inter. Conf. on Field Programmable Logic and Applications*, pp. 796–807, Lisbon, Portugal, Sept. 2003.

[11] 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.

[12] J. H. Mathews, *Numerical Methods for Computer Science, Engineering and Mathematics*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

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

[14] S. Nagayama and T. Sasao, "On the optimization of heterogeneous MDDs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Vol. 24, No. 11, pp. 1645–1659, Nov. 2005.

[15] S. Nagayama, T. Sasao, and J. T. Butler, "Programmable numerical function generators based on quadratic approximation: architecture and synthesis method," *Proc. of Asia and South Pacific Design Automation Conference (ASPDAC'06)*, Yokohama, Japan, pp. 378–383, 2006.

[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, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *Inter. Workshop on Logic Synthesis (IWLS'01)*, Lake Tahoe, CA, pp. 225–230, June 12–15, 2001.

[19] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," *41st Design Automation Conference*, San Diego, CA, pp. 428–433, June 2–6, 2004.

[20] T. Sasao, J. T. Butler, and M. D. Riedel, "Application of LUT cascades to numerical function generators," *Proc. the 12th workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI'04)*, Kanazawa, Japan, pp. 422–429, Oct. 2004.

[21] T. Sasao, S. Nagayama, and J. T. Butler, "Programmable numerical function generators: architectures and synthesis method," *Proc. Inter. Conf. on Field Programmable Logic and Applications (FPL'05)*, Tampere, Finland, pp. 118–123, Aug. 2005.

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

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