Numerical function generators using edge-valued binary decision diagrams

Nagayama, Shinobu

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.

II. PRELIMINARIES

A. Number Representation and Precision

Definition 1 A value X represented by the binary fixed-point representation is denoted by $X = (x_{n-1} x_{n-2} \ldots x_0 x_{-1} x_{-2} \ldots x_{-m})_2$, 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) 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.
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)

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 \( \sin(nX) \), 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 \( \varepsilon_a \). Then, this algorithm produces \( t \) segments \([A,P_0],[P_0,P_1],...,|P_{t-2},B]\) by recursively partitioning a segment into two equal-sized segments until achieving the acceptable approximation error \( \varepsilon_a \) in all segments. Note that this algorithm restricts the width \( w_i \) of each segment to \( w_i = 2^{b_i} \times 2^{-m_{in}} \), 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., \( P_i = (... p_{j+1} p_{j-1} 00 ... 0)_2 \), where \( j = m_{in} - h_i \)). As shown in Fig. 3, the number of segments depends on the maximum approximation error \( \varepsilon_{a}(A,B) \).

In this paper, we use the Chebyshev approximation polynomials. For a segment \([S,E]\) of \( f(X) \), the maximum approximation error of the \( d \)-th order Chebyshev approximation \( \varepsilon_{d}(S,E) \) is given by [12]:

\[
\varepsilon_{d}(S,E) = \frac{2(E-S)^{d+1}}{4^{d+1}(d+1)!} \max_{s \leq s \leq E} |f^{(d+1)}(s)|,
\]

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 \( g(X,i) = C_d(i)X^d + \)
The memory size and the number of levels of an EVBDD can be further reduced by decomposing the MTBDD and the EVBDD, respectively. In these figures, the column labeled as \( \text{r} \) in the table of each LUT denotes the rails that represent sub-functions in BDDs. And, the column ‘a’ 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, \( \text{“(a, r)”} \) 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 \( 2^2 \times 2^4 \times 2^4 = 1024 \) bits and 3 levels (3 LUTs). On the other hand, the SIE in Fig. 6(b) requires a memory size of \( 2^2 \times 2^4 \times 2^4 = 256 \) bits and 4 levels (3 LUTs + 1 adder).

(End of Example)

This paper uses two terms: \( \text{MT-SIE} \) and \( \text{EV-SIE} \) 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 \( \text{seg\_func}(X) \) be a segment index function with \( t \) segments. Then, there exists an \( \text{EV-SIE} \) for \( \text{seg\_func}(X) \) 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 \( \text{EV-SIE} \) depend on the decomposition of an EVBDD. To obtain the optimum decomposition, we use optimization algorithms for heterogeneous multi-valued decision diagrams (MDDs) [14].
of the EV_SIE. We reduce the size of segments by dividing the largest segment into two equal sized segments up to \( t = 2^5 \).

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 \( \tan(X) \). Many existing NFGs are based on uniform segmentation, and have not realized \( \tan(X) \) in domain \([0, 0.5)\), \( \tan(X) \) 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 \( \tan(X) \) 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 \( \sin(X) \), 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.
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(nX) 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(nX), the memory size of the EV, SIE is only 10% of the memory size needed by the MT, SIE. For tan(nX), 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(nX) 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>
<thead>
<tr>
<th>Function</th>
<th>Optimum non-uniform segmentation</th>
<th>Recursive segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MT, SIE</td>
<td>EV, SIE</td>
</tr>
<tr>
<td>e^x</td>
<td>26,368</td>
<td>73</td>
</tr>
<tr>
<td>sin(nX)</td>
<td>26,880</td>
<td>71</td>
</tr>
<tr>
<td>tan(nX)</td>
<td>1,802,240</td>
<td>-</td>
</tr>
<tr>
<td>arcsin(X)</td>
<td>61,440</td>
<td>72</td>
</tr>
<tr>
<td>( \sqrt{X} )</td>
<td>61,440</td>
<td>72</td>
</tr>
<tr>
<td>X ln(X)</td>
<td>266,240</td>
<td>81</td>
</tr>
<tr>
<td>( \sqrt{X} )</td>
<td>61,440</td>
<td>79</td>
</tr>
</tbody>
</table>

Note: It cannot be mapped into the FPGA due to insufficient RAM blocks.
maximum 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), 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, and Hiroshima City University Grant for Special Academic Research (General Studies), 6101, 2006.

REFERENCES


### TABLE III

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

<table>
<thead>
<tr>
<th>Function</th>
<th>MTNFG based on optimum nonuni.</th>
<th>EVNFG based on recursive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory [bits]</td>
<td>LE</td>
</tr>
<tr>
<td>$e^x$</td>
<td>39,040 689</td>
<td>10</td>
</tr>
<tr>
<td>$\sin(x)$</td>
<td>36,864</td>
<td>635</td>
</tr>
<tr>
<td>$\tan(x)$</td>
<td>2,867,200</td>
<td>–</td>
</tr>
<tr>
<td>$\sqrt{x}$</td>
<td>84,736</td>
<td>1,301</td>
</tr>
<tr>
<td>$\sqrt{-\ln(x)}$</td>
<td>83,712</td>
<td>1,041</td>
</tr>
<tr>
<td>$X/\ln(x)$</td>
<td>357,376</td>
<td>950</td>
</tr>
<tr>
<td>$\ln(x)$</td>
<td>83,200</td>
<td>988</td>
</tr>
</tbody>
</table>

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