Hierarchical Segmentation for Hardware Function Evaluation

Dong-U Lee, *Member, IEEE*, Ray C. C. Cheung, *Member, IEEE*, Wayne Luk, *Senior Member, IEEE*, and John D. Villasenor, *Senior Member, IEEE*

***Abstract*—This paper presents a method for evaluating functions based on piecewise polynomial approximations (splines) with a hierarchical segmentation scheme targeting hardware implementation. The methodology provides significant reduction in table size compared to traditional uniform segmentation approaches. The use of hierarchies involving uniform splines and splines with size varying by powers of two is particularly well suited for the coverage of nonlinear regions. The segmentation step is automated and supports user-supplied precision requirements and approximation method. Bit-widths of the coefficients and arithmetic operators are optimized to minimize circuit area and enable a guarantee of 1 unit in the last place (ulp) accuracy at the output. A coefficient transformation technique is also described, which significantly reduces the dynamic ranges of the fixed-point polynomial coefficients. The hierarchical segmentation method is illustrated using a set of functions including $-(x/2)\log_2 x$, $\cos^{-1}(x)$, $\sqrt{-\ln(x)}$, a high-degree rational function, $\ln(1+x)$, and $1/(1+x)$. Various degree-1 and degree-2 approximation results for precisions between 8 to 24 bits are given. Hardware realizations are demonstrated on a Xilinx Virtex-4 field-programmable gate array (FPGA).**

***Index Terms*—Circuit synthesis, design automation, digital systems, field-programmable gate arrays (FPGAs), piecewise polynomial approximation.**

I. Introduction

THE evaluation of mathematical functions is often central to numerous applications in communications, computer graphics, digital signal processing, and scientific computing. Examples of such functions include elementary functions such as $\ln(x)$ and $\cos^{-1}(x)$, and compound functions such as $-(x/2)\log_2 x$ and $\sqrt{-\ln(x)}$. Software environments such as C or MATLAB usually provide libraries for evaluating functions in floating-point precision. However, software implementations on instruction processors are often too slow for numerically intensive and/or real-time applications. The performance of such applications depends on the design of a fast and accurate hardware function evaluation unit, usually implemented on a field-programmable gate array (FPGA) or an application-specific integrated circuit (ASIC).

The evaluation of functions has received considerable interest in the research community. In particular, evaluation methods involving polynomials and splines have been extensively used in both hardware and software implementations. Spline approximations (piecewise polynomials) are often preferred over polynomial-only approximations due to the wide range of design tradeoffs they offer involving memory, computational complexity, and precision [1]. Traditionally, spline approximations use uniform segmentation, in which all splines cover segments of equal width and the total number of segments is constrained to a power of two. This has the advantage of simple coefficient address computation, but can be problematic for regions of the function in which the first or higher order derivatives have high absolute values.

The hierarchical segmentation method discussed in this paper employs hierarchies involving uniform splines and splines with size varying by powers of two. This segmentation technique is able to adapt to nonlinearities of a function, resulting in significant reduction in the number of splines compared to uniform segmentation. Each spline contains a set of polynomial coefficients corresponding to a particular region of a function. The polynomial is then evaluated in fixed-point arithmetic. One of the main challenges of such arithmetic structures lies in the determination of the required bit-widths of the coefficients and operators. For the work presented here, we use the MiniBit bit-width optimization approach [2]. MiniBit enables computation of the required integer and fractional bits for each signal in an analytical manner, making it possible to guarantee faithful rounding [1 unit in the last place (ulp) accuracy] at the output.

In the work presented here, compound functions are evaluated directly. This contrasts with the traditional approach involving computation via a series of elementary function evaluations, and the associated delays and rounding error accumulation that occur during range reduction and reconstruction for each component elementary function [3]. Thus, the advantages of direct evaluation as discussed here increase significantly as compound functions become more complex.

In contrast with much of the previous work, in this paper we are targeting environments in which the delay introduced by the coefficient address logic must be kept to a minimum. Additionally, we enable a designer to specify an error tolerance ϵ_{req} and to automatically obtain a segmentation that: 1) meets this tolerance; 2) requires a small number of segments M; and 3)

Manuscript received December 20, 2006; revised May 30, 2007 and November 25, 2007. First published November 25, 2008; current version published December 17, 2008. This work was supported in part by the Office of Naval Research under Contract N00014-06-1-0253, by the National Science Foundation under Grant CCR-0120778 and Grant CCF-0541453, by the Croucher Foundation, Xilinx Inc., and by the U.K. Engineering and Physical Sciences Research Council under Grant EP/C509625/1, Grant EP/C549481/1, and Grant GR/R 31409.

D. Lee is with Mojix, Inc., Los Angeles, CA 90025 USA (e-mail: dongu@mojix.com).

R. C. C. Cheung is with Solomon Systech Limited, Hong Kong (e-mail: cccheung@ieee.org).

W. Luk is with the Department of Computing, Imperial College London, London SW7 2BZ, U.K. (e-mail: w.luk@imperial.ac.uk).

J. D. Villasenor is with the Electrical Engineering Department, University of California, Los Angeles, CA 90095 USA (e-mail: villa@icsl.ucla.edu).

Digital Object Identifier 10.1109/TVLSI.2008.2003165

1063-8210/$25.00 © 2008 IEEE

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

leads to efficient hardware implementation. The proposed segmentation approach can be applied to a wide range of approximation methods. Without loss of generality, it will be illustrated with spline approximations involving polynomials. Some earlier aspects of this work were presented in [4]. This paper includes a number of additions and improvements, including automated analysis of function characteristics and use of the results in choosing a segmentation method, bit-width optimization to guarantee 1 ulp precision, coefficient transformation to reduce dynamic range, and hardware implementation results that include measurements of combinatorial delay.

The rest of this paper is organized as follows. Section II discusses background of function evaluation and discusses previous work on uniform and nonuniform segmentation. Section III presents an overview of the methodology for function evaluation unit design with hierarchical segmentation. Section IV describes the hierarchical segmentation method. Section V presents the hardware architecture for evaluating functions based on hierarchical segmentation. Section VI presents hardware realization results for a Xilinx Virtex-4 FPGA device. Concluding remarks are given in Section VII.

II. Background

Function evaluation methods can be classified into iterative methods and non-iterative methods. Iterative methods [5], [6] successively refine the output precision and are suitable for applications where arbitrary precisions are desired. However, they usually involve high latencies and low throughputs, making them unsuitable for high-performance applications.

Non-iterative methods include direct table lookups, table addition methods, polynomial approximations, and rational approximations. Direct table lookups are widely used for computations involving low-precision inputs, but are impractical for high-precision inputs due to high table size. Table addition methods [7], [8] use two or more parallel table lookups followed by multi-operand addition. Although this approach gives significant improvements in table size and potential speed improvements due to reduction in table access times over the direct table lookup approach, it still suffers from large table sizes for high precisions. Polynomial approximations [1] involve the evaluation of a polynomial over a given interval. The approximation accuracy can be controlled by varying the degree of the polynomial and choice of interval. Rational approximation [9] is a generalization of polynomial approximation in which the function is approximated using the ratio of two polynomials. For a given limit on numerator and denominator polynomial degree, it enables higher accuracy than polynomial approximation, but due to the divide operation, the circuit complexity is considerably higher. Non-iterative methods are often combined with segmentation in which the input range is split into multiple segments, each associated with a spline containing a particular set of coefficients.

By far the most common segmentation method is uniform, in which all segment lengths are equal [1], [7], [8], [10]–[12]. In addition, the choice of segment numbers is typically limited to powers of two. While this leads to simple coefficient address computation, in contrast with nonuniform segmentation, it does not allow segment lengths to be customized to the local function characteristics. The benefits of such customization using nonuniform segmentation have been noted in the past, particularly in association with logarithmic number systems (LNS). LNS involves the approximation of highly nonlinear functions for performing addition and subtraction operations in the logarithmic domain. Two-level uniform segmentation has been described by Lewis [13]. The segmentation in both levels is uniform, but the density of the second level segmentation is customized in accordance with local function characteristics. Coleman *et al.* [14] describe a two-level approach in which the first segmentation has segments that vary by powers of two and the segmentation at the second level is uniform. References [13] and [14] consider a subset of the segmentation options considered here, and neither discusses automating the segmentation process.

Balanced error segmentation is a form of nonuniform segmentation in which the maximum approximation error in all segments is equal. This is desirable since it minimizes the number of segments M needed to meet a given overall approximation error constraint ϵ_{req}, though as discussed further below it is not particularly conducive to efficient hardware mapping. Given a continuous function f, interval $[a, b)$, and number of segments M, the goal in balanced error segmentation is to identify segment boundaries u_m, where $a = u_0 \leq u_1 \leq \cdots \leq u_{M-1} \leq b$ and associated degree- d polynomials $p_m(x)$ such that the absolute maximum approximation errors

$$\epsilon_{\max_m} = \max_{u_m \leq x \leq u_{m+1}} |f(x) - p_m(x)| \tag{1}$$

are minimized and are equal for all segments [15].

Several algorithms have been described in the literature to address the balanced error segmentation problem [4], [15]–[17]. However, the main challenge with implementing balanced error approaches is that the arbitrary variation in segment lengths complicates the circuitry needed to identify the segment associated with a particular input. This challenge has been recognized by Sasao *et al.* [18]–[20], who present an algorithm for finding the balanced error segmentation and employ a large cascade of lookup tables (LUTs) for computing the coefficient address. This approach leads to the minimum number of segments, but may not always result in designs with minimized delay and/or memory requirements; we shall illustrate this point in Section VI.

III. Design Flow Overview

Fig. 1 gives an overview of the function evaluation unit design methodology based on hierarchical segmentation. The following input parameters must be supplied by the user:

1) function to be evaluated (e.g., $-(x/2)\log_2 x, \cos^{-1}(x)$, etc.);
2) input interval (e.g., $[0, 1), [1, 4)$, etc.);
3) input and output precisions (e.g., 16 bits, etc.);
4) approximation method (e.g., degree-1 splines, degree-2 splines, etc.).

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

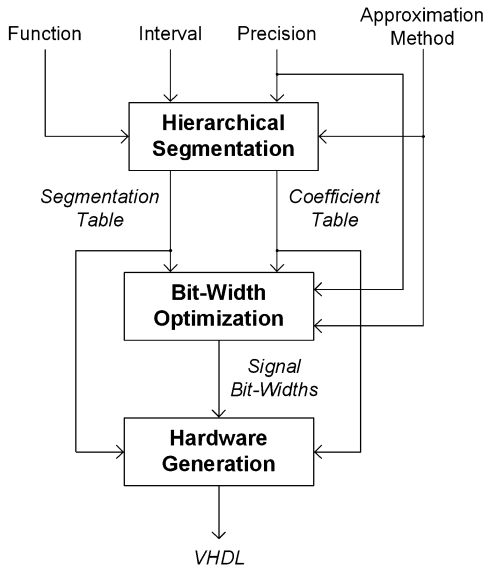


Fig. 1. Overview of the automated methodology for function evaluation unit design.

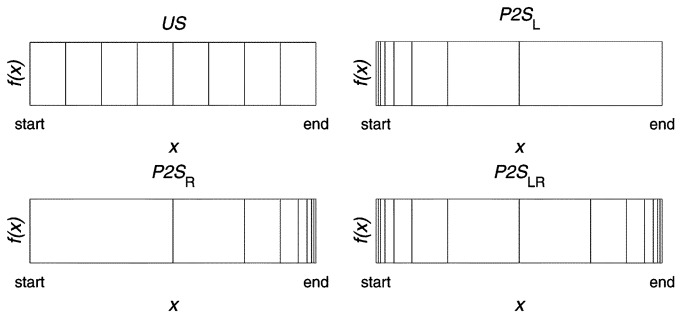


Fig. 2. Illustration of uniform segmentation US and three segmentations involving lengths that increase or decrease by powers of two: $\mathrm{P2S}_L$, $\mathrm{P2S}_R$, and $\mathrm{P2S}_{\mathrm{LR}}$.

The "Hierarchical Segmentation" step partitions the splines over the interval of the function in an adaptive manner via a hierarchy of segmentation schemes, and produces segment boundaries $u_{0\ldots M-1}$ and the corresponding polynomial coefficients. The details of the segmentation are described more fully in the following. Chebyshev coefficients [3] are used for the splines. Potentially, a minimax approximation could be performed on the Chebyshev coefficients to find a set of coefficients with slightly lower $\epsilon_{\max}$. Chebyshev is used in this paper instead of minimax due to its faster coefficient generation time, which was necessary due to the large collection of results presented in Section VI.

The "Bit-Width Optimization" step applies bit-width optimization to the circuit and determines the fixed-point bit-width required for each signal. The bit-widths determined should guarantee overflow protection and faithful rounding at the output signal. The final step, "Hardware Generation," utilizes the segmentation, coefficient, and bit-width information, and produces synthesizable VHDL code suitable for FPGA or ASIC implementation. The entire process is fully automated and has been implemented using MATLAB.

IV. Hierarchical Segmentation Method

A. Framework

The segmentation utilizes a selection from a library of four schemes when subdividing an interval, denoted $\mathrm{US}, \mathrm{P2S}_L, \mathrm{P2S}_R$, and $\mathrm{P2S}_{\mathrm{LR}}$, respectively, as illustrated in Fig. 2. In US, segments are uniformly sized. In $\mathrm{P2S}_L$, the segment sizes increase by powers of two from the beginning of the input interval to the end of the interval, while in $\mathrm{P2S}_R$ the segment sizes decrease by powers of two from start to end. In $\mathrm{P2S}_{\mathrm{LR}}$, segment sizes increase by powers of two until the midpoint of the interval and then decrease by powers of two until the end is reached. As can be seen in Fig. 2, this range of options offers a way to match the segmentation to portions of a function that have singularities or narrow peaks. The hierarchical aspect arises because segmentation is applied recursively. In the first pass, the entire interval $[a, b)$ is subdivided using one of the above four methods into smaller segments. In the next pass, each segment can be further subdivided, again using any of the four methods. The decision on which method to use is made independently on each segment. This process can continue to the depth necessary to meet the design requirements. More formally, the segmentation at each level, and of each segment, is selected from $\Lambda \in \{\mathrm{US}, \mathrm{P2S}_L, \mathrm{P2S}_R, \mathrm{P2S}_{\mathrm{LR}}\}$. In principle, any of the segments produced in the segmentation at level i can be further segmented using any of the four methods in level $i+1$. In practice, however, it is often sufficient to use the same segmentation Λ on all segments within the same segmentation level. This also simplifies the notation, allowing the full hierarchy to be expressed through $\Lambda_0(\Lambda_1(\ldots(\Lambda_L)))$, where $L+1$ is the number of levels in the hierarchy. When dealing with functions with multiple singularities distributed throughout the input interval, it may be beneficial to support arbitrary segmentation for each segment and each level. The methodologies presented here still apply in such cases, though the notation to describe the segmentation becomes more cumbersome. Alternatively, the input interval can be divided into multiple subintervals separated at the singularity regions and segmentation can be performed individually on these subintervals. In such cases, the singularities can potentially occur in arbitrary regions of the function, meaning that although the hardware-efficient segmentation described here can be applied within each subinterval, it cannot be applied to the outer most segmentation. Therefore, comparators or multi-valued decision diagrams [19] will need to be utilized for the outer most segmentation, which can add increased area and delay requirements.

This structure can be implemented via cascaded LUTs, where the output of one LUT is supplied as the address of the next (as illustrated in Fig. 8 which will be discussed further in Section V). Cascaded lookup tables are also used in the balanced error approach of Sasao *et al.* [18]–[20]. However, while Sasao *et al.* use them to realize multi-valued decision diagrams [21], in hierarchical segmentation, each LUT is used to store information of a given hierarchy level. Thus, the B_x-bit representation of an input x is split into $L+1$ partitions denoted $x_{0,\ldots,L}$, where B_{x_i} denotes the bit-width of the ith partition x_i.

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

TABLE I
$\mathrm{P2S_{LR}}$ SEGMENT RANGES IN BINARY REPRESENTATION FOR $B_x = 8, \Lambda_0 = \mathrm{P2S_{LR}}$, AND $B_{x_0} = 5$. THE FIVE BITS CORRESPONDING TO x_0 ARE HIGHLIGHTED IN BOLD. THE BITS TO THE LEFT OF THE VERTICAL PARTITION LINES CORRESPOND TO $\hat{x}_0$

Λ_0 Segment Number, j	Segment Range
0	**0 0 0 0 0** | 0 0 0 ~ **0 0 0 0 0** | 1 1 1
1	**0 0 0 0 1** | 0 0 0 ~ **0 0 0 0 1** | 1 1 1
2	**0 0 0 1** | **0** 0 0 0 ~ **0 0 0 1** | **1** 1 1 1
3	**0 0 1** | **0 0** 0 0 0 ~ **0 0 1** | **1 1** 1 1 1
4	**0 1** | **0 0 0** 0 0 0 ~ **0 1** | **1 1 1** 1 1 1
5	**1 0** | **0 0 0** 0 0 0 ~ **1 0** | **1 1 1** 1 1 1
6	**1 1 0** | **0 0** 0 0 0 ~ **1 1 0** | **1 1** 1 1 1
7	**1 1 1 0** | **0** 0 0 0 ~ **1 1 1 0** | **1** 1 1 1
8	**1 1 1 1 0** | 0 0 0 ~ **1 1 1 1 0** | 1 1 1
$9 = s_0 - 1$	**1 1 1 1 1** | 0 0 0 ~ **1 1 1 1 1** | 1 1 1

The number of addressable segments in the ith partition s_i is constrained as follows:

$$s_i = 2^{B_{x_i}}, \quad \text{if } \Lambda_i = \mathrm{US} \tag{2}$$

$$s_i \leq B_{x_i} + 1, \text{ if } \Lambda_i = \{\mathrm{P2S}_L, \mathrm{P2S}_R\} \tag{3}$$

$$s_i \leq 2B_{x_i}, \quad \text{if } \Lambda_i = \mathrm{P2S_{LR}}. \tag{4}$$

Segments within any given level of the hierarchy are indexed by j, where $0 \leq j \leq s_i - 1$. For uniform segmentation US, it is clear that $2^{B_{x_i}}$ segments can be formed, since $2^{B_{x_i}}$ uniform segments are addressed with B_{x_i} bits. However, for the three power of two schemes, P2S, the s_i constraints are not as intuitive. Consider the example of a two-level hierarchy in which $B_x = 8$, and the first partition is $\Lambda_0 = \mathrm{P2S_{LR}}$ and is addressed using 5 bits; e.g., $B_{x_0} = 5$. As noted earlier, it is assumed that the approximation interval is normalized to the range $a = 00000000_2$ to $b = 11111111_2$ (in the case where $B_x = 8$), where the leading binary point is implicit. In this example, it is possible to construct a maximum of 10 segments in the first level of the hierarchy as illustrated in Table I. Note that with the exception of the initial and final segments, the segment lengths increase by powers of two until $\mathbf{01111}111_2$ (end of location 4) and start decreasing by powers of two from the midpoint $\mathbf{10000}000_2$ (beginning of location 5) to the end. Fewer segments can be obtained by omitting parts of the table. For $\mathrm{P2S}_L$, locations 0 to 5 are used with the ending value of location 5 modified to $\mathbf{11111}111_2$. Analogously, for $\mathrm{P2S}_R$, locations 4 to 9 are used with the beginning value of location 4 modified to $\mathbf{00000}000_2$.

Computation of the segment address for a given partition x_i is based on detecting the number of leading zeros for segments beginning with a zero, and on detecting the number of leading ones for segments beginning with a one. Specifically, the P2S addresses (segment numbers) can be computed in the following manner:

$$\mathrm{P2S}_L\text{-addr} = \begin{cases} B_{x_i} - \mathrm{LZD}(x_i), & \text{if } \mathrm{MSB}(x_i) = 0 \\ B_{x_i}, & \text{if } \mathrm{MSB}(x_i) = 1 \end{cases} \tag{5}$$

$$\mathrm{P2S}_R\text{-addr} = \begin{cases} 0, & \text{if } \mathrm{MSB}(x_i) = 0 \\ \mathrm{LOD}(x_i), & \text{if } \mathrm{MSB}(x_i) = 1 \end{cases} \tag{6}$$

$$\mathrm{P2S_{LR}}\text{-addr} = \begin{cases} B_{x_i} - \mathrm{LZD}(x_i), & \text{if } \mathrm{MSB}(x_i) = 0 \\ B_{x_i} + \mathrm{LOD}(x_1) - 1, & \text{if } \mathrm{MSB}(x_i) = 1 \end{cases} \tag{7}$$

where $\mathrm{LZD}(x_i), \mathrm{LOD}(x_i)$, and $\mathrm{MSB}(x_i)$ return the number of leading zeros, number of leading ones, and the most significant bit of x_i, respectively.

B_{x_0} gives the number of bits used for indexing the segments in Λ_0 (5 in this case, shown in bold Table I). Let $\hat{x}_i$ denote the set of bits that remains constant within a given segment in Λ_i (bits left side of the vertical partition lines Table I). The $(i+1)$th level Λ_{i+1} uses the adjacent $B_{x_{i+1}}$ bits to the right of $\hat{x}_i$. For example, in Table I, $B_{\hat{x}_0} = B_{x_0} = 5$ for $j = 0$ and $j = s_i - 1 = 9$, and $B_{\hat{x}_0} = 3$ for $j = 3$ and $j = 6$. When uniform segmentation is used in the ith level of the hierarchy, then $\Lambda_i = \mathrm{US}$, and B_{x_i} bits of x_i remain constant over a given segment, so $B_{\hat{x}_i} = B_{x_i}$. Therefore, the $(i+1)$th level Λ_{i+1} simply uses the $B_{x_{i+1}}$ bits immediately to the right of $\hat{x}_i = x_i$.

However, if one of the three nonuniform segmentations is used, $\Lambda_i = \{\mathrm{P2S}_L, \mathrm{P2S}_R, \mathrm{P2S_{LR}}\}$, then the number of bits corresponding to Λ_{i+1} depends on the value of x_i, since x_i determines the value of $B_{\hat{x}_i}$. Consider again the case when $\Lambda_i = \mathrm{P2S_{LR}}$. Let $x_i[k]$ denote the kth bit from the least significant bit (LSB) of x_i; e.g., $x_i[0]$ is the LSB of $x_i, x_i[1]$ is the bit immediately to the left of the LSB, etc. In formal terms, x_{i+1} which has $B_{x_{i+1}}$ bits, is given by the bits to the right of the following:

- $\hat{x}_i[0] = x_i[0]$ for $j = 0$ and $j = s_i - 1$ (segment numbers 0 and 9 in Table I);
- $\hat{x}_i[0] = x_i[j-1]$ for $j = 1$ to $j = (s_i/2) - 1$ (segment numbers 1 to 4 in Table I);
- $\hat{x}_i[0] = x_i[s_i - 2 - j]$ for $j = s_i/2$ to $j = s_i - 2$ (segment numbers 5 to 8 in Table I).

With appropriate modifications to the asymmetry of the segmentation, an analogous procedure applies for determining x_{i+1} in the case of $\mathrm{P2S}_L$ and $\mathrm{P2S}_R$. The computation of $B_{\hat{x}_i}$ can be performed as follows:

$$\mathrm{US}\ B_{\hat{x}_i} = B_{x_i} \tag{8}$$

$$\mathrm{P2S}_L\ B_{\hat{x}_i} = \begin{cases} B_{x_i}, & \text{if } \mathrm{P2S}_L\text{-addr} = 0 \\ B_{x_i} - \mathrm{P2S}_L\text{-addr} + 1, & \text{otherwise} \end{cases} \tag{9}$$

$$\mathrm{P2S}_R\ B_{\hat{x}_i} = \begin{cases} B_{x_i}, & \text{if } \mathrm{MSB}(x_i) = 0 \\ \mathrm{P2S}_R\text{-addr} + 1, & \text{if } \mathrm{MSB}(x_i) = 1 \\ \mathrm{P2S}_R\text{-addr}, & \text{if } \mathrm{P2S_{LR}}\text{-addr} = s_i - 1 \end{cases} \tag{10}$$

$$\mathrm{P2S_{LR}}\ B_{\hat{x}_i} = \begin{cases} B_{x_i}, & \text{if } \mathrm{P2S_{LR}}\text{-addr} = 0 \text{ or } s_i - 1 \\ B_{x_i} - \mathrm{P2S_{LR}}\text{-addr} + 1, & \text{if } \mathrm{MSB}(x_i) = 0 \text{ and } \mathrm{P2S_{LR}}\text{-addr} \neq 0 \\ \mathrm{P2S_{LR}}\text{-addr} - B_{x_i} + 2, & \text{otherwise.} \end{cases} \tag{11}$$

In principle, it is possible to have any number of levels L of nested segmentation steps Λ, as long as $\sum_{i=0}^{L} B_{x_i} \leq B_x$. The more levels are used, the closer the total number of segments M will be to the optimal. However as L increases the partitioning problem becomes more complex, and the cascade of lookup tables gets deeper, increasing the delay involved in finding desired segment. Experiments show that the rate of reduction of M decreases rapidly as L increases, so that further increase in delay

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

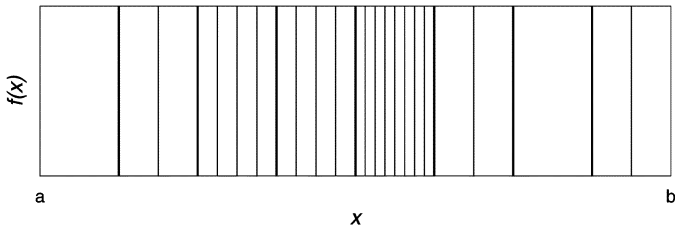


Fig. 3. Illustration of a two-level segmentation that is uniform at both levels, e.g., US(US). In this example $B_{x_0} = 3$, leading to 8 equal-width outer segments, and B_{x_1}, ranges from 0 to 3, so that each outer segment is partitioned into 1, 2, 4, or 8 uniform inner segments. The total number of segments is $M = 24$. The black and grey vertical lines indicate the boundaries for the outer segmentation Λ_0 and inner segmentation Λ_1, respectively. Hardware suitable for such a scheme is illustrated in Fig. 8, which is also discussed in Section V in detail.

arising from incrementing L gives diminishing returns in terms of approximation accuracy.

In most cases, $L = 2$, which consists of an outer segmentation scheme Λ_0 and an inner segmentation scheme Λ_1, gives an M that is close to the optimal while retaining acceptable partitioning complexity and delay. Therefore, in the results presented here $L = 2$ is used. For the first level of segmentation $\Lambda_0, \mathrm{P2S_{LR}}$ is used if the function has strong nonlinearities at both endpoints of the range a and b. $\Lambda_0 = \mathrm{P2S}_L$ is used if the function has strong nonlinearities at a but not b, and $\mathrm{P2S}_R$ is used if the opposite occurs. $\Lambda_0 = \mathrm{US}$ is used if the function is nonlinear in regions away from the endpoints. For the inner segmentation Λ_1, US is used for the results presented here, enabling straightforward and efficient control of the approximation error of an outer segment Λ_0. Fig. 3 illustrates an example of US(US), where $B_{x_0} = 3$, leading to 8 equal-width outer segments, and B_{x_1}, ranges from 0 to 3, so that each outer segment is partitioned into 1, 2, 4, or 8 uniform inner segments. The total number of segments is $M = 24$.

B. Segmentation Algorithm

The segmentation is demonstrated with the following six elementary and compound functions:

$$f_1 = -\frac{x}{2}\log_2 x \tag{12}$$

$$f_2 = \cos^{-1}(x) \tag{13}$$

$$f_3 = \sqrt{-\ln(x)} \tag{14}$$

$$f_4 = \frac{0.0004x + 0.0002}{x^4 - 1.96x^3 + 1.348x^2 - 0.378x + 0.0373} \tag{15}$$

$$f_5 = \ln(1 + x) \tag{16}$$

$$f_6 = 1/(1 + x) \tag{17}$$

where x is an n-bit number over $[0, 1)$ of the form $0.x_{n-1}, \ldots, x_0$. The function f_1 occurs in image warping algorithms [22] and f_3 is used in the Box–Muller algorithm for the generation of Gaussian noise [23]. Functions f_2, f_5, and f_6 are commonly used elementary functions [3], while function f_4 is an example of a complex high-degree rational function.

The determination of the appropriate segmentation hierarchy for a given function plays an important role. Choosing the wrong hierarchy for a given function can result in inefficient segmentation, leading to unnecessarily large numbers of segments. One way of finding the best hierarchy is to apply all possible segmentation hierarchies and pick the one that gives the smallest number of segments M. Instead of using this brute-force approach, we first find the balanced error segmentation of the given function. Then for each of the four possible segmentation schemes Λ, we obtain the variance of a histogram that holds the number of balanced error segmentation boundaries within each of the "partitions" created by the outer segmentation. The scheme that results in the smallest variance is chosen, since a small variance indicates a good match with the balanced error segmentation. This approach is illustrated in the pseudo-code shown in Algorithm 1. It can be applied recursively L times to find the appropriate inner segmentation schemes.

Algorithm 1 Select Best Hierarchy Scheme

// Parameters: Input Interval $(a, b]$,
// Balanced Error Segmentation Boundaries $\vec{D}$

// Use 8 Segments to Aid the Selection of the Best Scheme H

// $\mathrm{P2S}_L$
$\vec{\Lambda}(0,:) = [0; 0.0078125; 0.015625; 0.03125; 0.0625; 0.125; 0.25; 0.5; 1] \times (b-a) + a$
// $\mathrm{P2S}_R$
$\vec{\Lambda}(1,:) = [0; 0.5; 0.75; 0.875; 0.9375; 0.96875; 0.984375; 0.9921875; 1] \times (b-a) + a$
// $\mathrm{P2S_{LR}}$
$\vec{\Lambda}(2,:) = [0; 0.625; 0.125; 0.25; 0.5; 0.75; 0.875; 0.9375; 1] \times (b-a) + a$
// US
$\vec{\Lambda}(3,:) = [0; 0.125; 0.25; 0.375; 0.5; 0.625; 0.75; 0.875; 1] \times (b-a) + a$

for $i = 0$ to 3 **do**
  for $j = 0$ to 7 **do**
    for $k = 0$ to length $(\vec{D})$ -1 **do**
      if $\vec{\Lambda}(i,j) \le \vec{D}(k) < \vec{\Lambda}(i,j+1)$ **then**
        $\vec{\Lambda}_h(i,j)++$
      end if
    end for
  end for
end for

$H = \min(\mathrm{var}(\vec{\Lambda}_h(0,:)), \mathrm{var}(\vec{\Lambda}_h(1,:)), \mathrm{var}(\vec{\Lambda}_h(2,:)), \mathrm{var}(\vec{\Lambda}_h(3,:)))$

Another important issue is the determination of number of bits B_{x_0} to assign for the outer segmentation Λ_0. If B_{x_0} is too small, there will be insufficient granularity in the outer segments follow the local nonlinearities of a function. On the other hand, if B_{x_0} is too large, there will be too many outer segments and the total number of segments M will be unnecessarily large. Fig. 4 gives an example of how M varies with increasing B_{x_0} for the approximation of functions f_1 and f_4 accurate to 2^{-14}. The segmentation $\mathrm{P2S}_L(\mathrm{US})$ is used for f_1, while US(US) is

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

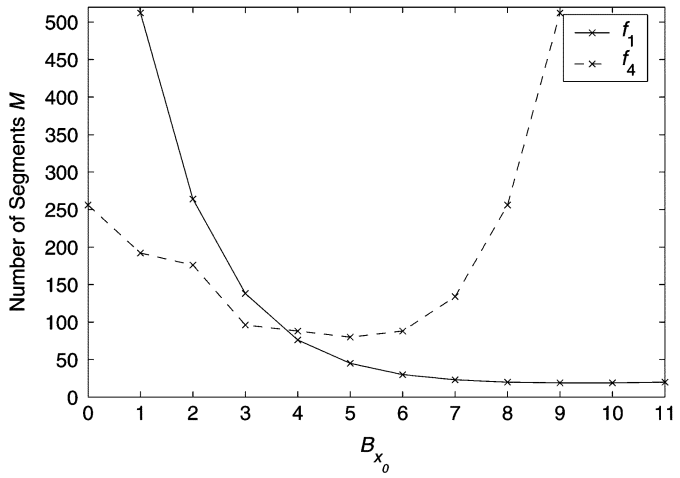


Fig. 4. Variation of the number of segments M against B_{x_0} for functions f_1 (12) and f_4 (15) accurate to 2^{-14}. For function f_1, M reaches a minimum at $B_{x_0} = 9$, while for function f_4, it reaches a minimum at $B_{x_0} = 5$. Segmentation $\mathrm{P2S}_L(\mathrm{US})$ is used for f_1, while $\mathrm{US}(\mathrm{US})$ is used for f_4.

TABLE II
CONTENTS OF ROM0 FOR THE $\mathrm{US}(\mathrm{US})$ EXAMPLE IN FIG. 3

index	0	1	2	3	4	5	6	7
B_{x_1}	0	1	2	2	3	1	0	1
offset	0	1	3	7	11	19	21	22

used for f_4. In both cases there is a single minimum, occurring at $B_{x_0} = 9$ bits for f_1 and $B_{x_0} = 5$ bits for f_2.

Algorithm 2 gives the pseudo-code of the hierarchical segmentation method. As noted earlier, four input parameters are required: the input interval $[a, b)$, the ulp of the input, the polynomial degree d to be used for the splines, and the desired maximum absolute error ϵ_{req} at the output. The notation $A = (A; B)$ concatenates matrix B to matrix A (e.g., line 42). First, the appropriate segmentation hierarchy is found (line 4 to line 6). Next, hierarchical segmentation is performed while searching for the optimal B_{x_0} that minimizes M. Initially $B_{x_0} = 0$, which corresponds to uniform segmentation. B_{x_0} is then incremented until the segmentation that gives the minimum number of segments M is found (line 8 to line 65).

The core of the algorithm lies in the for-loop from line 19 to line 51. For each segment in the outer segmentation, the Chebyshev coefficients for the approximating polynomial of appropriate degree are computed. If the approximation error $\epsilon_{\max}$ is too high, the number of segments in the inner segmentation is incremented by successive powers of two until the $\epsilon_{\max}$ of all inner segments are less or equal to the required error ϵ_{req}. This process is performed for all outer segments. The final output is the total number of inner segments M, the vector $\vec{\Lambda}$ containing the segment boundaries, ROM0 which is needed for ROM1 address computation, and ROM1 which holds the polynomial coefficients for each segment. ROM0 stores the B_{x_1} and the offset corresponding to each first level segment. The offset is simply the number of rows in ROM1 prior to the row in ROM1 corresponding to the current first level segment. Table II shows the contents of ROM0 for the $\mathrm{US}(\mathrm{US})$ example in Fig. 3.

Algorithm 2 Hierarchical Segmentation Method

```
 1: // Parameters: Function f, Input Interval (a, b], ulp
 2: //             Polynomial Degree d, Required Error ε_req
 3:
 4: // Select Segmentation Hierarchy H
 5: D⃗ = GetBalancedErrorBoundaries(f, a, b, d, ulp, ε_req)
 6: H = SelectHierarchy(D⃗)      //Algorithm 1
 7:
 8: // Find the Segmentation with the Optimal B_{x_0}
 9: Λ⃗ = [ ]
10: B_{x_0} = 0
11: M' = ∞
12: while 1 do
13:     s_0 = GetNumberOfΛ_0Segs(H, B_{x_0})
14:     Λ⃗_0 = GetΛ_0Boundaries(H, B_{x_0}, (a, b))
15:     ROM0 = [ ]
16:     ROM1 = [ ]
17:     offset = 0
18:     for i = 0 to s_0 − 1 do
19:         A = Λ⃗_0(i)
20:         B = Λ⃗_0(i + 1)
21:         (ε_max, Coeffs) = Chebyshev(f, d, A, B, ulp)
22:         if ε_max > ε_req
23:           Λ_0SegSize = B − A
24:           B_{x_1} = 0
25:           while ε_max > ε_req
26:               B_{x_1} = B_{x_1} + 1
27:               Λ_1SegSize = Λ_0SegSize/2^{B_{x_1}}
28:               Λ⃗_1 = [ ]
29:               Λ_1Coeffs⃗ = [ ]
30:               s_1 = 2^{B_{x_1}}
31:               for j = 0 to s_1 − 1 do
32:                   A = Λ⃗_0(i) + Λ_1SegSize × j
33:                   B = A + Λ_1SegSize
34:                   (ε_max, Coeffs) = Chebyshev
                                        (f, d, A, B, ulp)
35:                   if ε_max > ε_req then
36:                       break
37:                   end if
38:                   Λ⃗_1(j) = A
39:                   Λ_1Coeffs⃗(j) = Coeffs
40:               end for
41:           end while
42:           ROM0 = [ROM0; B_{x_1} offset]
43:           offset = offset + 2^{B_{x_1}}
44:         else
45:           Λ⃗_1 = A
46:           ROM1 = Coeffs
47:         end if
48:         Λ⃗ = [Λ⃗; Λ⃗_1]
49:         ROM1 = [ROM1; Λ_1Coeffs⃗]
50:     end for
51:     M = length(Λ⃗)
52:     if M > M' then
53:         M = M'
```

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

54: $\vec{\Lambda} = \vec{\Lambda}'$
55: $\mathrm{ROM0} = \mathrm{ROM0}'$
56: $\mathrm{ROM1} = \mathrm{ROM1}'$
57: break
58: **else**
59: $B_{x_0}++$
60: $M' = M$
61: $\vec{\Lambda}' = \vec{\Lambda}$
62: $\mathrm{ROM0}' = \mathrm{ROM0}$
63: $\mathrm{ROM1}' = \mathrm{ROM1}$
64: **end if**
65: **end while**

C. Segmentation Experiments

Applying the methods described above to the six functions $f_1, \ldots, f_6$ gives $\mathrm{P2S}_L(\mathrm{US}), \mathrm{P2S}_R(\mathrm{US}), \mathrm{P2S}_{\mathrm{LR}}(\mathrm{US})$, $\mathrm{US}(\mathrm{US}), \mathrm{US}(\mathrm{US})$, and $\mathrm{US}(\mathrm{US})$, respectively. Fig. 5 illustrates the segmentations when Algorithm 2 is applied to the six functions using degree-2 splines and an error requirement of 2^{-14}. In this example the number of levels L is fixed at two and US is used for the inner segmentation for all cases. The six functions $f_1, \ldots, f_6$ require 19, 28, 64, 80, 6, and 7 segments, respectively. The optimal B_{x_0} that lead to the minimal M are found to be 9, 10, 11, 5, and 6 bits, respectively. The importance of selecting the right scheme for a given function is apparent if $\mathrm{US}(\mathrm{US})$ is used for f_1 instead of $\mathrm{P2S}_L(\mathrm{US})$. While $\mathrm{P2S}_L(\mathrm{US})$ results in 19 segments, $\mathrm{US}(\mathrm{US})$ requires 63 segments to deliver the same precision. It can be seen that the hierarchical segmentation closely resembles the balanced error segmentation shown in Fig. 6. The main difference is the expense of increased M.

Table III shows a comparison of the number of segments M for uniform, hierarchical, and balanced error segmentation for several different error requirements. The balanced error results are obtained from the binary splitting algorithm [4]. It is clear that hierarchical segmentation greatly reduces M when compared to uniform segmentation, particularly for highly nonlinear functions or stringent precision requirements. For all of the functions, the hierarchical segmentation requires only modestly more segments than the balanced error approach, and significantly fewer segments than uniform segmentation. In other words, it results in memory requirements that are relatively similar to what is needed in balanced error segmentation, while also delivering significant hardware savings as discussed earlier.

Interestingly, notable improvements are also achieved for relatively linear functions such as $f_5 = \ln(1+x)$ and $f_6 = 1/(1+x)$. The approximation of the natural logarithm has been widely studied in the literature, mostly using uniform segmentation. Fig. 7 shows the variation of number of segments M with error requirement for uniform and hierarchical segmentation for f_5. Due to the rigid nature of uniform segmentation (i.e., in which the total number of segments M must be a power of two), the curve exhibits step discontinuities. However, hierarchical segmentation is more flexible, giving a smoother curve. Depending on the error requirement, M can be reduced by over a factor of two relative to uniform segmentation. Since M is directly correlated to the table size needed for the polynomial coefficients, these savings are significant. Similar reductions in segment numbers relative to uniform segmentation are observed for f_6 and other range-reduced elementary functions such as

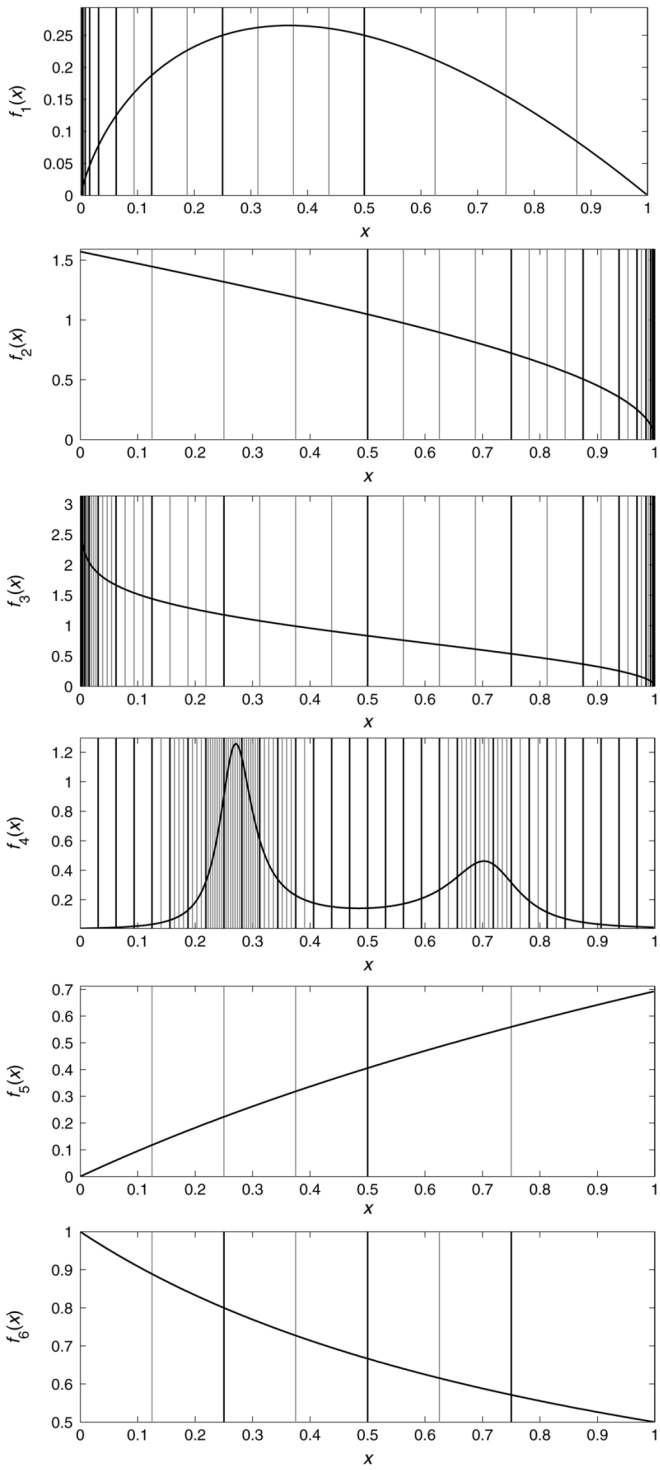


Fig. 5. Hierarchical segmentations to the six functions using degree-2 splines and an error requirement of 2^{-14}. Two levels of segmentation were used; the black and grey vertical lines indicate the boundaries for the outer segmentation Λ_0 and inner segmentation Λ_1. The six functions $f_1, \ldots, f_6$ employ segmentation using $\mathrm{P2S}_L(\mathrm{US}), \mathrm{P2S}_R(\mathrm{US}), \mathrm{P2S}_{\mathrm{LR}}(\mathrm{US}), \mathrm{US}(\mathrm{US}), \mathrm{US}(\mathrm{US})$, and $\mathrm{US}(\mathrm{US})$, respectively. They require 19, 28, 64, 80, 6, and 7 total segments, respectively.

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

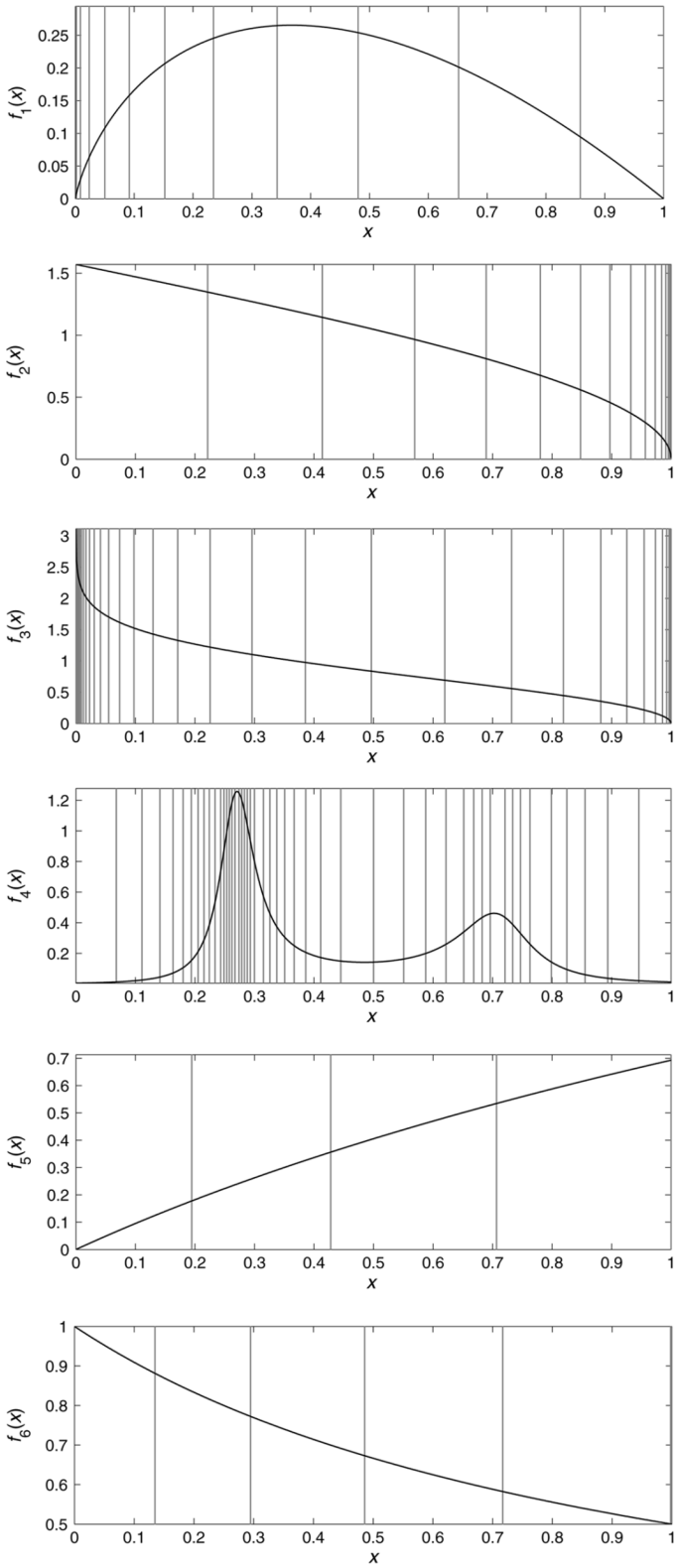


Fig. 6. Balanced error segmentation to the six functions using degree-2 splines and an error requirement of 2^{-14}. The vertical lines indicate the segment boundaries. The six functions $f_1, \ldots, f_6$ require 12, 18, 44, 48, 4, and 6 segments, respectively. The total segment count is reduced only slightly relative to hierarchical results in Fig. 5, but the hardware complexity is significantly increased.

$\cos(x)$ over $x = [0, \pi/2)$. For example, under the error requirement of 2^{-14} used in Fig. 5, $\cos(x)$ is uniformly segmented. For

TABLE III
COMPARISONS OF THE NUMBER OF SEGMENTS M FOR DEGREE-2 SPLINES. RESULTS FOR UNIFORM, HIERARCHICAL, AND BALANCED ERROR SEGMENTATION ARE SHOWN

Function	ϵ_{req}	Uniform	Hierarchical	Balanced
f_1	2^{-8}	8	5	3
	2^{-16}	4,096	31	19
	2^{-24}	1,048,576	191	120
f_2	2^{-8}	16	6	4
	2^{-16}	32,768	49	30
	2^{-24}	8,388,608	352	219
f_3	2^{-8}	128	11	7
	2^{-16}	262,144	126	78
	2^{-24}	67,108,864	1,076	547
f_4	2^{-8}	64	20	12
	2^{-16}	512	124	76
	2^{-24}	2,048	740	422
f_5	2^{-8}	2	2	2
	2^{-16}	8	8	7
	2^{-24}	128	55	45
f_6	2^{-8}	4	3	3
	2^{-16}	16	12	8
	2^{-24}	128	76	50

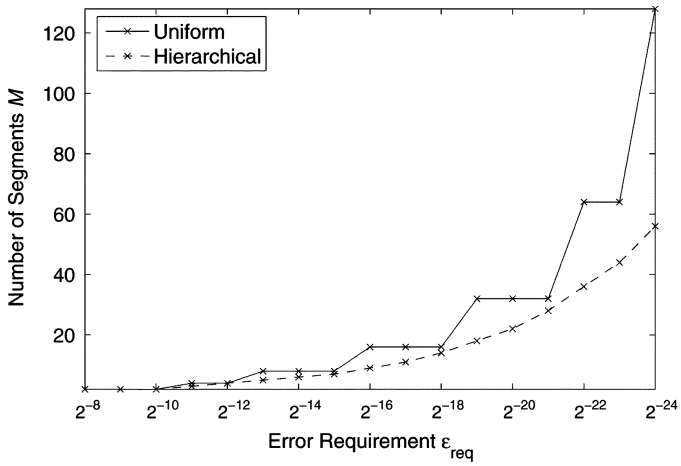


Fig. 7. Variation of the number of segments M with error requirement for uniform and hierarchical segmentation of the function $f_5 = \ln(1 + x)$.

tighter error requirements, nonuniform segmentation becomes superior.

V. HARDWARE ARCHITECTURE

Fig. 8 shows the hardware architecture for evaluating functions segmented with the hierarchical segmentation method. The P2S unit performs the P2S address decoding step [(5)–(7)] and the $B_{\hat{x}_i}$ computation (8)–(11). If $\Lambda_0 = \text{US}$, the P2S unit is bypassed. The bit selection unit selects the appropriate bits for x_0, x_1, and x_2 from the input x in conjunction with ROM0. Let $x[j : k]$ denote the set of consecutive bits from the jth to kth bit of x from its LSB. For US(US), the bit selection unit selects $x_0 = x[B_x - 1 : B_x - B_{x_0}], x_1 = x[B_x - B_{x_0} - 1 : B_x - B_{x_0} - B_{x_1}]$, and $x_2 = x[B_x - B_{x_0} - B_{x_1} - 1 : 0]$.

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

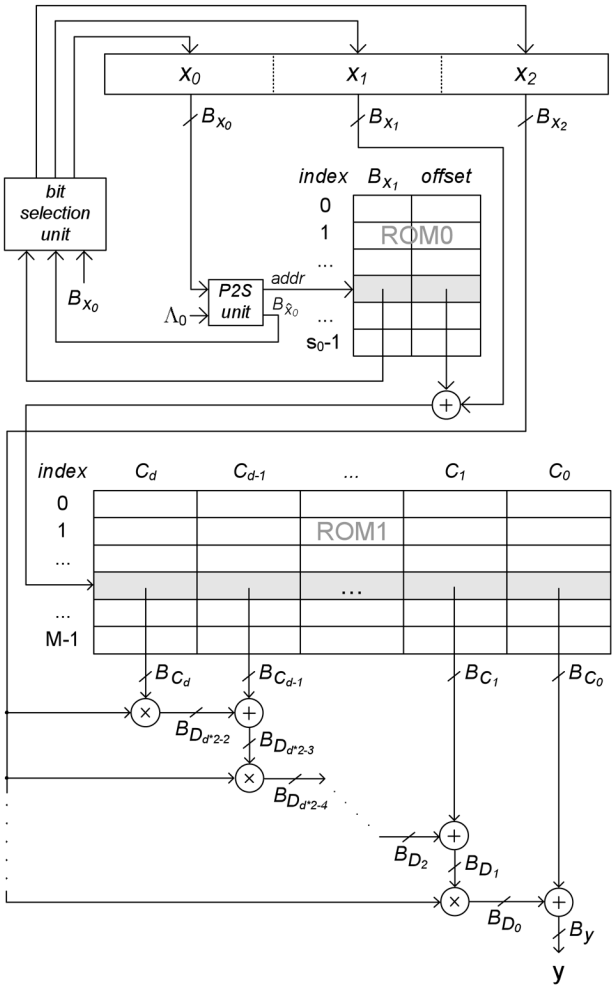


Fig. 8. Hardware architecture for evaluating functions segmented with the hierarchical segmentation method for degree-d splines. ROM0 contains information on the hierarchical segmentation, while ROM1 contains the polynomial coefficients to each spline segment.

A single barrel shifter is required for the selection of x_1 and x_2. For other hierarchy schemes, the selection process is more complex, since the bit locations of x_0 and x_1 can overlap as shown in the example of Table I. For these cases, two barrel shifters are used. The bit selection unit is illustrated in Fig. 9.

As described in Algorithm 2, ROM1 contains the polynomial coefficients to each segment. The depth s_0 of ROM0 is defined in (2)–(4), and the depth M of ROM1 is the total number of segments. The size of the two ROMs are defined as follows:

$$\mathrm{ROM0} = \{\lceil \log_2(\max(B_{x_1}))\rceil + \lceil \log_2(\max(\mathrm{offset}))\rceil\} \times s_0 \tag{18}$$

$$\mathrm{ROM1} = \sum_{i=0}^{d} B_{C_i} \times M. \tag{19}$$

In practice, ROM0 will be significantly smaller than ROM1, since its depth is bounded by s_0 which is generally small, and

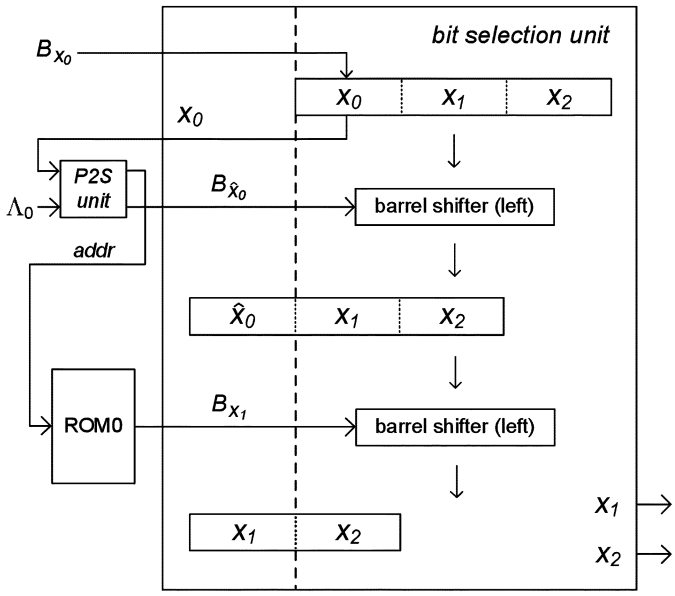


Fig. 9. Illustration of the bit selection unit in Fig. 8. The first barrel shifter can be omitted when $\mathrm{US(US)}$ because $B_{\hat{x}_0}$ stays constant.

the entries B_{x_1} and offset are also small. Horner's rule is used for the evaluation of the polynomials in the following form:

$$y = ((C_d x + C_{d-1})x + \cdots)x + C_0 \tag{20}$$

where x is the input, d is the degree, and $C_{0,\ldots,d}$ are the polynomial coefficients. Alternative polynomial evaluation techniques, such as the single multiplication degree-2 method in [10] can potentially be used as well.

Since x_0 and x_1 are implicitly known for a given segment, x_2 is used instead of x for the polynomial arithmetic to reduce the size of the operators. Let $x_{0,\ldots,1}$ denote the set of bits corresponding to x_0 and x_1, and $B_{x_{0,\ldots,1}}$ denote the bit-width of this set. x_2 is scaled occupy the range $[0,1)$. If $x = [0,1)$, this involves masking out the bits corresponding to $x_{0,\ldots,1}$, and shifting x by $B_{x_{0,\ldots,1}}$ to the left. One problem is the fact that the Chebyshev coefficients have originally been generated under the assumption that x will be used for the polynomial evaluation. Let us consider a segment with a degree-2 spline. x_2 is given by

$$x_2 = (x - x_{0,\ldots,1}) \times 2^{B_{x_{0,\ldots,1}}}. \tag{21}$$

Rearranging the equation gives

$$x = \frac{x_2}{2^{B_{x_{0,\ldots,1}}}} + x_{0,\ldots,1}. \tag{22}$$

Representing a degree-2 polynomial equation with the following coefficient labels:

$$y = C_2 x^2 + C_1 x + C_0 \tag{23}$$

in combination with (22) gives

$$y = \frac{C_2}{2^{2B_{x_{0,\ldots,1}}}} x_2^2 + \frac{C_1 + 2C_2 x_{0,\ldots,1}}{2^{B_{x_{0,\ldots,1}}}} x_2 + C_2 x_{0,\ldots,1}^2 + C_1 x_{0,\ldots,1} + C_0. \tag{24}$$

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

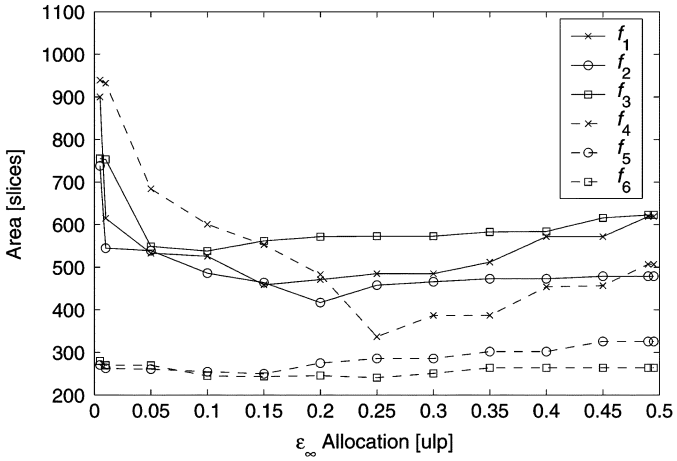


Fig. 10. Area variations with the inherent approximation error ϵ_∞ allocation for 16-bit degree-2 approximations.

Examining the second, first, and zeroth order terms, the newly transformed polynomial coefficients are

$$C_2' = \frac{C_2}{2^{2B_{x_{0,\ldots,1}}}} \tag{25}$$

$$C_1' = \frac{C_1 + 2C_2 x_{0,\ldots,1}}{2^{B_{x_{0,\ldots,1}}}} \tag{26}$$

$$C_0' = C_2 x_{0,\ldots,1}^2 + C_1 x_{0,\ldots,1} + C_0. \tag{27}$$

Note that the additions in (26) and (27) could lead to cancellation errors in floating-point arithmetic when the precisions are very high. In such cases it is desirable to use a multi-precision library.

Once the segmentation and coefficient generation have been completed, the next challenge is the determination of the signal bit-widths in the arithmetic data-path. Insufficient bit-widths can cause to overflows and error requirement violations, while excessive bit-widths can result in waste of valuable hardware resources. For the architecture in Fig. 8, bit-widths B_{C_0} to B_{C_d} and B_{D_0} to $B_{D_{d\times 2-2}}$ need to be determined. We use an adaptation of the MiniBit technique, optimized for polynomial-based function evaluation [2].

There are three main sources of errors when evaluating functions in digital arithmetic: 1) the inherent error ϵ_∞ due to approximating the function with polynomials; 2) quantization error ϵ_Q due to finite precision effects incurred when evaluating the polynomials; and 3) the error of the final output rounding step, which can cause a maximum error of $1/2$ ulp. In the worst case, ϵ_∞ and ϵ_Q will contribute additively, so to achieve faithful rounding, their sum must be less than $1/2$ ulp. We allocate a maximum of $1/4$ ulp for ϵ_∞ and the rest for ϵ_Q, which provides a good balance between the two error sources and will be discussed further in Section VI (see Fig. 10).

Quantization is usually performed in two modes: truncation which can cause a maximum error of 2^{-FB} (1 ulp), and round-to-nearest which can cause a maximum error of 2^{-FB-1} ($1/2$ ulp). Round-to-nearest must be performed at the output signal y to achieve faithful rounding, but either rounding mode can be used for the internal signals. Since truncation results in better delay and area characteristics over round-to-nearest, it is used for the internal signals. Note that the quantization of the coefficients $C_{d,\ldots,0}$ is performed with round-to-nearest at compile-time, while truncation is performed for the intermediate signals $d_{d\times 2-2,\ldots,0}$ at run-time (see Fig. 8).

TABLE IV
PROCESSING TIMES OF THE AUTOMATED HIERARCHICAL SEGMENTATION TOOL (SEE FIG. 1) ON AN INTEL PENTIUM-4 3.4 GHZ PC FOR DEGREE-2 APPROXIMATIONS TO FUNCTION f_4

Precision [bits]	Segmentation [s]	Bit-Width Optimization [s]	VHDL Generation [s]	Total [s]
8	8	7	2	17
12	29	9	2	40
16	84	14	3	101
20	202	22	4	228
24	511	50	5	566

VI. HARDWARE IMPLEMENTATION RESULTS

A Xilinx Virtex-4 XC4VLX100-12 FPGA is used for experimental implementation. Synplicity Synplify Pro 7.7.1 is used for synthesis, and Xilinx ISE 7.1.04i is used for placement and routing. Implementation results for degree-1 and degree-2 splines using Horner's rule are given. The designs are fully combinatorial with no pipeline registers. For the leading zero/one detectors required for the P2S unit, the design described by Oklobdzija [24] is adopted.

The primary building block of Xilinx Virtex series FPGA is the "slice," which consists of two four-input lookup tables (LUTs), two registers and two multiplexors, and some additional circuitry such as carry logic and AND/XOR gates. The four input LUT can also be used as 16 $\times$ 1 RAM or a 16-bit shift register. Although the Virtex-4 FPGA contains hardwired dedicated RAMs and multiply-and-add blocks, we consider implementations based on slices only, in order to obtain unbiased area and delay comparisons. The "precision" in the results refers to total number of bits (sum of number of integer and fractional bits) at the output.

Table IV shows the processing times of the automated hierarchical segmentation tool (see Fig. 1) on an Intel Pentium-4 3.4 GHz PC with 2 GB RAM for degree-2 approximations to f_4. The segmentation step described in Algorithm 2 takes the largest portion of the processing time followed by the bit-width optimization and VHDL generation steps. The main bottleneck of the segmentation algorithm is the sequential search for the optimal B_{x_0}. If the optimal B_{x_0} is used initially for 24-bit precision for instance, the segmentation time is reduced to 57 s.

Fig. 10 explores the area variations with ϵ_∞ allocation discussed in Section V for 16-bit degree-2 approximations to the six functions. When the ϵ_∞ allocation is close to zero, the number of segments are excessive leading to large areas. On the other hand, when ϵ_∞ allocation is close to 1/2, errors allocated for quantization effects ϵ_Q are too small, resulting larger operator and coefficient table sizes. As mentioned in Section V, we have chosen 1/4 for the ϵ_∞ allocation which provides a good balance between the two error sources for the case studies concerned in this work.

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

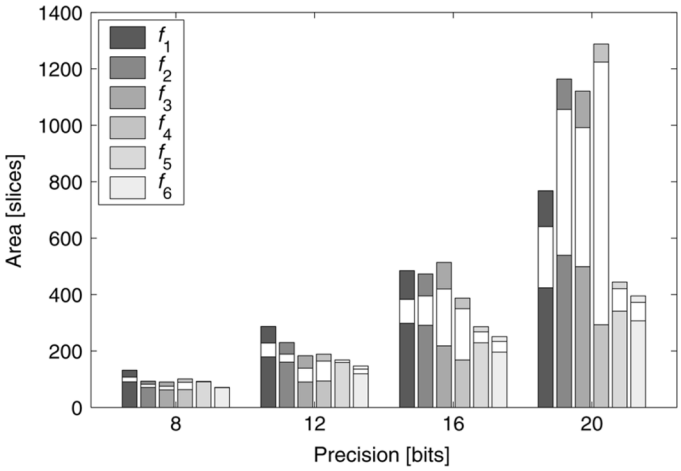


Fig. 11. Area comparisons for degree-2 approximations. The top (grey), middle (white), and bottom (grey) parts of each bar indicate the area portion for address decoding, ROM1, and polynomial evaluation, respectively.

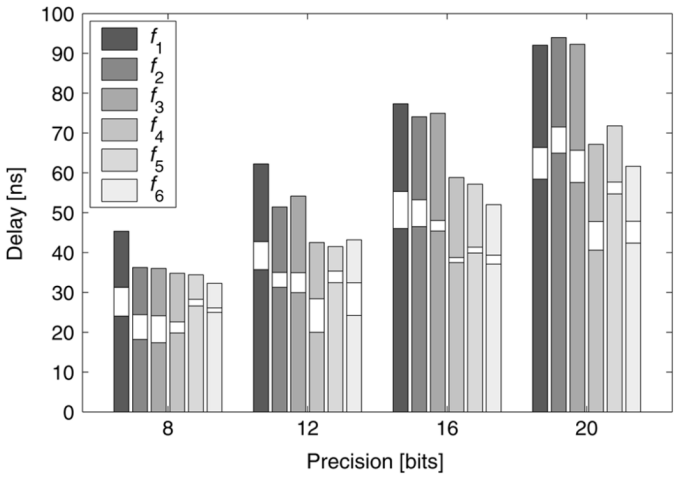


Fig. 12. Delay comparisons for degree-2 approximations. The top (grey), middle (white), and bottom (grey) parts of each bar indicate the delay portion for address decoding, ROM1, and polynomial evaluation, respectively.

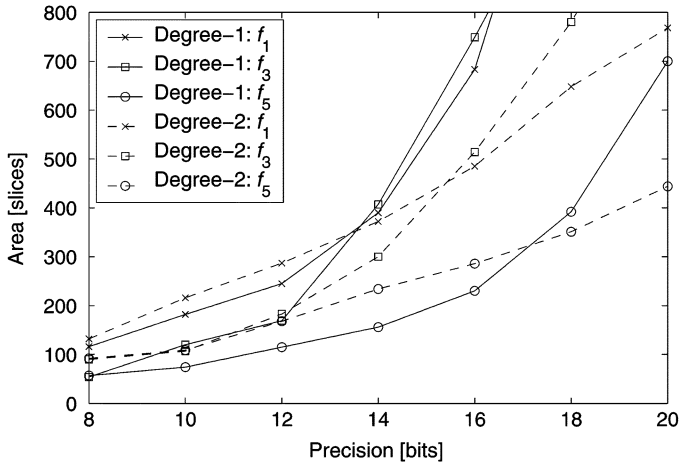


Fig. 13. Area comparisons for degree-1 and degree-2 approximations.

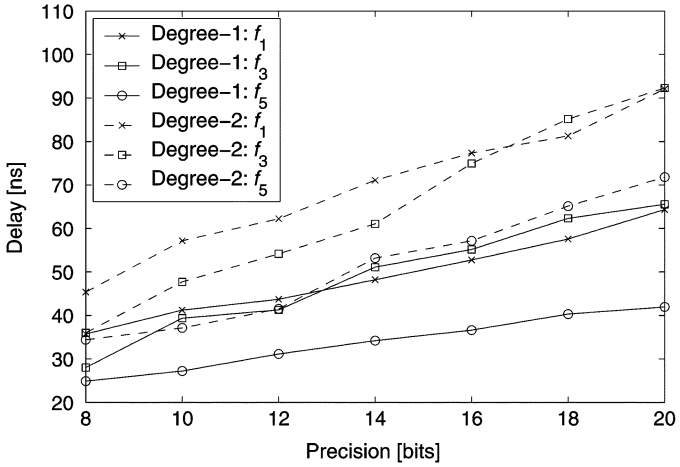


Fig. 14. Delay comparisons for degree-1 and degree-2 approximations.

Figs. 11 and 12 examine area and delay behavior of degree-2 approximations. The top (grey), middle (white), and bottom (grey) parts of each bar indicate the area portion for address decoding (top part of Fig. 8 for computing the address for ROM1), ROM1, and polynomial evaluation respectively. The area results indicate that the portion for address decoding is generally small in all cases. The address decoding for $f_1, \ldots, f_3$ is slightly larger than $f_4, \ldots, f_6$, because $f_1, \ldots, f_3$ employ power of two based outer segmentations which are more complex to decode than the uniform based outer segmentations employed in $f_4, \ldots, f_6$. The portion for ROM1 increases with precision due to increasing number of required segments M (as shown in Table III). For a given precision, the ROM1 portion varies across functions due to different M requirements and the degree of logic minimization performed during synthesis. The portion for polynomial evaluation increases with precision due to increased bit-width requirements in the data path. While an exponential increase in area with precision was observed in Fig. 11, the results in Fig. 12 indicate that delay increases in a linear fashion with precision. Compared to the single multiplication degree-2 method by Detrey and de Dinechin [10], our results for f_5 indicate similar area characteristics but notably slower speeds. This suggests that the performance of the degree-2 designs shown here can be further improved by utilizing optimized polynomial evaluation architectures over the standard Horner's method.

Figs. 13 and 14 show area and delay comparisons between degree-1 and degree-2 approximations to f_1, f_3, and f_5. The area for degree-1 approximations increase more rapidly than the area degree-2 approximations. For f_1, precisions exceeding 13 bits, degree-2 splines start to become more cost effective in terms of area than the degree-1 splines. This intersection point occurs at 11 bits for f_3 and at 17 bits for f_5. The delay exhibits a linear increase with precision for all cases due to the increasing internal bit-widths. Degree-1 approximations are faster than degree-2 approximations due to their shallower polynomial computation chains.

Table V compares the area and delay results between uniform and hierarchical segmentation for degree-2 approximations to f_1 and f_4. When hierarchical segmentation is used, significant area savings are obtained at the expense of moderate increase in delay. The area savings occur due to the fewer numbers of

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

TABLE V
AREA AND DELAY COMPARISONS BETWEEN UNIFORM AND HIERARCHICAL SEGMENTATION FOR DEGREE-2 APPROXIMATIONS TO FUNCTIONS f_1 AND f_4

Function	Precision [bits]	Area [slices]		Delay [ns]	
		Uniform	Hierarchical	Uniform	Hierarchical
f_1	8	58	132	20.7	45.4
	12	523	287	29.0	62.2
	16	6992	485	36.5	77.4
	20	-	768	-	92.1
	24	-	1715	-	98.6
f_4	8	84	101	24.1	34.8
	12	226	189	39.4	42.5
	16	882	387	44.2	58.8
	20	1947	1288	58.5	67.2
	24	7432	3311	71.6	88.8

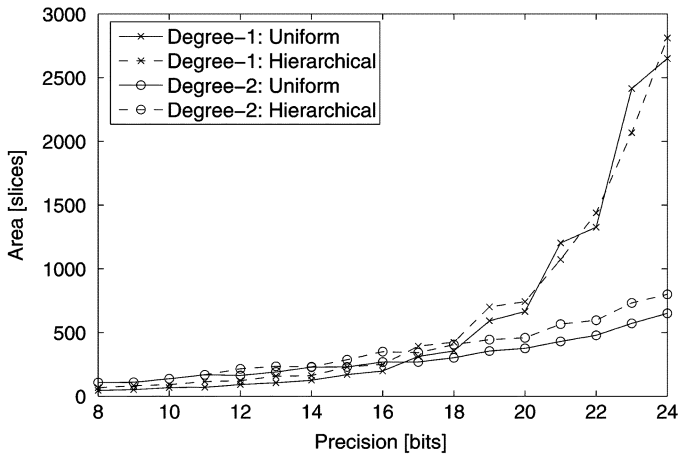


Fig. 15. Area comparisons of uniform and hierarchical segmentations for approximating $f_5 = \ln(1 + x)$.

segments leading to smaller coefficient tables. The delay penalties are caused by the coefficient address decoding circuitry. For function f_1, precisions beyond 16 bits could not be mapped on to the FPGA due to the rapid area increase (mainly memory) of uniform segmentation.

Fig. 15 presents area comparisons of uniform and hierarchical segmentations for approximating $f_5 = \ln(1 + x)$. Earlier, Fig. 7 and Table III indicated that even for relatively linear functions such as f_5, hierarchical segmentation always leads to smaller number of segments (or memory requirements) than uniform segmentation. However, when the overall hardware area is examined, the results in Fig. 15 show that hierarchical segmentation does not necessarily result in the smallest area; though memory reductions are achieved for ROM1, there are area overheads associated with address decoding. For degree-1 approximations, uniform segmentation is always more area efficient for precisions up to 20 bits. Beyond 20 bits, the most area efficient method depends on the precision. With degree-2 approximations however, there are relatively few segments required for the precisions shown (up to 24 bits). Thus, the reduction in ROM1 achieved with hierarchical segmentation has little impact on the overall hardware area. At higher precisions however, we suspect there will be an intersection point (where hierarchical segmentation can be more area efficient), analogous to the degree-1 case.

For an 8-bit degree-1 approximation to $\cos(x)$ over $x = [0, \pi/2)$ on a Xilinx Virtex-II XC2V4000-6 device, the balanced error approach by Sasao *et al.* [19] results in a delay of 67 ns, while the hierarchical segmentation results in 33 ns. We expect that the delay gap between the two methods will be wider for approximations with larger M due to the increased burden on the address decoding process. In other words, hierarchical segmentation leads to significant reduction in delay over the balanced error approach with modest increase in M (the behavior of $\cos(x)$ is similar to that of f_5 and f_6 in Table III).

It is also instructive to compare memory requirements between the balanced error approach in [18] and hierarchical segmentation. The two main sources of memory utilization are the tables for coefficient address computation and coefficient storage. In hierarchical segmentation the coefficient table (ROM1) is dominant and memory for coefficient address computation (ROM0) accounts for only a small fraction of the overall memory utilization. By contrast, in the balanced error approach the large number of cascaded LUTs that are used for address computation can account for over 90% of the total memory utilization [18], with the remaining memory used for the coefficient tables. Thus, while the balanced error method requires fewer segments and thus less memory for coefficients, these savings can be more than consumed by the memory for the address computation.

These issues are illustrated in Table VI with specific reference to the functions $f_3 = \sqrt{-\ln(x)}$ over $x = [2^{-5}, 1)$ and $f_5 = \ln(1 + x)$ over $x = [0, 1)$. Memory requirements for address computation and coefficient storage are given for hierarchical segmentation. For balanced error, the results presented in [20] for the sum of address computation and coefficient memory are incorporated. The table does not specify the contribution of the address computation memory to the overall memory for balanced error as this information is not directly available in [20], though, as noted above, the cascaded LUTs for coefficient address computation in the balanced error implementation can consume significantly more memory than the corresponding structure in the hierarchical approach. It should also be noted that the memory utilization for balanced error method could likely be improved through the use of more memory-efficient address computation strategies.

VII. CONCLUSION

We have presented an efficient method for evaluating functions via splines with a hierarchical segmentation scheme. The use of segmentation hierarchies involving uniform splines and splines with size varying by powers of two enables efficient, high precision coverage of highly nonlinear function regions. The methodology is automated given user-specified precision requirements and choice of approximation method. The hierarchical segmentation method results in significant reduction in number of required spline segments for given precision requirements compared to the commonly-used uniform segmentation approach. Compared to balanced error segmentation, the method presented here gives significantly reduced delay and memory requirements. The bit-widths of the fixed-point coefficients and arithmetic operators are optimized via the MiniBit approach, enabling us to guarantee 1 ulp accuracy at the output

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

TABLE VI
MEMORY REQUIREMENT COMPARISONS IN BITS BETWEEN THE SASAO BALANCED ERROR APPROACH AND HIERARCHICAL SEGMENTATION FOR THE EVALUATION OF $f_3 = \sqrt{-\ln(x)}$ OVER $x = [2^{-5}, 1)$ AND $f_5 = \ln(1+x)$ OVER $x = [0, 1)$. THE NUMBERS SHOWN FOR HIERARCHICAL SEGMENTATION ARE THE SUM OF ROM0 AND ROM1 (SEE FIG. 8), WHILE THE NUMBERS SHOWN IN BRACKETS ARE ROM0. THE REDUCTION FACTOR IS THE RATIO OF THE TWO MEMORY SIZES

Method	f_3				f_5			
	16-bit Precision		24-bit Precision		16-bit Precision		24-bit Precision	
	Degree-1	Degree-2	Degree-1	Degree-2	Degree-1	Degree-2	Degree-1	Degree-2
Sasao [20]	74,240	11,264	2,662,400	173,056	20,096	2,448	700,416	19,136
Hierarchical	16,548 (240)	5,836 (180)	524,571 (652)	76,387 (468)	3,238 (448)	522 (32)	63,392 (896)	4,921 (448)
Reduction Factor	4.5	1.9	5.1	2.3	6.2	4.7	11.0	3.9

in an analytical manner. A hardware architecture for evaluating functions segmented hierarchically has been presented with various realizations on a Xilinx Virtex-4 FPGA.

Current and future work includes exploring the use of hierarchical segmentation for approximation methods other than piecewise polynomials, applying the proposed approach to a wide range of examples, and evaluating its effectiveness for various FPGA devices.

ACKNOWLEDGMENT

The authors would like to thank H. Kim and the anonymous reviewers for their constructive comments and suggestions.

REFERENCES

[1] D. Lee, A. A. Gaffar, O. Mencer, and W. Luk, "Optimizing hardware function evaluation," *IEEE Trans. Computers*, vol. 54, no. 12, pp. 1520–1531, Dec. 2005.

[2] D. Lee and J. Villasenor, "A bit-width optimization methodology for polynomial-based function evaluation," *IEEE Trans. Computers*, vol. 56, no. 4, pp. 567–571, Apr. 2007.

[3] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, 2nd ed. Basel, Switzerland: Birkhauser, 2006.

[4] D. Lee, W. Luk, J. Villasenor, and P. Cheung, "Hierarchical segmentation schemes for function evaluation," in *Proc. IEEE Int. Conf. Field-Program. Technol.*, 2003, pp. 92–99.

[5] M. Ercegovac and T. Lang, *Division and Square Root: Digit Recurrence Algorithms and Implementations*. Norwell, MA: Kluwer, 1994.

[6] Y. Hu, "CORDIC-based VLSI architectures for digital signal processing," *IEEE Signal Process. Mag.*, vol. 9, no. 3, pp. 17–34, Mar. 1992.

[7] J. Stine and M. Schulte, "The symmetric table addition method for accurate function approximation," *J. VLSI Signal Process.*, vol. 32, no. 2, pp. 167–177, 1999.

[8] F. de Dinechin and A. Tisserand, "Multipartite table methods," *IEEE Trans. Computers*, vol. 54, no. 3, pp. 319–330, Mar. 2005.

[9] I. Koren and O. Zinaty, "Evaluating elementary functions in a numerical coprocessor based on rational approximations," *IEEE Trans. Computers*, vol. 39, no. 8, pp. 1030–1037, Aug. 1990.

[10] J. Detrey and F. de Dinechin, "Table-based polynomials for fast hardware function evaluation," in *Proc. Int. Conf. Appl.-Specific Syst., Arch., Process.*, 2005, pp. 328–333.

[11] A. Noetzel, "An interpolating memory unit for function evaluation: Analysis and design," *IEEE Trans. Computers*, vol. 38, no. 3, pp. 377–384, Mar. 1989.

[12] J. Piñeiro, S. Oberman, J.-M. Muller, and J. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Trans. Computers*, vol. 54, no. 3, pp. 304–318, Mar. 2005.

[13] D. Lewis, "Interleaved memory function interpolators with application to an accurate LNS arithmetic unit," *IEEE Trans. Computers*, vol. 43, no. 8, pp. 974–982, Aug. 1994.

[14] J. Coleman, E. Chester, C. Softley, and J. Kadlec, "Arithmetic on the European logarithmic microprocessor," *IEEE Trans. Computers*, vol. 49, no. 7, pp. 702–715, Jul. 2000.

[15] C. Lawson, "Characteristic properties of the segmented rational minimax approximation problem," *Numerische Mathematik*, vol. 6, pp. 293–301, 1964.

[16] R. Esch and W. Eastman, "Computational methods for best spline approximation," *J. Approximation Theory*, vol. 2, pp. 85–96, 1969.

[17] T. Pavlidis and A. Maika, "Uniform piecewise polynomial approximation with variable joints," *J. Approximation Theory*, vol. 12, pp. 61–69, 1974.

[18] T. Sasao, J. Butler, and M. Riedel, "Application of LUT cascades to numerical function generators," in *Proc. Workshop Synthesis Syst. Integr. Mixed Inf. Technol.*, 2004, pp. 422–429.

[19] T. Sasao, S. Nagayama, and J. Butler, "Programmable numerical function generators: Architectures and synthesis method," in *Proc. IEEE Int. Conf. Field-Program. Logic Its Appl.*, 2005, pp. 118–123.

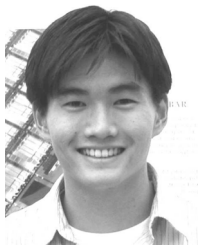
[20] S. Nagayama, T. Sasao, and J. Butler, "Programmable numerical function generators based on quadratic approximation: Architecture and synthesis method," in *Proc. IEEE Asia South Pac. Des. Autom. Conf.*, 2006, pp. 378–383.

[21] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2004, pp. 428–433.

[22] J. Jiang, S. Schmidt, W. Luk, and D. Rueckert, "Parameterizing reconfigurable designs for image warping," in *Proc. SPIE*, 2002, vol. 4867, pp. 86–97.

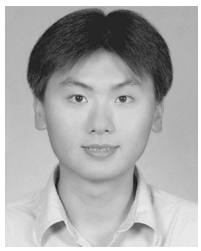
[23] D. Lee, W. Luk, J. Villasenor, and P. Cheung, "A Gaussian noise generator for hardware-based simulations," *IEEE Trans. Computers*, vol. 53, no. 12, pp. 1523–1534, Dec. 2004.

[24] V. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 1, pp. 124–128, Jan. 1994.



Dong-U Lee (M'05) received the B.Eng. degree in information systems engineering and the Ph.D. degree in computing, both from Imperial College London, London, U.K., in 2001 and 2004, respectively.

From 2005 to 2007, he was a Postdoctoral Researcher with the Electrical Engineering Department, University of California, Los Angeles (UCLA), where he developed high-performance hardware designs for wireless communications and mathematical function evaluations. He is now a Research Scientist with Mojix, Inc., Los Angeles, CA, where he is specializing in hardware implementation aspects of RFID readers. His research interests include computer arithmetic, communications, design automation, reconfigurable computing, and video image processing.



Ray C. C. Cheung (M'07) received the B.Eng. and M.Phil. degrees in computer engineering and computer science and engineering from The Chinese University of Hong Kong (CUHK), Hong Kong, in 1999 and 2001, respectively, and the Ph.D. degree in computing from Imperial College London, London, U.K., in 2007.

He is currently a Research Engineer with Solomon Systech Limited, Hong Kong. From January 2002 to December 2003, he was an instructor with the Department of Computer Science and Engineering, CUHK. He visited Stanford University and the University of California at Los Angeles (UCLA) in 2005 and 2006 as a Visiting Scholar. He worked as a Postdoctoral Researcher with the Electrical Engineering Department, UCLA. His current research interests include reconfigurable computing and computer arithmetic hardware designs.

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.

Wayne Luk (SM'06) received the M.A., M.Sc., and D.Phil. degrees in engineering and computer science from the University of Oxford, Oxford, U.K.

He is Professor in Computer Engineering with the Department of Computing, Imperial College London, London, U.K., and a Visiting Professor with Stanford University, Stanford, CA, and with Queen's University Belfast, Belfast, U.K. His research interests include theory and practice of customizing hardware and software for specific application domains, such as multimedia, communications, and finance. Much of his current work involves high-level compilation techniques and tools for parallel computers and embedded systems, particularly those containing reconfigurable devices such as field-programmable gate arrays.

John D. Villasenor (SM'98) received the B.S. degree from the University of Virginia, Charlottesville, in 1985, and the M.S. and the Ph.D. degrees from Stanford University, Stanford, CA, in 1986 and 1989, respectively, all in electrical engineering.

He joined the Electrical Engineering Department , University of California, Los Angeles (UCLA), in 1992, where he is currently a Professor. From 1990 to 1992, he was with the Radar Science and Engineering Section, Jet Propulsion Laboratory, Pasadena, CA, where he developed methods for imaging the earth from space. He served as Vice Chair of the Department from 1996 to 2002. At UCLA, his research efforts lie in communications, computing, imaging and video compression, and networking.

Authorized licensed use limited to: Imperial College London. Downloaded on December 18, 2008 at 20:04 from IEEE Xplore. Restrictions apply.