A Unified Architecture for the Accurate and High-Throughput Implementation of Six Key Elementary Functions

Amirhossein Alimohammad, *Member*, *IEEE*, Saeed Fouladi Fard, and Bruce F. Cockburn, *Member*, *IEEE*

Abstract—This paper presents a unified architecture for the compact implementation of several key elementary functions, including reciprocal, square root, and logarithm, in single-precision floating-point arithmetic. The proposed high-throughput design is based on uniform domain segmentation and curve fitting techniques. Numerically accurate least-squares regression is utilized to calculate the polynomial coefficients. The architecture is optimized by analyzing the trade-off between the size of the required memory and the precision of intermediate variables to achieve the minimum 23-bit accuracy required for single-precision floating-point representation. The efficiency of the proposed unified data path is demonstrated on a common field-programmable gate array.

Index Terms—Floating-point arithmetic, single-precision arithmetic, reciprocal, square root, logarithm, computer arithmetic.

1 Introduction

Many scientific algorithms rely on floating-point representations and arithmetic to ensure a sufficiently large dynamic range. Numerous algorithms have been proposed previously for implementing key elementary floating-point functions, such as reciprocal and square root. The major algorithms fall into two main categories: iterative methods [1], [2], [3], [4] and noniterative techniques [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23]. Iterative methods can be subclassified into multiplicative algorithms and digit recurrence approaches. The multiplicative algorithms use a series of multiplications, subtractions, bit inversions, and shifts to iteratively refine a sufficiently accurate initial guess. The multiplicative algorithms typically converge at a quadratic rate, which means that the number of accurate bits in the approximation roughly doubles at every iteration. In contrast, the subtractive digit recurrence approaches compute the elementary functions directly and have a linear convergence rate, with one bit of increased accuracy produced at every iteration. Due to their relatively long latencies, the subtractive algorithms usually require dedicated hardware [24]. The multiplicative algorithms are better suited for high-performance computations due to their faster convergence.

Noniterative techniques compute the function value directly without convergent iterations and can be subdivided into table-bound methods [5], [6], [7], [8], [9], compute-bound methods [10], [11], [12], [13], and hybrid methods [14], [15], [16], [17], [18], [19], [20], [21], [22], [23]. Table-bound methods, such as partial product arrays [5], add-table lookup-add [6], and bipartite tables [7], [8], [9], which require one or only a few tables and adders, are inefficient for single-precision computations due to the rapid growth in the size of the lookup tables (LUTs) with the accuracy of the result. For example, the bipartite table method in [7] uses two relatively large tables of sizes of about $2^{16} \times 24$ and $2^{16} \times 8$ bits and a subtractor, but no multipliers, to achieve single-precision accuracy.

Compute-bound schemes minimize the storage size but require a relatively large number of multiplications and additions, resulting in longer execution times. For example, the rational approximation scheme in [10] requires at least one division operation and the methods in [11], [12] use a very small LUT together with square, cubic, or higher degree polynomial approximations.

Hybrid methods provide high-speed function approximation by combining relatively small LUTs with the evaluation of linear or quadratic polynomials, achieving both a reduction in memory size compared with the table-bound schemes and significant speedups compared with the compute-bound methods [25]. For example, [21] and [22], [23] use first-order and second-order interpolating polynomials, respectively. The designs in [18], [19], [20] use second-order piecewise polynomial approximations of the functions based on the Chebyshev approximations, Taylor expansion, and minmax approximations, respectively.

While most of the published schemes utilize a separate data path for implementing each elementary function [2], we propose a unified architecture for the compact and high-throughput implementation of six key elementary functions. Sharing data-path units can permit extensive hardware reduction and an especially compact transcendental functional unit (TFU) design. Moreover, a single shared data path implies the same latency for all function

- *A. Alimohammad and S. Fouladi Fard are with Ukalta Engineering, 4344 Enterprise Square, 10230 Jasper Avenue, Edmonton, AB T5J 4P6, Canada. E-mail: {amir, saeed}@ukalta.com.*
- *B.F. Cockburn is with the Department of Electrical and Computer Engineering, University of Alberta, ECERF Bldg., 2nd Floor 9107-116 Street NW, Edmonton, AB T6G 2V4, Canada. E-mail: cockburn@ece.ualberta.ca.*

Manuscript received 22 Dec. 2008; revised 01 June 2009; accepted 19 Aug. 2009; published online 29 Oct. 2009.
Recommended for acceptance by W. Najjar.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-12-0632.
Digital Object Identifier 10.1109/TC.2009.169.

approximations, which simplifies the design of the TFU's control unit. Our design is based on joint uniform segmentation and curve fitting approximations to accurately implement the desired elementary functions to single-precision floating-point accuracy. We use linear and quadratic polynomials and fully pipelined data paths to achieve high-throughput implementations. We also compare the accuracy and numerical stability of several alternative curve fitting algorithms to select a robust technique. The size of the required memories and the precision of the shared data-path's intermediate variables were optimized using exhaustive simulations across the approximation interval $[1, 2)$ to ensure 23-bit final accuracy, to minimize resource requirement, and to maximize the data throughput.

The rest of this paper is organized as follows: Section 2 briefly reviews the widely used segmentation and curve fitting techniques for approximating elementary functions. The numerical stability of alternative curve fitting techniques is investigated. We propose a numerically stable solution for the accurate approximation of six important elementary functions. Section 3 describes the proposed shared data-path architecture. Section 4 compares the proposed technique with previous high-throughput techniques, including designs that use a separate data path for implementing each function and designs that use a unified architecture for implementing multiple different elementary functions. Finally, Section 5 makes some concluding remarks.

2 Segmentation and Curve Fitting Techniques

Normalized IEEE floating-point numbers are given in $(-1)^s x 2^e$ format where $s \in \{0, 1\}$ is the sign, the mantissa x lies in $1 \leq x < 2$ with an implicit leading 1 (only the fractional 23 bits are given), and e is the signed integer exponent. We will denote by $f(x)$ any one of the elementary functions x^{-1}, $x^{0.5}$, $x^{-0.5}$, $\log(x)$, $\ln(x)$, and 2^x over the interval of interest $x \in [1, 2)$. Our method focuses on function approximation within this predefined input interval. The input domain differs from the domains $[1, 4)$ and $[0, 1)$ that are normally used for square root and exponentiation, respectively. Reducing the argument x to the input interval of $1 \leq x < 2$ and then reconstructing the value $f(x)$ is assumed to be performed before and after function approximation, respectively, using conventional techniques [19], [23]. The common domain simplifies the design of a shared data path for all six functions.

For an efficient approximation of function $f(x)$, instead of using a single polynomial with a relatively large degree, the interval $x \in [1, 2)$ is partitioned into subintervals (segments). Different functions may require different segmentations. For example, in [26] we used a hybrid segmentation (i.e., a combination of linear and nonlinear segmentations) for $f(x) = \sqrt{-2\ln(x)}$. For clarity, Fig. 1 plots $f(x)$ for four of the six elementary functions over $[1, 2)$. Since over this interval all six functions are well behaved and slowly varying, we can use a uniform domain segmentation. We use $n = 2^m$ segments for each $f(x)$ where the m most significant bits of the normalized mantissa (i.e., $d = x - 1$) are used as a segment index.

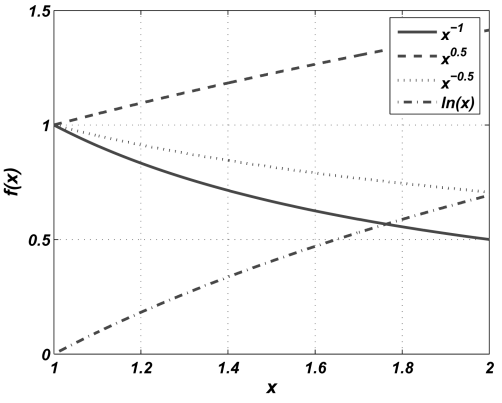

Fig. 1. Plots of $1/x$, $\sqrt{x}$, $1/\sqrt{x}$, and $\ln(x)$ over $x \in [1, 2)$.

Then, a low-degree fitting polynomial $g(x) = \sum_{j=0}^{p} a_j x^j$ can be used to approximate $f(x)$ within each segment. After defining $g(x)$ for each segment (i.e., finding its coefficients), the value of $f(x_i)$ can be approximated by calculating the value of the fitting polynomial $g(x)$ at $x = x_i$.

Curve fitting is the process of finding the coefficients a_j of each $g(x)$ using a given set of distinct data points (x_i, y_i), where $i = 0, 1, \ldots, n$ and $n > p$. One standard optimization method for finding the coefficients of $g(x)$ is least-squares regression [27]. In the regression method, the coefficients a_j of the best fitting curve $g(x) = \sum_{j=0}^{p} a_j x^j$ are obtained by minimizing the sum of squared residuals (or deviations) from a given set of n data points (x_i, y_i). For a given set of data points, the best fitting curve is generally not unique and different fitting curve choices are possible depending on the maximum allowed deviation of the fitting curve $g(x)$ from $f(x)$. The approximation error can be defined as follows:

$$e(\mathbf{a}) = \sum_{i=1}^{n} r_i^2(\mathbf{a}) = \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{p} a_j x_i^j \right)^2. \quad (1)$$

Minimization of (1) produces a set of normal equations (i.e., a set of linear equations in the polynomial coefficients), which are solved to yield an estimate of a_j, where $j = 0, \ldots, p$. When the number n of given data points is more than the number of unknown coefficients (i.e., $n > p + 1$), the set of normal equations is an overdetermined system of n linear equations in $p + 1$ unknowns a_j. We evaluated the numerical stability of four different algorithms that optimize the polynomial coefficients to minimize the error in (1), as reviewed below.

One optimization technique is to form $p + 1$ gradient equations with respect to each parameter (i.e., $\partial e / \partial a_j$) and then solve for the zeros. In this case, a set of normal equations is obtained, which can be written in matrix notation as $(\mathbf{X^T X})\tilde{\mathbf{a}} = \mathbf{X^T y}$, where $\mathbf{X}$ is an $n \times (p + 1)$ matrix. When the $p + 1$ columns of matrix $\mathbf{X}$ are linearly independent (i.e., $\mathbf{X}$ is positive definite and has full rank) and $n > p + 1$ (as otherwise the matrix $\mathbf{X}^T\mathbf{X}$ is not invertible), the above set of normal equations has the unique solution $\tilde{\mathbf{a}} = (\mathbf{X^T X})^{-1}\mathbf{X^T y}$, which yields the vector $\tilde{\mathbf{a}} = \tilde{a}_0, \ldots, \tilde{a}_p$ containing the optimal coefficient values.

The second standard technique for finding the minima of the quadratic error function (1) is to use Newton's iterative optimization equation $\mathbf{a}_{k+1} = \mathbf{a}_k - \mathbf{d}_k$, where $k \geq 0$

denotes the iteration number and the vector increment $\mathbf{d}_k$ is the shift vector. After choosing sufficiently accurate initial estimates for the coefficients [27], their values are refined by successive approximations of

$$\mathbf{d}_k = \left[\mathbf{H}e(\mathbf{a}_k)\right]^{-1}\nabla e(\mathbf{a}_k),$$

where $\nabla e(\mathbf{a}_k)$ and $\mathbf{H}e(\mathbf{a}_k)$ denote the gradient and Hessian of $e(\mathbf{a}_k)$, respectively. Another technique for finding the minimum of a sum of squared functions, without the need for calculating the second-order Hessian matrix, is the Gauss-Newton algorithm [27]. The Gauss-Newton's step $\mathbf{d}_k$ can be computed from

$$\mathbf{d}_k = \left(\mathbf{J}_k^T\mathbf{J}_k\right)^{-1}\left(\mathbf{J}_k^T\mathbf{r}\right),$$

where $\mathbf{J}_k = \mathbf{J}_\mathbf{g}(\mathbf{x}_k)$ is the $n \times (p+1)$ Jacobian matrix of $g(\cdot)$ with respect to $\mathbf{a}$.

We found that when the number n of segments is chosen to be much larger than the polynomial degree (e.g., the interval $[1, 2)$ is segmented into 1,024 segments and linear or quadratic approximations are chosen), then all three optimization algorithms are unstable when inverting matrices $\mathbf{X^T X}$, $\mathbf{J}_k^T\mathbf{J}_k$, and the Hessian matrix. Intuitively, when the intervals are very narrow (in our case, for example, only 2^{-10} apart), then the coefficients differ only very slightly in value, making the matrices ill conditioned (i.e., close to a singular matrix). Hence, the least-squares estimate amplifies the approximation errors leading to inaccurate results [28]. Moreover, we found that the double-precision floating-point arithmetic supported by many computers is not sufficient to accurately represent the intermediate values of the variables in these optimization algorithms. Newton's iterative approach has the additional problem that when there are multiple local minima in (1), the algorithm may fail to converge to the global minimum. A more robust algorithm for finding the global minimum of the sum of squares in (1) is to use regularization. More specifically, we selected the Levenberg-Marquardt regularization scheme in which the vector increment $\mathbf{d}_k$ can be calculated as

$$\mathbf{d}_k = \left(\mathbf{J}_k^T\mathbf{J}_k + \mu\mathbf{\Lambda}\right)\left(\mathbf{J}_k^T\mathbf{r}\right),$$

where μ is the damping parameter and $\mathbf{\Lambda}$ is a positive diagonal matrix [29]. This technique provides a robust solution for finding the coefficients of approximating polynomials while minimizing the error function (1).

After approximating the values of the coefficients a_j of each polynomial for the n different segments of $f(x)$, they are stored in an on-chip memory. To minimize the size of this memory, we need to minimize the number of segments and the word length of the computed coefficients. For the most compact possible design, we also need to find the minimum precision of the intermediate signals in the polynomial data path that achieves 23-bit-accurate function values.

3 Optimization of the Unified Architecture

The accuracy of a polynomial approximation depends on the size of the interval over which the approximation is made, the order of the polynomial, and the method for computing the coefficients. The minimum number of segments is determined by the degree of the approximation and by the required accuracy. The order p of the segment polynomial directly impacts the computational complexity, the number of coefficients, and the data-path latency. For example, for a desired maximum approximation error, increasing the order p of the polynomial increases the number of coefficients per segment, increases the number of adders and multipliers, and increases the latency; however, the number of segments can also be reduced, which reduces the required coefficient memory size. Hence, the maximum acceptable approximation error and the computational complexity define a trade-off between the number 2^m of equal segments and the degree p of the polynomial.

To provide a quantitative foundation for the analysis of the design trade-offs, such as the size of the on-chip memories and the precision of coefficients and intermediate signals, we implemented both fixed-point and floating-point arithmetic and logical libraries in Mex-C [30]. These libraries include parameterizable modules, with variable exponent and mantissa bit-widths, that provide a flexible simulation environment for the bit-true comparison of approximated values of $f(x)$ with accurate function values. The minimum number of segments and the precision of the polynomials coefficients and intermediate signals were obtained through exhaustive simulation across the approximation interval $x \in [1, 2)$ at a precision of 2^{-23} to minimize the hardware requirements while guaranteeing accurate 23-bit results (i.e., achieving absolute errors of less than $1.192e^{-7}$). Minimizing the word length of the intermediate signals reduces the size of the logic blocks used in the polynomial evaluation data path. Minimizing the polynomial coefficient word lengths further reduces the size of on-chip memories and also the size of logic blocks.

Since the size of the required on-chip memory grows exponentially with m, minimizing m is crucial to minimizing the hardware requirements of the function implementation. Our exhaustive simulation starts by minimizing the value of m for a predefined polynomial order p so that the approximation is accurate to the target precision. In this step, all variables (i.e., the polynomial coefficients and the intermediate signals of the polynomial evaluation data path) are considered to have full precision. After the minimum value of m has been determined, in the second step, the word lengths of the intermediate signals and polynomial coefficients are jointly minimized such that the final error is still kept less than the 2^{-23} worst-case bound. Tables 1 and 2 present the optimized characteristics of the six elementary functions obtained using our bit-true exhaustive simulation-based approach assuming linear and quadratic interval polynomials, respectively. The row labeled "Data-path precision" gives the precision of the intermediate variables used in the arithmetic, logical, and routing modules. The precisions of the polynomial coefficients and data paths are denoted by (*WL*,*WF*), where *WL* and *WF* denote the word length and the fraction length of the variables, respectively. The accuracies yielded by such an optimized configuration are also shown. Note that the number of stored bits in coefficient LUTs is not identical with the precision of polynomial coefficients as the range of the coefficient values sometimes allows removal of the most significant bits. These implied constant bits are appended to the LUT contents when they are read in the polynomial data path.

TABLE 1
Implementation Characteristics Assuming Linear Polynomials

	x^{-1}	$x^{0.5}$	$x^{-0.5}$	$\log_2(x)$	$\ln(x)$	2^x
m	11	10	10	10	10	11
LUT size	$2^{11} \times 37$	$2^{10} \times 33$	$2^{10} \times 38$	$2^{10} \times 41$	$2^{10} \times 39$	$2^{11} \times 43$
Coefficient precision	$(26, 24)$	$(27, 25)$	$(27, 25)$	$(27, 25)$	$(27, 25)$	$(28, 25)$
Datapath precision	$(27, 25)$	$(27, 25)$	$(27, 25)$	$(29, 27)$	$(29, 27)$	$(29, 26)$
Maximum error	$1.188e^{-7}$	$8.455e^{-8}$	$1.069e^{-7}$	$1.158e^{-7}$	$9.610e^{-8}$	$1.026e^{-7}$

TABLE 2
Implementation Characteristics Assuming Quadratic Polynomials

	x^{-1}	$x^{0.5}$	$x^{-0.5}$	$\log_2(x)$	$\ln(x)$	2^x
m	7	5	6	6	6	7
LUT size	$2^7 \times 52$	$2^5 \times 54$	$2^6 \times 51$	$2^6 \times 58$	$2^6 \times 52$	$2^7 \times 57$
Coefficient precision	$(26, 24)$	$(27, 25)$	$(26, 24)$	$(27, 25)$	$(26, 24)$	$(28, 25)$
Datapath precision	$(29, 27)$	$(30, 28)$	$(30, 28)$	$(29, 27)$	$(29, 27)$	$(29, 26)$
Maximum error	$1.059e^{-7}$	$9.130e^{-8}$	$1.139e^{-7}$	$1.110e^{-7}$	$1.191e^{-7}$	$1.089e^{-7}$

Fig. 2 shows the data path of the fully pipelined first-order (a) and second-order (b) polynomial implementations using Horner's rule. The LUTs are addressed using the m most significant bits of the normalized mantissa. To minimize the required LUT size, the *Coefficient adjustment* block prepends any constant most significant bits of the polynomial coefficients to the values read from the LUTs. The data paths in Fig. 2 utilize small first-in first-out (FIFO) buffers to permit fully pipelined implementations which, after some fixed latency, generate one new result per clock cycle. The labels D_{mem}, D_{add}, and D_{mul} in Fig. 2 denote the corresponding latencies of the memory, adder, and multiplier, respectively. The formats (including the precisions) of the intermediate data-path signals in Fig. 2 are determined by the largest word lengths and fraction lengths that ensure perfect accuracy at the required precision for all six functions. The *Rounding* block transforms the approximated values into 23-bit rounded results in order to limit the maximum error to half a unit in the last place (ulp). The rounding step ensures that for a given function, the data path will always produce the same results as the other designs that have the same floating-point format.

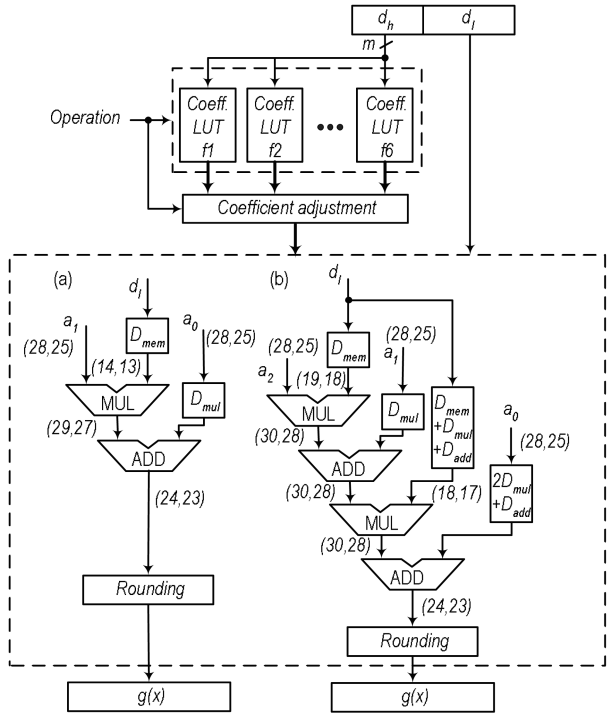

Fig. 2. Block diagram of the fully pipelined (a) first-order and (b) second-order polynomial implementations.

To further reduce the size of the arithmetic modules in the two data paths in Fig. 2, we normalized the value of the calculated polynomial coefficients as follows: Let $x = 1.d_1d_2 \ldots d_md_{m+1} \ldots d_n$, where $d_h = d_1d_2 \ldots d_m$ and $d_l = d_{m+1}d_{m+2} \ldots d_n$ denote the m most significant and $\ell = n - m$ least significant bits of $d = x - 1$, and the full fraction width is $n = m + \ell = 23$. Thus, x can be specified as $1 + \sum_{j=1}^{m} d_j 2^{-j} + \sum_{j=m+1}^{23} d_j 2^{-j}$. Since d_h is used to address the coefficient memory, d_l can be shifted left m bits, where the multiplication of d_l by 2^m is compensated for by scaling the coefficients of the polynomials.

As shown in Tables 1 and 2, when using a quadratic polynomial, a smaller memory is sufficient at the expense of one additional adder and multiplier compared with the linear polynomial implementation (see Fig. 2). Moreover, the latency of the quadratic implementation is longer than the linear implementation. To avoid performing the two serial multiplications and additions used in the conventional implementation of a quadratic polynomial, various techniques have been proposed for a faster evaluation with a significant reduction in latency [18], [19], [31]. For example, the method in [18] uses specially designed multipliers and redundant arithmetic and the design in [19] uses a specialized squaring unit, multioperand adders, and also specially designed multipliers. Our bit-true simulation environment and our design methodology are independent of the implementation details and can be efficiently used to optimize any polynomial-based function approximation data paths. Our strategy has been to prefer generic designs so as to ensure portability and validity.

4 Comparative Analysis

Our design methodology enables compact and versatile implementations of shared-data-path multifunction floating-point units on custom VLSI and configurable hardware.

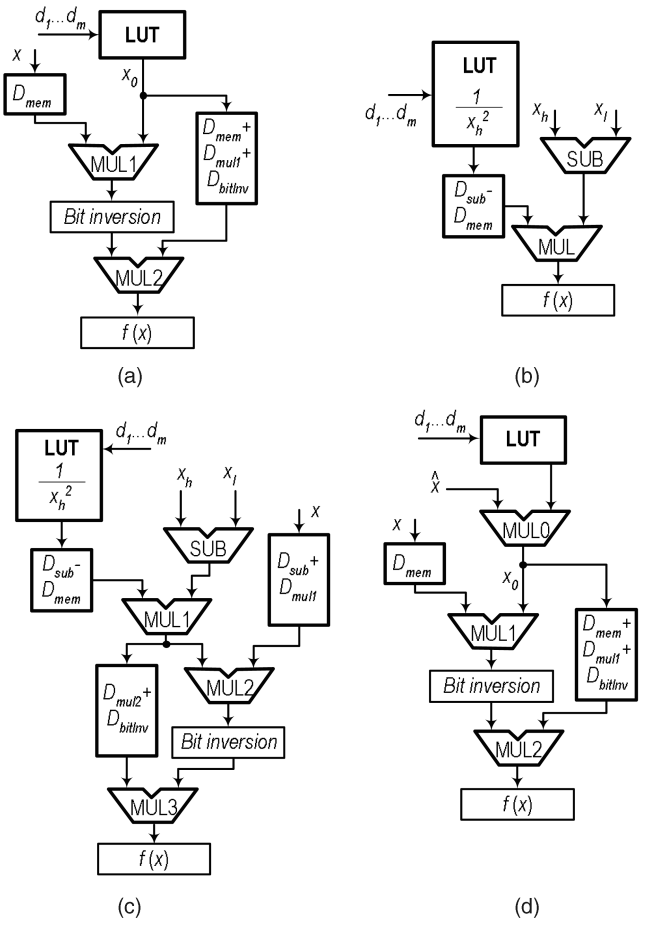

Fig. 3. Four fully pipelined data paths of comparable multiplicative reciprocal implementations: (a) NR+REC, (b) HUNG+REC [14], (c) JEONG+REC [15], and (d) NR+REC+ITO [33].

In this section, we present the implementation results for the proposed architecture on a contemporary Xilinx Virtex4 field-programmable gate array (FPGA) [32]. We compare in detail our designs with other high-throughput designs with respect to the reciprocal and reciprocal square root functions. Comparisons with implementations of the other four operations would produce similar observations.

The comparison is organized in two parts. We start by comparing with designs that use separately designed data paths for implementing reciprocal and (reciprocal) square root. One widely used technique for approximating elementary functions is the Newton-Raphson (NR) method [1]. This method adopts an initial approximation and improves upon it by a converging algorithm. Approximating the reciprocal $1/x$ using the quadratically converging NR recurrence can be derived from the Taylor series expansion as $x_{i+1} = x_i(2 - x\, x_i)$. Fig. 3a shows the fully pipelined data path of the resulting NR reciprocal NR+REC implementation when no iterations are required to achieve the desired accuracy. Here, 2^m sufficiently accurate starting reciprocal estimates between $(0.5, 1]$ are precomputed and stored in an LUT to obtain a more accurate second estimate. In this data path, the difference $2 - dx_i$ is replaced with a simple bit inversion of dx_i, performed by the *Bit inversion* block. D_{mulk} and D_{bitInv} denote the latencies of multiplier k and the Bit inversion block, respectively, which are used to define FIFO latencies.

Hung et al. [14] also use a Taylor series expansion for approximating the reciprocal operation in which $1/x$ is obtained as $(x_h - x_l)/x_h^2$ where x_h denotes the $(m+1)$st most significant bits of x (i.e., $x_h = 2^0 + 2^{-1}d_1 + \ldots + 2^{-m}d_m$ and $1 \le x_h \le 2 - 2^{-m}$) and $x_l = x - x_h$ (i.e., $0 \le x_l < 2^{-m}$). Fig. 3b shows the corresponding data path HUNG+REC that approximates the reciprocal of x using only one subtractor, one multiplier, and one on-chip memory to store $1/x_h^2$ and a small FIFO (of latency $D_{sub} - D_{mem}$) to support a fully pipelined implementation. An improved scheme that requires a smaller memory is proposed by Jeong et al. [15]. This scheme is based on the modified Taylor expansion and approximates $1/x$ as $A(2 - Ax)$ where $A = (x_h - x_l)/x_h^2$. The data path for Jeong's approach (JEONG+REC), shown in Fig. 3c, is essentially the same data path as the NR+REC data path with A as the initial approximation.

Ito et al. [33] proposed an efficient initial approximation scheme using a piecewise linear approximation for the reciprocal operation. This initial approximation can be combined with a multiplicative iterative technique, such as the NR algorithm. This approximation was further improved to remove the addition in the linear approximation by slightly modifying the operands used in the initial reciprocal approximation. This modification also reduces the LUT size as only one coefficient, instead of two, has to be stored. Hence, Ito's data path NR+REC+ITO, as shown in Fig. 3d, uses the same NR+REC data path (with possibly different precisions for the intermediate signals, as shown later) and utilizes one additional multiplier for the modified initial linear approximation.

Table 3 summarizes the implementation characteristics of the four comparable reciprocal implementations on a Xilinx Virtex4 LX200FF1513-11 FPGA. The size of the LUTs and the precision of intermediate signals were optimized using bit-true exhaustive simulation. The data paths all

TABLE 3
Characteristics of the Optimized Data Paths for Reciprocal on a Xilinx Virtex4 LX200FF1513-11 FPGA

Technique	NR+REC	HUNG+REC [14]	JEONG+REC [15]	NR+REC+ITO [33]
Adders	-	$1(24, 24)$	$1(24, 24)$	-
Multipliers	$1(24, 15), 1(26, 15)$	$1(26, 24)$	$1(24, 14), 1(31, 24), 1(31, 31)$	$1(24, 14), 1(28, 24), 1(28, 28)$
Memory	$1(2^{12} \times 15)$	$1(2^{12} \times 26)$	$1(2^6 \times 14)$	$1(2^6 \times 13)$
Configurable slices	$155(< 1\%)$	$243(< 1\%)$	$435(< 1\%)$	$230(< 1\%)$
Block memories	$4(1\%)$	$6(1\%)$	$1(< 1\%)$	$1(< 1\%)$
DSP48 slices	$4(4\%)$	$4(4\%)$	$10(10\%)$	$6(6\%)$
Latency (CLKs)	10	12	23	14
Latency (ns)	33	38	74	45
Throughput (Ms/sec)	301	308	307	304

used pipelined arithmetic modules to maximize performance. The row labeled "Latency (CLKs)" gives the number of clock cycles required to obtain a result after applying the corresponding input to a data path. An entry of the form $k(c_1, \ldots, c_b)$ in Table 3 denotes k instances of the adder or multiplier module, where the input operands are $c_1, \ldots, c_b$ bits wide, respectively. As shown in Table 3, the HUNG+ REC scheme requires the most memory while the JEONG+REC method requires the greatest number of multipliers. The NR+REC+ITO data path provides a good balance between resource utilization, latency, and throughput. Note that the implementation details may vary as a designer may choose to implement the arithmetic modules using configurable slices and/or the dedicated resources now available on many FPGAs. Similarly, one may choose to implement relatively small LUTs on distributed memories and larger ones on dedicated memory blocks.

For approximating $1/\sqrt{x}$, the NR algorithm provides the iterative equation $x_{i+1} = 0.5 \times x_i(3 - x\, x_i^2)$ [2]. The corresponding fully pipelined data path NR+ISQT, when no iterations are required to obtain the desired accuracy, is shown in Fig. 4a. Note that subtracting from three was approximated by bit conversion since $3 - x\, x_i^2 = 1 + \ (2 - x\, x_i^2)$ and the term $2 - x\, x_i^2$ corresponds to a two's complement, which can in turn be approximated by bit inversion. Another efficient technique for estimating $1/\sqrt{d_0}$ is to use the binomial series expansion, also called the Goldschmidt (GS) algorithm [4]. This scheme starts with an initial approximation Y_0 to $1/\sqrt{d_0}$. Then, the product $g_n = Y_0 Y_1 \ldots Y_n$ will approach $1/\sqrt{d_0}$. Each GS square root iteration involves computing $d_j = d_{j-1} Y_{j-1}^2$ and $Y_j = 1 + 0.5(1 - d_j)$, where the reciprocal square root approximation is updated by $g_j = g_{j-1} Y_j$. Fig. 4b shows the fully pipelined data path GS+ISQT of the GS reciprocal square root algorithm when no iterations are required to obtain the desired accuracy. The *SHR+1* block implements a logical shift right and increment operation. Using the same definitions for x_h and x_l, Hung et al. also proposed the $\sqrt{x} \approx \frac{x(x_h - x_l/2)}{x_h^{1.5}}$ approximation for square root [14]. The corresponding data path HUNG+SQT is shown in Fig. 4c. Fig. 4d shows Ito's approach (NR+ISQT+ ITO) for estimating the reciprocal square root. It uses the same NR+ISQT data path but requires one additional multiplier for the modified initial linear approximation.

Table 4 summarizes the characteristics of the optimized data paths for high-throughput (reciprocal) square root implementations on a Xilinx Virtex4 LX200FF1513-11 FPGA. As shown in Table 4, the HUNG+SQT scheme requires the largest memory and the NR+ISQT+ITO method requires the fewest memories and the greatest number of multipliers. Note that the characteristics presented in Tables 3 and 4 were obtained assuming that the results had to be correct to one ulp (i.e., less than one ulp deviation from the infinitely precise result for an arbitrarily precise real input).

The second group of designs uses a unified architecture to implement multiple elementary functions [18], [19], [20], [22]. Piñeiro et al. [18] proposed an enhanced minmax quadratic approximation and an optimized evaluation of the second-order polynomial. The coefficients were approximated using the symbolic algebra system Maple [34]. The work by

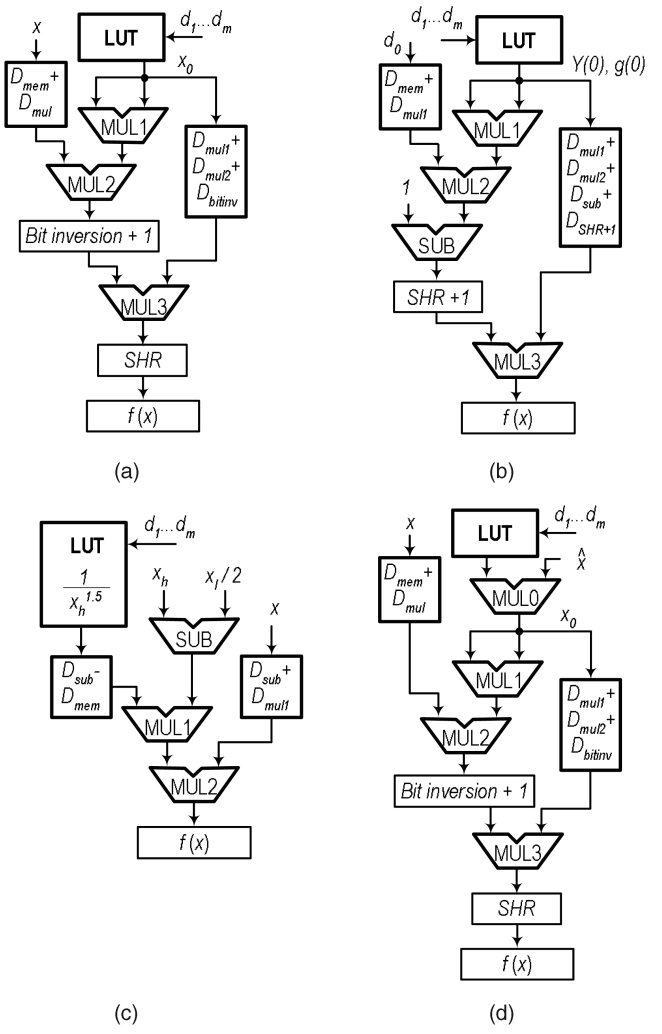

Fig. 4. Four fully pipelined data paths of comparable multiplicative (reciprocal) square root implementations: (a) NR+ISQT, (b) GS+ISQT, (c) HUNG+SQT, and (d) NR+ISQT+ITO.

Schulte and Swartzlander [19] also used a uniform segmentation and polynomial approximation (SCH+REC+ISQT) in which the coefficients for each segment were determined using a Chebyshev series approximation. The polynomial terms are generated in parallel and summed using a multioperand adder. Ercegovac et al. [20] proposed a three-step method (ERC+REC+ ISQT) based on argument reduction, evaluation using a few Taylor series terms, and then postprocessing. In the argument reduction step the input is scaled to be close to one, and after evaluation of the function using a few Taylor series terms, the result is postprocessed.

In Schulte's scheme, the size of the required memory (see Table 2 in [19]) using a linear approximation that produces exactly rounded results is actually too large (i.e., $2^{18} \times 57$) to fit on a currently available FPGA. Thus, we present the characteristics of this method utilizing a quadratic polynomial approximation. Table 5 presents the characteristics of the data paths for the reciprocal and reciprocal square root functions implemented using Ercegovac's scheme, the quadratic polynomial using Schulte's approach, and our linear and quadratic polynomial implementations. As shown in Table 5, when using quadratic polynomials, a smaller memory is required at the expense

TABLE 4
Characteristics of the Optimized Data Paths for High-Throughput Implementations of (Reciprocal) Square Root on a Xilinx Virtex4 LX200FF1513-11 FPGA

Technique	NR+ISQT	GS+ISQT	HUNG+SQT [14]	NR+ISQT+ITO [33]
Adders	–	$1(26,26)$	$1(24,24)$	–
Multipliers	$1(15,15), 1(24,29)$ $1(29,15)$	$1(15,15), 1(24,26),$ $1(26,15)$	$2(27,24)$	$1(24,13), 2(27,27),$ $1(27,24)$
Memory	$1(2^{11} \times 15)$	$1(2^{12} \times 15)$	$1(2^{12} \times 27)$	$1(2^{6} \times 13)$
Configurable slices	$240(< 1\%)$	$308(< 1\%)$	$377(< 1\%)$	$295(< 1\%)$
Block memories	$2(< 1\%)$	$4(< 1\%)$	$6(1\%)$	$1(< 1\%)$
DSP48 slices	$7(7\%)$	$7(7\%)$	$8(8\%)$	$9(9\%)$
Latency (CLKs)	15	21	18	19
Latency (ns)	49	68	58	62
Throughput (Ms/sec)	307	304	308	306

TABLE 5
Characteristics of the Optimized Data Paths for Both Reciprocal and Reciprocal Square Root on a Xilinx Virtex4 LX200FF1513-11 FPGA

Technique	ERC+REC+ISQT [20]	SCH+REC+ISQT [19]	Proposed Linear	Proposed Quadratic
Adders	$3(14,14), 1(14,7),$ $2(14,21)$	$1(40,33,21)$	$1(29,28)$	$2(30,28)$
Multipliers	$1(8,28), 3(7,7),$ $1(22,23), 1(22,8)$	$1(19,16), 1(31,12)$ $1(12,12)$	$1(28,14)$	$1(28,19), 1(30,18)$
Memory	$1(2^{7} \times 8), 1(2^{7} \times 28)$	$1(2^{13} \times 90)$	$1(2^{11} \times 37), 1(2^{10} \times 38)$	$1(2^{7} \times 52), 1(2^{6} \times 51)$
Configurable slices	$540(< 1\%)$	$384(< 1\%)$	$117(< 1\%)$	$310(< 1\%)$
Block memories	$2(< 1\%)$	$40(< 11\%)$	$8(2\%)$	$2(< 1\%)$
DSP48 slices	$8(8\%)$	$5(5\%)$	$2(2\%)$	$4(4\%)$
Latency (CLKs)	29	17	12	18
Latency (ns)	95	56	37	55
Throughput (Ms/sec)	307	306	330	328
Accuracy	< 1 ulp	Exactly rounded	Exactly rounded	Exactly rounded

of longer latency compared to the linear polynomial implementation. The implementation results in Table 5 show that our two shared data-path designs require significantly fewer adders, multipliers, and configurable slices than the two other proposed techniques. Our quadratic polynomial approximation that produces exact results also requires 1.4 times fewer memory bits for commonly supported functions than the unified designs in [18] that produce results with an accuracy of one ulp (see Table 2 in [18]). Even though the design in [20] requires fewer memory bits than the other three designs in Table 5, it uses the greatest number of configurable slices and dedicated multipliers and it also has the longest latency.

The sum of the required resources for implementing just the reciprocal and square root using the multiplicative techniques, whose characteristics are shown in Tables 3 and 4, is also significantly more than the resources required by our two proposed schemes. While each function requires a separate memory to store the polynomial coefficients, sharing a single shared data path among all functions produces an especially compact implementation. Note that our proposed linear and quadratic designs both support more than just the reciprocal and square root functions and can be easily extended to implement other elementary functions, such as exponentiation and trigonometric functions.

5 Conclusions

This paper described an efficient scheme, based on uniform domain segmentation and curve fitting, that yields especially compact and accurate implementations of various widely used elementary functions to single-precision floating-point accuracy. The same data path can be used to produce 23-bit accurate results for the six elementary functions x^{-1}, $x^{0.5}$, $x^{-0.5}$, $\log(x)$, $ln(x)$, and 2^{x} over the interval $x \in [1, 2)$. The proposed approach is applicable to many other nonlinear functions such as trigonometric functions and exponentiation.

We used parameterized fixed-point and floating-point routines to optimize the size of the memory blocks and minimize the word lengths of the intermediate variables. Exhaustive bit-true simulations confirmed that the proposed designs provide the 23-bit accuracy required for single-precision floating-point arithmetic. Our implementation results for a Xilinx FPGA show that our proposed architecture requires significantly fewer logic resources (e.g., for adders, registers, and routing), and uses fewer dedicated multipliers than previously proposed designs. Our scheme also provides higher throughput and smaller latency than other proposed approaches. One may choose either the proposed linear approximation (for shorter latency) or the proposed quadratic implementation (for smaller memory space) based on the available configurable logic slices and dedicated memory blocks. The new unified architecture for implementing multiple elementary functions in single precision should be attractive for applications that require both high performance and flexible floating-point unit designs.

References

[1] M.J. Flynn, "On Division by Functional Iteration," *IEEE Trans. Computers*, vol. 19, no. 8, pp. 702-706, Aug. 1970.
[2] M.D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2004.

[3] M.D. Ercegovac and T. Lang, *Division and Square Root: Digit Recurrence Algorithms and Implementations.* Kluwer Academic, 1994.
[4] M.D. Ercegovacet et al., "Improving Goldschmidt Division, Square Root, and Square Root Reciprocal," *IEEE Trans. Computers,* vol. 49, no. 7, pp. 759-763, July 2000.
[5] E.M. Schwarz and M.J. Flynn, "Hardware Starting Approximation for the Square Root Operation," *Proc. IEEE Symp. Computer Arithmetic,* pp. 103-111, 1993.
[6] W.F. Wong and E. Goto, "Fast Evaluation of the Elementary Functions in Single Precision," *IEEE Trans. Computers,* vol. 44, no. 3, pp. 453-457, Mar. 1995.
[7] D. Das Sarma and D.W. Matula, "Faithful Bipartite ROM Reciprocal Tables," *Proc. IEEE Symp. Computer Arithmetic,* pp. 17-28, 1995.
[8] J.-M. Muller, "A Few Results on Table-Based Methods," *Reliable Computing,* vol. 5, no. 3, pp. 279-288, 1999.
[9] M.J. Schulte and J.E. Stine, "Approximating Elementary Functions with Symmetric Bipartite Tables," *IEEE Trans. Computers,* vol. 48, no. 8, pp. 842-847, Aug. 1999.
[10] 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.
[11] P.T.P. Tang, "Table-Lookup Algorithms for Elementary Functions and Their Error Analysis," *Proc. IEEE Symp. Computer Arithmetic,* pp. 232-236, 1991.
[12] D. Wong and M.J. Flynn, "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations," *IEEE Trans. Computers,* vol. 41, no. 8, pp. 981-995, Aug. 1992.
[13] A.A. Liddicoat and M.J. Flynn, "High-Performance Floating Point Divide," *Proc. Euromicro Symp. Digital Systems Design,* pp. 354-361, 2001.
[14] P. Hung, H. Fahmy, O. Mencer, and M.J. Flynn, "Fast Division Algorithm with a Small Lookup Table," *Proc. Asilomar Conf. Signals, Systems, and Computers,* pp. 1465-1468, 1999.
[15] J.-C. Jeong et al., "A Cost-Effective Pipelined Divider with a Small Lookup Table," *IEEE Trans. Computers,* vol. 53, no. 4, pp. 489-495, Apr. 2004.
[16] N. Takagi, "Powering by a Table Look-Up and a Multiplication with Operand Modification," *IEEE Trans. Computers,* vol. 47, no. 11, pp. 1216-1222, Nov. 1998.
[17] J.-A. Piñeiro and J.D. Bruguera, and J.-M. Muller, "Faithful Powering Computation Using Table Look-Up and a Fused Accumulation Tree," *Proc. IEEE Symp. Computer Arithmetic,* pp. 40-47, 2001.
[18] J.-A. Piñeiro, S.F. Oberman, J.-M. Muller, and J.D. Bruguera, "High-Speed Function Approximation Using a Minimax Quadratic Interpolator," *IEEE Trans. Computers,* vol. 54, no. 3, pp. 304-318, Mar. 2005.
[19] M.J. Schulte and E.E. Swartzlander, "Hardware Designs for Exactly Rounded Elementary Functions," *IEEE Trans. Computers,* vol. 43, no. 8, pp. 964-973, Aug. 1994.
[20] M.D. Ercegovac, T. Lang, J.-M. Muller, and A. Tisserand, "Reciprocation, Square Root, Inverse Square Root, and Some Elementary Functions Using Small Multipliers," *IEEE Trans. Computers,* vol. 49, no. 7, pp. 628-637, July 2000.
[21] D.D. Sarma and D.W. Matula, "Faithful Interpolation in Reciprocal Tables," *Proc. IEEE Symp. Computer Arithmetic,* pp. 82-91, 1997.
[22] V.K. Jain, S.A. Wadekar, and L. Lin, "A Universal Nonlinear Component and its Application to WSI," *IEEE Trans. Components, Hybrids, and Manufacturing Technology,* vol. 16, no. 7, pp. 656-664, Nov. 1993.
[23] J. Cao and B. Wei, "High-Performance Hardware for Function Generation," *Proc. IEEE Symp. Computer Arithmetic,* pp. 184-188, 1997.
[24] P. Soderquist and M. Leeser, "Division and Square Root: Choosing the Right Implementation," *IEEE Micro,* vol. 17, no. 4, pp. 56-66, July/Aug. 1997.
[25] J.-M. Muller, *Elementary Functions. Algorithms and Implementation.* Birkhauser, 1997.
[26] A. Alimohammad, S.F. Fard, B.F. Cockburn, and C. Schlegel, "A Compact and Accurate Gaussian Variate Generator," *IEEE Trans. Very Large Scale Integration Systems,* vol. 16, no. 5, pp. 517-527, May 2008.
[27] N.R. Draper and H. Smith, *Applied Regression Analysis.* John Wiley & Sons, Inc., 1998.
[28] G.H. Golub and C.F.V. Loan, *Matrix Computations.* Johns Hopkins Univ. Press, 1996.
[29] K. Levenberg, "A Method for the Solution of Certain Non-Linear Problems in Least Squares," *Quarterly of Applied Math.,* vol. 2, pp. 164-168, 1994.
[30] *MATLAB 7 C and Fortran API Reference,* The Mathworks, Inc., 2008.
[31] J. Detrey and F.D. Dinechin, "Second Order Function Approximation Using a Single Multiplication on FPGAs," *Proc. IEEE Int'l Conf. Field Programmable Logic and Applications (FPL),* pp. 221-230, 2004.
[32] *Virtex-4 User Guide,* Xilinx, Inc., June 2008.
[33] M. Ito, N. Takagi, and S. Yajima, "Efficient Initial Approximation for Multiplicative Division and Square Root by a Multiplication with Operand Modification," *IEEE Trans. Computers,* vol. 46, no. 4, pp. 495-498, Apr. 1997.
[34] *Maple 8 Programming Guide,* Waterloo Maple, Inc., 2002.

Amirhossein Alimohammad (S'01-M'07) received the BSc degree from the University of Isfahan, Iran, the MSc degree from the University of Tehran, Iran, and the PhD degree from the University of Alberta, Edmonton, Canada. From 1998 to 2000, he was with Informatics Services Corporation, Tehran, Iran, while he was a PhD student at the Sharif University of Technology, Tehran. In 2001, he was with the Institute of Microelectronics, University of Ulm, Germany, Atmel Germany GmbH, and Get2Chip GmbH, Munich, Germany. From 2007 to 2008, he was a postdoctoral fellow at the Department of Electrical and Computer Engineering, University of Alberta. Currently, he is the president of Ukalta Engineering, which he cofounded in 2009. His research interests include reconfigurable architectures, VLSI design and test, wireless communications, and signal processing. He is a member of the IEEE.

Saeed Fouladi Fard (S'03) received the BSc degree in electrical engineering from the Faculty of Communications, Tehran, Iran, in 2000, the MSc degree in electrical engineering from the University of Tehran, Iran, in 2003, and the PhD degree in electrical and computer engineering from the University of Alberta, Edmonton, Canada, in 2009. He is currently the chief technology officer at Ukalta Engineering, Edmonton, Alberta, Canada. His general research interests include signal processing, communications, reconfigurable computing, and VLSI design and testing. His current research focuses on multiple antenna transceivers, fading simulation, and efficient hardware computation techniques.

Bruce F. Cockburn (S'86-M'90) received the BSc degree in engineering physics in 1981 from Queen's University at Kingston, in Canada. In 1985 and 1990, he received the MMath and PhD degrees, respectively, in computer science from the University of Waterloo. His doctoral thesis considered the design and analysis of provably optimal test algorithms for classes of faults in semiconductor memories. He is currently an associate professor in the Department of Electrical and Computer Engineering at the University of Alberta in Edmonton, Canada. From 1981 to 1983, he worked as a test engineer and a software designer at Mitel Corporation, Kanata, Ontario, Canada. His research interests are diverse and include VLSI design and test, multilevel memory and novel memory hierarchies, parallel signal processing algorithms and architectures, iterative decoder architectures, and FPGA-based hardware accelerators for communications system characterization. He is a member of the IEEE, the IEEE Computer Society, the IEEE Communications Society, the IEEE Solid-State Circuits Society, the IEEE Signal Processing Society, and the Association for Computing Machinery. He is a registered professional engineer in the Province of Alberta, Canada.