A Medium-Grain Reconfigurable Architecture for DSP: VLSI Design, Benchmark Mapping, and Performance

Mitchell J. Myjak, *Member, IEEE*, and José G. Delgado-Frias, *Senior Member, IEEE*

***Abstract*—Reconfigurable hardware has become a well-accepted option for implementing digital signal processing (DSP). Traditional devices such as field-programmable gate arrays offer good fine-grain flexibility. More recent coarse-grain reconfigurable architectures are optimized for word-length computations. We have developed a medium-grain reconfigurable architecture that combines the advantages of both approaches. Modules such as multipliers and adders are mapped onto blocks of 4-bit cells. Each cell contains a matrix of lookup tables that either implement mathematics functions or a random-access memory. A hierarchical interconnection network supports data transfer within and between modules. We have created software tools that allow users to map algorithms onto the reconfigurable platform. This paper analyzes the implementation of several common benchmarks, ranging from floating-point arithmetic to a radix-4 fast fourier transform. The results are compared to contemporary DSP hardware.**

***Index Terms*—Digital signal processing (DSP), floating-point arithmetic, medium-grain reconfigurable hardware, synthesis tools.**

I. Introduction

MANY applications rely on digital signal processing (DSP) to achieve their functionality. However, these computationally-intensive algorithms place great demands on the processing power of the underlying hardware. As technology continues to advance, reconfigurable hardware has become a viable option for implementing DSP [1]. Not only do these devices enable rapid development and prototyping, but the configuration can be changed at any time—even after deployment.

Using reconfigurable hardware for DSP requires fundamental tradeoffs between performance and flexibility. Traditional fine-grain devices such as field-programmable gate arrays (FPGAs) can implement arbitrary logic equations. However, implementing a multiplier on a fine-grain device requires a large number of cells. The resulting interconnection delays hinder the resulting performance. For this reason, many FPGAs embed dedicated multipliers within the reconfigurable fabric. The Xilinx Virtex-4, for example, contains special XtremeDSP slices that can perform 18-bit multiplication and 48-bit addition [2].

Recently, researchers have proposed new reconfigurable architectures in which each cell performs 16-bit or 32-bit operations [3]. These coarse-grain devices achieve high performance for DSP. Some examples are the 1-D RaPiD array [4], the 2-D KressArray [5], and the heterogeneous Pleiades [6] and MONTIUM [7] architectures that combine fine-grain and coarse-grain components. On the commercial front, the PACT XPP architecture contains one or more arrays of processing elements, each of which holds an arithmetic logic unit (ALU) or a memory [8]. The Adapt2000 ACM from QuickSilver Technology uses a hierarchy of heterogeneous nodes; different types of nodes can implement 32-bit binary arithmetic, bit-oriented operations, general-purpose code, and memory control [9].

As a third alternative, medium-grain reconfigurable hardware attempts to balance performance and flexibility. Each cell works with 4- or 8-bit data, so word-length modules require a group of cells. However, cells typically support a wide variety of operations. Examples of proposed architectures include the hexagonal CHESS array [10], the rectilinear PipeRench [11] and DReAM [12] architectures, and the Elixent D-Fabrix that uses 4-bit cells and switchboxes [13]. In previous work, we have developed a novel medium-grain reconfigurable architecture that strives for efficient circuit-level implementation [14], [15]. Each cell can perform binary arithmetic or memory operations.

Mapping DSP onto reconfigurable hardware requires sophisticated software tools. Most commercial platforms support a wide range of features, including design synthesis, timing analysis, and circuit simulations. As an initial step, we have created software tools that allow users to map algorithms by hand. A simulator is provided to verify the designs. We have used the software to implement several benchmarks and evaluate the performance of the medium-grain architecture. This analysis forms the main focus of this paper. We have also presented initial results in [16] and [17].

Section II summarizes the VLSI design of the medium-grain architecture. Section III describes the software tools used to implement and test algorithms. Section IV shows the basic modules used for fixed-point and floating-point arithmetic. Section V summarizes the implementation of the selected

Manuscript received April 27, 2006; revised June 29, 2007. The work of M. Myjak was supported by an appointment to the U.S. Department of Homeland Security (DHS Scholarship and Fellowship Program, administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and DHS. ORISE is managed by Oak Ridge Associated Universities (ORAU) under DOE Contract DE-AC05-06OR23100. This research was supported in part by the Boeing Endowed Chair at the School of Electrical Engineering and Computer Science, Washington State University.

M. Myjak was with the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752 USA. He is now with Pacific Northwest National Laboratory, Richland, WA 99352 USA (e-mail: mitchell.myjak@pnl.gov).

J. G. Delgado–Frias is with the School of Electrica Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752 USA (e-mail: jdelgado@eecs.wsu.edu).

Digital Object Identifier 10.1109/TVLSI.2007.912080

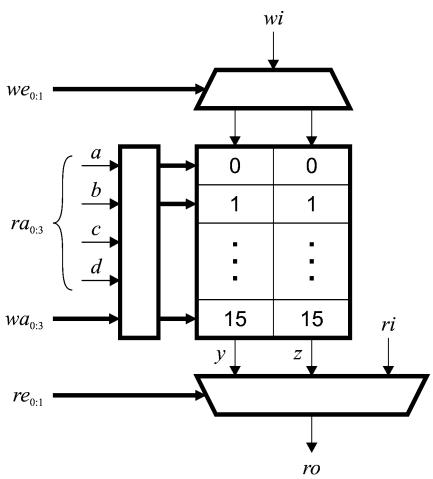

Fig. 1. Functional diagram of element.

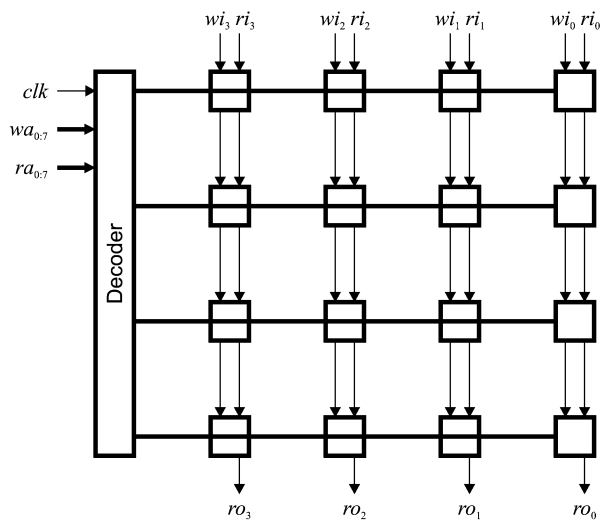

Fig. 2. Cell in memory mode.

benchmarks and gives the estimated execution times. Section VI compares the performance and features of the proposed architecture with digital signal processors and other types of reconfigurable hardware. Finally, Section VII concludes this paper.

II. Hardware Architecture

Our medium-grain architecture uses an innovative two-level approach to combine high performance with high flexibility. The device as a whole contains an array of 4-bit cells. Each cell, in turn, consists of a 4×4 matrix of lookup tables, or "elements." The lookup tables can assume only two structures: one for memory operations, and the other for binary arithmetic [14]. To implement DSP, individual cells are grouped into word-length modules, such as multipliers, adders, and memory units. These modules are then connected together into entire algorithms. A dual interconnection structure optimizes data transfer both within and between modules [16]. Typical devices might contain a 32×32 or 64×64 array of cells.

A key difference between the proposed architecture and fine-grain devices is the interconnection network. In an FPGA, each logic block can connect to any other logic block via a rich set of local and global lines. In our design, elements within a cell can only connect to neighboring elements in two ways. Similarly, each cell can only connect to neighboring cells and a hierarchical routing backbone. This optimized design offers the advantages of higher performance and lower power consumption. We demonstrate in this paper that the resulting medium-grain architecture retains sufficient flexibility to implement DSP.

In this section, we describe the VLSI design of the elements, cells, and interconnection network. We also give circuit simulations that validate the performance of the architecture. More information about the VLSI design appears in [15].

A. Elements

Fig. 1 depicts a functional diagram of an element. The component behaves as a 32-bit random-access memory (RAM) with separate read and write ports. This feature allows the cell to read from one address and write to another address simultaneously. The memory is divided into two banks of 16 latches, numbered 0 to 15 in Fig. 1. For read operations, address $ra_{0:3}$ selects one latch in each bank, and read enable $re_{0:1}$ determines which latch should drive the output ro. If neither bank is enabled, the element connects input ri to this output. Write operations proceed in a similar manner: address $wa_{0:3}$ selects one latch in each bank, and write enable $we_{0:1}$ determines which bank should receive the input wi.

Mathematics functions use the element as a four-input, two-output lookup table. The four inputs are placed onto the bits of $ra_{0:3}$. We also refer to these bits as a, b, c, and d. The outputs of the two selected latches drive outputs y and z, which connect to inputs c and d of neighboring elements.

The element may use a variety of circuit styles to implement this functionality [15]. These designs must optimize the read datapath to achieve high performance for mathematics computations. One simple approach divides the datapath into a series of n-transistors controlled by a CMOS decoder. The complementary outputs of the selected latch propagate through the n-transistors toward y and z. Weak cross-coupled p-transistors restore the rail-to-rail voltage swing. This approach has low latency and does not require internal clocking.

B. Cells

As stated previously, each cell uses a matrix of elements to perform memory and mathematics operations. This approach provides fine-grain flexibility without the large interconnection overhead of FPGAs, since the cell can only assume two structures. In memory mode, shown in Fig. 2, the elements implement a 128 × 4 bit RAM. Each element manages a 32 × 1-bit portion of the memory. Input $ra_{0:7}$ specifies the 7-bit read address, with ra_7 serving as the read enable. Similarly, $wa_{0:7}$ specifies the write address and write enable. Lines $wi_{0:3}$ and $ro_{0:3}$ define the input data and output data, respectively. Finally, $ri_{0:3}$ specifies the default value of $ro_{0:3}$ when the read enable is off.

In mathematics mode, shown in Fig. 3, the elements are configured to execute 4-bit binary arithmetic. Each element now acts as a 16 × 2-bit lookup table. The cell broadcasts inputs

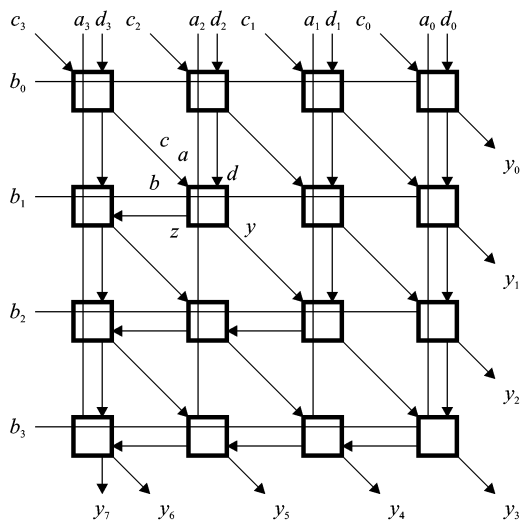

Fig. 3. Cell in mathematics mode.

$a_{0:3}$ and $b_{0:3}$ across the rows and columns of the structure. Inputs $c_{0:3}$ and $d_{0:3}$ are passed to the four elements in the top row. The cell collects the output $y_{0:7}$ from the seven elements on the right and bottom. The critical path through the structure consists of seven elements.

Mathematics mode is optimized for the powerful multiply-accumulate (MAC)

$$y_{7:0} = (a_{3:0} \times b_{3:0}) + c_{3:0} + d_{3:0}. \quad (1)$$

Here, each element evaluates the 1-bit MAC

$$(2z + y) = (a \times b) + c + d. \quad (2)$$

The cell can implement many other functions as well, including addition, subtraction, bit shifting, and control logic. The fine-grain flexibility of the elements permits cells to work with unsigned or two's-complement data.

Alternative designs of the cell are also possible, including a bit-serial design that uses five elements [18], and a super-pipelined design that achieves very high throughput [19].

C. Interconnection Network

The interconnection scheme used in the medium-grain architecture facilitates the mapping of word-length modules such as multipliers, adders, and memory units. Fig. 4 depicts the local network. A mesh of 4-bit busses running horizontally, vertically, and diagonally allows cells to exchange data with their eight neighbors. Crossbar switches connect the inputs and outputs of the cell to the local network. The local mesh contains pipeline latches that improve throughput by exploiting the data-parallel nature of DSP. The architecture dedicates one clock cycle for all cell computations, and one clock cycle for local communication between cells. Additional pipeline registers can be placed in the datapath if required.

Fig. 5 illustrates the global network, which resembles an H-tree. This structure can transfer data efficiently between any two cells. Each level of the tree contains four input busses and four output busses. The number of bits per bus starts at 4 bits

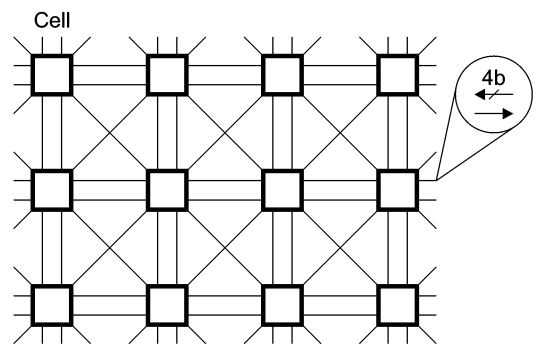

Fig. 4. Cells with local interconnection network.

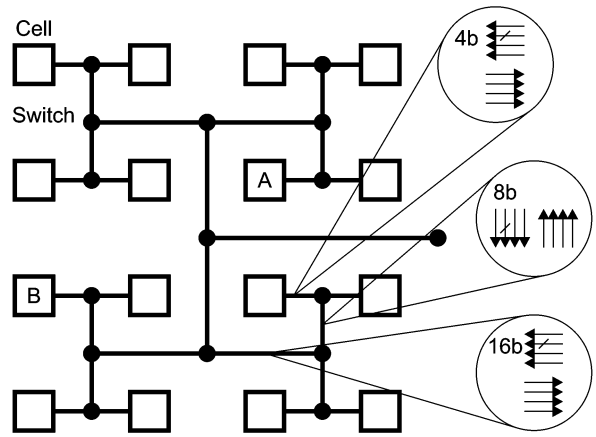

Fig. 5. Global network.

and doubles at every level. (To save area, one could limit the bandwidth to a predetermined value.) Switches in the global network route data in word units, resulting in lower complexity and configuration overhead.

Like the local mesh, the global H-tree contains pipeline latches to improve throughput. The latency between any two cells equals half the number of busses traversed. Hence, cells A and B in Fig. 5 are separated by four clock cycles. This architecture simplifies the mapping process, as the outputs of a module can be collected onto a single bus and routed to the inputs of another module. All 4-bit portions of the data incur the same latency over the H-tree.

D. Simulations and Prototype

We have performed layout simulations of the basic cell in 180-nm CMOS technology with all parasitic capacitances extracted [15], [17]. The simulations show that the cell can be pipelined at a clock rate of 267 MHz. We have also created layouts of the crossbar switches used in the interconnection network [20]. According to simulations, the latency through the switches will not constrain the overall clock rate. Given the deep pipelining of the interconnection network, we do not anticipate any problems with maintaining the clock rate across a fully populated chip.

We have also performed circuit simulations of the cell in a more aggressive 90-nm technology. In this case, the design runs at speeds of 1 GHz without accounting for parasitic capacitances. We estimate that the clock rate would decrease to

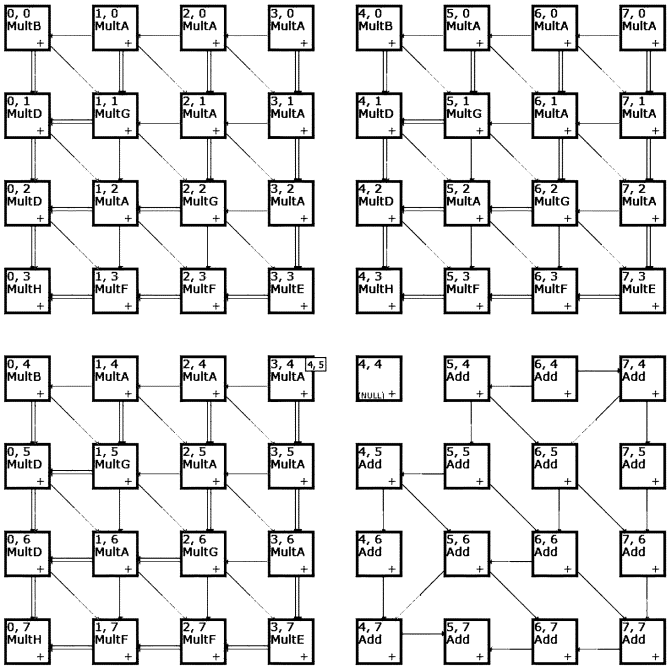

Fig. 6. Screenshot of software tools.

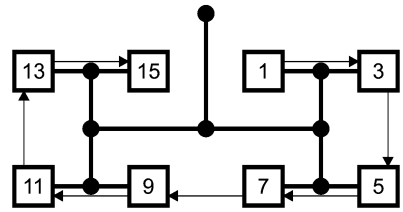

Fig. 7. 32-bit fixed-point adder.

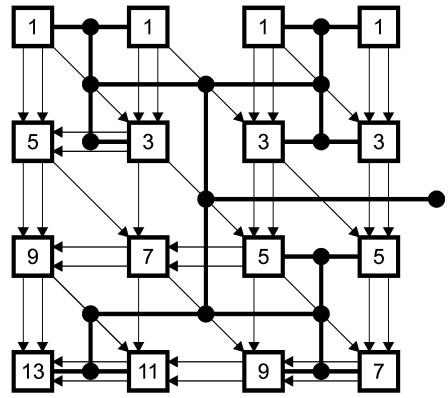

Fig. 8. 16-bit fixed-point multiplier.

720 MHz in 90-nm layout, based on the corresponding results in 180-nm technology.

Finally, we have fabricated initial prototypes of the cell in 500-nm technology. Functional testing has verified that the cell operates correctly in memory mode and mathematics mode.

III. Software Tools

We have created a suite of software tools for mapping algorithms onto the medium-grain reconfigurable architecture. Currently, the software allows users to construct modules such as multipliers, place these modules on the array of cells, and connect their inputs and outputs. In this way, users can easily assemble entire algorithms. A built-in emulator gives users the capability to feed data into their designs and obtain cycle-by-cycle results. This component contains an accurate model of the cells and interconnection structures. Future developments will automate the synthesis process.

Fig. 6 contains a screenshot of the editor used for designing modules such as multipliers. The user can define the configuration of each cell in a separate window and select between memory mode and mathematics mode. Shown on the screen is part of the finite-impulse response (FIR) filter described in Section V. The user can toggle the display of the interconnections.

Unlike software tools for other platforms, the emulator does not estimate the propagation delays on the physical device. The pipelined interconnection network allows the architecture to run a fixed clock frequency, regardless of the current configuration. Instead, the emulator allows users to count the number of cycles required by the algorithm. Multiplying the cycle count by the clock period produces the total execution time. Thus, the medium-grain reconfigurable architecture decouples the circuit design from the software interface.

IV. Arithmetic Modules

Most algorithms used in DSP contain a large number of multiplications and additions. As described in this section, mapping both fixed-point and floating-point arithmetic onto the reconfigurable architecture is quite straightforward. The diagrams that appear here illustrate the basic structure of such modules. (The software tools necessarily require more details.) Users can create libraries of common modules to quickly assemble entire algorithms.

A. Fixed-Point Adder

Fig. 7 depicts a 32-bit fixed-point adder. Essentially, the structure is the pipelined equivalent of a ripple-carry adder with 4-bit computational units. Notice the carry signal propagating from cell to cell across the local network. The adder generates the output in digit-serial order during the clock cycles indicated. For maximum efficiency, the modules driving the adder should generate data in this fashion as well. The H-tree preserves the relative ordering of the data between modules.

B. Fixed-Point Multiplier

Fig. 8 illustrates a 16-bit fixed-point multiplier. The structure of this module resembles a carry-save multiplier, and can be extended to any word length [15]. One input is applied to the top row of cells in digit-parallel order; the other input is applied to the right column of cells in digit-serial order. The output is generated in digit-serial order by the seven cells on the right and bottom. The matrix of elements inside each cell supports both unsigned and two's-complement arithmetic.

C. Shifter

Shifters are a necessary component of many computations, including floating-point arithmetic and CORDIC rotations. One

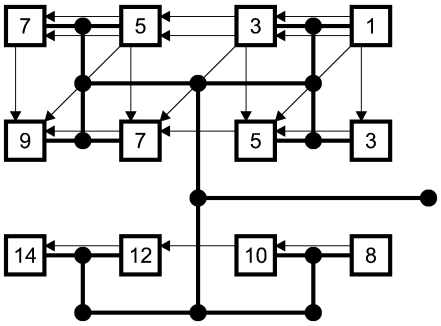

Fig. 9. 16-bit logarithmic shifter.

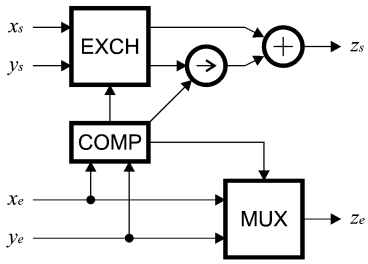

Fig. 10. Diagram of floating-point adder.

can implement either a linear or logarithmic shifter on the reconfigurable architecture. A linear shifter has a simple structure that resembles a fixed-point multiplier. However, a logarithmic shifter usually requires fewer cells, as in the 16-bit left shifter in Fig. 9. The top row of cells can shift the data from zero to four bits to the left. The second and third rows act as multiplexers that apply optional shifts of 4 and 8 bits, respectively. The module uses the global network to connect these two rows, as some data must travel between non-neighboring cells.

D. Floating-Point Adder

The reconfigurable architecture can implement floating-point arithmetic using fixed-point modules as building blocks. For example, Fig. 10 gives a diagram of a floating-point adder that operates on inputs x and y. A comparator first determines the difference between the exponents x_e and y_e. This result is used to align the significands x_s and y_s by shifting one or the other to the right. A fixed-point adder computes the sum, while a multiplexer selects the larger of the two exponents. For completeness, the module should also realign the result, but this operation is best performed after completing all floating-point manipulations.

Although one could design an adder to work with floating-point numbers in IEEE format, a hybrid representation reduces the hardware required for individual operations. Suppose that numbers contain a 28-bit denormalized significand and a 10-bit exponent, both in two's-complement format. Also, assume that the last two bits of the exponent are always zero. This property implies that the significand is only shifted in 4-bit units, reducing the size of the shifter. Conversion to and from IEEE single-precision format would be straightforward.

Fig. 11 shows the resulting implementation of the floating-point adder. The structure requires 52 cells, or the equivalent of a 208-bit fixed-point adder. The latency through the module is

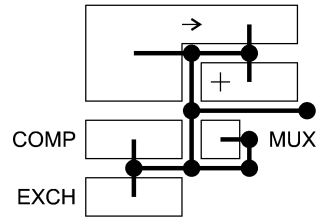

Fig. 11. Implementation of floating-point adder.

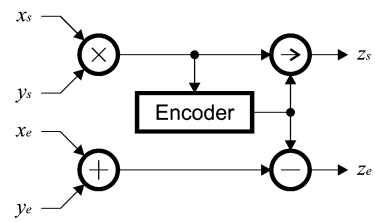

Fig. 12. Diagram of floating-point multiplier.

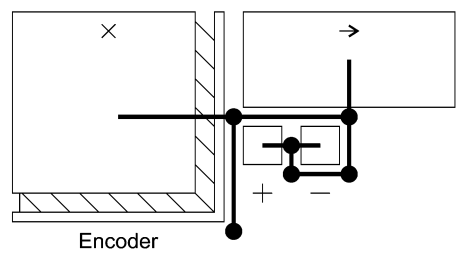

Fig. 13. Implementation of floating-point multiplier.

57 clock cycles, measured from the first input to the final output.

E. Floating-Point Multiplier

A diagram of a floating-point multiplier appears in Fig. 12. As before, the operation can be reduced to fixed-point operations. The module multiplies the two significands x_s and y_s, and adds the two exponents x_e and y_e. The final stages realign the result by shifting the significand left and subtracting the appropriate value from the exponent. An encoder determines the appropriate degree of shifting.

Fig. 13 depicts the implementation of the floating-point multiplier. This example uses the same number format as the floating-point adder. Notice that the outputs of the multiplier connect directly to the encoder for high efficiency. In all, the structure requires 104 cells and has a latency of 74 clock cycles. Much of the latency originates from the dependency of the shift register on the final output of the multiplier and encoder. Notice that all modules mapped on the reconfigurable architecture can initiate one operation per clock cycle.

V. DSP Benchmarks

To evaluate the medium-grain architecture, we have used the software tools to implement and test several benchmarks for DSP hardware. This section presents three examples: a 12-tap FIR filter, a 16-stage CORDIC unit, and a 256-point fast fourier transform (FFT). We assume that the input data has 16-bit fixed-

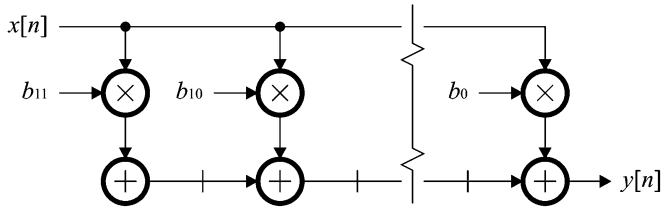

Fig. 14. Diagram of FIR filter.

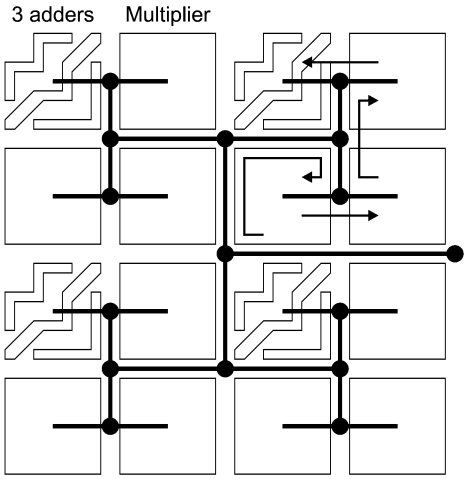

Fig. 15. Implementation of FIR filter.

point format, although the algorithms may calculate intermediate results to higher precision. This section describes the implementation of each benchmark and compares the execution time to other solutions. For clarity, the diagrams shown are much simpler than the view provided by the software.

A. FIR Filter

Fig. 14 gives a block diagram of the 12-tap FIR filter. This structure harnesses the power of the reconfigurable architecture by performing all operations in parallel. The input x is passed to 12 multipliers. Each unit i is configured to multiply the data by a fixed coefficient b_i. The outputs of the multipliers are added in pipelined fashion so that the output y becomes

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_{11} x[n-11]. \quad (3)$$

Fig. 15 depicts the implementation of the filter on the reconfigurable architecture. The structure consists of four identical modules, each of which handles three coefficients. Each module occupies an 8×8 block of cells and contains three multipliers and three adders. To implement higher-order filters, one would simply add more modules. The current design allows filter coefficients within the range $[-1, 1)$. However, the design uses 20-bit adders to mitigate rounding errors.

The global interconnection network offers several features that simplify the mapping process. The input lines of the H-tree broadcast the x input to all modules simultaneously. Each module collects the outputs of the three multipliers into a bundle and passes them to the inputs of the corresponding adders. All three outputs incur the same latency over the H-tree.

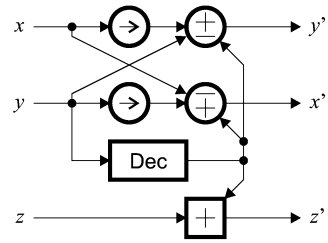

Fig. 16. Diagram of one CORDIC stage.

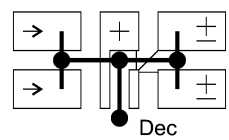

Fig. 17. Implementation of one CORDIC stage.

The tree structure is also useful for connecting adjacent modules in sequence. The arrows in Fig. 15 suggest one method for traversing the tree.

The 12-tap filter has a latency of 61 clock cycles from the arrival of the first input to the computation of the last output. The critical path involves 20 cells, 15 local busses, and 74 levels of the H-tree. The total execution time for a 256-point data stream equals the latency plus 244 clock cycles, or 316 clock cycles. Hence, the expected execution time of the 12-tap filter is 0.44 μs when running at the estimated clock rate of 720 MHz in 90-nm technology.

B. CORDIC Unit

CORDIC transformations offer one way to perform complex mathematical operations, such as division, square root, and rectangular-to-polar conversion. A typical CORDIC transformation for n-bit inputs contains n stages, each of which follows the diagram in Fig. 16. Initially, the system sets x and y to the inputs (such as the rectangular coordinates) and z to zero. Iteration i then updates the data as follows:

$$\begin{aligned} x' &= x \mp 2^{-i} y \\ y' &= y \pm 2^{-i} x \\ z' &= z + f(i). \end{aligned} \quad (4)$$

The value of $f(i)$ and the choice of addition or subtraction depends on the sign of y. After n stages, x and/or z will contain the desired results and y will be essentially zero.

We have mapped a 16-bit CORDIC stage on the reconfigurable architecture, as shown in Fig. 17. A 4×8 block of cells contains two adder/subtracters, two shifters, one constant adder, and a small decoder that determines the sign of y. The implementation contains a number of features to improve performance. The two shifters translate data no more than four bits to the right. The remaining shift amount is performed with hardwired connections. In addition, the two values of $f(i)$ needed for the current stage are hardcoded into the constant adder. Finally, all modules work with 24-bit data rather than 16-bit data to alleviate rounding errors.

Algorithms could compute CORDIC transformations in two ways. A low-area approach would use a single CORDIC stage to

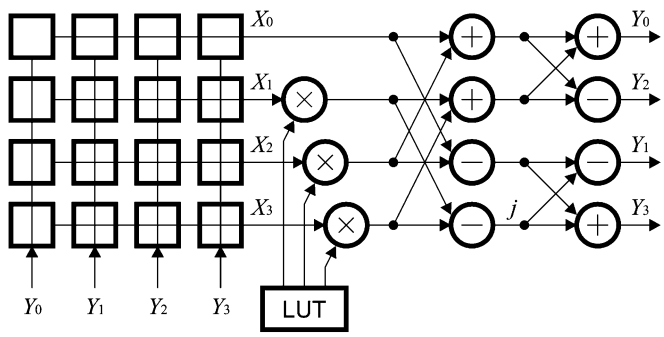

Fig. 18. Diagram of FFT.

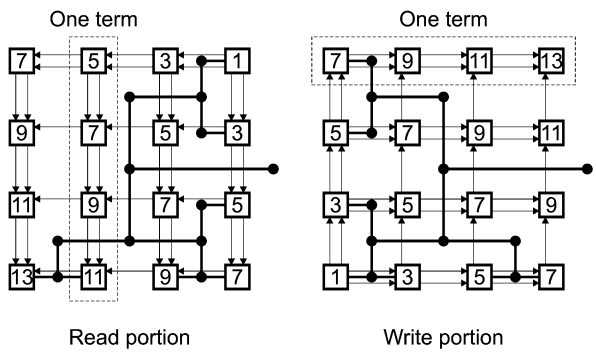

Fig. 19. FFT memory unit.

process a group of data points one iteration at a time. After partial reconfiguration, the stage would be ready for the next iteration. A high-performance approach would cascade 16 CORDIC stages to form a 16 × 32 block. The latency through a single stage is 17 clock cycles; the latency through the entire block would be 313 clock cycles, including the communication delay over the global network. This latency translates into 0.43 μs at 720 MHz. The throughput of the CORDIC transformation is the maximum possible: 720 Msamples/s.

C. FFT

Fig. 18 gives a diagram of a radix-4 FFT. Each sample is represented as a complex number with 16-bit real and 16-bit imaginary portions. The algorithm as a whole consists of four computational stages. During each stage, the system reads four samples from memory, calculates a so-called "dragonfly" operation, and stores the results back into memory at different addresses. A sophisticated memory unit allows all eight operations to occur simultaneously. The process repeats for the remaining groups of samples in the dataset.

The "dragonfly" is described by the matrix equation

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} X_0 \\ W_1X_1 \\ W_2X_2 \\ W_3X_3 \end{bmatrix}. \tag{5}$$

This operation consists of three complex multiplications and three complex additions or subtractions. "Twiddle factors" W_1, W_2, and W_3 are generated by a lookup table.

Fig. 19 gives more detail about the special memory unit. The two structures shown are superimposed on top of each other.

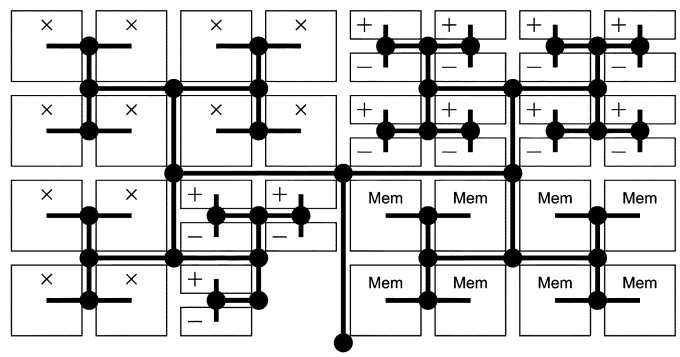

Fig. 20. Implementation of FFT.

Each cell divides its internal memory into two banks: one for reads and one for writes. The two banks operate independently, with separate address and control signals. Each column of cells forms a read memory for one of the inputs to the kernel. Each row, in turn, forms a write memory for one of the outputs. A 4 × 4 block of cells can manage one 4-bit portion of all eight inputs and outputs.

The overall implementation of the FFT appears in Fig. 20. The kernel of the algorithm maps onto a 32 × 16 array of cells. Input data arrives in memory as a fixed-point number with 16-bit real and 16-bit imaginary portions. The adders work with 24-bit data to avoid rounding errors. A separate lookup table, not shown in Fig. 20, generates the "twiddle factors" used in the multipliers.

The 256-point FFT has a latency of 68 clock cycles between the two memories. The critical path involves 19 cells, 13 local busses, and 72 levels of the H-tree. Since each processing stage handles 64 groups of samples, the time required per stage equals the latency plus 63 clock cycles, or 131 clock cycles. Multiplying this number by 4 yields the total execution time, 524 clock cycles or 0.73 μs at 720 MHz.

VI. Analysis

In this section, we analyze the performance and flexibility of the medium-grain architecture. We first compare the execution times of the DSP benchmarks to several digital signal processors and FPGAs. We then compare the alternatives on a more architectural level. Initial results have been reported in [16] and [17].

A. Performance Comparison

Table I compares the execution times of the FIR filter and FFT to the reported results for several fixed-point DSPs. The devices selected for this comparison represent some of the most advanced DSP engines available today: a Blackfin processor (ADSP-BF533) from Analog Devices and two TMS320 processors (TMS320C2 and TMS320C4) from Texas Instruments. The execution times for these platforms were calculated from data available at the manufacturers' websites. For comparison, we also include the execution time for a 1.4-GHz Pentium 4 running the FFT benchmark [21]. The "scaled time" values are normalized to 180-nm technology, assuming that the clock period scales linearly. All implementations work with a 256-sample dataset with 16-bit fixed-point data, except for the Pentium 4,

TABLE I
COMPARISON WITH DIGITAL SIGNAL PROCESSORS

Parameter		ADSP-BF533	TMS320C2	TMS320C4	Pentium 4	Proposed	Proposed
Technology (nm)			150	90	180	180	90
Clock frequency (MHz)		750	300	1000	1400	267	∼720
FIR:	Execution time (μs)	2.39	9.02	1.29		1.18	0.44
	Scaled time		10.82	2.58		1.18	0.88
FFT:	Execution time (μs)	3.10	9.25	1.24	14.29	1.96	0.73
	Scaled time		11.10	2.48	14.29	1.96	1.45

TABLE II
COMPARISON WITH RECONFIGURABLE HARDWARE

Parameter		XC2V500	XC2VP4	XC4VSX25	Proposed	Proposed
Technology (nm)		150	130	90	180	90
Slices / cells		3,072	6,768	10,240	4,096	4,096
Package size / estimated area (mm^2)		384	384	729	∼492	∼123
FIR:	Clock frequency (MHz)	218	258	323	267	∼720
	Utilization	0.32	0.27	0.05	0.06	0.06
	Equivalent area (mm^2)	129	104	36	30	7
	Execution time (μs)	1.27	1.07	0.87	1.18	0.44
	Area × time	164	111	63	35	3
	Scaled time	1.52	1.48	1.74	1.18	0.88
CORDIC:	Clock frequency (MHz)	190	198	299	267	∼720
	Utilization	0.20	0.20	0.06	0.13	0.13
	Equivalent area (mm^2)	77	77	44	64	16
	Execution time (μs)	1.45	1.39	0.92	1.17	0.43
	Area × time	112	107	40	75	7
	Scaled time	1.74	1.92	1.84	1.17	0.87
FFT:	Clock frequency (MHz)	232	293	458	267	∼720
	Utilization	0.52	0.23	0.12	0.13	0.13
	Equivalent area (mm^2)	200	88	87	64	16
	Execution time (μs)	1.34	1.06	0.72	1.96	0.73
	Area × time	268	93	63	125	12
	Scaled time	1.61	1.47	1.44	1.96	1.45

which handles floating-point data. The time required to load the data into memory is not included.

As shown, the proposed architecture surpasses the performance of DSPs in similar technologies. These results are not surprising, as reconfigurable hardware can exploit the parallelism of DSP. The deeply pipelined design does have one drawback: filling the pipeline at the beginning of each execution stage consumes many cycles. Processing larger sample sets would improve the results, since the modules would have higher utilization.

Table II repeats the comparison with three Xilinx FPGAs: a Virtex-II (XC2V500), a Virtex-II Pro (XC2VP4), and a Virtex-4 (XC4VSX25). All three devices contain special features to accelerate DSP, including dedicated 18-bit multipliers, and in the case of the Virtex-4, 48-bit adders as well. The clock frequencies for the FIR and CORDIC benchmarks were calculated by mapping the respective intellectual property (IP) cores using maximum pipelining and the highest effort level. The results for the FFT were taken directly from the datasheet for the FFT IP core using a radix-4, burst-mode algorithm with the data load time subtracted.

Comparing the scaled execution times, the Virtex-4 does slightly outperform the proposed architecture for the FFT benchmark. We anticipate that further optimizations to the algorithm—in particular, decreasing the pipeline fill time—would improve the result. The proposed architecture does achieve notably higher performance for the other two benchmarks.

Table II also gives an estimate of the device utilization for each benchmark. For the Xilinx devices, the value given is the number of slices consumed by the algorithm, divided by the total number of slices in the device. For the proposed architecture, we assume that the target device has a 64 × 64 array of cells. We then estimate that the device would have an active area of 492 mm^2 in 180-nm technology, based on individual layouts of the cell and switches. This result would scale down to 123 mm^2 in 90-nm technology. The package sizes of the Xilinx devices are given for comparison.

Although direct comparison of the areas is difficult, due to the differences in technologies and die sizes, the results indicate that the medium-grain architecture incurs significantly less area overhead than commodity FPGAs. Note especially that the device utilization is roughly equivalent to that of a larger Virtex-4 in significantly better technology. The area × execution time parameter, which provides some indication of the power consumption, is significantly lower for the proposed architecture in 90-nm technology. However, we should stress that these values are only preliminary estimates.

B. Architecture Comparison

As demonstrated in Section VI-A, the medium-grain reconfigurable architecture does not implement algorithms in the same manner as FPGAs. Synthesis tools for FPGAs translate an algorithm into a series of logic expressions, which map onto the 1-bit cells. The interconnection structure routes data between

cells in bit-length units. In contrast, the proposed architecture represents algorithms as a set of modules composed of 4-bit cells. The elements inside the cell allow the architecture to maintain 1-bit flexibility. The global H-tree routes data between modules in word-length units. The pipelined interconnection network unifies all computations, so that the device always runs at the maximum clock frequency.

The method used to implement several important operations also differs from FPGAs. For example, the Xilinx Virtex-II features embedded 18-bit multipliers and fast carry logic within the basic cells. The Xilinx Virtex-4 uses special DSP blocks that can perform 18-bit multiplication and 48-bit addition. In contrast, the medium-grain cell array integrates multipliers and adders with other modules. Each cell can perform mathematics functions or memory operations, so designers can create powerful memory units as well. As always, designers can customize the word length and number of modules to suit application requirements.

The proposed architecture also has several advantages over coarse-grain reconfigurable devices. Coarse-grain cells generally perform a limited number of 16- or 32-bit operations. To implement the control logic necessary for DSP, some architectures integrate the coarse-grain array with a separate fine-grain array or microprocessor. Another option is to use a heterogeneous array of cells, as with the PACT XPP and QuickSilver Adapt2000. In contrast, the proposed architecture uses the same cells to perform binary arithmetic, memory operations, and control logic. The 4-bit granularity also gives designers more flexibility over the word length.

VII. Conclusion

In this paper, we have discussed the implementation of several DSP benchmarks on a medium-grain reconfigurable architecture. The initial results provided by the software emulator show great promise with respect to other solutions. On one hand, the execution times of several benchmarks meets or exceeds the performance of advanced DSPs and FPGAs. On the other hand, a number of architectural innovations make it possible for the architecture to implement DSP efficiently.

Some key features of the architecture are as follows.

- *Two-Level Array:* The proposed architecture contains a two-level array of 4-bit cells and 1-bit elements. This approach allows the design to achieve the performance required for binary arithmetic, as well as the flexibility required for control logic. For instance, we demonstrated that the same cells can implement fixed-point and floating-point arithmetic, CORDIC stages, and sophisticated memory units for a radix-4 FFT.
- *Two-Mode Cell Configurations:* Traditional fine-grain reconfigurable hardware suffers from complex interconnection structures. In contrast, the cell only assumes two configurations: one optimized for memory operations and the other for mathematics functions. Since the design requires minimal routing resources at the fine-grain level, each cell behaves like a medium-grain computational unit. The reduced area overhead can be clearly seen in the comparisons with FPGAs.
- *Hierarchical Interconnection Structure:* The interconnection network used on the device recognizes that algorithms are composed of discrete modules, such as multipliers and adders. Hence, the architecture provides a mesh of busses for data transfer within a module, as well as a global H-tree for connecting modules together. The higher levels of the H-tree manipulate data in word-length units. All 4-bit portions incur the same latency between modules. We found that this interconnection scheme greatly simplifies the mapping process.
- *Pipelined Execution:* All cell computations and data transfers are pipelined for high throughput. The uniform pipelining scheme allows the device to maintain the maximum clock frequency at all times, regardless of the current configuration. One drawback of this approach is that the deep pipelines require a significant number of cycles to fill up during each execution stage. The performance results for the FFT illustrate this trend. However, once the pipeline is filled, the device can generate results at very high throughput, leading to lower execution times. Algorithms that work with larger sample sizes achieve greater speedup.

Taken as a whole, the proposed architecture supports a large application space with a number of orthogonal axes. Designers can customize the word length, amount of parallelism, and number of modules to meet the needs of the application. In this manner, systems can balance performance and flexibility while minimizing development costs.

Acknowledgment

All opinions expressed in this paper are the authors' and do not necessarily reflect the policies and views of DHS, DOE, or ORAU/ORISE.

References

[1] R. Tessier and W. Burleson, , Y. Hu, Ed., "Reconfigurable computing for digital signal processing: a survey," in *Programmable Digital Signal Processors*. New York: Marcel Dekker, 2001.

[2] Xilinx, San Jose, CA, "Virtex-4 family overview," Literature Number DS112, Feb. 2007, vol. 1.5 [Online]. Available: http://www.xilinx.com/bvdocs/publications/ds112.pdf

[3] R. Hartenstein, "Coarse grain reconfigurable architectures," in *Proc. 6th Asia South Pacific Des. Autom. Conf.*, 2001, pp. 564–570.

[4] C. Ebeling, D. Cronquist, P. Franklin, and C. Fisher, "RaPiD—A configurable computing architecture for compute-intensive applications," Dept. Comput. Sci. Eng., Univ. Washington, Pullman, Tech. Rep. TR-96-11-03, Nov. 1996.

[5] R. Hartenstein, M. Herz, T. Hoffmann, and U. Nageldinger, "Using the KressArray for reconfigurable computing," *Proc. SPIE*, vol. 3526, pp. 150–161, Oct. 1998.

[6] H. Zhang *et al.*, "A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1697–1704, Nov. 2000.

[7] P. Heysters and G. Smit, "Mapping of DSP algorithms on the MONTIUM architecture," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2003, pp. 180–185.

[8] PACT Informations Technologie, "GmbH the XPP white paper," Mar. 2002, vol. 2.1.

[9] B. Plunkett and J. Watson, "Adapt2400 ACM architecture overview," QuickSilver Technology, Inc., San Jose, CA, 2004 [Online]. Available: http://www.qstech.com/pdfs/Adapt2000_Overview.pdf

[10] A. Marshall *et al.*, "A reconfigurable arithmetic array for multimedia applications," in *Proc. 7th ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 1999, pp. 135–143.

[11] S. C. Goldstein *et al.*, "PipeRench: A reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, Apr. 2000.

[12] J. Becker, T. Pionteck, C. Habermann, and M. Glesner, "Design and implementation of a coarse-grained dynamically reconfigurable hardware architecture," in *Proc. IEEE Computer Soc. Workshop VLSI*, 2001, pp. 41–46.
[13] Elixent, Bristol, U.K., "Applications of the D-fabrix array," White Paper WP0001, 2001.
[14] M. Myjak and J. Delgado-Frias, "A two-level reconfigurable architecture for digital signal processing," *Microelectron. Eng.*, vol. 84, no. 2, pp. 244–252, Feb. 2007.
[15] M. J. Myjak, "A medium-grain reconfigurable architecture for digital signal processing," Ph.D. dissertation, School Elect. Eng. Comput. Sci., Washington State Univ., Pullman, May 2007.
[16] M. J. Myjak, J. K. Larson, and J. G. Delgado-Frias, "Mapping and performance of DSP benchmarks on a medium-grain reconfigurable architecture," in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms*, 2006, pp. 123–129.
[17] M. J. Myjak and J. G. Delgado-Frias, "Medium-grain cells for reconfigurable DSP hardware," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 6, pp. 1255–1265, Jun. 2006.
[18] M. J. Myjak and J. G. Delgado-Frias, "A bit-serial cell for reconfigurable DSP hardware," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, 2005, pp. 960–963.
[19] M. J. Myjak and J. G. Delgado-Frias, "Superpipelined reconfigurable hardware for DSP," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2007, pp. 3670–3673.
[20] J. Larson and K. Prajapati, "VLSI implementation of an H-tree switch node," Washington State Univ., Pullman, Design Project Report EE 586, (2005, Nov.) [Online]. Available: http://www.eecs.wsu.edu/~jdelgado/reports/Larson_Prajapati.pdf
[21] D. Etiemble, "Optimizing DSP and media benchmarks for Pentium 4: hardware and software issues," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2002, pp. 109–112.

Mitchell J. Myjak (S'99–M'06) received the B.S. degree in electrical engineering from the University of Portland, Portland, OR, in 2002, the M.S. degree in electrical engineering and the Ph.D. degree in electrical and computer engineering from Washington State University, Pullman, WA, in 2004 and 2006, respectively.

He is currently a Research Engineer with Pacific Northwest National Laboratory, Richland, WA. His research interests include reconfigurable hardware, nuclear instrumentation, and electronic systems for homeland security applications.

Dr. Myjak's graduate studies were supported in part by a fellowship from the U.S. Department of Homeland Security.

José G. Delgado-Frias (SM'90) received the B.S. degree from the National Autonomous University of Mexico, Mexico City, Mexico, the M.S. degree from the National Institute for Astrophysics, Optics, and Electronics (INAOE), Puebla, Mexico, and the Ph.D. degree from Texas A&M University, College Station, TX, all in electrical engineering.

He is currently a Professor with the School of Electrical Engineering and Computer Science, Washington State University (WSU), Pullman, where he holds the Boeing Centennial Chair in Computer Engineering. Prior to joining WSU in 2001, he was a Faculty Member with the Electrical and Computer Engineering Department, University of Virginia, Charlottesville, and the Electrical and Computer Engineering Department, State University of New York (SUNY), Binghamton. He was a Post-Doctoral Research Fellow with the Engineering Science Department, University of Oxford, Oxford, U.K. His research interests include high-performance VLSI systems, reconfigurable architectures, network routers, and optimization using genetic algorithms. He is a coauthor of over 160 journal and conference papers and has coedited five books. He holds 27 patents. He cochaired three international workshops on VLSI for neural networks and AI that were held in Oxford, U.K., in 1988, 1990, and 1992. He is Conference Chair of the 2007 International Conference on Circuits, Signals, and Systems (CSS) and the 53rd IEEE Midwest Symposium on Circuits and Systems 2010.

Dr. Delgado-Frias was a recipient of the SUNY System Chancellor's Award for Excellence in Teaching in 1994. He is a member of the Association for Computer Machinery (ACM) and the American Society for Engineering Education (ASEE).