Mapping and Performance of DSP Benchmarks on a Medium-Grain Reconfigurable Architecture

Mitchell J. Myjak, Jonathan K. Larson, and José G. Delgado-Frias
School of Electrical Engineering and Computer Science
Washington State University
Pullman, Washington 99164-2752 USA
Email: jdelgado@eecs.wsu.edu

***Abstract*— Reconfigurable hardware has become a well-accepted option for implementing digital signal processing. 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 recently created software tools that allow users to map algorithms onto the reconfigurable platform. This paper analyzes the implementation of several common benchmarks, ranging from simple floating-point arithmetic to a radix-4 Fast Fourier Transform. The results are compared to contemporary digital signal processing hardware.**

***Index Terms*— Digital signal processing, floating-point arithmetic, medium-grain reconfigurable architectures, 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 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 [9].

Recently, researchers have proposed new reconfigurable architectures in which each cell performs 16-bit or 32-bit operations [2]. These coarse-grain devices achieve high performance for DSP. Some examples are the one-dimensional RaPiD array [3], the two-dimensional KressArray [4], and the heterogeneous Pleiades [5] and MONTIUM [6] 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 ALU or a memory [7]. 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 [8].

As a third alternative, medium-grain reconfigurable hardware attempts to balance performance and flexibility. Each cell works with 4-bit or 8-bit data, so modules such as 16-bit multipliers would require several 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 act as a small memory.

Mapping DSP onto reconfigurable hardware requires a sophisticated set of computer-aided design (CAD) tools. Most commercial software supports design synthesis, timing analysis, and circuit simulations. As an initial step, we have created CAD 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 benchmark algorithms and evaluate the performance of the medium-grain architecture. This analysis forms the main focus of this paper.

Section II gives a brief overview of the medium-grain reconfigurable 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 benchmarks and compares the resulting execution times with other DSP hardware. Section VI discusses the proposed architecture with respect to traditional fine-grain and coarse-grain devices. Finally, Section VII concludes the paper.

II. Hardware Architecture

Our medium-grain reconfigurable architecture uses an innovative two-level approach to combine high performance with high flexibility [16]. The device as a whole contains an array

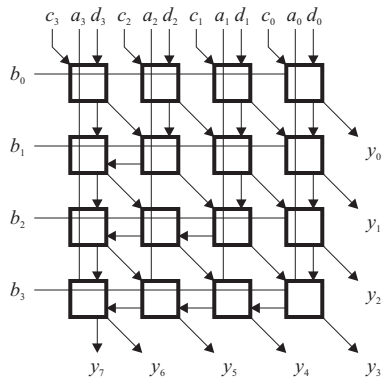

Fig. 1. Cell in mathematics mode.

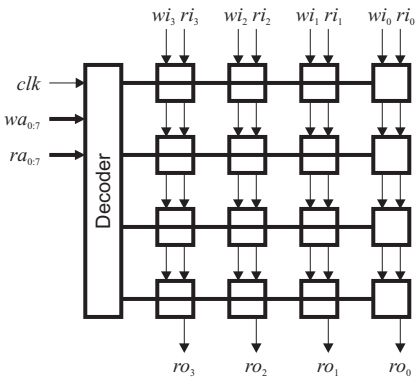

Fig. 2. Cell in memory mode.

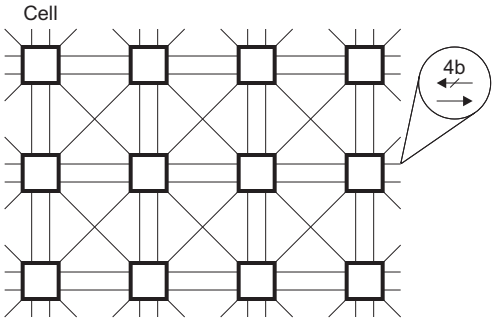

Fig. 3. Cells with local interconnection network.

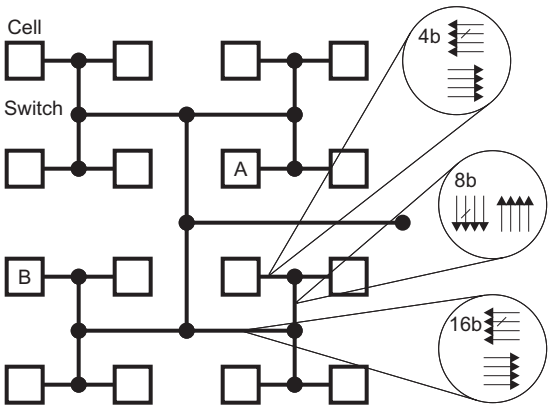

Fig. 4. Global network.

of 4-bit cells. Each cell, in turn, consists of a 4×4 matrix of 1-bit lookup tables, or "elements". This structure can be configured in two ways. In mathematics mode, shown in Fig. 1, the elements are configured to execute 4-bit binary arithmetic. In memory mode, shown in Fig. 2, the elements implement a 128×4-bit random-access memory with separate read and write ports.

To implement DSP, the array of cells is partitioned into blocks of various sizes. Each block is configured to implement a specific module, such as a multiplier, adder, or memory unit. Cells within a module exchange inputs and outputs with their immediate neighbors, whereas modules work with data in word-length units. We previously proposed a dual interconnection network that optimizes both types of communication [17]. A local mesh connects neighboring cells, and a global H-tree routes data between modules. We recently enhanced this design to simplify the mapping process.

Fig. 3 depicts the redesigned 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. 4 illustrates the global network, which resembles an H-tree. This structure can transfer data efficiently between any two cells on the array. Each level of the tree contains four input busses and four output busses. The number of bits per bus starts at 4 bits 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 these 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 the figure 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 will incur the same latency over the H-tree.

III. Software Tools

We have created a set of CAD tools for the medium-grain reconfigurable architecture [18]. These tools accurately model the cells and interconnection structures. Currently, the software allows users to construct modules such as multipliers, place these modules on the array of cells, and manually connect their inputs and outputs. In this way, users can assemble entire algorithms. A built-in emulator gives users the capability to feed data into their designs and obtain cycle-by-cycle results.

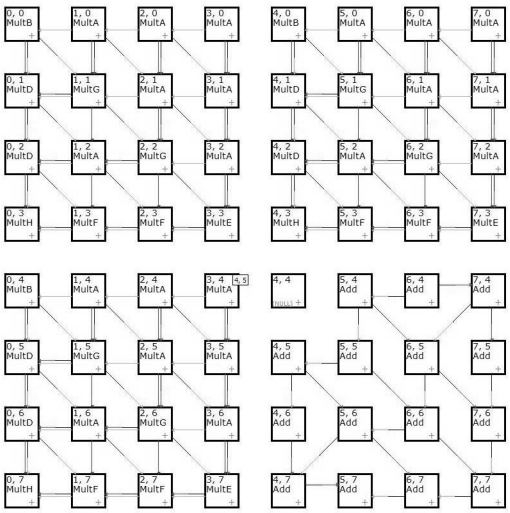

Fig. 5. Screenshot of CAD tools.

Future developments will automate this synthesis process.

Figure 5 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 FIR filter described in Section V. The user can toggle the display of the interconnections.

Unlike CAD 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 CAD tools necessarily require more details.) Users can create libraries of common modules to quickly assemble entire algorithms. Compared to the examples in [17], the redesigned interconnection structure offers greater modularity, as the local network operates independently of the global network.

A. Fixed-Point Adder

Fig. 6 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.

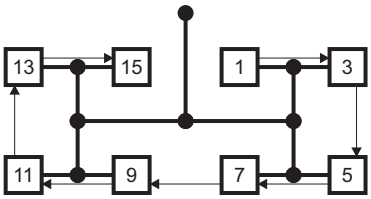

Fig. 6. 32-bit fixed-point adder.

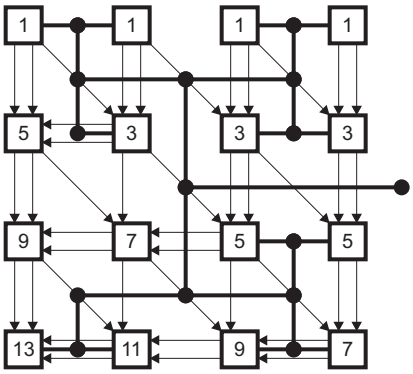

Fig. 7. 16-bit fixed-point multiplier.

B. Fixed-Point Multiplier

Fig. 7 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 [19]. 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 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. 8. 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 four and eight 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. 9 gives a diagram of a floating-point adder

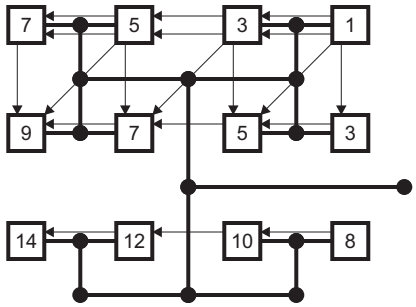

Fig. 8. 16-bit logarithmic shifter.

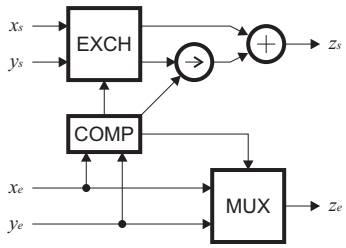

Fig. 9. Diagram of floating-point adder.

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. 10 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 57 clock cycles, measured from the arrival of the first input to the computation of the final output.

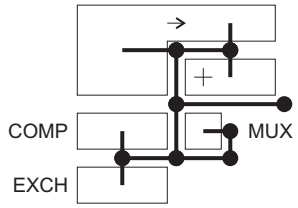

Fig. 10. Implementation of floating-point adder.

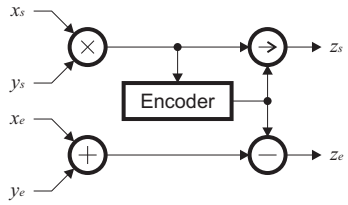

Fig. 11. Diagram of floating-point multiplier.

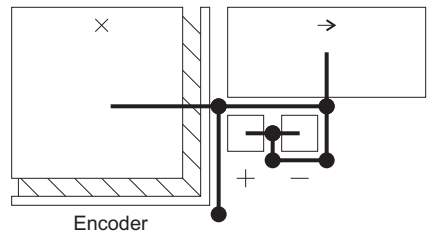

Fig. 12. Implementation of floating-point multiplier.

E. Floating-Point Multiplier

A diagram of a floating-point multiplier appears in Fig. 11. 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. 12 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 performance of the medium-grain reconfigurable architecture, we have used the CAD 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-point format, although the algorithms may calculate intermediate results to higher precision. For each benchmark, we describe the final implementation and compare the execution time to other solutions. For clarity, the diagrams shown are much simpler than the view provided by the CAD tools.

A. FIR Filter

Fig. 13 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 twelve multipliers. Each unit i is configured to multiply the

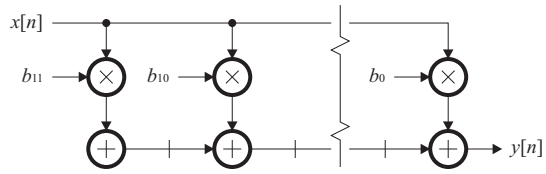

Fig. 13. Diagram of FIR filter.

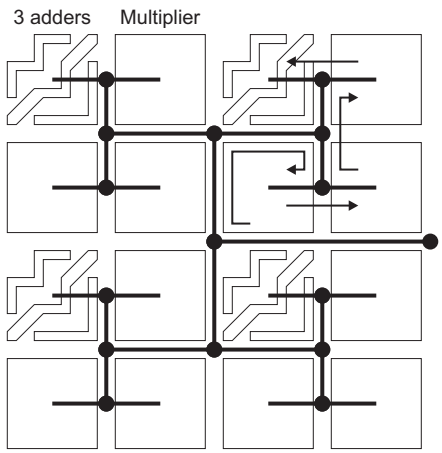

Fig. 14. Implementation of FIR filter.

data by a fixed coefficient b_i. The outputs of the multipliers are added in pipelined fashion so that the output y becomes

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

Fig. 14 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. The tree structure is also useful for connecting adjacent modules in sequence. The arrows in the figure 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. Simulations in 90-nm CMOS technology indicate that the architecture can operate at a clock rate of 750 MHz. Hence, the expected execution time of the 12-tap filter is 0.42 μs. Table I compares the execution time with several advanced digital signal processors. (Results from FPGAs were not available.)

TABLE I
EXECUTION TIME OF 12-TAP FIR FILTER

Device	Technology	Frequency	Time
Analog ADSP-BF533		750 MHz	2.39 μs
TI TMS320C62	150-nm	300 MHz	9.02 μs
TI TMS320C64	90-nm	1000 MHz	1.29 μs
Our approach	90-nm	750 MHz	0.42 μs

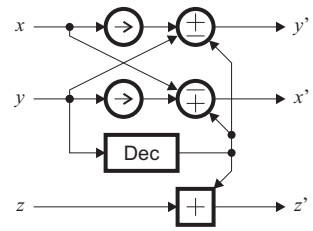

Fig. 15. Diagram of one CORDIC stage.

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. 15. 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:

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

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. 16. 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.

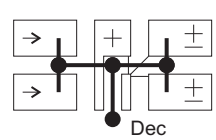

Fig. 16. Implementation of one CORDIC stage.

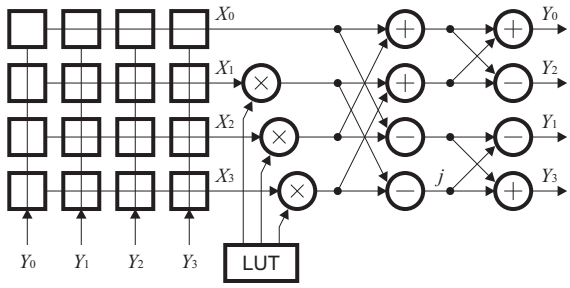

Fig. 17. Diagram of FFT.

Algorithms could compute CORDIC transformations in two ways. A low-area approach would use a single CORDIC stage to 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 0.42 μs at 750 MHz. The throughput of the CORDIC transformation is the maximum possible: 750 Msamples/s.

C. Fast Fourier Transform

We previously mapped a 256-point FFT on the reconfigurable architecture using a simple radix-2 algorithm [17]. With the redesigned interconnection network, we can now upgrade to a radix-4 FFT. Fig. 17 gives a diagram of this algorithm. 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

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} X_0 \\ W_1X_1 \\ W_2X_2 \\ W_3X_3 \end{bmatrix}. \quad (3)$$

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. 18 gives more detail about the special memory unit. The two structures shown are superimposed on top of each other. Each cell divides its internal memory into two banks: one for reads and the other 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.

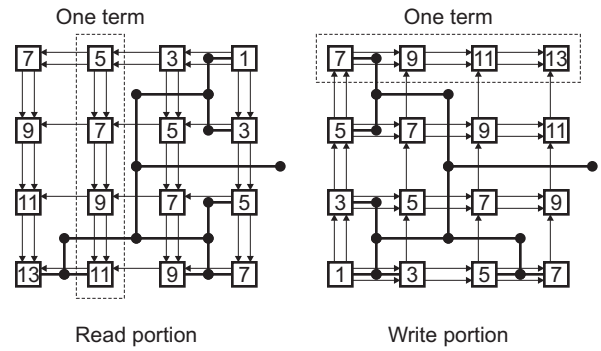

Fig. 18. FFT memory unit.

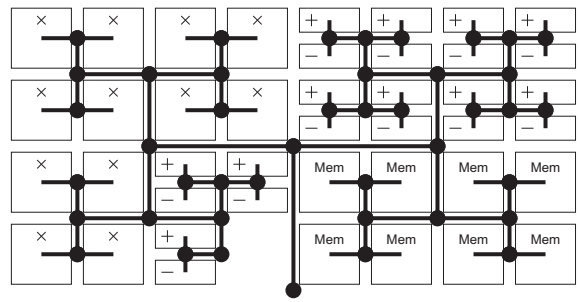

Fig. 19. Implementation of FFT.

The overall implementation of the FFT appears in Fig. 19. 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 the figure, 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 0.70 μs at 750 MHz. The execution time is slightly greater than the Xilinx Virtex-4, as shown in Table II, but less than contemporary digital signal processors. The results would improve for larger sample sets, since the modules would have higher utilization. Filling the pipeline at the beginning of each execution stage limits performance.

TABLE II
EXECUTION TIME OF 256-POINT FFT

Device	Technology	Frequency	Time
Analog ADSP-BF533		750 MHz	3.10 μs
TI TMS320C62	150-nm	300 MHz	9.25 μs
TI TMS320C64	90-nm	1000 MHz	1.24 μs
Xilinx Virtex-II	150-nm	195 MHz	1.48 μs
Xilinx Virtex-II Pro	130-nm	223 MHz	1.30 μs
Xilinx Virtex-4	90-nm	421 MHz	0.69 μs
Our approach	90-nm	750 MHz	0.70 μs

VI. Architecture Comparison

As demonstrated in the previous section, 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 local mesh connects cells within a module, while 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-bit 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. We found that the global interconnection network greatly simplifies data communication throughout the device. The hierarchical nature of the H-tree should also facilitate the development of automated synthesis tools. This subject remains a goal for future work.

Acknowledgment

M. Myjak is supported by a graduate fellowship from the U.S. Department of Homeland Security (DHS). The DHS Scholarship and Fellowship Program is administered by the Oak Ridge Institute for Science and Education (ORISE) for DHS through an interagency agreement with the U.S. Department of Energy (DOE). ORISE is managed by Oak Ridge Associated Universities under DOE contract number DE-AC05-00OR22750. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of DHS, DOE, or ORISE.

This research has been sponsored in part by the Boeing Endowed Chair at the School of EECS, Washington State University.

References

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

[2] R. Hartenstein, "Coarse grain reconfigurable architectures," in *Proc. 6th Asia South Pacific Design Automation Conference*, Yokohama, Japan, pp. 564–570, 2001.

[3] C. Ebeling, D. Cronquist, P. Franklin, and C. Fisher, "RaPiD—a configurable computing architecture for compute-intensive applications," *University of Washington Department of Computer Science & Engineering Tech Report TR-96-11-03*, Nov. 1996.

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

[5] H. Zhang et al, "A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing," *IEEE J. Solid-State Circuits*, vol. 35, iss. 11, pp. 1697–1704, Feb. 2000.

[6] P. Heysters and G. Smit, "Mapping of DSP algorithms on the MONTIUM architecture," *in Proc. International Parallel and Distributed Processing Symposium*, pp. 180–185, Apr. 2003.

[7] PACT Informationstechnologie GmbH, "The XPP white paper," v. 2.1, Mar. 2002.

[8] B. Plunkett and J. Watson, "Adapt2400 ACM Architecture Overview," QuickSilver Technology, Inc., white paper, 2004.

[9] Xilinx, Inc., "Virtex-4 Family Overview," literature number DS112, v. 1.5, Feb. 2006.

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

[11] S.C. Goldstein et al., "PipeRench: a reconfigurable architecture and compiler," *Computer*, vol. 33, iss. 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 on VLSI*, Orlando, FL, pp. 41–46, Apr. 2001.

[13] Elixent, Inc., "Applications of the D-Fabrix Array," white paper WP0001, 2001.

[14] M.J. Myjak, "A two-level reconfigurable cell array for digital signal processing," M.S. thesis, Washington State University, May 2004.

[15] M. Myjak and J. Delgado-Frias, "A two-level reconfigurable architecture for digital signal processing," *Microelectronic Engineering*, in press.

[16] M. Myjak and J. Delgado-Frias, "A two-level reconfigurable architecture for digital signal processing," in *Proc. 2003 International Conference on VLSI*, Las Vegas, NV, pp. 21–27, Jun. 2003.

[17] M. Myjak, F. Anderson, and J. Delgado-Frias, "H-tree interconnection structure for reconfigurable DSP hardware," in *Proc. 2004 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, NV, pp. 170–176, Jun. 2004.

[18] J.K. Larson, "CAD tool emulation for a two-level reconfigurable cell array for digital signal processing," M.S. thesis, Washington State University, Dec. 2005.

[19] M. Myjak and J. Delgado-Frias, "Pipelined multipliers for reconfigurable hardware," in *Proc. 2004 Reconfigurable Architectures Workshop*, Santa Fé, NM, pp. 150–156, Apr. 2004.