Medium-Grain Cells for Reconfigurable DSP Hardware

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

***Abstract*—Reconfigurable hardware contains an array of programmable cells and interconnection structures. Field-programmable gate arrays use fine-grain cells that implement simple logic functions. Some proposed reconfigurable architectures for digital signal processing (DSP) use coarse-grain cells that perform 16-b or 32-b operations. A third alternative is to use medium-grain cells with a word length of 4 or 8 b. This approach combines high flexibility with inherent support for binary arithmetic such as multiplication. This paper presents two medium-grain cells for reconfigurable DSP hardware. Both cells contain an array of small lookup tables, or "elements", that can assume two structures. In memory mode, the elements act as a random-access memory. In mathematics mode, the elements implement 4-b arithmetic operations. The first design uses a matrix of 4×4 elements and operates in bit-parallel fashion. The second design uses an array of five elements and computes arithmetic functions in bit-serial fashion. Layout simulations in 180-nm CMOS indicate that the parallel cell operates at 267 MHz, whereas the serial cell runs at 167 MHz. However, the parallel design requires over twice the area. The proposed medium-grain cells provide the performance and flexibility needed to implement DSP. To evaluate the designs, the paper estimates the execution time and resource utilization for common benchmarks such as the fast Fourier transform. The architecture model used in this analysis combines the cells with a pipelined hierarchical interconnection network. The end results show great promise compared to other devices, including field-programmable gate arrays.**

***Index Terms*—Digital signal processors (DSPs), digital systems, medium-grain reconfigurability, reconfigurable architectures, very-large-scale integration (VLSI).**

I. Introduction

RECONFIGURABLE hardware has become a well-accepted option for implementing complex processing stages without incurring the development costs of custom integrated circuits [1]. Such devices generally contain an array of programmable cells and interconnection structures. Users can map algorithms onto the array using appropriate software tools and can change the algorithm at any time. The application domain of reconfigurable hardware has broadened in recent years to include digital signal processing (DSP) [2]. However, DSP is computationally intensive: common algorithms such as the fast Fourier transform (FFT) apply a number of multiplications and additions to a large data set. Applications thus require sophisticated hardware architectures to take advantage of the performance acceleration over conventional microprocessors.

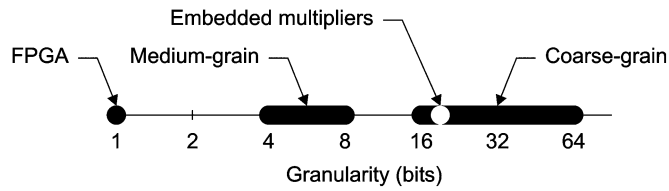


Fig. 1. Granularity of reconfigurable hardware.

Designing reconfigurable hardware for DSP involves basic tradeoffs between performance, area, and flexibility. Traditional devices such as field-programmable gate arrays (FPGAs) place little functionality in each cell. These fine-grain cells can be combined into many different modules but require a complex interconnection structure. Now, multiplication is perhaps the most critical operation that determines the overall performance of DSP. However, mapping a multiplier onto an array of fine-grain cells yields a complex structure that achieves poor performance. For this reason, many FPGAs contain a number of embedded multipliers of fixed word length, such as 18 b.

Extending this idea, researchers have proposed new reconfigurable architectures in which each cell performs 16-b or 32-b operations [3]. For example, the RaPiD architecture contains a linear array of 16-b functional units [4]. The KressArray family features a matrix of 32-b processing units with a hierarchical interconnection structure [5]. The Pleiades architecture combines a microprocessor with a heterogeneous set of components, such as ALUs, memory modules, and an embedded FPGA [6]. The MONTIUM architecture integrates a microprocessor with an FPGA and an array of coarse-grain cells, each containing several 16-b ALUs [7]. These coarse-grain cells achieve high performance, but their functionality typically is limited to a set of basic functions such as multiplication and addition.

Midway between these alternatives are medium-grain architectures. Here, cells work with partial words—typically 4 or 8 b—and are grouped together to implement full-scale operations. Fig. 1 compares the granularity of this approach with fine-grain and coarse-grain architectures. To balance performance

Manuscript received September 15, 2005; revised November 29, 2006. This work was supported in part by the Boeing Endowed Chair at the School of Electrical Engineering and Computer Science, Washington State University, Pullman. The work of M.J. 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 number DE-AC05-06OR23100. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of DHS, DOE, or ORAU/ORISE.

M. J. 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 Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752 USA (e-mail: jdelgado@eecs.wsu.edu).

Digital Object Identifier 10.1109/TCSI.2007.895384

and flexibility, medium-grain cells must implement a variety of functions while keeping the overhead as small as possible. Our research has focused on designing reconfigurable hardware that uses the same circuitry to perform essential DSP operations: multiplication, addition, bit shifting, memory access, and control logic.

In previous research, we have developed a novel two-level architecture in which each cell contains a number of smaller processing elements [8], [11]. Each element is simply a small lookup table (LUT). The main difference between this architecture and FPGAs is that each cell can only assume two structures. In memory mode, the elements collectively implement a random-access memory. This functionality is useful for specifying constant coefficients and storing intermediate results. In mathematics mode, the elements act as LUTs to perform functions such as multiply-accumulates (MACs). A hierarchical interconnection structure combines the cells into modules, and then connects the modules to implement DSP algorithms [8], [12].

This paper presents two medium-grain cells based on the two-level reconfigurable architecture. Section II describes a parallel cell that uses a 4×4 matrix of elements to perform 4-b operations. Section III shows the VLSI implementation of the basic element. Section IV describes an alternative cell that uses five elements to compute binary arithmetic in a bit-serial fashion. Section V gives an overview of a hierarchical interconnection structure that can connect the cells together. Section VI analyzes the performance, area, and configuration overhead of the two designs. We also compare the execution time of several DSP algorithms to conventional microprocessors and FPGAs. Finally, Section VII concludes the paper.

II. Parallel Cell

This section describes the parallel cell that contains a 4×4 matrix of elements. The design is similar to the cell presented in [8] and [11], but does implement more functionality. After illustrating the two modes of operation, we discuss how the cell can perform binary arithmetic, bit shifting, and control logic. We also give a layout simulation validating the design.

A. Memory Mode

In memory mode, shown in Fig. 2, the matrix of elements implements a 128×4-b RAM. The system can read from one address and write to another address simultaneously. Inputs $ra_{0:3}$ and $ra_{4:7}$ specify the 7-b read address, with ra_7 serving as the read enable. Similarly, $wa_{0:3}$ and $wa_{4:7}$ specify the write address and write enable. Each element manages a 32×1-b portion of the memory. A decoder generates the required control signals for each row of elements. Lines $wi_{0:3}$ and $ro_{0:3}$ are the input data and output data, and $ri_{0:3}$ serves as the default value of $ro_{0:3}$.

The parallel cell runs at a frequency determined by a global clock *clk*. The cell may use this signal to isolate write operations from input transitions at the beginning of the clock cycle. Read operations do not require a clock.

Multistage DSP algorithms can use memory cells to store intermediate data. Having separate read and write ports allows the algorithm to load data from memory, process each sample, and store the results back into memory in pipelined fashion. Cells in memory mode can also implement an LUT or specify simple logic functions. For example, a cell could generate the control signals for a multiplexer.

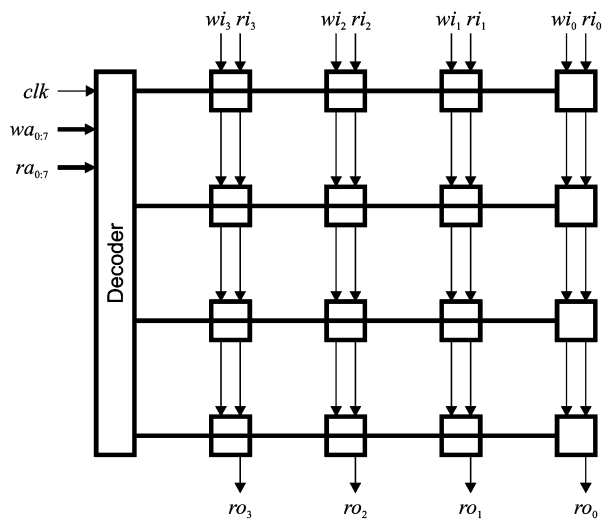


Fig. 2. Parallel cell in memory mode.

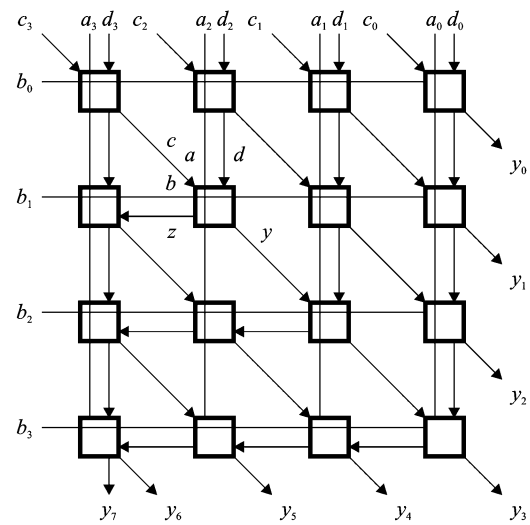


Fig. 3. Parallel cell in mathematics mode.

B. Mathematics Mode

In mathematics mode, shown in Fig. 3, the cell reuses the elements to implement binary arithmetic. The matrix of elements now assumes a structure resembling a carry-save multiplier [14]. In fact, the structure is optimized for the MAC function

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

However, the matrix of elements can implement many other operations as well. The critical path through the structure consists of seven elements. Observe that the cell does not require a clock.

Each element now acts as a 16×2-b LUT. The system defines the contents of the LUTs during reconfiguration. For example, let a, b, c, and d be the inputs to the element and y and z the

TABLE I
EXAMPLES OF CELL OPERATIONS

Operation	Mode	Remarks
MAC	Mathematics	Unsigned/two's-complement
Add/Subtract	Mathematics	Unsigned/two's-complement
Shift	Mathematics	Right or left
Logic	Both	AND-OR, decoders, multiplexers
RAM	Memory	Can read and write simultaneously
Lookup table	Memory	128 words per cell
Reconfiguration	Memory	128 write operations

outputs. To perform the 4-b MAC above on unsigned data, each element calculates

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

The LUT give the architecture great flexibility. The cell can perform the same 4-b MAC on two's-complement data with the following element configurations:

$$\begin{aligned}(2z + y) &= (a \times b) + c + d\\ (2z - y) &= (a \times b) + c - d\\ (2z - y) &= (a \times b) - c + d\\ (-2z + y) &= (a \times b) - c - d.\end{aligned}$$

Further details about multiplication appear in [15].

C. Functionality

The matrix of elements can perform many other operations besides memory access and multiplication. For example, the 4-b MAC reduces to addition when $b_{0:3}$ is fixed at 1. Thus, the system can implement a 4-b adder by configuring the top row of elements to evaluate

$$(2y + z) = a + c + d,$$

and all other elements to evaluate

$$(2y + z) = c + d.$$

Similarly, the two's-complement MAC function reduces to subtraction when $b_{0:3}$ is fixed to -1.

Another operation that appears in DSP is bit shifting. A cell in mathematics mode can implement a 4-b shifter that works with inputs $c_{0:3}$ and $d_{0:3}$ [8]. The elements can shift $c_{0:3}$ left and shift in $d_{0:3}$ or, equivalently, shift $d_{0:3}$ right and shift in $c_{0:3}$. Inputs $a_{0:3}$ and $b_{0:3}$ control the operation of the shifter.

DSP algorithms are not composed exclusively of binary arithmetic, but also require some control logic for proper operation. The parallel cell can evaluate basic logic expressions in both modes. In mathematics mode, the 16×2-b memory inside each element can define two equations of up to four variables. The system can configure the first row of elements to implement the desired function, and have the remaining elements pass the results to the outputs without modification. In memory mode, the 128×4-b memory can act as a large LUT. Possible functions include AND–OR expressions, decoders, and multiplexers.

Table I lists some examples of cell operations. All of these functions are possible with suitable configurations of the elements. Unlike other reconfigurable architectures, the cells do not use a separate datapath for reconfiguration. Instead, cells default to memory mode so the system can load new data into the elements. This process requires a sequence of 128 write operations, although the system can configure multiple cells simultaneously to save time.

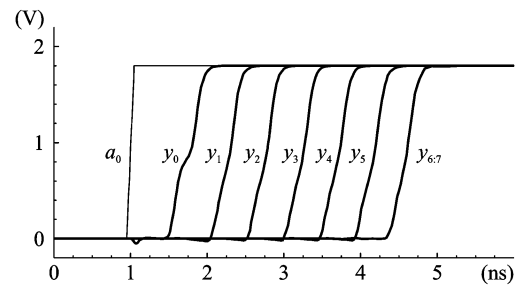


Fig. 4. Worst case simulation of parallel cell.

D. Simulation

We implemented the parallel cell in 180-nm technology and performed layout simulations with all parasitic capacitances extracted. Fig. 4 contains a simulation of the matrix of elements during a worst case operation. This operation occurs in mathematics mode when the calculated result consists of all ones. Bit y_0 evaluates first, followed by y_1, y_2, and so forth. The last element generates y_6 and y_7 simultaneously.

Recall that the parallel cell does not require a clock for mathematics mode. The propagation delay through the cell, measured from the input data to $y_{6:7}$, is slightly less than 3.75 ns. Hence, the parallel cell can run at 267 MHz.

III. ELEMENT IMPLEMENTATION

A notable feature of the parallel cell, as well as the serial cell described later, is the absence of function units such as adders. Both designs use the same basic element to perform computations and store data. The element itself contains an array of data latches with some simple glue logic. This strategy leads to a compact very large-scale integration (VLSI) implementation that achieves high performance.

A. Organization

Fig. 5 gives a simplified diagram of the reconfigurable element. The structure behaves as a 16×2-b RAM with separate read and write addresses. The memory is organized into a 4×4 array of 2-b latches. Row decoders controlled by the lower address bits enable latches for reading and writing. Column decoders and selectors controlled by the upper address bits connect the appropriate latches to the inputs and outputs. Notice that the read address $ra_{0:3}$ doubles as the bits a, b, c, and d in mathematics mode.

B. Read Operations

Fig. 6 depicts the datapath of the element for read operations. Computations in mathematics mode use this circuitry as well. Each 2-b latch produces a pair of complementary outputs. The

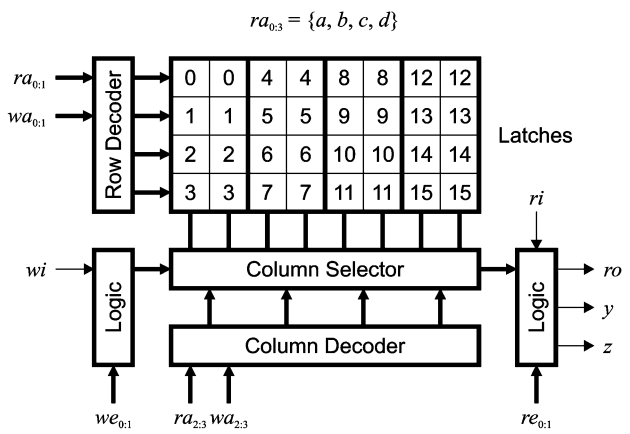


Fig. 5. Organization of element (simplified).

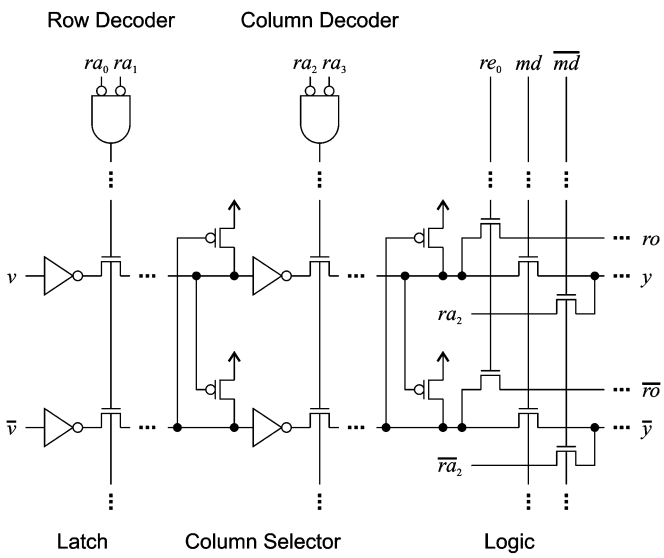


Fig. 6. Datapath for read operations.

lower bits v-$\overline{v}$ propagate through a series of n-transistors controlled by the row and column decoders. Weak cross-coupled p-transistors restore the rail-to-rail voltage swing. Data from the selected latch ultimately drives the outputs y-$\overline{y}$. The upper bits of the selected latch, not shown in the figure, connect to outputs z-$\overline{z}$ in a similar fashion. The static signal md is logic 1 for mathematics mode and logic 0 for memory mode.

In mathematics mode, the y-$\overline{y}$ and z-$\overline{z}$ lines set the c-$\overline{c}$ and d-$\overline{d}$ inputs of the next elements. We have optimized the read datapath to perform mathematics computations quickly, as this operation determines the speed of both medium-grain cells. Using complementary logic allows y-$\overline{y}$ to drive the column decoders of the next elements without additional inverters. In addition, the intermediate buffers in the datapath allow data to propagate halfway through the element when a and b become available. This operation is useful because the lower address bits a and b typically resolve before the upper address bits c and d. We explored several clock strategies for the read datapath, including a domino-logic scheme [11], but settled on the current design to eliminate the time and power overhead of precharge logic.

In memory mode, the element connects one of the latches to the output data ro-$\overline{ro}$, as gated by the read enable $re_{0:1}$. Outputs

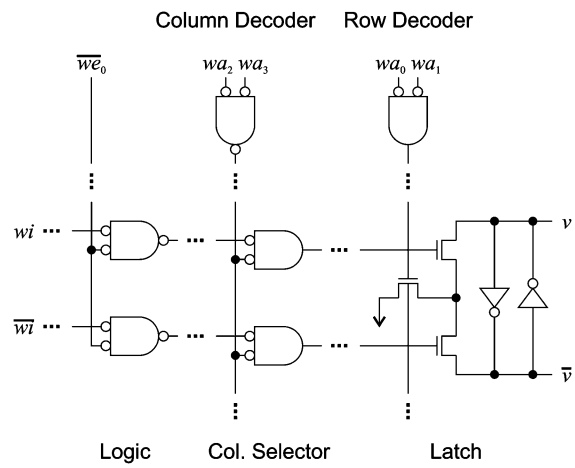


Fig. 7. Datapath for write operations.

y-$\overline{y}$ and z-$\overline{z}$ are set to the incoming $ra_{2:3}$-$\overline{ra}_{2:3}$ bits so that all elements receive the same input address.

C. Write Operations

Write operations use a separate datapath, outlined in Fig. 7. Each bit of the 2-b latch uses a pair of minimum-size inverters to store the logic value. A network of n-transistors implements the set and reset circuitry. The complementary data inputs both revert to low unless the latch is enabled for writing. This condition only occurs if the column decoder and write enable input $we_{0:1}$ both permit the operation. The datapath uses NOR gates to combine the enable signals with the input data wi-$\overline{wi}$.

IV. Serial Cell

Computing binary arithmetic in bit-serial fashion greatly reduces the circuit complexity—particularly for multiplication, which forms the bottleneck of most DSP algorithms. Bit-serial designs also require fewer data lines and hence fewer interconnection resources on reconfigurable hardware. Recognizing these benefits, researchers have proposed reconfigurable architectures that perform bit-serial computations. Both fine-grain [16], [17] and coarse-grain [18] devices appear in the literature. The drawback of these designs is that the entire device runs off a single clock signal. Aside from the problem of clock distribution, the interconnection structures within the critical path limit the maximum clock frequency and hence the processing speed.

The serial cell described in this section actually uses a serial-parallel approach. All inputs are collected in parallel at the beginning of each clock cycle. The cell then applies the inputs to an array of five elements and computes the results in serial fashion. The outputs are collected and passed to the interconnection structure. Unlike other architectures, the serial processing stages are confined within the cell, enabling the computations to run significantly faster. The description here parallels the discussion in [19].

A. Memory Mode

Like the parallel design, the serial cell can operate in memory mode or mathematics mode. Both alternatives use the elements described in Section III. Fig. 8 depicts the cell in memory mode.

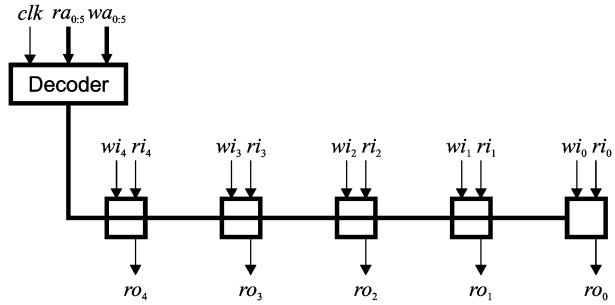


Fig. 8. Serial design in memory mode.

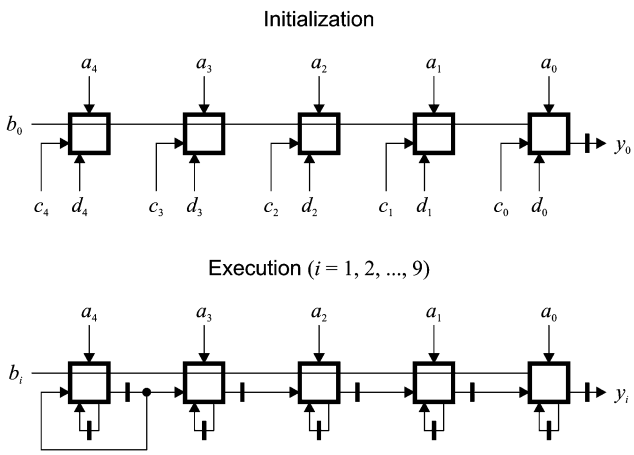


Fig. 9. Serial design in mathematics mode.

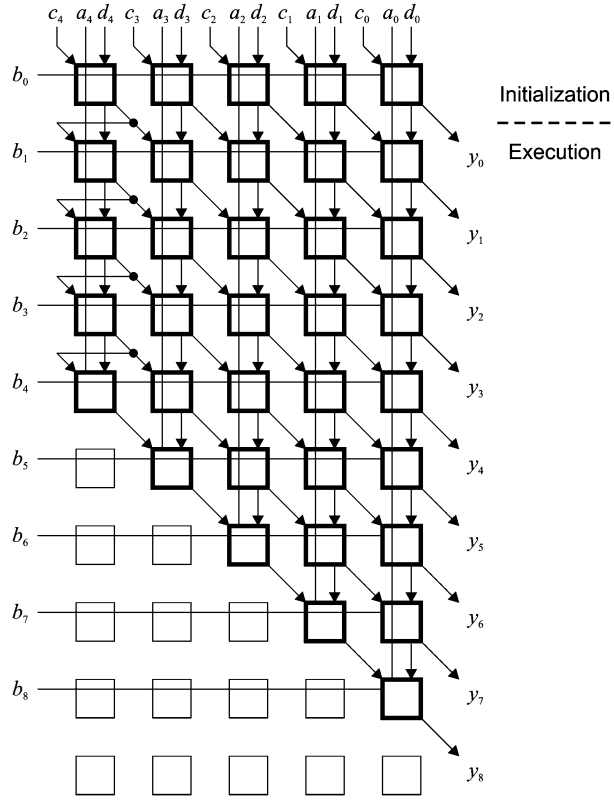


Fig. 10. Equivalent parallel model of mathematics mode.

The array of elements implements a 32-word RAM with separate read and write addresses. As with the parallel cell, this capability helps multistage DSP algorithms work with large datasets. Each element manages a 32×1-b piece of the total memory. With five elements, the memory works with 5-b words rather than 4-b words. The motivation behind this additional element will become clear shortly.

In the figure, $ra_{0:4}$ and $wa_{0:4}$ contain the read address and write address, respectively. Bits ra_5 and wa_5 specify the read and write enable. A decoder uses these inputs to generate the control signals for the five elements. The data lines include the input $wi_{0:4}$ and output $ro_{0:4}$.

B. Mathematics Mode

Fig. 9 illustrates the cell in mathematics mode. The array of elements now operates in bit-serial fashion with inputs $a_{0:4}$, b_i, $c_{0:4}$, and $d_{0:4}$. Input b_i as well as output y_i have bit-serial format. Like the parallel cell, each element acts as a 16×2-b LUT.

Each computation consists of an initialization stage and nine execution stages. During initialization, the cell applies the initial values of the inputs to the elements. During execution, the cell takes away the $c_{0:4}$ and $d_{0:4}$ inputs and connects the elements into the structure shown. The elements then run for nine cycles to produce the output $y_{0:8}$. Pipeline latches, denoted by hash marks in the figure, separate data from adjacent cycles.

To clarify how the cell operates in mathematics mode, Fig. 10 expands the chain of elements into an equivalent parallel model. Each row of elements represents one stage. Observe that the structure resembles a linear shifter or an array multiplier. The cell collects the first output y_0 when $i = 1$, and the final output y_8 when $i = 9$. The light-colored elements in the final stages do not contribute to the result.

C. Functionality

Although the expanded model of mathematics mode has a different structure than the parallel cell, the serial cell can still implement the same functionality. For example, the structure can perform the 4-b MAC

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

As before, each element evaluates the expression

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

The cell actually works with 5-b and 9-b words rather than 4-b and 8-b words. This feature is necessary when implementing a MAC that works with a mixture of unsigned and two's-complement data, a situation that occurs when combining cells together to form larger multipliers [15]. The extra bits prevent output overflow. As a side effect, all elements evaluate the same expression for both unsigned and two's-complement MAC.

The serial cell can perform addition and subtraction with the same LUTs by setting $b_{0:3}$ to a constant value of 1 or -1. The cell allows the system to specify this value during reconfiguration.

In addition, the bit-serial methodology of mathematics mode is ideal for implementing a shifter. This function only requires

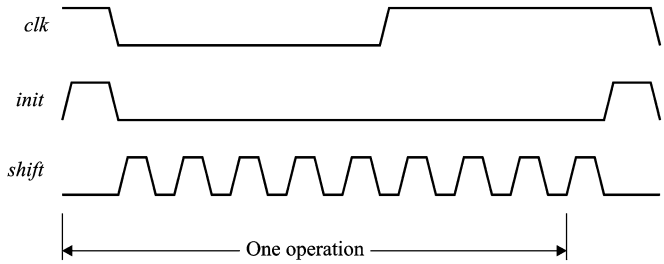


Fig. 11. Clock waveforms for mathematics mode.

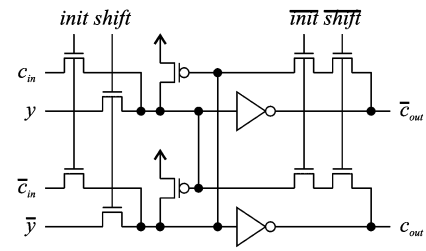


Fig. 12. Pipeline latch used in serial cell.

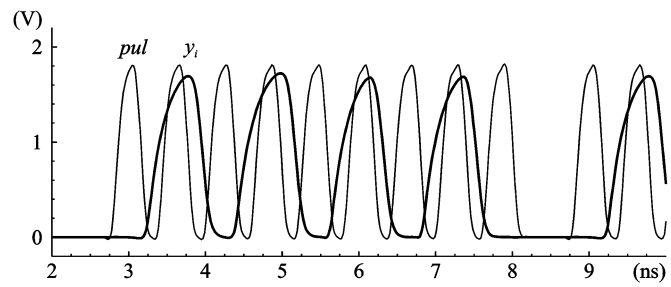


Fig. 13. Worst case simulation of serial cell.

the $a_{0:4}$ and $b_{0:8}$ inputs. Each element is configured to act as a multiplexer controlled by the a input. When $a = 1$, the element passes the value of b_i diagonally to the outputs. When $a = 0$, the element uses the output of the previous element instead. With appropriate values of $a_{0:4}$, the cell can copy any 5-b substring of $b_{0:8}$ to $y_{4:8}$. One can view this operation as shifting $b_{4:8}$ left or shifting $b_{0:4}$ right.

The simplest way for the serial cell to implement control logic is to specify a LUT in memory mode. Recall that the array of elements can act as a 32×5-b memory. Hence, one cell can implement any combinational logic function with up to five inputs and five outputs.

The serial cell also uses memory mode to load the initial state of the elements during reconfiguration. One advantage over the parallel design is that most mathematics functions use the same LUTs for each element. This property allows the system to write to all five elements simultaneously, saving time. In all, the system must specify around 42 configuration bits for mathematics functions: 32 for the LUTs, 1 for the mode selection, and 9 for the initial value of $b_{0:8}$.

D. Clock Circuitry

The serial cell must generate several control signals to drive computations in mathematics mode. Like the parallel design, the cell is driven by a global clock *clk*. However, the cell must subdivide *clk* into the *init* and *shift* signals shown in Fig. 11. The *init* signal governs the initialization stage, and *shift* demarcates the nine execution stages. Note that initialization occurs at the end of the clock period, so that each operation spans part of two cycles. This feature allows *clk* to be held high for testing.

Generating *init* and *shift* locally allows the serial cell to achieves higher performance than other bit-serial reconfigurable architectures. The frequency of *shift* does not depend on the propagation delay between cells, but rather the latency of the elements in mathematics mode. In addition, the system does not have to distribute a fast clock to every cell. As described in [20], the clock generator contains a dual ring oscillator to generate complementary outputs $init$-$\overline{init}$ and $shift$-$\overline{shift}$. The oscillator uses cross-coupled p-transistors to ensure that the outputs are symmetric.

The control signals drive a series of pipeline latches to separate adjacent execution stages. Fig. 12 depicts the design of one pipeline latch. This circuit replaces the output logic in the read datapath, shown in Fig. 6. Using pipeline latches rather than pipeline registers maximizes the performance of the serial cell, but does impose a minimum frequency on *shift*. We present an analysis and performance comparison of the circuit in [21].

E. Simulation

A layout simulation of the serial cell in 180-nm technology appears in Fig. 13. The cell has been configured to perform the 4-b MAC. In the simulation, the elements perform the following calculation:

$$(01111 \times 01010) + 01010 + 01010 = 010101010.$$

These values represent a worst case situation, since the output y_i alternates between logic 0 and logic 1. As shown, the elements produce the correct output. The signal *shift* runs at approximately 2 GHz; the master clock *clk* has a frequency of 167 MHz.

V. Interconnection Network

To map DSP algorithms onto the medium-grain reconfigurable architecture, one can imagine the array of cells as being partitioned into blocks of various sizes. Each block implements 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 forms of data transfer [12]. A local mesh connects neighboring cells, and a global H-tree routes data between modules. We recently enhanced the design to support high data rates and simplify the mapping process [22]. The resulting interconnection network, summarized in this section, represents one possible method for connecting the proposed cells. The structure can transfer data both in bit-parallel or bit-serial fashion; both designs are compatible with either alternative.

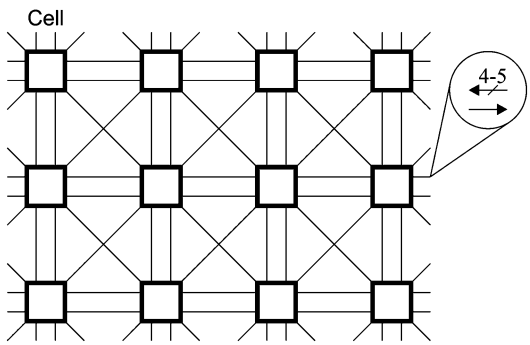


Fig. 14. Local mesh.

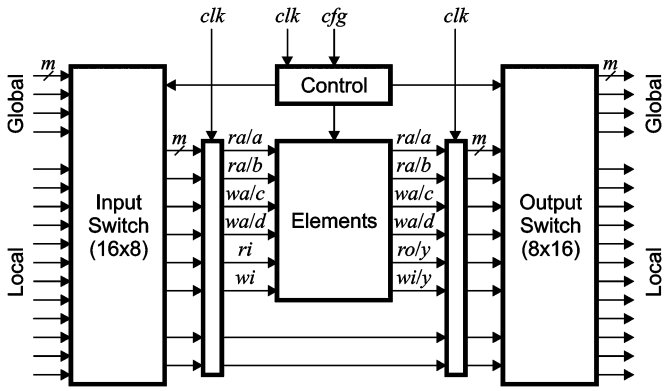


Fig. 15. Interface between cell and interconnection network.

A. Local Network

Fig. 14 depicts the local mesh. A series of busses running horizontally, vertically, and diagonally allow cells to exchange data with their eight neighbors. Each bus contains m bits, where m could equal 4 b for the parallel cell, 5 b for the serial cell, or 1 b for a bit-serial implementation.

Fig. 15 shows the interface between the cell and the interconnection network in more detail. Two crossbar switches route the inputs and outputs of the elements to the correct locations. A control unit manages the reconfiguration process. Notice that the inputs and outputs of the elements may assume different roles for memory mode and mathematics mode. A large number of outputs are simply copies of the inputs to facilitate passing the same data to multiple cells.

As shown in the figure, the architecture pipelines all operations in the cells and interconnection network. This technique allows the system to achieve high throughput by taking advantage of the data-parallel nature of DSP. The architecture allows one cycle for cell operations, and one cycle for communication over the local mesh. Additional pipeline registers can be placed in the datapath if required.

B. Global Network

Fig. 16 illustrates the global 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 m 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 substantially lower complexity and configuration overhead.

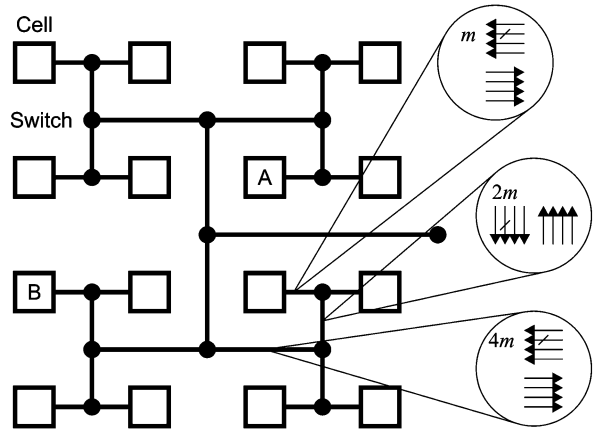


Fig. 16. Global H-tree.

Like the local mesh, the H-tree contains pipeline latches to improve throughput. The total latency between two cells equals half the number of busses traversed on the tree, since each bus represents half a pipeline stage. For example, cells A and B in the figure are separated by four clock cycles.

This pipelined interconnection network allows synthesis tools to implement entire algorithms by connecting modules together. The H-tree naturally collects the outputs of a module onto a single bus. Similarly, the structure can route that bus to the inputs of another module such that each 4-b portion incurs the same latency from end to end. This property holds true for modules of any size and shape, as long as the topmost bus is sufficiently high in the hierarchy. If the algorithm requires additional pipeline delays on a certain bus, the system can simply route that bus to a higher level. More details about the mapping process appear in [8].

Finally, the H-tree plays an important role in the reconfiguration process. The process begins when the system asserts a program signal for the device, or a portion of the device. All switches in the affected area assume a default state to pass data from top to bottom. The system then loads configuration commands into the topmost level of the H-tree, which propagate into the cells along the input busses. Cells are programmed using standard write operations in memory mode, whereas switches are programmed by placing control words on specific data lines. More details about reconfiguration schemes appear in [9], [10].

C. Example

Fig. 17 gives an example of a module mapped onto the interconnection network. This module implements a 16-b fixed-point multiplier with inputs A and B and output Y. Each cell performs the 4-b MAC described previously. The local mesh handles data transfer between cells. The global H-tree applies the 4-b portions of A to the top row of cells, and the portions of B to the right column. The seven cells marked with Y generate the 4-b portions of the output (the last cell generates two). The H-tree collects these outputs and passes them to the next module. The

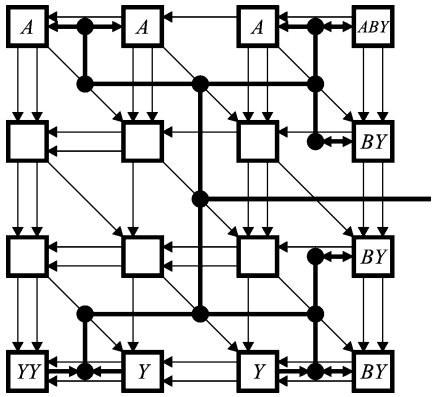


Fig. 17. 16-b fixed-point multiplier.

TABLE II
COMPARISON OF MEDIUM-GRAIN CELLS

Parameter	Parallel	Serial
Number of elements	16	5
Granularity (bits)	4	4 or 5
Memory per cell (words)	128	32
Memory configuration bits	513	161
Math configuration bits	513	42
Configuration latency (cycles)	129	33
Area (mm^2)	0.040	0.017
Clock frequency (MHz)	267	167

interconnection network pipelines the execution process so that the module can initiate one operation per clock cycle.

More examples of modules appear in [8], [11], and [12].

VI. ANALYSIS

This section analyzes the performance and flexibility of the proposed medium-grain cells. After comparing the two alternatives with each other, we show how several DSP benchmarks can be mapped onto the array of cells. We then compare the execution times of these benchmarks with other implementations.

A. Comparison of Cells

Table II summarizes the performance and overhead of the two cells. The area and clock frequency were taken from the 180-nm layouts. Clearly, the parallel design achieves higher speed at the expense of area.

One advantage of the serial design is the reduced number of configuration bits required for mathematics operations. As described in Section IV, the system can load the same LUT into all five elements. In contrast, the system may have to configure each element separately in the parallel cell. Each element stores 32 b. The number of configuration bits includes 1 b to select memory mode or mathematics mode and 9 b for the serial cell to set one input in mathematics mode. Although the configuration latency seems high, the system can easily program multiple cells at the same time.

Having a larger amount of memory per cell does offer an advantage when implementing large memory units. DSP algorithms that work with large datasets would need fewer cells to implement a memory unit of the same size. For all other operations, the two cells essentially implement the same functionality.

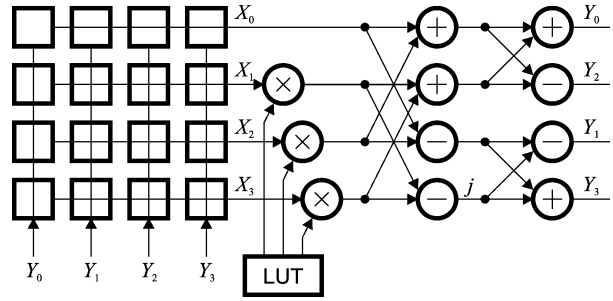


Fig. 18. Diagram of radix-4 FFT.

TABLE III
PERFORMANCE OF DSP BENCHMARKS

Parameter		Parallel	Serial
FIR:	Cells	256	256
	Cycles	316	316
	Area (mm^2)	30.7	13.1
	Time (μs)	1.18	1.90
FFT:	Cells	512	916
	Cycles	524	588
	Area (mm^2)	61.4	46.7
	Time (μs)	1.96	3.52
CORDIC:	Cells	512	512
	Cycles	313	313
	Area (mm^2)	61.4	26.1
	Time (μs)	1.17	1.88

B. Interconnection Structure

We used the interconnection network described in Section V as a model to permit further analysis of the proposed medium-grain cells in a practical setting. We should emphasize that the cells could be integrated with many other types of interconnection structures without affecting the flexibility of the reconfigurable platform. The proposed interconnection network does work well with the proposed cells, allowing fast communication within modules and word-length communication between modules. In addition, the pipelined design maintains the throughput of the cells. We include the latency of the interconnection network when estimating the execution time of DSP algorithms.

To enable comparisons with other devices, the following discussion makes two assumptions about the interconnection network: that the area accounts for two thirds of the total area on the device, and the clock frequency matches the frequency of the basic cell. Preliminary layouts of the H-tree switches indicate that the proposed design meets these criteria [13].

C. DSP Benchmarks

To evaluate the performance and flexibility of the proposed cells, we have mapped several DSP benchmarks onto the medium-grain architecture. These benchmarks include a 12-tap FIR filter, an FFT, and a CORDIC execution stage. We then estimated the number of cycles required to process a 16-b, 256-point dataset. We have developed a suite of CAD tools to assist in the mapping and simulation process. At this point, the mapping process consists of manual placement and routing of the modules required by the algorithm.

To illustrate one example, Fig. 18 gives a diagram of the FFT. We used a radix-4 decomposition for high performance. Each stage in the algorithm requires the system to load four samples from memory, compute a "dragonfly" operation, store the four

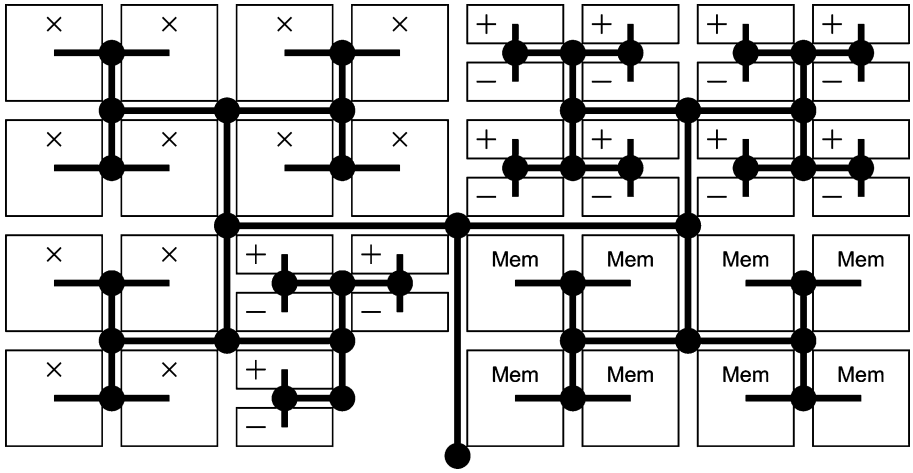


Fig. 19. Implementation of FFT on parallel architecture.

TABLE IV
COMPARISON WITH DSPS

Parameter		ADSP-BF533	TMS320C2	TMS320C4	Parallel	Serial
Technology (nm)			150	90	180	180
Clock (MHz)		750	300	1000	267	167
FIR:	Time (μs)	2.39	9.02	1.29	1.18	1.90
	Scaled time		10.82	2.58	1.18	1.90
FFT:	Time (μs)	3.10	9.25	1.24	1.96	3.52
	Scaled time		11.10	2.48	1.96	3.52

intermediate results back into memory, and repeat for all points in the dataset. Four stages are required for the 256-point dataset. The dragonfly requires three complex multiplications and eight complex additions or subtractions.

Fig. 19 depicts the implementation of the FFT on the parallel architecture. The three complex multipliers each break down into four real multipliers, one real adder, and one real subtracter. (Section V depicts one real multiplier.) A sophisticated memory unit allows the system to read four 16-b inputs and write four 16-b results simultaneously. The implementation occupies a 32×16 block of cells, not including a separate LUT to generate the "twiddle factors" needed for the dragonfly stage. On the serial architecture, the memory unit takes up more space since memory cells have a smaller capacity.

Table III lists the estimated area and execution time of the three benchmarks. Following the assumptions about the interconnection network, the given area is three times the area consumed by the cells alone. The execution time includes the latency caused by the cells and the interconnection network.

D. Performance Comparison

Table IV compares the execution times of the finite-impulse-response (FIR) filter and FFT to the reported results for several fixed-point digital signal processors. 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 processors (TMS320C2 and TMS320C4) from Texas Instruments. The execution times for these platforms were calculated from data available at the manufacturers' websites. 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-b fixed-point data. The time required to load the data into memory is not included.

The medium-grain architecture based on the parallel cell surpasses the performance of digital signal processors in similar technologies. The serial cell also achieves good performance, although the execution time of the FFT lags behind the parallel cell due to the greater area of the structure and the resulting interconnection latency. Overall, the performance increase over digital signal processors is not surprising, since reconfigurable hardware can exploit the parallelism of DSP.

Table V 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-b multipliers, and in the case of the Virtex-4, 48-b adders as well. The clock frequencies for the FIR and CORDIC benchmarks were calculated by mapping the respective IP cores using maximum pipelining and the highest effort level. The results for the FFT were taken directly from the datasheet for the Fast Fourier Transform IP core using a radix-4, burst-mode algorithm with the data load time subtracted.

The 90-nm Virtex-4 does outperform the proposed architecture, but the gap narrows significantly with the 150-nm Virtex-II. We expect the clock rates of the proposed cells to increase significantly in 90-nm technology. Preliminary circuit simulations indicate that the parallel cell can run at speeds over 1 GHz.

Table V 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

TABLE V
COMPARISON WITH RECONFIGURABLE HARDWARE

Parameter		XC2V500	XC2VP4	XC4VSX25	Parallel	Serial
Technology (nm)		150	130	90	180	180
Slices / cells		3,072	6,768	10,240	4,096	4,096
Total area (mm^2)		384	384	729	492+	209+
FIR:	Clock (MHz)	218	258	323	267	167
	Utilization	0.32	0.27	0.05	0.06	0.06
	Time (μs)	1.27	1.07	0.87	1.18	1.90
	Scaled time	1.52	1.48	1.74	1.18	1.90
FFT:	Clock (MHz)	232	293	458	267	167
	Utilization	0.52	0.23	0.12	0.13	0.22
	Time (μs)	1.34	1.06	0.72	1.96	3.52
	Scaled time	1.61	1.47	1.44	1.96	3.52
CORDIC:	Clock (MHz)	190	198	299	267	167
	Utilization	0.20	0.20	0.06	0.13	0.13
	Time (μs)	1.45	1.39	0.92	1.17	1.88
	Scaled time	1.74	1.92	1.84	1.17	1.88

number of slices in the device. For the proposed architectures, we assume that the target device has a 64×64 array of cells. This size resulted in active areas of 492 mm^2 for the parallel architecture and 209 mm^2 for the serial architecture. Even accounting for the packaging overhead, the areas seem comparable to the package sizes of the given Xilinx devices.

Although direct comparison of the utilization 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. As mentioned previously, the area estimations for the proposed architectures do depend on the assumption that interconnection network accounts for two thirds of the total area.

E. Architecture Comparison

On the architectural level, both medium-grain cells offer several advantages over other DSP hardware.

- *High flexibility.* The same cells can implement all basic DSP operations, including MACs, bit shifting, control logic, and data storage. The elements within the cell allow a great deal of fine-grain flexibility. For example, a multiplier can work with data in two's-complement format, or an adder can apply a specific rounding method to the output. The 4-b cells can be combined together to implement modules of the precise word length required.
- *Low complexity.* Each cell can assume only two structures: memory mode and mathematics mode. Hence, the design requires minimal interconnection resources at the fine-grain level. This parameter becomes especially important for reconfigurable hardware, which typically dedicates the vast majority of the circuit area to interconnection structures.
- *Inherent support for multiplication.* The medium-grain cells can perform a powerful MAC with two addition terms. This function allows the hardware to implement large multipliers that match the speed of the other modules in the design. Unlike FPGAs, the number, word length, and location of these multipliers is only limited by the size of the array. This property makes the proposed architecture a promising choice for cryptography and other applications that use multiplication extensively.
- *Simple clocking scheme.* A key difference between the medium-grain architecture and FPGA platforms is that all algorithms operate at the maximum frequency. With a pipelined interconnection network, the clock rate remains fixed. In contrast, the clock rate in FPGAs varies between algorithms, depending on the complexity of each module as well as the sophistication of the routing tools. The global clock runs at moderate frequency to mitigate clock distribution problems. The serial cell generates all high-frequency control signals locally.

VII. CONCLUSION

In this paper, we have described two novel medium-grain cells for reconfigurable DSP hardware. The designs use small LUTs or "elements" to perform computations. The memory inside the elements can also store intermediate results. The first design, which computes all results in bit-parallel fashion, offers the best performance but consumes a relatively large area. The second design, which computes binary arithmetic in bit-serial fashion, is significantly smaller but incurs a performance penalty. Both alternatives allow cells to be combined into larger modules. We estimated the execution time of various DSP benchmarks and found that our approach achieves comparable performance to digital signal processors and FPGAs in the same technology.

In future work, we plan to migrate the medium-grain cells to 90-nm technology to enable more accurate comparison with current DSP hardware. We will also continue to develop the interconnection network and analyze the execution time of other algorithms. Finally, we will consider ways to reduce the power consumption of the architecture.

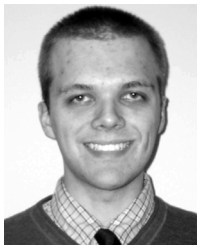
ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers of this paper for their constructive comments and suggestions.

REFERENCES

[1] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, Jun. 2002.

[2] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: A survey," in *Programmable Digital Signal Processors*, Y. Hu, Ed. New York: Marcel Dekker Inc., 2001.
[3] R. Hartenstein, "Coarse grain reconfigurable architectures," in *Proc. 6th Asia South Pacific Design Autom. Conf.*, Yokohama, Japan, 2001, pp. 564–570.
[4] C. Ebeling, D. Cronquist, P. Franklin, and C. Fisher, RaPiD—A Configurable Computing Architecture for Compute-Intensive Applications Univ. of Washington Dept. Comput. Sci. Eng., Seattle, Tech Rep. TR-96-11-03, 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. 2, pp. 1697–1704, Feb. 2000.
[7] P. Heysters and G. Smit, "Mapping of DSP algorithms on the MONTIUM architecture," in *Proc. Int. Parallel Distrib. Process. Symp.*, Apr. 2003, pp. 180–185.
[8] M. Myjak, "A medium-grain reconfigurable architecture for digital signal processing," Ph.D. dissertation, School of Eng. Comput. Sci., Washington State Univ., Pullman, 2006.
[9] A. Widjaja and J. Delgado-Frias, "A high-performance unicast configuration scheme for an H-tree based reconfigurable hardware," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, Cincinnati, OH, Aug. 2005, vol. 2, pp. 1215–1218.
[10] A. Widjaja, "H-tree based configuration schemes for a reconfigurable DSP architecture," M.S. thesis, School of Eng. Comput. Sci., Washington State Univ., Pullman, 2005.
[11] M. Myjak and J. Delgado-Frias, "A two-level reconfigurable architecture for digital signal processing," in *Proc. Int. Conf. VLSI*, Las Vegas, NV, Jun. 2003, pp. 21–27.
[12] M. Myjak, F. Anderson, and J. Delgado-Frias, "H-tree interconnection structure for reconfigurable DSP hardware," in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms (ERSA)*, Las Vegas, NV, Jun. 2004, pp. 170–176.
[13] J. Larson and K. Prajapati, VLSI Implementation of an H-Tree Switch Node Washington State University, 2005 [Online]. Available: http://www.eecs.wsu.edu/~jdelgado/reports/Larson_Prajapati.pdf
[14] J. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Upper Saddle River, NJ: Pearson Education, 2003, pp. 591–592.
[15] M. Myjak and J. Delgado-Frias, "Pipelined multipliers for reconfigurable hardware," in *Proc. 11th Reconfigurable Architectures Workshop*, Santa Fé, NM, Apr. 2004, pp. 150–154.
[16] T. Isshiki *et al.*, "High density bit-serial FPGA with lut embedding shift register function," in *Proc. Asia-Pacific Conf. Circuits Syst.*, Oct. 2002, vol. 1, pp. 475–480.
[17] S. A. Rahim and L. E. Turner, "A field programmable bit-serial digital signal processor," in *Proc. IEEE Int. Workshop Syst.-on-Chip for Real-Time Applications*, Jul. 2004, pp. 295–298.
[18] A. Tisserand, P. Marchal, and C. Piguet, "An on-line arithmetic based FPGA for low-power custom computing," in *Proc. 9th Int. Workshop on Field Programmable Logic Applicat.*, London, England, Sep. 1999, vol. 1673, LNCS, pp. 264–273.
[19] M. Myjak and J. Delgado-Frias, "A bit-serial cell for reconfigurable DSP hardware," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, Cincinnati, OH, Aug. 2005, pp. 960–963.
[20] ——, "A symmetric differential clock generator for bit-serial hardware," in *Proc. Conf. Comput. Design*, Las Vegas, NV, Jun. 2005.
[21] M. J. Myjak, J. G. Delgado-Frias, and S. K. Jeon, "An energy-efficient differential flip-flop for deeply pipelined systems," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, San Juan, Puerto Rico, Aug. 2006.
[22] M. Myjak and J. Delgado-Frias, "Superpipelined reconfigurable hardware for DSP," in *Proc. IEEE Int. Symp. Circuits Syst.*, Kos, Greece, May 2006, pp. 3670–3673.



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

He is currently 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) at 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 has coauthored over 150 journal and conference papers and coedited four books. He holds over 25 patents.

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