MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications

Hartej Singh, *Member*, *IEEE*, Ming-Hau Lee, *Student Member*, *IEEE*,
Guangming Lu, *Member*, *IEEE*, Fadi J. Kurdahi, *Member*, *IEEE*,
Nader Bagherzadeh, *Senior Member*, *IEEE*, and Eliseu M. Chaves Filho, *Member*, *IEEE*

Abstract—This paper introduces *MorphoSys*, a reconfigurable computing system developed to investigate the effectiveness of combining reconfigurable hardware with general-purpose processors for word-level, computation-intensive applications. *MorphoSys* is a coarse-grain, integrated, and reconfigurable system-on-chip, targeted at high-throughput and data-parallel applications. It is comprised of a reconfigurable array of processing cells, a modified RISC processor core, and an efficient memory interface unit. This paper describes the MorphoSys architecture, including the reconfigurable processor array, the control processor, and data and configuration memories. The suitability of MorphoSys for the target application domain is then illustrated with examples such as video compression, data encryption and target recognition. Performance evaluation of these applications indicates improvements of up to an *order of magnitude* (or more) on MorphoSys, in comparison with other systems.

Index Terms—Reconfigurable systems, reconfigurable cell array, Single Instruction Multiple Data, dynamic reconfiguration, target recognition, bit-correlation, multimedia applications, video compression, MPEG-2, data encryption.

1 Introduction

Reconfigurable systems are computing systems that combine a reconfigurable hardware processing unit with a software-programmable processor. These systems allow customization of the reconfigurable processing unit in order to meet the specific computational requirements of different applications. Reconfigurable computing represents an intermediate approach between the extremes of Application Specific Integrated Circuits (ASICs) and general-purpose processors. A reconfigurable system generally has wider applicability than an ASIC. In addition, the combination of a reconfigurable component with a general-purpose processor results in better performance (for many application classes) than the general-purpose processor alone.

The significance of reconfigurable systems can be illustrated through the following example. Many applications have a heterogeneous nature and comprise of several subtasks with different characteristics. For instance, a multimedia application may include a data-parallel task, a bit-level task, irregular computations, high-precision word operations and a real-time component. For such complex applications with wide-ranging subtasks, the ASIC approach would lead to an uneconomical die size or a large number of separate chips. Also, most general-purpose processors would very likely not satisfy the performance constraints for the entire application. However, a reconfigurable system (that combines a reconfigurable component with a mainstream microprocessor) may be optimally reconfigured for each subtask, meeting the application constraints within the same chip. Moreover, it would be useful for more general-purpose applications, too.

This paper describes **MorphoSys**, a novel model for reconfigurable computing systems, targeted at applications with inherent data-parallelism, high regularity, and high throughput requirements. Some examples of these applications are video compression (discrete cosine transforms, motion estimation), graphics and image processing, data encryption, and DSP transforms.

The *MorphoSys* architecture, shown in Fig. 1, is comprised of a reconfigurable processing unit, a general-purpose (core) processor, and a high-bandwidth memory interface, all implemented as a single chip. Given the nature of target applications, the reconfigurable component is organized in SIMD fashion as an *array of Reconfigurable Cells* (RCs). Since most of the target applications possess word-level granularity, the RCs are also coarse-grain. The core (RISC) processor controls the operation of the Reconfigurable Cell Array (RC Array). The high-bandwidth *data interface* consists of a specialized streaming buffer (and controller) to handle data transfers between external memory and the RC Array.

The intent of the *MorphoSys* implementation is to study the viability of this integrated reconfigurable computing model to satisfy the increasing demand for low cost stream

• *H. Singh, M.-H. Lee, G. Lu, N. Bagherzadeh, and F.J. Kurdahi are with the Department of Electrical and Computer Engineering, University of California, Irvine, Irvine, CA 92697.*
E-mail: {hsingh, mlee, glu, nader, kurdahi}@ece.uci.edu.
• *E.M. Chaves Filho is with the Department of Systems and Computer Engineering, COPPE/Federal University of Rio de Janeiro, PO Box 68511, 21945-970 Rio de Janeiro, RJ Brazil.*

Manuscript received 4 Feb. 1999; accepted 6 Mar. 2000.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 109134.

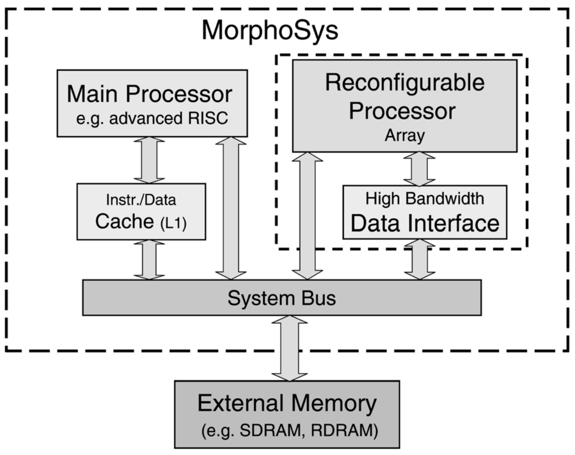

Fig. 1. An integrated architecture for reconfigurable processor systems.

or frame data processing needed for important application classes, like video and image processing, multimedia, digital signal processing, and data encryption.

Organization of paper: Section 2 provides brief explanations of some terms and concepts used frequently in reconfigurable computing, along with a tabular review of previous contributions in this field. Section 3 introduces the *system model for MorphoSys*, our prototype reconfigurable computing system. Section 4 describes the architecture of MorphoSys *Reconfigurable Cell Array* and associated components. Next, important differences between previous research work and MorphoSys are discussed in Section 5. Section 6 describes the *software environment* with the programming and simulation tools for MorphoSys. Section 7 illustrates the *mapping of some applications* (video compression, automatic target recognition, and data encryption) to MorphoSys. Performance estimates obtained from simulation of behavioral models (C++, VHDL) are provided for these applications and compared with other systems. Finally, some conclusions of this research are mentioned in Section 8.

2 Taxonomy for Reconfigurable Systems and Previous Work

This section introduces a set of criteria that are frequently used to characterize the design of a reconfigurable computing system. These criteria are *granularity, depth of programmability, reconfigurability, interface*, and *model of computation*.

1. *Granularity*: This refers to the data size for operations of the reconfigurable component (or reconfigurable processing unit, RPU) of a system. An RPU is a logic block of configurable functionality, having a framework of reconfigurable interconnect. In *fine-grain* systems, processing elements in the RPU are typically logic gates, flip-flops, and look-up tables. They operate at the bit level, implementing a Boolean function of a finite-state machine. On the other hand, in *coarse-grain* systems, the processing elements in the RPU may contain complete functional units, like ALUs and/or multipliers that operate upon multiple-bit words. A system that combines both the above types has *mixed-grain* granularity.
2. *Depth of Programmability*: This pertains to the number of configuration programs or *contexts* stored within the RPU. An RPU may have a *single* context or *multiple* contexts. For single-context systems, only one context is resident in the RPU. Therefore, the RPU's functionality is limited to the context currently loaded. On the other hand, a *multiple-context* RPU has several contexts concurrently residing in the system. This enables the execution of different tasks simply by changing the operating context without having to reload the configuration program.
3. *Reconfigurability*: An RPU may need to be reconfigured frequently for executing different applications. Reconfiguration is the process of reloading configuration programs (context). This process is either *static* (execution is interrupted) or *dynamic* (in parallel with execution). A single context RPU typically has static reconfiguration. Dynamic reconfiguration is more relevant for a multiple-context RPU. It implies that such an RPU can execute some part of its configuration program while the other part is being changed. This feature significantly reduces the overhead for reconfiguration.
4. *Interface*: A reconfigurable system has a *remote* interface if the system's host processor is not on the same chip/die as the RPU. A *local interface* implies that the host processor and the coprocessor RPU reside on the same chip, or that the RPU is unified into the datapath of the host processor.
5. *Computation model*: Many reconfigurable systems follow the *uniprocessor* computation model. However, there are several others that follow *SIMD* or *MIMD* computation models [4], [7], [8], [11]. Some systems may also follow the *VLIW* model [2].

Conventionally, the most common devices used for reconfigurable computing are *field programmable gate arrays* (FPGAs) [1]. FPGAs allow designers to manipulate gate-level devices such as flip-flops, memory, and other logic gates. This makes FPGAs quite useful for complex bit-oriented computations. Examples of reconfigurable systems using FPGAs are [9], [10], [27], [29]. However, FPGAs have some disadvantages, too. They are slower than ASICs and have inefficient performance for coarse-grained (8 bits or more) datapath operations.

Hence, many researchers have proposed other models of reconfigurable systems targeting different applications. *PADDI* [2], *MATRIX* [4], *RaPiD* [6], and *REMARC* [7] are some of the coarse-grain prototype reconfigurable computing systems. Research prototypes with fine-grain granularity (but not based on FPGAs) include *DPGA* [3] and *Garp* [5]. Tables 1 and 2 summarize the characteristics of various reconfigurable systems according to the criteria introduced above.

2.1 Reconfigurable Systems versus SIMD Array Processors

Reconfigurable systems, in general, have built upon existing ideas and concepts in the field of computer architecture. For

TABLE 1
Classification of Reconfigurable Systems: Part A

System	Granu- -larity	Program- -mability	Reconfi- -guration
DPGA[3]	Fine	Multiple	Dynamic
Garp [5]	Fine	Multiple	Static
Splash [9],	Fine	Multiple	Static
DEC PeRLe-1 [10]	Fine	Single	Static
Chimaera [27]	Fine	Single	Static
OneChip [28]	Fine	Single	Static
DISC [29]	Fine	Single	Dynamic
PADDI [2]	Coarse	Multiple	Static
MATRIX [4]	Coarse	Multiple	Dynamic
RaPiD [6]	Coarse	Single	Mostly static
Remarc [7]	Coarse	Multiple	Static
RAW [8]	Mixed	Single	Static
PipeRench [36]	Mixed	Multiple	Dynamic
MorphoSys [37]	Coarse	Multiple	Dynamic

TABLE 2
Classification of Reconfigurable Systems: Part B

System	Interface	Comp. Model	Application Domain
DPGA [3]	Remote	Uni- -processor	Bit-level computations
Garp [5]	Local	Uni- -processor	Bit-level computations
Splash [9]	Remote	Uni- -processor	Bit-level computations
DEC PeR- -Le 1 [10]	Remote	Uni- -processor	Bit-level computations
Chimaera [27]	Local	Uni- -processor	Bit-level computations
OneChip [28]	Local	Uni- -processor	Embedded controllers, accelerators
DISC [29]	Local	Uni- -processor	General purpose
PADDI [2]	Remote	VLIW, SIMD	DSP applications
MATRIX [4]	Not defined	MIMD	Not defined
RaPiD [6]	Remote	Linear array	Systolic arrays, data-intensive
Remarc [7]	Local	SIMD	Data-parallel applications
RAW [8]	Local	MIMD	General purpose
PipeRench [36]	Remote	Pipe- -lined	Data-parallel, DSP apps.
MorphoSys [37]	Local	SIMD	Data-parallel, image apps.

example, several systems [2], [7], [8], [11] have arrays of processing units organized in an SIMD or MIMD manner. Necessarily, these systems have drawn from the research done for SIMD array processors, such as *Illiac-IV* [32], and *NASA Goodyear MPP* [33]. There are many reasons for the reemergence of the SIMD approach (after the apparent failure of the erstwhile SIMD processors to gain widespread usage). Recent years have seen the introduction of many computation-intensive, high-throughput tasks as mainstream applications. These tasks are performed efficiently on SIMD architectures. At the same time, VLSI technology has made such astronomical progress that systems which cost billions of dollars (and involved hundreds of boards) two decades ago can now be cheaply produced on a single chip.

However, there are still many differences between the erstwhile SIMD array processors and SIMD-based reconfigurable systems. A basic difference is that reconfigurable systems are configurable in terms of functionality and interconnect. Specifically, MorphoSys, when compared with SIMD array processors, also has other significant enhancements. It has a multilevel interconnection network (as compared to straight 2D mesh for Illiac-IV). MorphoSys incorporates a streaming buffer to provide efficient data transfers (no similar component in Illiac-IV) and a separate memory for storing context data. Finally, MorphoSys utilizes submicron technology (0.35 micron) to implement a system-on-chip (instead of many boards, as in Illiac-IV).

3 MorphoSys: Components, Features and Program Flow

Fig. 2 depicts the components and organization of the integrated *MorphoSys* reconfigurable computing system. The components include an array of *Reconfigurable Cells* (RC Array) with configuration memory (*Context Memory*), a control processor (*TinyRISC*), data buffer (*Frame Buffer*), and a *DMA* controller.

The correspondence between Fig. 2 and the architecture in Fig. 1 is as follows: The *RC Array* with its *Context Memory* corresponds to the reconfigurable processor array (SIMD coprocessor), *TinyRISC* corresponds to the Main Processor, and the high-bandwidth memory interface is implemented through the *Frame Buffer* and the *DMA* controller. Typically, the core processor, TinyRISC, executes sequential tasks of the application, while the reconfigurable component, the RC Array, is dedicated to the exploitation of parallelism available in an application's algorithm.

3.1 System Components

3.1.1 Reconfigurable Cell (RC) Array

The main component of MorphoSys is the 8×8 RC Array, shown in Fig. 3. Each RC has an ALU (fixed-point operations), a multiplier, and a register file and is configured through a 32-bit context word. The context words are stored in the Context Memory and are broadcast to the RC Array in two modes: *column broadcast* and *row broadcast*. For column (row) broadcast, all eight RCs in the same column (row) are configured by the same context word.

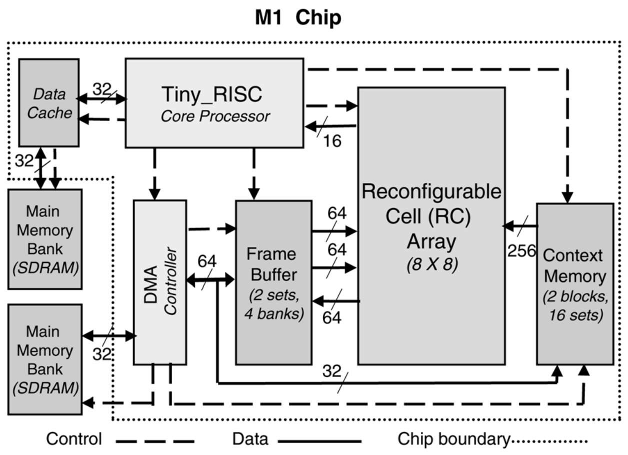

Fig. 2. Components of MorphoSys implementation (M1 chip).

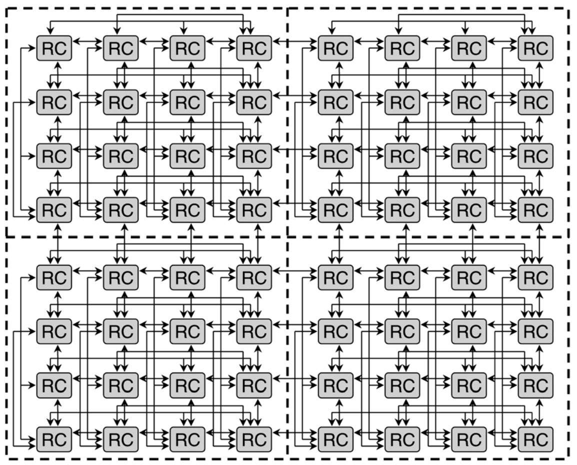

Fig. 3. MorphoSys 8×8 RC array with 2D mesh and complete row/column connectivity per quadrant.

3.1.2 Host/Control Processor

The controlling component of MorphoSys is a 32-bit processor, called TinyRISC, based on the design of a RISC processor in [12]. TinyRISC handles general-purpose operations and controls operation of the RC Array through special instructions added to its ISA. It also initiates all data transfers to or from the Frame Buffer and configuration program load for the Context Memory.

3.1.3 Frame Buffer

An important component is the Frame Buffer (FB), which is analogous to a data cache. The FB has two sets, each of which has two banks of memory. This buffer makes memory accesses transparent to the RC Array by *overlapping data load and store* with computation, alternately using the two sets. MorphoSys performance benefits tremendously from this overlap. A dedicated data buffer has been missing in most of the contemporary reconfigurable systems, with consequent degradation of performance.

3.2 Features of MorphoSys

The RC Array is configured through context words, which are broadcast from the Context Memory (Section 4.2). Each context word specifies an instruction opcode for the RC. Since the RC Array follows the *SIMD* model of computation, all RCs in the same row or column therefore share the same context word. However, each RC operates on different data. Sharing the context across a row or column is useful for data-parallel applications. In brief, important features of MorphoSys are:

- *Coarse-grain*: MorphoSys is designed to operate on 8 or 16-bit data, which ensures faster performance for word-level operations as compared to FPGAs. MorphoSys is free from variable wire propagation delays that are characteristic of FPGAs.
- *Dynamic reconfiguration*: Context data may be loaded into a inactive part of Context Memory without interrupting RC Array operation. Context loads or reloads are specified through TinyRISC and done by DMA controller.
- *Considerable depth of programmability*: The Context Memory can store up to 32 planes of configuration and provides two context broadcast modes, one along *columns* and the other along *rows*.
- *Tightly coupled interface with core processor and main memory*: The control processor (TinyRISC) and the reconfigurable component (RC Array) are resident on the same chip. The on-chip DMA controller enables fast data transfers between main memory and Frame Buffer.

3.3 TinyRISC Instructions for MorphoSys

Several new instructions were introduced in the TinyRISC instruction set for effective control of the MorphoSys RC Array execution. These instructions, described in Table 3, perform the following tasks:

- control execution of the RC Array,
- load context program to the Context Memory from the main memory, and
- transfer data between the main memory and the Frame Buffer.

There are two categories of these instructions: RC instructions (*CBCAST, SBCB, DBCBC, and DBCBR*) and DMA instructions (*LDCTXT, LDFB, and STFB*). The RC instructions control the execution of the RC Array by specifying the context to be executed, Frame Buffer address, and context broadcast mode (row or column, broadcast versus selective). The DMA instructions initiate data and context transfers between main memory, the Frame Buffer, and the Context Memory by specifying operation type (load or store), memory address, number of bytes to be transferred, and Frame Buffer or Context Memory address. These instructions are described in detail in [37].

3.4 MorphoSys System Operation

Next, the typical operation of the MorphoSys system is illustrated. TinyRISC handles the general-purpose operations, while the data-parallel parts of applications are mapped to the RC Array. Fig. 4 denotes the steps to execute these parallel tasks. A description follows:

TABLE 3
New TinyRISC Instructions for *MorphoSys*

Mnemonic	Description of operation
LDCTXT	Initiate loading of context program into *Context Memory*
LDFB, *STFB*	Initiate data transfer between main memory and *Frame Buffer*
CBCAST	Broadcast context from *Context Memory* and execute it in *RC Array* in row or column mode
SBCB	Execute *RC Array* with context (row or column mode), access single *FB* bank to load 8 data values in *RC Array* column
DBCBC (column mode)	Execute *RC Array* with context, with double bank access of *FB* to load 8 sets of 2 data values in *RC Array* column
DBCBR (row mode)	Execute *RC Array* with context, with double bank access of *FB* to load 8 sets of 2 data values in *RC Array* column
WFB	Write data from *RC Array* column to given address in *Frame Buffer*
WFBI	Write data from *RC Array* column to *Frame Buffer* (*imm. address*)
RCRISC	Write data from an RC in the top row of *RC Array* to a *TinyRISC* register

a. *Load context words into Context Memory (CM) from external memory*. TinyRISC executes the LDCTXT instruction (Table 3) and signals the DMA controller to perform this transfer.
b. *Load computation data into the first set of Frame Buffer (FB) from external memory*. TinyRISC executes the LDFB instruction (Table 3) and signals the DMA controller to perform the transfer.
c. *Execute RC Array and concurrently load data into the second set of FB*. TinyRISC issues one of CBCAST, SBCB, DBCBR, or DBCBC instructions (Table 3) each cycle to enable execution of the RC Array. These instructions specify the particular context word (among multiple contexts words in Context Memory) to be executed by the RCs and the context broadcast mode. For this step, the computations in the RC Array use data from the first set of FB. Within this time, TinyRISC also issues a single instruction (LDFB) to load computation data into the second FB set or a LDCTXT instruction to reload the Context Memory. Either of these operations is done concurrently with RC Array execution. This represents *overlap of computation* with data transfers. This step concludes when all the data in the first FB set has been processed.
d. *Execute RC Array, concurrently store data from first FB set to memory, load new data into FB set*. Once again, TinyRISC issues one of CBCAST, SBCB, DBCBR, or DBCBC instructions (Table 3) for RC Array execution. For this step, the RC Array computations use data from the second FB set. While the computations proceed, TinyRISC also issues the STFB and LDFB instructions to store data from first FB set into main memory and load new data into this set. This step terminates when data from the second FB set has been processed.
e. *Continue execution and data or context transfers till completion*. Steps c and d are repeated till the application kernel concludes after processing the entire input data.

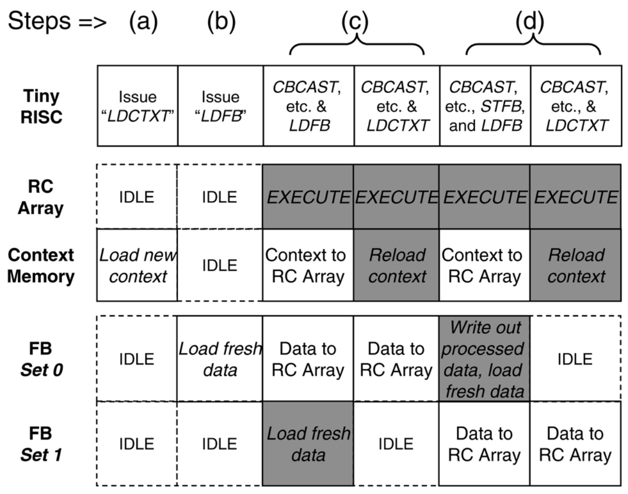

Fig. 4. System operation for MorphoSys (with overlap of computation with data transfers).

The shaded blocks of Fig. 4 represent the *overlap* of RC Array execution with data transfers or context reloads.

4 Design of MorphoSys Components

In this section, the major components of MorphoSys, namely, the Reconfigurable Cell, the Context Memory, the Frame Buffer, and the RC Array interconnection network, are described. More details are in [37]. Some aspects of the ongoing physical implementation of these components are also mentioned.

4.1 Architecture of Reconfigurable Cell (RC)

The Reconfigurable Cell (RC) Array is the programmable core of MorphoSys. It consists of an 8×8 array (Fig. 3) of identical processing elements called Reconfigurable Cells (RCs). Each RC (Fig. 5) is the basic unit of reconfiguration. Its functional model is similar to the datapath of a conventional microprocessor.

As shown in Fig. 5, an RC is composed of an ALU-multiplier, a shift unit, and two multiplexers to select its inputs. Each RC also has an output register and a register file. A context word, broadcast from Context Memory and stored in the RC *Context Register*, defines the functionality of the RC.

4.1.1 *ALU-Multiplier Unit*

This includes a 16×12 multiplier and a 28-bit fixed-point ALU. The ALU adder has been designed for 28-bit inputs to prevent loss of precision during multiply-accumulate operation since each multiplier output may be as large as 28 bits. Besides standard logic and arithmetic functions, the

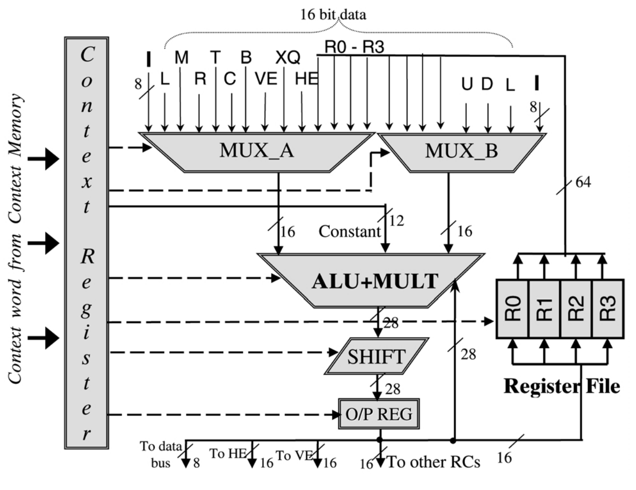

Fig. 5. Reconfigurable Cell (RC) architecture.

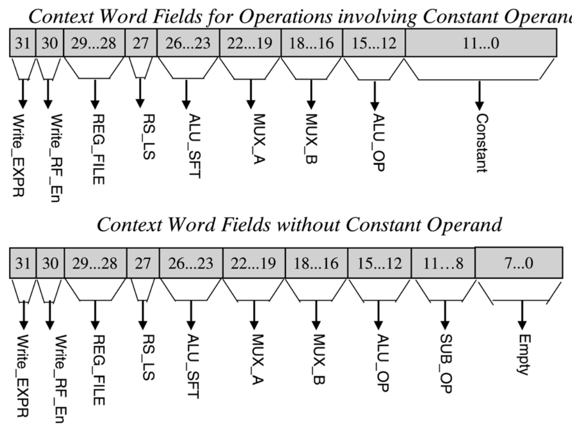

Fig. 6. RC context word definition.

ALU has other functions, such as computation of absolute value of the difference of two operands and a single cycle multiply-accumulate operation. There are a total of 25 ALU functions [37].

The two *input multiplexers* (Fig. 5) select one of several inputs for the ALU-multiplier, based on control bits from the context word in the RC Context Register. These inputs include the outputs of the four nearest neighbor RCs, outputs of other RCs in the same row and column (within the quadrant), horizontal and vertical express lanes (Section 4.3), FB data bus, and the RC register file. The RC register file is composed of four 16-bit registers. The output register is 28 bits wide so as to conform to the ALU-multiplier bitwidth.

4.1.2 Context Register

This 32-bit register contains the context word for configuring each RC. It is included in each RC of the RC Array, whereas the Context Memory is separate from the RC Array (Fig. 2).

The different fields for the context word are shown in Fig. 6. There are two variations which differ in the usage of the first field, named *Constant*. For operations that involve constant operands, this field may be used to supply these operands to the ALU-multiplier unit in each RC. An example is a multiplication by a constant over several computations, in which case the constant operand may be provided through the context word. For the second case, some bits of the *Constant* field are used as the *SUB_OP* field, which enables expansion of the ALU function set.

The field *ALU_OP* specifies the function for the ALU-Multiplier unit. *MUX_A* and *MUX_B* specify control bits for the input multiplexers of the RC. Other fields determine the direction (*RS_LS*) and amount of shift (*ALU_SFT*) applied at the ALU output. *Write_RF_En* specifies whether the RC output will be written to the RC register file. In case this field is set, the *REG_FILE* field determines the RC register to store the result of an operation. The context word also specifies whether a particular RC writes to its row or column express lane through the field, *Write_EXPR*.

4.2 Context Memory

The Context Memory broadcasts context words to the RC Array. A context word is loaded every execution cycle from the Context Memory into the *context register* of each RC. These context words configure the RC and program the interconnection network.

4.2.1 Context Memory Organization

The Context Memory is organized into *two blocks* (for row and column contexts) which store the contexts for row-wise and column-wise operation of the RC Array, respectively. Each block has *eight sets*, with each set corresponding to a row or column of the RC Array. Each set can store *16 context words*. The RC Array *configuration plane* (set of context words to program entire RC Array for one cycle) is comprised of eight context words (one from each set) from a row or column block. Thus, 16 configuration planes may be stored in each block of the Context Memory, for a total of 32 configuration planes.

4.2.2 Context Broadcast

For MorphoSys, the major focus is on regular and data-parallel applications. Based on this idea of regularity and parallelism, each context word is broadcast to a row or column of RCs. Thus, all eight RCs in a row or column, respectively, share the same context word and perform the same operations. For example, for DCT computation, eight 1D DCTs need to be computed, across eight rows. This is achieved with just eight context words to program the RC Array for each step of the computation and it takes 10 cycles (only 80 context words) to complete a 1D DCT (refer Section 7.1.2). Context broadcast enables storage of more context planes in the Context Memory than if a separate context word was used to configure each RC.

4.2.3 Dynamic Reconfiguration

The Context Memory can be updated concurrently with RC Array execution. There are 32 context planes and this depth facilitates dynamic (run-time) reloading of the contexts. Dynamic reconfiguration allows reduction of effective reconfiguration time to zero.

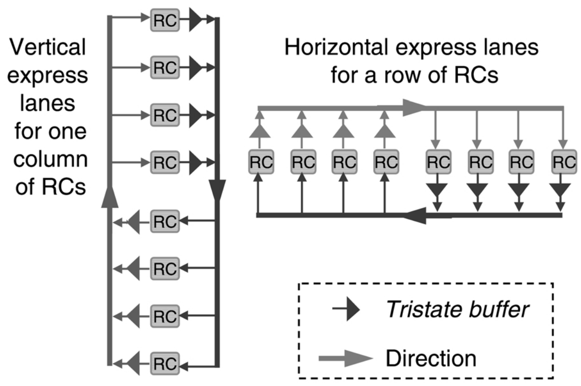
Vertical express lanes for one column of RCs
Horizontal express lanes for a row of RCs
RC
Tristate buffer
Direction

Fig. 7. Express lane connectivity (two express lanes for each row or column, in opposite directions).

4.2.4 *Selective Context Enabling*

It is possible to enable the operation of only one specific column and disable the other columns of the RC Array. One benefit of this feature is that only one context plane (eight context words) is needed for transfer of data to or from the entire RC Array. Otherwise, eight context planes (out of the 32 available) would have been required just to read or write data.

4.3 Interconnection Network

The RC Array interconnection network consists of three hierarchical levels.

Nearest neighbor connectivity: The first level of interconnections in the RC Array (Fig. 3) is a 2D mesh. This provides nearest neighbor connectivity along rows and columns.

Intraquadrant (complete row and column) connectivity: The second layer of connectivity is at the quadrant level (a quadrant is a 4×4 group of RCs). The RC Array has four quadrants (Fig. 3). Within a quadrant, each cell can access the output of any other cell in its row and column.

Interquadrant (express lane) connectivity: At the global level, there are horizontal and vertical buses called express lanes, that transmit data between RCs of adjacent quadrants. Fig. 7 shows the express lanes for a column and a row of the RC Array. These lanes provide data from any one RC (out of four) in a row or column of a quadrant to other RCs of an adjacent quadrant but the same row or column. Thus, up to four RCs in a row or column of one quadrant can access the output value of one of four RCs in the same row or column, respectively, of an adjacent quadrant.

The express lanes greatly enhance global connectivity. For example, an 8-point butterfly operation is accomplished in only three cycles.

4.4 Frame Buffer

The Frame Buffer (FB) is a data memory organized into *two sets*, Set 0 and Set 1, where each set further has two banks of memory. The two-set structure of the FB helps provide *overlap of data transfers* with *computation*. One FB set provides data for the RC Array computations and also stores processed data from the RC Array. The other FB set stores previously processed data into the main memory through the DMA controller and reloads data for the next round of computations. These operations proceed concurrently, thus preventing the latency of data I/O from adversely affecting system performance. An FB bank has 64 rows of 8 bytes each and each FB set has two banks.

4.5 Physical Implementation

MorphoSys is being implemented as the M1 chip, using both custom and standard cell design methodologies for 0.35 micron, four metal layer CMOS (3.3V) technology. The main constraint for this implementation is a clock period of *10 ns* (100 MHz freq.). The total area of the chip is estimated to be 180 sq. mm. The layout for the Reconfigurable Cell (*20,000 transistors, 1.5 sq. mm area*) is now complete. It has been simulated at the transistor level using HSPICE with appropriate output loads due to fanout and interconnect lengths to obtain accurate delay values. The multiplier ($\approxeq$ *10,000 transistors*) delay is 4 ns and the ALU ($\approxeq$ *6,500 transistors*) delay is 3 ns. The critical path delay in an RC (which corresponds to a single cycle multiply-accumulate operation) is less than 9 ns. Similarly, the TinyRISC, Frame Buffer, Context Memory, and DMA controller are also being designed to perform within the 10 ns clock constraint. Preliminary estimates for area and delay are: TinyRISC (*100,000 transistors, delay: 10 ns*), Frame Buffer (*150,000 transistors, access time: 5 ns*), and the Context Memory (*100,000 transistors, access time: 5 ns*). The three-level interconnection network is made feasible by the four metal layer technology. Simulations of the network indicate that, with the use of appropriate buffers at RC outputs, interconnect delays can be limited to 1 ns. Thus, it is reasonable to expect that the M1 chip will perform at a clock rate of 100 MHz.

5 Comparison with Related Research

Since MorphoSys architecture falls into the category of coarse-grain reconfigurable systems, it is meaningful to compare it with other coarse-grain systems (PADDI [2], MATRIX [4], RaPiD [6], REMARC [7], and RAW [8]). Many of the designs mentioned above have not actually been implemented, whereas MorphoSys has been developed down to physical layout level.

5.1 PADDI

PADDI [2] has a distinct VLIW nature because each EXU uses a 53-bit instruction word (which may be different for different EXUs), whereas MorphoSys exhibits SIMD functionality (each column or row performs the same function). PADDI cannot be configured dynamically since the instruction decoders are SRAMs that are loaded at setup time, while MorphoSys has dynamic reconfiguration. MorphoSys has a deeper depth of programmability (32) than PADDI (8). The latter employs a crossbar interconnection network, whereas MorphoSys uses 2D mesh and a hierarchical bus network.

PADDI is a stand-alone system targeting DSP applications; it does not employ any streaming buffer to speed up I/O transfers and it is not integrated with any core processor. But, the reconfigurable component in MorphoSys is integrated with a general-purpose controller (as also is REMARC, Garp, etc.) and it incorporates a data buffer (FB) for efficient data transfers.

5.2 MATRIX

The MATRIX [4] approach proposes the design of a basic unit (BFU) for a reconfigurable system. This 8-bit BFU unifies resources used for instruction storage with resources needed for computation and data storage, assumes a three level interconnection network, and may be configured for operation in VLIW or SIMD fashion. A complete system organization based on the BFU is not presented, while MorphoSys is a well-defined system. This leaves many system-level issues, such as integration with host processor, external memory interface, I/O parameters, performance, and reconfiguration (static or dynamic), open to conjecture.

The BFU design can be pipelined, but the hierarchical switch-based interconnection network (similar to FPGA interconnect) has variable interconnect delay. This could be a limiting factor for stream processing since two concurrent streams may have different processing times (because of different interconnect delays for each). However, the interconnection network for MorphoSys has uniform delays. The MATRIX approach is too generic and at least one potential problem is the complexity and overhead of the BFU control unit. Further, performance comparison of MATRIX for target applications with other systems has not been provided (as has been done for MorphoSys and most other reconfigurable systems).

5.3 RaPiD

The RaPiD [6] design is organized as a linear array of reconfigurable processing units, which is not very appropriate for block-based applications (for example, 2D signal processing tasks). This approach exemplifies provision of datapath parallelism in the temporal domain, whereas MorphoSys provides parallelism in the spatial domain. Due to its organization, the potential applications of RaPiD are those of a systolic nature or applications that can be easily pipelined.

Once again, a complete system implementation is not described. At the very least, the issue of memory stalls (mentioned in passing by the authors) could be a significant bottleneck. However, MorphoSys uses the combination of Frame Buffer and DMA controller to address this issue. The performance analysis does mention some applications from the target domain of MorphoSys (DCT, motion estimation, etc.), but estimated performance figures (in cycle times) are not given.

5.4 REMARC

The REMARC system [7] is similar in design to the MorphoSys architecture and targets the same class of data-parallel and high-throughput applications. Like MorphoSys, it also consists of 64 programmable units (organized in an SIMD manner) that are tightly coupled to a RISC processor. REMARC also uses a modified MIPS-like ISA for the RISC processor (as in the case of MorphoSys) for control of the reconfigurable component. However, REMARC has been implemented *only in software* and not at the physical level, whereas MorphoSys is being designed as an actual chip.

Also, REMARC is configured statically, it lacks a direct interface to external memory, and data transfers cannot be overlapped with computation. It does have MIMD capability, but there is no evaluation of applications to demonstrate the usefulness of this mode for REMARC. Performance figures for applications (e.g., 2D IDCT) reflect that REMARC is significantly slower than MorphoSys (Sections 7.1.2 and 7.1.3).

5.5 RAW

The RAW [8] design implements a highly parallel architecture as a Reconfigurable Architecture Workstation (RAW). The architecture is organized in an MIMD manner with multiple instruction streams. It has multiple RISC processors, each having fine-grain logic as the reconfigurable component. However, the architecture has a dispersed nature and the RISC processors do not exhibit the close coupling present in the RC Array elements. This may have an adverse effect on performance for high-throughput applications that involve many data exchanges. Some other differences are: RAW has variable communication overhead from one unit to the other, the target applications may be irregular or general purpose (instead of data-parallel, and regular applications), and coarse-grain functional reconfiguration is absent.

5.6 Comparison Summary

As is evident, all the above systems vary greatly. However, the MorphoSys architecture puts together, in a cohesive structure, the prominent features of previous reconfigurable systems (coarse-grain granularity, SIMD organization, depth of programmability, multilevel configurable interconnection network, and dynamic reconfiguration). This architecture then adds some innovative features (control processor with modified ISA, streaming buffer that allows overlap of computation with data transfer, context broadcast along rows and columns, selective context enabling), while avoiding many of the pitfalls (single contexts, I/O bottlenecks, static reconfiguration, remote interface) of previous systems. In this sense, MorphoSys is a unique implementation.

In summary, the important features of the MorphoSys architecture are:

- *Integrated model*: MorphoSys is a complete system-on-chip except for main memory.
- *Innovative memory interface*: MorphoSys employs a two-set data buffer that enables overlap of computation with data transfers.
- *Multiple contexts on-chip*: MorphoSys has multiple configuration planes (32) with *dynamic* and *single-cycle* reconfiguration.
- *On-chip general-purpose processor*: The TinyRISC processor, which is also the system controller, allows efficient execution of complex applications that include both serial and parallel tasks.

To the best of our knowledge, the *MorphoSys* architecture, as described earlier, is unique with respect to other published reconfigurable systems.

6 Programming and Simulation Environment

A comprehensive software and programming environment has been developed for MorphoSys, as depicted in Fig. 8.

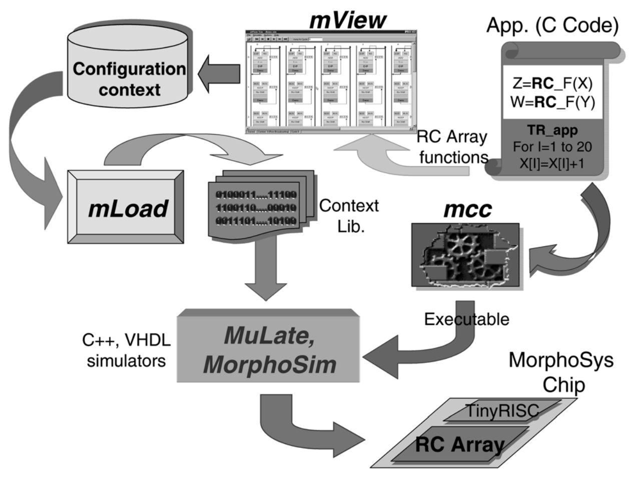

Fig. 8. Software and simulation environment for MorphoSys.

This includes two simulators, a graphical user interface (GUI), a context parser, and a C compiler. These components are described in the following subsections.

6.1 Behavioral Models for MorphoSys: MuLate and MorphoSim

The functionality of the MorphoSys reconfigurable system has been specified in C++ as well as VHDL. The system components, namely, the 8×8 RC Array, the 32-bit TinyRISC host processor, the Context Memory, Frame Buffer, and the DMA controller, have been modeled along with the external memory. The C++ model, *MuLate*, also has a graphical user interface (GUI) to display and debug application program execution. The VHDL model, *MorphoSim*, is used to simulate various applications using the QuickVHDL simulation environment. These simulations utilize several test-benches, real world input data sets, the RC context instructions, and assembly code for TinyRISC.

6.2 GUI for RC Array: mView

Another graphical user interface, *mView*, has been developed to aid the designer in mapping algorithms to the RC Array and in verifying and debugging simulation runs. *mView* takes user input for each application (specification of operations, data sources, and destinations for each RC) and generates the context program for the MorphoSys RC Array. *mView* is also used for studying RC Array simulation behavior. This GUI, based on Tcl/Tk [13] displays graphical information about the functions being executed at each RC, the active interconnections, the sources and destination of operands, usage of data buses and the express lanes, and the RC output values. It has several debugging features that allow single-step simulation runs with backward, forward, and continuous execution. It operates in one of two modes: *programming mode* or *simulation mode*.

In the *programming* mode, the user sets functions and interconnections for each row or column of the RC Array corresponding to each context (row or column mode) for the application. *mView* then generates a context file that represents the user-specified application. In the *simulation* mode, *mView* takes a context file or a simulation output file as input and provides a display of the state of each RC as the application represented by the context or simulation file is executed.

6.3 Context Generation: mLoad

For system simulation, each application has to be coded into context words and TinyRISC instructions. For the former, an assembler-parser, *mLoad*, generates contexts from programs written in the RC instruction set by the user or generated through *mView*. The next step is to determine the sequence of TinyRISC instructions for appropriate operation of RC Array and required data and context transfers. This is done using the *mcc* compiler.

6.4 mcc: Compiler for MorphoSys

An important aspect of our research is an ongoing effort to develop a programming environment for automatic mapping and code generation for MorphoSys. As part of this effort, a prototype C language compiler, *mcc*, has been developed to compile hybrid code for MorphoSys (the hybrid code contains serial, as well as parallel,functions). This compiler is based on the SUIF compiler environment [14]. The compilation is done after partitioning the code between the TinyRISC processor and the RC Array.

Currently, this partitioning is accomplished manually by inserting a particular prefix to functions that are to be mapped to the RC Array. The *mcc* compiler generates the instructions for TinyRISC (including instructions to control the RC Array execution for parallel computations). Another issue under focus is the generation of assembled context programs for MorphoSys directly from high-level application code. At an advanced development stage, the MorphoSys software environment would perform online profiling of applications and dynamically adjust the reconfiguration profile for enhanced efficiency. A detailed description of the software and programming environment is provided in [37].

7 Mapping Applications to MorphoSys

In this section, we discuss the mapping of *video compression*, an important *target recognition* application (Automatic Target Recognition), and *data encryption* algorithms to the MorphoSys architecture. *Video compression* has a high degree of data-parallelism and tight real-time constraints. *Automatic Target Recognition* (ATR) is one of the most computation-intensive applications. The *International Data Encryption Algorithm* (IDEA) [30] for data encryption is typical of data-intensive applications. Performance estimates for these applications are provided from C++/VHDL simulations. Pending the development of an automatic mapping tool, all these applications were mapped to MorphoSys either by using *mView* or manually. More details are provided in [37].

7.1 Video Compression (MPEG)

Video compression is an integral part of many multimedia applications. In this context, MPEG standards [15] for video compression are important for realization of digital video services, such as video conferencing, video-on-demand, HDTV, and digital TV.

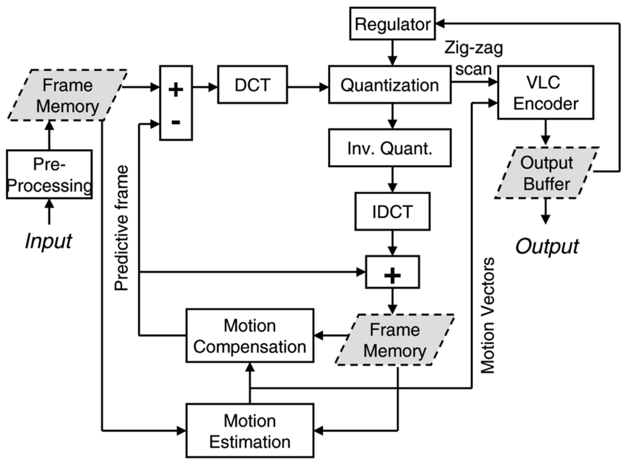

Fig. 9. Sequence of computations involved in an MPEG Encoder.

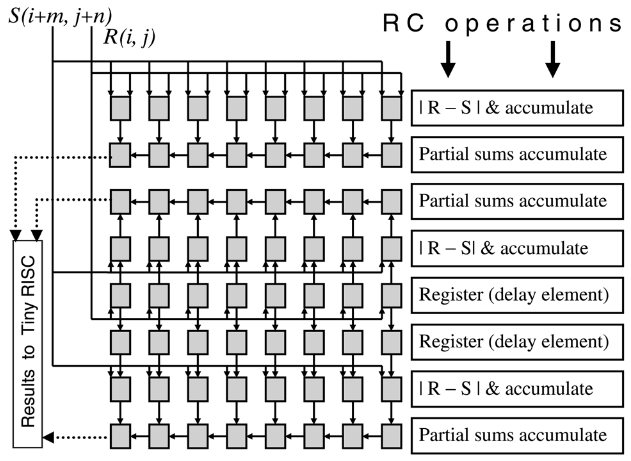

Fig. 10. Configuration of RC array for full search block matching.

As depicted in Fig. 9, the functions required of a typical MPEG encoder are:

- *Preprocessing*: for example, color conversion to YCbCr, prefiltering, and subsampling.
- *Motion Estimation and Compensation*: After preprocessing, motion estimation of image pixels is done to remove temporal redundancies in successive frames (predictive coding) of P type and B type. Algorithms such as Full Search Block Matching (FSBM) may be used for motion estimation.
- *Transformation and Quantization*: Each macroblock (typically consisting of six blocks of size 8×8 pixels) is then transformed using the Discrete Cosine Transform (DCT). The resulting DCT coefficients are quantized to enable compression.
- *Zigzag scan and VLC*: The quantized coefficients are rearranged in a zigzag manner (in order of low to high spatial frequency) and compressed using variable length encoding.
- *Inverse Quantization and Inverse Transformation*: The quantized blocks of I type and P type frames are inverse quantized and transformed back by the Inverse Discrete Cosine Transform (IDCT). This operation yields a copy of the picture, which is used for future predictive coding.

Next, we discuss two major functions, *motion estimation using FSBM* and *transformation using DCT*, of the MPEG video encoder, as mapped to MorphoSys. Finally, we discuss the overall performance of MorphoSys for the entire video encoder. (*Note*: VLC operations are not mapped to MorphoSys, but Section 7.1.3 shows that adequate time is available to execute VLC after finishing the other computations involved in MPEG encoding.)

7.1.1 Video Compression: Motion Estimation for MPEG

Motion estimation is widely adopted in video compression to identify redundancy between frames. The most popular technique for motion estimation is the block-matching algorithm because of its simple hardware implementation [17]. Some standards also recommend this algorithm. Among the different block-matching methods, Full Search Block Matching (FSBM) involves the maximum computations. However, FSBM gives an optimal solution with low control overhead.

Typically, FSBM is formulated using the mean absolute difference (MAD) criterion as follows:

$$MAD(m,n) = \sum_{i=1}^{N} \sum_{j=1}^{N} \Big| R(i,j) - S(i+m, j+n) \Big|$$

for all $m, n \in \{-p \leq m, n \leq +p\}$, where p is the maximum displacement, $R(i,j)$ is the reference block of size $N \times N$ pixels at coordinates (i,j), and $S(i+m, j+n)$ is the candidate block within a search area of size $(N+2p) \times (N+2p)$ pixels in the previous frame. The displacement vector is represented by (m,n), and the *motion vector* is determined by the least $MAD(m,n)$ among all the $(2p+1)^2$ possible displacements within the search area.

Fig. 10 shows the configuration of RC Array for FSBM computation. Initially, one reference block and the search area associated with it are loaded into one set of the Frame Buffer. The RC Array starts the matching process for the reference block resident in the Frame Buffer. During this computation, another reference block and the search area associated with it are loaded into the other set of Frame Buffer. In this manner, data loading and computation time are overlapped.

For each reference block, three consecutive candidate blocks are matched concurrently in the RC Array. As depicted in Fig. 10, each RC in the first, fourth, and seventh row performs the computation:

$$P_j = \sum_{1 \leq i \leq 16} \Big| R(i,j) - S(i+m, j+n) \Big|,$$

where P_j is the partial sum. Data from a row of the reference block is sent to the first row of the RC array and passed to the fourth row and seventh row through delay elements. The eight partial sums (P_j) generated in these rows are then passed to the second, third, and eighth row, respectively, to perform the computation:

$$MAD(m,n) = \sum_{1 \leq i \leq 16} P_j.$$

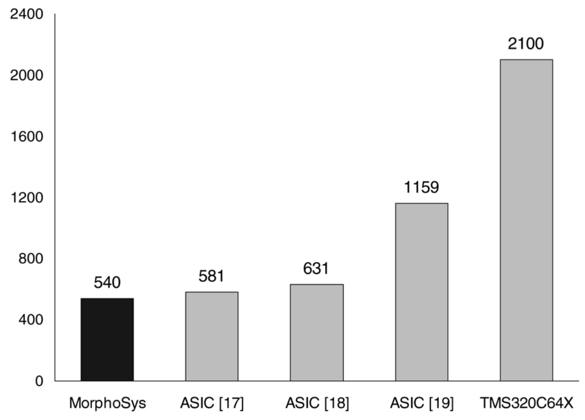

Fig. 11. Performance comparison (in cycles) for motion estimation.

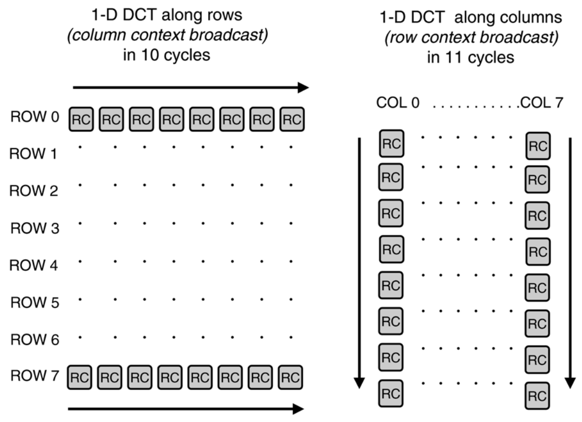

Fig. 12. Computation of 2D DCT across rows/columns.

Subsequently, three MAD values corresponding to three candidate blocks are sent to TinyRISC for comparison and the RC array starts block matching for the next three candidate blocks.

Computation cost: Based on the mapping described earlier, and using N = 16, for a reference block size of 16×16, 36 clock cycles are required to finish the matching of three candidate blocks. Ten cycles are then used by the TinyRISC for comparing the three MAD results and updating the motion vectors for the best match. There are 289 candidate blocks (102 iterations) in each search area, and VHDL simulation results show that a total of 4,692 cycles are required to match the search area. For an image frame size of 352×288 pixels at 30 frames per second (MPEG-2 main profile, low level), the processing cost is 1.85×10^6 cycles. The computation time for MorphoSys (@100 MHz) is 18.5 ms. This is smaller than the frame period of 33.33 ms. The context loading time is only 73 cycles.

Performance Analysis: MorphoSys performance is compared with three ASIC architectures implemented in [17], [18], and [19] for matching one 8×8 reference block against its search area of eight pixels displacement. The result is shown in Fig. 11. The ASIC architectures employ customized hardware units such as parallel adders to enhance performance. A high performance DSP processor, TMS320C64X [34] requires 2,100 cycles for performing the same computation. The number of processing cycles for MorphoSys is even less than the number of cycles required by the ASIC designs. Since MorphoSys is not an ASIC, its performance with regard to these ASICs is significant. In Section 7.1.3, it is shown that this performance level enables implementation of MPEG-2 encoder on MorphoSys.

Using the same parameters above, Pentium MMX [20] takes almost 29,000 cycles for the same task. When scaled for clock speed and same technology (fastest Pentium MMX fabricated with 0.35 micron technology operates at 233 MHz, therefore the cycles are divided by 2.33 as MorphoSys operates at 100 MHz), this amounts to more than 20X difference in performance.

7.1.2 Video Compression: Discrete Cosine Transform (DCT) for MPEG

The forward and inverse DCT are used in MPEG encoders and decoders. The DCT and IDCT are applied on a 2D image pixel block. It is possible to implement the 2D DCT (and IDCT) using 1D DCT on the rows and then the columns (or vice versa) of the image block, using the separability property. In the following analysis, a fast eight-point 1D DCT algorithm [21] is used. This algorithm involves 16 multiplies and 26 additions, resulting in 256 multiplies and 416 additions for a 2D implementation.

Mapping to RC Array: The standard block size for DCT/IDCT in most image and video compression standards is 8×8 pixels. Since the RC Array has the same size, each pixel of the image block may be directly mapped to each RC. Each pixel of the input block is stored in one RC. The following sequence of steps is followed:

- *Load input data*: An 8×8 block is loaded from the Frame Buffer to RC Array. The data bus between Frame Buffer and RC Array allows concurrent loading of eight pixels. An entire block is loaded in eight cycles. The same number of cycles is required to write out the processed data to the Frame Buffer.
- *Row-column approach*: The 1D DCT is first computed along the rows using the context broadcast along the columns. The DCT coefficients are provided as constants in context words. Eight 1D DCTs are computed in parallel across the eight rows as shown in Fig. 12. Next, the 1D DCT along the columns is computed (Fig. 12) using context broadcast along the rows.
- Each sequence of 1D DCT [21] involves:
  - *Butterfly computation*: It requires three cycles to perform using express lanes (interquad connectivity).
  - *Computation and rearrangement*: This takes six cycles, with an extra cycle for rearrangement.

Computation cost: The total cost for computing 2D DCT on an 8×8 block of the image, inclusive of data I/O, is 37 cycles. This estimate is verified by C++, VHDL simulation. Assuming the data blocks to be present in the Frame Buffer (through overlapping of data load/store with

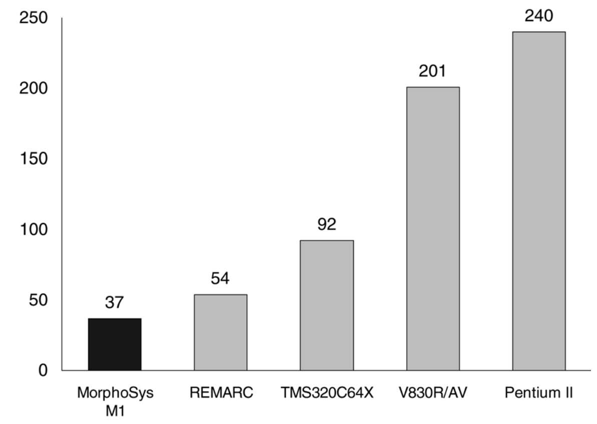

Fig. 13. DCT/IDCT performance comparison (cycles).

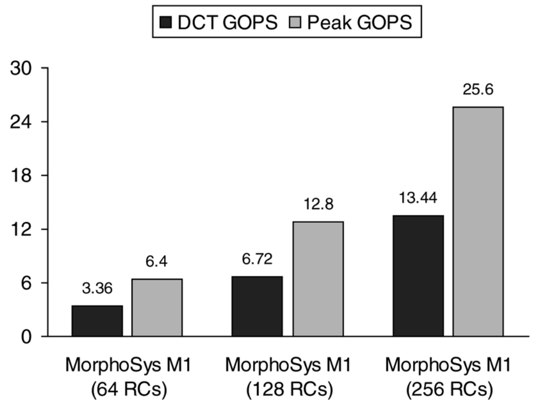

Fig. 14. Performance for DCT/IDCT—Giga operations per second (GOPS).

computation cycles), it takes 0.9 ms for MorphoSys @100 MHz to compute the DCT for the 2,376 (396×6) blocks of 8×8 pixels in one frame of a 352×288 image. The cost of computing the 2D IDCT is the same because the operations involved are similar. Context loading time is quite significant at 258 cycles. However, transformation of a large number of blocks (typically 2,376) before a different configuration is loaded minimizes this effect.

Performance analysis: MorphoSys requires 37 cycles to complete 2D DCT (or IDCT) on an 8×8 block of pixel data. The reconfigurable coprocessor, REMARC [7] takes 54 cycles. A high performance DSP video processor, TMS320C64X [34] needs 92 cycles, while a dedicated superscalar multimedia processor, the V830R/AV [22], requires 201 cycles. However, Pentium II [20] uses 240 cycles to compute the 2D DCT/IDCT with manually optimized code using MMX instructions. If performance of Pentium II is scaled for same fabrication technology (0.35 micron) as MorphoSys, the former still needs 103 cycles for this computation. The relative performance of MorphoSys and other processors is illustrated in Fig. 13.

Notably, MorphoSys performance scales linearly with the array size. For a 256 element RC Array, the number of operations possible per second would increase fourfold, with corresponding effect on throughput for 2D DCT and other algorithms. The performance figures (in GOPS) are summed up in Fig. 14 and these are more than 50 percent of the peak values. The figures are scaled for future generations of MorphoSys, conservatively assuming a constant clock of 100 MHz.

Some other points are worth noting: First, all rows (columns) perform the same computations, hence they can be configured by a common context (thus enabling broadcast of context word), which leads to saving in Context Memory space. Second, the RC Array provides the option of broadcasting context either across rows or across columns. This allows computation of the second round of 1D DCTs without transposing the data. Elimination of the transpose operation saves a considerable amount of cycles and is important for high performance. The transpose operation generally consumes valuable cycle time. For example, even a hand-optimized version of IDCT code for Pentium MMX (that uses 64-bit registers) needs at least 25 register-memory instructions for completing the transpose [20]. Other processors, such as the TMS320 series [16], also expend some cycle time on transposing data.

Precision analysis for IDCT: Experiments were conducted for measuring the precision of MorphoSys IDCT output values as specified in the IEEE Standard [23]. Considering that MorphoSys is not an ASIC, and performs fixed-point operations, the results were impressive. The worst case pixel error was satisfied and the Overall Mean Square Error (OMSE) was within 15 percent of the reference value.

Zigzag Scan: The zigzag scan function consists of irregular accesses and is therefore implemented through transfer of image data to the TinyRISC processor. This operation can also be carried out in the RC Array using a modified version of the *selective context enabling* feature in 32 cycles. For this feature, only one column or row of the RC Array operates in a cycle and each of the eight active RCs receives a different context word from the Context Memory (currently, *selective context* provides the same context word to the eight active RCs).

7.1.3 Mapping MPEG-2 Video Encoder to MorphoSys

It is remarkable that, because of the computation-intensive nature of motion estimation, only dedicated processors or ASICs have been used to implement MPEG-2 video encoders. Most reconfigurable systems, DSP processors, or multimedia processors (e.g., [16]) consider only MPEG decoding or a subtask (e.g., IDCT). Our mapping of the complete MPEG-2 encoder to MorphoSys is perhaps the first time that a reconfigurable system has been able to meet the high throughput requirements of the MPEG-2 video encoder.

We mapped all the functions for MPEG-2 video encoder, except VLC operations, to MorphoSys. We assume that the Main profile (low level) is being used. The maximum resolution at this level is 352×288 pixels per frame at 30 frames per second. The group of pictures consists of a sequence of four frames in the order IBBP (a typical choice for broadcasting applications). The number of cycles required for each task of the MPEG encoder, for each macroblock type is listed in Tables 4 and 5. In Table 5, MC and IMC refer to *Motion Compensation* and its *inverse* and Q and IQ refer to *Quantization* and its *inverse*. Cycles required

TABLE 4
Performance of *MorphoSys* for Motion Estimation

Macroblock type vs. MPEG function (clock cycles)	Motion Estimation		
	Context Load	FB Load	Comput--ation
P type	73	322	4692
B type	73	579	9384

for loading context and data from memory are also included.

All macro-blocks in each P frame and B frame are first subjected to *motion estimation*. Then, *motion compensation, DCT,* and *quantization* are performed on each macroblock in a frame. The processed macroblocks are sent to frame storage in main memory. Finally, we perform *inverse quantization, inverse DCT,* and *reverse motion prediction* for each macroblock of I frames and P frames. Each frame has 396 macroblocks, and the clock cycles required for encoding each frame type are depicted in Fig. 15. It may be noted that *motion estimation* takes up more than 75 percent of the computation time for P type and B type frames.

From the data in Fig. 15, and assuming IBBP frame sequence, total encoding time is 108.4 ms. This is 81 percent of available time (134 ms). From empirical data values in [22], 19 percent (remaining time) of available time is sufficient to compute VLC. Table 6 shows that figures for MorphoSys MPEG encoder (without VLC) are at least an order of magnitude less than the corresponding figures for REMARC [35]. The algorithm (FSBM) for motion estimation, which is the major computation, is the same for REMARC and MorphoSys and the frame parameters are also similar.

7.2 Automatic Target Recognition (ATR)

Automatic Target Recognition (ATR) is the machine function of automatically detecting, classifying, recognizing, and identifying an object. The ACS Surveillance challenge has been quantified as the ability to search 40,000 square nautical miles per day with one-meter resolution [24]. The computation levels reach the hundreds-of-teraops range when targets are partially obscured. There are many algorithmic choices available to implement an ATR system.

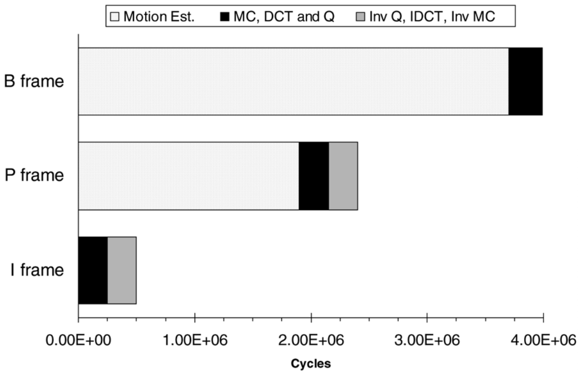

Fig. 15. MorphoSys performance for I, P, and B frames (MPEG-2 video encoder).

The ATR processing model developed at Sandia National Laboratory [25], [26] has three major parts, namely, *Focus-of-Attention* (FOA), *Second Level of Detection* (SLD), and *Final Identification* (FI). This model was designed to detect partially obscured targets in Synthetic Aperture Radar (SAR) images generated by the radar imager in real time. SAR images (8-bit pixels) are input to a focus-of-attention processor to identify regions of interest (called chips). These chips are thresholded to generate binary images and the binary images are then matched against binary target templates. Target templates appear in pairs of a bright and a surround template. The bright template identifies locations where a strong radar return is expected, while the surround template identifies locations where strong radar absorption is expected. The SLD step is the most computation-intensive and is shown in Fig. 16.

Computations for SLD (Sandia ATR model): Based on [25], the sequence of steps involved in SLD are:

- First, the 128×128 chip (with 8-bit pixel values) is sliced into eight bitplanes to compute the *shapesum*, which is a weighted sum of the eight results obtained by correlating each bitplane with the bright template.
- Next, the target template pairs are *matched* with the chip. The matching is based on bit-correlation and is performed on eight different binary images that are generated by applying eight threshold values to the chip. The binary images are correlated with both the *bright* and *surround* templates to generate eight pairs of correlation results and the shapesum is used to select one of the eight results.

TABLE 5
MorphoSys Performance for MC, DCT, and Q

Macro-block type vs. MPEG functions (clock cycles)	MC, DCT, Q (/ for IQ, IDCT, IMC)		
	Context Load	FB Load	Comput--ation
I type	258/258	102/102	624/624
P type	258/258	204/204	672/672
B type	258/N-A	306/N-A	720/N-A

TABLE 6
MPEG Encoder Comparison of MorphoSys and Remarc [35]

System / Frame type	*MorphoSys* (clock cycles)	Remarc [35] (clock cycles)
I frame	0.5×10^6	52.9×10^6
P frame	2.3×10^6	69.6×10^6
B frame	4.0×10^6	81.5×10^6

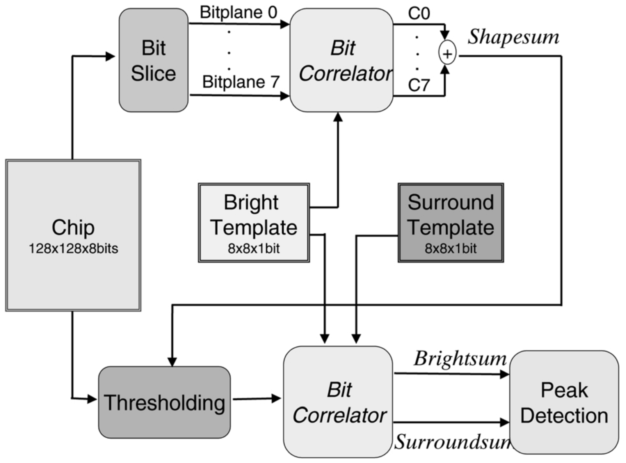

Fig. 16. ATR SLD processing model.

- The selected pair of results is subsequently forwarded to the *peak detector*.

Bit-correlation: Both shapesum computation and target template matching, which are the most computation-intensive steps in the SLD process, require *bit-correlation*. This operation is performed in highly parallel fashion in the MorphoSys RC Array. Each row of the 8×8 target template is packed as an 8-bit number and loaded in the RC Array. All the candidate blocks in the chip are correlated with the target template. Each column of the RC Array performs correlation of one target template with one candidate block, hence, eight templates are correlated concurrently in the RC Array.

Fig. 17 illustrates the operation of the *bit-correlator* implemented in MorphoSys. In order to perform *bit-correlation*, two bytes (16 bits) of image data are input to each RC. In the first step, the eight most significant bits of the image data are ANDed with the template data and a special adder tree (implemented as custom hardware in each RC) is used to count the number of ones of the ANDed output to generate the correlation result. Then, the image data is shifted left one bit and the process is repeated again to perform matching of the second block. After the image data is shifted eight times, a new 16-bit data is loaded and the RC starts another correlation of eight consecutive candidate blocks. For this mapping scheme, each column of the RC Array uses only four clock cycles to correlate one 8×8 binary image with an 8×8 target template.

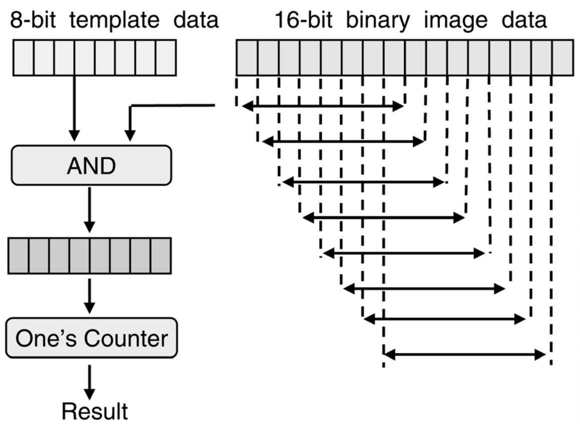

Fig. 17. Bit-correlation process in each RC.

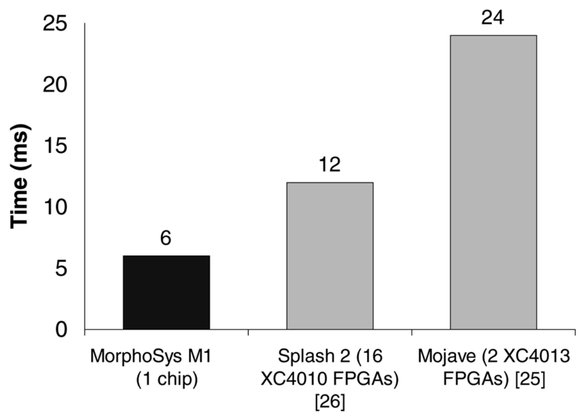

Fig. 18. Performance comparison of MorphoSys for SLD (ATR).

Performance analysis: For analysis, we choose system parameters implemented in [25]. The ATR systems from [25] and [26] are used for comparison. Two Xilinx XC4013 FPGAs are used in the *Mojave* system [25]. The Splash 2 system, consisting of 16 Xilinx XC4010 FPGAs, is used in [26]. The image size of the chip is 128×128 pixels and the template size is 8×8 bits. The complete SLD processing time for one pair of templates is 6.0 ms for MorphoSys. About 24 ms are required for the *Mojave* system and each processing element (eight FPGAs) of the Splash 2 system [26] performs the computation in 12 ms. These results are shown in Fig. 18.

The operating frequency of the FPGA-based *Mojave* and Splash 2 systems is about 20 MHz, but this is relatively fast for the FPGA domain. The frequency difference is also offset by the fact that ATR operations are inherently fine-grain and involve bit-level computations, which map very well on FPGAs (but FPGAs still have other limitations, as listed in [36]). Even though MorphoSys is a coarse-grained system, it achieves better performance than the above FPGA-based systems for the bit-level ATR operations.

ATR System Specification: A quantified measure of the ATR problem [25] states that 100 chips have to be processed each second for a given target. Each target has 40 pairs of templates for every five-degree rotation (72×40 pairs for full 360-degree rotation). Considering these requirements, 1,728 chips of MorphoSys (M1) would be needed to satisfy this specification, as compared to about 7,000 sets of the 2-FPGA Mojave system [25] and 1,728 boards of the 16-FPGA Splash 2 system [26].

7.3 Data Encryption Using IDEA

Data security is a key application domain. The *International Data Encryption Algorithm* (IDEA) [30] is a typical example of this application class. IDEA involves processing of plaintext data (data to be encrypted) in 64-bit blocks with a 128-bit encryption/decryption key. The algorithm performs eight iterations of a core function. After the eighth iteration, a final transformation step produces a 64-bit ciphertext (encrypted data) block. IDEA employs three operations: *bitwise exclusive-OR, addition*

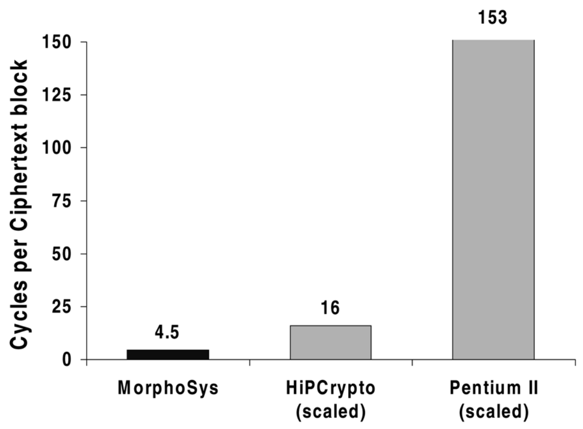

Fig. 19. Performance comparison for IDEA mapping on MorphoSys.

modulo 2^{16} and *multiplication modulo* $2^{16}+1$. It is assumed that encryption/decryption keys are generated externally and then loaded once into the Frame Buffer of MorphoSys.

When mapping IDEA to MorphoSys, some operations of IDEA's core function can be performed in parallel, while others are performed sequentially. The maximum number of operations that are performed in parallel is four. To exploit this parallelism, clusters of four cells in the RC Array columns are allocated to operate on each plaintext block. Thus, the whole RC Array can operate on 16 plaintext blocks in parallel.

Performance analysis: As two 64-bit blocks are transferred simultaneously through the operand data bus, it takes only eight cycles to load 16 plaintext blocks into the RC Array. Each of eight iterations of the core function takes seven clock cycles to execute within a cell cluster. The final transformation step requires one additional cycle. Once the ciphertext blocks have been produced, eight cycles are necessary to write back to the Frame Buffer before loading the next plaintext. It takes 73 cycles to produce 16 ciphertext blocks.

A software implementation of the IDE algorithm on a Pentium II processor requires 357 clock cycles to generate one ciphertext block (performance profiled using Intel *Vtune*). An ASIC, *HiPCrypto* [31], that provides a dedicated hardware implementation of IDEA produces seven ciphertext data every 56 cycles. The performance of these two systems is scaled to the 100 MHz operating frequency of MorphoSys, resulting in 153 effective cycles for Pentium II (normally operating at 233 MHz for 0.35 micron fabrication technology) and 16 effective cycles for HiPCrypto (which operates at 53 MHz). This scaled relative performance is depicted in Fig. 19.

8 Conclusions and Future Work

This paper has presented *MorphoSys*, a new reconfigurable architecture. Its performance has been evaluated for many of the target applications with impressive results that validate this architectural model. Work on the physical implementation of MorphoSys on a custom-designed chip, M1, is in progress.

Extensions for MorphoSys model: It may be noted that the MorphoSys architecture is not limited to using a simple RISC processor as the main processor. There are many options for the main processor, such as using an advanced general-purpose processor in conjunction with TinyRISC (which would then function as an I/O processor for the RC Array). Also, an advanced processor with multithreading may be used to enable concurrent processing of application programs by the RC Array and the main processor.

Another potential focus is the RC Array. For this implementation, the array has been designed for data-parallel and computation-intensive tasks. However, the design model allows other versions, too. For example, a suitably designed RC Array may be used for a different application class, such as high-precision signal processing, bit-level computations, control-intensive applications, or dynamic stream processing.

Based on this, we visualize that the MorphoSys architecture may be the precursor of a generation of systems that integrate advanced general-purpose processors with a specialized reconfigurable component, in order to meet the constraints of mainstream, high-throughput and computation-intensive applications.

Acknowledgments

This research is supported by DARPA (DoD) under contract F-33615-97-C-1126 and the US National Science Foundation under grant CCR-9877171. We express thanks to Ms. Kerry Hill of the Air Force Research Laboratory for her constructive feedback and Prof. Tomas Lang and Prof. Walid Najjar for their useful and incisive comments.

We acknowledge the contributions of Dexter Bayani, Maneesha Bhate, Matthew Campbell, Sadik Can, Benjamin U-Tee Cheah, Alexander Gascoigne, Nambao Van Le, David Liu, Rafael Maestre, Robert Powell, Rei Shu, Lingling Sun, Cesar Talledo, Eric Tan, Timothy Truong, Tom Truong, and Swapna Yasarapu, all of whom have been associated with the development of MorphoSys architecture and application mapping.

References

[1] S. Brown and J. Rose, "Architecture of FPGAs and CPLDs: A Tutorial," *IEEE Design and Test of Computers*, vol. 13, no. 2, pp. 42-57, 1996.

[2] D. Chen and J. Rabaey, "Reconfigurable Multi-Processor IC for Rapid Prototyping of Algorithmic-Specific High-Speed Datapaths," *IEEE J. Solid-State Circuits*, vol. 27, no. 12, Dec. 1992.

[3] E. Tau, D. Chen, I. Eslick, J. Brown, and A. DeHon, "A First Generation DPGA Implementation," *Proc. Canadian Workshop Field-Programmable Devices*, May 1995.

[4] E. Mirsky and A. DeHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources," *Proc. IEEE Symp. FPGAs for Custom-Computing Machines*, pp. 157-166, Apr. 1996.

[5] J.R. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Co-Processor," *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, Apr. 1997.

[6] C. Ebeling, D. Cronquist, and P. Franklin, "Configurable Computing: The Catalyst for High-Performance Architectures," *Proc. IEEE Int'l Conf. Application-Specific Systems, Architectures, and Processors*, pp. 364-72, July 1997.

[7] T. Miyamori and K. Olukotun, "A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications," *Proc. IEEE Symp. FPGAs for Custom Computing Machines*, Apr. 1998.

[8] J. Babb, M. Frank, V. Lee, E. Waingold, R. Barua, M. Taylor, J. Kim, S. Devabhaktuni, and A. Agrawal, "The RAW Benchmark Suite: Computation Structures for General-Purpose Computing," *Proc. IEEE Symp. Field-Programmable Custom Computing Machines,* pp. 134-143, Apr. 1997.

[9] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti, "Building and Using a Highly Parallel Programmable Logic Array," *Computer,* pp. 81-89, Jan. 1991.

[10] P. Bertin, D. Roncin, and J. Vuillemin, "Introduction to Programmable Active Memories," *Systolic Array Processors,* pp. 300-309, Prentice Hall, 1989.

[11] A.K. Yeung and J.M. Rabaey, "A 2.4 GOPS Data-Driven Reconfigurable Multiprocessor IC for DSP," *Proc. IEEE Solid-State Circuits Conf.,* pp. 108-109, 346, 440, Feb. 1995.

[12] A. Abnous, C. Christensen, J. Gray, J. Lenell, A. Naylor, and N. Bagherzadeh, "Design and Implementation of TinyRISC Microprocessor," *Microprocessors and Microsystems,* vol. 16, no. 4, 1992.

[13] B. Welch, *Practical Programming in Tcl and Tk,* second ed. Prentice Hall, 1997.

[14] The Stanford SUIF Compiler Group, "SUIF Compiler System," http://suif.stanford.edu.

[15] ISO/IEC JTC1 CD 13818, "Generic Coding of Moving Pictures, MPEG-2 Standard," 1994.

[16] F. Bonomini, F. De Marco-Zompit, G.A. Mian, A. Odorico, and D. Palumbo, "Implementing an MPEG2 Video Decoder Based on TMS320C80 MVP: SPRA 332," technical report, Texas Instruments, Sept. 1996.

[17] C. Hsieh and T. Lin, "VLSI Architecture for Block-Matching Motion Estimation Algorithm," *IEEE Trans. Circuits and Systems for Video Technology,* vol. 2, no. 2, pp. 169-175, June 1992.

[18] S.H. Nam, J.S. Baek, T.Y. Lee, and M.K. Lee, "A VLSI Design for Full Search Block Matching Motion Estimation," *Proc. IEEE ASIC Conf.,* pp. 254-257, Sept. 1994.

[19] K.-M. Yang, M.-T. Sun, and L. Wu, "A Family of VLSI Designs for Motion Compensation Block Matching Algorithm," *IEEE Trans. Circuits and Systems,* vol. 36, no. 10, pp. 1,317-1,325, Oct. 1989.

[20] "Application Notes for Pentium MMX,"http://developer.intel.com/drg/mmx/appnotes/.

[21] W.-H. Chen, C.H. Smith, and S.C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform," *IEEE Trans. Comm.,* vol. 25, no. 9, pp. 1,004-1,009, Sept. 1977.

[22] T. Arai, I. Kuroda, K. Nadehara, and K. Suzuki, "V830R/AV: Embedded Multimedia Superscalar RISC Processor," *IEEE Micro,* pp. 36-47, Mar./Apr. 1998.

[23] IEEE, "IEEE Standard Specifications for the Implementation of 8×8 Inverse Discrete Cosine Transform, Std. 1180-1990," 1990.

[24] Defense and Advanced Research Projects Agency (DARPA), "Challenges for Adaptive Computing Systems," http://www.darpa.mil/ito/research/acs/challenges.html.

[25] K. Chia, H.J. Kim, S. Lansing, and W.H. Mangione-Smith, J. Villasenor, "High-Performance Automatic Target Recognition through Data-Specific VLSI," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 6, no. 3, pp. 364-371, Sept. 1998.

[26] M. Rencher and B.L. Hutchings, "Automated Target Recognition on SPLASH 2," *Proc. IEEE Symp. Field-Programmable Custom Computing Machines,* pp. 192-200, Apr. 1997.

[27] S. Hauck, T.W. Fry, M.M. Hosler, and J.P. Kao, "The Chimaera Reconfigurable Functional Unit," *Proc. IEEE Symp. Field-Programmable Custom Computing Machines,* pp. 87-96, Apr. 1997.

[28] R.D. Wittig and P. Chow, "OneChip: An FPGA Processor with Reconfigurable Logic," *Proc. IEEE Symp. FPGAs for Custom Computing Machines,* pp. 126-135, Apr. 1996.

[29] M.J. Wirthlin and B.L. Hutchings, "A Dynamic Instruction Set Computer," *Proc. IEEE Symp. FPGAs for Custom Computing Machines,* pp. 99-107, Apr. 1995.

[30] B. Schneier, "Other Block Ciphers," *Applied Cryptography,* pp. 319-325, New York: John Wiley, 1996.

[31] S. Salomao, V. Alves, and E.C. Filho, "HiPCrypto: A High Performance VLSI Cryptographic Chip," *Proc. IEEE ASIC Conf.,* pp. 7-13, Sept. 1998.

[32] W.J. Bouknight, S.A. Denenberg, D.E. McIntyre, J.M. Randall, A.H. Sameh, and D.L. Slotnick, "The Illiac IV System," *Proc. IEEE,* vol. 60, no. 4, pp. 369-388, Apr. 1972.

[33] K.E Batcher, "Design of Massively Parallel Processor," *IEEE Trans. Computers,* vol. 29, no. 9, pp. 836-840, Sept. 1980.

[34] Texas Instruments Inc., "TMS320C6000 Assembly Benchmarks at Texas Instruments: C64X DSP Benchmarks," www.ti.com/sc/docs/products/dsp/c6000/benchmarks/.

[35] T. Miyamori and K. Olukotun, "REMARC: Reconfigurable Multimedia Array Co-Processor," *IEICE Trans. Information Systems,* vol. E82-D, no. 2, pp. 389-397, Feb. 1999.

[36] S.C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R.R. Taylor, and R. Laufer, "PipeRench: A Coprocessor for Streaming Multimedia Acceleration," *Proc. Int'l Symp. Computer Architecture,* pp. 28-39, May 1999.

[37] H. Singh, "Reconfigurable Architectures for Multimedia and Data-Parallel Application Domains," PhD thesis, Univ. of California, Irvine, 2000.

Hartej Singh received the BS degree in electrical engineering from Thapar Institute of Engineering and Technology, Patiala, India, in 1993. He was a design engineer (R&D) at Punjab Wireless Systems (India) from 1993 to 1994. He received the MS degree and PhD degrees in electrical and computer engineering from the University of California, Irvine, in 1997 and 2000, respectively. He has been associated with the MorphoSys project at the UCI Reconfigurable Computing Lab since 1997. Dr. Singh's areas of interest are processor architectures, reconfigurable processors, VLSI system design, low power optimizations, application analysis and architectural mapping, and multimedia applications. He is a member of the IEEE.

Ming-Hau Lee received the BSc degree from the Department of Electrical Engineering at the National Taiwan University, Taipei, Taiwan, in 1993. In 1997, he received the MS degree in electrical and computer engineering from the University of California, Irvine (UCI), where he is currently working toward the PhD degree. He is currently a research assistant at the UCI Reconfigurable Computing Laboratory. His research interests are in computer architecture, reconfigurable computing systems, and VLSI low-power design. He is a student member of the IEEE.

Guangming Lu received the BEng degree in electronic engineering from the Tsinghua University, Beijing, China, in 1994. From 1994 to 1996, he was with the Department of Computer Science and Technology at Peking University, Beijing, China. He received the MS degree from the Department of Electrical and Computer Engineering at the University of California, Irvine, in 1997, where he is currently working toward the PhD degree. Since 1997, he has been a research assistant at the UCI Reconfigurable Computing Laboratory. His areas of interest are VLSI systems design, CAD tools, design automation, communication, and computer architecture. He is a member of the IEEE.

Fadi J. Kurdahi (M'87) received the BEng degree in electrical engineering from the American University of Beirut, Lebanon, in 1981. He received the MS degree in electrical engineering and the PhD degree in computer engineering from the University of Southern California, Los Angeles, in 1982 and 1987, respectively. Since 1987, he has been with the Department of Electrical and Computer Engineering at the University of California, Irvine, where he is currently a professor of electrical and computer engineering and information and computer science.

He received a US Nation Science Foundation Research Initiation Award in 1989, and two ACM/SIGDA fellowships in 1991 and 1992. His areas of interest are reconfigurable computing, high-level synthesis of digital circuits, VLSI systems design and layout, and design automation. He has served on program committees of ISSS, ED&TC, ISSS, ISCAS, among others. Dr. Kurdahi was an associate editor of the *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* from 1993-1995. He is a member of the IEEE and the ACM.

Nader Bagherzadeh (M'79, SM'93) holds a PhD in computer engineering from the University of Texas at Austin. He is a professor and the chair of the Department of Electrical and Computer Engineering at the Henry Samueli School of Engineering at the University of California, Irvine. Before joining UCI, he was a member of the technical staff (MTS) at AT&T Bell Laboratories from 1979 to 1984. His areas of interest include computer architecture, reconfigurable computing, VLSI chip design, computer graphics, and high performance algorithms. He is a senior member of the IEEE.

Eliseu M. Chaves Filho received the BS degree in electrical engineering from the University of Brasilia in 1983, the MSc degree in computer science from the Catholic University of Rio de Janeiro in 1987, and the DSc degree in systems and computer engineering from the Federal University of Rio de Janeiro in 1994. He is currently an associate professor in the Department of Systems and Computer Engineering at the Federal University of Rio de Janeiro. His research interests include computer architecture with emphasis in parallel microarchitectures, VLSI systems, and operating systems. He is a member of the IEEE.