

MorphoSys: An Integrated Re-configurable Architecture

Hartej Singh, Ming-Hau Lee, Guangming Lu,
Fadi J. Kurdahi, Nader Bagherzadeh, and Tomas Lang,

*University of California, Irvine,
ET 544 F, Irvine, CA 92697, United States*

Robert Heaton,
Obsidian Technology, and

Eliseu M. C. Filho,
Federal University of Rio de Janeiro, Brazil

Summary: In this paper, we present the MorphoSys re-configurable architecture, which combines a configurable array of processing elements with a RISC processor core. We provide a system-level model, describing the array architecture and the inter-connection network. We give several examples of applications that can be mapped to the MorphoSys architecture. We also show that MorphoSys achieves performance improvements of more than an order of magnitude as compared to other implementations and processors.

1. Introduction

Re-configurable computing systems are systems that combine programmable hardware with programmable processors. At one extreme of the computing spectrum, we have general-purpose processors that are programmed entirely through software. At the other extreme are application-specific ICs (Asics) that are custom designed for particular applications. The former has wider applicability, while the latter is specialized but very efficient. Re-configurable computing is a hybrid of the two approaches. It involves configuration or customization of hardware for a range of applications [4]. Conventionally, the most common devices used for re-configurable 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. However, FPGAs have certain inherent disadvantages such as bit-level operation and inefficient performance for ordinary arithmetic or logic operations. Hence, many researchers have focused on a more general and higher level model of configurable computing systems. As a result, the PADDI [5], rDPA [6], DPGA [7], MATRIX [8], Garp [9], RaPiD [10,11], and Raw [12,13] are some of the systems that have been developed as prototypes of re-

configurable computing systems. These are discussed briefly in a following section.

Target applications: Over the last decade, configurable computing systems have demonstrated significant potential for a range of applications. Many of these tasks (e.g. real-time signal processing) are computation-intensive and have high throughput requirements. Other applications are inherently complex (e.g. real-time speech recognition). In general, conventional microprocessor-based architectures fail to meet the performance needs for most of the applications in the realm of image processing, image understanding, signal processing, encryption, information-mining, etc. Automatic target recognition, feature extraction, surveillance, video compression are among those applications that have shown performance improvements of over an order of magnitude when implemented on configurable systems [4]. Other target applications for configurable systems are data parallel computations, convolution, stream processing, template matching, image filtering, etc.

Organization of paper: Section 2 provides definitions for terms relevant to re-configurable computing. Then, we present a brief review of previous research work in this sphere. Section 4 introduces the system model for MorphoSys, our prototype configurable computing system, MorphoSys. The following section (Section 5) describes the architecture of the basic cell of MorphoSys programmable hardware. Next, we discuss the mapping of a set of applications from image processing domains (video compression and automatic target recognition). We provide performance estimates for these applications and compare them with other systems and processors. Section 7 describes the MorphoSys simulation environment and graphical user interface. Finally, we present some conclusions from our research in Section 8.

2. Taxonomy

In this section, we provide definitions for parameters that are typically used to characterize the design of a re-configurable computing system.

- (a) *Granularity (fine versus coarse)*: This refers to the level of operation, i.e. bit-level versus word-level. Bit-level operations correspond to fine-grain granularity but coarse-grain granularity implies word-level operations. Depending upon the granularity, the configurable component may be a look-up table, a gate or an ALU-multiplier.
- (b) *Depth of Programmability (single versus multiple)*: This is defined as the number of configuration planes resident in a re-configurable system. Some systems may have only a single resident configuration plane. This means that system functionality is limited to that plane. On the other hand, a system may have multiple configuration planes. In this case, different tasks may be performed by choosing varying planes for execution.
- (c) *Re-configurability (static versus dynamic)*: A system may be frequently reconfigured for executing different applications. Re-configuration is either static (execution is interrupted) or dynamic (in parallel with execution). Single context systems can typically be reconfigured only statically. Multiple context systems favor dynamic reconfiguration
- (d) *Interface (remote versus local)*: A configurable system has remote interface if the system's host processor is not on the same chip/die as the programmable hardware. The system has a local interface if the host processor and programmable logic reside within the same chip.
- (e) *Computation model*: For most configurable systems, the computation model may be described as either SIMD or MIMD. Some systems may follow the VLIW model.

3. Related Work

There has been considerable research effort to develop prototypes for configurable computing. In this section, we shall present the salient architectural features of each system.

The *Splash* [2] and *DECPeRLe-1* [3] computers were among the first research efforts in configurable computing. *Splash* consists of a linear array of processing elements with limited routing resources. It is useful for linear

systolic applications. *DECPeRLe-1* is organized as a two-dimensional array of 16 FPGAs. The routing is more extensive, with each FPGA also having access to a column and row bus. Both systems are fine-grained, with remote interface, single configuration and static re-configurability.

Other research prototypes with fine-grain granularity include *DPGA* [7] and *Garp* [9]. Systems with coarse-grain granularity include *PADDI* [5], *rDPA* [6], *MATRIX* [8], *RaPiD* [10] and *Raw* [12].

PADDI [5] has a set of concurrently executing 16-bit functional units (EXUs). Each of these has an eight-word instruction memory. The communication network between EXUs uses crossbar switches for flexibility. Each EXU has dedicated hardware for fast arithmetic operations. Memory resources are distributed among the EXUs.

rDPA: The re-configurable data-path architecture (*rDPA*) [6] aims for better performance for word-level operations through data-paths wider than typical FPGA data-paths. The *rDPA* consists of a regular array of identical data-path units (DPUs). Each DPU consists of an ALU, a micro-programmable control and four registers. There are two levels of interconnection: local (mesh network of short wires) and global (long wires). The *rDPA* array is dynamically re-configurable.

MATRIX: This architecture [8] aims to unify resources for instruction storage and computation. The basic unit (BFU) can serve either as a memory or a computation unit. The 8-bit BFUs are organized in an array, where each BFU has a 256-word memory, ALU-multiply unit and reduction control logic. The interconnection network has a hierarchy of three levels (nearest neighbor, length four bypass connection and global lines).

RaPiD: This is a linear array of functional units [10], which is configured mostly to form a linear computation pipeline. The identical array cells each have an integer multiplier, three ALUs, six registers and three small local memories. A typical array has 8 to 32 of these cells. It uses segmented buses for efficient utilization of interconnection resources.

Raw: The main idea of this approach [12] is to implement a highly parallel architecture and fully expose low-level details of the hardware architecture to the compiler. The Re-configurable Architecture Workstation (*Raw*) is a set of replicated tiles, wherein each tile contains a simple RISC like processor, small amount of bit-level configurable logic and some memory for instructions and data. Each *Raw* tile has an associated programmable switch which connects the tiles in a wide-channel point-to-point interconnect.

DPGA: A fine-grain prototype system, the Dynamically Programmable Gate Arrays (*DPGA*) [7] use traditional 4-input lookup tables as the basic array element. Each cell

can store 4 context words. DPGA supports rapid run-time reconfiguration. Small collections of array elements are grouped as sub-arrays that are tiled to form the entire array. A sub-array has complete row and column connectivity. Configurable crossbars are used for communication between sub-arrays.

Garp: This fine-grained approach [9] has been designed to fit into an ordinary processing environment, where a host processor manages main thread of control while only certain loops and subroutines use the re-configurable array for speedup in performance. The host processor is responsible for loading and execution of configurations on the re-configurable array. The instruction set of the host processor has been expanded to accommodate instructions for this purpose. The array is composed of rows of blocks. These blocks resemble CLBs of Xilinx 4000 series [25]. There are at least 24 columns of blocks, while number of rows implementation specific. The blocks operate on 2-bit data. There are vertical and horizontal block-to-block wires for data movement within the array. Separate memory buses move information (data as well as configuration) in and out of the array.

4. MorphoSys System Model

Figure 1 shows the organization of the MorphoSys re-configurable computing system. It is composed of a re-configurable array, a control processor, a data buffer and a DMA controller. It is coarse-grain (16-bit data-path), and the main thread of control is managed by an on-chip host processor. The programmable part is an 8 by 8 array of re-configurable cells (Figure 2), with multiple context words, operating in SIMD fashion. MorphoSys is targeted at image processing applications. Automatic target recognition and video compression (block motion estimation and discrete cosine transform) are some of the important tasks for which we have performed simulations. The system model and architecture details for the first implementation of MorphoSys (M1 chip) are described hereafter.

4.1 System Overview

Re-configurable Cell Array: The main component of MorphoSys is the Re-configurable Cell (RC) Array (Figure 2). It has 64 re-configurable cells, arranged as an 8 by 8 array. Each cell has an ALU/multiplier and register file (16-bit data-path). The RC Array functionality and interconnection network are configured through 32-bit context words. The context words are stored in a Context Memory in two blocks (one for rows and the other for columns). Each block has eight sets of sixteen contexts.

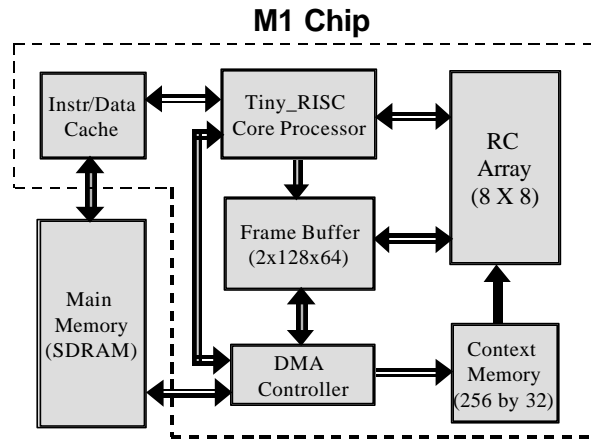


Figure 1: Block diagram of MorphoSys (M1 chip)

Host/Control processor: The controlling component of MorphoSys is a 32 bit RISC processor, called Tiny RISC. This is largely based on the design and implementation in [14]. Tiny RISC controls operation of the RC array, as well as data transfer to and from the array. Several new types of instructions were added to the Tiny RISC instruction set to enable it to perform these additional operations.

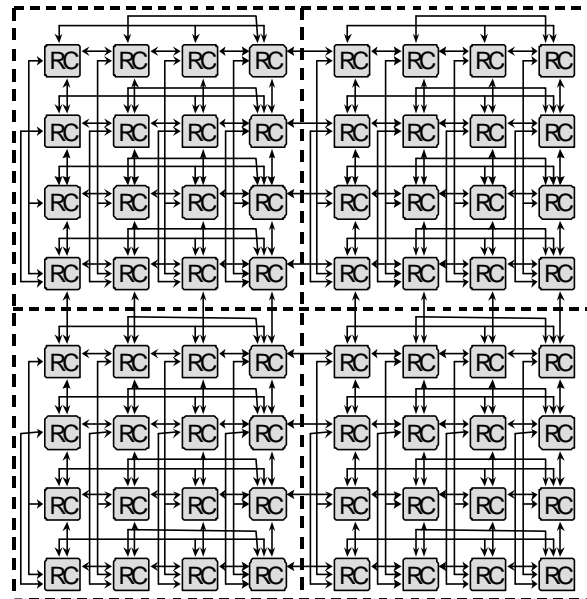


Figure 2: MorphoSys 8 x 8 Re-configurable Array

4.2 Program Flow

The MorphoSys system operates as follows: The Tiny RISC processor loads the configuration data from Main Memory into Context Memory through DMA Controller (Figure 1). Next, it enables the Frame Buffer to be loaded with image data from Main Memory. This data transfer is

also done by the DMA unit. At this point, both configuration as well as data are ready. Now, Tiny RISC issues instructions to RC Array for execution. These instructions specify the particular context (among the multiple contexts in Context Memory) to be executed. Tiny RISC can also enable selective functioning of a row/column, and can access data from selected RC outputs.

4.3 Features of MorphoSys

The RC Array follows the *SPMD* (Single Program Multiple Data) model of computation. Each row/column is configured by one context, which serves as an instruction word. However, each of these cells operates on different data. This model serves the target applications (i.e. applications with large number of data-parallel operations) for MorphoSys very well.

In brief, the important features of the MorphoSys computation model are:

Coarse-level granularity: Each cell of the RC array function is configured by the context word. The context word specifies one of several instruction opcodes for the RC array, and provides control bits for input multiplexers. It also specifies constant values that are needed for computations.

Considerable depth of programmability: The context memory can store up to 16 contexts corresponding to a specific row and 16 contexts corresponding to a specific column. Our design provides the option of broadcasting contexts across rows or columns.

Dynamic reconfiguration capability: This is achieved by changing some portion of the context memory while the RC array is executing contexts from a different portion. For example, while the RC array is operating on the 16 contexts in row broadcast mode, the other 16 contexts for column broadcast mode can be reloaded. Context loads and reloads are done through Tiny RISC instructions.

Local Interface: The control processor (Tiny RISC) and the RC Array are on the same chip. This prevents I/O limitations from affecting performance. In addition, the memory interface is through an on-chip DMA Controller, for faster data transfers between external memory and the Frame Buffer. It also helps in decreasing the configuration loading time.

4.4 TinyRISC Instructions for MorphoSys

Several new instructions (Table 1) were introduced in the Tiny RISC instruction set for effectively controlling the

MorphoSys RC Array operations. These instructions enable data transfer between main memory (SDRAM) and frame buffer, load configuration from main memory into context memory, and control RC array execution.

Table 1: Modified Tiny RISC Instructions for MorphoSys M1 chip

<i>CBCAST</i>	: Execute specific context in <i>RC Array</i>
<i>DBCB</i>	: Execute RC context, read two operand data into <i>RC Array</i> from <i>Frame Buffer</i>
<i>LDCTXT</i>	: Load context into <i>Context Memory</i>
<i>LDFB</i>	: Load data into <i>Frame Buffer</i> from memory
<i>RCRISC</i>	: Write data from <i>RC Array</i> to <i>Tiny RISC</i>
<i>SBCB</i>	: Execute RC context, read one operand data into <i>RC Array</i> from <i>Frame Buffer</i>
<i>STFB</i>	: Store data from <i>Frame Buffer</i> to memory
<i>WFB</i>	: Write data from specific column of <i>RC Array</i> into <i>Frame Buffer</i>

There are two categories of these instructions: DMA instructions and RC instructions. The DMA instruction fields specify load/store, memory address (indirect), number of bytes/contexts to be transferred and frame buffer or context memory address. The RC instruction fields specify address of context to be executed, address of frame buffer (if RC needs to read/write data) and broadcast mode (row/column). The instructions are summarized in Table 1.

5. RC Array Architecture

In this section, we describe three major features of MorphoSys. First, the architecture of each re-configurable cell is detailed (Figure 3), with description of different functional, storage and control components. Next, we discuss the context memory, its organization, field specification and broadcast mechanism. Finally, we describe the three-level hierarchical interconnection network of the RC array.

5.1 Re-configurable Cell Architecture

The re-configurable cell (RC) array is the programmable core of MorphoSys. It consists of 64 identical Re-configurable Cells (RC) arranged in a regular fashion to form an 8x8 array (Figure 2). The basic configurable unit, is the RC (Figure 3). Its functional model is similar to the data-path of a conventional processor, but the control is modeled after the configuration bits in FPGA chips. As Figure 3 shows, the RC comprises an ALU-multiplier, a shift unit, and two multiplexers for ALU inputs. There are registers at the output and for feedback, and a register file with four registers. A context word, loaded from Context Memory and stored in the context register (Section 5.2),

defines the functionality of the ALU and direction/amount of shift at the output. It provides control bits to input multiplexers and determines which registers are written after an operation. In addition, the context word (stored in the context register) can also specify an immediate value (referred to as a constant).

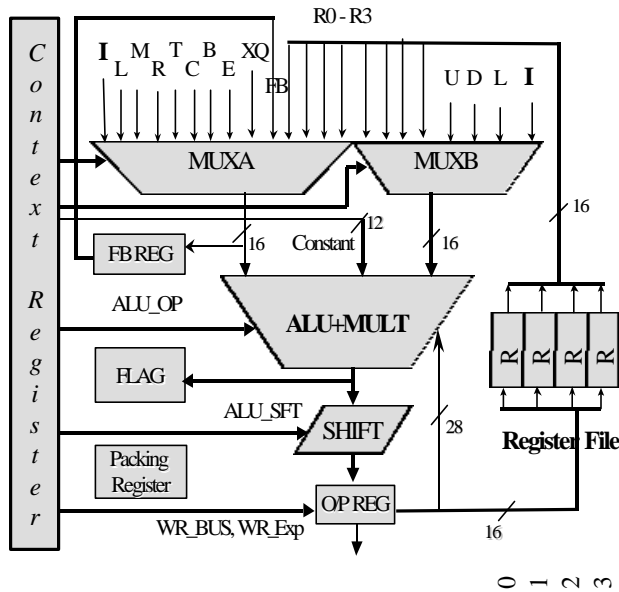


Figure 3 : Re-configurable Cell Architecture

ALU-Multiplier unit: The ALU has 16-bit inputs, and the multiplier has 16 by 12 bit inputs, producing an output of up to 28 bits. Externally, the ALU-multiplier has four input ports. Two ports, Port A and Port B are for data from outputs of input multiplexers. The third input (12 bits) takes a value from the constant field in the context register (Figure 4). The fourth port takes its input from the output register. The ALU has standard logic functions. Among its arithmetic functions are addition, subtraction and a function to compute absolute value of difference of two numbers. The ALU also has some functions that take one operand from port A, and the other from constant input port. The unit is capable of doing a multiply-accumulate operation in one cycle, wherein two data are multiplied and added to the previous output value. The ALU adder has been designed for 28 bit inputs. This prevents loss of precision during multiply-accumulate operation, even though each multiplier output may be much more than 16 bits, i.e. a maximum of 28 bits.

Input multiplexers: The two input multiplexers select one of several inputs for the ALU. Mux A is a 16-to-1 mux, whereas Mux B is an 8-to-1 mux (Figure 3). Mux A provides inputs from the four nearest neighbors, and from the other cells in the same row and column within the quadrant. It also provides an express lane input (as explained in sub-section on Interconnection network), array data bus input, a feedback input, a cross-quadrant input and

four inputs for register file. Mux B provides four register file outputs, array data bus input and inputs from three of the nearest neighbors.

Registers: The register file is composed of four registers (16-bit), which prove adequate for most applications. The output register is 32 bits wide (to accommodate intermediate results of multiply-accumulate instructions). The shift unit is also 32 bits wide and can perform logical right or left shifts of 1 to 15 bits (Figure 3). A flag register indicates sign of input operand at port A of ALU. It is zero for positive operands and one for negative operands. A flag is useful when the operation to be performed depends upon the sign of the operand, as in the quantization step during image compression. A feedback register is also available in case an operand needs to be re-used, as in motion estimation.

Custom hardware: This is used to implement special functions, especially bit-level function, for e.g. a one's counter or a packing register (for merging binary data into words).

5.2 Context Memory

The configuration information for the RC Array is stored in the Context Memory. Also, each RC has a Context Register, in which the current context word is stored. The Context Memory is organized into two blocks with each block having eight sets of sixteen contexts.

Context register: This register (32 bits), specifies the context (i.e. the configuration control word) for the RC. The Context Register is a part of each re-configurable cell, whereas the Context Memory (SRAM) is separate from the RC Array (Figure 1).

The different fields for the context word are defined in Figure 4. The field ALU_OP specifies ALU function. The control bits for Mux A and Mux B are specified in the fields MUX_A and MUX_B. Other fields determine the registers to which the result of an operation is written (REG #), and the direction (RS_LS) and amount of shift (ALU_SFT) applied to output. One interesting feature is that the context includes a 12 bit-field for the constant. If the ALU-Multiplier functions do not need a constant, an ALU-Multiplier sub-operation is defined. These sub-operations are used to expand the functionality of the ALU unit.

The context word also specifies whether a particular RC writes to its row/column express lane (WR_Exp). Whether or not the RC array will write out the result to Frame Buffer, is also specified by the context data (WR_BUS). (Figure 4)

The programmability of interconnection network is also derived from the context word. Depending upon the context, an RC can access the input of any other RC in its column or row within the same quadrant, or else select an input from its own register file. Functional programmability is achieved by configuring the row/column ALUs to perform functions as specified in the context word.

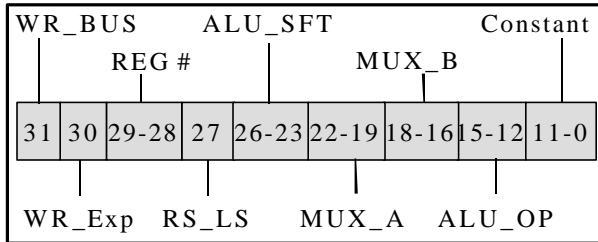


Figure 4: RC Context word definition

Context broadcast: The major focus of the RC array is on data-parallel applications, which exhibit a definite regularity. Based on this idea of regularity and parallelism, the context is broadcast to a row (column) of RCs. That implies that all eight RCs in a row (column) share the same context, and perform the same operations. For example, for DCT computation, eight 1-D DCTs need to be computed, across eight rows. This is easy to achieve with just eight context words to program eight rows of 64 RCs.

Context memory organization: The context memory is designed as an SRAM, with total memory of 256 (16X16) 32 bit words (Figure 1). The context may be broadcast across rows or columns. Corresponding to that, sixteen context sets/cells are needed (eight for row broadcast to eight rows, and eight for column broadcast to eight columns). The computation model for the RC supports multiple contexts. In other words, there are multiple context memory words for each of the above sixteen sets. Based on studies of relevant applications, a depth of sixteen for each context set has been found sufficient for most applications studied for this project. Each context word is 32 bits, there are sixteen words in a context set, and there are sixteen context sets, giving the above figures.

Dynamic reconfiguration: MorphoSys supports dynamic re-configuration. This implies that while the RC Array is executing a series of context words, another set of contexts may be reloaded from main memory through the DMA controller. The context depth allows the RC Array to operate continuously even when portions of the Context Memory need to be changed.

5.3 Interconnection Network

The RC interconnection network is comprised of three hierarchical levels.

RC array mesh: The underlying network throughout the array is a 2-D mesh. This provides nearest neighbor connectivity. Each RC is connected to its North, South, East and West neighbors (Figure 5).

Intra-quadrant (complete row/col) connectivity: The second layer of connectivity is at the quadrant level (a quadrant is a 4 by 4 RC group). In the current MorphoSys specification, the RC array has four quadrants (Figure 2). Within each quadrant, each cell has complete connectivity in two dimensions, as shown in Figure 5. Each cell can access the output of any other cell in its row (column).

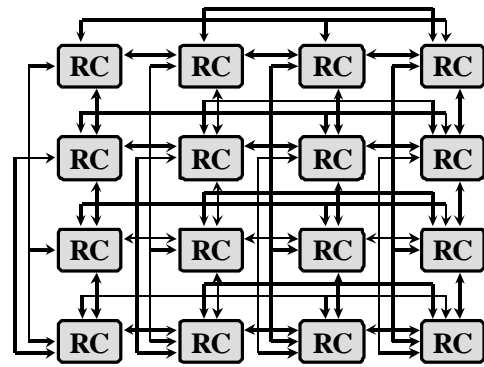


Figure 5: Connectivity within a quadrant

Inter-quadrant (express lane) connectivity: At the highest or global level, there are buses for routing connections between adjacent quadrants. These buses, also called express lanes, run across rows as well as columns. Figure 6 shows two express lanes going in each direction across a row. These lanes can supply data from any one cell (out of four) in a row (column) of a quadrant to other cells in adjacent quadrant but in same row (column). This means that up to four cells in a row (column) may access the output value of any one of four cells in the same row (column), but in an adjacent quadrant. The express lanes greatly enhance global connectivity. Even irregular communication patterns, that require extensive interconnections, can be handled efficiently. For e.g., an eight-point butterfly is accomplished in only three cycles.

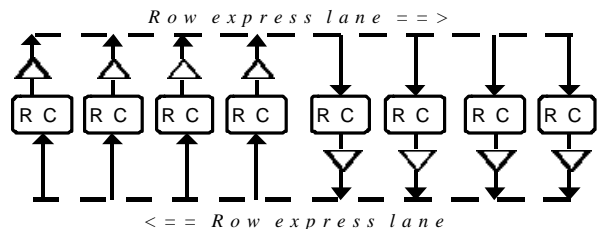


Figure 6: Express lane connectivity (between two groups of cells in same row but adjacent quadrants)

Data bus: A 128-bit data bus from Frame buffer to RC array is linked to column elements of the array. It provides two operands of eight bits to each of the eight column cells. It is possible to load two operand data (port A and port B) in an entire column in one cycle. Thus, only eight cycles are required to load the entire RC array. The outputs of RC elements of each column are written back to frame buffer through Port A data bus.

Context bus: When a Tiny RISC instruction specifies that a particular group of contexts (configuration words) be executed, these contexts must be distributed to each RC from the Context Memory. The context bus communicates this context data to the Context Register in each RC in a row (column). Each context word is 32 bits wide, and there are eight rows (columns), hence the context bus is 256 bits wide. When the row (column) contexts are activated, the same context word is broadcast to all RCs in a particular row (column).

6. MorphoSys: Application Mapping

In this section, we shall consider the mapping of several relevant applications to the MorphoSys architecture. These applications were chosen because of their computation-intensive nature. Among the video compression applications are motion estimation, DCT and quantization. Automatic target recognition is an important military application. We also provide performance estimates (based on a first order analysis) for each application.

6.1 Example: Motion Estimation

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 [15]. Block matching algorithm is also recommended by several standard committees (e.g., MPEG and H.261 standards) [16]. Among several possible searching methods, the expense of full search block matching (FSBM) is obviously greatest. FSBM, however, gives the optimal solution and low control overhead and is found an ideal candidate for VLSI implementation.

Typically, the mean absolute difference (MAD) criterion is the one implemented in VLSI due to its relative accuracy and compactness of implementation in hardware [16]. With the MAD criterion, the FSBM can be formulated as follows:

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

where p and q are the maximum displacements, $R(i, j)$ 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 $(N+p+q)^2$ 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 $(p+q+1)^2$ possible displacements within the search area. Table 2 shows the specification of a video codec.

Table 2 : Specification of a Video Codec

Image Size	352 x 240	Pixels
Frame Rate	15	Frames/s
Block size	8 x 8	Pixels
Max. Displacement	$-8 \leq m, n \leq 8$	Pixels

Figure 7 shows the configuration of RC array for FSBM computation. The operations of RC array are denoted row by row. Initially, one reference block and the search area associated with it are loaded into one set of the frame buffer, and then the RC array starts the matching process for the reference block resident in the frame buffer. During the computation, another reference block and the search area associated with it are loaded into the other set of the frame buffer so that the loading and computing time are overlapped.

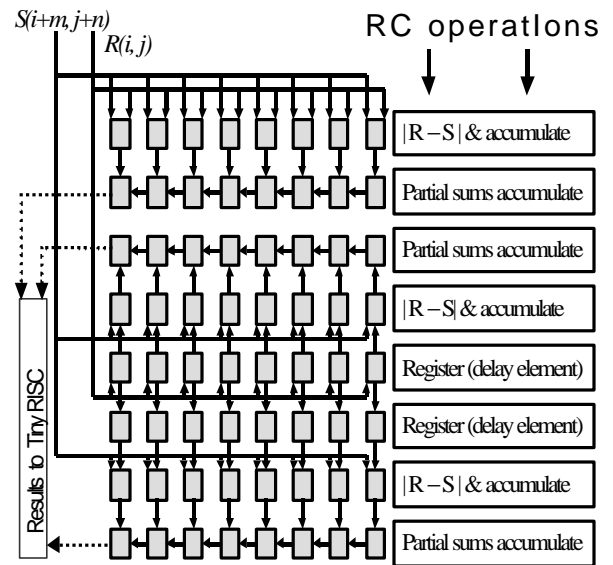


Figure 7: Configuration of RC Array for Full Search Block Matching

For each reference block, three consecutive candidate blocks are matched concurrently in the RC array. As shown in Figure 7, each RC in first, fourth, and seventh

row performs $P_j = S_{1 \leq i \leq 8} |R(i, j) - S(i+m, j+n)|$. The data of the reference block is sent to the first row and passed to the fourth row and seventh row through delay elements. The eight partial sums (P_j) generated by the first, fourth, and seventh row are then passed to the second, third, and eighth row respectively to perform $MAD(m, n) = S_{1 \leq i \leq 8} P_j$. Subsequently, three MAD values corresponding to three candidate blocks are sent to Tiny RISC for comparison, and the RC array starts the block matching of another three candidate blocks.

Computation cost: Based on the computation model shown above, it takes ten clock cycles to finish the matching of three candidate blocks. There are $(8+8+1)^2 = 289$ candidate blocks in each search area, and it takes a total of $(102 \times 10) 1020$ cycles to finish the matching of the whole search area. According to the specification of Table 2, the image size is 352×240 pixels which consists of $44 \times 30 = 1320$ reference blocks. The total processing time of an image frame is $1320 \times 1020 = 1346400$ cycles. The anticipated clock rate of MorphoSys is 100 MHz, so the computation time would be $1346400 \times 10 \times 10^{-9} \approx 13.5$ ms. This is much smaller than frame period of $1/15$ s shown in Table 2.

Performance Analysis: MorphoSys performance is compared with the ASIC architecture implemented in [15] and Intel MMX instructions [24] using the criterion of the number of cycles needed for matching one 8×8 reference block against its search area of 8 pixels displacement. The result is shown in Table 3.

Table 3: Performance Comparison

System	MorphoSys (8X8)	ASIC ^[15]	MMX ^[24]
(cycles)	1020	581	28900

The processing cycles of MorphoSys are about two times compared to ASIC design in [15], however, MorphoSys provides the advantage of hardware reuse. The number of cycles needed for MMX is about 28 times more than the cycles needed for MorphoSys, which shows the greater effective computing power of MorphoSys.

6.2 Example: Discrete Cosine Transform

The Discrete cosine transform (DCT) [17] and inverse discrete cosine transform (IDCT) form an integral part of the JPEG [19] and MPEG [20] standards. MPEG encoders use both DCT and IDCT, whereas IDCT is used in MPEG decoders. Compression is achieved in MPEG through the combination of DCT and quantization.

Fast algorithms for DCT: In the following analysis, we consider one of several fast DCT algorithms. This algorithm [18] for a fast 8-point 1-D DCT involves 16 multiplications and 26 additions, leading to 256 multiplications and 416 additions for a 2-D implementation. The 1-D algorithm is first applied to rows (columns) of input 8×8 image block, and then to columns (rows). The eight row (column) DCTs may be computed in parallel, for high throughput.

Mapping to RC Array: The block size for DCT in most image and video compression standards is 8×8 . The RC array also has a size of 8×8 . Thus, each pixel of the image block may be directly mapped to each RC. The input block is loaded into the array, such that one pixel value is stored in each RC. The next step is to perform eight 1-D DCTs along the rows (columns). In other words, row (column) parallel operations are required. The context for configuration needs to be broadcast along columns (rows). Under this condition, different RCs within a row (column) of the array communicate using three-layer interconnection network to compute outputs for 1-D DCT. The coefficients needed for the computation are provided as constants in context words. When 1-D DCT along rows (columns) is complete, the next step is to compute 1-D DCT along columns (rows). For column parallel operation, context words are broadcast along rows. Once again, RCs within a column communicate using the three-layer interconnections, to compute the results. As previously, coefficients are provided through context words. At the conclusion of this step, the 2-D DCT is complete.

Some points may be noted: first, all rows (columns) perform the same computations, hence they can be configured by a common context (thus enabling broadcast of context word). Second, the RC array provides the option of broadcasting context either across rows or across columns. This allows computation of second 1-D DCT without transposing the data. Elimination of the transpose operation saves a considerable amount of cycles, and is important for high performance.

Sequence of steps:

Loading the input data: The first step is to load the input image pixel block into the RC array. This data block is previously loaded into the frame buffer from external memory. The data bus between the frame buffer and the RC array is wide enough to load eight pixels at a time. The entire block is loaded in eight cycles.

Row-column approach: Following the separability property, the 2-D DCT is carried out in two steps. First, a 1-D DCT along rows is computed. Next, we compute 1-D DCT on the results along columns (Figure 8). Each sequence of 1-D DCT [18] involves:

- i. *Butterfly computation:* Once the data is available, the next step is to compute the intermediate variables. The inter-quad connectivity layer of express lanes enables completion in three cycles.
- ii. *Computation and re-arrangement:* In the next few steps, the actual computations pertaining to 1-D DCT algorithm [2] are performed. For 1-D DCT (row/column), the computation takes six cycles. An extra cycle is used for re-arrangement of computed results.

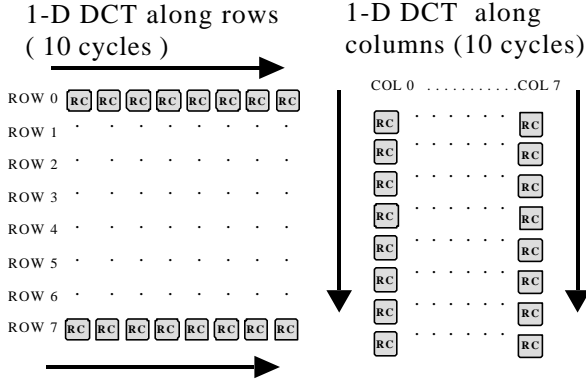


Figure 8: Computation of 2-D DCT across rows/columns (without transposing)

Computation cost: The cost for computing 2-D DCT on an 8x8 block of the image is as follows: 8 cycles for data input, 6 cycles for butterfly, and 12 cycles for both 1-D DCT computations. 2 cycles are used for re-arrangement of data. For 1024 by 768 pixel image, it would take 3.44 ms @ 100 MHz to complete the 2-D DCT computation.

Performance analysis: MorphoSys requires 21 cycles to complete 2-D DCT on 8x8 block of pixel data. This is in contrast to 240 cycles required by Pentium MMXTM [23]. Even this number is achieved by using special MMX instructions in Pentium instruction set, assuming that all data is present in the cache. MorphoSys takes 8 cycles to load data and 8 for write-back while Pentium cache miss penalty is 12 cycles. Notably, MorphoSys performance scales linearly with the array size. For a 256 element RC array, the throughput for 2-D DCT algorithm would increase four times, giving an effective performance of 5 cycles per 8x8 block. The performance figures are summed up in Table 4.

Table 4: DCT Performance Comparison (cycles)

MorphoSys (8 X 8)	MorphoSys 2 (8 X 8)	MorphoSys 4 (8 X 8)	Pentium MMX TM
21	10	5	240

6.3 Example: Automatic Target Recognition (ATR)

Automatic Target Recognition (ATR) is the machine function of detecting, classifying, recognizing, and identifying an object without the need of human intervention. ATR is one of the most computation intensive applications in existence. The ACS Surveillance Challenge has been quantified as the ability to search 40,000 square nautical miles per day with a resolution of one meter [21]. The computation levels for this problem when the targets are partially obscured reaches the hundreds-of-teraflops range. By considering the number of computations that must be carried out for real-time images, it is obvious that a hardware, which provides great computing power, is essential for solving ATR problem. In the previous sections, we have shown that MorphoSys with 8x8 RC array can perform FSBM motion estimation on a 352x240 pixels image at 70 frames per second. We believe that MorphoSys has a great opportunity to provide the computing power for ATR.

Currently, there are many algorithmic choices available to implement an ATR system [22]. A simplified ATR processing model used in this paper is shown in Figure 9 [23]. We assume that Synthetic Aperture Radar (SAR) images generated by the radar imager in real time are used. SAR images consist of 8-bits pixels and are input to a focus-of attention processor to identify the regions of interest (ROI). These ROI's are thresholded to generated binary images and the binary images are then matched against binary target templates.

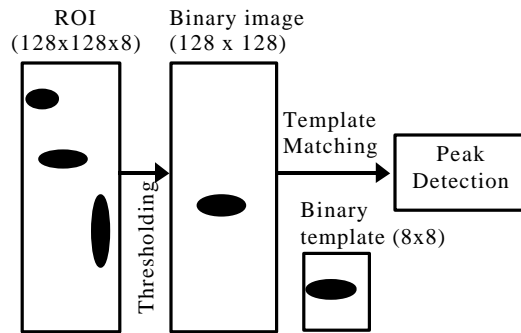


Figure 9: A Simplified ATR Processing Model

Before the thresholding, 64 pixels of ROI are loaded into RC array. Each pixel is then compared with the threshold value T and is set to a binary data based on the following equation:

$$\begin{aligned} \text{If } A_{ij} - T > 0, & \quad A_{ij} \leftarrow 1 \\ \text{If } A_{ij} - T < 0, & \quad A_{ij} \leftarrow 0, \end{aligned} \quad \text{where } A_{ij} \text{ represents the 8-bit pixels in ROI}$$

The threshold value is assumed to be a pre-selected constant and is put in the context at compile time. Because 2's complement representation is used, the most significant bit of the output register represents the result of the thresholding. There is an 8-bits packing register in each RC of the first column. These registers are used to collect the thresholding results of the RCs in the same row. The data in the packing registers are then sent back to the frame buffer, and another 64 pixels of ROI are loaded to RC array for thresholding.

After the thresholding, a 128x128 binary image is generated and is stored in the frame buffer. This binary image is then matching against the target template. Each target consists of 72 templates at five degree rotations. Each row of the 8x8 target template is packed as an 8-bits number and eight templates are resident in the RC array concurrently. The template matching is similar to the FSBM described in the previous section. All of the candidate blocks in the ROI are correlated with the target template. One column of the RC array performs matching of one target template and eight target templates can be matched concurrently in the 8x8 RC array. Figure 10 illustrates the matching process in each RC.

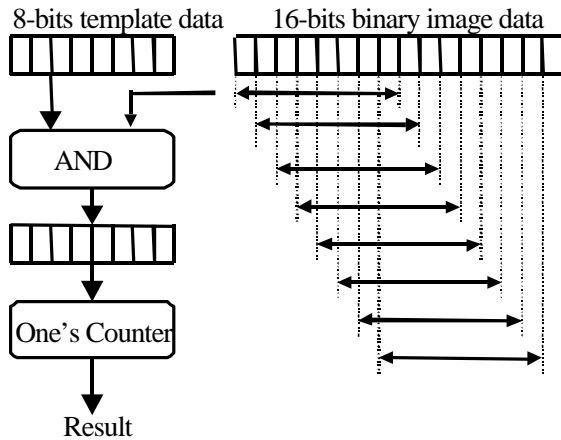


Figure 10: Matching Process in Each RC

In order to perform bit-level template matching, two bytes (16 bits) of image data are input to a RC. In the first step, the 8 most significant bits of the image data are ANDed with the template data and a special adder tree is used to count the number of 1's of the ANDed output. The result is passed to the peak detector. Then, the image data is shifted left one bit and the process is repeated again to perform the matching of the second block. After the image data is shifted eight times, new 16-bit data is loaded and the RC Array starts another matching of eight consecutive candidate blocks.

Performance analysis: For performance analysis, we choose the system parameters that are implemented in [23]. The image size of ROI is 128x128 pixels, and the template size is 8x8 bits. For 16 pairs of target templates, the

processing time is approximately 16 ms in MorphoSys assuming a clock of 100 MHz. This is more than one order of magnitude less than the 210 ms processing time for the system implemented in [23], which uses Xilinx XC4010 FPGAs. The performance figures are listed in Table 5.

Table 5: ATR Performance Comparison (ms) (MorphoSys @ 100 MHz)

MorphoSys (8 X 8)	MorphoSys 2 (8 X 8)	MorphoSys 4 (8 X 8)	Xilinx FPGA
16	8	4	210

7. Interactive Software Environment

The MorphoSys re-configurable system has been specified in behavioral VHDL. We have coded a simple assembler-like parser, *mload*, for context generation. This context is then used as input for simulation. We have simulated motion estimation algorithm, the discrete cosine transform, and automatic target recognition among others.

A graphical user interface, *mView*, has been prepared for programming and observing simulation behavior for MorphoSys. This interface is based on Tcl/Tk [26]. It operates in one of two modes: programming mode or simulation mode. In the programming mode, the user sets functions and interconnections corresponding to each context (row/column broadcasting) for the application. Then, *mView* generates a context file for representing the user-specified application.

In the simulation mode, *mView* takes as input either a context file, or a simulation output file. For either of these, it provides a graphical display of the interconnections and RC outputs at each cycle of the simulation. Use of *mView* has been found to decrease the programming time, and also increase debugging efficiency.

8. Conclusions

In this paper, we presented a new model of re-configurable architecture in the form of MorphoSys, and mapped several applications to it. The results have shown that this architectural model enables impressive performance for several target applications. We plan to implement MorphoSys on an actual chip for practical laboratory evaluation. We conceive that MorphoSys may be the precursor of a generation of general-purpose processors that have a specialized re-configurable component, designed for image and signal processing applications.

9. Acknowledgements

This research is supported by Defense and Advanced Research Projects Agency (DARPA) of the Department of Defense under contract number F-33615-97-C-1126. We express thanks to Prof. Walid Najjar for his incisive comments. We acknowledge the contributions of Maneesha Bhate, Matt Campbell, Benjamin U-Tee Cheah, Alexander Gascoigne, Nambao Van Le, Robert Powell, Rei Shu, Lingling Sun, Cesar Talledo, Eric Tan, Tom Truong, and Tim Truong; all of whom have been associated with the development of MorphoSys.

References:

1. Stephen 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. M. Gokhale et al, "Building and Using a Highly Parallel Programmable Logic Array," *IEEE Computer*, pp. 81-89, Jan. 1991
3. P. Bertin, D. Roncin, and J. Vuillemin, "Introduction to Programmable Active Memories," in *Systolic Array Processors*, J. McCanny, J. McWhirther and E. Swartzlander, eds., Prentice Hall, 1989, pp. 300-309
4. W. H. Mangione-Smith et al, "Seeking Solutions in Configurable Computing," *IEEE Computer*, pp. 38-43, December 1997
5. D. C. Chen and J. M. Rabaey, "A Re-configurable Multi-processor IC for Rapid Prototyping of Algorithmic-Specific High-Speed Datapaths," *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 12, December 1992
6. R. Hartenstein and R. Kress, "A Datapath Synthesis System for the Re-configurable Datapath Architecture," *Proc. of Asia and South Pacific Design Automation Conference*, 1995, pp. 479-484
7. E. Tau, D. Chen, I. Eslick, J. Brown and A. DeHon, "A First Generation DPGA Implementation," *FPD'95, Canadian Workshop of Field-Programmable Devices*, May 29-June 1, 1995
8. E. Mirsky and A. DeHon, "MATRIX: A Re-configurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources," *Proc. IEEE Sym. on FPGAs for Custom Comp. Mach.*, 1996, pp.157-66
9. J. R. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Re-configurable Co-processor," *Proc. of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1997
10. C. Ebeling, D. Cronquist, and P. Franklin, "Configurable Computing: The Catalyst for High-Performance Architectures," *Proc. of IEEE Int'l Conference on Application-specific Systems, Architectures and Proc.*, July 1997, pp. 364-72
11. C. Ebeling, D. Cronquist, P. Franklin, J. Secosky, and S. G. Berg, "Mapping Applications to the RaPiD configurable Architecture," *Proc. IEEE Symposium of Field-Programmable Custom Computing Machines*, Apr 1997, pp. 106-15
12. E. Waingold et al, "Baring it all to Software: The Raw Machine," *IEEE Computer*, Sep 1997, pp. 86-93
13. E. Waingold et al, "The RAW Benchmark Suite: computation structures for general-purpose computing," *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, FCCM 97, 1997, pp. 134-43
14. A. Abnous, C. Christensen, J. Gray, J. Lenell, A. Naylor and N. Bagherzadeh, "Design and Implementation of the Tiny RISC microprocessor," *Microprocessors and Microsystems*, Vol. 16, No. 4, pp. 187-94, 1992
15. C. Hsieh and T. Lin, "VLSI Architecture For Block-Matching Motion Estimation Algorithm," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 2, pp. 169-175, June 1992.
16. A. Bugeja and W. Yang, "A Re-configurable VLSI Coprocessing System for the Block Matching Algorithm," *IEEE Trans. On VLSI Systems*, vol. 5, September 1997.
17. N. Ahmed, T. Natarajan, and K.R. Rao, "Discrete cosine transform," *IEEE Trans. On Computers*, vol. C-23, pp. 90-93, Jan 1974
18. W-H Chen, C. H. Smith and S. C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform," *IEEE Trans. on Comm.*, vol. COM-25, No. 9, September 1977
19. *ISO/IEC JTC1 CD 10918*. Digital compression and coding of continuous-tone still images – part 1, requirements and guidelines, ISO, 1993 (JPEG)
20. *ISO/IEC JTC1 CD 13818*. Generic coding of moving pictures and associated audio: video, ISO, 1994 (MPEG-2 standard)
21. *Challenges for Adaptive Computing Systems*, Defense and Advanced Research Projects Agency, at www.darpa.mil/ito/research/acs/challenges.html
22. J. A. Ratches, C. P. Walters, R. G. Buser, and B. D. Guenther, "Aided and Automatic Target Recognition Based Upon Sensory Inputs From Image Forming Systems," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, September 1997.
23. J. Villasenor, B. Schoner, K. Chia, C. Zapata, H. J. Kim, C. Jones, S. Lansing, and B. Mangione-Smith, "Configurable Computing Solutions for Automatic Target Recognition," *Proceedings of IEEE Conference on Field Configurable Computing Machines*, April 1996.
24. *Intel Application Notes for Pentium MMX*, <http://developer.intel.com/>
25. *Xilinx, the Programmable Logic Data Book*, 1994
26. *Practical Programming in Tcl and Tk*, 2nd edition, by Brent B. Welch, Prentice-Hall, 1997