MorphoSys: Case Study of A Reconfigurable Computing System Targeting Multimedia Applications

Hartej Singh, Guangming Lu, Ming-Hau Lee, Fadi Kurdahi, Nader Bagherzadeh
University of California, Irvine
Dept of Electrical & Computer Eng.,
Irvine, CA 92697-2625
hsingh, glu, mlee, kurdahi, nader @ece.uci.edu

Eliseu Filho
Dept of Systems and Computer Engineering, COPPE/Federal University of Rio De Janeiro,
Rio de Janeiro, RJ Brazil
eliseu@cos.ufrj.br

Rafael Maestre
Dept. de Arquitectura de Comp. y Automatica, Escuela Superior de Informatica, Universidad Complutense,
28040, Madrid, Spain
maestre@dacya.ucm.es

ABSTRACT

In this paper, we present a case study for the design, programming and usage of a reconfigurable system-on-chip, MorphoSys, which is targeted at computation-intensive applications. This 2-million transistor design combines a reconfigurable array of cells with a RISC processor core and a high bandwidth memory interface. The system architecture, software tools including a scheduler for reconfigurable systems, and performance analysis (with impressive speedups) for target applications are described.

Keywords

Reconfigurable processors, SIMD, dynamic configuration, automatic target recognition, scheduling, MPEG-2

1. INTRODUCTION

Reconfigurable computing represents an intermediate approach between the extremes of Application Specific Integrated Circuits (ASICs) and general-purpose processors. Reconfigurable systems combine a reconfigurable-hardware unit with a software-programmable processor. These systems allow customization of the reconfigurable processing unit, which helps meet real-time computational requirements of an application. Designing a reconfigurable system and its software environment for enhanced performance over a wide range of applications is a challenging research problem.

This paper describes the MorphoSys reconfigurable computing system-on-chip that has been designed for multimedia applications having data-parallelism, and high throughput constraints, such as video compression. Section 2 presents the MorphoSys architecture, while Section 3 compares it with related work. The design methodology is illustrated in Section 4, and programming and CAD tools are presented in Section 5. Section 6 refers to application performance analysis, and Section 7 lists the conclusions.

DAC 2000, Los Angeles, California

2. ARCHITECTURE OVERVIEW

The MorphoSys architecture (Figure 1) includes a reconfigurable processing unit *(RC Array)*, a general-purpose (core) processor *(TinyRISC)* and a high bandwidth memory interface implemented as a single chip. Based on the data-parallel nature of target applications, the reconfigurable component is organized as an SIMD array of coarse-grain Reconfigurable Cells (RCs) to handle computation-intensive operations. The TinyRISC performs sequential processing and controls operation of the RC Array. The *Context Memory* stores the RC Array configuration program, while the *Frame Buffer* is a specialized streaming buffer. The *DMA controller* performs data transfers between external memory and the RC Array. These components are shown in Figure 1.

2.1 MorphoSys Components

RC Array: This is an array of processing elements called reconfigurable cells (RCs). Considering that many target applications (video compression, etc.) tend to be processed in clusters of 8x8 data elements, the RC Array has 64 cells in a 2-D matrix (Figure 2). This organization helps exploit the inherent parallelism of an application for greater throughput.

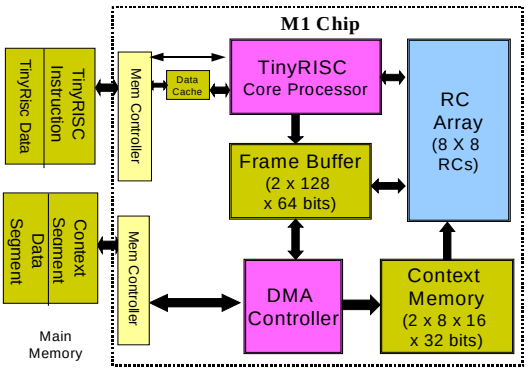

Figure 1: MorphoSys M1 System-on-Chip

The RC Array follows the *SIMD* model. All RCs in the same row or column share the configuration (context word) as illustrated in Figure 2. But, each RC operates on different data. Sharing the context word across a row or column is useful for data-parallel applications. A reconfigurable three-layer inter-connection network facilitates fast data exchange between the RCs.

Each RC incorporates a 28-bit ALU, a 16x12 bit multiplier, a shift unit, 16-input and 8-input multiplexers and a 16-bit register file.

A 32-bit *context register* stores the current context word and provides configuration signals for the RC functional units.

TinyRISC control processor: Most target applications involve some sequential processing. Therefore, the TinyRISC, a RISC processor [1], is integrated in the system. This processor has a 4-stage pipeline with a 32-bit ALU, register file and an on-chip data cache. The TinyRISC coordinates system operation using specific instructions (Table 1) that have been added to the TinyRISC ISA to activate the DMA Controller to transfer data, and provide control signals to Frame Buffer and RC Array for executing applications.

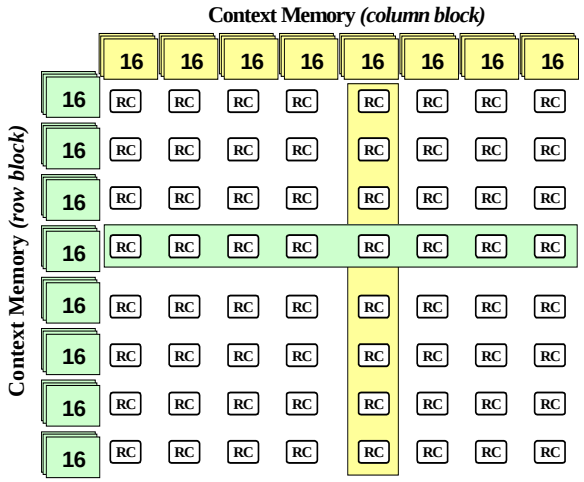

Figure 2: Context Memory (shaded) and Context Propagation to 8x8 RC Array

Context Memory: The Context Memory (CM) stores the configuration program (context words) for the RC Array. It is organized into 2 *context blocks*, with each block containing 8 *context sets*. Each context set has 16 *context words* (Figure 2).

The focus of the RC Array is on data-parallel applications, which exhibit regularity. Therefore, the context word is broadcast on a row or column basis. The context words from the *CM row block* are broadcast along the rows, while context words from the *column block* are broadcast along the columns. Each block has eight context sets and each context set is associated with a specific row (or column) of the RC Array. The context word from the context set is broadcast to each of the eight RCs in the corresponding row (or column). Thus, all RCs in a row (or column) share a context word and perform the same operations.

The CM provides a context word to the *Context Register* in each RC every clock cycle. The data provided to the RC is also registered, to prevent setup or hold time violations. The context word specifies ALU function, control bits for input multiplexers, the registers to store the result of an operation, and the direction and amount of shift applied to output. Corresponding to either row/column broadcast of the context word, a set of eight context words can specify the complete configuration *(context plane)* for the RC Array. As there are sixteen context words in a context set, up to sixteen context planes may be simultaneously resident in each of the two CM blocks. The CM supports *dynamic reconfiguration*, therefore when the context needs to be changed, it can be updated concurrently with RC Array execution. Dynamic reconfiguration enables the reduction of effective reconfiguration time to zero.

Frame Buffer and DMA Controller: The high parallelism of RC Array will be ineffective if the memory interface is unable to transfer data at an adequate rate. Hence, an innovative memory interface consisting of the Frame Buffer (FB) and DMA controller is incorporated in the system.

The FB serves as a data cache for the RC Array. This buffer has two sets, each of which further consists of two banks of memory. Each bank has 64x8 bytes of storage. The double-set FB enables overlapping of computation with data transfers. Data from one set is used for computation, while fresh data is loaded into the second set. This makes memory accesses transparent to the RC Array and greatly benefits MorphoSys performance. The Frame Buffer is *byte-addressable*, and can provide any 8 consecutive data bytes from each of the 2 banks in a set to the RC array in one cycle.

Interconnection Network: The RC Array has a hierarchical interconnection network with three layers. The basic network connects all RCs in a 2-D mesh providing nearest neighbor connections. The second layer provides complete row and column connectivity within a quadrant (a 4x4 RC group). Figure 3 depicts these interconnection layers for a quadrant. Each RC can access data from any other RC in its row or column in the same quadrant.

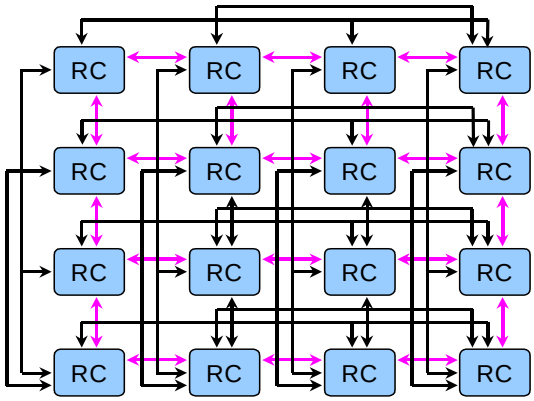

Figure 3: RC Array Quadrant Interconnection Network

At the global level, buses called express lanes run across rows as well as columns. These lanes supply data from any one cell (out of four) in a row (or column) of a quadrant to other cells in adjacent quadrant but same row (or column). Thus, up to four cells in a row (or column) may access the output value of any one of four cells in the same row (or column) of an adjacent quadrant.

Data bus: A 128-bit bus from the Frame Buffer to the RC array columns provides two 8-bit operands to each of the 8 column RCs. Eight cycles are required to load the entire RC array.

Context bus: The context words in the Context Memory are distributed to each RC through the row and column context buses.

2.2 System Control Mechanism

The RC Array is configured through context words. Each context word specifies an instruction opcode for the RC. Context words are stored in the Context Memory. All RCs in the same row/column share the same context word but each RC operates on different data (SIMD model). The TinyRISC controls system execution by initiating CM and FB loads using DMA controller,

and providing address and control signals to FB, CM and RC Array through the new TinyRISC instructions (listed in Table 1).

LDCTXT	Load context words to Context Memory.
LDFB (STFB)	Load data from main memory to Frame Buffer (FB) (store data from FB to main memory)
CBCAST	Execute RC Array through context broadcast
SBCB	Execute RC Array with context broadcast, and also provide 64-bit data from one FB bank to RC Array
DBCBC	Execute RC Array with column context broadcast, provide 128-bit data from both FB banks
DBCBR	Execute RC Array with row context broadcast, provide 128-bit data from both FB banks
WFB	Write data back to FB (in address from register file)
RCRISC	Write one 16-bit data from RC Array to TinyRISC

Table 1 : New TinyRISC Instructions

During operation of the MorphoSys system for executing specific applications (e.g. multimedia, encryption) on the RC Array, the instructions in Table 1 are used as follows to:

a) Load context words into Context Memory from main memory

b) Load computation data into the first set of Frame Buffer (FB) from external main memory

c) Execute RC Array with context broadcast and concurrently load data into second FB set

d) *Execute RC Array, concurrently store data from first FB set to memory, and load new data into FB set.*

3. RELATED WORK

There are two major classes of reconfigurable systems: *fine-grain* (processing units have datapath widths of a few bits such as *DPGA* [20]) and *coarse-grain* (processing elements have data-paths of 8 bits or more). Examples of the latter are *PADDI* [3], *MATRIX* [13], *RaPiD* [5], and *REMARC* [14]. *MorphoSys* is compared with these processors, since it is also a coarse-grain architecture. Many of the previous designs are only in software, whereas MorphoSys has been developed from software down to the physical (chip) level.

PADDI [3] 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 nature since each column/row performs the same function. PADDI has static reconfiguration but MorphoSys has dynamic reconfiguration. MorphoSys has more context planes (32) than PADDI (eight).

MATRIX [13] proposes the design of a basic reconfigurable unit (BFU). This 8-bit BFU unifies resources used for instruction storage with resources needed for computation and data storage, and assumes a multi-level interconnection network. But the 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, memory interface, I/O bandwidth, and performance unknown.

The *RaPiD* [5] design is organized as a linear array of reconfigurable processing units, which is not appropriate for block-based applications (for example, 2-D processing). Due to its organization, the potential applications of RaPiD are those of a systolic nature or applications that can be easily pipelined. A complete system implementation is not described.

REMARC [14] is similar to MorphoSys and targets the same class of applications. However, it has static reconfiguration, lacks a direct interface to external memory, and data transfers cannot be overlapped with computation. Performance figures for applications (e.g. 2-D IDCT) reflect that REMARC is significantly slower than MorphoSys (Section 6).

In summary, the major features of MorphoSys are:

Integrated model: MorphoSys is a complete system-on-a-chip except for main memory.

Multiple contexts on-chip: MorphoSys has multiple contexts with dynamic and single-cycle reconfiguration.

On-chip general-purpose processor: This processor, which is also the system controller, enables efficient execution of complex applications that include both serial and parallel tasks.

Innovative memory interface: In contrast to other reconfigurable systems, MorphoSys provides overlap of data transfers with computation through a two-set data buffer.

4. IMPLEMENTATION METHODOLOGY

MorphoSys components are implemented [11] in 0.35 micron technology with four metal layers using the twin approaches of custom design and standard cell design. The components on the critical path (e.g. RC) or those with a regular structure (e.g. Context Memory, and Frame Buffer) are custom designed so that they may be optimized for delay and area. This enables MorphoSys to be designed for operation at 100 MHz. The control intensive components (TinyRISC and DMA controller) are designed using Synopsys logic synthesis and Mentor Graphics layout synthesis tools. The complete design has about two million transistors and occupies an area of 170 sq. mm (Figure 4).

The top two metal layers are reserved for power routing and routing between the component blocks. Only two layers are used for routing within a component such as the RC. Custom designs are verified using switch-level and transistor-level simulators. For synthesized blocks, gate-level, switch-level and transistor-level simulators are used for functional and timing verification.

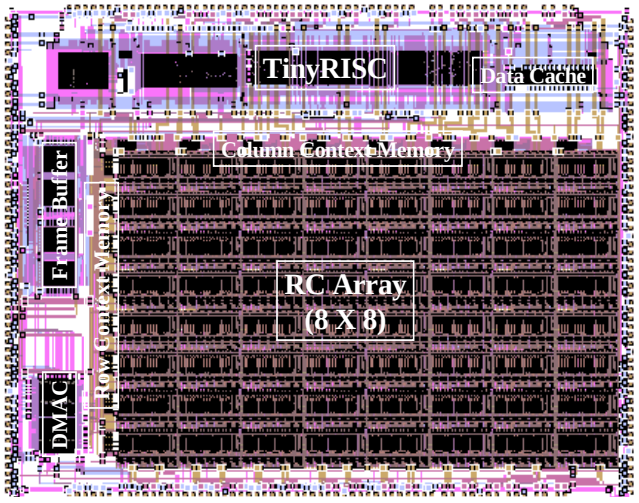

Figure 4: MorphoSys M1 Chip Layout

Global Routing: This network consists of the RC Array interconnections, data/context buses, clock tree, and the power supply network. Automatic routing tools were unable to maintain the regularity of the global routing. Hence, this routing layout was accomplished using a combination of procedural and custom design approach. More implementation details are in [11].

The clock tree within the RC Array was designed as a custom layout using an H tree pattern. Buffers were then inserted in paths to other components of MorphoSys (e.g. TinyRISC, DMAC, and Frame Buffer) to minimize the skew. The width of metal layers used for power was based on the technology electron-migration rules and worst-case power consumption. The M1 chip taped out in Dec. 1999.

5. PROGRAMMING AND CAD TOOLS

For each application, two different kinds of programs are required. These are the RC Array configuration program (context words) and the TinyRISC instruction program.

Context Generation: This is done using an assembler-parser, *mLoad*. An assembly format has been specified for the context words, and the user defines the application computations in this syntax, either directly or through a GUI, *mView* (Section 5.2). *mLoad* converts this assembly code into the context program.

TinyRISC program: This is generated through a C language compiler, *mcc*, developed using the SUIF compiler environment [19]. Currently, the compiler produces code for the application tasks to be executed on the TinyRISC. Special TinyRISC instructions (Table 1) are then added by the designer to execute the context for the RC Array. Figure 5 depicts the software environment with its different components explained below.

5.1 MorphoSys Simulators

Two different simulators have been developed to analyze application mapping and performance issues for different applications, such as Motion Estimation (video compression), Discrete Cosine Transform (DCT), and Automatic Target Recognition (ATR). Both simulators use configuration program, TinyRISC program and the input data to execute applications. *mULATE* is a C++ simulator while *MorphoSim* is the VHDL-based simulator. The former enables simulation with breakpoints, and displays the contents of the RC Array, FB, CM, and data cache.

5.2 RC Array GUI: *mVIEW*

A graphical user interface, *mView*, has been developed for the RC Array. It may be used to generate the configuration program for an application or to simulate an application and its data movements and functions across the RC Array. The user provides input for each application in terms of specification of operations and data sources or destinations for each RC. *mView* then generates context code for the RC Array. It operates in one of two modes: programming mode or simulation mode.

In the programming mode, *mView* generates the application context file based on the user input and specifications. For the simulation mode, *mView* takes an application context file, or VHDL simulation output file and displays the state of each RC as the application is executed per cycle. *mView* is an efficient tool for verifying or debugging simulation runs.

5.3 Code Generation for MorphoSys

As part of an ongoing effort to develop tools for automatic mapping and code generation for MorphoSys, the next step is to enhance the *mcc* compiler to support inline assembly code for application kernels that are mapped to the RC Array. The application code will thus constitute of TinyRISC functions and RC Array functions. The RC Array functions will be coded using functions representing the special TinyRISC instructions and the corresponding context code will be provided in a configuration library. From this code with inline assembly functions, the *mcc* compiler will generate TinyRISC object code. At an advanced level, MorphoSys would perform online profiling and dynamic adjustment of reconfiguration profile for enhanced performance.

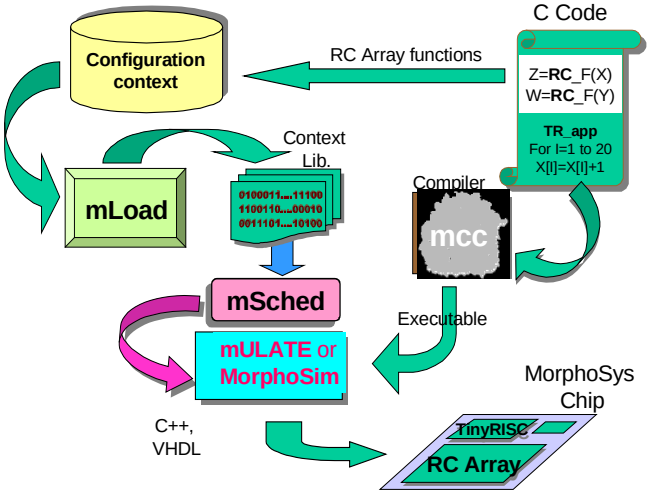

Figure 5: Software Environment for MorphoSys

5.4 Application Kernel Scheduling: *mSched*

Problem Overview: A typical complex application consists of a sequence of tasks that are executed repeatedly. These tasks or kernels are well defined, and can be executed independently after the previous tasks in the execution flow-graph. Once the context program corresponding to the kernels has been determined as explained earlier, the kernel execution and data movements have to be scheduled for minimizing total execution time.

This scheduling problem for reconfigurable computing is a new and significant issue. Previous approaches for scheduling tasks in reconfigurable computing are high-level synthesis technique extensions for specific features of reconfigurable systems [6,10,15,21]. However, these fail to consider unique aspects of reconfigurable computing, such as multiple contexts, data memories, and variable reconfiguration time.

We have developed an algorithm (implemented as *mSched*) to schedule the kernels for optimized performance within the architectural constraints of MorphoSys. This assumes that the kernel timing is available, the application flow graph is known, the kernel configuration does not exceed the context memory size and two kernels can not be executed simultaneously.

An application example is MPEG, a video compression standard, whose encoder kernel sequence is given in Figure 6a. The kernels are Motion Estimation (ME), Motion Compensation (MC),

Discrete Cosine Transform (DCT), Quantization (Q), Inverse Quant. (IQ), Inv. DCT (IDCT), and Inv.Motion Comp. (IMC).

Although each application imposes some constraints to the execution order (DFG), it is possible to have different execution sequences that result in different total execution times. Also, partitioning of the execution graph may improve the final result. Thus, it is not possible to know the best solution in advance. It is necessary to explore the design space, using the three criteria of: *Context reloading* (minimize), *Data reuse* (maximize) and *Overlap of computation and data transfers* (maximize).

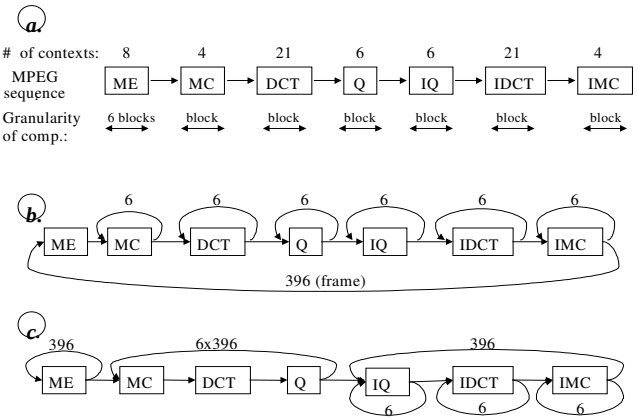

Figure 6: a) MPEG sequence, b) possible schedule of image frame and c) alternative schedule

Proposed Solution: We find the actual schedule to obtain the total execution time for a given application using a two-pronged approach of partitioning the DFG and then scheduling each partition. Our algorithm [12] uses the application data-flow graph, a kernel library (Figure 5) for computation time, input and output data sizes, for each kernel and utilizes system parameters for the MorphoSys architecture such as Frame Buffer size, access times, context load time, and bus width. We use a backtracking technique to support the search process in both tasks. This is guided by a heuristic that tries to explore best candidates first. Bounding heuristics are employed to prune the solution space early on.

Explo-ration #	Cover	LEE	$\sum LB(P_i)$ *cycles*	Executi on time
1	{ME,MC,...,IMC}	Ø	5110	*NNS*
2	{ME} {MC,...,IMC}	{1}	4941	4941
15	{ME} {MC,DCT,Q} {IQ,IDCT,IMC}	{1,2,5}	5080	*NNS*
30	{ME,..,Q}{IQ,...,IMC}	{2,5}	5036	*NNS*
Not explored	{ME}{MC}{DCT}{Q}{IQ}{IDCT}{IMC}	{1,2,3,4,5,6,7}	5806	

Note: Lower Bound = 4894 cc.; NNS= not necessary to schedule.

Table 2: Experimental data for MPEG

Experiments and Results: The MPEG encoder application is used to demonstrate our algorithm. Table 2 lists some of the best results generated during the search. One solution that was not explored (due to search space pruning) is included to illustrate the validity of the bounding check.

The bounding check reduces the search space from 64 different covers to only 31. The best solution is obtained in the second iteration, but the partitioning process continues because the lower bound is not reached. The execution time for second partitioning (obtained after detailed scheduling) is lower than lower bound for all other solutions. Detailed scheduling is not performed for other covers, as no other solution can have lesser execution time.

6. PERFORMANCE ANALYSIS

Several target applications have been mapped to MorphoSys [11]. Performance analysis for some applications is given below.

6.1 Video Compression: Motion Estimation

Motion Estimation is the most computation-intensive task in the MPEG standard [9] for video compression. MorphoSys performance is compared with ASIC architectures implemented in [7], [22] using full search block matching for matching one 8x8 reference block against its search area in Table 3, after accounting for the clock rate and technology. The result shows that MorphoSys can deliver similar performance compared to ASICs and an order of magnitude speedup over Pentium MMX [8].

	MorphoSys	ASIC[7]	ASIC[22]	Pentium Mmx
Cycles	1020	581	1159	29000
Time	10.2 µs	2.9 µs	5.8 µs	145 µs

Table 3: Performance Comparison for Motion Estimation

6.2 Discrete Cosine Transform (DCT)

The forward and inverse DCT are used in MPEG encoders and decoders. Table 4 compares MorphoSys performance for 2-D DCT/IDCT on a 8x8 block with Remarc [14] reconfigurable system (64 processors), V830R/AV [2] superscalar multimedia processor, and optimized Pentium MMX instructions [8].

	MorphoSys	Remarc	V830R/AV	Pentium Mmx
Cycles	37	54	201	240
T (ns)	370	540	1005	1200

Table 4: Performance Comparison for DCT/IDCT

6.3 Automatic Target Recognition (ATR)

Automatic Target Recognition (ATR) is the machine function of detecting, classifying, recognizing, and identifying an object. The ATR processing model [4] developed at Sandia National Labs has been mapped to MorphoSys. The system parameters used in [4] are used for comparison of MorphoSys with the fine-grain FPGA-based ATR systems [4] and [16] in Table 5. MorphoSys outperforms Mojave and Splash 2 by a factor of 2 for performing ATR on an image (one template pair). ATR computations are inherently fine-grain, and map very well to FPGA-based systems of [4] and [16]. Therefore, MorphoSys performance enhancement is significant. Even though the FPGA-based systems have a lower operating frequency, this frequency is relatively fast for FPGAs.

System	MorphoSys	Mojave [4]	Splash 2 [16]
Time	6 ms	24 ms	12 ms

Table 5: ATR Performance Comparison

6.4 Data Encryption (IDE Algorithm)

Data security is a key application domain. The International Data Encryption Algorithm (IDEA) [18] is a typical example of this

application class. IDEA involves processing of plain-text data in 64-bit blocks with a 128-bit encryption/decryption key. Table 6 depicts the performance of MorphoSys, as compared to a software implementation on a Pentium II (after scaling for clock speed of 233 MHz assuming same fabrication technology). MorphoSys even excels the HiPCrypto[17] ASIC designed for the IDEA algorithm.

System	MorphoSys	HiPCrypto	Pentium II
Cycles for 64 bit data	4.5	8	153

Table 6: Performance Comparison for IDE Algorithm

7. CONCLUSIONS

This paper represents a case study of the unique MorphoSys system with its architecture, implementation, programming and scheduling aspects. A software environment has been developed and a novel application kernel scheduling algorithm has been formulated. MorphoSys has been shown to achieve performance levels similar to ASICs for the target applications. We are working to design a suite of CAD tools to provide advanced application mapping features. Future work involves enhancing the functionality and reconfigurability of MorphoSys for widening the range of target applications.

8. ACKNOWLEDGMENTS

This research is supported by DARPA (DoD) under contract F-33615-97-C-1126 and NSF under grant CCR-9877171.

9. REFERENCES

[1] Abnous, A., Christensen, C., Gray, J., Lenell, J., Naylor, A., and Bagherzadeh, N. Design and Implementation of TinyRISC microprocessor in Microprocessors and Microsystems, Vol. 16, 1992

[2] Arai, T., Kuroda, I., Nadehara, K., and Suzuki, K. V830R/AV: Embedded Multimedia Superscalar RISC Processor in IEEE MICRO, Mar/April 1998, pp. 36-47

[3] Chen, D., and Rabaey, J. Reconfigurable Multi-processor IC for Rapid Prototyping of Algorithmic-Specific High-Speed Datapaths, in IEEE Journal of Solid-State Circuits, V. 27, No. 12, Dec 92

[4] Chia, K., Kim, H.J., Lansing, S., Mangione-Smith, W.H., and Villasenor, J. High-Performance Automatic Target Recognition Through Data-Specific VLSI, in IEEE Trans. on VLSI Systems, Vol.6, No.3, 1998

[5] Ebeling, C., Cronquist, D., and Franklin, P. Configurable Computing: The Catalyst for High-Performance Architectures, in Proceedings of IEEE International Conference on Application-specific Systems, Arch. and Processors, July 1997, pp. 364-72

[6] GajjalaPurna, K.M., and Bhatia, D. Temporal partitioning and scheduling for reconfigurable computing, in Proc., IEEE Sym. on FPGAs for Custom-Computing Machines,FCCM'98, pp. .329-330.

[7] Hsieh, C., and Lin, T. VLSI Arch. For Block-Matching Motion Estimation Algorithm, IEEE Transactions on Circuits and Systems for Video Technology, vol.2, pp.169-175, June 1992

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

[9] ISO/IEC JTC1 CD 13818. Generic coding of moving pictures, 1994 (MPEG-2 standard)

[10] Kaul, M., and Vemuri, R. Optimal Temporal Partitioning and Synthesis for Reconfigurable Arch., in Proc. of Design & Test in Europe, pp. 389-396, 1998.

[11] Lee, M.H., Singh, H., Lu, G., Bagherzadeh, N., and Kurdahi, F.J. Design and Implementation of the MorphoSys Reconfigurable Computing Processor, Journal of VLSI and Signal Processing- Systems for Signal, Image and Video Technology, March, 2000

[12] Maestre, R., Kurdahi, F.J., Bagherzadeh, N., Singh, H., Hermida, R., and Fernandez, M. Kernel Scheduling in Reconfigurable Computing, Proceedings of Design and Test In Europe, DATE, 1999

[13] Mirsky, E., and DeHon, A. MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources," Proc. of IEEE FCCM, 1996, pp.157-66

[14] Miyamori, T., and Olukotun, K. A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications," Proc. of IEEE FCCM, 1998

[15] Ouaiss, I., Govindarajan, S., Srinivasan, V., Kaul, M., Vemuri, R. An Integrated Partitioning and Synthesis system for Dynamically Reconfigurable Multi-FPGA Architectures", Reconfig. Arch. Workshop, RAW 1998

[16] M. Rencher and B.L. Hutchings, " Automated Target Recognition on SPLASH 2, IEEE FCCM, 1997

[17] Salomao, S., Alves, V., and Filho, E.C. HiPCrypto: A High Performance VLSI Cryptographic Chip," Proceedings of IEEE ASIC conference, 1998, pp. 7-13

[18] Schneier, B. Applied Cryptography, John Wiley, New York, NY, 1996.

[19] SUIF Compiler system, The Stanford SUIF Compiler Group, http://suif.stanford.edu

[20] Tau, E., Chen, D., Eslick, I., Brown, J., DeHon, A. A First Generation DPGA Implementation, Canadian Workshop of Field-Programmable Devices, FPD'95

[21] Vasilko, M., and Ait-Boudaoud, D. Architectural Synthesis Techniques for Dynamically Reconfigurable Logic, 6th Int'l Workshop on Field-Prog. Logic and Applications, FPL '96 Proceedings, p.290-296

[22] Yang, K-M., Sun, M-T., and Wu, L. A Family of VLSI Designs for Motion Compensation Block Matching Algorithm," IEEE Trans. on Circuits and Systems, Vol. 36, No. 10, October 1989, pp. 1317-25