# RECONFIGURABLE COMPUTING FOR RC6 CRYPTOGRAPHY

May Itani, and Hassan Diab
*Department of Electrical and Computer Engineering*
*American University of Beirut*
*P.O. Box 11-0236, Beirut, Lebanon*
*may01@cyberia.net.lb; diab@aub.edu.lb*

## Abstract

*This paper presents an emerging reconfigurable hardware that potentially delivers flexible high performance for cryptographic algorithms. MorphoSys, a dynamic reconfigurable architecture that sustains implementations that can yield into equally or even better performance results than custom-hardware and yet preserves all the flexibility of general-purpose processors. With today's great demand for secure communications systems, networks and the Internet, there is a growing demand for real-time implementation of cryptographic algorithms. As a case study, this paper presents the mapping and performance analysis for one of the five AES finalists cryptographic algorithms, namely RC6. Being an important and secure cryptographic algorithm, RC6 is considered well chosen to be mapped in order to test and evaluate the suitability of dynamic reconfigurable computer (RC) systems such as MorphoSys.*

## 1. Introduction

Cryptographic algorithms play a crucial role in the information society. With the digital world we are currently living in, these algorithms guarantee that nobody can access an account without authorization, listen to phone calls, or get unauthorized access to sensitive data. It is clear that information technology will become increasingly pervasive: in the short term we expect to see more of e-government, e-voting, e-commerce, etc. with the emergence of ubiquitous computing. These new environments and applications will present new security challenges, and there is no doubt that cryptographic algorithms and protocols will form part of the solution. While cryptography is an essential component, the importance of cryptography should be put in the correct perspective. Indeed, failure of security systems can often be blamed on other reasons than failure of cryptography. Nevertheless, cryptographic algorithms are part of the foundations of the security house, and any house with weak foundations will collapse. One main reason that plays an important role in the performance of cryptographic algorithms is the progress in computational power: Moore's law (1965) predicts that every 18 months transistor density will double. Empirical observations have proved him right (at least for data density) and experts believe that this law will be holding for at least another 15 years. The variation of Moore's law for computational power states that the amount of computation that can be done for the same cost doubles every 18 month [10].

In cryptography, the Advanced Encryption Standard (AES) finalists have an unbreakable security. The five AES finalists were designed from the ground up to be fast, unbreakable and able to support the tiniest computing devices imaginable. The National Institute of Standards and Technology (NIST) worked for three years to develop a new encryption standard to keep the United States government information secure. For this standard, one or more encryption algorithms selected from among the five finalists: MARS, RC6, Serpent, Rijndael, and Twofish.

In this paper, the M1 MorphoSys RC system was used to implement the RC6 block cipher. RC6 is one of the five finalist algorithms chosen for the AES competition held by NIST. It is considered the second secure algorithm after "Serpent" amongst the five finalists. Its large number of diffusions per round gives good security performance against theoretical attacks.

MorphoSys is an RC system-on-chip targeted at data-parallel and computation-intensive applications. The MorphoSys architecture consists of an array of reconfigurable cells combined with a RISC control processor and a high bandwidth memory interface. Simulation results for applications like video compression and automatic target recognition indicate that MorphoSys in particular, and dynamic RC systems in general, can achieve significantly better performance than general-purpose systems.

Performance analysis studies of MorphoSys for several applications have been carried out and the performance of the MorphoSys M1 chip was evaluated for two typical application classes: Applications in multimedia engineering, namely, motion estimation, video encoding,

0-7803-8577-2/04/$20.00 ©2004 IEEE

and Automatic Target Recognition (ATR), and applications in image processing [5]. The applications demonstrate the flexibility of the MorphoSys architecture. Other applications including DSP applications specifically FIR-filters, Discrete Cosine Transform (DCT) and Inverse Discrete Cosine Transform (IDCT), which are parts of the JPEG and MPEG standards, were mapped on Morphosys M1 [2,3]. Moreover, applications in linear algebraic functions namely matrix operations; and the two security algorithms IDEA (International Data Encryption Algorithm) and TWOFISH block cipher algorithm were coded on MorphoSys M1 [1]. The applications highlighted an improved performance of MorphoSys over other parallel architectures.

The Morphosys block diagram is shown in Figure 1. The figure depicts the components and organization of the integrated M1 RC system. The components include an array of *REconfigurable Cells* (REC Array) with configuration memory (*Context Memory*), a control processor (*TinyRISC*), data buffer (*Frame. Buffer*), and a DMA (*Direct Memory Access*) controller [4].

The paper addresses the viability of using reconfigurable computing for cryptographic algorithms to optimize their performance. It starts with an overview of the RC6 cryptographic algorithm. Then the mapping procedure of the algorithm onto Morphosys is presented. Next the performance study results are given with illustrations on the performance of this system upon execution of this specific algorithm; in addition to comparison with other systems. Finally, a conclusion summarizing the performance results with problems faced, and possible future enhancements are stated.

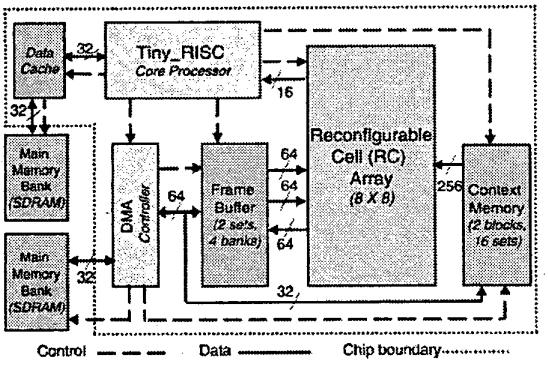


**Figure 1. MorphoSys (M1) block diagram**

## 2. RC6 Algorithm

RC6 is a 128-bit cipher that divides 128-bit strings into four 32-bit blocks and performs various logical/mathematical operations and essential use of multiplications and data-dependent rotations to encrypt. It performs this in twenty rounds with scheduled keys. The structure of RC6 is illustrated in the figure 2.

The first and last two steps are called pre-whitening and post-whitening steps respectively. Without these steps the plain text reveals part of the input to first round of encryption. These steps help to disguise this and thus the encryption is made complex. As defined in chart, the round function is one round of RC6, which is applied twenty times. The structure includes S-box substitutions at the beginning and the end, with the key scheduled rounds taking place in the middle. The rounds involve the f-function (mathematical operation), logical operations (XORs and left shifts), and S-box substitutions consisting of indirect addressing involving setup keywords substitutions. The new features of RC6 over RC5 include the use of four working registers instead of two, and the inclusion of integer multiplication as an additional primitive operation. The use of multiplication greatly increases the diffusion achieved per round, allowing for greater security, fewer rounds, and increased throughput [6,7].

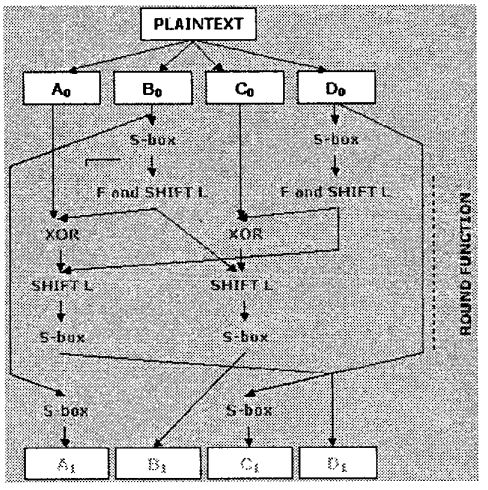


**Figure 2. RC6 flowchart for a single round**

The encryption code is presented below:

```
Input: Plaintext in four 32-bit
variables A,B,C,D
Number of rounds = 20
Expanded key table S[0,...,43]
Output: Ciphertext stored in A,B,C, & D
B=B+S[0]
D=D+S[1]
For i=1 to 20
t=(B*(2B+1)) <<< lg 32
u=(D*(2D+1)) <<< lg 32
A=((A XOR t) <<< u)+ S[2i]
C=((C XOR u) <<< t) + S[2i+1]
```

```
(A,B,C,D)=(B,C,D,A)
A=A+S[42]
C=C+S[43]
lg operation = log base 2 operation
<<< = rotational shift left
```

## 3. Mapping onto MorphoSys

The first step in implementation is to decide what portions of the code can be parallelized and how can they be organized in a way that can be loaded into the REC array and be executed. Since all the variables are 32-bit variables, they should be partitioned into two 16-bit variables (since REC register sizes are 16-bit) when loaded into RECs. The parallelized portions of the code that are implemented on RECs are:

1. Pre-whitening phase done in parallel for B and D where each is a 32 bit block (word).
2. Multiplication, addition, and rotational shift left were done in parallel to compute t and u
3. XORing A and C with t and U
4. S-box computations after Xoring and before shifting left by t and u times
5. Final post-whitening phase or S-box computations done outside the loop

Data was distributed as follows: The plain text words were loaded into the frame buffer Bank A, and the setup key words were loaded into frame buffer bank B. Instructions were stored in the row block in context memory. Since REC array allows two modes of work, the chosen mode was the row mode. That is, the data when loaded to RECs is distributed to different rows in a single column. Context broadcasts used were row wise. Instructions are broadcasted from the row block in context memory. The mapping procedure can be divided into three phases illustrated below.

### 3.1. Phase I

The first phase is the pre-whitening phase which constitutes of two instructions to be executed in parallel. Each instruction involves adding two 32-bit words. To add two 32-bit numbers, each was split into two 16-bit words. Each 16-bit word (starting with LSB) will be added to the corresponding 16 bits from the other word and carry bits from a less-SB will be added correspondingly with a more-SB to get a valid addition result.

### 3.2. Phase II

The next phase is inside the loop. It constitutes of 32-bit by 32-bit multiplication, addition and rotational shifts. Since data to work on cannot be more than 16-bits at a time, all 32 by 32-bit multiplication were breaked into 3 multiplications and 4 additions (taking into consideration carry bits to be added) to get a valid 32 bit multiplication result.

There exist two 32 by 32-bit multiplications, so these will be done in parallel. Since the multiplication procedure was divided individual results were to be shifted then added correspondingly. For that, the MUL, MULADD, and MUL32 instructions available in the REC instruction set were used. MUL32 is a new important instruction that gives a 32-bit multiplication result (of 16*16 bit multiplication) written to two 16-bit registers; the MUL instruction and the MULADD instruction. After multiplication the result is to be shifted to the right 5 times (rotational shift). Each REC content will be shifted 5 times to the right then both RECs will bypass its contents shifted to the left 11 times then add both to get a valid rotational shift result.

### 3.3. Phase III

The third phase involves calculating updated values of A and C. After executing phase 2 instructions, the results t and U are XORed with the old versions of A and C and this will take place in the REC array. However, the rotational shifts of the XORed results will take place in TinyRISC's main memory; i.e. they will be the only two sub-instructions to be executed sequentially. After getting the result, it will be loaded into frame buffer to perform S-box substitutions at the end of the loop and these will take place in parallel in the REC array. This phase is similar to phase 1. All the above steps will be repeated 20 rounds and the last stage after the loop is post-whitening or S-box substitutions for the final result. This phase is again similar to phase 1.

### 3.4. Plane Configurations on MorphoSys

Tables below show the plane configurations for each phase. Before starting with plane configurations used for the mapping, it is important to state that four files were used in the mapping: Context words file, TinyRISC instructions file, key and plain text data files. The context words loaded into the context memory are in the file rc6cntxts.hex. The file corresponds to the row and column block in context memory. Data in files key.hex that contains the setup key and plain.hex that contains the plain text to be ciphered was saved into frame buffer. Finally the main program was loaded in TinyRISC main memory.

As to plane configurations; starting with phase 1, each reconfigurable cell in the first column of RECs receives two 16 bit words from the two banks in the frame buffer. The first 16 bits represents part of the 32-bit plaintext word and the other 16 from the 32-bit word of the setup key. At the beginning, plane 0 will run to add the two bytes coming from the frame buffer and will save the result in register 0 of the REC (Table 1). The first four rows are going to

perform the same instruction; the plane configuration was saved in the row context.

When executing the above planes correspondingly, respective carry bits from addition results of less significant bytes will be added to more significant bytes to get the required 32 bit addition result. Table 2 shows plane configuration for the next phase: The f-function that includes shifting, constant addition and multiplication.

Table 3 shows the third phase computations including Xoring, rotational shift left and S-box computations. Rotational shift left t and U times is done in TinyRISC. As to S-box computations these are similar to table one but on different data.

**Table 1. Plane configurations (RC6 add operation)**

| Plane | Row | A I/p from | B I/pt from | Operation Done | Output Result | Save in |
|---|---|---|---|---|---|---|
| 0 Row Block | 0 | FB | FB | $(B_{31}-B_{16})+(S[0]_{31\text{-}16})$ | $B'_{31}-B'_{16}$ | $R_0$ |
| | 1 | FB | FB | $(B_{15}-B_0)+(S[0]_{15\text{-}0})$ SHR 15 | preliminary carry | none |
| | 2 | FB | FB | $(D_{31}-D_{16})+(S[1]_{31\text{-}16})$ | $D'_{31}-D'_{16}$ | $R_0$ |
| | 3 | FB | FB | $(D_{15}-D_0)+(S[1]_{15\text{-}0})$ SHR 15 | preliminary carry | none |
| 1 Row Block | 0 | Top cell | none | SHR 1 | the carry bit | $R_1$ |
| | 1 | FB | FB | $(B_{15}-B_0)+(S[0]_{15\text{-}0})$ | $B_{15}-B_0$ | $R_0$ |
| | 2 | Bottom Cell | none | SHR 1 | carry bit | $R_1$ |
| | 3 | FB | FB | $(D_{15}-D_0)+(S[1]_{15\text{-}0})$ | $D_{15}-D_0$ | $R_0$ |
| 2 Row Block | 0 | $R_0$ self | $R_1$ self | Add $R_0+R_1$ | $B_{31}-B_{16}$ | $R_0$ |
| | 1 | non | non | keep | $B_{15}-B_0$ | none |
| | 2 | $R_0$ self | $R_1$ self | Add $R_0+R_1$ | $D_{31}-D_{16}$ | $R_0$ |
| | 3 | non | non | keep | $D_{15}-D_0$ | none |

**Table 2. Plane configurations of RC6 multiplication operation in phase II**

| Plane | Row | A I/p from | B I/pt from | Const | Operation Done | Output Result | Save in |
|---|---|---|---|---|---|---|---|
| 3 Row Block | 0 | FB | none | none | SHL 1 | $B_{30}-B_{16}0$ | $R_0$ |
| | 1 | FB | none | none | SHR 15 | $B_{15}$ | none |
| | 2 | FB | none | none | SHL 1 | $D_{30}-D_{16}0$ | $R_0$ |
| | 3 | FB | none | none | SHR 15 | $D_{15}$ | none |
| 4 Row Block | 0 | Top Cell | none | none | Bypass | $B_{15}$ | $R_1$ |
| | 1 | FB | none | none | SHL 1 | $B_{14}-B_00$ | $R_0$ |
| | 2 | Top Cell | none | none | Bypass | $D_{15}$ | $R_1$ |
| | 3 | FB | none | none | SHL 1 | $D_{14}-D_00$ | $R_0$ |
| 5 Row Block | 0 | $R_0$ self | $R_1$ self | none | Add | $2B+1_{31}$ - $2B+1_{16}$ | $R_0$ |
| | 1 | none | none | 1 | Add | $2B+1_{15}$ - $2B+1_0$ | $R_0$ |
| | 2 | $R_0$ self | $R_1$ self | none | Add | $2D+1_{31}-2D+1_{16}$ | $R_0$ |
| | 3 | none | none | 1 | Add | $2D+1_{16}-2D+1_0$ | $R_0$ |
| 6 Row Block | 0 | Top | None | none | Bypass | $2B+1_{15}$ - $2B+1_0$ | $R_1$ |
| | 1 | FB | none | none | MUL32 | To further calculation | $R_2R_6$ |
| | 2 | Bottom | None | none | Bypass | $2D+1_{16}-2D+1_0$ | $R_0$ |
| | 3 | FB | none | none | MUL32 | To further calculation | $R_2R_6$ |
| 7 Row Block | 0 | $R_1$ self | FB | none | MUL | MS 16 bits of B * LS 16 bits of 2B+1 | $R_5$ |
| | 1 | none | none | none | keep | LS 16 bits of B * LS 16 bits of 2B+1 | $R_2R_6$ |
| | 2 | $R_1$ self | FB | none | MUL | MS 16 bits of D * LS 16 bits of2D+1 | $R_5$ |
| | 3 | none | none | none | keep | LS 16 bits of D * LS 16 bits of2D+1 | $R_2R_6$ |
| 8 Row Block | 0 | $R_0$ self | none | none | Bypass | To bottom cell | none |
| | 1 | Top cell | $R_6$ self | none | Add | Preliminary multiplication result | $R_7$ |
| | 2 | $R_0$ self | none | none | Bypass | To bottom cell | none |
| | 3 | Bottom cell | $R_6$ self | none | Add | Preliminary multiplication result | $R_7$ |
| 9 Row Block | 0 | none | none | none | keep | LS 16 bits of B*2B+1 | none |
| | 1 | Top cell | none | none | Bypass | | $R_3$ |
| | 2 | none | none | none | keep | LS 16 bits of D*2D+1 | none |
| | 3 | Bottom cell | none | none | Bypass | | $R_3$ |
| 10 Row Block | 0 | none | none | none | keep | Final mul. result | O/P reg |
| | 1 | R3 | FB | R7 | MULADD | | |
| | 2 | none | none | none | keep | | |
| | 3 | R3 | FB | R7 | MULADD | | |

The final phase is another set of S-box computation instructions. These were given in the first table.

### 3.5. RC6 TinyRISC Instructions

The following table includes a sample of RC6 TinyRISC instructions with illustrations on each command.

**Table 3. Plane configurations (RC6 Phase III insts)**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 11 Row Block | 0 | Top cell | none | none | SHL 5 | $(B*2B+1)_{26}$ – $(B*2B+1)_{16}00000$ | $R_0$ |
| | 1 | none | none | none | keep | $(B*2B+1)_{15}$ – $(B*2B+1)_0$ | none |
| | 2 | Bottom cell | none | none | SHL 5 | $(D*2D+1)_{26}$ – $(D*2D+1)_{16}00000$ | $R_0$ |
| | 3 | none | none | none | keep | $(D*2D+1)_{15}$ – $(D*2D+1)_0$ | none |
| 12 Row Block | 0 | Top Cell | none | none | SHR 11 | 5 bits of $(B*2B+1)_{15-11}$ | $R_1$ |
| | 1 | $R_2$ self | none | none | SHL 5 | $(B*2B+1)_{10}$ – $(B*2B+1)_0 00000$ | $R_0$ |
| | 2 | Bottom Cell | none | none | SHR 11 | 5 bits of $(D*2D+1)_{15-11}$ | $R_1$ |
| | 3 | $R_2$ self | none | none | SHL 5 | $(D*2D+1)_{10}$ – $(D*2D+1)_0 00000$ | $R_0$ |
| | Add R0 + R1 self to get valid rotational shift result and hence t and u | | | | | | |
| 13 Row Block | 0 | $R_0$ self | FB | none | XOR | $(t \oplus A)_{31-16}$ | $R_0$ |
| | 1 | $R_0$ self | FB | none | XOR | $(t \oplus A)_{15-0}$ | $R_0$ |
| | 2 | $R_0$ self | FB | none | XOR | $(u \oplus C)_{31-16}$ | $R_0$ |
| | 3 | $R_0$ self | FB | none | XOR | $(u \oplus C)_{15-0}$ | $R_0$ |

**Table 4. RC6 TinyRISC Sample Code**

| | |
|---|---|
| ldi $10,256 | Load 100 hex into TinyRISC register 10 |
| ldctxt $10,0,1,0,104 | Load context words from main memory starting at address stored in r10 into plane 0 block 0 and starting at word 0 in context memory |
| dbcbar 0,0,0,0 | Double Bank context Broadcast. It sends data from both Banks set 0 address 0 and broadcasts the data according to our chosen mode. In this case row wise. (Different rows receive different data). It executes plane 0 of context memory which starts the addition. |
| dbcbar 0,1,0,0 | Execute plane 1 to continue addition. |
| cbcast 1,0,1,2 | Execute plane 2 which ends addition. |
| wfbi 0,0,0,0,0 | Write the addition result back to FB. Update values. |
| % Loop Round | Start the 20 rounds loop |
| sbcb 1,0,1,3,0,0,0 | Single Bank Broadcast. It sends data from Bank A in frame buffer and executes plane 3 of context memory |
| sbcb 1,0,1,4,0,0,0 | Single Bank broadcast and execution of plane 4 of context memory. |
| cbcast 1,0,1,6 | Execution of plane 6 context memory |
| cbcast 1,0,1,7 | Execution of plane 7 context memory |
| cbcast 1,0,1,2 | Execution of plane 2 context memory |
| sbcb 1,0,1,13,0,0,0 | |
| wfbi 0,1,0,0,100 # 40960 # 0xa000 | Write result from output registers of column 0 cells into frame buffer |
| rcrisc $0,0 | Write result also into TinyRISC registers |

## 4. Results and Performance

One of the characteristics of the RC6 algorithm is the essential use of data-dependent rotations and 32-bit data multiplications which hurts performance on most CPUs. The RC6 algorithm requires platforms supporting fast multiplications for efficient implementation.

In mapping the RC6 on MorphoSys, there was a problem in the fact that the variables are all 32-bit words which obliged us to divide every instruction into several sub instructions. That is, the addition of two 32-bit words is divided into three cycles where each 16 bits are added together and then respective carry bits are accounted for. The 32 bit multiplication also was divided into three 16-bit multiplications and 4 additions. Moreover, there was a problem in the rotational shift by variable (i.e. not constants) amounts t/u. this had to take place in TinyRISC main memory. Xoring does not consume cycles; however loading the values from/into RECs into/from main memory costs much. Every stw/ldw counts 7 cycles.

Performance analysis was done using the number of clock cycles taken by the whole encryption code to execute. Figure 3 shows the times consumed by different operations categorized as: loading instructions/data from main memory into context memory/frame buffer, computations taking place inside RECs, loading from RECs to main memory and computations taking place inside main memory. As seen, 34% of the code constitutes of loading data into main memory (from RECs), and 18% loading data/instructions from main memory into context memory and frame buffer. Main REC computations are only 30% of the whole.

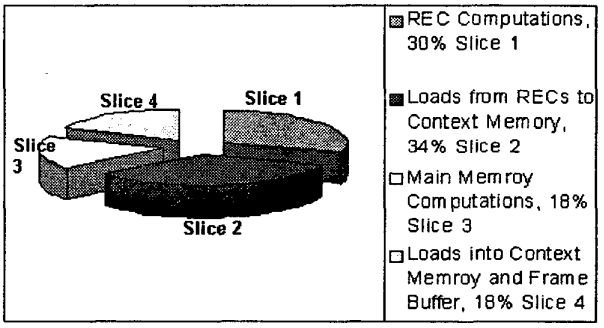


**Figure 3. Time consumed by various operations**

Figure 4 shows performance measurements compared to other PCs [8,9]. First, the number of cycles taken by the program to run was around 2,000 so it was reduced by decreasing the amount of loading to memory and the algorithm was tested for each step on the Mulator till reduced by around 1000 cycles. The way this was done is

by reducing as much as possible sub-operations taking place in main memory and doing them in RECs. Also by reducing the costly communication between RECs and TinyRISC; the number of cycles can be far decreased.

As the figure reveals, RC6 is considered to be fast only on Pentium Pro since it has high-speed 32-bit multiplication and rotational shift instructions.

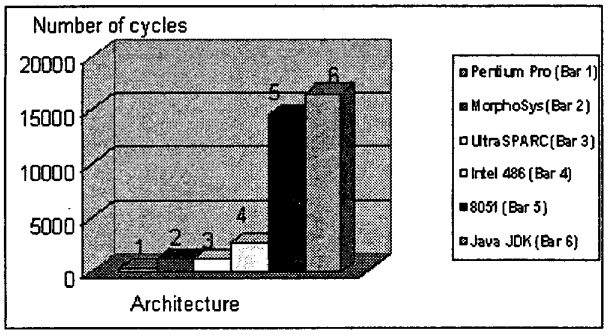


**Figure 4. Performance measures on different architectures**

As seen, MorphoSys reveals good results for the mapping done taking into consideration that the application is 32 bits and is divided into many subtasks and not all instructions were parallelized. The results prove the validity and the promise of such architecture.

Finally, yet importantly, the mapping of the RC6 algorithm is considered one of the important applications mapped onto MorphoSys. In our research, we examined the mapping of the data-dependent rotations, shifting with different number of bits, additions, xors, 32-bit multiplications, and indirect addressing through S box table lookups. These processes mapped onto MorphoSys proved the high flexibility and performance of this hardware. Table 5 summarizes the important results and gives the speed up factors normalized to M1.

**Table 5. Summary of RC6 performance results**

| System | Speedup Factor Normalized to M1 | CPU frequency | Comments |
|---|---|---|---|
| Pentium Pro | 4.346 | 200 MHz | Very fast 32 bit multiplication registers |
| MorphoSys | 1 | 100 MHz | |
| UltraSparc | 0.952 | 160 MHz | Superscalar 64 bit processor |
| Intel 486 | 0.4265 | 33 MHz | |
| 8051 | 0.0788 | | FPGA Flip core |
| Java JDK | 0.069 | 200 MHz | Run time processor / SUN machines |

As to possible enhancements, these can be done on two levels: the application mapping, and the Morphosys M1 chip. For the code mapping, there may be a more efficient way of mapping that may reduce the number of computations in TinyRISC and hence the writing to/from TinyRISC registers from/to the REC array. At the chip level; the REC and TinyRISC instruction sets should have more primitive commands. Or even an advanced general purpose processor could be used in conjunction with the RISC processor. The register size and the number of registers could be increased so that one can work on data without dividing it. Another suggestion may be implementing a small FPGA unit in conjunction with the REC array. This unit may execute bit-level instructions which are available in other important cryptographic algorithms as Twofish and Serpent.

## 5. Conclusion

This paper has presented the mapping of a major cryptographic algorithm on *MorphoSys*, a new reconfigurable architecture. After introducing the architecture of the M1 chip and presenting the RC6 algorithm, an overall explanation of how the mapping was achieved was provided. Testing of the code was performed using the Mulator of the M1 chip which simulates M1 operations. *MorphoSys* performance has been evaluated for this specific application with impressive results that validate the potential of RC paradigms as an efficient architectural platform. Furthermore, some suggestions for enhancing the RC architecture were provided as possible future implementations. As for performance-cost ratio (PCR) analysis, when comparing the hardware platforms that yield a performance close to that of MorphoSys, namely the Pentium Pro and UltraSPARC, the RC paradigm also offers a better PCR value.

Table 5 showed that the algorithm's performance was best on the MorphoSys reconfigurable system after the Pentium Pro system. This result is illustrated by the fact that the algorithms are 32 bit based as is Pentium Pro. Moreover, Pentium Pro has very fast multiplication registers so every instruction is performed as a single instruction and not divided to several sub instructions as in the MorphoSys case. With all the limitations encountered all over the mapping procedure, the MorphoSys mapped code proved to be better than all other implementations and that shows that the trend of reconfigurable computing is paving its way towards being the best.

To conclude, we may give some suggestions concerning the M1 architecture. MorphoSys architecture may not be 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 REC Array and the main processor.

Another potential focus is the REC 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 in particular and RC systems in general, 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.

## 6. References


[1] S. Majzoub, and H. Diab: "Mapping and performance analysis of the Twofish algorithm on Morphosys." *Submitted to the ACS/IEEE International Conference on Computer Systems and Applications*, Tunisia, July 14-18, 2003.

[2] I. Damaj, and H. Diab, "Performance analysis of some geometrical transformation algorithms using reconfigurable computing," *International Symposium on Innovation in Information and Communication Technology (ISIICT 2001),* Philadelphia University, Aman, Jordan, September 26-27, 2001.

[3] I. Damaj, and H. Diab, "Performance Analysis of Extended Vector-Scalar Operations Using Reconfigurable Computing," *ACS International Conference on Computer Systems and Applications (AICCSA 2001),* Beirut, Lebanon, June 26-29 2001.

[4] M. Lee, H. Singh, G. Lu, N. Bagherzadeh, F. Kurdahi, "Design and Implementation of the MorphoSys Reconfigurable Computing Processor", *Journal of VLSI Signal Processing Systems,* March 2000, pp. 147-164. Available www.eng.uci.edu/morphosys/docs/JVSP.pdf.

[5] H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel Computation-Intensive Applications", *IEEE Transactions on Computers.* May 2000, pp. 465-481. www.eng.uci.edu/morphosys/docs/ieee.pdf.

[6] Pritchett, D. Chu, "The Advanced Encryption Standard, An Analysis of Advanced Encryption Algorithms". www.cc.gatech.edu/classes/AY2002/cs4803d_fall/project.html

[7] NIST Report, "Development of Advanced Encryption Standard", AES Conference presentation, 2000. Available @ ece.gmu.edu/courses/ECE543/viewgraphs/lecture9_aes_s3.pdf

[8] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, "AES Performance Comparisons", NIST Report, AES Conference, 2000. Available @ csrc.nist.gov/encryption/aes/round1/conf2/Schneier.pdf

[9] "AES Timings of the Best Known Implementations". Table of best known implementations presented in AES Conference. www.di.ens.fr/~granboul/recherche/AES /timings.html

[10] B. Preneel, "The NESSIE Project: Towards New Cryptographic Algorithms," *The 3rd Int'l Workshop on Information Security Applications*, August 28-30, 2002, Korea. http://www.stork.eu.org/papers/03_nessiev2.pdf