

# RSA encryption using Extended Modular Arithmetic on the Quicksilver COSM Adaptive Computing Machine

Kiran Puttegowda and Peter Athanas

Configurable Computing Laboratory, Virginia Tech, Blacksburg, Virginia 24061  
{kputtego, athanas}@VT.edu

## Abstract

*Modular arithmetic is typically the computational bottleneck in a hardware implementation of public key cryptography algorithms. This paper focuses on an implementation of modular multiplication on the Quicksilver COSM adaptive computing machine as a run-time-reconfigurable user authentication context candidate. The design is targeted specifically to the COSM adaptive computing machine, taking into account the underlying architecture of the device*

## Introduction

There are few applications that better exemplify the utility of run-time reconfiguration than a hand-held cellular phone. The popular method of constructing a cell phone with these capabilities is to implement these modes in software and execute it on a microprocessor. This approach has proven to be acceptable for first-generation phones, but is not viable for future phone standards with orders-of-magnitude higher computational complexity. The Quicksilver COSM adaptive computing prototype chip is a run-time reconfigurable device that has been demonstrated to be a versatile machine for a fairly wide class of applications. Master et al [10] have demonstrated how this simple prototype can be contextually reconfigured from a CDMA pilot searcher, to a channel sounder, to a W-CDMA decoder [11].

In this paper, the versatility of the COSM architecture is tested on an application domain that resides outside of the targeted class of algorithms selected by the COSM architects. Here, the core computation in a cryptographic algorithm, which is central to a user authentication application, is presented.

Public key algorithms are quite robust and secure. The execution of these algorithms is computationally taxing, and a fatal mistake often adopted is to reduce the key sizes in order to reduce the computation time to within the patience tolerance of a user. Instantiating dedicated cryptographic accelerators in a product can be prohibitively expensive and difficult to justify. It

follows that a reconfigurable computing solution may be worth investigation.

## Quicksilver COSM Architecture and Compiler

The Quicksilver COSM device is an early prototype that was designed to demonstrate a few of the conceptual features potentially in the full-blown Quicksilver adaptive computing products.

The COSM prototype chip has a modest number of computational elements that can all execute concurrently. Manually mapping an algorithm to a device with many diverse functional units to maximize the exploitation of algorithmic concurrency could be quite challenging, as was discovered in with Colt/Stallion [5]. The designers of COSM adopted a more robust approach, and developed the device architecture and the compiler in concert. As a result, the complexity of spatial and temporal binding of computational operators is reduced.

COSM consists of processing nodes interconnected with a network. The architecture consists of a Kernel(K)-node, a RISC processor and a Matrix(M)-node, a scalable programmable DSP engine.

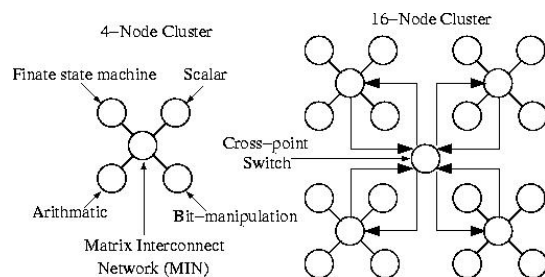


Fig. 1: COSM's Fractal cluster Architecture.

A 4-node cluster consists of an arithmetic, bit-manipulation, FSM and scalar node, which are connected through a cross-point switch called as the Matrix Interconnect Network (MIN). 4 such clusters are connected to form the 16-node cluster. The internal organization of a node is shown in Figure 2.

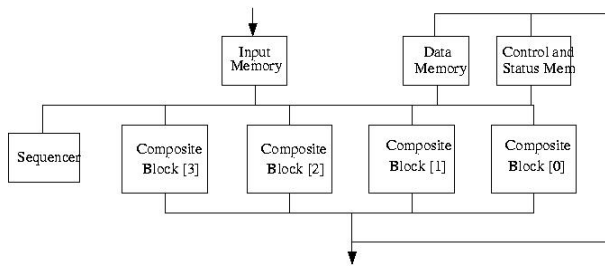


Fig. 2: Internal organization of the M-node.

It consists of the Composite Blocks (CB), which are 16-bit wide processing elements controlled by the sequencer. Data enters through the MIN network from one of the other peer nodes into the input memory. Data are processed in any of the four composite blocks. Intermediate data are stored in the data memory. The CBs are basically 16-bit multiplier-ALU-shifters and have an extensive set of registers that can store intermediate values.

Several layers of abstraction are available to the designer for expressing a design. In this paper, we chose to express the design using the Mini-Matrix Instruction Set. This is essentially a set of macros for pre-defined configurations of the Composite Blocks. The programming paradigm hides the clock cycle, processing resources and register usage from the programmer and the scheduler takes care of such details.

### Montgomery Implementation on COSM

In the RSA scheme, to encrypt a plaintext message  $P$ , the entire message must be represented as a sequence of integers between 0 and  $M-1$  and then raised to the  $E^{th}$  power modulo  $M$  to obtain the coded message  $C$ . To decrypt the encrypted text, it must be raised to the  $D^{th}$  power modulo  $M$ . Therefore

$$C = P^E \text{ mod } M$$

$$P = C^D \text{ mod } M$$

The pair of positive integers  $(E, M)$  is the public encryption key and the positive integer pair  $(D, M)$  form private decryption key. Modular exponentiation is the fundamental operation, which can be achieved by repeated modular multiplications. So the heart of RSA public key algorithms is modular multiplication, which can be succinctly expressed as:

$$P = X * Y \text{ mod } M, \quad (1)$$

Where  $X, Y$ , and  $M$  are typically very large integers, on the order of 1024 bits or more. One of the most

popular methods of calculating this computation is with Montgomery's algorithm, which can be summarized as:

Inputs:  $X, Y, M$  with  $0 < X, Y < M$   
 Output:  $P = (X * Y * (2^{(n-1)})) \text{ mod } M$   
 $n$ : number of bits in  $X$ ,  
 $x_i$ :  $i$ th bit of  $X$   
 $p_0$ : LSB of  $P$

1.  $P := 0;$
2. for  $i:=0$  to  $n-1$  loop
3.      $P := P + x_i * Y;$
4.      $P := P + p_0 * M;$
5.      $P := P \text{ div } 2;$
6. end loop;
7. if  $P \geq M$  then  $P := P - M;$

Each statement within the loop body is in itself a loop since the numerical precision of  $P, X$ , and  $Y$  far exceed the arithmetic bus width of the underlying hardware (16 bits for the integer datapaths within COSM in this case). These variables are stored as arrays in local data memories distributed within COSM for fast access, and are streamed into appropriate pathways, controlled by the on-chip sequencers. One optimization realized here is to reduce the number of operations within the inner loop. Statements 3 and 4 are essentially conditional additions. For Statement 3, if the  $X_i$  bit is a zero, no addition is necessary. Similarly, if  $P$  is an even number, no addition is necessary. These conditions can be tested in advance – depending upon the values of these two bits, a loop with the appropriate additions can be selected; thus, eliminating the need to check these bits repetitively within the loop.

One architectural dependent optimization is to exploit the distributed memory banks within COSM. The data arrays for  $X, Y, M$ , and  $P$  can be stored within separate local memories, and streamed to the computational units independently. Furthermore, the Data Address Generators (DAGs) associated with each memory can handle the indexing computation independently; thus, eliminating several increments and additions within the loop body.

Another optimization can be found in the last step, Statement 7. This step is essentially a final normalization that insures that the result is less than the modulus. This final pass-through of the accumulated result can be eliminated by speculatively computing both the normalized and non-normalized result. These optimizations are effective in scaling the complexity for a COSM implementation. While the COSM has dedicated pathways for the addition and

multiplication of integers up to 40 bits wide, 16-bit arithmetic was chosen for the implementation presented here since the major pathways through the device are 16 bits wide.

Once this kernel is resident within the COSM array, a higher-level shell of, say, RSA can be implemented on a Scalar node (RISC processor). The authentication application would then benefit from the accelerated modular arithmetic core.

Similar algorithms implemented in FPGA occupy most of the chip area available in the FPGA [12]. The proposed approach used a single composite block in a single cluster of the chip, a small part of the logic available. This is cost effective, as this logic is not utilized after the initial authentication process. The use of dedicated hardware (DAG) to stream data into the composite blocks reduces the cost of data access.

### Results and Conclusion

In this paper, we presented the implementation of a core cryptographic function commonly used in public key protocols, such as RSA. This core was designed to exploit the properties within the Quicksilver COSM device. The design enhances Montgomery's algorithm by reducing the number of operations required within the inner loop.

### References

- [1] Montgomery, P. L., *Modular Multiplication without trial division*, Mathematics of Computation, 44, pp. 519-521, 1985.
- [2] Blum, T., Paar, C., *Montgomery Modular Multiplication on Reconfigurable Hardware*, 14th IEEE Symposium on Computer Arithmetic (ARITH-14), April 14-16, IEEE 1999.
- [3] Bunimov, V., Schimmler, M., Tolg, B., *A Complexity-Effective Version of Montgomery's*

*Algorithm*, in the Workshop on Complexity-Effective Design, Anchorage, Alaska, May 25, 2002.

- [4] Brickell, E. F., *A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography*, Proc. of Crypto'82, pp. 51-60, Plenum Press 1983.

- [5] Kahne, B. *A Genetic Algorithm-Based Place-and-Route Compiler For A Run-time Reconfigurable Computing System*, Masters Thesis, Virginia Tech, Department of Electrical and Computer Engineering, May 1997.

- [6] Eldridge, S. E., Walter, C. D., *Hardware implementation of Montgomery's modular multiplication algorithm*, IEEE Transaction on Computers, Vol. 42, pp. 693-699, July 1993.

- [7] Koc, C.K., Acar, T., Kaliski, B.S., *Analyzing and Comparing Montgomery Multiplication Algorithms*, IEEE Micro, pp. 26-33, June 1996.

- [8] Blakley, G. R., *A Computer Algorithm for Calculating the Product A\*B modulo M*, IEEE on Computers, Vol. c-32, No. 5, May 1983, pp. 497-500.

- [9] Shand, M., Vuillemin, J., *Fast implementation of RSA cryptography*, Proc. 11th IEEE Symposium on Computer Arithmetic, pp. 252-259, IEEE 1993.

- [10] Master, P., *The Age of Adaptive Computing Is Here*, in Field-Programmable Logic and Applications, 12th International Conference, FPL 2002, Montpellier, France, September 2-4, 2002.

- [11] Master, P., *Keynote: The Next Big Leap in Reconfigurable Systems*, at the International Conference on Field-Programmable Technology, December 2002.

- [12] A Dally, W Marnane, *Efficient Architectures for implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic*, at the FPGA'02, Monterey CA, February 2002.