Reconfigurable Implementation for the AES Algorithm

Raoel Ashruf, Georgi Gaydadjiev, Stamatis Vassiliadis
Computer Engineering Laboratory,
Electrical Engineering Department,
Delft University of Technology,
Delft, The Netherlands
{rmas5, Georgi}@Dutepp0.ET.TUDelft.NL

Abstract— **The choice of a platform, software, ASIC or FPGA, is driven by several aspects, such as algorithm performance, cost and flexibility. Although ASIC has the highest performance and the lowest unit cost, it has no flexibility at all. While software has the most flexibility of all, the performance is very low. The MOLEN architecture, developed at the Techinical University of Delft, open up new perspective for this problem. The architecture is based on a cooperation between a general purpose core processor and reconfigurable hardware, for example a FPGA. Since cryptographic algorithms are relative frequently upgraded, this high flexibility is desperately needed. There are several reasons for upgrading such as when the algorithm is broken or there is a better algorithm or even in case where an algorithm independent protocol is needed. To put it briefly the $\rho\mu$-coded processor put a new perspective on great performance and high flexibility. In this paper we investigate several Rijndael (AES) implementations based on the Molen $\rho\mu$-coded processor. Two types of FPGAs, Xilinx and Altera, are evaluated in order to produce realistic results. In addition, a modified simulator based on the SimpleScalar Toolset (v3.0)is used to estimate the performance potential. The profiling of the AES code is done based on different methodologies. In this paper an analogy is drawn between. The VHDL descriptions produced, are enhanced with standard interface making them reusable for other research projects based on the same architecture and dealing with similar data processing problems. The AES running on the top of Molen performs equaly good as other FPGA based solutions. The main advantage of our solution is that reconfiguration from one keysize to another is done faster than on pure FPGA implementations. Our solution is applicable in a wide range of devices - staring from mainframes and going down to very limited architectures, e.g. smart card microcontrollers.**

Keywords— **Reconfigurable Processors, VHDL, FPGA, AES**

I. Introduction

Since privacy issues and network security are emerging due to the wide internet penetration, the research in cryptography and application of cryptographical algorithms is increasing. Not only the algorithm reliability, but also the speed performance and implementation flexibility are considered as major factors for improvement. In the early days, ASICs (Application Specific Integrated Circuits) where used to implement such systems. Considering the variety of algorithms and the fact that many new standards are developed continuously, the next logical step was to use reconfigurable hardware for implementing such algorithms. Substantial amount of work has been reported on cryptography implementations based on FPGAs (Field Programmable Gate Arrays). Such systems, however, are idle for the time a new hardware configuration is being loaded. This fact is unacceptable for some of the systems, e.g. transaction verification systems. Architectures consisting of reconfigurable hardware and one or more general purpose processors are considered as a good candidates for speeding up the performance and are capable of reconfiguration without execution interrupting. The MOLEN $\rho\mu$-coded processor [3], designed at the Delft University of Technology is one example for such an architecture. This paper concentrates on the speed performance of the AES encryption algorithm mapped on the $\rho\mu$-coded processor. The performance results are compared to the ANSI C software implementation of AES as presented in [1]. Since the realization of the used architecture was still not finalized at the time this paper was written, the whole architecture is simulated in a modified Simple Scalar cycle accurate simulator (v3.0) [2].

The organization of this paper is as follows. Section II describes the AES algorithm and analyzes different algorithm parts eligible for FPGA implementation. Section III introduces the MOLEN $\rho\mu$-coded architecture used for the implementation. Section IV shows the performance numbers of our AES implementation based on the MOLEN processor. Section V concludes the discussion and presents the future developments to be completed.

II. AES algorithm analysis and implementation

The Advanced Encryption Standard (AES), also known as the Rijndael algorithm, is a block cipher that can encrypt data blocks of 128, 192 or 256 bits using symmetric keys of 128, 192 or 256 bits. The main idea behind introducing AES was to replace the DES (and triple DES) algorithm used for more than 20 years now. Due to its performance, security, efficiency, ease of implementation and flexibility Rijndael was chosen for AES standard in the year 2001. According to the National Institutes of Standards and Technology a computer that could crack 56-bit DES key in just one second would need 149 trillion years to do the same with a 128-bit AES key. Our initial investigation will only focus on the 128 bit implementation, however, the same techniques are applicable also for bigger key sizes.

A. The algorithm

Rijndael is designed to use only simple byte (8-bits) operations. This makes Rijndael implementations possible even on very simple microcontrollers. The encryption of a data block is composed of an initial XOR step with a temporally key, several round transformations (depending on the key or block size) and an additional round performed at the end with one round step omitted. In case of 128 bits block size, there are 9 rounds, each involving the following four transformations (also known as *round transformations*):

$$ByteSub$$

$$ShiftRow$$

$$Mixcolumn$$

$$AddRoundKey$$

For the final step, the Mixcolumn transformation is not performed. For data decryption, however, different transformations are used for the first three round transformations, respectively *InvByteSub*, *InvShiftRow* and *InvMixColumn*. In addition, the round transformations are applied in an inverse sequence inside one round. This all makes the encryption and the decryption procedures different, enforcing separate performance investigation for both directions.

B. Performance analysis

First we analyze each transformations based on how many execution cycles it would take on the proposed architecture. This done to estimate which parts of the algorithm should be implemented in hardware. As described earlier, the AES algorithm consists of many different transformations. The first step in ciphering mode is the initialization step. In this step the input data is xor-ed with the 128-bit key. Since the proposed architecture will contain 4 ALUs, it would just take 1 execution cycle to perform this operation considering the data is in place. This all makes this step not interesting for hardware implementation. After the initialization step, the first round will be initiated. The first round transformation of this step will be the byte substitution (*ByteSub*) where 16 transformed data bytes are substituted. This is most probably fetching data out of the memory or inside the register file, which will also take a limited amount of clock cycles. The next round transformation inside the round is *ShiftRows*. This is a simple reordering of the bytes, which isn't really explicitly needed and can be performed together with the next transformation, that is *MixColumn*. *Mixcolumn* is based on matrix multiplication in GF(2^8), which execution will definitely cost significant amount of cycles. Profiling of the algorithm shows that this stage will cost 4948605 processor cycles while encrypting a 4k input data. The last round transformation is again simple arithmetic operation not interesting for hardware implementation. The following encryption process is based on repetition of the transformations described above. For the deciphering mode only the Inverse Mixcolumn (*InvMixcolumn*) transformation, will take significant amount of clock cycles making it a good candidate for implementation in hardware. To summarize it, only the Mixcolumn and the Inverse Mixcolumn transformation are illegible candidates for implementing it in hardware.

C. The hardware implementation of Mixcolum and Inverse Mixcolumn

The MixColumn transformation is a based on matrix multiplication in $GF2^8$. Every 16 input bytes are multiplied by the following matrix:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \quad (1)$$

The multiplications are performed in $GF(2^8)$ domain. Multiplication by a constant in $GF(2^8)$ will result in xoring the bits of the input byte in a particular way. For example multiplication by the constant ”03” in $GF(2^8)$ is depicted in figure 1.

For the inverse Mixcolumn transformation all columns are multiplied by a different matrix, which is depicted below. Please note, that the matrix elements are denoted as hexadecimal values.

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \quad (2)$$

This matrix is the inverse matrix of the one used in Mixcolumn. The primary difference, compared to Mixcolumn,

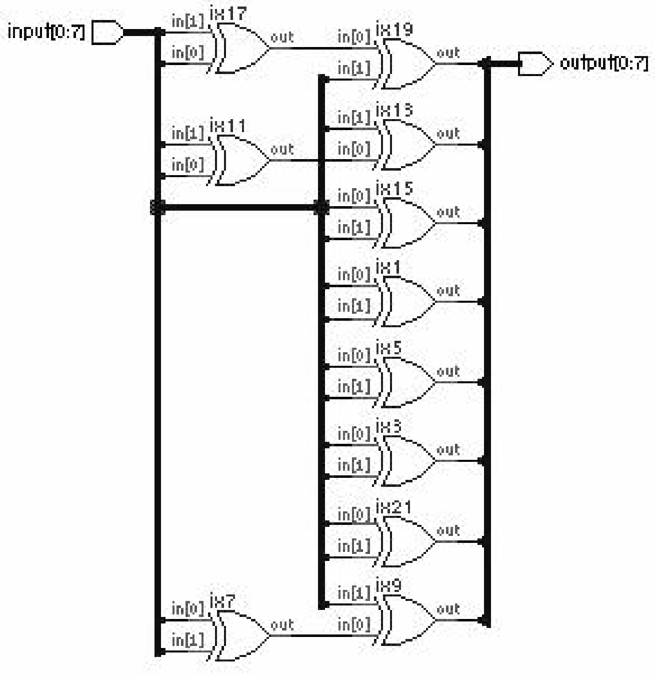

Fig. 1. Multiplication by constant 3 in $GF(2^8)$

are the bigger hexadecimal values of the matrix coefficients. Multiplication by these constant elements of the Galois Field leads to more complex dependency. Since more gates are used the Inverse Mixcolumn transformation has a longer critical path compared to the Mixcolumn transformation. This is the main property that the decryption process is more time consuming than the encryption process.

III. The MOLEN $\rho\mu$-coded processor

The MOLEN [3] architecture is based on a cooperation between a general purpose core processor and reconfigurable hardware capable of partial reconfiguration. A new hardware/software co-design methodology forms the basis of the MOLEN approach. In this methodology, $\rho\mu$-code is utilized that controls the reconfiguration and execution processes of the reconfigurable hardware (unit). The MOLEN approach can support an infinite number of implementations on the FPGA structure as long as they fit on the available FPGA. Consequently, a storage unit (called $\rho\mu$-code unit) is present on-chip that permanently stores frequently used $\rho\mu$codes and temporarily stores less frequently used $\rho\mu$-code in order to diminish their loading times. This all provides a suitable platform for exploring the hardware/software paradigm to gain the best algorithm performance.

A. *The Architecture*

The machine organization is shown in figure 2. The Instruction Buffer is an additional instruction cache, which stores the instruction that are fetched from the memory. The arbiter partially decodes these instructions in order to determine if it should be issued by the core processor (CP) or the reconfigurable hardware unit. The needed data are fetched from the general-purpose registers (GPRs) or the data cache. The results are written back to the same GPRs or the data cache, while the control register (CR) stores other status information. The reconfigurable unit consists of a custom configured unit (CCU) and a $\rho\mu$-code unit. This allows hardware reconfiguration by firmware via an extension of the classical microcode. There are two new instructions, which allow partial and complete reconfiguration of the CCU. These are p-set and c-set instructions. The p-set instructions configures the CCU only partially, while the c-set instructions does it totally. For actually executing microcode on the configured CCU, there is an extra instruction called $execute$. These new instructions point to the memory location where the reconfiguration or execution microcode is stored. The sequencer is responsible for the microinstruction execution sequence for the CCU, while the p-CONTROL STORE unit only store the microcodes. The sequencer uses a residence table to determine whether the microcode is already cached in the p-CONTROL STORE unit. The table consists the most frequently used translations and keeps track of these translations.

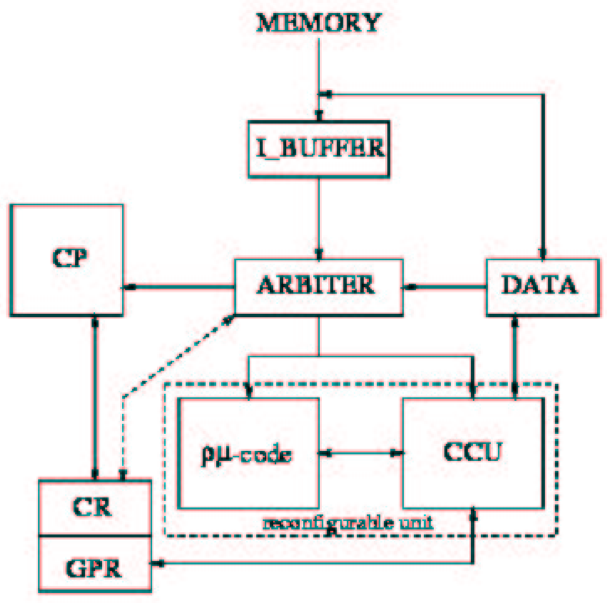

Fig. 2. Block Diagram of the MOLEN architecture

TABLE I
THE SPEED OF THE AES IN SOFTWARE AND ON THE $\rho\mu$-CODED PROCESSOR. (BARRING UNFORESEEN CIRCUMSTANCES)

	Encryption	Decryption
(ANSI C) Software based	68.75 kb/s	82.75 kb/s
$\rho\mu$-coded processor with Xilinx FPGA	33.75 kb/s	34.25 kb/s
$\rho\mu$-coded processor with Altera FPGA	33.50 kb/s	34.25 kb/s
Gain with the Altera FPGA	51%	59 %
Gain with the Xilinx FPGA	51%	59%

B. *The advantage of using this architecture*

The choice of a platform (software, ASICs or FPGAs) is driven by several aspects such as algorithm performance, costs and flexibility. It is known that ASICs have the highest speed performance while its flexibility is very limited. For software it is quite the opposite. Since cryptographic algorithms are relative frequently upgraded, not only the high flexibility is needed, but also the speed performance is desperately needed. These needs can be fulfilled by a FPGA. The only disadvantage of FPGAs is the unit-cost, which is dependent of the FPGA size. Since the MOLEN architecture is based on a combination of software and FPGA, we get high flexibility, good performance and reduced cost.

IV. THE PERFORMANCE RESULTS

Both the Mixcolumn and the Inverse Mixcolumn transformations are simulated in Modelsim (v5.5) and synthesized with Leonardo spectrum (v2001-1a32) CAD tools. The VHDL sources were synthesized for both Altera EPEX20k 400FC672 and Xilinx Spartan2 2s15cs144 FPGAs devices. The results are presented in table II.

TABLE II
THE CRITICAL PATH DELAY & USED CLBS OF MIXCOLUMN AND INVERSE MIXCOLUMN

	Altera	Xilinx
Mixcolumn	15.24 ns	12.73ns
Inverse Mixcolumn	21.78 ns	14,82ns
Used CLBs for Mixcolumn	1.35%	6.94%
Used CLBs for InvMixcolumn	3.41%	18.23%

Due to the bigger hexadecimal values of the matrix coefficients an increased amount of CLBs is used for the Inverse Mixcolumn implementation compared to Mixcolumn. The syntehsis results vallidate the assumption, that Inverse Mixcolumn is more compute intensive. Since the critical path delays are known, we were able to calculate the maximum frequencies for both Mixcolumn and Inverse Mixcolumn. For example, the maximum clock frequency for Mixcolumn on the Altera FPGA was determined on65 MHz.

Since the realization of the Molen $\rho\mu$-coded processor was still under development, the whole architecture was simulated in a modified version of sim-outorder simulator from the Simple Scalar (V3.0) [3] toolset. We assumed that the general purpose processor of the proposed machine organization is clocked on 1 GHz. The simulator configuration was as follow: 4 integer ALUs, 1 integer MULT/Div-unit, 4 FP adders, 1 FP MULT/Div-unit, 2 memory ports (R/W), 1 mixcolumn-unit and 1 inverseMixcolumn-unit. A 4kbyte text file with a 128-bit key in ECB mode [1] was used as a benchmark. With the help of microcode annotation and the synthesis results the speed performance results were obtained. The normalized results of the simulations are presented in table I.

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented the speed performace of the AES on the Molen $\rho\mu$-coded processor. The $\rho\mu$-coded processor allows pieces of the algorithm to execute directly on hardware. The whole architecture was simulated in a modifed sim-outorder simulator of the Simple Scalar (V3.0) [3]. The obtained speed gain on this architecture was 51% for encryption mode and 59% for decryption mode.

When the Molen realization is finalized, the results of this paper are to be vallidated on real hardware. In addition to this, future work will concentrate on speed performance for other ciphering algorithms on the Molen $\rho\mu$-coded processor. Anoter important aspect for future research is the vulnerability of algorithms on the Molen $\rho\mu$-coded processor.

REFERENCES

[1] J. Daemen and V. Rijmen. The design of rijndael, 2002.
[2] M. Austin E. Larson and D. Erns. The simplescalar tool set, version 2.0, doug burger. *http://www.simplescalar.com*
[3] S. Vassiliadis S.Cotofana and S. Wong. The molen $\rho\mu$-coded processor. In *The 11th International Conference on Field Programmable Logic and Application (FPL)*, Augustus 2001.