Performance Evaluation of AES Candidates using FPGA Implementation

Ashish
Department of Computer Science,
Oregon State University, Corvallis, Oregon 97331 -USA.
E-mail: ashish@cs.orst.edu

Abstract— **The Data Encryption Standard (DES) expired in 1998, now the Advanced Encryption Standard is underway. The reprogrammable devices such as Field Programmable Gate Arrays (FPGAs) are are attractive options for encryption algorithms using hardware because they provide agility for cryptographic algorithms, physical security and better performance than software implementations. Among the various time-space implementation tradeoffs we focus primarily on time performance. This paper investigates the significance of FPGA implementation of serpent algorithm, one of the AES candidate algorithm. One of the main finding of the paper is that Serpent algorithm can be implemented with encryption rates beyond 4 Gbits/ sec on current FPGAs.**

I. Introduction

After the expiration of DES standards in 1998, the National Institute of Standards and Technology (NIST) has initiated a process to develop a Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard specifying an Advanced Encryption Algorithm (AES). The AES will specify a non-classifed, publicly disclosed encryption algorithm that will be as widely accepted as DES in the private and public sectors. NIST has developed candidate algorithms for inclusion in AES, resulting in fifteen official candidate algorithms. AES candidates can be efficiently implemented in both hardware and software.

The advantages of a software implementation include ease of use, ease of upgrade, portability, and flexibility. A software implementation offers only limited physical security, especially with respect to key storage. Conversely, cryptographic algorithms and their associated keys that are implemented in hardware are, more physically secure as they cannot easily be modified by an outside attacker

Reconfigurable devices such as FPGAs are a attractive option for a hardware implementation as they provide the flexibility of dynamic system evolution as well as the ability to easily implement a wide range of functions/algorithms. It appears to be especially relevant to focus on high-throughput implementations for FPGA-based encryption. Private-key cryptographic algorithms seem to fit extremely well with the characteristics of the FPGAs. The fine-granularity of FPGAs matches extremely well the operations required by private-key cryptographic algorithms such as bit- permutations, bit-substitutions, look-up table reads, and boolean functions. On the other hand, the constant bit-width required alleviates accuracy-related implementation problems and facilitates efficient designs. Moreover, the inherent parallelism of the algorithms can be efficiently exploited in FPGAs. Multiple operations can be executed concurrently resulting in higher through- put compared with software-based implementations. Moreover, the key-setup circuit can run concurrently with the cryptographic core circuit resulting in low latency time and agile

Author is a graduate student at the Department of Computer Science, Oregon State University, Corvallis, Oregon 97331. E-mail: ashish@ece.orst.edu

II. FPGA Overview

Processors are general purpose and can virtually execute any operation. However, their performance is limited by the restricted interconnect, data path, and instruction set provided by the architecture. Conversely, ASICs are application specific and can achieve superior performance compared with processors. However,the functionality of an ASIC design is restricted by the designed parameters provided during fabrication. Any update to an ASIC-based platform incurs high cost. As a result, ASIC-based approaches lack flexibility.FPGA technology is a growing area of research that has the potential to provide the performance benefits of ASICs and the flexibility of processors. Application specific hardware circuits can be created on demand to meet the computing and interconnect requirements of an application. Moreover, these hardware circuits can be dynamically modified partially or completely in time and in space based on the requirements of the operations under execution. As a result, superior performance can be expected compared with the performance of the equivalent software implementation executed on a processor.

The revolution of the configurable system technology led to the development of configurable devices and architectures with great computational power. As a result, new application domains become suitable for FPGAs beyond the initial applications of rapid prototyping and circuit emulation. FPGA-based solutions have shown significant speedups (compared with software and DSP based approaches) for several application domains such as signal & image processing, graph algorithms, genetic algorithms, and cryptography among others. The basic feature underlying FPGAs is the programmable logic element which is realized by either using anti-fuse technology or SRAM-controlled transistors. FPGAs have a matrix of logic cells overlaid with a network of wires. Both the computation performed by the cells and the connections between the wires can be configured. Current devices mainly use SRAM to control the configurations of the cells and the wires. Loading a stream of bits onto the SRAM on the device can modify the configurations. Furthermore, current FPGAs can be reconfigured very quickly, allowing their functionality to be altered at runtime according to the requirements of the computation.

key-context switching.

A. FPGA-based Cryptography

FPGA devices are a highly promising alternative for implementing private-key cryptographic algorithms. Compared with software-based implementations, FPGA implementations can achieve superior performance. The fine-granularity of FPGAs matches extremely well the operations required by private-key cryptographic algorithms (e.g. bit-permutations, bit-substitutions, look-up table reads, Boolean functions).As a result, such operations can be executed more efficiently in FPGAs than in a general-purpose computer. Furthermore, the inherent parallelism of the algorithms can be efficiently exploited in FPGAs as opposed to the serial fashion of computing in an uni-processor environment. At the cryptographic-round level, multiple operations can be executed concurrently. On the other hand, at the block-cipher level, certain operation modes allow concurrent processing of multiple blocks of data. For example, in the ECB mode of operation, multiple blocks of data can be processed concurrently since each data block is encrypted independently. Consequently, if p rounds are implemented, a throughput speed-up of p can be achieved compared with FPGAs, their flexibility is restricted. Thus, the replacement of such application-specific chips becomes very costly [11] while FPGA-based implementations can be adapted to new algorithms and standards. However, if ultimate performance is essential, ASICs solutions are superior.

III. Implementation and Design Decisions

Among the various time-space tradeoffs, we focused primarily on time performance. Our goal was to maximize throughput for the cryptographic core of each candidate algorithm. We have exploited the inherent parallelism of each cryptographic core and the low-level hardware features of FPGAs to enhance the performance. Moreover, the latency issue was of primary interest, that is, the cryptographic core has to commence as early as possible. Based on the achieved throughput, we designed the key-setup component to sustain the data rate of the cryptographic core and to achieve minimal latency. Even if an algorithm does not support on-the-y key generation (in the software domain), the key setup can be executed concurrently with the cryptographic core.

For each algorithm we implemented the encryption block cipher for 128-bit data blocks using 128-bit keys. A single-round" based design was chosen for each implementation. Similar performance analysis can be performed for larger sizes of data blocks and keys as well as for implementations that process multiple blocks of data concurrently.

Lat	Lat/encry	Through	Through(Kbps/slice)
1.96	3.12	203.77	29.55

TABLE I
MARS Time Performance.

For each algorithm, we have also implemented the key-setup circuit and the cryptographic core separately. For all the implementations, the maximum clock speed of the key-setup circuit was higher than the maximum clock speed of the cryptographic core. Based on the results of these individual implementations, we also provide latency estimates in case two different clocks are used. Regarding the cryptographic cores, the majority of the required operations fit extremely well in Virtex FPGAs. The permutations and substitutions can be hard-wired while distributed memory can be used as look-up tables. In addition, Boolean functions, data- dependent rotations, and addition can be mapped very efficiently onto Virtex FPGA. Wherever a multiplication with a constant was required, constant coefficient multipliers were utilized to enhance the performance compared with "regular" multipliers. Regular multiplication is required only by the MARS and RC6 block ciphers. In both cases, two 32-bit numbers are multiplied and the lower 32-bit of the output are used in the encryption process. We tried the multiplier- macros provided for Virtex FPGAs but we found that they were a performance bottleneck. Besides the excessive latency that was introduced due to the numerous pipeline stages, excessive area was also required since the full multiplier was mapped onto the FPGA. Instead of using these macros, a multiplier that computes partial results in parallel and outputs only the required 32-bits was used. As a result, the latency was reduced by more than 50reduced significantly.

IV. Implementation Results

In the following, implementation results as well as relevant performance issues specific to each algorithm are provided. The latency results are represented both as absolute time and as the fraction of the corresponding encryption time of one 128-bit block of data. In addition, the throughput results are represented both as encryption rate and as encryption rate elaborated on area. Finally, area requirements results are pro- vided for both the key-setup and the cryptographic core circuits.

A. MARS

The MARS block cipher is the IBM submission to AES [6]. The time performance and area requirements results for our MARS implementation are shown in Tables 1 and 2.

A.1 Key Schedule

The MARS key expansion procedure expands the input 128-bit key into a 1280-bit key. First a linear-key expansion occurs following by stirring the key-words based on

Area Req	# slices	# slices/area
Total	6896	1.00
Key Scheduling	2275	0.33
Crypto Core	4621	0.67

TABLE II
MARS Area Requirement.

Lat	Lat/encry	Through	Through(Kbps/slice)
0.17	0.15	112.87	42.59

TABLE III
RC6 Time Performance.

Area Req	# slices	# slices/area
Total	2650	1.00
Key Scheduling	901	0.34
Crypto Core	1749	0.66

TABLE IV
RC6 Area Requirement.

Lat	Lat/encry	Through	Through(Kbps/slice)
0.07	0.15	112.87	42.59

TABLE V
Rijndael Time Performance.

an S-box. Both processes involve simple operations performed repeatedly. However, the final stage of modifying the multiplication key-words involves string-matching operations that are relatively expensive functions. String matching is an expensive operation compared with the rest of the operations required by MARS. A compact implementation of string-matching introduces high latency while a high-performance implementation increases the area requirements dramatically. In our implementation, the last stage of the key-expansion process (i.e. string-matching) was not implemented.

A.2 Cryptographic Core

The cryptographic core of MARS consists of a 16-round cryptographic layer wrapped with two layers of 8-round “forward and backward mixing” [6]. In our implementation only one round of each layer was implemented that was used repeatedly. In our implementation, while the encryption time for the first block of data is 32 clock cycles, the encryption time for every following block of data is 16 clock cycles. We have achieved this improvement by increasing the utilization factor of the processing stages (i.e. all the three processing stages execute in parallel). As a result, high throughput was achieved.

B. RC6

The RC6 block cipher is the AES proposal of the RSA Laboratories and R. L. Rivest from the MIT Laboratory for Computer Science [12]. The implemented block cipher corresponds to w = 32-bit round keys, r = 20 rounds, and b = 14-byte input key. The time performance and area requirements results for our RC6 implementation are shown in Tables 3 and 4.

B.1 Key Schedule

The RC6 key scheduling expands the input 128-bit key into 42 round keys. The key for each round corresponds to a 32-bit word. The key scheduling is fairly simple. The round-keys are initialized based on two constants. We have implemented the initialization procedure using a look-up table since it is the same for any input key. Then, the contents of the look-up table are used to generate the round-keys with respect to the input key. As a result, remarkably low latency can be achieved that is equal to the 15of the time for encrypting a block of data.

B.2 Cryptographic Core

The cryptographic core of RC6 consists of 20 rounds. The symmetry and regularity found in the RC6 block cipher resulted in a compact implementation. The entire data-block is processed at the same time by using two identical circuits. The achieved throughput depended mainly on the efficiency of the multiplier.

C. Rijndael

The Rijndael block cipher is the AES proposal of J. Daemen and V. Rijmen from the Katholieke Universiteit Leuven [7]. The implemented block cipher corresponds to Nb = 4, Nk = 4, and Nr = 10. The time performance and the area requirements results of our implementation are shown in Tables 5 and 6.

C.1 Key Schedule

The Rijndael key scheduling expands the input 128-bit key into a 1408-bit key. Simple operations are used that result in extremely low latency. ROM-based look-up tables are utilized to perform the Sub Byte transformation. The achieved latency is the lowest among all the implementations considered in this paper.

Area Req	# slices	# slices/area
Total	5673	1.00
Key Scheduling	1361	0.24
Crypto Core	4312	0.76

TABLE VI
Rijndael Area Requirement.

Lat	Lat/encry	Through	Through(Kbps/slice)
0.08	0.09	148.95	66.20

TABLE VII
Serpent Time Performance.

Area Req	# slices	# slices/area
Total	2550	1.00
Key Scheduling	1300	0.51
Crypto Core	1250	0.49

TABLE VIII
Serpent Area Requirement.

Lat	Lat/encry	Through	Through(Kbps/slice)
0.18	0.25	173.06	18.48

TABLE IX
Two Fish Time Performance.

Area Req	# slices	# slices/area
Total	9363	1.00
Key Scheduling	6554	0.70
Crypto Core	2809	0.30

TABLE X
Two Fish Area Requirement.

C.2 Cryptographic Core

The cryptographic core of Rijndael consists of 10 rounds. The cryptographic core is ideal for implementations on FPGAs. It combines fine-grain parallelism with look-up table operations. The round transformation can be represented as a look-up table resulting in extremely high speed. We have implemented a ROM-based fully-parallel version of the look-up table. By combining common references to the look-up table, we have achieved a 25implementation suggested in the AES proposal [7]. The simplicity of the operations and the inherent fine-grain parallelism resulted in the highest throughput among all the implementations.

D. Serpent

The Serpent block cipher is the AES proposal of R. Anderson, E. Biham, and L. Knudsen from Technion, Cambridge University, and University of Bergen respectively [2]. The time performance and area requirements results for our Serpent implementation are shown in Tables 7 and 8.

D.1 Key Schedule

The Serpent key scheduling expands the input 128-bit key into a 4224-bit key. First, the input key is padded to 256 bits and then it is expanded to an intermediate key by iterative mixing of the key data. Finally, by using look-up tables, the keys for all the rounds are calculated. The simplicity of the required operations results in extremely low latency (the second lowest among all the implementations considered in this paper).

D.2 Cryptographic Core

The cryptographic core of serpent consists of 32 rounds. The round transformation is a linear transform consisting of rotations, shifts, and XOR operations. Neither multiplication nor addition is required. As a result, the highest clock speed and the most compact implementation are achieved among all the implementations. Furthermore, the Ser pent implementation has the highest area utilization factor (i.e. throughput per area unit).

E. Two Fish

The Two fish block cipher is the AES proposal of the Counterpane Systems, Hi/fn, Inc., and D. Wagner from the University of California Berkeley [16]. The time performance and area requirements results of our implementation are shown in Tables 9 and 10.

E.1 Key Schedule

The Two fish key scheduling expands the input 128-bit key into a 1280-bit key. Moreover, it generates the key-dependent S-box es used in the cryptographic core. Four 128-bit S-boxes are generated. Since our goal is to minimize latency, we have implemented a parallel version of the key scheduling consisting of 24 q0=q1 permutation boxes and 2 MDS matrices [16]. Moreover, the RS matrix was implemented for the S-box generation. The matrices are used for constant matrix"-to-matrix multiplication over GF(28). The best known implementation of a constant coefficient multiplier in FPGAs is by using a look-up table. As a result, low latency was achieved but excessive area was required. The area requirements represent the 70a more compact design (e.g. reusing processing elements), increases the latency.

E.2 Cryptographic Core

The cryptographic core of Two fish consists of 16 rounds. The structure of the round transformation is similar to the structure of the key-expansion circuit. The only major difference is the S-boxes that the cryptographic core uses.

V. FPGA Implementations Comparisons

In Table 11, latency comparisons are made among the FPGA implementations. The comparisons are made in terms of absolute time and the ratio of the latency time to the time required to encrypt one block of data. The latter metric represents the capability of agile key-context switching with respect to the encryption rate.

In Table 12, throughput comparisons are made among the FPGA implementations. The comparisons are made in terms of the encryption rate and the ratio of the encryption rate over the area requirements. Then latter metric

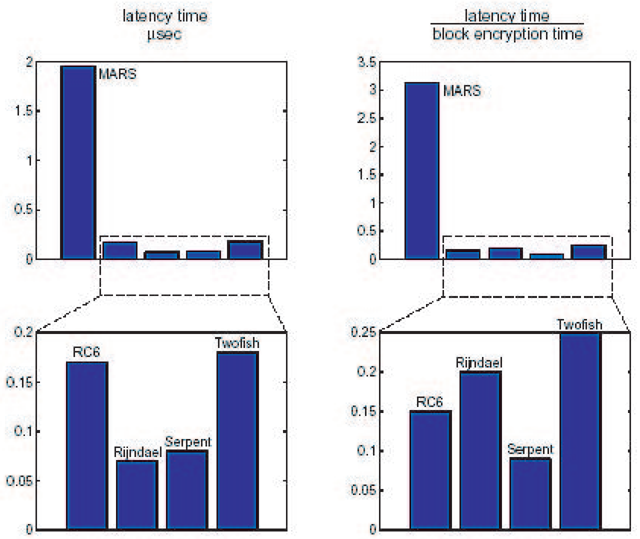

Fig. 1. Latency comparision of FPGA implementations.

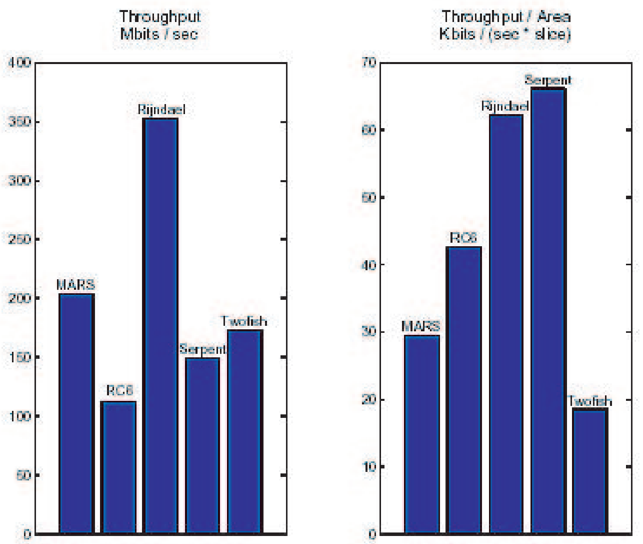

Fig. 2. Throughput comparision of FPGA implementations.

reveals the hardware utilization efficiency of each implementation. Rijndael achieves the highest encryption rate due to the matching of its algorithmic characteristics with the hardware characteristics of FPGAs. In addition, the encryption rate of Rijndael is higher than the ones achieved by the other algorithms by a factor of 20-700. Moreover, Rijndael also achieves very efficient hardware utilization. The best hardware utilization is achieved by Serpent followed closely by Rijndael. The latter metric combines , for each algorithm, the computational demands in terms of an FPGA implementation with the inherent parallelism of the cryptographic round. Finally, in Table 13, area comparisons are made among the FPGA implementations. The comparisons are made in terms of the total area as well as the area required by each of the key-setup and the cryptographic core circuits. Serpent and RC6 have the most compact implementations. Serpent also has the most compact cryptographic core circuit while RC6 has the most compact key-setup circuit. For the MARS block cipher, the result shown is based on an implementation that does not include the circuit for modifying the multiplication key-words [6].

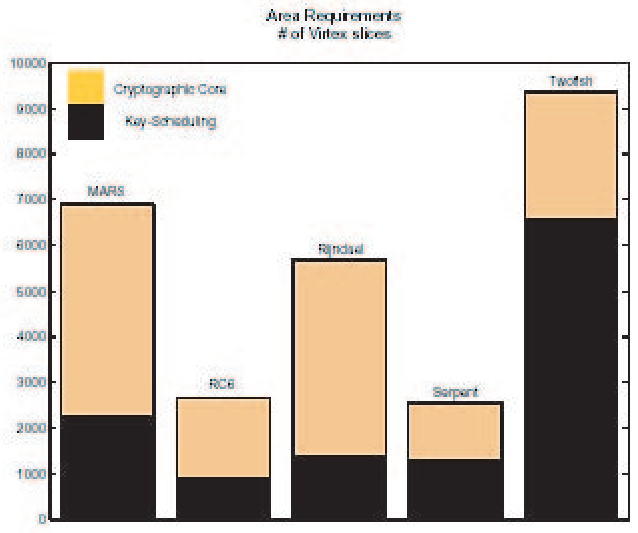

Fig. 3. Area Requirements of Virtex Slices

VI. Conclusion

In this paper we have provided precise time performance and area requirements results for the implementations of the five final AES candidates (MARS,RC6, Rijndael, Serpent, and Two Fish) using FPGAs. To the best of our knowledge, we are not aware of any published extensive results for all the AES final candidates. Our implementations show that, compared with software implementations (NIST Efficiency Testing [1]), superior performance can be achieved. In particular, the latency is reduced by a factor of 20-700 while the throughput speedup is 4-20. In addition, the key-setup process can be performed in parallel with the encryption process regardless the capability of the software implementation to support on-the-fly key scheduling. Based on the time performance results, the Rijndael implementation achieves the highest encryption rate and the lowest latency time due to the ideal matching of its algorithmic characteristics with the characteristics of FPGAs. The work reported here is part of the USCMAARCII project http://maarcII.usc.edu). This project is developing novel mapping techniques to exploit dynamic reconfiguration and facilitate run-time mapping using configurable computing devices and architectures. The goal is to alleviate the long map-ping time required by conventional CAD tools. Computational models and algorithmic techniques based on these models are being developed to exploit self-reconfiguration using FPGAs. Moreover, a domain-specific mapping approach is being developed to sup-port instance-dependent mapping. Finally, the idea offl" libraries is exploited to develop a framework for automatic dynamic reconfiguration [3, 4, 8, 9, 18].

References

[1] "A comparative study of performance of aes final candidates using fpgas," .

[2] Cameron Patterson, "A dynamic fpga implementation of the serpent block cipher," in *Cryptographic Hardware and Embedded Systems -CHES 1999*, 1998, AES Proposal, pp. 141–156.

[3] V. K. Prasanna K. Bondalapati, "Dynamic precision management for loop computation on recon
gurable architectures," 1999, IEEE Symposium on FPGAs for Custom Computing Machines.

[4] S. Choi, "Active library for con
gurable systems," 2000.

[5] J. Rose S. Brown, "Fpga and cpld architectures," 1996, IEEE Design Test of Computers,.

[6] C. Burwick et al., "Mars - a candidate cipherf or aes," 1999, AES Proposal.

[7] V. Rijmen J. Daemen, "The rijndael block cipher," 1999, AES Proposal.

[8] A. Dandalis, "Dynamic logic synthesis for reconfigiurable devices," 2000.

[9] V. K. Prasanna A. Dandalis, A. Mei, "Domain specific mapping for solving graph problems on reconfigurable devices," 1999, Reconfigurabel Architecture Workshop.

[10] C. Paar A. J. Elbirt, "An fpga implementation and performance evaluation of the serpent block cipher," 2000, Eighth ACM International Symposium on Field-Programmable Gate Arrays.

[11] D. Fowler, "Virtual private networks: Making the right connection," 1999.

[12] R. Sidney R. L. Rivest, M. J. B. Robshaw, "The rc6 block cipher," 1998, AES Proposal.

[13] A. Sangiovanni-Vincentelli J. Rose, A. El Gamal, "Architecture of field programmable gate arrays," 1993, Proceedings of the IEEE.

[14] S. C. Goldstein R. R. Taylor, "A high performance flexible architecture for cryptography," 1999, Workshop on Cryptographic Hardware and Embedded Systems.

[15] B. Schneier, "Applied cryptography," 1996.

[16] B. Schneier et al., "Performance comparison of the aes submissions," 1999, Second AES Candidate Conference.

[17] V. K. Prasanna R. P. Sidhu, A. Mei, "Genetic programming using self-reconfigurable fpgas," 1999, International Workshop on Field Programmable Logic and Applications.

[18] "Virtex series fpgas," .