A Comparative Study of Performance of AES Final Candidates Using FPGAs*

Andreas Dandalis, Viktor K. Prasanna, and Jose D. P. Rolim†
Department of Electrical Engineering-Systems
University of Southern California, Los Angeles, USA
{dandalis, prasanna, rolim}@halcyon.usc.edu
http://maarcII.usc.edu/

Abstract

In this paper we study and compare the performance of FPGA-based implementations of the five final AES candidates (MARS, RC6, Rijndael, Serpent, and Twofish). FPGAs seem to match extremely well with the operations required by the final candidates. Among the various time-space implementation tradeoffs, we focused primarily on time performance. The time performance metrics are throughput and latency. Throughput corresponds to the amount of data processed per time unit while latency is the time required to adapt an algorithm to the input key. Time performance and area requirement results are provided for all the final AES candidates. To the best of our knowledge, we are not aware of any published extensive results for all the AES final candidates. Our FPGA implementations show that superior performance can be achieved compared with software implementations. In particular, the latency is reduced by a factor of 20-700 while the throughput speedup is 4-20.

1 Introduction

The projected key role of AES in the 21st century cryptography led us to implement the AES final candidates using Field Programmable Gate Arrays (FPGAs). The goal of this study is to evaluate the performance of the AES final candidates on FPGAs and to make performance comparisons. In addition, we evaluate the suitability of reconfigurable hardware as an alternative solution for AES implementations.

In this study, we concentrate only on performance issues. We assume that all the considered algorithms are secure. Time performance and area requirements results are provided for all the final candidates. The time performance metrics are throughput and latency. Throughput corresponds to the amount of data processed per time unit while latency is the time required to adapt an algorithm to the input key (i.e. key-setup). Besides the throughput metric, the latency metric is the key measure for applications where a small amount of data is processed per key and key context switching occurs repeatedly. To the best of our knowledge, we are not aware of any published extensive results for all the AES final candidates.

In [17], the results are based on estimates and are focused on high-level issues that affect the time performance. In [10], the cryptographic core of *Serpent* was implemented using FPGAs. Only performance results for the cryptographic core were shown. In addition, in [14], the cryptographic cores of *RC6* and *Twofish* were implemented using their own non FPGA-based reconfigurable architecture. Again, only performance results for the cryptographic core were shown. No results regarding the key-setup of the algorithms were provided in [10, 14].

FPGA technology is a growing area that has the potential to provide the performance benefits of ASICs and the flexibility of processors. This technology allows application specific hardware circuits to 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 [5, 13].

Software-based AES implementations provide supe-

*This research was performed as part of the MAARCII project. This work is supported by the DARPA Adaptive Computing Systems program under contract no. DABT63-99-1-0004 monitored by Fort Huachuca.

†J. D. P. Rolim is with the Centre Universitaire d'Informatique, Universite de Geneve, 24 Rue General Dufour, 1211 Geneve 4, Switzerland. This work was performed while he was visiting the University of Southern California.

rior flexibility since any algorithm can be virtually executed on a processor. However, for data rates higher than those found in a T1 line, software-driven solutions are inadequate. In this case, ASIC-based solutions can provide the required time performance. However, the functionality of an ASIC design is restricted by the designed parameters provided during fabrication. Hence, any update to an ASIC-based platform incurs high cost. As a result, ASIC-based approaches lack flexibility. On the other hand, FPGA-based solutions can offer an alternative approach that combines flexibility, agile key switching, and high performance.

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 throughput 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 key-context switching.

In our implementations, we focus on the time performance. Our goal is to exploit, for each candidate, the inherent parallelism of the cryptographic core to optimize performance. Moreover, we exploit the low-level hardware features of FPGAs to enhance the performance of individual processing elements. Our time performance results are compared with the software-based results of the "NIST's Efficiency Testing for Round1 AES Candidates" [1]. Finally, comparisons are made among the implementations in terms of time performance and area requirements.

An overview of FPGAs and FPGA-based cryptography is given in Section 2. In Section 3, general aspects of our implementations are discussed. The implementation results for each algorithm are described in Section 4. In Section 5, time performance comparisons with software implementations are made. A comparative analysis among the results of all the candidates is performed in Section 6. Finally, in Section 7, possible future work is described and concluding remarks are made.

2 FPGA Overview

Processors and ASICs are the cores of the two major computing paradigms of our days. Processors are general purpose and can virtually execute any operation. However, their performance is limited by the restricted interconnect, datapath, 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.

FPGAs were initially an offshoot of the quest for ASIC prototyping with lower design cycle time. The evolution 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 [5, 13] 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.

2.1 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 uniprocessor 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 a "single-round" based implementation (one round is implemented and is reused repeatedly). On the contrary, in feedback modes of operation (e.g. CBC, CFB), where encryption results of each block are fed back into the encryption of the current block [15], encryption can not be parallelized among consecutive blocks of data. As a result, the maximum throughput that can be achieved is equal to the throughput achieved by a "single-round" based implementation.

Besides throughput, FPGA implementations can also achieve agile key-context switching. Key-context switching includes the generation of the required key-dependent data for each cryptographic round (e.g. subkeys, key-dependent S-boxes). A cryptographic round can commence as soon as its key-related data is available. In the case of software implementations, the cryptographic process can not commence before the key-setup process for all the rounds is completed. As a result, excessive latency is introduced making key-context switching inefficient. On the contrary, in FPGAs, each cryptographic round can commence as early as possible since the key-setup process can run concurrently. As a result, minimal latency can be achieved.

Security issues also make FPGA implementations more advantageous than software-based solutions. An encryption algorithm running on a generalized computer has no physical protection [15]. Hardware cryptographic devices can be securely encapsulated to prevent any modification of the implemented algorithm. In general, hardware-based solutions are the embodiment of choice for military and serious commercial applications (e.g. NSA authorizes encryption only in hardware) [15].

Finally, even if ASICs can achieve superior performance 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.

3 Implementation & Design Decisions

As a hardware target for the proposed implementations, we have chosen the Xilinx Virtex family of FPGAs. Virtex is a high-capacity, high-speed performance FPGA providing a superior system integration feature set [19]. For mapping onto VIRTEX devices, we used the Foundation Series v2.1i software development tool. The configuration of the tool remained the same for all the implementations. All the results were based on placed-and-routed implementations that included both the key-setup component and the cryptographic core along with their control circuit.

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-fly 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. Since one round is implemented and is reused repeatedly, the throughput results correspond to $\frac{128}{n * t_{round}}$, where n and t_{round} are the the number of required rounds and the encryption time per round respectively. 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.

To implement efficient key-setup circuits, we took advantage of the embedded memory modules (Block SelectRAM) of the Virtex FPGAs [19]. The Virtex FPGA Series provides dedicated on-chip blocks of true dual-read/write port synchronous RAM, with 4096 memory cells each. Depending on the size of the device, 32-132 Kbits of data can be stored using the Block SelectRAM memory modules. The key-setup circuit utilizes these memory modules to pass its results to the cryptographic core. As a result, the cryptographic core can commence as soon as the key-dependent data (e.g. subkeys, S-boxes) for the first encryption round is available in the memory modules. Then, during each encryption round, the cryptographic core reads the corresponding data from the memory modules.

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 50% and the area requirements were also reduced significantly.

4 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 provided for both the key-setup and the cryptographic core circuits. In the following, the order of presenting the algorithms is alphabetic. Detailed algorithmic information for each candidate can be found in [6, 12, 7, 2, 16].

4.1 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.

Table 1: MARS Time Performance

Latency		Throughput	
µs	Latency time / block encryption time	MBits / sec	KBits / (sec*slice)
1.96	3.12	203.77	29.55

Table 2: MARS Area Requirements

	Area Requirements	
	# of slices	# of slices / total area
Total	6896	1.00
Key-Scheduling	2275	0.33
Cryptographic Core	4621	0.67

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 an S-box. Both processes involves simple operations performed repeatedly. However, the final stage of modifying the mul-

tiplication 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. In spite of this, the introduced latency was still relatively high (the worst among all the implementations considered in this paper).

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.

4.2 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.

Table 3: RC6 Time Performance

Latency		Throughput	
µs	Latency time / block encryption time	MBits / sec	KBits / (sec*slice)
0.17	0.15	112.87	42.59

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 15% of the time for encrypting a block of data.

Table 4: RC6 Area Requirements

	Area Requirements	
	# of slices	# of slices / total area
Total	2650	1.00
Key-Scheduling	901	0.34
Cryptographic Core	1749	0.66

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.

4.3 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 $N_b = 4$, $N_k = 4$, and $N_r = 10$. The time performance and the area requirements results of our implementation are shown in Tables 5 and 6.

Table 5: Rijndael Time Performance

Latency		Throughput	
µs	Latency time / block encryption time	MBits / sec	KBits / (sec*slice)
0.07	0.20	353.00	62.22

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

Table 6: Rijndael Area Requirements

	Area Requirements	
	# of slices	$\frac{\text{\# of slices}}{\text{total area}}$
Total	5673	1.00
Key-Scheduling	1361	0.24
Cryptographic Core	4312	0.76

latency. ROM-based look-up tables are utilized to perform the *SubByte* transformation. The achieved latency is the lowest among all the implementations considered in this paper.

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 25% savings in ROM compared with the straightforward implementation 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.

4.4 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.

Table 7: Serpent Time Performance

Latency		Throughput	
µs	$\frac{\text{Latency time}}{\text{block encryption time}}$	MBits / sec	KBits / (sec*slice)
0.08	0.09	148.95	66.20

Table 8: Serpent Area Requirements

	Area Requirements	
	# of slices	$\frac{\text{\# of slices}}{\text{total area}}$
Total	2550	1.00
Key-Scheduling	1300	0.51
Cryptographic Core	1250	0.49

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).

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 Serpent implementation has the highest area utilization factor (i.e. throughput per area unit).

4.5 Twofish

The Twofish 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.

Table 9: Twofish Time Performance

Latency		Throughput	
µs	$\frac{\text{Latency time}}{\text{block encryption time}}$	MBits / sec	KBits / (sec*slice)
0.18	0.25	173.06	18.48

Table 10: Twofish Area Requirements

	Area Requirements	
	# of slices	# of slices / total area
Total	9363	1.00
Key-Scheduling	6554	0.70
Cryptographic Core	2809	0.30

Key Schedule The Twofish key scheduling expands the input 128-bit key into a 1280-bit key. Moreover, it generates the key-dependent S-boxes 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 q_0/q_1 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 $GF(2^8)$. 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 70% of the total area. However, by implementing a more compact design (e.g. reusing processing elements), increases the latency.

Cryptographic Core The cryptographic core of Twofish 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.

4.6 Latency Improvements

For each algorithm, we have also implemented the key-setup circuit and the cryptographic core separately. For each algorithm, the maximum clock speed of the key-setup circuit was higher than the maximum clock speed of the cryptographic core. Thus, by clocking each circuit at its maximum clock speed, improvement in latency can be achieved. No additional synchronization hardware is required since we can configure the read/write ports of the Block SelectRAMs having different clock speeds. In Table 11, based on the results of these individual implementations, we provide the potential latency time improvement by using two different clocks.

Clearly, the RC6 block cipher can achieve the best latency time improvement by clocking the key-setup and the cryptographic core circuits at their maximum clock speeds. For the MARS block cipher, the results shown are based on an implementation that does not include the circuit for modifying the multiplication key-words.

Table 11: Latency Time Improvement by using 2 clocks

	Latency time (2-clocks)	
	µs	Latency time (1-clock) / Latency time (2-clocks)
MARS	1.45	1.35
RC6	0.06	2.96
Rijndael	0.05	1.43
Serpent	0.08	1.00
Twofish	0.16	1.15

5 Comparison with Software Implementations

Our performance results are compared with the software-based results of the "NIST's Efficiency Testing for Round1 AES Candidates" [1]. The reference platform for the NIST's efficiency testing was a Pentium Pro with 64 MB RAM running at 200 MHz. As noted in the corresponding report, NIST used only the optimized code provided by the submitters of the candidate algorithms.

In Table 12, the latency results of our implementations and those of the software implementations are shown. The results are represented both as absolute time and as a fraction of the corresponding encryption time of one 128-bits block of data. Clearly, the FPGA implementations achieve significant reduction in the key-setup time (by a factor of 20-700). On the contrary, the key-setup time of the software implementations is equal to the time for encrypting 3-13 blocks of data. In FPGAs, each cryptographic round can commence as early as possible since the key-setup process can run concurrently with the cryptographic core. In the case of software implementations, the cryptographic core can not commence before the key-setup process for all the rounds is completed. Thus, while FPGA implementations favor agile key-context switching, the software implementations require relatively long time for key-context switching. The latency metric is the key performance measure for applications where small amount of data is processed per key and key context switching occurs repeatedly. For example, in the case that a block cipher is used

Table 12: Latency comparisons with NIST Efficiency Testing results [1]

AES Algorithm	Latency (μs)		Latency / block encryption time	
	Software	Our	Software	Our
MARS	38.11	1.96	7.91	3.12
RC6	25.07	0.17	5.93	0.15
Rijndael	33.93	0.07	8.39	0.20
Serpent	56.99	0.08	3.33	0.09
Twofish	63.99	0.18	13.15	0.25

to perform a hash function, the input key changes for every other block of data [17]. In addition, the latency metric is critical for IPSec since the input key changes frequently depending on the lifetime of the established security association.

In Table 13, encryption throughput results are shown and comparisons with the software implementations are made. The throughput improvements are 4-20 times compared with the software-based results. While the known software implementations do not achieve processing rates higher than 30 Mbits/sec, our FPGA implementations achieve processing rates higher than 100 Mbits/sec. For one reason, software implementations can not exploit the inherent parallelism of a cryptographic round. For another, the operations required by each cryptographic round can be executed more efficiently in FPGAs than in a general-purpose computer. The throughput speed-up can be further improved for implementations that process multiple blocks of data (see Section 3).

Table 13: Throughput comparisons with NIST Efficiency Testing results [1]

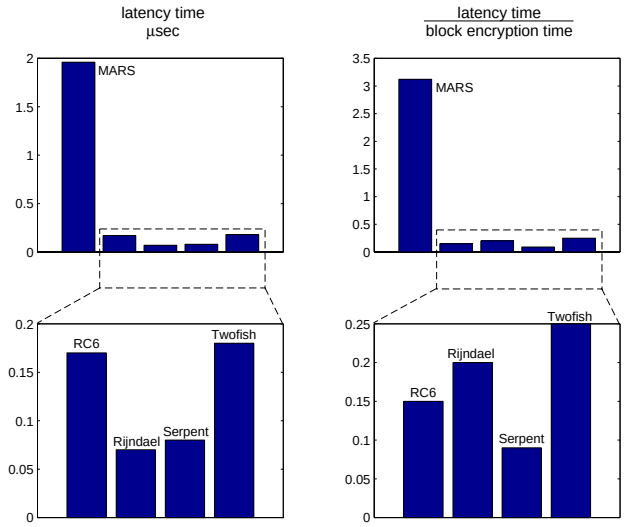
AES Algorithm	Throughput (MBits/sec)		Speed-up
	Software	Our	
MARS	26.56	203.77	7.67
RC6	30.29	112.87	3.72
Rijndael	31.64	353.00	11.15
Serpent	7.48	148.95	19.91
Twofish	26.31	173.06	6.58

By using a superior platform configuration than the reference platform of NIST efficiency testing, higher throughput can be achieved for the software implementations. However, even in this case, the speed-up of the FPGA implementations would be remarkable. On the other hand, the latency results would not be affected since the fraction over the corresponding encryption time of one block of data would be the same.

6 FPGA Implementations Comparisons

In Table 14, 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.

Table 14: Latency comparisons of the FPGA implementations

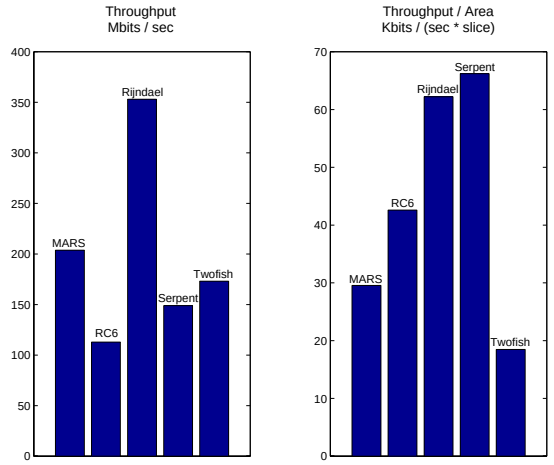


Clearly, Rijndael and Serpent achieve the lowest latency times while the latency times for RC6 and Twofish are higher by a factor of 2.5. As we have mentioned in Section 4, the latency introduced by MARS is the highest. All the algorithms (except MARS) achieve latency time that is equal to the 7-25 % of the time for encrypting a block of data.

In Table 15, 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. The latter metric reveals the hardware utilization efficiency of each implementation.

Rijndael achieves the highest encryption rate due to the matching of its algorithmic characteristics with

Table 15: Throughput comparisons of the FPGA implementations



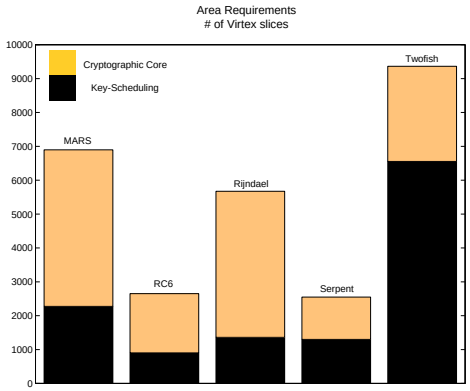
the hardware characteristics of FPGAs. In addition, the encryption rate of Rijndael is higher than the ones achieved by the other algorithms by a factor of $1.7 - 3.12$. Moreover, Rijndael also achieves very efficient hardware utilization. The best hardware utilization is achieved by Serpent followed 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 16, 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].

7 Conclusions

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 Twofish) 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.

Table 16: Area comparisons of the FPGA implementations



The work reported here is part of the USC MAARCII 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 mapping 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 support instance-dependent mapping. Finally, the idea of "active" libraries is exploited to develop a framework for automatic dynamic reconfiguration [3, 4, 8, 9, 18].

References

[1] Advanced Encryption Standard, http://www.nist.gov/aes/

[2] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard", AES Proposal, June 1998.

[3] K. Bondalapati and V. K. Prasanna, "Dynamic Precision Management for Loop Computations on Reconfigurable Architectures", IEEE Symposium on FPGAs for Custom Computing Machines, April 1999.

[4] S. Choi, "Active Library for Configurable Systems", PhD Qualification Exam Report, University of Southern California, February 2000.

[5] S. Brown and J. Rose, "FPGA and CPLD Architectures: A Tutorial", IEEE Design & Test of Computers, Summer 1996.

[6] C. Burwick et al., "MARS - a candidate cipher for AES", AES Proposal, August 1999.

[7] J. Daemen, V. Rijmen, "The Rijndael Block Cipher", AES Proposal, September 1999.

[8] A. Dandalis, "Dynamic Logic Synthesis for Reconfigurable Devices", PhD Thesis, University of Southern California. Under Preparation.

[9] A. Dandalis, A. Mei, and V. K. Prasanna, "Domain Specific Mapping for Solving Graph Problems on Reconfigurable Devices", Reconfigurable Architectures Workshop, April 1999.

[10] A. J. Elbirt, C. Paar, "An FPGA Implementation and Performance Evaluation of the Serpent Block Cipher", Eighth ACM International Symposium on Field-Programmable Gate Arrays, February 2000.

[11] D. Fowler, "Virtual Private Networks: Making the Right Connection", Morgan Kaufmann Publishers, Inc., San Francisco, California, 1999.

[12] R. L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin, "The RC6^{TM} Block Cipher", AES Proposal, June 1998.

[13] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of Field Programmable Gate Arrays", Proceedings of the IEEE, July 1993.

[14] R. R. Taylor, S. C. Goldstein, "A High-Performance Flexible Architecture for Cryptography", Workshop on Cryptographic Hardware and Embedded Systems, August 1999.

[15] B. Schneier, "Applied Cryptography", John Willey & Sons, Inc., 2nd edition, 1996.

[16] B. Schneier, J. Kelsey, D. Whitingz, D. Wagnerx, and C. Hall, "Twofish: A 128-Bit Block Cipher", AES Proposal, June 1998.

[17] B. Schneier et al., "Performance Comparison of the AES Submissions", Second AES Candidate Conference, April 1999.

[18] R. P. Sidhu, A. Mei, and V. K. Prasanna, "Genetic Programming using Self-Reconfigurable FPGAs", International Workshop on Field Programmable Logic and Applications, September 1999.

[19] Virtex Series FPGAs, http://www.xilinx.com/products/virtex.htm