Comparison of the hardware performance of the AES candidates using reconfigurable hardware

Kris Gaj and Pawel Chodowiec
George Mason University
kgaj@gmu.edu, pchodowi@gmu.edu

Abstract
The results of implementations of all five AES finalists using Xilinx Field Programmable Gate Arrays are presented and analyzed. Performance of four alternative hardware architectures is discussed and compared. The AES candidates are divided into three classes depending on their hardware performance characteristics. Recommendation regarding the optimum choice of the algorithms for AES is provided.

1. Introduction

Hardware implementations of cryptography will thrive in the new century because of the growing requirements for high-speed, high-volume secure communications combined with physical security. In the presence of no major breakthroughs in cryptanalysis of the AES candidates, and relatively inconclusive results of their software performance evaluation [NBD+99, SKW+99], the comparison of the hardware performance of the AES algorithms may provide a major indicator for a final decision regarding the new standard.

Very few results regarding hardware implementations of the AES candidates have been published so far. Original documentation provided by designers of the submitted algorithms contains typically only rough estimates of the hardware performance [BCD+98, RRS+98, SKW+98]. Additionally, these estimates are very difficult to compare among each other because of large differences in assumptions regarding the technology, and because of different architecture choices. The results of actual implementations of individual algorithms, published recently by independent researchers [EP99, RH99], provide only a very fragmentary knowledge, not suitable for reliable comparison.

This situation will be certainly remedied by the publication of the NSA findings regarding hardware performance of the AES candidates. Nevertheless, the NSA evaluation plan [NSA98] targets only implementations using *semi-custom Application Specific Integrated Circuits* (ASICs), providing no data regarding other technologies. In this article, we focus on comparing AES candidates using an alternative hardware technology based on Field Programmable Gate Arrays (FPGAs). This technology, referred to as *reconfigurable hardware*, offers many advantages for future vendors and users of cryptographic equipment. It assures a short time to the market, high flexibility (including a capability for frequent modifications of hardware), low development costs, and low cost of the final product - the result of the algorithm agility - capability to use the same integrated circuit with time sharing for the execution of various secret-key and public-key algorithms. Our comparison supplements the NSA effort by covering the second primary way of implementing cryptographic algorithms in hardware.

2. Reconfigurable hardware

2.1 Operation and internal structure of an FPGA device

Field Programmable Gate Array (FPGA) is an integrated circuit that can be bought off the shelf and reconfigured by designers themselves. With each reconfiguration, which takes only a fraction of a second, an integrated circuit can perform a completely different function. FPGA consists of thousands of universal building blocks, known as *Configurable Logic Blocks* (*CLBs*), connected using programmable interconnects, as shown in Fig. 1a. Reconfiguration is able to change a function of each CLB and connections among them, leading to a functionally new digital circuit.

From several FPGA families available on the market, we have chosen for implementing AES candidates two families from Xilinx, Inc.: high performance Virtex family, and a low-cost XC4000 family. Each family consists of several FPGA devices, manufactured in the same technology, covering certain range of maximum circuit sizes.

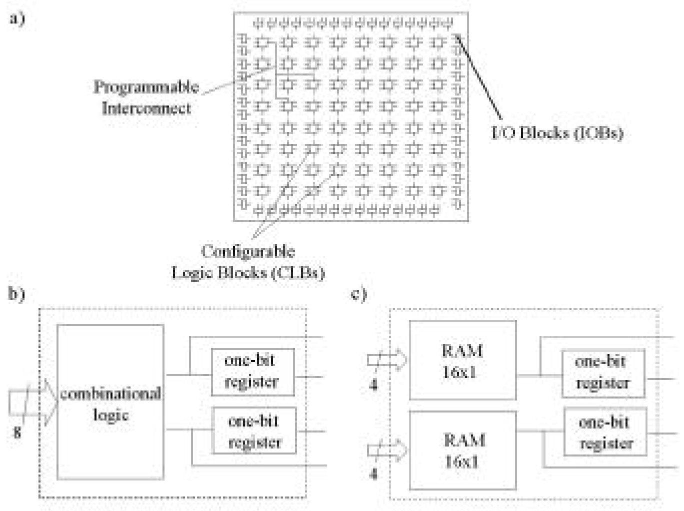

Fig. 1 FPGA device. a) General structure and main components. b) Internal structure of a CLB configured in the logic mode. c) Internal structure of a CLB configured in the memory mode.

A simplified internal structure of a CLB in the XC4000 family, and a *CLB slice* (1/2 of a CLB) in the Virtex family is shown in Figs. 1bc. In the logic mode (Fig. 1b), each of these elementary units contains a small block of combinational logic, implemented using programmable look-up tables, and two one-bit registers. In the memory mode, combinational logic is replaced by two small memories. A CLB in the XC4000 family of FPGA devices and a CLB slice in Virtex are functionally almost identical. Therefore, we will use a number of these elementary units, necessary to build a given circuit, as a measure of the circuit area and cost.

2.2 Advantages of using reconfigurable hardware for comparison of the AES candidates

For implementing cryptography in hardware, FPGAs provide the only major alternative to *custom and semi-custom Application Specific Integrated Circuits* (ASICs), integrated circuits that must be designed all the way from the behavioral description to the physical layout, and sent for an expensive and time-consuming fabrication. The comparison of the AES candidates based on FPGA devices has the following advantages over the comparison based on ASICs:

- Shorter design cycle leading to fully functioning device prototypes.
- Lower cost of the computer-aided design tools, verification, and testing.
- Potential for fast, low-cost multiple reprogramming and experimental testing of a large number of various architectures and revised versions of the same architecture.
- Higher accuracy of comparison: in the absence of the physical design and fabrication, ASIC designs are compared based on inaccurate pre-layout simulations [NSA98]; FPGA designs are compared based on very accurate post-layout simulations and experimental testing.

3. Alternative architectures

3.1 Basic organization of a block cipher implementation

The basic organization of the hardware implementation of a symmetric block cipher is shown in Fig. 2. All five AES candidates investigated in this paper can be implemented using this organization. The organization includes the following units:

a. *Encryption/decryption unit*, used to encipher and decipher input blocks of data.
b. *Key scheduling unit*, used to compute a set of internal cipher keys based on a single external key.
c. *Memory of internal keys*, used to store internal keys computed by the key scheduling unit, or loaded to the integrated circuit through the input interface.
d. *Input interface*, used to load blocks of input data and internal keys to the circuit, and to store input blocks awaiting encryption/decryption.

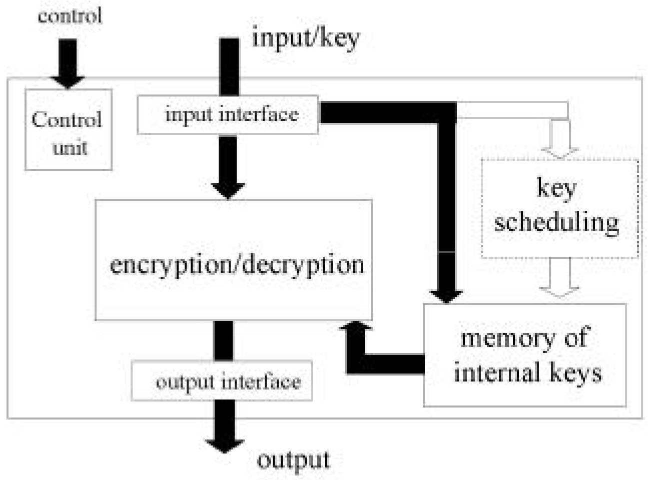

Fig. 2 Block diagram of the hardware implementation of a symmetric-block cipher.

e. *Output interface*, used to temporarily store output from the encryption/decryption unit and send it to the external memory.
f. *Control unit*, used to generate control signals for all other units.

3.2 Feedback vs. non-feedback operating modes

Today's symmetric block ciphers are used in several operating modes. From the point of view of hardware implementations, these modes can be divided into two major categories:
a. *Non-feedback modes*, such as Electronic Code Book mode (ECB), and counter mode.
b. *Feedback modes*, such as Cipher Block Chaining mode (CBC), Cipher Feedback Mode (CFB), and Output Feedback Mode (OFB).

In the non-feedback modes, encryption of each subsequent block of data can be performed independently from processing other blocks. In particular, all blocks can be encrypted in parallel. In the feedback modes, it is not possible to start encrypting the next block of data until encryption of the previous block is completed. As a result, all blocks must be encrypted sequentially, with no capability for parallel processing.

According to current security standards, the encryption of data is performed primarily using feedback modes, such as CBC and CFB. Non-feedback modes, such as ECB, are used primarily to encrypt session keys during key distribution. As a result, using current standards does not permit to fully utilize the performance advantage of the hardware implementations of secret key cryptosystems, based on parallel processing of multiple blocks of data.

3.3 Alternative architectures for the encryption/decryption unit

a. Basic architecture

The basic hardware architecture used to implement an encryption unit of a typical secret-key cipher is shown in Fig. 3a. One round of the cipher is implemented as a combinational logic, and supplemented with a single register and a multiplexer. In the first clock cycle, input block of data is fed to the circuit through the multiplexer, and stored in the register. In each subsequent clock cycle, one round of the cipher is evaluated, the result is fed back to the circuit through the multiplexer, and stored in the register. The number of clock cycles necessary to encrypt a single block of data is equal to the number of cipher rounds, #rounds.

We define the *speed* of the cipher implementation as the number of bits of data encrypted in a unit of time. Speed calculated this way is often referred to as the circuit *throughput*. The speed of the basic architecture, speed_{ba}, is given by

$$\text{speed}_{ba} = 128/\ \#\text{rounds} \cdot \text{clock\_period}\ . \qquad (1)$$

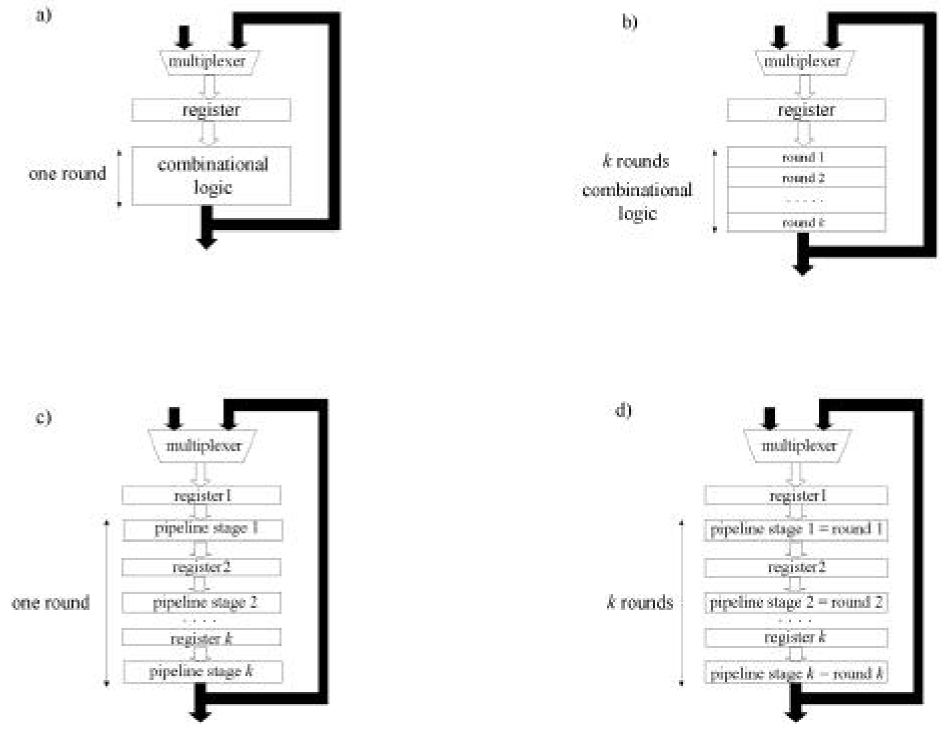

Fig. 3 Four alternative architectures for implementation of an encryption/decryption unit of a block cipher: a) basic architecture, b) architecture with the *k*-round loop unrolling, c) architecture with the *k*-stage inner-round pipelining, d) architecture with the *k*-stage outer-round pipelining.

The basic architecture combines a good speed with the relatively modest area requirements. However there exist several alternative architectures that permit to improve either one or both of these performance measures.

b. Loop unrolling

Architecture with loop unrolling is shown in Fig. 3b. The only difference compared to the basic architecture is that the combinational part of the circuit implements *k* rounds of the cipher, instead of a single round. The maximum value of *k* is equal to the number of cipher rounds. The number of clock cycles necessary to encrypt a single block of data decreases by a factor of *k*. At the same time the minimum clock period increases by a factor slightly smaller than *k*, leading to an overall relatively small increase in the cipher speed, given by

$$\text{speed}_{lu}/\text{speed}_{ba} = (1 + \tau)/(1+\tau/k), \qquad (2)$$

where τ is the ratio of the sum of the multiplexer delay, the register delay and the register setup time to the delay of a single cipher round. This increase in speed is obtained at the cost of the circuit area. Because the combinational part of the circuit constitutes the majority of the circuit area, the total area of the encryption/decryption unit increases almost proportionally to the number of unrolled rounds, *k*. Additionally, the number of internal keys used in a single clock cycle increases by a factor of *k*, which in FPGA implementations typically implies the almost proportional growth in the number of CLBs used to store internal keys.

In summary, loop unrolling enables increasing the circuit speed in both feedback and non-feedback operating modes. Nevertheless this increase is relatively small, and incurs a large area penalty.

c. Inner-round pipelining

Pipelining is a general method of increasing the amount of data processed by a digital circuit in a unit of time. The idea is to introduce evenly spaced extra registers in the middle of the combinational circuit, in such a way that several blocks of data can be processed by the circuit at the same time. Parts of the combinational logic divided by adjacent registers are called pipeline stages (see Fig. 3c). In each clock cycle the partially processed data block moves to the next pipeline stage. Its place is taken by the subsequent data block. This way, a pipelined circuit can encrypt simultaneously as many blocks of data, as the number of pipeline stages it contains.

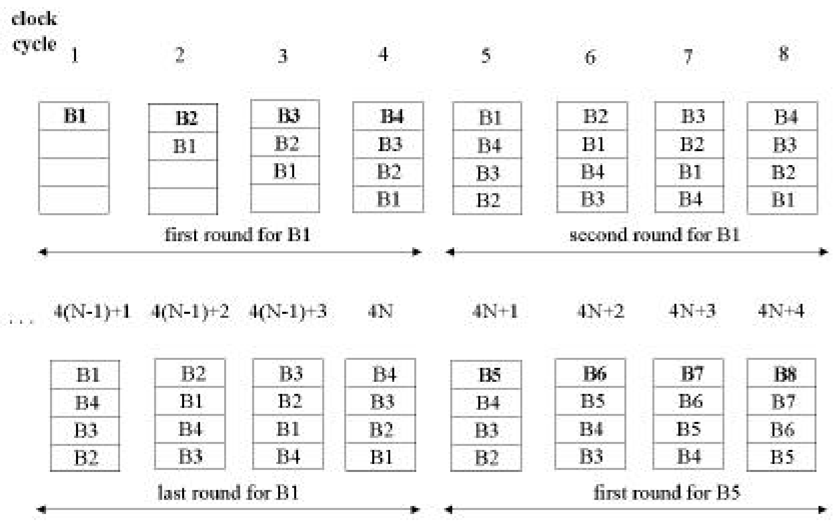

Fig. 4 Operation of the architecture with 4-stage inner-round pipelining for an N-round cipher.

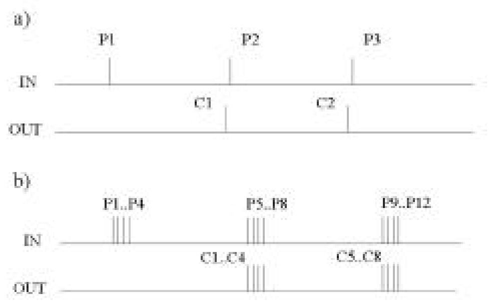

Fig. 5 Timing of input and output blocks in a) basic architecture, b) architecture with a 4-stage inner-round pipelining.

The flow of data through the pipeline during encryption is shown in Fig. 4. The number of pipeline stages in this example is four. During the first four clock cycles four subsequent blocks of data enter the pipeline. In the subsequent clock cycles, these blocks circulate in the pipeline. Each four clock cycles correspond to a single cipher round. In the cycle number 4·#rounds+1, the first block, B1, leaves the pipeline, and the fifth block, B5, is introduced to the empty pipeline stage. In the following three clock cycles, blocks B2, B3, and B4, leave the pipeline, substituted by blocks B6, B7, and B8. The timing diagram of the input and output of the circuit is shown in Fig. 5b. Speed of the circuit, expressed as the number of bits processed by the circuit in a unit of time is given by

$$\text{speed} = 128/\ \#\text{rounds} \cdot \text{reduced\_clock\_period} \qquad (3)$$

where reduced_clock_period is a minimum clock period after pipelining.

The dependence between the cipher speed-up resulting from the inner-round pipelining and the number of evenly spaced pipeline stages is shown in Fig. 6. There exists a maximum number of pipeline stages that still improves the circuit throughput. Adding additional registers will not affect the throughput. The maximum number of pipeline stages is determined by the delay of the largest indivisible combinational portion of the circuit. For majority of ciphers it is difficult to divide the cipher round into combinational stages with equal delays (especially, when the circuit is described in a high-level hardware description language, such as VHDL),

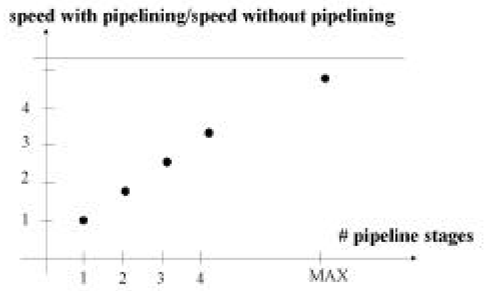

Fig. 6 Speed of the architecture with *k*-round inner-round pipelining as a function of the number of evenly spaced pipeline stages.

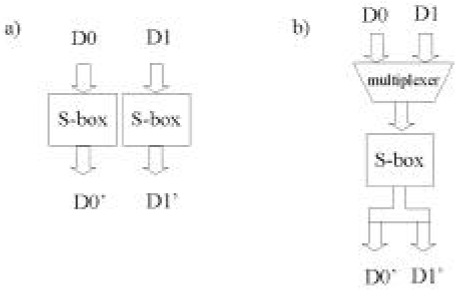

Fig. 7 Resource sharing of an S-box. a) basic operation of two parallel S-boxes, b) operation with resource sharing.

which further limits the circuit speed-up. Area of the circuit with inner-round pipelining increases only by a small percentage (area of a single 128-bit register) with each additional pipeline stage. This is especially true for FPGA circuits, in which CLBs used to implement combinational logic often contain registers not utilized in the non-pipelined implementation.

d. Outer-round pipelining

Outer-round pipelining is created by loop unrolling followed by introducing extra registers between parts of the combinational logic corresponding to each cipher round, as shown in Fig. 3d. The number of unrolled loops *k* is typically a divisor of the total number of cipher rounds, #rounds.

Area of the encryption unit with outer-round pipelining is directly proportional to the number of pipeline stages *k*. In the non-feedback cipher modes, such as ECB, the speed (throughput) of the cipher increases proportionally to the number of pipeline stages, *k*. Therefore, the outer-round pipelining enables to directly trade circuit speed with circuit area. In the feedback cipher modes, the speed of the cipher remains independent of the number of outer pipeline stages, and therefore, this kind of pipelining is not recommended for these modes.

e. Resource sharing

For some ciphers, it is possible to further decrease circuit area by time sharing of certain resources (e.g., function *h* in Twofish, 4x4 S-boxes in Serpent, 8x32 S-boxes S0, S1 in the mixing transformation of Mars, multiplication units in RC6). This is accomplished by using the same functional unit to process two (or more) parts of the data block in different clock cycles, as shown in Fig. 7b. In Fig. 7a, two parts of the data block, D0 and D1, are processed in parallel, using two independent S-boxes. In Fig. 7b, a single S-box is used to process two parts of the data block sequentially, during two subsequent clock cycles.

The use of resource sharing in real life implementations is expected to be limited, because

- Gain in the circuit area is always smaller than the loss in the circuit speed.
- The amount of area used by a basic implementation of a symmetric cipher is typically already quite small.

3.4. Choice of the figure of merit

The choice of a single figure of merit is difficult, because the optimization criteria may vary depending on the application. In our comparison, we took into account three basic figures of merit: maximum speed (throughput), minimum area, and the maximum speed/area ratio.

Optimization for maximum speed will be done in applications where communication requirements force the use of a very high speed encryption, and/or the cost of the cryptographic hardware constitutes only a small portion of the entire system. Examples of such applications include ATM and ISDN switches, Virtual Private

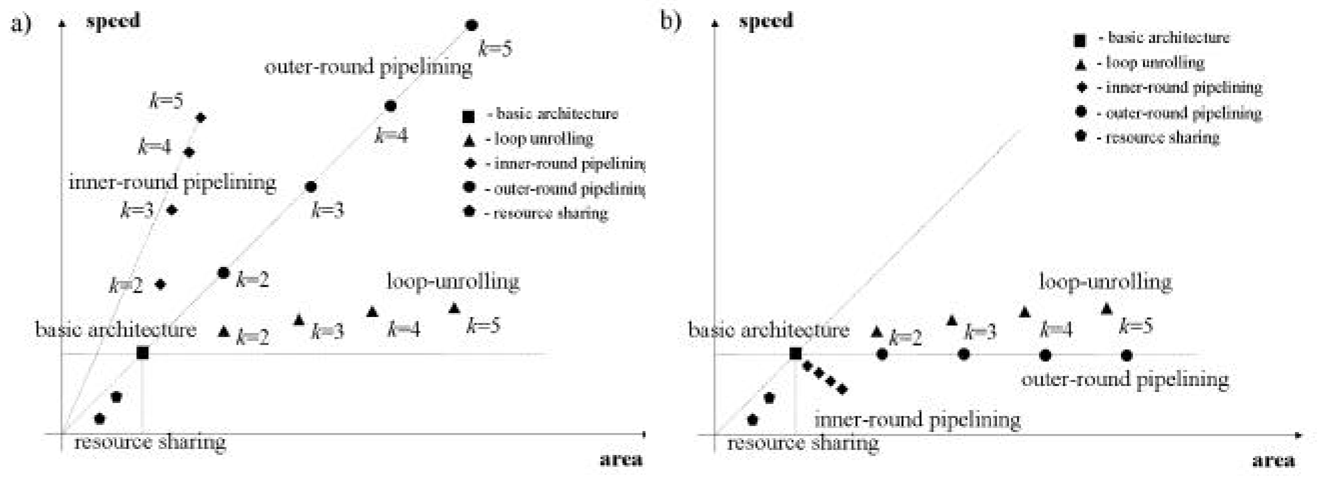

Fig. 8 Hardware performance of various alternative architectures in a) non-feedback cipher modes, such as ECB and counter mode, b) feedback cipher modes, such as CBC, CFB, and OFB.

Network routers and firewalls, WWW and database servers. In such applications, it may be justified to trade the cost of the cryptographic hardware (proportional to the circuit area) for greater speed.

In the second class of applications, the designer's goal is to obtain the maximum speed, assuming a given limit on the circuit area (cost). In such situations, the more appropriate figure of merit is the speed/area ratio. This figure of merit is particularly appropriate for non-feedback cipher modes, which enable one to directly trade circuit area for speed by using the outer-round pipelining, as shown in Fig. 8a. The examples of cost critical applications of cryptography include pagers, digital video recorders, and PCMCIA cards.

Applications that require optimization for minimum area include smart cards, embedded systems, and cellular phones. As the basic architecture may be still too big for such applications, they may enforce resource sharing. Taking into account the size and power limitations, these applications will be typically implemented using custom ASICs, not FPGAs.

3.5 Comparison of various architectures

Dependencies between the speed and the area of the encryption/decryption unit of a block cipher, for architectures discussed in section 3.3, are shown in Fig. 8.

a. Non-feedback modes

For non-feedback modes, the best speed/area ratio can be obtained by using inner-round pipelining with the maximum number of pipeline stages that still increases circuit clock frequency, as shown in Fig. 8a. The largest possible speed can be obtained by combining inner-round pipelining with outer-round pipelining. The only limit on the circuit speed is imposed in this case by the maximum circuit area (cost) and/or the maximum number of the outer-round pipeline stages (equal to the number of the cipher rounds). The smallest possible area can be obtained using the basic architecture with resource sharing.

b. Feedback-modes

For feedback modes, the basic architecture offers the best value of the ratio speed/area, as shown in Fig. 8b. Larger speed can only be obtained using loop unrolling, at the cost of a very significant increase in the circuit area (cost). Smaller area can only be obtained using resource sharing, at the cost of the significant reduction in the circuit speed.

Outer-round pipelining is inefficient in these modes, as it does not increase circuit speed, and significantly increases circuit area. Inner-round pipelining decreases speed, and increases circuit area. As a result, neither type of pipelining should be used in these operating modes.

4. Assumptions

4.1 Primary assumptions

The following tentative assumptions have been made in order to simplify the task of comparing AES candidates:

a. Key size 128 bits.

Our implementations are intended to support only one key size, 128 bits. Other key sizes required by AES (192 and 256 bits), or supported by a particular algorithm will be added in the future.

b. No key scheduling unit.

Our implementations do not support the on-chip generation of internal keys from a single external key. Instead, our implementations include a memory of internal keys loaded with the keys generated externally, and the circuitry necessary to distribute these keys from the memory to the encryption/decryption unit.

c. *Block size 128 bits.*

Only one input/output block size, 128 bits, has been considered, even if the given AES candidate supports other block sizes.

d. Basic architecture

The encryption part of all AES candidates has been implemented using basic architecture shown in Fig. 3a. This architecture has been chosen for the following reasons:

* As shown in Fig. 8b, the basic architecture assures the maximum *speed/area* ratio for feedback operating modes (CBC, CFB), now commonly used for bulk data encryption. It also guarantees near optimum speed, and near optimum area for these operating modes.

* The basic architecture is relatively easy to implement in a similar way for all AES candidates, which supports fair comparison. For architectures with inner-round pipelining, it is relatively difficult to determine and implement the maximum number of pipeline stages that still increases circuit speed and speed/area ratio.

* The implementations of the basic architecture exemplify larger differences among five AES algorithms compared to the architectures with inner-round pipelining. Inner-round pipelining permits decreasing the differences in speed among various ciphers because ciphers with longer critical path (lower speed) may be sped up by a larger factor by introducing proportionally more pipeline stages.

* Based on the performance measures for basic architecture, it is possible to derive analytically *approximate* formulas for parameters of more complex architectures, including architectures with outer-round pipelining (near proportional scaling of both area and speed), loop-unrolling (see formula (2)), and inner-round pipelining (see formula (3) and Fig. 6). Nevertheless, these formulas should be treated only as a first approximation, and the more detailed comparison requires the actual implementation of all ciphers using alternative architectures. Only such implementations may take into account the exact structure of all ciphers, limitations imposed by the FPGA architecture and the design entry method (e.g., VHDL description), and the optimization capabilities of the FPGA computer-aided design tools.

e. Resource sharing between the encryption and decryption part

In order to minimize circuit area, it was assumed that the encryption and decryption parts share as many resources as possible by the given cipher type. The effort was made to maximally decrease the effect of resource sharing on the speed of encryption and decryption.

4.2 Deviations from the basic architecture

Three ciphers, Twofish, RC6, and Rijndael, have been implemented using exactly the basic architecture shown in Fig. 3a. This was possible because all rounds of these ciphers perform exactly the same operation. For the remaining two ciphers, Serpent and Mars, this condition is not fulfilled, and as a result, small deviations from the basic architecture appeared to be necessary.

Serpent consists of 8 different rounds repeated 4 times. Therefore, it is advantageous to treat 8 official cipher rounds as a single *implementation round,* and assume that the cipher has 4 rounds. This way, 8 official cipher rounds are implemented in the basic architecture as a combinational logic. This implementation guarantees the maximum speed/area ratio typical for the basic architecture.

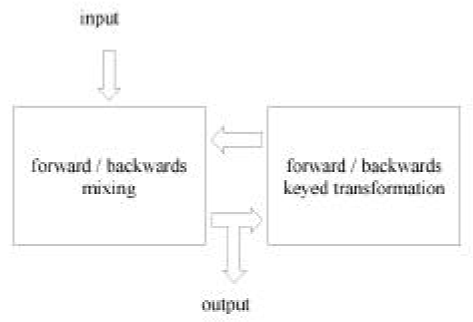
input
forward / backwards mixing
forward / backwards keyed transformation
output

Fig. 9 Deviation from the basic architecture in Mars.

In Mars, there exist four different kinds of rounds, each repeated 8 times: forward mixing, forward keyed transformation, backwards keyed transformation, and backwards mixing. It is possible to implement forward and backwards mixing using the same functional unit; the same holds for the forward and backwards keyed transformation. The structure of the mixing transformation and the keyed transformation are significantly different, and as a result they must be implemented using separate units, as shown in Fig. 9. Both of these units have an internal structure that corresponds to the basic architecture (multiplexer + register + combinational logic). Additionally, both units share the look-up table implementing two 8x32 S-boxes.

5. Results

5.1 Results for the Virtex family

The results of implementing AES candidates, according to the assumptions summarized in section 4, using the largest currently available Xilinx Virtex device, XCV1000BG560-6, are summarized in Fig. 10. For comparison, the results of implementing the current NIST standard, Triple DES, are also provided It should be stressed that all results come either from simulation or from reports generated by Xilinx tools, and have not as yet been confirmed experimentally. The details of all implementations, including the detailed block diagrams, and the description of simulation and test experiments will be provided in the technical report available at the AES conference [CG00]. Part of this report, describing Twofish, is already available on the web [CG99].

Implementations of all ciphers take from 9% (for Twofish) to 38% (for Serpent) of the total number of 12288 CLB slices available in the Virtex device used in our designs. It means that less expensive Virtex devices could be used for all implementations. Additionally, the key scheduling unit can be easily implemented within the same device as the encryption/decryption unit.

5.2 Results for the XC4000 family

For the low-cost, medium-size family of Xilinx FPGA devices, XC4000, only two ciphers, Twofish and RC6, were able to fit within the largest device from this family. The relative performance of these ciphers is similar to the relative performance in Virtex implementations. It is interesting to notice that for the two different FPGA devices from this family, the smaller one guarantees the higher speed.

Cipher	**Speed [Mbit/s]**		**Area [CLBs]**		**Speed/Area [kbit/s·CLB]**	
	4028/4036	4085	4028/4036	4085	4028/4036	4085
Twofish	90.9	89.2	907	907	100.2	98.3
RC6	45.9	43.1	1222	1222	37.6	35.3

Table I. Results of implementing Twofish and RC6 using the largest available FPGA device from the XC4000XL family, XC4085XL, and the largest device fitting the implementation of the respective cipher, i.e., XC4028XL for Twofish, and XC4036XL for RC6.

5.3 Resource sharing between encryption and decryption

The amount of resource sharing between encryption and decryption is considerably different for various AES candidates, depending on the type of the cipher. Resource sharing is close to 100% for Feistel ciphers and modified Feistel ciphers, and close to zero for S-P networks. The level of resource sharing can be described by the amount and type of the extra logic that must be added to the circuit implementing encryption, so that the modified circuit can perform both encryption and decryption, as shown in Table II.

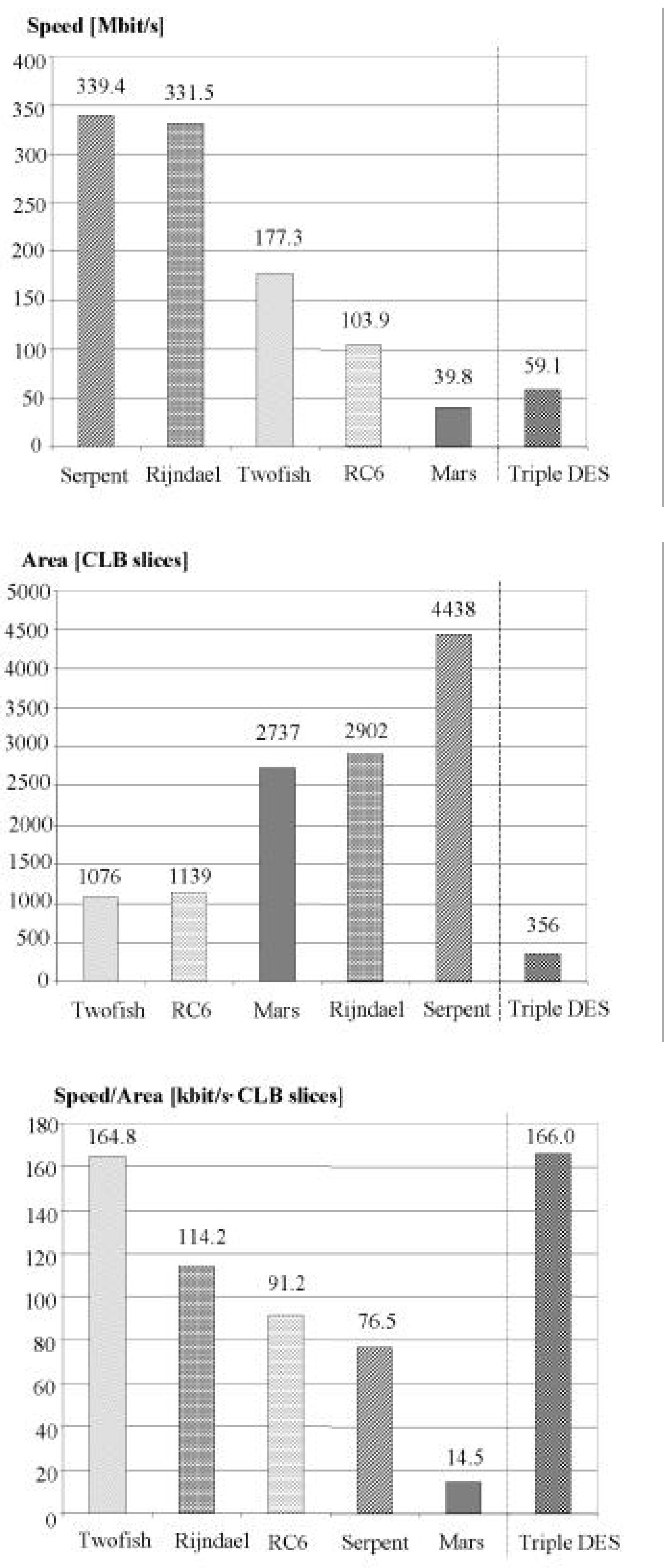

Fig. 10 Results of implementing AES candidates using Xilinx Virtex FPGA devices.

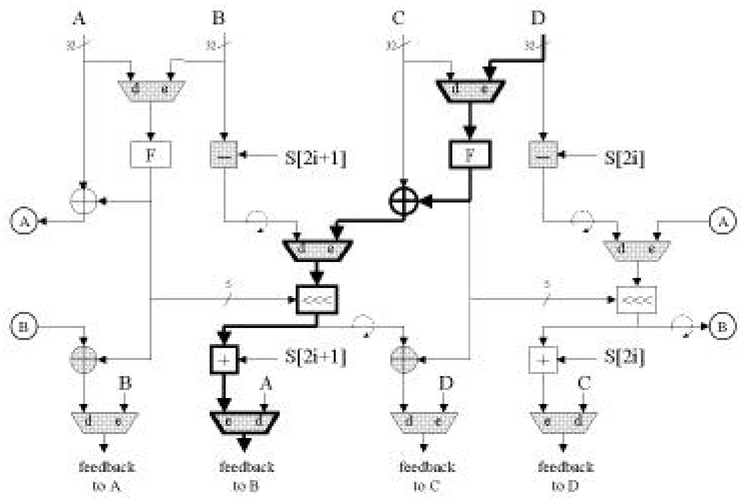

Fig. 11 Combinational part of a single round of RC6 implemented using basic architecture. Shaded components had to be added to the encryption unit, so it could perform decryption. The thick line shows the critical path in the circuit. Unit F performs operation $(2(X^2 \bmod 2^{32}) + X) \bmod 2^{32} <<< 5$. An arrow around a line means inverting the order of bits.

The relative size of the extra circuitry is the smallest for Mars and Twofish (less than 10%), and about 20% for RC6 (see Fig. 11). For Serpent and Rijndael, encryption and decryption are performed by two independent units of equal size. For Rijndael, these two units share 16 look-up tables implementing inversions in the Galois Field GF(2^8). These look-up tables take about 45% of the area used for encryption. Thus, the extra decryption circuitry takes for Serpent 100%, and for Rijndael about 55% of the area required for encryption itself.

Cipher	Extra logic	Extra logic area /encryption logic area
Twofish	2 32-bit XOR2, 2 32-bit MUX2	6%
Mars	2 SUB32, 3 32-bit MUX2	3%
RC6	2 SUB32, 2 32-bit XOR2, 8 32-bit MUX2 (see Fig. 11)	20%
Rijndael	Decryption independent of encryption, except 16 S-boxes 8x8	55%
Serpent	Decryption independent of encryption	100%

Table II. Extra logic that must be added to the circuit implementing encryption, so that the modified circuit can perform both encryption and decryption. Notation: XOR2 - 2-input XOR, MUX2 - 2-input multiplexer, SUB32 - 32-bit subtractor.

5.4 Critical path

The critical paths of all five AES candidates are characterized in Table III. As an example, the critical path of RC6 (without init MUX) is shown in Fig. 11.

Based on the characteristics of the critical path, the AES candidates can be divided into two main categories. Ciphers from the first category, RC6 and Mars, include in the critical path one complex arithmetic operation, such as modular multiplication or modular squaring, which determines the minimum clock period of these ciphers. The second category includes Rijndael, Twofish, and Serpent. In these ciphers, the critical path includes one or several S-boxes, and several multiple-input XORs. The minimum clock period is the sum of the access time to memories used to implement S-boxes, and delays introduced by multiple-input XORs and other simple auxiliary operations. The critical path of Twofish contains additionally two 32-bit additions.

The effect of resource sharing between encryption and decryption on the critical path is the strongest for RC6 (three encryption/decryption multiplexers in the critical path), very small for Rijndael, Twofish and Mars (one encryption/decryption multiplexer in the critical path), and negligible for Serpent. In Mars, additional delay (2 multiplexers) is caused by sharing resources between the forward and backwards keyed transformations.

Cipher	Minimum clock period - Virtex [ns]	Minimum clock period - XC4000 [ns]	Number of rounds	Components in the critical path (path flow / list of operations)
Rijndael	38.6	-	10	E/D MUX → S-box → affine transformation → MixColumn → init MUX
				S-box 8x8, XOR6, XOR5, XOR4, XOR2, 2 MUX2
Twofish	45.1	88.0	16	S-box → MDS → PHT → key addition → xor → E/D MUX → init MUX
				6 S-box 4x4, 2 ADD32, 9 XOR2, XOR4, XOR5, 2 MUX2
Serpent	94.3	-	4	8 x {key mixing → S-box → linear transformation) → init MUX
				8 S-box 4x4, 8 XOR2, 8 XOR7, MUX2
RC6	61.6	139.5	20	E/D MUX → squaring → addition → xor → E/D MUX → variable rotation → addition → E/D MUX → init MUX
				SQR32, 2 ADD32, ROT32, XOR2, 4 MUX2
Mars	100.6	-	32	2 mode MUXes → E/D MUX → multiplication → XOR → init MUX
				MUL32, XOR2, 4 MUX2

Table III. Critical paths in the implementation of the basic architecture for all AES candidates. Notation:
E/D MUX - encryption/decryption multiplexer, i.e., multiplexer used to change the data flow between encryption and decryption; mode MUX - multiplexer used to change the data flow depending on the mode of transformation (e.g., forward and backwards transformation in Mars); init MUX - multiplexer used to select between loading a new block of data and feeding back data from the end of the cipher round (the only multiplexer shown in Fig. 3a); XOR*n* - *n*-input XOR, MUX2 - 2-input multiplexer, ADD32 - 32-bit adder, MUL32 - 32-bit multiplier mod 2^{32}, SQR32 - 32-bit squaring mod 2^{32}, ROT32 - variable rotation of a 32-bit word.

5.5 Area critical components

The components contributing most to the circuit area, for each AES candidate, are shown in Table IV. The ciphers fall clearly into two groups: Twofish and RC6 have the area approximately three to four times smaller than the area of the remaining three candidates, Mars, Rijndael, and Serpent. The relatively small area of Twofish and RC6 comes from the fact that both ciphers are of the Feistel type. The relatively large size of Serpent and Rijndael comes from the fact that both ciphers are S-P networks, and the amount of resource sharing between encryption and decryption is limited (no resource sharing for Serpent, about 45% resource sharing for Rijndael). Additional factor contributing to the large size of Serpent is the use of eight different types of S-boxes in eight subsequent cipher rounds.

Cipher	# of CLB slices - Virtex	# of CLBs - XC4000	Area critical components
Twofish	1076	907	96 S-box 4x4 (6 kbit), 18 32-bit XOR2, 24 MUL GF(2^8)
RC6	1139	1222	2 SQR32, 12 32-bit MUX2, 2 ROT32
Serpent	4438	-	512 S-box 4x4 (32 kbit), 2048 XORn (linear transformation, n=2..7)
Mars	2737	-	4 S-box 8x32 (32 kbit), MUL32, 22 32-bit MUX2
Rijndael	2902	-	16 S-box 8x8 (32 kbit), 24 MUL GF(2^8), 256 XOR5 (affine and inverse affine transformation)

Table IV. Cipher components contributing most to the circuit area. Notation: MUL GF(2^8) - multiplication in the Galois Field GF(2^8), XOR*n* - *n*-input XOR, MUX2 - 2-input multiplexer, MUL32 - 32-bit multiplier mod 2^{32}, SQR32 - 32-bit squaring mod 2^{32}, ROT32 - variable rotation of a 32-bit word.

The relatively large size of Mars is the result of the design decisions, such as

a. using two different kinds of rounds (mixing vs. keyed transformation). For the basic non-pipelined architecture, only one type of round is active at a time.
b. using 4 large S-boxes 8x32 in a single round of the mixing transformation. Sharing two of these S-boxes during mixing transformation is possible only at the cost of doubling the number of clock cycles required for this transformation. (Our implementation still shares two S-boxes between the mixing transformation and the keyed transformation.)
c. using area-consuming 32x32 bit modular multiplication.

The area of Mars, Serpent, and Rijndael is dominated by S-boxes. Even though the number and size of these S-boxes is very different for each cipher, the total number of bits in memories implementing S-boxes, 32 kbits, is identical for all three ciphers. This may explain the relatively similar size of all three implementations expressed in number of CLBs.

5.6 Potential for inner-round pipelinig

Inner round pipelining can be most effectively applied to the ciphers with the following features:

a. the cipher round is composed of a large number of layers, with all layers performing simple operations with comparable delays;
b. the cipher round does not contain large hard-to-divide functional units.

Additionally, for FPGA implementations, it is advantageous if the implementation of the basic architecture contains large number of CLBs with unused flip-flops (one bit registers).

The above conditions are the best fulfilled by Serpent. It is straightforward to introduce 8 internal pipeline stages to the implementation round of Serpent (one implementation round = 8 regular cipher rounds), one after each regular cipher round. Implementing pipeline stages inside of the regular cipher round is possible in theory, but may be difficult in practice because of the clock frequency limitations imposed by the control unit.

The second cipher best suited for inner-round pipelining is Twofish. According to Table III, the critical path of Twofish contains a large number of simple operations with comparable delays, including a 4x4 S-box read-out, XOR operations, and additions. The most complex of these operations is a 32-bit addition. It is likely that this operation may need to be implemented using multilevel carry-lookahead architecture to take the full advantage of the inner-round pipelining in Twofish. Additionally, the FPGA implementation of basic architecture of Twofish contains a relatively small number of unused flip-flops, which will cause that the circuit area will increase by a larger percentage than for Serpent with the same number of inner-round pipeline stages.

Rijndeal is relatively easy to pipeline, but its critical path contains only 7 elementary operations. Additionally, the most time-consuming of these operations, the 8x8 S-box read-out, is hard to divide into extra pipeline stages. RC6 can be efficiently pipelined at the cost of increase in the circuit area resulting from using fast architectures for addition and multiplication (e.g., carry lookahead and carry save). Mars is the most difficult to pipeline because of the

a. irregular structure with different operations in various paths;
b. two types of rounds (mixing and keyed transformation) both using large S boxes;
c. need for the complex fast architectures for the pipelined multiplication and addition.

5.7 Potential for loop unrolling

The largest gain from loop unrolling can be achieved by ciphers with the following properties:

* small area used by the combinational part of a single round, which permits fitting a large amount of rounds in the largest available FPGA device;
* small delay of a single round compared to the sum of delays eliminated by loop unrolling, including the round multiplexer delay, the register delay, and the register setup time (as shown in formula (2)).
* potential for optimizations at the boundary between the last and the first operation of the cipher round.

Assuming the use of the largest available Virtex chip, RC6 and Twofish have the highest potential for loop unrolling. The largest Virtex chip can easily fit ten RC6 rounds and eight Twofish rounds. Mars can be implemented with four rounds unrolled; Rijndael and Serpent with only two rounds unrolled.

5.8 Potential for outer-round pipelining and mixed outer-inner-round pipelining

The largest gain from outer-round pipelining can be achieved by ciphers with the smallest area. The largest number of pipelined rounds fitting within the largest available Virtex chip is the same as in the architecture with loop unrolling. As a result, Twofish and RC6 can benefit most from the outer-round pipelined architecture. The throughput of both these ciphers exceeds 1 Gbit/s for the architectures with the maximum number of outer-round pipeline stages. Additional speed-up can be obtained by combining outer and inner round pipelining, leading to the mulitigigabit-per-second performance. For Serpent, the most straightforward form of mixed pipelining, with 16 regular cipher rounds unrolled and a register after each regular cipher round (1/8 of the implementation round), would result in an even higher performance. Mars can benefit substantially from both forms of pipelining; Rijndael primarily from the inner-round pipelining.

6. Design procedure and tools

The design flow and tools used in our group for implementation of algorithms in FPGA devices are shown in Fig. 12. All five AES ciphers were first described in VHDL, and their description verified using the functional simulator from Aldec, Inc. Test vectors and intermediate results from the reference software implementations were used for debugging and verification of VHDL codes. The revised VHDL code became an input to Xilinx tools performing the automated logic synthesis, mapping, placing, and routing. These tools generated reports describing the area and speed of implementations, a netlist used for timing simulations, and a bitstream to be used to program an actual FPGA device. A final step is to verify the design experimentally, using physical FPGA devices. We plan to perform these experiments using a PCI FPGA board from Virtual Computer Corporation [VCC]. The most complex PCI board currently available from VCC is based on the XC4062XL FPGA device. This device is able to fit full implementations of Twofish and RC6, and an encryption portion of Serpent. All details of our implementations and experiments will be described in the technical report [CG00].

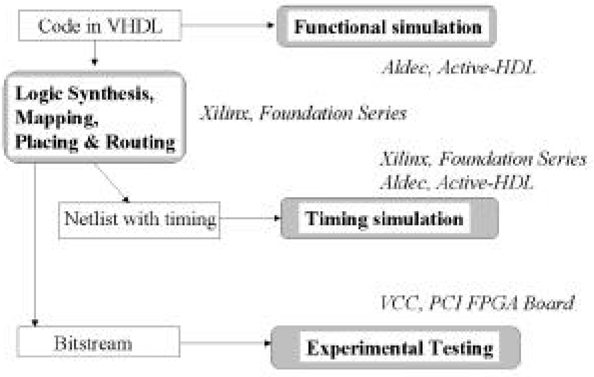

Fig. 12 Design flow for implementing AES candidates using Xilinx FPGA devices.

7. Need for interleaved operating modes

The full potential of hardware implementations of symmetric block ciphers can only be utilized in cipher modes that support efficient use of pipelining, as shown in Fig. 8. To date, the ECB mode is the only operating mode standardized by NIST that supports efficient pipelining. However, ECB is not regarded secure for transmissions of large volumes of data, and most standard protocols recommend using CBC or CFB modes instead. Therefore, we propose to speed-up the standardization effort, and include in the AES standard interleaved modes of operation, such as the interleaved CBC mode defined by:

$$C_i = \text{AES}(M_i \oplus IV_i) \text{ for } i=1 \text{ to } N, \text{ and } C_i = \text{AES}(M_i \oplus C_{i-N}) \text{ for } i>N \,. \quad (4)$$

The standard should support arbitrary values of the interleaving factor N, smaller than a certain maximum.

8. Conclusions

The results and analyses presented in this paper show that the differences in hardware performance of the AES candidates are bigger and more significant than the corresponding differences in software performance. No correlation between software and hardware performance was found. On the contrary, Serpent, believed to be the slowest candidate in software, appeared to be the fastest of the five AES candidates in hardware. We believe that the large differences among parameters of all five AES algorithms in hardware resulted primarily from internal structure of these algorithms, and were not significantly affected by our implementation decisions. On the other hand, we could not completely eliminate or predict the influence of the FPGA design tools and the VHDL design entry method on the results of the comparison. Assessed exclusively from the hardware performance point of view, the five AES finalists fall into the three distinct classes with different performance characteristics.

The first class includes Twofish and RC6. Both ciphers guarantee compact low-cost implementations with medium speed compared to other candidates. In particular, because of the area constraints, Twofish and RC6 are the only ciphers that can be implemented using low cost FPGA devices from the Xilinx XC4000 family. Both ciphers can be substantially sped-up by outer-round pipelining (for non-feedback modes (ECB, counter mode)), and - to the lesser extent - by loop-unrolling (for cipher feedback modes (CBC, CFB)). Among the two, Twofish is in some respects superior to RC6. It is about 70% faster and is more suitable for inner-round pipelining. Both ciphers use comparable area, and as a result their potential for loop unrolling and outer-round pipelining is similar.

The second class includes Serpent and Rijndael. Both ciphers guarantee very high speed at the cost of the relatively large area compared to the ciphers from the first class. The primary way of speeding up these ciphers for non-feedback cipher modes (ECB and counter mode) is inner-round pipelining. Both ciphers have a similar speed in the basic architecture. Rijndael can be implemented using about 35% less area. The more regular architecture of Serpent makes it significantly more suitable for a multi-stage inner-round pipelining.

The third class is composed of Mars itself. This cipher shows the worst hardware characteristics of all five candidates. It is over twice as slow than the next slowest candidate (RC6), and over 8 times slower than the fastest AES cipher (Serpent). It also takes over twice the area used by ciphers from the first group, Twofish and RC6. Further optimizations of the Mars implementation are certainly possible, but would require the higher development effort than that devoted to other AES candidates.

It is interesting to notice that although four out of five candidates outperform Triple DES in terms of speed, only Twofish has a comparable performance in terms of the speed/area ratio. Three other candidates, Rijndael, RC6, and Serpent, have a similar, and much lower than triple DES, value of this performance parameter.

Out of all five candidates, Twofish seems to be the most suitable for applications where the primary requirement is the limited cost or area of the cryptographic hardware. Serpent and Rijndael both offer superior performance for applications where the speed itself is a criterion of primary concern.

Acknowledgments
The authors would like to thank Christof Paar and his students, as well as Miles Smid and other members of the NIST Computer Security Division for valuable comments and discussions. We also would like to thank our students, Po Khuon and Tanvir Joy, for their work on implementation of Triple DES.

Literature:

[BCD+98] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. M. Matyas, L. O'Connor, M. Peyravian, D. Safford, and N. Zunic, "Mars - A Candidate Cipher for AES," NIST AES Proposal, June 1998.
[CG99] P. Chodowiec and K. Gaj, "Implementation of the Twofish Cipher Using FPGA Devices", Technical Report, George Mason University, July 1999; available at http://www.counterpane.com/twofish.html.
[CG00] P. Chodowiec and K. Gaj, "Implementations of the AES Candidate Algorithms using FPGA Devices," Technical Report, George Mason University, April 2000 (to be published on the web).
[EP99] A.J. Elbirt and C. Paar, "An FPGA Implementation and Performance Evaluation of the Serpent Block Cipher," Eighth ACM International Symposium on Field-Programmable Gate Arrays, Monterey, California, February 10-11, 2000. Preprint available at http://ece.wpi.edu/Research/crypt/publications/index.html.
[NBD+99] James Nechvatal, Elaine Barker, Donna Dodson, Morris Dworkin, James Foti, Edward Roback, "Status Report on the First Round of the Development of the Advanced Encryption Standard," NIST report, August 1999.
[NSA98] National Security Agency, "Initial plans for estimating the hardware performance of AES submissions," http://csrc.nist.gov/encryption/aes/round2/round2.htm.
[RH99] M. Riaz and H. Heys, "The FPGA Implementation of RC6 and CAST-256 Encryption Algorithms," accepted for CCECE'99, Edmonton, Alberta, Canada, 1999.
[RRS+98] R. Rivest, M. Robshaw, R. Sidney, and Y. L. Yin, "The RC6 Block Cipher," NIST AES Proposal, June 1998.
[SKW+98] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, "Twofish: A 128-Bit Block Cipher," NIST AES Proposal, June 1998.
[SKW+99] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, "Performance Comparison of the AES Submissions," Second AES Candidate Conference, Rome, April 1999.
[VCC] Virtual Computer Corporation, http://www.vcc.com/