An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists

AJ Elbirt[1], W Yip[1], B Chetwynd[2], C Paar[1]
ECE Department, Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609, USA
[1] Email: {aelbirt, waihyip, christof}@ece.wpi.edu
[2] Email: spunge@alum.wpi.edu

To Appear in the IEEE Transactions on VLSI

Abstract—
The technical analysis used in determining which of the potential Advanced Encryption Standard candidates was selected as the Advanced Encryption Algorithm includes efficiency testing of both hardware and software implementations of candidate algorithms. Reprogrammable devices such as Field Programmable Gate Arrays (FPGAs) are highly attractive options for hardware implementations of encryption algorithms as they provide cryptographic algorithm agility, physical security, and potentially much higher performance than software solutions. This contribution investigates the significance of FPGA implementations of the Advanced Encryption Standard candidate algorithms. Multiple architectural implementation options are explored for each algorithm. A strong focus is placed on high throughput implementations, which are required to support security for current and future high bandwidth applications. Finally, the implementations of each algorithm will be compared in an effort to determine the most suitable candidate for hardware implementation within commercially available FPGAs.

Keywords: cryptography, algorithm-agility, FPGA, block cipher, VHDL

I. Introduction

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 (AES), specifying an Advanced Encryption Algorithm to replace the Data Encryption Standard (DES) which expired in 1998 [1]. NIST has solicited candidate algorithms for inclusion in AES, resulting in fifteen official candidate algorithms of which five have been selected as finalists. Unlike DES, which was designed specifically for hardware implementations, one of the design criteria for AES candidate algorithms is that they can be efficiently implemented in both hardware and software. Thus, NIST has announced that both hardware and software performance measurements will be included in their efficiency testing. However, prior to the third AES conference in April 2000, virtually all performance comparisons have been restricted to software implementations on various platforms [2]. In October 2000, NIST chose Rijndael as the Advanced Encryption Algorithm.

The advantages of a software implementation include ease of use, ease of upgrade, portability, and flexibility. However, a software implementation offers only limited physical security, especially with respect to key storage [1] [3]. Conversely, cryptographic algorithms (and their associated keys) that are implemented in hardware are, by nature, more physically secure as they cannot easily be read or modified by an outside attacker [3]. The down side of traditional (ASIC) hardware implementations are the lack of flexibility with respect to algorithm and parameter switching. Reconfigurable hardware devices such as Field Programmable Gate Arrays (FPGAs) are a promising alternative for the implementation of block ciphers. FPGAs are hardware devices whose function is not fixed and which can be programmed in-system. The potential advantages of encryption algorithms implemented in FPGAs include:

Algorithm Agility This term refers to the switching of cryptographic algorithms during operation. The majority of modern security protocols, such as SSL or IPsec, allow for multiple encryption algorithms. The encryption algorithm is negotiated on a per-session basis; e.g., IPsec allows among others DES, 3DES, Blowfish, CAST, IDEA, RC4 and RC6 as algorithms, and future extensions are possible. Whereas algorithm agility can be very costly with traditional hardware, FPGAs can be reprogrammed on-the-fly. While reconfiguration time is still an open issue to be solved, algorithm agility through FPGAs appears to be an attractive possibility [4] [5].

Algorithm Upload It is perceivable that fielded devices are upgraded with a new encryption algorithm which did not exist (or was not standardized) at design time. In particular, it is very attractive for numerous security products to be upgraded for use of AES once the selection process is over. Assuming there is some kind of (temporary) connection to a network such as the Internet, FPGA-equipped encryption devices can upload the new configuration code.

Algorithm Modification There are applications which require modification of a standardized algorithm, e.g., by using proprietary S-boxes or permutations. Such modifications are easily made with reconfigurable hard-

ware. Similarly, a standardized algorithm can be swapped with a proprietary one. Also, modes of operation can be easily changed.

Architecture Efficiency In certain cases, a hardware architecture can be much more more efficient if it is designed for a specific set of parameters; e.g., constant multiplication (of integers or in Galois fields) is far more efficient than general multiplication. With FPGAs it is possible to design and optimize an architecture for a specific parameter set.

Throughput Although typically slower than an ASIC implementations, FPGA implementations have the potential of running substantially faster then software implementations.

Cost Efficiency The time and costs for developing an FPGA implementation of a given algorithm are much lower than for an ASIC implementation. (However, for high-volume applications, ASIC solutions usually become the more cost-efficient choice.)

For these reasons, this paper describes a thorough comparison of four of the AES finalist algorithms with respect to implementation on state-of-the-art FPGAs. One aspect that seems to be especially relevant is the investigation of achievable encryption rates for FPGA-based implementations. We demonstrate that FPGA solutions encrypt at rates in the Gigabit range for all four algorithms investigated, which is at least one order of magnitude faster than most reported software implementations [6].

What follows is an investigation of four of the five AES finalists to determine the nature of their underlying components. Due to resource limitations, the MARS algorithm was beyond the scope of this investigation. The characterization of the algorithms' components will lead to a discussion of the hardware architectures best suited for implementation of the AES finalists. A performance metric to measure the hardware cost for the throughput achieved by each algorithm's implementations will be developed and a target FPGA will be chosen so as to yield implementations that are optimized for high-throughput operation within the commercially available device. Finally, multiple architecture options of the algorithms within the targeted FPGA will be discussed and the overall performance of the implementations will be evaluated versus typical software implementations.

II. Previous Work

As opposed to custom hardware or software implementations, little work exists in the area of block cipher implementations within existing FPGAs. DES, the most common block cipher implementation targeted to FPGAs, has been shown to operate at speeds of up to 400 Mbit/s [7]. We believe that this performance can be greatly enhanced using today's technology. These speeds are significantly faster than the best software implementations of DES [8] [9] [10], which typically have throughputs below 100 Mbit/s, although implementations operating in the 130 Mbit/s range have been reported as well [8] [11]. This performance differential is an expected result of DES having been designed in the 1970s with hardware implementations in mind.

Other block ciphers have been implemented in FPGAs with varying degrees of success. A typical example is the IDEA block cipher which has been implemented at speeds ranging from 2.8 Mbit/s [12] to 528 Mbit/s [13]. Note that while the 528 Mbit/s throughput was achieved in a fully pipelined architecture, the implementation required four Xilinx XC4000 FPGAs.

Some FPGA implementation throughputs for the AES candidates have been shown to be far slower than their software counterparts. Hardware throughputs of about 12 Mbit/s [14] [15] have been achieved for CAST-256. However, software implementations have resulted in throughputs of 37.8 Mbit/s for CAST-256 on a 200 MHz PentiumPro PC [6], a factor of three faster than FPGA implementations. When scaled to a more current 600 MHz PentiumPro PC, it is expected that the same software implementation would outperform FPGA implementations by an even larger factor. While an FPGA implementation of RC6 has been shown to operate at data rates of 37.8 Mbit/s [15], our findings indicate that considerably higher data rates are achievable. Additional FPGA implementation studies of the AES finalists may be found in [16] [17] [18] [19].

When examining the AES finalists, it is important to note that they do not necessarily exhibit similar behavior to DES when comparing hardware and software implementations. One reason for this is that the AES finalists have been designed with efficient software implementations in mind. Additionally, software implementations may be executed on processors operating at frequencies as high as 800 MHz while typical implementations that target FPGAs reach a maximum clock frequency of 50 MHz.

III. Methodology

A. Design Methodology

There are two basic hardware design methodologies currently available: language based (high level) design and schematic based (low level) design. Language based design relies upon synthesis tools to implement the desired hardware. While synthesis tools continue to improve, they rarely achieve the most optimized implementation in terms of both area and speed when compared to a schematic implementation. As a result, synthesized designs tend to be (slightly) larger and slower than their schematic based counterparts. Additionally, implementation results can greatly vary depending on the synthesis tool as well as the design being synthesized, leading to potentially increased variances in the synthesized results when comparing synthesis tool outputs. This situation is not entirely different from a software implementation of an algorithm in a high-level language such as C, which is also dependent on coding style and compiler quality. As shown in [20], schematic based design methodologies are no longer feasible for supporting the increase in architectural complexity evidenced by modern FPGAs. As a result, a language based design methodology was chosen as the implementation form for

the AES finalists with VHDL being the specific language chosen.

B. Implementations — General Considerations

Each AES finalist was implemented in VHDL which led to the use of a bottom-up design and test methodology [20]. The same hardware interface was used for each of the implementations. In an effort to achieve the maximum efficiency possible, note that key scheduling and decryption were not implemented for each of the AES finalists. Because FPGAs may be reconfigured in-system, the FPGA may be configured for key scheduling and then later reconfigured for either encryption or decryption. This option is a major advantage of FPGAs implementations over classical ASIC implementations. Note that should the overall system include an external processor to control FPGA reconfiguration, this processor could be used instead of the FPGA to perform the key scheduling. Round keys for encryption are loaded from the external key bus and are stored in internal registers and all keys must be loaded before encryption may begin. Key loading is disabled until encryption is completed. Each implementation was simulated for functional correctness using the test vectors provided in the AES submission package [21] [22] [23] [24]. After verifying the functionality of the implementations, the VHDL code was synthesized, placed and routed, and re-simulated with annotated timing using the same test vectors, verifying that the implementations were successful.

C. Selection of a Target FPGA

When examining the AES finalists for hardware implementation within an FPGA, a number of key aspects emerge. First, it is obvious that the implementation will require a large amount of I/O pins to fully support the 128-bit data stream at high speeds where bus multiplexing is not an option. It is desirable to decouple the 128-bit input and output data streams to allow for a fully pipelined architecture. Since the round keys cannot change during the encryption process, they may be loaded via a separate key input bus prior to the start of encryption. Additionally, to implement a fully pipelined architecture requires 128-bit wide pipeline stages, resulting in the need for a register-rich architecture to achieve a fast, synchronous implementation. Moreover, it is desirable to have as many register bits as possible per each of the FPGA's configurable units to allow for a regular layout of design elements as well as to minimize the routing required between configurable units. Finally, it is critical that fast carry-chaining be provided between the FPGA's configurable units to maximize the performance of AES finalists that utilize arithmetic operations [15] [14], a feature commonly available in commercial FPGAs at no additional cost.

In addition to architectural requirements, scalability and cost must also be considered. We believe that the chosen FPGA should be the most high-end chip available, capable of providing the largest amount of hardware resources as well as being highly flexible so as to yield optimal performance. Unfortunately, the cost associated with current high-end FPGAs is relatively high (several hundred US dollars per device). However, it is important to note that the FPGA market has historically evolved at an extremely rapid pace, with larger and faster devices being released to industry at a constant rate. This evolution has resulted in FPGA cost-curves that decrease sharply over relatively short periods of time. Hence, selecting a high-end device provides the closest model for the typical FPGA that will be available over the lifespan of AES.

Based on the aforementioned considerations, the Xilinx Virtex XCV1000BG560-4 FPGA was chosen as the target device. The XCV1000 has 128K bits of embedded RAM divided among thirty-two RAM blocks that are separate from the main body of the FPGA. The 560-pin ball grid array package provides 512 usable I/O pins. The XCV1000 is comprised of a 64×96 array of look-up-table based Configurable Logic Blocks (CLBs), each of which acts as a 4-bit element comprised of two 2-bit slices for a total of 12288 CLB slices. Additionally, the XCV1000 may be configured to provide a maximum of 384K bits of RAM independant of the embedded RAM [25]. This type of configuration results in a highly flexible architecture that will accommodate the round functions' use of wide operand functions. Note that the XCV1000 also appears to be a good representative for a modern FPGA and that devices from other vendors are not fundamentally different. It is thus hoped that our results carry over, within limits, to other devices.

D. Design Tools

FPGA Express by Synopsys, Inc. and Synplify by Synplicity, Inc. were used to synthesize the VHDL implementations of the AES finalists. As this study places a strong focus on high throughput implementations, the synthesis tools were set to optimize for speed. As will be discussed in Section VI, the resultant implementations exhibit the best possible throughputs with the associated cost being an increase in the area required in the FPGA for each of the implementations. Similarly, if the synthesis tools were set to optimize for area, the resultant implementations would exhibit reduced area requirements at the cost of decreased throughput. However, this theory does not always hold true for certain algorithms and architectures. This contradiction is caused by the underlying proprietary synthesis tool algorithms — different synthesis algorithms tend to yield different implementations for the same VHDL code. As will be evidenced in later discussions, these synthesis algorithms may lead to unexpected results based on how a design implementation is interpreted.

XACTstep 2.1i by Xilinx, Inc. was used to place and route the synthesized implementations. For the sub-pipelined architectures, a 40 MHz timing constraint was used in both the synthesis and place-and-route processes as it resulted in significantly higher system clock frequencies. However, the 40 MHz timing constraint was found to have little affect on the other architecture types, resulting in nearly identical system clock frequencies to those achieved without the timing constraint.

Finally, Speedwave by Viewlogic Systems, Inc. and

Active-HDL^{TM} by ALDEC, Inc. were used to perform behavioral and timing simulations for the implementations of the AES finalists. The simulations verified both the functionality and the ability to operate at the designated clock frequencies for the implementations.

E. Evaluation Metrics

When evaluating a given implementation, the throughput of the implementation and the hardware resources required to achieve this throughput are usually considered the most critical parameters. Other aspects, such as power consumption, are not addressed in this study. No established metric exists to measure the hardware resource costs associated with the measured throughput of an FPGA implementation. To address this issue, the Xilinx XCV1000 FPGA must be examined. Two area measurements are readily apparent — logic gates and CLB slices. It is important to note that the logic gate count does not yield a true measure of how much of the FPGA is actually being used. Hardware resources within CLB slices may not be fully utilized by the place-and-route software so as to relieve routing congestion. This results in an increase in the number of CLB slices without a corresponding increase in logic gates. To achieve a more accurate measure of chip utilization, CLB slice count was chosen as the most reliable area measurement. Therefore, to measure the hardware resource cost associated with an implementation's resultant throughput, the Throughput Per Slice (TPS) metric is used. We defined TPS as:

$$TPS \quad := \quad \text{(Encryption Rate)/(\# CLB Slices Used)}$$

When appropriate, the TPS metric will be normalized. Therefore, the optimal implementation will display the highest throughput and have the largest TPS. Note that the TPS metric behaves inversely to the classical time-area (TA) product.

When comparing implementations using the TPS and throughput metrics, it is required that the architectures are implemented on the same FPGA. Different FPGAs within the same family yield different timing results as a function of available logic and routing resources, both of which change based on the die size of the FPGA. Additionally, it is impossible to legitimately compare FPGAs from separate families (such as comparing an implementation on a Xilinx Virtex FPGA with an implementation on a Xilinx XC4000 FPGA) as each family of FPGAs has a unique architecture. Quite obviously, different FPGA architectures will greatly affect the TPS and throughput measurements — each architecture may be comprised of different logic elements and have different amounts of logic and routing resources. Therefore, all of the implementations were performed on the Xilinx Virtex XCV1000BG560-4 to guarantee consistency.

Finally, it is critical to note that TPS and throughput may not scale linearly based on the number of rounds implemented for the three architecture types detailed in Section IV-A. Therefore, it is imperative to examine multiple implementations for each architecture type, varying the round count to determine the optimal number of rounds per implementation. As will be shown in Section V, TPS and throughput do not scale linearly for certain algorithms when implemented using some or all of the three architecture types.

The number of CLBs required as well as the maximum operating frequency for each implementation was obtained from the Xilinx place-and-route report files. The computed throughput for implementations that employ any form of hardware pipelining (to be discussed in Section IV) are made assuming that the pipeline latency has been met.

IV. Architecture Options and the AES Finalists

Before attempting to implement the AES finalists in hardware, it is important to understand the nature of each algorithm as well as the hardware architectures most suited for their implementation. What follows is an investigation into the key components of the AES finalists. Based on this breakdown, a discussion is presented on the hardware architectures most suited for implementation of the AES finalists.

A. Core Operations of the AES Finalist Algorithms

Algorithm	**XOR**	**Mod 2^{32} Add**	**Mod 2^{32} Sub**	**Fixed Shift**
MARS	•	•	•	•
RC6	•	•		•
Rijndael	•			•
Serpent	•			•
Twofish	•	•		•

Algorithm	**Var. Rot.**	**Mod 2^{32} Mult**	**GF(2^8) Mult**	**S-Box**
MARS	•	•		•
RC6	•	•		
Rijndael			•	•
Serpent				•
Twofish			•	•

TABLE I
AES finalists core operations [26]

Modern FPGAs have a structure comprised of a two-dimensional array of configurable function units interconnected via horizontal and vertical routing channels. Configurable function units are typically comprised of look-up-tables and flip-flops. Look-up-tables may be configured as either combinational logic or memory elements. Additionally, many modern FPGAs provide variable-size SRAM blocks that may be used as either memory elements or look-up-tables [27].

In terms of complexity, the operation detailed in Table I which requires the most hardware resources as well as computation time is the modulo 2^{32} multiplication [26]. Implementing wide multipliers in hardware is an inherently costly task that requires significant hardware resources. S-Boxes may be implemented in either combinatorial logic or embedded RAM — the advantages of each of these options are discussed in Section IV-B. Fast operations such as bit-wise XOR, modulo 2^{32} addition and subtraction, fixed value shifting, and variable rotations are constructed from simple hardware elements. Additionally, the Galois field multiplications required in Rijndael and Twofish can also be implemented very efficiently in hardware as they are multiplications by a constant. Galois field constant multiplication requires simple boolean equations as compared to general multiplications [28].

B. Hardware Architectures

The AES finalists are all comprised of a basic looping structure (some form of either Feistel or substitution-permutation network) whereby data is iteratively passed through a round function. Based on this looping structure, the following architecture options were investigated so as to yield optimized implementations:

- Iterative Looping
- Loop Unrolling
- Partial Pipelining
- Partial Pipelining with Sub-Pipelining

Iterative looping over a cipher's round structure is an effective method for minimizing the hardware required when implementing an iterative architecture. When only one round is implemented, an n-round cipher must iterate n times to perform an encryption. This approach has a low register-to-register delay but requires a large number of clock cycles to perform an encryption. This approach also minimizes in general the hardware required for round function implementation but can be costly with respect to the hardware required for round key and S-Box multiplexing. This cost may be alleviated by configuring the 4-bit look-up-tables within the CLB slice to operate in RAM mode and then storing one bit of each round key within the look-up-table. One look-up-table would support up to sixteen round keys, leading to an implementation requiring k look-up-tables to store round keys that are k bits in length, removing the need to store each round key in a separate register. Additionally, multiple banks of look-up-tables may be used to support algorithms that require more than sixteen round keys. These look-up-table banks would then be sequentially addressed based on the current round to access the appropriate key. However, this methodology is less efficient when multiple rounds are unrolled or pipelined as each round requires its own bank of associated look-up-tables for round key storage. This methodology becomes completely inefficient for a fully unrolled or fully pipelined architecture as each look-up-table bank would only contain a single round key. Because this methodology does not provide a performance enhancement for all architectures, its use was not considered for this study to maintain consistency between architectures.

Iterative looping is a subset of loop unrolling in that only one round is unrolled whereas a loop unrolling architecture allows for the unrolling of multiple rounds, up to the total number of rounds required by the cipher. As opposed to an iterative looping architecture, a loop unrolling architecture where all n rounds are unrolled and implemented as a single combinatorial logic block maximizes the hardware required for round function implementation while the hardware required for round key and S-Box multiplexing is completely eliminated. However, while this approach minimizes the number of clock cycles required to perform an encryption, it maximizes the worst case register-to-register delay for the system, resulting in an extremely slow system clock.

A partially pipelined architecture offers the advantage of high throughput rates by increasing the number of blocks of data that are being simultaneously operated upon. This is achieved by replicating the round function hardware and registering the intermediate data between rounds. Moreover, in the case of a full-length pipeline (a specific form of a partial pipeline), the system will output a 128-bit block of ciphertext at each clock cycle once the latency of the pipeline has been met. However, an architecture of this form requires significantly more hardware resources as compared to a loop unrolling architecture. In a partially pipelined architecture, each round is implemented as the pipeline's atomic unit and are separated by the registers that form the actual pipeline. However, many of the AES finalists cannot be implemented using a full-length pipeline due to the large size of their associated round function and S-Boxes, both of which must be replicated n times for an n-round cipher. As such, these algorithms must be implemented as partial pipelines. Additionally, a pipelined architecture can be fully exploited only in modes of operations which do not require feedback of the encrypted data, such as Electronic Code-Book or Counter Mode. When operating in feedback modes such as Ciphertext Feedback Mode, the ciphertext of one block must be available before the next block can be encrypted. As a result, multiple blocks of plaintext cannot be encrypted in a pipelined fashion when operating in feedback modes. For the remainder of our discussion, feedback mode will be abbreviated as FB and non-feedback mode will be abbreviated as NFB.

Sub-pipelining a (partially) pipelined architecture is advantageous when the round function of the pipelined architecture is complex, resulting in a large delay between pipeline stages. By adding sub-pipeline stages, the atomic function of each pipeline stage is sub-divided into smaller functional blocks. This results in a decrease in the pipeline's delay between stages. However, each sub-division of the atomic function increases the number of clock cycles required to perform an encryption by a factor equal to the number of sub-divisions. At the same time, the number of blocks of data that may be operated upon in NFB mode is increased by a factor equal to the number of sub-divisions. Therefore, for this technique to be effective, the worst case delay between stages will be decreased by a factor of m

where m is the number of added sub-divisions.

Many FPGAs provide embedded RAM which may be used to replace the round key and S-Box multiplexing hardware. By storing the keys within the RAM blocks, the appropriate key may be addressed based on the current round. However, due to the limited number of RAM blocks, as well as their restricted bit width, this methodology is not feasible for architectures with many pipeline stages or unrolled loops. Those architectures require more RAM blocks than are typically available. Additionally, the switching time for the RAM is more than a factor of three longer than that of a standard CLB slice element, resulting in the RAM element having a lesser speed-up effect on the overall implementation. Therefore, the use of embedded RAM is not considered for this study to maintain consistency between architectural implementations.

V. Architectural Implementation Analysis

For each of the AES finalists, the four architecture options described in Section IV-B were implemented in VHDL using a bottom-up design and test methodology. The same hardware interface was used for each of the implementations. Round keys are stored in internal registers and all keys must be loaded before encryption may begin. Key loading is disabled until encryption is completed. These implementations yielded a great deal of knowledge in regards to the FPGA suitability of each AES finalist. What follows is a discussion of the knowledge gained regarding each algorithm when implemented using the four architecture types.

A. Architectural Implementation Analysis — RC6

When implementing the RC6 algorithm, it was first determined that the RC6 modulo 2^{32} multiplication was the dominant element of the round function in terms of required logic resources. Each RC6 round requires two copies of the modulo 2^{32} multiplier. However, it was found that the RC6 round function does not require a general modulo 2^{32} multiplier. The RC6 multipliers implement the function $A(2A + 1)$ which may be implemented as $2A^2 + A$. Therefore, the multiplication operation was replaced with an array squarer with summed partial products, requiring fewer hardware resources and resulting in a faster implementation. The remaining components of the RC6 round function — fixed and variable shifting, bit-wise XOR, and modulo 2^{32} addition — were found to be simple in structure, resulting in these elements of the round function requiring few hardware resources. Finally, it was found that the synthesis tools could not minimize the overall size of an RC6 round sufficiently to allow for a fully unrolled or fully pipelined implementation of the entire twenty rounds of the algorithm within the target FPGA. The results for each of the RC6 implementations using the four architecture types is found in Appendix A.

The 2-stage partial pipeline architecture optimized for speed was found to yield both the highest throughput and TPS when operating in FB mode, outperforming the single round iterative looping implementation by achieving a significantly higher system clock frequency. When operating in NFB mode, a partially pipelined architecture optimized for speed with two additional sub-pipeline stages was found to offer the advantage of extremely high throughput rates and TPS once the latency of the pipeline was met, with the 10-stage partial pipeline implementation displaying the best throughput and TPS results. Based on the delay analysis of the partial pipeline implementations, it was determined that nearly two thirds of the round function's associated delay was attributed to the modulo 2^{32} multiplier. Therefore, two additional pipeline sub-stages were implemented so as to subdivide the multiplier into smaller blocks, resulting in a total of three pipeline stages per round function. Further sub-pipelining was not implemented as this would require sub-dividing the adders used to sum the partial products (a non-trivial task) to balance the delay between sub-pipeline stages.

B. Architectural Implementation Analysis — Rijndael

When implementing the Rijndael algorithm, it was first determined that the Rijndael S-Boxes were the dominant element of the round function in terms of required logic resources. Each Rijndael round requires sixteen copies of the S-Boxes, each of which is implemented as an 8-bit to 8-bit look-up-table, requiring significant hardware resources. However, the remaining components of the Rijndael round function — byte swapping, constant Galois field multiplication, and key addition — were found to be simple in structure, resulting in these elements of the round function requiring few hardware resources. Additionally, it was found that the synthesis tools could not minimize the overall size of a Rijndael round sufficiently to allow for a fully unrolled or fully pipelined implementation of the entire ten rounds of the algorithm within the target FPGA. The results for each of the Rijndael implementations using the four architecture types are found in Appendix A.

2-stage loop unrolling optimized for speed was found to yield the highest throughput while iterative looping optimized for speed was found to yield the highest TPS when operating in FB mode. However, the measured throughput for the iterative looping implementation was within 10% of the 2-stage loop unrolled implementation. Due to the probabilistic nature of the place-and-route algorithms, one can expect a variance in performance based on differences in the starting point of the process. When performing this process multiple times, known as multi-pass place-and-route, it is likely that the single round implementation would achieve a throughput similar to that of the 2-stage loop unrolled implementation. However, the use of multi-pass place-and-route was beyond the scope of this investigation and was therefore not performed.

When operating in NFB mode, partial pipelining was found to offer the advantage of extremely high throughput rates once the pipeline latency was met. 5-stage partial pipelining with one sub-pipeline stage optimized for speed was found to yield the best throughput results while 2-stage partial pipelining with one sub-pipeline stage optimized for speed was found to yield the best TPS results. However,

the measured TPS for the 5-stage partial pipeline with one sub-pipeline stage was within 10% of the 2-stage partial pipeline with one-subpipeline stage. As is the case in FB mode, it is likely that the 5-stage partial pipeline with one sub-pipeline stage would achieve a TPS similar to the 2-stage partial pipeline with one sub-pipeline stage should multi-pass place-and-route be performed. While Rijndael cannot be implemented using a fully pipelined architecture due to the large size of the round function, significant throughput increases were seen as compared to the loop unrolling architectures. Sub-pipelining of the partially pipelined architectures was implemented by inserting a pipeline sub-stage within the Rijndael round function. Based on the delay analysis of the partial pipeline implementations, it was determined that nearly half of the round function's associated delay was attributed to the S-Box substitutions. Therefore, the additional pipeline sub-stage was implemented so as to separate the S-Boxes from the rest of the round function. Further sub-pipelining was not implemented as this would require sub-dividing the S-Boxes (a non-trivial task) to balance the delay between sub-pipeline stages.

C. Architectural Implementation Analysis — Serpent

When implementing the Serpent algorithm, it was first determined that since the Serpent S-Boxes are relatively small (4-bit to 4-bit), it is possible to implement them using combinational logic as opposed to clocked memory elements. Additionally, the combinatorial logic that comprises the S-Boxes map extremely well to the Xilinx CLB slice, which is comprised of 4-bit look-up-tables. This allows one S-Box to be implemented in a total of two CLB slices (requiring four 4-bit look-up-tables), yielding a compact implementation which minimizes routing between CLB slices. Finally, the components of the Serpent round function — key masking, S-Box substitution, and linear transformation — were found to be simple in structure, resulting in the round function requiring few hardware resources. The results for each of the Serpent implementations using the four architecture types is found in Appendix A.

Implementing a single round of the Serpent algorithm provides the greatest area-optimized solution. However, a significant performance improvement was achieved by performing partial loop unrolling. When operating in FB mode, the 8-round loop unrolled implementation optimized for speed was found to yield both the highest throughput and TPS. By removing the need for S-Box multiplexing (as one copy of each possible S-Box grouping is now included within one of the eight rounds) while minimizing the round key multiplexing, significantly higher performance values were seen as compared to those of the 1-round loop unrolled and 32-round loop unrolled architectures.

When operating in NFB mode, a full-length pipelined architecture was found to offer the advantage of extremely high throughput rates once the latency of the pipeline was met, outperforming smaller partially pipelined implementations. In the fully pipelined architecture, all of the elements of a given round function are implemented as combinatorial logic. Other AES finalists cannot be implemented using a fully pipelined architecture due to the larger round functions. However, due to the small size of the Serpent S-Boxes (4-bit look-up-tables), the cost of S-Box replication is minimal in terms of the required hardware. As a result, the TPS exhibited by the full-length pipeline implementation was significantly higher than that of any other implementation when operating in NFB mode. Interestingly, for the full-length pipeline implementation, area optimization yielded the highest throughput while speed optimization yielded the best TPS. This speaks to the relative uncertainty of the synthesis results for a given architecture and algorithm as discussed in Section III-D. Finally, sub-pipelining of the partially pipelined architectures was determined to yield no throughput increase. Because the round function components are all simple in structure, there is little performance to be gained by subdividing them with registers in an attempt to reduce the delay between stages. As a result, the increase in the system's clock frequency would not outweigh the increase in the number of clock cycles required to perform an encryption, resulting in a performance degradation.

D. Architectural Implementation Analysis — Twofish

When implementing the Twofish algorithm using full keying [22], it was first determined that the synthesis tools were unable to minimize the Twofish S-Boxes to the extent of other AES finalist algorithms due to the S-Boxes being key-dependent. Therefore, the overall size of a Twofish round was too large to allow for a fully unrolled or fully pipelined implementation of the algorithm within the target FPGA. Moreover, the key-dependent S-Boxes were found to require nearly half of the delay associated with the Twofish round function. The results for each of the Twofish implementations using the four architecture types is found in Appendix A.

Iterative looping optimized for area was found to yield both the best throughput and TPS when operating in FB mode. The iterative looping architecture was able to reach a significantly faster system clock frequency as compared to the other loop unrolling and pipeline implementations, resulting in superior performance results. When operating in NFB mode, a partially pipelined architecture optimized for area with two additional sub-pipeline stages was found to offer the advantage of extremely high throughput rates and TPS once the latency of the pipeline was met, with the 8-stage partial pipeline implementation displaying the best throughput and TPS results. While Twofish cannot be implemented using a fully pipelined architecture due to the large size of the round function, significant throughput increases were seen as compared to the loop unrolling architectures. Sub-pipelining of the partially pipelined architectures was implemented by inserting a pipeline sub-stage within the Twofish round function. Based on the delay analysis of the partial pipeline implementations, it was determined that nearly half of the round function's associated delay was attributed to the S-Box substitutions. Therefore,

an additional pipeline sub-stage was implemented so as to separate the S-Boxes from the rest of the round function. As a result, an increase by a factor of nearly 2 was seen in the system's clock frequency, resulting in a similar increase in throughput when operating in NFB mode. Further sub-pipelining was implemented by sub-dividing the S-Boxes with a pipeline sub-stage and moving the previously inserted sub-pipeline stage to balance the delay between sub-pipeline stages. This resulted in an increase in system clock frequency by a factor of nearly 3 as compared to the pipelined implementation with no sub-stages. When additional sub-pipelining was investigated, it was found that the logic functions implemented between the two sub-stages were all simple in structure. As a result, further sub-pipelining was not implemented as the increase in clock frequency achieved by adding an additional sub-stage would not outweigh the number of cycles required to perform an encryption, resulting in a performance degradation.

VI. Performance Evaluation

The data shown in Tables II, III, IV, and V detail the optimal implementations (in terms of throughput and TPS) for each of the AES finalists. The architecture types — loop unrolling (LU), full or partial pipelining (PP), and partial pipelining with sub-pipelining (SP) — are listed along with the number of stages and (if necessary) sub-pipeline stages in the associated implementation; e.g., LU-4 implies a loop unrolling architecture with four rounds, while SP-2-1 implies a partially pipelined architecture with two stages and one sub-pipeline stage per pipeline stage. As a result, the SP-2-1 architecture implements two rounds of the given cipher with a total of two stages per round. Throughput is calculated as:

$$Throughput := (128 \text{ Bits } * \text{ Clock Frequency}) / (\text{Cycles Per Encrypted Block})$$

Note that the implementation of a one stage partial pipeline architecture, an iterative looping architecture, and a one round loop unrolled architecture are all equivalent and are therefore not listed separately. Also note that the computed throughput for implementations that employ any form of hardware pipelining (as discussed in Section IV) are made assuming that the pipeline latency has been met.

The number of CLBs required as well as the maximum operating frequency for each implementation was obtained from the Xilinx report files. Note that the Xilinx tools assume the absolute worst possible operating conditions — highest possible operating temperature, lowest possible supply voltage, and worst-case fabrication tolerance for the speed grade of the FPGA [29]. As a result, it is common for actual implementations to achieve slightly better performance results than those specified in the Xilinx report files.

Alg.	Arch.	Opt.	Cycles Per Enc. Block	Throughput (Gbit/s)
RC6	SP-10-2	Speed	2	**2.40**
Rijndael	SP-5-1	Speed	2.1	**1.94**
Serpent	PP-32	Area	1	**5.04**
Twofish	SP-8-2	Speed	2	**2.40**

TABLE II
AES finalist throughput evaluation — best non-feedback mode implementations

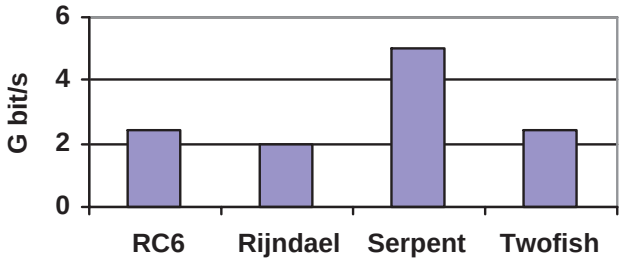

Fig. 1. Best throughput — non-feedback mode

Alg.	Arch.	Opt.	Cycles Per Enc. Block	Throughput (Mbit/s)
RC6	PP-2	Speed	20	**126.5**
Rijndael	LU-2	Speed	6	**300.1**
Serpent	LU-8	Speed	4	**444.2**
Twofish	SP-1-1	Area	32	**127.7**

TABLE III
AES finalist throughput evaluation — best feedback mode implementations

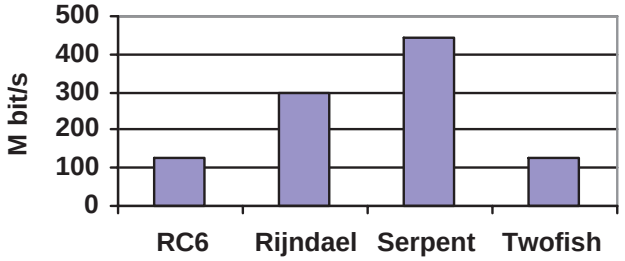

Fig. 2. Best throughput — feedback mode

Alg.	Arch.	Opt.	Slices	Key Storage min Slices	TPS
RC6	SP-10-2	Speed	10856	704	**220881**
Rijndael	SP-2-1	Speed	4871	704	**194837**
Serpent	PP-32	Speed	9004	2112	**539778**
Twofish	SP-8-2	Area	9486	672	**248666**

TABLE IV
AES FINALIST TPS EVALUATION — BEST NON-FEEDBACK MODE IMPLEMENTATIONS

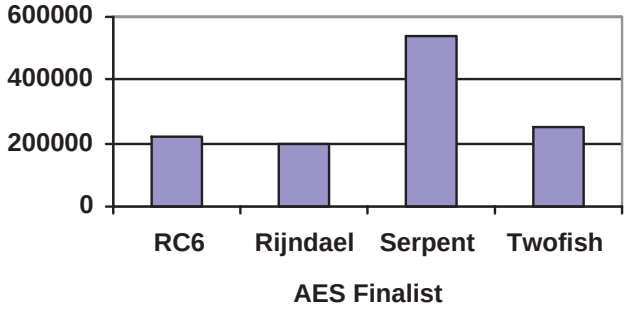

Fig. 3. Best TPS — non-feedback mode

Alg.	Arch.	Opt.	Slices	Key Storage min Slices	TPS
RC6	PP-2	Speed	3189	704	**39662**
Rijndael	LU-1	Speed	3528	704	**83387**
Serpent	LU-8	Speed	7964	2112	**55771**
Twofish	SP-1-1	Area	2695	672	**47380**

TABLE V
AES FINALIST TPS EVALUATION — BEST FEEDBACK MODE IMPLEMENTATIONS

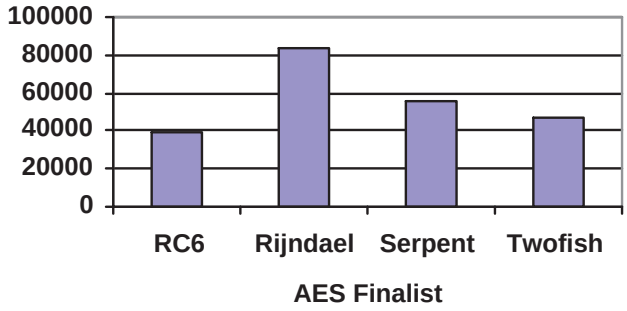

Fig. 4. Best TPS — feedback mode

Based on the data shown in Tables II and III, the Serpent algorithm clearly outperforms the other evaluated AES finalists in both modes of operation in terms of throughput. As compared to its nearest competitor, Serpent exhibits a throughput increase of a factor 2.1 in NFB mode and a factor 1.5 in FB mode. From the data shown in Tables IV and V, the Serpent algorithm outperforms the other evaluated AES finalists in terms of TPS when operating in NFB mode. As compared to it's nearest competitor, Serpent exhibits a TPS increase of a factor of 2.2 in NFB mode. When operating in FB mode, Rijndael outperforms the other evaluated AES finalists in terms of TPS, outperforming its nearest competitor by a factor of 1.5. Interestingly, RC6, Rijndael, and Twofish all exhibit similar throughput and TPS results in NFB mode. Additionally, Rijndael exhibits significantly improved performance in respect to the other evaluated AES finalists when operating in FB mode, significantly outperforming its competitors in TPS while outperforming RC6 and Twofish in throughput. However, Rijndael is still 50% slower than Serpent when operating in FB mode.

One of the main findings of our investigation, namely that Serpent appears to be especially well suited for an FPGA implementation, seems especially interesting considering that Serpent is clearly not the fastest algorithm with respect to most software comparisons [6]. Another major result of our study is that all four algorithms considered easily achieve Gigabit encryption rates with standard commercially available FPGAs. The algorithms are at least one order of magnitude faster than the best reported software realizations when operating in NFB mode. These speed-ups are essentially achieved by parallelization (pipelining and sub-pipelining) of the loop structure and by wide operand processing (e.g., processing of 128 bits in once clock cycle), both of which are not feasible on current processors. We would like to stress that the pipelined architectures cannot be used to their maximum ability for modes of operation which require feedback (CFB, OFB, etc.). However we believe that for many applications which require high encryption rates, non-feedback modes (or modified feedback modes such as interleaved CFB) will be the modes of choice. Note that the Counter Mode (in which pseudo-random input is encrypted by the block cipher, the output of which is XORed with the plaintext) grew out of the need for high speed encryption of ATM networks which required parallelization of the encryption algorithm.

VII. Conclusions

The importance of the Advanced Encryption Standard and the significance of high throughput implementations of the AES finalists has been examined. A design methodology was established which in turn led to the architectural requirements for a target FPGA. The core operations of the AES finalists were identified and multiple architecture options were discussed. For each of the AES finalists, an implementation analysis for each architecture option when optimized for both area and speed was performed to determine the suitability for hardware implementation of each finalist. Based on the implementation results, the best implementations were identified for each AES finalist in both non-feedback and feedback modes in terms of throughput and area efficiency. Upon comparison, it was determined that the Serpent algorithm yielded the best throughput results in both modes of operation. When evaluating throughput per slice, the Serpent algorithm was also found to yield the best results when operating in non-feedback mode while the Rijndael algorithm was found to yield the best results when operating in feedback mode. The Serpent algorithm is clearly the best choice when throughput is the key evaluation characteristic regardless of the mode of operation. When considering throughput per slice, the Serpent algorithm is the best choice for non-feedback mode operation while the Rijndael algorithm is the best choice for feedback mode operation. This study was completed prior to NIST's selection of Rijndael as the Advanced Encryption Algorithm. However, the findings of this study support the NIST opinion that Rijndael shows a strong relative performance in hardware implementations.

VIII. Acknowledgement

We would like to thank Pawel Chodowiec and Kris Gaj from George Mason University for their helpful discussion and the VHDL code modules that were provided to assist in the implementation of the AES finalists. We would also like to thank Alan Martello from the University of Pittsburgh for his public-domain VHDL code module that was used in implementation of the AES finalists.

References

[1] B. Schneier, *Applied Cryptography*. New York, New York, USA: John Wiley & Sons Inc., 2nd ed., 1996.

[2] National Institute of Standards and Technology (NIST), *Second Advanced Encryption Standard (AES) Conference*, (Rome, Italy), March 1999.

[3] R. Doud, "Hardware Crypto Solutions Boost VPN," *Electronic Engineering Times*, pp. 57–64, April 12 1999.

[4] C. Patterson, "High Performance DES Encryption in Virtex FPGAs Using JBits," in *IEEE Symposium on Field–Programmable Custom Computing Machines (FCCM 2000)* (K. L. Pocek and J. M. Arnold, eds.), April 2000.

[5] C. Patterson, "A Dynamic Implementation of the Serpent Block Cipher," in *Cryptographic Hardware and Embedded Systems (CHES 2000)* (C. K. Koc and C. Paar, eds.), (Worcester, Massachusetts, USA), pp. 142–156, Springer-Verlag, August 17–18 2000.

[6] B. Gladman, "Implementation Experience with AES Candidate Algorithms," in *Proceedings: Second AES Candidate Conference (AES2)*, (Rome, Italy), March 1999.

[7] J. P. Kaps and C. Paar, "Fast DES implementation on FPGAs and its application to a universal key-search machine," in *Fifth Annual Workshop on Selected Areas in Cryptography* (S. Tavares and H. Meijer, eds.), vol. LNCS 1556, (Berlin, Germany), Springer-Verlag, August 1998. Conference Location: Queen's University, Kingston, Ontario, Canada.

[8] E. Biham, "A Fast New DES Implementation in Software," in *Fourth International Workshop on Fast Software Encryption*, vol. LNCS 1267, (Berlin, Germany), pp. 260–272, Springer-Verlag, 1997.

[9] A. Pfitzmann and R. Assman, "More Efficient Software Implementations of (Generalized) DES," *Computers & Security*, vol. 12, no. 5, pp. 477–500, 1993.

[10] J. Hughes, "Implementation of NBS/DES Encryption Algorithm in Software," in *Colloquium on Techniques and Implications of Digital Privacy and Authentication Systems*, 1981.

[11] B. Preneel, V. Rijmen, and A. Bosselaers, "Recent Developments in the Design of Conventional Cryptographic Algorithms," in *Computer Security and Industrial Cryptography, State of the Art and Evolution* (B. Preneel and V. Rijmen, eds.), vol. LNCS 1528, pp. 106–131, Springer-Verlag, 1998.

[12] D. Runje and M. Kovac, "Universal Strong Encryption FPGA Core Implementation," in *Proceedings of Design, Automation, and Test in Europe*, (Paris, France), pp. 923–924, February 1998.

[13] O. Mencer, M. Morf, and M. J. Flynn, "Hardware Software Tri-Design of Encryption for Mobile Communication Units," in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, (New York, New York, USA), pp. 3045–3048, IEEE, May 1998. Conference Location: Seattle, Washington, USA.

[14] A. Elbirt, "An FPGA Implementation and Performance Evaluation of the CAST-256 Block Cipher," Technical Report, Cryptography and Information Security Group, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, May 1999.

[15] M. Riaz and H. Heys, "The FPGA Implementation of RC6 and CAST-256 Encryption Algorithms," in *Proceedings: IEEE 1999 Canadian Conference on Electrical and Computer Engineering*, (Edmonton, Alberta, Canada), March 1999.

[16] A. Dandalis, V. K. Prasanna, and J. D. P. Rolim, "A Comparative Study of Performance of AES Final Candidates Using FPGAs," in *Workshop on Cryptographic Hardware and Embedded Systems - CHES '00* (Ç. Koç and C. Paar, eds.), (Worcester, Massachusetts, USA), Springer-Verlag, August 2000.

[17] A. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists," in *The Third Advanced Encryption Standard Candidate Conference*, (New York, New York, USA), pp. 13–27, National Institute of Standards and Technology, April 13–14 2000.

[18] N. Weaver and J. Wawrzynek, "A Comparison of the AES Candidates Amenability to FPGA Implemenation," in *The Third Advanced Encryption Standard Candidate Conference*, (New York, New York, USA), pp. 28–39, National Institute of Standards and Technology, April 13–14 2000.

[19] K. Gaj and P. Chodowiec, "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware," in *The Third Advanced Encryption Standard Candidate Conference*, (New York, New York, USA), pp. 40–54, National Institute of Standards and Technology, April 13–14 2000.

[20] C. Phillips and K. Hodor, "Breaking the 10k FPGA Barrier Calls For an ASIC-Like Design Style," *Integrated System Design*, 1996.

[21] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," in *First Advanced Encryption Standard (AES) Conference*, (Ventura, California, USA), 1998.

[22] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, and C. Hall, "Twofish: A 128-Bit Block Cipher," in *First Advanced Encryption Standard (AES) Conference*, (Ventura, California, USA), 1998.

[23] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," in *First Advanced Encryption Standard (AES) Conference*, (Ventura, California, USA), 1998.

[24] R. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin, "The RC6$^{\mathrm{TM}}$ Block Cipher," in *First Advanced Encryption Standard (AES) Conference*, (Ventura, California, USA), 1998.

[25] Xilinx Inc., San Jose, California, USA, *Virtex 2.5V Field Programmable Gate Arrays*, 1998.

[26] B. Chetwynd, "Universal Block Cipher Module: Towards a Generalized Architectures for Block Ciphers," Master's thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, November 1999.

[27] S. Brown and J. Rose, "FPGA and CPLD Architectures: A Tutorial," *IEEE Design & Test of Computers*, vol. 13, no. 2, pp. 42–57, 1996.

[28] C. Paar, "Optimized arithmetic for Reed-Solomon encoders," in *1997 IEEE International Symposium on Information Theory*, (Ulm, Germany), p. 250, June 29–July 4 1997.

[29] P. Alfke, "Xilinx M1 Timing Parameters." Electronic Mail Personal Correspondance, December 1999.

Appendix A — Implementation Data

Tables VI, VII, VIII, and IX detail the throughput measurements for the implementations of the four architecture types for each of the AES finalists for both NFB and FB mode. The architecture types — loop unrolling (LU), full or partial pipelining (PP), and partial pipelining with sub-pipelining (SP) — are listed along with the number of stages and (if necessary) sub-pipeline stages in the associated implementation; e.g., LU-4 implies a loop unrolling architecture with four rounds, while SP-2-1 implies a partially pipelined architecture with two stages and one sub-pipeline stage per pipeline stage. As a result, the SP-2-1 architecture implements two rounds of the given cipher with a total of two stages per round.

It is interesting to note that the slice counts do not increase linearly based on the number of rounds in the implementation of a given architecture. The round function for each algorithm is comprised of combinatorial logic which is implemented via the look-up-tables of the CLB slice while the round keys occupy the register bits of the CLB slice. While the total number of register bits required for round key storage is fixed, the number of slices required for storage of the round keys varies with the number of rounds implemented. Because the round functions do not make use of the associated register bits within a CLB slice, these bits may be used for round key storage, effectively packing the round keys with the round functions to minimize CLB slice use. Little packing is possible when only one round is implemented, resulting in most of the round keys being stored in CLB slices whose look-up-tables are unused. However, as more rounds are unrolled or pipelined, more round keys may occupy the unused register bits of the CLB slices used to implement the round functions, resulting in greater packing. As a result, while the size of the implementation increases when the number of rounds unrolled or pipelined is increased, this increase is partially offset by the packing of the round keys within the round structure.

Algorithm	Architecture	Slices	Clock Frequency (MHz)	Cycles per Block	Throughput (Mbit/s)	TPS
RC6	LU-1	2638	13.8	20	88.5	33558
RC6	LU-2	3069	7.3	10	94.0	30638
RC6	LU-4	4070	3.7	5	94.8	23304
RC6	LU-5	4476	2.9	4	92.2	20604
RC6	LU-10	6406	1.5	2	97.4	15206
RC6	PP-2	3189	19.8	10	253.0	79325
RC6	PP-4	4411	12.3	5	315.5	71524
RC6	PP-5	4848	12.1	4	386.7	79762
RC6	PP-10	7412	13.3	2	848.1	114426
RC6	SP-1-1	2967	26.2	20	167.6	56498
RC6	SP-2-1	3709	26.4	10	337.8	91087
RC6	SP-4-1	5229	24.6	5	629.8	120436
RC6	SP-5-1	5842	25.8	4	825.2	141250
RC6	SP-10-1	8999	26.6	2	1704.6	189425
RC6	SP-1-2	3134	39.1	20	250.0	79769
RC6	SP-2-2	4062	38.9	10	497.4	122448
RC6	SP-4-2	5908	31.3	5	802.3	135795
RC6	SP-5-2	6415	33.3	4	1067.0	166330
RC6	**SP-10-2**	**10856**	**37.5**	**2**	**2397.9**	**220881**
Rijndael	LU-1	3528	25.3	11	294.2	83387
Rijndael	LU-2	5302	14.1	6	300.1	56605
Rijndael	LU-5	10286	5.6	3	237.4	23084
Rijndael	PP-2	5281	23.5	5.5	545.9	103372
Rijndael	PP-5	10533	20.0	2.2	1165.8	110680
Rijndael	SP-1-1	3061	40.4	10.5	491.9	160702
Rijndael	**SP-2-1**	**4871**	38.9	5.25	949.1	**194837**
Rijndael	**SP-5-1**	10992	**31.8**	**2.1**	**1937.9**	176297
Serpent	LU-1	5511	15.5	32	61.9	11236
Serpent	LU-8	7964	13.9	4	444.2	55771
Serpent	LU-32	8103	2.4	1	312.3	38544
Serpent	PP-8	6849	30.4	4	971.8	141886
Serpent	**PP-32**	**9004**	**38.0**	**1**	**4860.2**	**539778**
Twofish	LU-1	2666	13.0	16	104.2	39079
Twofish	LU-2	3392	7.1	8	113.6	33495
Twofish	LU-4	4665	3.3	4	106.8	22890
Twofish	LU-8	6990	1.7	2	108.1	15464
Twofish	PP-2	3519	11.9	8	190.4	54115
Twofish	PP-4	5044	11.5	4	369.3	73218
Twofish	PP-8	7817	10.8	2	689.5	88210
Twofish	SP-1-1	3053	29.9	16	239.2	78339
Twofish	SP-2-1	3869	28.6	8	457.1	118137
Twofish	SP-4-1	5870	27.3	4	872.3	148606
Twofish	SP-8-1	9345	24.8	2	1585.3	169639
Twofish	SP-1-2	2954	37.9	16	303.4	102719
Twofish	SP-2-2	4012	45.8	8	732.5	182580
Twofish	SP-4-2	6101	38.8	4	1242.1	203597
Twofish	**SP-8-2**	**10008**	**37.4**	**2**	**2395.7**	**239380**

TABLE VI

AES FINALIST PERFORMANCE EVALUATION — SPEED OPTIMIZED NON-FEEDBACK MODE

Algorithm	Architecture	Slices	Clock Frequency (MHz)	Cycles per Block	Throughput (Mbit/s)	TPS
RC6	LU-1	2784	14.6	20	93.6	33630
RC6	LU-2	3165	6.9	10	88.4	27934
RC6	LU-4	4058	3.7	5	94.7	23335
RC6	LU-5	4465	3.0	4	97.1	21744
RC6	LU-10	6398	1.5	2	97.3	15205
RC6	PP-2	3215	19.1	10	244.6	76091
RC6	PP-4	4322	14.1	5	362.0	83760
RC6	PP-5	4881	12.8	4	411.1	84225
RC6	PP-10	7393	11.7	2	751.3	101623
RC6	SP-1-1	3012	28.1	20	180.1	59778
RC6	SP-2-1	3609	24.3	10	311.3	86245
RC6	SP-4-1	4969	25.3	5	648.0	130411
RC6	SP-5-1	5700	24.8	4	792.7	139065
RC6	SP-10-1	8687	23.6	2	1509.1	173714
RC6	SP-1-2	3136	37.4	20	239.1	76233
RC6	SP-2-2	3942	37.7	10	483.1	122558
RC6	SP-4-2	5637	34.8	5	891.3	158123
RC6	SP-5-2	6471	26.2	4	839.7.0	129760
RC6	**SP-10-2**	**10308**	**31.2**	**2**	**1996.9**	**193720**
Rijndael	LU-1	3488	24.9	11	290.1	83159
Rijndael	LU-2	5276	13.9	6	296.4	56184
Rijndael	LU-5	10276	5.7	3	243.1	23654
Rijndael	PP-2	5275	24.7	5.5	575.3	109066
Rijndael	PP-5	10550	16.4	2.2	956.8	90692
Rijndael	SP-1-1	3540	40.0	10.5	487.7	137763
Rijndael	**SP-2-1**	**5449**	40.0	5.25	975.6	**179034**
Rijndael	**SP-5-1**	10923	**30.8**	**2.1**	**1878.5**	**171976**
Serpent	LU-1	5890	19.2	32	77.0	13065
Serpent	LU-8	6467	10.4	4	333.3	51535
Serpent	LU-32	10936	2.8	1	355.8	32538
Serpent	PP-8	6849	38.8	4	1241.6	181277
Serpent	**PP-32**	**11988**	**39.3**	**1**	**5035.0**	**420004**
Twofish	LU-1	2695	16.0	16	127.7	47380
Twofish	LU-2	3372	6.6	8	106.1	31469
Twofish	LU-4	4642	3.5	4	112.2	24162
Twofish	LU-8	7011	1.6	2	103.4	14752
Twofish	PP-2	3467	12.8	8	204.7	59030
Twofish	PP-4	5033	11.5	4	366.8	72882
Twofish	PP-8	7814	10.6	2	675.9	86499
Twofish	SP-1-1	2865	28.2	16	225.7	78771
Twofish	SP-2-1	3619	28.7	8	459.6	126988
Twofish	SP-4-1	5439	26.5	4	849.2	156129
Twofish	SP-8-1	8745	26.7	2	1709.0	195425
Twofish	SP-1-2	2976	41.4	16	331.2	111306
Twofish	SP-2-2	3840	38.6	8	617.6	160842
Twofish	SP-4-2	5720	35.0	4	1120.4	195883
Twofish	**SP-8-2**	**9486**	**36.9**	**2**	**2358.8**	**248666**

TABLE VII

AES FINALIST PERFORMANCE EVALUATION — AREA OPTIMIZED NON-FEEDBACK MODE

Algorithm	Architecture	Slices	Clock Frequency (MHz)	Cycles per Block	Throughput (Mbit/s)	TPS
RC6	LU-1	2638	13.8	20	88.5	33558
RC6	LU-2	3069	7.3	10	94.0	30638
RC6	LU-4	4070	3.7	5	94.8	23304
RC6	LU-5	4476	2.9	4	92.2	20604
RC6	LU-10	6406	1.5	2	97.4	15206
RC6	**PP-2**	**3189**	**19.8**	**20**	**126.5**	**39662**
RC6	PP-4	4411	12.3	20	78.9	17881
RC6	PP-5	4848	12.1	20	77.3	15952
RC6	PP-10	7412	13.3	20	84.8	11443
RC6	SP-1-1	2967	26.2	40	83.8	28249
RC6	SP-2-1	3709	26.4	40	84.5	22772
RC6	SP-4-1	5229	24.6	40	78.7	15055
RC6	SP-5-1	5842	25.8	40	82.5	14125
RC6	SP-10-1	8999	26.6	40	85.2	9471
RC6	SP-1-2	3134	39.1	60	83.3	26590
RC6	SP-2-2	4062	38.9	60	82.9	20408
RC6	SP-4-2	5908	31.3	60	66.9	11316
RC6	SP-5-2	6415	33.3	60	71.1	11089
RC6	SP-10-2	10856	37.5	60	79.9	7363
Rijndael	**LU-1**	**3528**	25.3	11	294.2	**83387**
Rijndael	**LU-2**	5302	**14.1**	**6**	**300.1**	56605
Rijndael	LU-5	10286	5.6	3	237.4	23084
Rijndael	PP-2	5281	23.5	11	273.0	51686
Rijndael	PP-5	10533	20.0	11	233.2	22136
Rijndael	SP-1-1	3061	40.4	21	246.0	80351
Rijndael	SP-2-1	4871	38.9	21	237.3	48709
Rijndael	SP-5-1	10992	31.8	21	193.8	17630
Serpent	LU-1	5511	15.5	32	61.9	11236
Serpent	**LU-8**	**7964**	**13.9**	**4**	**444.2**	**55771**
Serpent	LU-32	8103	2.4	1	312.3	38544
Serpent	PP-8	6849	30.4	32	121.5	17736
Serpent	PP-32	9004	38.0	32	151.9	16868
Twofish	LU-1	2666	13.0	16	104.2	39079
Twofish	LU-2	3392	7.1	8	113.6	33495
Twofish	LU-4	4665	3.3	4	106.8	22890
Twofish	LU-8	6990	1.7	2	108.1	15464
Twofish	PP-2	3519	11.9	16	95.2	27058
Twofish	PP-4	5044	11.5	16	92.3	18305
Twofish	PP-8	7817	10.8	16	86.2	11026
Twofish	**SP-1-1**	**3053**	29.9	32	119.6	**39169**
Twofish	SP-2-1	3869	28.6	32	114.3	29534
Twofish	SP-4-1	5870	27.3	32	109.0	18576
Twofish	SP-8-1	9345	24.8	32	99.1	10602
Twofish	SP-1-2	2954	37.9	48	101.1	34240
Twofish	**SP-2-2**	4012	**45.8**	**48**	**122.1**	30430
Twofish	SP-4-2	6101	38.8	48	103.5	16966
Twofish	SP-8-2	10008	37.4	48	99.8	9974

TABLE VIII

AES FINALIST PERFORMANCE EVALUATION — SPEED OPTIMIZED FEEDBACK MODE

Algorithm	Architecture	Slices	Clock Frequency (MHz)	Cycles per Block	Throughput (Mbit/s)	TPS
RC6	LU-1	2784	14.6	20	93.6	33630
RC6	LU-2	3165	6.9	10	88.4	27934
RC6	LU-4	4058	3.7	5	94.7	23335
RC6	LU-5	4465	3.0	4	97.1	21744
RC6	LU-10	6398	1.5	2	97.3	15205
RC6	**PP-2**	**3215**	**19.1**	**20**	**122.3**	**38046**
RC6	PP-4	4322	14.1	20	90.5	20940
RC6	PP-5	4881	12.8	20	82.2	16845
RC6	PP-10	7393	11.7	20	75.1	10162
RC6	SP-1-1	3012	28.1	40	90.0	29889
RC6	SP-2-1	3609	24.3	40	77.8	21561
RC6	SP-4-1	4969	25.3	40	81.0	16301
RC6	SP-5-1	5700	24.8	40	79.3	13907
RC6	SP-10-1	8687	23.6	40	75.5	8686
RC6	SP-1-2	3136	37.4	60	79.7	25411
RC6	SP-2-2	3942	37.7	60	80.5	20426
RC6	SP-4-2	5637	34.8	60	74.3	13177
RC6	SP-5-2	6471	26.2	60	56.0	8651
RC6	SP-10-2	10308	31.2	60	66.6	6457
Rijndael	**LU-1**	**3488**	24.9	11	290.1	**83159**
Rijndael	**LU-2**	5276	**13.9**	**6**	**296.4**	56184
Rijndael	LU-5	10276	5.7	3	243.1	23654
Rijndael	PP-2	5274	24.7	11	287.7	54544
Rijndael	PP-5	10550	16.4	11	191.4	18138
Rijndael	SP-1-1	3540	40.0	21	243.8	68881
Rijndael	SP-2-1	5449	40.0	21	243.9	44758
Rijndael	SP-5-1	10923	30.8	21	187.8	17198
Serpent	LU-1	5890	19.2	32	77.0	13065
Serpent	**LU-8**	**6467**	**10.4**	**4**	**333.3**	**51535**
Serpent	LU-32	10936	2.8	1	355.8	32538
Serpent	PP-8	6849	38.8	32	155.2	22660
Serpent	PP-32	11988	39.3	32	157.3	13125
Twofish	**LU-1**	**2695**	**16.0**	**16**	**127.7**	**47380**
Twofish	LU-2	3372	6.6	8	106.1	31469
Twofish	LU-4	4642	3.5	4	112.2	24162
Twofish	LU-8	7011	1.6	2	103.4	14752
Twofish	PP-2	3467	12.8	16	102.3	29515
Twofish	PP-4	5033	11.5	16	91.7	18221
Twofish	PP-8	7814	10.6	16	84.5	10812
Twofish	SP-1-1	2865	28.2	32	112.8	39386
Twofish	SP-2-1	3619	28.7	32	114.9	31747
Twofish	SP-4-1	5439	26.5	32	106.1	19516
Twofish	SP-8-1	8745	26.7	32	106.8	12214
Twofish	SP-1-2	2976	41.4	48	110.4	37102
Twofish	SP-2-2	3840	38.6	48	102.9	26807
Twofish	SP-4-2	5720	35.0	48	93.4	16324
Twofish	SP-8-2	9486	36.9	48	98.3	10361

TABLE IX

AES FINALIST PERFORMANCE EVALUATION — AREA OPTIMIZED FEEDBACK MODE