CONSIDERATIONS ON HARDWARE IMPLEMENTATIONS OF ENCRYPTION ALGORITHMS

Ioan Mang
University of Oradea, Faculty of Electrotechnics and Informatics,
Computer Science Department, 3, Armatei Romane Str., 3700 Oradea, Romania

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 [2]. From the fifteen official candidate algorithms 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 both hardware and software. NIST has announced that both hardware and software performance measurements will be included in their efficiency testing. So far, however, virtually all performance comparisons have been restricted to software implementations on various platforms [3].

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 [2]. 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. The downside of traditional (ASIC) hardware implementation is the lack of flexibility with respect to algorithm and parameter switch. A promising alternative for implementing block cipher are reconfigurable hardware devices, such as Field Programmable Gate Arrays (FPGAs). The potential advantages of encryption algorithms implemented in FPGAs include: algorithm agility, algorithm upload or modification possibilities, architecture efficiency, throughput, cost efficiency.

This paper describes a thorough comparison the AES finalist algorithms RC6, Rijndael, Serpent, and Twofish with respect to implementation on state-of-the-art FPGAs. 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 [7].

What follows is an investigation of the AES finalists to determine the nature of their underlying components. 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.

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 [8]. 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 [5], which typically have throughputs below 100 Mbit/s, although a
137 Mbit/s implementation has been reported as well [5]. 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 [11] to 528 Mbit/s. 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 [9] 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 [7], a factor of three faster than FPGA implementations. While an FPGA implementation of RC6 achieved data rates of 37.8 Mbit/s [9], our findings indicate that considerably higher data rates are achievable. 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 implement-tations. 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 high frequencies while typical implementations that target FPGAs reach a maximum clock frequency of 50 MHz.

IMPLEMENTATION - GENERAL CONSIDERATIONS

Each AES finalist was implemented in VHDL using a bottom-up design and test methodology. The same hardware interface was used for each of the implementations. In an effort to achieve the maximum efficiency, 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. 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 [4],[12],[6],[10]. 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, proofing that the implementations were successful. When examining the AES finalists for hardware implementation within a 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 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, a fully pipelined architecture requires 128-bit wide pipeline stages, resulting in the need of a register-rich architecture in order to achieve a fast, synchronous implementation. Moreover, to allow for a regular layout of design elements as well as to minimize the routing required between configurable units, it is desirable to have as many register bits as possible for each of the FPGA's configurable units [9]. In addition to architectural requirements, scalability and costs must be also considered. We believe that the chosen FPGA should be the best chip available, capable of providing the largest amount of hardware resources as well as being highly flexible so as to yield optimal performance.

Based on the aforementioned considerations, the Xilinx Virtex XCV1000BG560-4 FPGA was chosen as the target device. 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. XACTstep 4.1i by Xilinx, Inc. was used to place and route the synthesized implementations and 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.

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.

4.1 Core Operations of AES Finalist Algorithms. In terms of complexity, the operations detailed in Table 1 that require the most hardware resources as well as computation time are the mod2^{32} multiplication and the variable rotation operations. Implementing wide multipliers in hardware is an inherently difficult task that requires significant hardware resources. Additionally, algorithms that employ large variable rotations require a moderate amount of multiplexing hardware if carefully designed. S-Boxes may be implemented in either combinatorial logic or embedded RAM. Fast operations such as bit-wise XOR, mod2^{32} addition and subtraction, and fixed value shifting 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 far less resources than general multiplications [1].

Based on our evaluation of the AES finalists, the MARS algorithm appeared to be the most resource intensive based on its use of large S-Boxes, and mod2^{32} multiplications. Due to this evaluation and a lack of development resources, the MARS algorithm was omitted from this study.

Table 1 - AES finalists core operations

algorithm	XOR	mod 2^{32} add	mod 2^{32} substr.	fixed shift	varia-ble rotate	mulmod 2^{32}	GF(2^8) multipl.	LUT
MARS	●	●	●	●	●	●		●
RC6	●	●		●	●	●		
Rijndael	●			●			●	●
Serpent	●			●				●
Twofish	●	●		●			●	●

4.2 Hardware Architecture. The AES finalists are all comprised of a basic looping structure 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. It 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. 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 the system will output a 128-bit block of ciphertext at each clock cycle once the latency of the pipeline has been met. 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. 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 [2]. When operating in feedback modes (FB), 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. Sub-pipelining a 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 non-feedback (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.

5 ARCHITECTURAL IMPLEMENTATION ANALYSIS

For each of the AES finalists, the four architecture options described in Section 4.2 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.

RC6. When implementing the RC6 algorithm, it was first determined that the RC6 mod2^{32} multiplication was the dominant element of the round function in terms of required logic resources. Each RC6 round requires two copies of the mod2^{32} multiplier. However, it was found that the RC6 round function does not require a general mod2^{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 mod2^{32} addition - were found to be simple in structure, resulting in these elements of the round function requiring few hardware resources. While variable shifting operations have the potential to require considerable hardware resources, the 5-bit variable shifting required by the RC6 round function needs few hardware resources. Instead of implementing a 32-to-1 multiplexer for each of the 32 rotation output bits, a five-level multiplexing approach was used. The variable rotation is broken into five stages, each of which is controlled by one of the five shifting bits. For each rotation output bit of a given stage, a 2-to-1 multiplexer controlled by the stage's shifting bit is used. This result in an overall implementation that is smaller and faster when compared to the one-stage barrel-shifter implementation [10]. Finally, it was found that the synthesis tools could not minimize the overall size of a 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. Implementing a single round of the RC6 algorithm provides the greatest area-optimized solution. Further loop unrolling provided only minor throughput increases as the decrease in the number of cycles per encrypted block was offset by the rapidly decreasing system clock frequency. 2-stage partial pipelining was found to yield the highest throughput 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 with two additional sub-pipeline stages was found to offer the advantage of extremely high throughput rates once the latency of the pipeline was met, with the 10-stage partial pipeline implementation displaying the best throughput and 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 mod2^{32} multiplier. Therefore, two additional pipeline sub-stages were implemented so as to subdivide the multiplier into smaller blocks, resulting in a total of 3 pipeline stages per round function. As a result, an increase by a factor of more than 2.5 was seen in the system's clock frequency, resulting in a similar increase in throughput when operating in NFB mode.

Rijndael. When implementing the Rijndael algorithm, it was first determined that the S-Boxes were the dominant element of the round function in terms of required logic resources. Each round requires 16 copies of the S-Boxes, each of which is an 8-bit to 8-bit look-up-table, requiring signify-cant hardware resources. However, the remaining compo-nents 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. Also, 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. Surprisingly, a 1 round partially pipelined implementation with 1 sub-pipeline stage provided the most area-optimized solution. As compared to a 1-stage implementation with no sub-pipelining, the addition of a sub-pipeline stage afforded the synthesis tool greater flexibility in its optimizations, resulting in a more area efficient implementation. While 2-stage loop unrolling was found to yield the highest throughput when operating in FB mode, the measured throughput was within 10% of the single stage implementation.

When operating in NFB mode, partial pipelining was found to offer the advantage of extremely high throughput rates once the pipeline latency was met, with the 5-stage partial pipeline implementation displaying the best throughput results. 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, 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. 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.

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 memory elements. Additionally, the S-Boxes map extremely well to the Xilinx CLB slice, which is comprised of 4-bit look-up-tables, allowing one S-Box to be implemented in a total of two CLB slices, yielding a compact implemen-tation which minimizes routing between CLB slices. Finally, the components of the Serpent round function were found to be simple in structure, resulting in the round function requiring few hardware resources. Implementing a single round of the Serpent algorithm provides the greatest area-optimized solution. A signifi-cant performance improvement was achieved by performing 8-round loop unrolling, removing the need for S-Box multiplexing hardware as one copy of each possible S-Box grouping is now included within one of the eight rounds. This amount of loop unrolling achieved a significant performance increase with little increase in hardware resources due to the compact nature of the Serpent round function. As expected, unrolling 32 rounds of the Serpent algorithm resulted in a lesser performance when compared to the eight round implementation. Implementing the 32 rounds of the algorithm in combina-torial logic severely hampered the overall clock frequency of the system, overriding the performance increase caused by the removal of the multiplexing hardware required to switch between keys.

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 implement-tations. 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, the cost of S-Box replication is minimal in terms of the required hardware. 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 performance degradation.

Twofish. When implementing the Twofish algorithm, 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. As expected, implementing a single round of the Twofish algorithm provides the greatest area-optimized solution in terms of total required CLB slices. Additional loop unrolling provided minor throughput increases as the decrease in the number of cycles per encrypted block was offset by the rapidly decreasing system clock frequency. However, single stage partial pipelining with one sub-pipeline stage was found to yield the best throughput and when operating in FB mode. With small increases in the required hardware resources, the sub-pipelined architecture was able to reach a significantly faster system clock frequency as compared to the loop unrolling and partial pipeline implementations.

When operating in NFB mode, a partially pipelined architecture was found to offer the advantage of extremely high throughput rates once the latency of the pipeline was met, with the 8-stage partial pipeline implementation displaying the best throughput results. While Twofish cannot be implemented using a fully pipelined architecture, significant throughput increases were seen as compared to the loop unrolling architecture.

Finally, 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. So, the 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.

6 PERFORMANCE EVALUATION

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 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:

$$\text{Throughput} := (128\text{Bits} * \text{ClockFrequency}) =$$
$$= (\text{CyclesPerEncrypted Block})$$

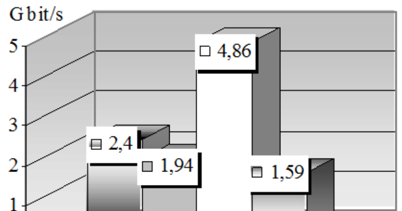

Figure 1 - AES finalist evaluation - best throughput | non-feedback mode speed optimized implementations

The required number of CLBs, as well as the maximum operating frequency for each implementation, was obtained from the Xilinx report files. While this study focuses on high throughput implementations, the hard-ware resources required to achieve this throughput is also a critical parameter. Two area measurements of FPGA utilization are readily apparent: logic gates and CLB slices. To measure the hardware resource cost associated with an implementation's resultant throughput, the ThroughputPerSlice (TPS) metric is used. TPS is:

TPS := (Encryption Rate)=(# CLB Slices Used)

The optimal implementation will display the highest throughput and will have the largest TPS.

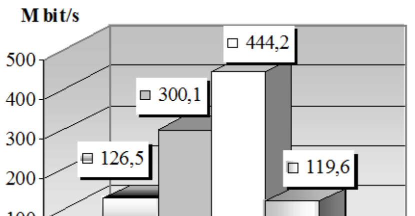

Figure 2 - AES finalist evaluation - best throughput | feedback mode speed optimized implementations

Figure 1 and Figure 2 detail the optimal implementations of the AES finalists in both FB and NFB modes. Additionally, TPS is also shown for each of the implementations. It is critical to note that for the purposes of this study, the optimal implementation for an AES finalist is defined to yield the highest throughput.

7 CONCLUSIONS

The Serpent algorithm clearly outperforms the other AES finalists in both modes of operation. As compared to its nearest competitor, Serpent exhibits a throughput increase of a factor 2.2 in NFB mode and a factor 1.5 in FB mode. Interestingly, RC6, Rijndael, and Twofish all exhibit similar performance results in NFB mode. However, Rijndael exhibits significantly improved performance in FB mode as compared to RC6 and Twofish, although it is still 50% slower than Serpent.

One of the main findings of our investigation, namely that Serpent appears to be especially well suited for an FPGA implementation from a performance perspective, seems especially interesting considering that Serpent is clearly not the fastest algorithm with respect to most software comparisons [7]. 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. These speedups are essentially achieved by pipelining and sub-pipelining of the loop structure and by wide operand processing, both of which are not feasible on current processors.

REFERENCES

1. C. Paar, “Optimized Arithmetic for Reed-Solomon Encoders”, in 1997 IEEE International Symposium on Information Theory, pp. 250, 1997.
2. B. Schneier, “Applied Cryptography”, John Wiley&Sons Inc., 2nd ed., 1995.
3. National Institute of Standards and Technology, Second Advanced Encryption Standard Conference, March 1999.
4. R. Anderson, E. Biham, and L. Knudsen, “Serpent: A Proposal for the Advanced Encryption Standard”, First Advanced Encryption Standard Conference, 1998.
5. E. Biham, “A Fast New DES Implementation in Software”, in Fast Software Encryption. 4thInternational Workshop, Proceedings, pp. 260-272, Springer-Verlag, 1997.
6. J. Daemen and V. Rijmen, “AES Proposal: Rijndael”, First Advanced Encryption Standard Conference, 1998.
7. B. Gladman, “Implementation Experience with AES Candidate Algorithms”, Second AES Conference, 1999.
8. J. Kaps and C. Paar, “Fast DES Implementations for FPGAs and its Application to a Universal Key-Search Machine”, in 5th Annual Workshop on Selected Areas in Cryptography, vol. LNCS 1556, Springer-Verlag, 1998.
9. E.Mang, I.Mang,”VLSI Implementation of the Rijndael Cipher”, Proceedings of Electronic, Computers and Informatics Conference, Slovacia, pp. 159-164, 2002.
10. R. Rivest, M. Robshaw, R. Sidney, and Y. Yin, “The RC6TM Block Cipher”, First AES Conference, 1998.
11. D. Runje and M. Kovac, “Universal Strong Encryption FPGA Core Implementation”, Proceedings of Design, Automation and Test in Europe, pp. 923-924, 1998.
12. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, and C. Hall, “Twofish: A 128-Bit Block Cipher”, First Advanced Encryption Standard Conference, 1998.

Надійшла до редакції 23 березня 2004р.