2Gbit/s Hardware Realizations of RIJNDAEL and SERPENT: A Comparative Analysis

A.K. Lutz[1], J. Treichler[1], F.K. Gürkaynak[2], H. Kaeslin[3],
G. Basler[1], A. Erni[1], S. Reichmuth[1], P. Rommens[1],
S. Oetiker[2], and W. Fichtner[2]

[1] Department of Information Technology and Electrical Engineering
ETH Zürich, CH-8092 Zürich Switzerland
{alutz, jtreichl, gbasler, aerni, sreichmu, prommens}@ee.ethz.ch
[2] Integrated Systems Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich, CH-8092 Zürich Switzerland
{kgf,oes,fw}@iis.ee.ethz.ch
[3] Microelectronics Design Center
ETH Zürich, CH-8092 Zürich Switzerland
kaeslin@ee.ethz.ch

Abstract. We present and evaluate efficient VLSI implementations of both Rijndael and Serpent. The two cipher algorithms have been implemented by two comparable design teams within the same timeframe using the same fabrication process and EDA tools. We are thus in a position to compare to what degree the Rijndael and Serpent ciphers are suitable for dedicated hardware architectures. Both ASICs support encryption as well as decryption in ECB mode and include on-chip subkey generation. The two designs have been fabricated in a 0.6μm 3LM CMOS technology. Measurement results verified an encryption and decryption throughput of 2.26Gbit/s and 1.96Gbit/s for Rijndael and Serpent respectively. Circuit complexity is in the order of 300k transistors in either case.

1 Introduction

While efficient hardware implementation was one of the evaluation criteria [3] of the Advanced Encryption Standard (AES), relatively few hardware designs with FPGAs have been presented [7,9,10] and even less so as ASICs [4,8]. There have been very few reports on hardware implementations [6] even after Rijndael was declared the AES standard. A detailed summary of the above mentioned implementations can be found in [11].

While FPGA based crypto-system solutions offer significant performance, especially for System-on-a-Chip (SoC) designs and large scale productions, customized ASIC modules of crypto-systems are indispensable. Our study focuses on actual ASIC implementations of the AES cipher Rijndael and the runner-up algorithm Serpent on silicon.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 144–158, 2003.

The hardware evaluation of the AES candidates by Weeks [4] and Ichikawa [8] are based on synthesis results only and are very general in nature. Rather than finding an optimum implementation for each of the implemeted algorithms, they concentrate on comparing all algorithms using a similar architectural approach. Area and speed estimations based on synthesis results may be used to compare different architectural choices, but not all architectures obtained by logic synthesis will lend themselves to physical design with the same efficiency.

In our study we have set strict limits on the maximum area, usable clock frequency range, number of parallel inputs and outputs and design time and formed two separate design teams with identical ASIC design experience to optimize the algorithms for maximum throughput. In section 2 we present an overview of the architectural options and define the solution space. Details on the hardware optimizations for the Rijndael cipher are given in section 3 while those of Serpent follow in section 4. Both circuits are then compared to each other in section 5 before section 6 presents our conclusions.

2 Common Design Issues

2.1 The Rijndael and Serpent Algorithms

Figure 1 shows the main algorithmic structure of Rijndael and Serpent for encrypting one block of data with 128bit keys. The names of the operations correspond to those described in [1,2]. Both algorithms can be seen as a succession of several transformation rounds which all data blocks have to undergo. Note that the initial and final rounds may be, depending on the algorithm, slightly modified versions of the regular transformation rounds. Each round uses at least one subkey derived from the user key. While Rijndael uses the same transformation round throughout, each Serpent round makes use of one out of eight different S-boxes.

	Rijndael	Serpent
Initial Transformation	AddRoundKey	
Round Transformation	9 x SubBytes, ShiftRows, MixColumns, AddRoundKey	31 x Key Mixing, S-Boxes, Linear Transformation
Final Transformation	SubBytes, ShiftRows, AddRoundKey	Key Mixing, S-Boxes, Key Mixing

Fig. 1. Algorithmic operations of Rijndael and Serpent

2.2 A Fair Comparison

For either of the two algorithms, a team of three was assigned the task to develop a VLSI circuit with the best throughput they could obtain from a die size of 50mm^2 in a 0.6μm 3LM CMOS technology. While such a hard bound on the circuit area limited the design space for both designs, it is typical for real-world applications. A large chip not only costs more to manufacture, but also suffers from a lower yield and higher parasitic interconnect capacitances.

Since a high throughput rate was desired, the work concentrated on designing a 128bit core that supports a simple ECB mode. It can be shown that the general architecture developed for the ECB mode (or simple derivations thereof) will lend itself to optimal implementations supporting other modes of operations such as CFB, CBC, OFB and CTR.

Both chips had to be designed to share the same pinout in a 144pin PGA package to ease testing and system integration. Also note that the number and rate of off-chip connections may account for a significant portion of the power budget of the design. Both designs feature a cryptographic core that operates on 128bit parallel data words internally. The external interface of both chips consists of three separate 32bit I/O channels for plaintext, ciphertext and the user key respectively. A separate I/O controller is used to schedule the data transfers from the 32bit external interface to the 128bit internal core.

Both design teams were given 14 weeks to complete the project. All team members were graduate students in EE and, hence, had very similar backgrounds and levels of expertise in IC design. Also, the EDA tools, cell libraries, computing resources and fabrication processes made available to them were the same. Incidentally, front-end design was carried out with tools by ModelSim and Synopsys while SiliconEnsemble by Cadence Design Systems was used for the back-end design. Cell library and chip fabrication on multi-project wafers were provided by austriamicrosystems (AMS). This arrangement has made it possible to compare the hardware realizations of the two cipher algorithms on a level ground.

2.3 Overall Architectural Choices

VLSI designers always strive to maximize hardware efficiency or, which is the same, to minimize the area-time-product (AT). Figure 2 illustrates the most prominent architectural transforms for arithmetic/logic hardware along with their impact on chip area and throughput.

Fairly small circuits are obtained from iteratively decomposing the computation such as to make it run on a single hardware round. Each data block must then be recycled through that datapath as many times as the cipher algorithm has transformation rounds. Extra control logic is required if one round computationally differs from the next such as in the occurrence of the Serpent cipher. Conversely, fast but large architectures result from mapping all transformation rounds into hardware directly followed by a generous addition of pipeline registers.

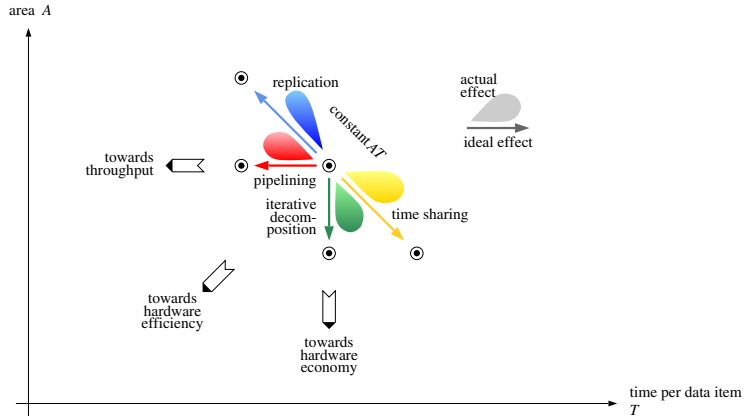

Fig. 2. Architectural transforms along with their impact (after [5]).

Many more architectures are situated somewhere in between these extremes. Their construction asks for a carefully balanced combination of pipelining, iterative decomposition, and possibly also replication of hardware units.

Key choices include:

- the number of rounds to instantiate in hardware,
- the degree of pipelining, that is number of registers per round,
- the organization of the datapath hardware, e.g. deciding on the optimum locations of [de]muxes and of pipeline registers, and
- the cycle-by-cycle schedule for the entire computation run.

The result of a simple analysis comparing the ECB throughput and datapath area for different architectural choices of both Serpent and Rijndael using the target 0.6μm technology can be seen in fig.3. The data for the graph has been compiled by synthesizing a single hardware round for each algorithm and extrapolating the performance based on the performance of this round. Note that this simplistic analysis only contains the hardware required for the rounds. Subkey generation and/or storage units, controllers and I/O circuitry are not included in this calculation. Additional effects like increased interconnection delay for larger designs, and clock distribution problems related to high clock rates have also not been considered. While the architectures selected for implementation are shown to have datapath areas of only 12mm^2 both implementations ended up being 50mm^2 in total, a fourfold increase.

In fig.3, a 14-round Rijndael cipher capable of running with a 256bit user key is considered. This architecture can be realized by instantiating 1, 2, 7 or all 14 rounds in hardware. Similarly for Serpent, realizations instantiating 1, 2, 4, 8, 16 or all 32 rounds have been considered. The lower part of the graph has been magnified in the inset. The inset roughly covers the solution space that was actually available in the context of our ASIC design project.

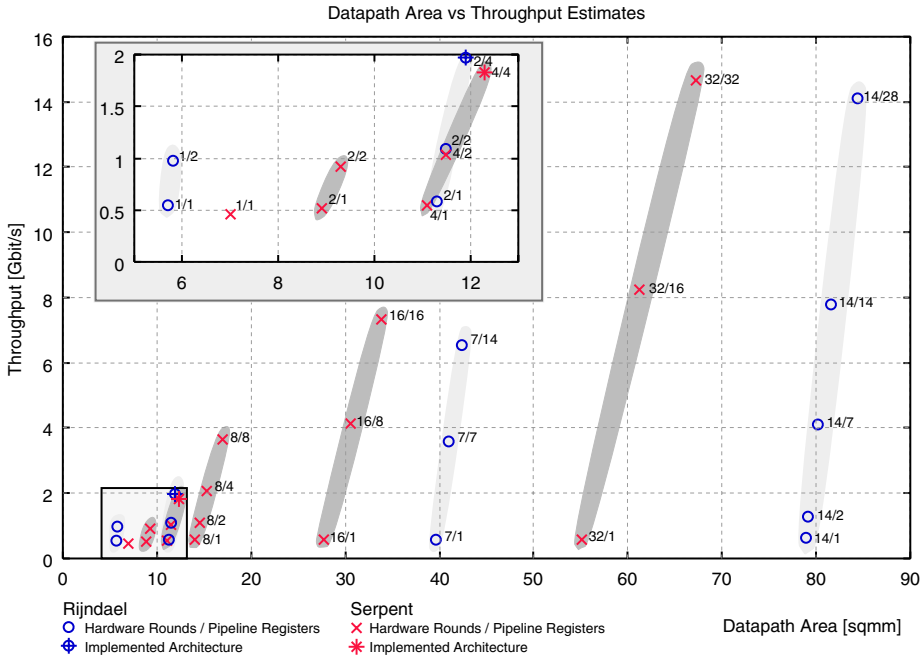

Fig. 3. Architectural choices for Rijndael and Serpent, throughput and datapath area estimations as a function of instantiated hardware rounds and pipeline stages for Rijndael and Serpent. The throughput estimations are for 128bit keys.

Subkey generation is another issue as both algorithms require several subkeys that must be derived from the user key. These subkeys are then applied to a data block while it moves through the individual rounds. One has the choice to compute the subkeys on the fly for each round or to generate all of them in advance before any data processing takes place, storing them in registers until needed. Depending on the number of subkeys, the chip area to be set aside for storage may be substantial.

The decryption operation places additional constraints on hardware. In its most basic form, both the ordering and function of the rounds must be reversed, so that the last encryption round is undone first during decryption. For a realization that does not store all the subkeys, the last subkey must thus be computed prior to the decryption operation.

Some cipher algorithms use reversible round structures where the same hardware can be used for encryption and decryption. As opposed to this, both Rijndael and Serpent rely on certain computational operations within their transformation rounds that require separate datapath elements for encryption and decryption, thereby increasing hardware complexity.

3 The Rijndael Implementation

3.1 Sharing Look-Up Tables between En- and Decryption

As illustrated by fig.1, each encryption round consists of four consecutive operations named SubBytes, ShiftRows, MixColumns and AddRoundKey. ShiftRows is a fixed permutation of the byte order and needs no extra circuitry. MixColumns can be implemented as a sequence of a few XOR gates while AddRoundKey is a simple XOR operation on all 128bits. The only operation that is onerous to implement in hardware is SubBytes.

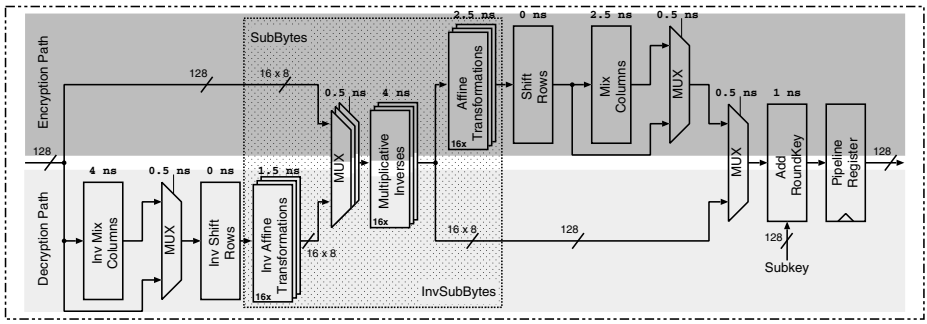

Fig. 4. A Rijndael round that shares LUTs and AddRoundKey function.

SubBytes consists of 16 concurrent S-box operations for which no more efficient solution than using 8bit×8bit look-up tables (LUT) seems to exist. The 16 LUT's account for about 85% of a round's combinational logic, so this was where to look for area reductions. Our idea was to find out whether the LUT's could somehow be shared between encryption and decryption in spite of the fact that the two operations are computationally different.

A Rijndael S-box is composed of two transformations [1]:

1. Take the multiplicative inverse in the finite field GF(2^8) with the element {00} being mapped onto itself.
2. Apply an affine transformation.

As opposed to the expensive LUT needed to implement the multiplicative inverse, the affine transformation is easily obtained from a few XOR gates.

The inverse S-box operation consists of the inverse affine transformation followed by the multiplicative inverse. Instead of using two separate LUT's, it is thus possible to compute both the S-box and the inverse S-box operation from a single LUT used in conjunction with either the affine or the inverse affine transformation, see the framed part of fig.4. The savings in the order of 30% to 50% of area so obtained eventually made it possible to instantiate two such hardware rounds on the chip, see fig.7, thereby almost doubling overall throughput.

3.2 Reorganizing a Cipher Round for Pipelining

As can be seen in fig.4, which also includes the propagation delays in the datapath, the longest path in this configuration is about 12ns. The next goal was to maximize throughput by recurring to intraround pipelining, that is by inserting pipeline registers into the datapath hardware of one round. The architecture of fig.4 is unsuitable for doing so because no location can be found for an extra register that would significantly cut down the longest path for both encryption and decryption.

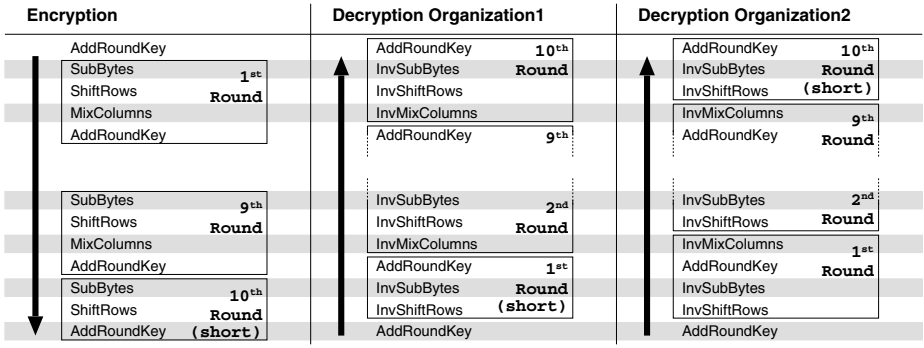

Fig. 5. The two options for delimiting one Rijndael decryption round.

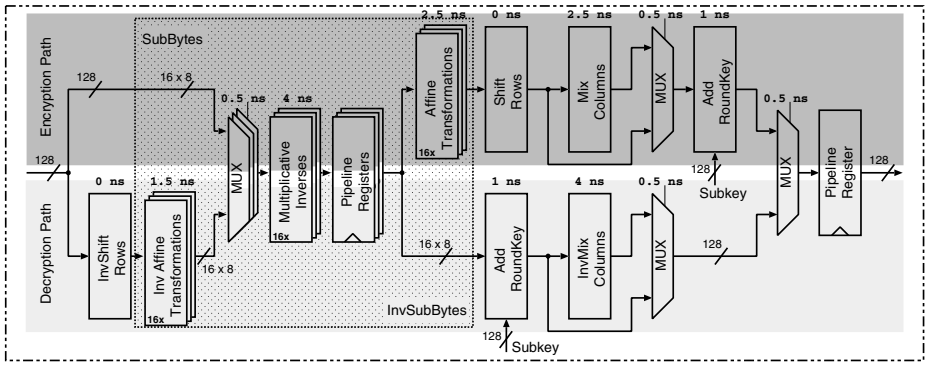

Fig. 6. A reorganized Rijndael round amenable to intraround pipelining.

Therefore, the hardware architecture of one round was reorganized without altering the circuit's functionality. Figure 5 displays two options for delimiting decryption rounds. The first option corresponds to fig.4 whereas the second option is shown in fig.6. This second option is amenable to pipelining while the

first is not. By inserting pipeline registers after the multiplicative inverse, the longest path is cut down from 12ns to 7ns thereby greatly improving throughput once more at little extra cost.

3.3 Precomputing Subkeys

The chosen architecture with two pipelined hardware rounds implies that a total of four 128bit data blocks are being processed concurrently at any given time. Computing the subkeys on the fly seemed no desirable option in this case because four subkey computation units would be required to provide the four different subkeys. Therefore all eleven subkeys are precomputed and stored in registers.

Also, working from precomputed subkeys avoids any key setup time when changing from encryption to decryption or vice versa. Each 128bit data block can thus either be encrypted or decrypted independently of the precedent block.

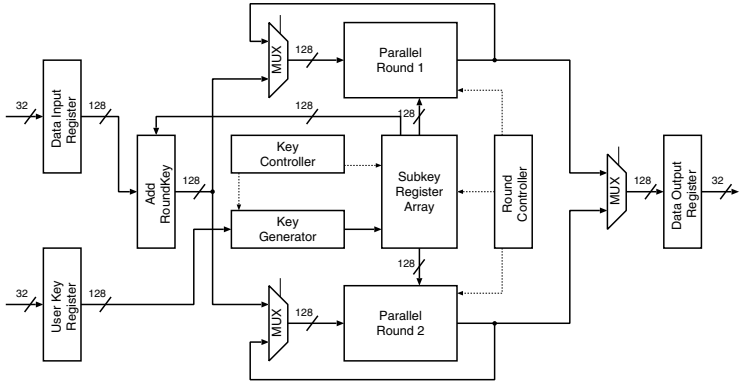

Fig. 7. Final architecture of the Rijndael chip.

4 The Serpent Implementation

4.1 Separating Decryption from Encryption

All of the 32 Serpent encryption rounds follow the same pattern, see fig.1. A key mixing operation where a 128bit subkey is XORed with the data block comes first, followed by an array of 32 parallel 4bit×4bit S-boxes. A subsequent linear transformation concludes the transformation round. Only in the final round is the linear transformation replaced by an extra key mixing operation with a 33rd subkey.

During decryption, both the S-boxes and the linear transformation need to be inverted. As opposed to the Rijndael algorithm, no method was found to reuse the same S-boxes for both encryption and decryption. This is no real

handicap because the much smaller size of the Serpent LUT's would hardly justify introducing multiplexers and other control hardware anyway. Only the relatively small key mixer might be reused. In this situation, it seemed more efficient to implement separate datapaths for encryption and decryption with no elements shared. The result is shown in fig.8.

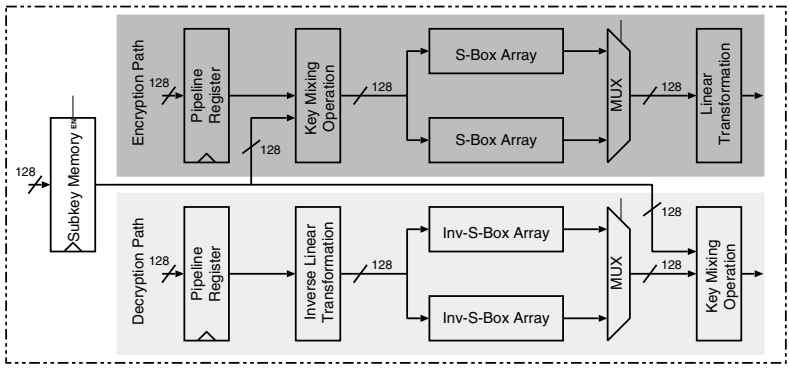

Fig. 8. A Serpent round with two separate datapaths.

4.2 Systematic Allocation of Hardware Resources

The basic solution space for the realization of the Serpent algorithm has been presented in fig.3. The computed figures suggested that a solution that includes a total of four hardware rounds followed by a pipeline register after each round was the best choice, considering that the maximum chip area of 50mm^2 was a hard limit.

As the Serpent algorithm makes use of eight different types of S-boxes, there is no way to avoid implementing all of them in hardware. Unless one can afford to instantiate eight or more hardware rounds, multiplexers and control logic must be included to switch look-up tables in and out depending on the cipher round currently being processed. This adds to the cumulative area occupied by one round and introduces extra data delay. The four hardware rounds are clearly visible in fig.9 which shows the chip's overall architecture.

4.3 Generating Subkeys on the Fly

Precomputing the full set of Serpent subkeys and storing them would require more than 4Kbit of memory which corresponds to an area of more than 3.6mm^2. This seeming impractical, it was decided to compute all subkeys on the fly concurrently with data processing.

Each round has a single associated subkey register, that stores the subkey to be applied to a data block at a certain time. Four consecutive data blocks

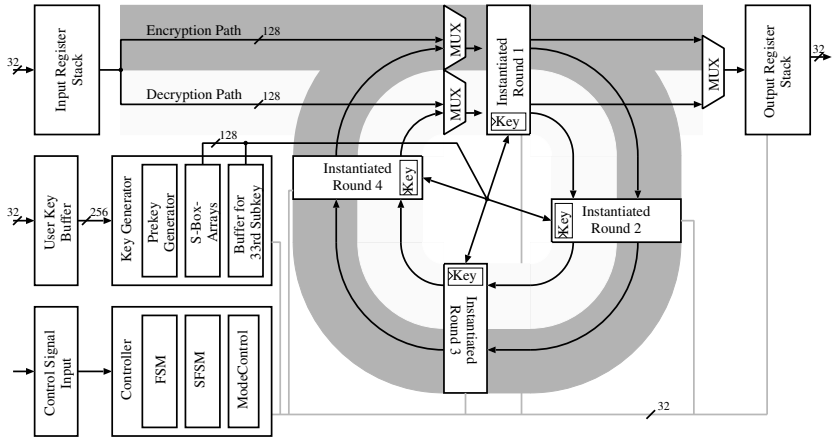

Fig. 9. Final architecture of the Serpent chip.

are processed one after the other in the same hardware round using the same subkey. Thus, the content of a subkey register can be applied four times in a row. As a consequence, three of the four existing rounds can always reuse the subkey stored within the local subkey register. One round, however, requires that a new subkey be delivered. This new subkey is supplied by a key generator that is capable of computing one new subkey during each clock cycle. After the last subkey has been prepared, the user key gets reloaded from an internal buffer to serve as starting point for computing the first round key again.

The last round of the Serpent algorithm presents a small problem as it uses an additional round key instead of the linear transformation. Since the key generator is not able to supply more than a single subkey per clock cycle, this 33rd subkey is computed ahead of time and stored in a register. Consequently, the key generator has to complete one full run to obtain the last round key prior to data processing, which results in a relatively long key setup time.

As the subkeys are not stored, the key generator also needs to be able to calculate the subkeys in reverse order. To accomplish this, once the encryption mode has generated the last two subkeys, a second, parallel, key generation unit is used to generate the subkeys in reverse order.

5 The Two Integrated Circuits Compared

Figure 10 shows the floorplans of the two VLSI chips while table 1 compares their key technical characteristics. Recurring to a multiproject wafer (MPW) service, both the Rijndael and the Serpent designs have been fabricated in prototype quantities. All measured figures in table 1 refer to the physical circuits so obtained. In either case, throughput figures in the order of 2Gbit/s have been obtained in a mature 0.6μm technology.

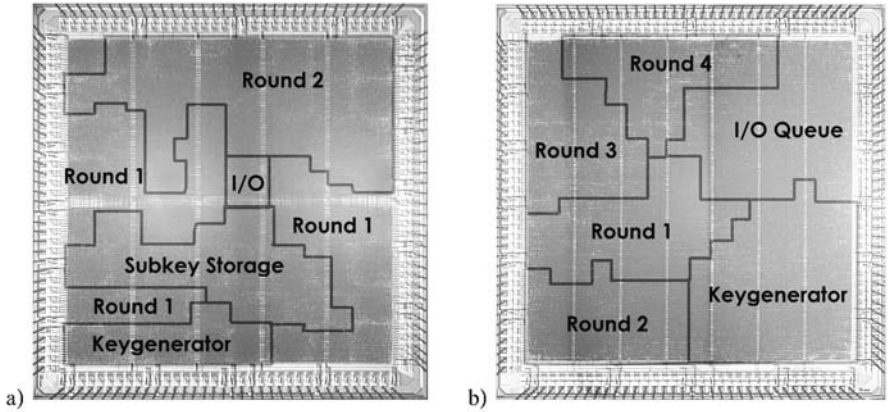

Fig. 10. The photomicrographs of a) Rijndael and b) Serpent with their main components highlighted.

While the Serpent circuits were quickly found to work correctly, the Rijndael chips exhibited a systematic malfunction. The problem has eventually been traced back to a mistake in the postprocessing of layout data for MPW assembly. A couple of lines running on the topmost metal have inadvertently been shorted together.

Coincidentally, only one of the two hardware encryption/decryption rounds of fig.6 has been afflicted. This made it possible to work around the defect and to verify the correct operation of the second round. The throughput figure presented for Rijndael is the throughput figure for the design with both parallel rounds working at the clock rate at which the second round was shown to be working.

The various functional blocks have been identified in the physical layouts of the two chips. In either design, the round controller occupies a relatively small area and is not particularly marked. Also note that approximately half of the large I/O queue region shown on the Serpent floorplan (fig.10b) includes subcircuits from other functional blocks.

The Serpent key generator can be seen to occupy a larger area than its Rijndael counterpart. This is mainly because all eight separate 4-bit S-box variants of the Serpent cipher need to be instantiated 32 times in order to calculate the subkeys on the fly.

In either floorplan, a significant proportion of the core area is lost to routing overhead. Also observe that Synopsys area estimations are off by a factor of two. In our opinion, the reasons for this poor area utilization are very wide data words in conjunction with a target technology that provides just three layers of metal, and standard cells with overly many routing blockages.

In theory, a more aggressive pipelining strategy should result in still higher throughput rates. Yet, we felt that clock frequencies much beyond 100MHz in

Table 1. Our Rijndael and Serpent implementations compared.

	Rijndael	Serpent
rounds in the algorithm	10	32
rounds instantiated in hardware	2	4
key length	128bit	256bit
subkey computation	stored	on the fly
core key agility [clock cycles (ns)]		
new encryption key	3 (34)	21 (171)
new decryption key	23 (260)	21 (171)
switch between encryption and decryption	0 (0)	21 (171)
number of flip-flops	2'607	3'274
number of transistors	300k	300k
technology	0.6μm 3LM	0.6μm 3LM
process name	AMS CUA	AMS CUA
area per hardware round	6.3mm^2	3.1mm^2
area for subkey generation	4.5mm^2	3.8mm^2
estimated chip area (after synthesis)	22.5mm^2	21.6mm^2
actual chip area (after physical layout)	49.0mm^2	49.0mm^2
data throughput in ECB mode (encr or decr)	2.26Gbit/s	1.96Gbit/s
@ clock frequency	88.5MHz	122.9MHz
latency [clock cycles (ns)]	28 (316)	56 (455)

our target technology would give rise to significant difficulties with off-chip data transfer and with clock distribution.

Both designs have been balanced for encryption and decryption and indeed achieve similar throughput rates for either operation. The Rijndael implementation sports remarkably short key setup times and, most notably, does not require any additional setup time when switching between encryption and decryption with the same user key. The Serpent circuit's key setup time, on the other hand, suffers from the necessity to calculate the last subkey in advance.

The core circuits have been designed to run with 128bit data but practical considerations have limited input and output width to 32bit. Nevertheless, no performance is lost thanks to careful scheduling of I/O operations.

The original AES specification calls for three key lengths of 128, 192 and 256bit respectively. To simplify design, only 128bit implementations have been considered in this study. The Serpent design is capable of using key lengths of up to 256bit without any modification, whereas the Rijndael circuit would need to be adapted to accommodate multiple key lengths.

The throughput figures presented refer to the ECB mode of operation. Of the various feedback modes proposed for AES use [12], only the CTR mode will achieve similar data throughput. For CFB, OFB and CBC modes of operation without interleaving the highest throughput per area can be obtained if and only if an iterative architecture with only a single hardware round is implemented without any pipelining. This solution is on the bottom left corner of the solution

space presented in fig.3 and is expected to have resulted in a throughput in the order of 500Mbit/s for both algorithms.

6 Conclusions

To begin with, note that the two ciphers have many traits in common, both lend themselves fairly well for hardware implementation. Most importantly, there are no feedback loops whatsoever in ECB and CTR mode that would present unsurmountable bottlenecks when in search of maximum throughput. A number of tricks are instrumental in turning a purely algorithmic prescription into a highly efficient architecture, but this is common practice in VLSI design.

We have made valuable contributions towards designing optimum hardware for Rijndael by relocating the boundary between two consecutive rounds and by restating table look-up operations. Both designs benefit from the evaluation of distinct subkey generation schemes and from the systematic exploration of architectural trade-offs.

Table 2 compares the two cipher algorithms from the perspective of a VLSI architect. The fact that the same S-box LUT can be reused for encryption and decryption throughout all rounds probably is the most important advantage of Rijndael. Conversely, the fact that the number of rounds and subkey generation are dependent on the width of the user key are less desirable features.

Table 2. The two ciphers compared from a VLSI architect's point of view.

Rijndael	Serpent
+Small number of rounds (10...14). +Small number of subkeys (11...15). −No. of rounds depends on key length.	−Large number of rounds (32). −Large number of subkeys (33). +Fixed number of rounds.
+No complex mathematical operations. +All rounds are identical (same S-box type throughout). −Large S-box (8bit×8bit).	+No complex mathematical operations. −Eight different S-box types. +Small S-boxes (4bit×4bit).
−Cipher is not involutory. +Look-up tables can be made to share between en- and decryption. ∘ Not all hardware components can be shared between en- and decryption.	−Cipher is not involutory. −All S-boxes and their inverses must be implemented in hardware. −Almost no hardware components can be shared between en- and decryption.
−Key generation varies with key width.	+Wider key entails no extra complexity. −No efficient way to compute the 33rd subkey from the user key directly.

To our knowledge this is the only published study where the actual ASIC implementations of two AES candidates have been compared. In this respect this study differs from previous comparisons [4,8] and realizations [6], as it also takes into account real-life problems of ASIC integration such as placement and routing, interconnection parasitics, clock distribution and I/O limitations.

In [4] two extreme cases are considered: for an iterative architecture Rijndael was shown to have three times the throughput of Serpent with an estimated area 1.5 times larger. For a fully pipelined architecture, with almost identical area requirements the throughput of Serpent was thirty percent higher than Rijndael. In another study that concentrated on finding the critical path for feedback modes [8] Rijndael was found to be more than twice as fast as Serpent with an estimated area approximately twenty percent larger.

A direct comparison of our Rijndael implementation to the one presented in [6] is difficult as that implementation uses a much more advanced 0.18μm technology, estimated to be almost 10 times smaller and faster than the 0.6μm technology used in this study. Also the implementation in [6] does not support decryption and the stated throughput of 1.82Gbit/s is for 256bit data blocks. In this respect our implementation with a measured 2.26Gbit/s throughput using 128bit data blocks compares fairly well.

Considering that the two algorithms are rather different in nature, their respective performances in hardware come remarkably close. From our experience with designing circuits for a fixed key length of 128bit and for throughputs in the order of a few Gbit/s, we consider Rijndael to be more favorable than Serpent, although only slightly.

References

1. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. Submission to the First Advanced Encryption Standard Candidate Conference. Ventura CA, August 1998.
2. Anderson, R., Biham, E., Knudsen, L.: Serpent: A Proposal for the Advanced Encryption Standard. Submission to the First Advanced Encryption Standard Candidate Conference. Ventura CA, August 1998.
3. Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., Roback, E.: Report on the Development of the Advanced Encryption Standard (AES). NIST, Computer Security Division, Information Technology Laboratory, October 2000.
4. Weeks, B., Bean, M., Rozylowicz, T., Ficke, C.: Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms. National Security Agency (NSA).
5. Kaeslin, H.: From Algorithms to Architectures. Lecture Notes in VLSI Design, Microelectronics Design Center, ETH Zürich.
6. Kuo, H., Verbauwhede, I.: Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the Rijndael Algorithm. Proceedings of the Third International Workshop of Cryptographic Hardware and Embedded Systems CHES 2001, Paris, May 2001.
7. Elbirt, A., Yip, W., Chetwynd, B., Paar, C.: An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. IEEE Transactions on VLSI, August 2001, vol. 9, no. 4, 545–557.
8. Ichikawa, T., Kasuya, T., Matsui, M.: Hardware Evaluation of the AES Finalists. Proceedings of the Third Advanced Encryption Standard Candidate Conference, New York, April 2000, 279–285.
9. Gaj, K., Chodowiec, P.: Comparison of the hardware performance of the AES candidates using reconfigurable hardware. Proceedings of the Third Advanced Encryption Standard Candidate Conference, New York, April 2000, 40–54.

10. Weaver, N., Wawrzynek, J.: A Comparison of the AES Candidates Amenability to FPGA Implementation. Proceedings of the Third Advanced Encryption Standard Candidate Conference, New York, April 2000, 28–39.
11. Gaj, K., Chodowiec, P.: Hardware performance of the AES finalists – survey and analysis of results. Technical Report, George Mason University, September 2000.
12. Dworkin, M.: Recommendation for Block Cipher Modes of Operation – Methods and Techniques. NIST Special Publication 800-38A, 2001.