

Design and Performance Testing of a 2.29-GB/s Rijndael Processor

Ingrid Verbauwhede, *Senior Member, IEEE*, Patrick Schaumont, *Student Member, IEEE*, and Henry Kuo

Abstract—This contribution describes the design and performance testing of an Advanced Encryption Standard (AES) compliant encryption chip that delivers 2.29 GB/s of encryption throughput at 56 mW of power consumption in a 0.18- μ m CMOS standard cell technology. This integrated circuit implements the Rijndael encryption algorithm, at any combination of block lengths (128, 192, or 256 bits) and key lengths (128, 192, or 256 bits). We present the chip architecture and discuss the design optimizations. We also present measurement results that were obtained from a set of 14 test samples of this chip.

Index Terms—Advanced Encryption Standard (AES), application-specific integrated circuit (ASIC), processor, Rijndael, secret key cryptography, very-large-scale integration (VLSI).

I. INTRODUCTION

WE developed a high-throughput Rijndael encryption [1] processor with customized target architecture. It implements a superset of the Advanced Encryption Standard (AES) encryption standard [2]. At peak performance, our processor achieves 2.29 Gb/s of encryption throughput at 56 mW of power consumption. The processor is programmable and supports Rijndael in any combination of key lengths (128, 192, or 256 bits) and block lengths (128, 192, or 256 bits). It is integrated into the host platform through a 16-bit data bus and operates with a small instruction set. The implementation uses 173K gates in a 1.8-V 0.18- μ m CMOS standard cell technology, and has been verified operational at up to 154-MHz clock frequency in a prototype setup.

In Sections II and III, we will review the Rijndael algorithm and present the processor architecture. This includes a discussion at system level and at block level, as well as a review of the design optimizations that have been applied. The chip has been implemented and verified in a test setup. The schmo plots and power consumption figures in this paper present, to our knowledge, the first verified application-specific integrated-circuit (ASIC) implementation of the Rijndael encryption algorithm.

II. RIJNDAEL CIPHER

The Rijndael encryption algorithm [1] is a block cipher that converts cleartext data blocks of 128, 192, or 256 bits into ciphertext blocks of the same length. It uses a key of selectable length (128, 192, or 256 bits). The encryption algorithm is organized as a set of iterations called *round transformations*, as

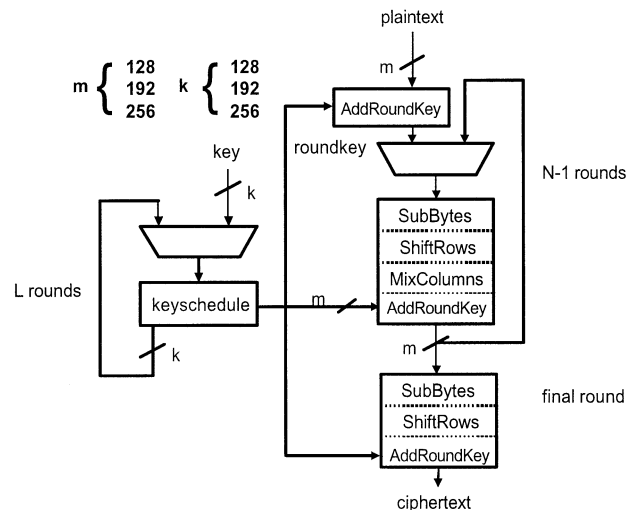


Fig. 1. Rijndael encryption.

illustrated in Fig. 1. In each round, a data block is transformed by a series of operations. The total number of rounds is dependent on the largest of data block length m or key length k , and equals 10, 12, and 14 for lengths of 128, 192, and 256 bits, respectively. All round transformations are identical, apart from the final one.

The operations performed during a round transformation include the following.

- **AddRoundKey** performs modulo-2 addition of a roundkey with a data block.
- **SubBytes** replaces each byte of the data block with the S-box lookup table value of that byte. The contents of an S-box is the multiplicative inverse in Galois Field (GF) (2^8), combined with a permutation affine over GF(2).
- **ShiftRows** organizes the data block in a four-row row-major ordered matrix and circularly shifts each row over a parameter and row-index dependent shift.
- **MixColumns** uses the same four-row matrix organization but transforms each column of the matrix by multiplying it with a constant GF polynomial.

III. CHIP DESIGN AND IMPLEMENTATION

The system architecture of our implementation is shown in Fig. 2. The central block of the architecture, **encrypt**, implements one round of a Rijndael encryption in a fully parallel non-pipelined fashion. This Rijndael encryption can be completed at one clock cycle per round.

The processor has three controllers, two for input/output (I/O) interfacing and one for instruction sequencing. They communicate through request/acknowledge protocols with the host system. This asynchronous interfacing method allows the chip to be clocked much faster than the bus it is connected to.

Manuscript received April 17, 2002; revised October 24, 2002. This work was supported by the National Science Foundation and by the University of California Microelectronics Innovation and Computer Research Opportunities (UC Micro).

I. Verbauwhede and P. Schaumont are with the Department of Electrical Engineering, University of California, Los Angeles, CA 90095 USA (e-mail: ingrid@ee.ucla.edu).

H. Kuo is with Atmel Corporation, San Jose, CA 95131 USA.

Digital Object Identifier 10.1109/JSSC.2002.808300

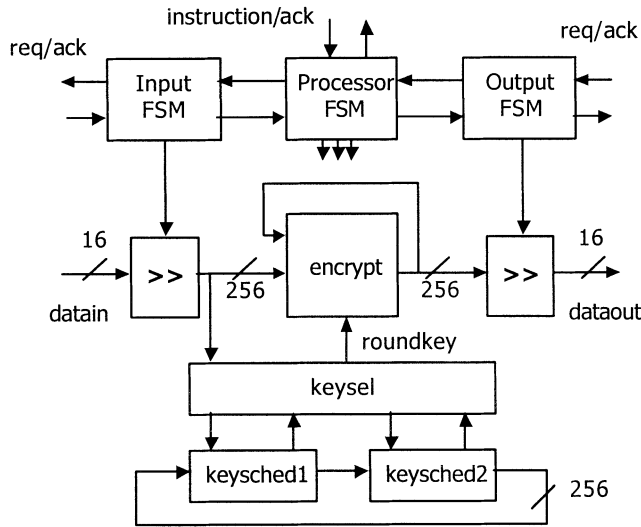


Fig. 2. System architecture.

It also brings considerable simplification of the performance testing process. Separation of I/O controllers from the instruction sequencing controller makes this block easily portable as an IP block to a different context, where a different databus or I/O interface protocol would be used.

A. Encryption Datapath

During encryption, the data is organized conceptually in a 4×8 matrix of bytes, totaling 32 bytes. This organization is used for data block sizes of 256 bits. For smaller data block sizes (128 or 192 bit), the leftmost columns of the matrix are unused. The encryption datapath processes a full 32-byte block in parallel. A complete round transformation executes in a single clock cycle. Each transformation (AddRoundKeys, SubBytes, ShiftRows, MixColumns) was optimized appropriately for maximal performance.

The SubBytes transformation relies on S-box lookup tables. Since all bytes are processed in parallel, we need 32 lookup tables (of 256 8-bit entries). Their optimization will be discussed further later.

The ShiftRows block transformation can be expressed as a rearrangement of the matrix using an address expression for each element. The address expressions calculate row-dependent circular shifts of the rows of a 4-, 6-, or 8-column matrix, depending on a block length of 128, 192, or 256 bits, respectively. These expressions then can be optimized using code hoisting and constant propagation techniques [1], [10].

The GF multiplication in MixColumns uses a constant as one operand. This constant multiplication is converted to an XOR operation [1], [10].

While a complete round transformation is completed within one clock cycle, careful design and optimization allows obtaining a critical path of 6 ns (typical) in 0.18- μ m technology. The overall critical path of the design is 10 ns (typical) and resides in the key-schedule block.

B. Key Scheduling

The architecture can support a chosen combination of key length k and data-block length m . Therefore, the number of

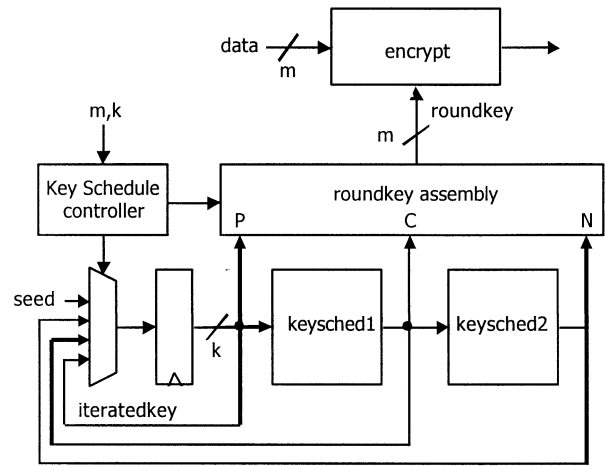


Fig. 3. Key-scheduling architecture.

key-schedule iterations L and the number of round transformations N can be different. We also want to have a constant encryption rate of one round transformation per clock cycle. Consequently, the speed of the key-scheduling process must be adapted as k and m change. Depending on the parameter values, we have to complete 0, 1, or 2 of the L key-scheduling iterations per clock cycle in order to keep up with the pace of one round transformation per cycle. For example, when 256-bit data blocks and 128-bit subkeys are needed ($m = 256$, $k = 128$), then two key-schedule iterations are needed for each data block. Double-rate key scheduling is only a worst case situation, but nonintegral rates can also occur. The combination of 192-bit data blocks with 128-bit keys ($m = 192$, $k = 128$), for example, requires 1.5 key-schedule iterations per data block. The implementation of these nonintegral rates requires special architecture support. Fig. 3 shows the key-scheduling architecture in more detail.

The key scheduling has to provide one m -bit *roundkey* per clock cycle to the encrypt block. The roundkey has to be constructed out of k -bit subkeys. When m is larger than k , multiple subkeys are required within one clock cycle. Two key-scheduling blocks *keysched1* and *keysched2* allow evaluation of two iterations of the key scheduling. The m -bit roundkey is assembled out of k -bit subkeys P , C , N (*Previous*, *Current*, and *Next* key-schedule iteration, respectively). This assembly is under control of a key-schedule controller. This controller also steers the pace of the key-schedule iterations by selecting which subkey is used as iterated key. With the key P , the key schedule does not advance. With the key C , a single iteration per clock cycle is taken, and with the subkey N , two iterations are taken per clock cycle.

C. S-Box Design

Because of the high amount of parallelism in our architecture, most tables are instantiated multiple times. The S-box table requires 32 instances in the encryption unit and 16 instances in the key-scheduling part. This makes area optimization of a single S-box instance an important factor in control of the overall chip area. The factors controlling the size of an S-box are a combination of the design and the logic-synthesis effort. The design is specified as the contents of an S-box, which is

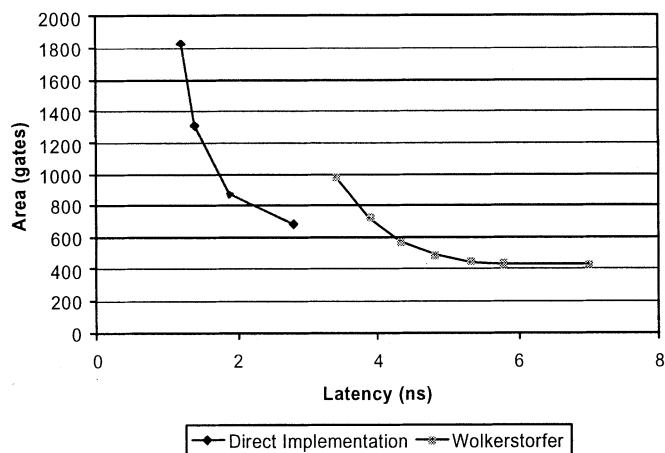


Fig. 4. Area-speed tradeoff for S-box.

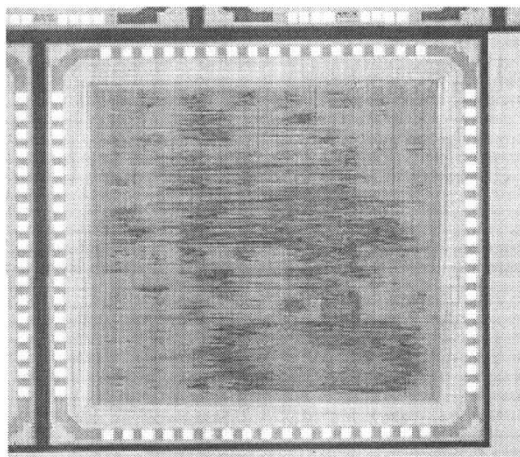


Fig. 5. Chip micrograph.

mathematically well defined as the combination of a $GF(2^8)$ multiplicative inverse combined with a permutation, affine over $GF(2^8)$ [2]. These mathematical properties have been used by Wolkerstorfer [5] and others to formulate smaller S-box implementations by expressing the elements of the single-field $GF(2^8)$ as a polynomial of elements of the smaller field $GF(2^4)$. In our case, however, it is equally important to have also the *fastest* S-box implementation. Fig. 4 illustrates the area-latency tradeoff curves as obtained out of logic synthesis with $0.18\text{-}\mu\text{m}$ standard cells for 1) a direct implementation and 2) the area-optimized implementation presented by Wolkerstorfer [5]. The fastest implementation is the direct one, and is three times better than the fastest Wolkerstorfer table. Since our primary concern was latency performance, we opted for the fastest direct implementation. Using the same constraint considerations, a similar conclusion can also be reached using figures from other authors [6].

D. Related Work

Both academia [3], [8], [9] and industry seem to have focused on the design of Rijndael cores in reconfigurable hardware. These implementations show that contemporary reconfigurable

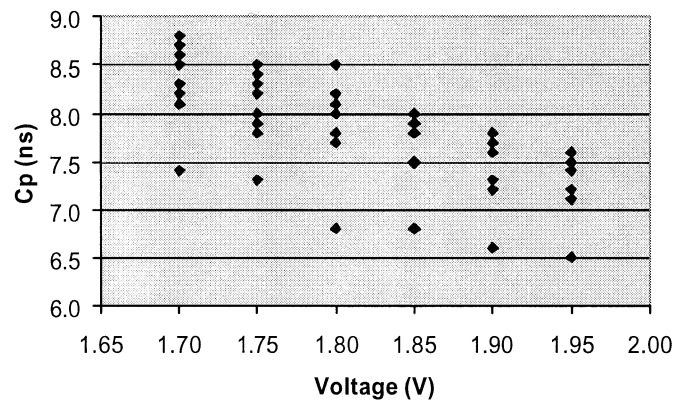


Fig. 6. Schmo plot: voltage versus critical path (CP).

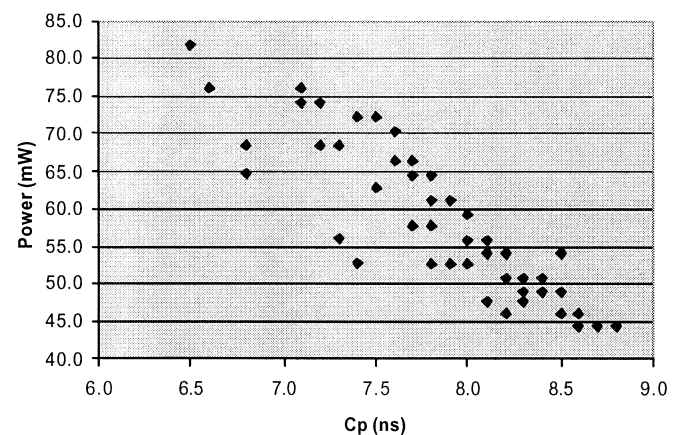


Fig. 7. Power consumption versus critical path (CP).

platforms with their rich distributed memory architecture are well suited for Rijndael prototype implementations. Most of them use precomputed subkeys. For pipelined ciphers, very high performances have been obtained. For nonpipelined implementation however, the throughput is considerably lower, in the order of 1 Gb/s. Unfortunately, no power consumption figures have been published up to now. Our own estimations show that our implementation is at least ten times more energy efficient than a commercial field-programmable gate-array (FPGA) implementation, and three orders of magnitude more energy efficient than a commercial performance-optimized software implementation [7]. Several ASIC designs have been presented as well [10], [11], but none of them was a verified and tested chip design.

E. Implementation and Test

The chip was implemented with a standard Hardware Description Language (HDL) design flow. The chip was processed by National Semiconductor Corporation. Of the 16 test samples that were received from the foundry, 14 were operational. Measured key performance metrics are shown in Table I. Fig. 5 shows the die micrograph. A test setup was made using standard off-the-shelf equipment. Using the 14 operational samples, we created schmo plots as well as power consumption plots, shown in Figs. 6 and 7, respectively.

TABLE I
KEY PERFORMANCE METRICS

| Feature | Value | Unit |
|------------------------|----------------|-----------------|
| CMOS Stcell Technology | 0.18 | um |
| Core Voltage (VDD) | 1.8 | V |
| Core Area | 3.96 | mm ² |
| IO Ring (VDDIO) | 3.3 | V |
| Package | 68-pin PLCC | |
| Nominal Clock Period | (VDD = 1.95 V) | 6.5 |
| | (VDD = 1.9 V) | 7.5 |
| | (VDD = 1.8V) | 8 |
| Power Consumption | (VDD = 1.95 V) | 82 |
| | (VDD = 1.9 V) | 67 |
| | (VDD = 1.8V) | 56 |
| Gate Count | 173 | Kgate |

| Throughput (Gbit/s) | Data/Key | 128 | 192 | 256 |
|---------------------|----------|------|------|------|
| | 128 | 1.6 | 1.33 | 1.14 |
| | 192 | 2.0 | 2.0 | 1.71 |
| | 256 | 2.29 | 2.29 | 2.29 |

IV. CONCLUSION

In this contribution, we presented a 173K-gate Rijndael encryption core that was verified at 2.29-GB/s encryption speed. The architecture was designed for best performance over several different cryptographic modes and several different host systems. The AES standard [2] has fixed the datablock length to 128 bits only. This can be used to optimize our architecture further: the key scheduling can be simplified and the encryption datapath can be cut in half. Consequently, we can reduce the gate count as well as the critical path to half the current amount. We are currently extending the architecture to include various modes of operation on-chip and to integrate it in a number of applications.

ACKNOWLEDGMENT

The authors would like to acknowledge National Semiconductor Corporation for chip processing.

REFERENCES

- [1] J. Daemen and V. Rijmen, *The Design of Rijndael: AES—The Advanced Encryption Standard*. New York: Springer-Verlag, 2002.
- [2] NIST Federal Information Processing Standards (FIPS) PUB 197 Advanced Encryption Standard (2001, Nov.). [Online]. Available: <http://www.nist.gov/aes/>
- [3] M. McLoone and J. McCanny, "Single-chip FPGA implementation of the advanced encryption standard algorithm," in *Proc. 11th Int. Conf. Field-Programmable Logic and Applications (FPL 2001)*, LNCS 2147, 2001, pp. 152–161.
- [4] H. Kuo and I. Verbauwhede, "Architectural optimization for a 1.82 Gb/s VLSI implementation of the AES Rijndael algorithm," in *Proc. 3rd Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS 2162, Paris, France, May 2001, pp. 51–64.
- [5] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC implementation of the AES S-boxes," in *Proc. RSA Conf. 2002*, San Jose, CA, Feb. 2002, pp. 67–78.
- [6] U. Mayer, C. Oelsner, and T. Keihler, "Evaluation of different Rijndael implementations for high-end servers," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 2, 2002, pp. 348–351.
- [7] H. Lipmaa. AES candidates: A survey of implementations. Lab. Theoretical Comput. Sci., Dept. Comput. Sci. Eng., Helsinki Univ. Technol., Helsinki, Finland. [Online]. Available: <http://www.tcs.hut.fi/~helger/aes/rijndael.html>
- [8] A. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Trans. VLSI Syst.*, vol. 9, pp. 545–557, Aug. 2001.
- [9] P. Chodowiec, K. Gaj, P. Bellows, and B. Schott, "Experimental testing of the Gigabit IPsec-compliant implementations of Rijndael and triple DES using SLAAC-1V FPGA accelerator board," in *Proc. Information Security Conf.*, LNCS 2200, Malaga, Spain, 2001, p. 220.
- [10] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," in *Proc. ASIACRYPT 2001*, LNCS 2248, 2001, pp. 239–254.
- [11] T. Ichikawa, T. Kasuya, and M. Matsui, "Hardware evaluation of the AES finalists," in *Proc. 3rd AES Candidate Conf.*, New York, Apr. 2000, pp. 279–285.