Design and Implementation of Low-area and Low-power AES Encryption Hardware Core

Panu Hämäläinen, Timo Alho, Marko Hännikäinen, and Timo D. Hämäläinen
Tampere University of Technology / Institute of Digital and Computer Systems
P. O. Box 553, FI-33101, Tampere, Finland
{panu.hamalainen, timo.a.alho, marko.hannikainen, timo.d.hamalainen}@tut.fi

Abstract

The Advanced Encryption Standard (AES) algorithm has become the default choice for various security services in numerous applications. In this paper we present an AES encryption hardware core suited for devices in which low cost and low power consumption are desired. The core constitutes of a novel 8-bit architecture and supports encryption with 128-bit keys. In a 0.13 μm CMOS technology our area optimized implementation consumes 3.1 kgates. The throughput at the maximum clock frequency of 153 MHz is 121 Mbps, also in feedback encryption modes. Compared to previous 8-bit implementations, we achieve significantly higher throughput with corresponding area. The energy consumption per processed block is also lower.

1. Introduction

Cryptographic algorithms are utilized for security services in various environments in which low cost and low power consumption are key requirements. Wireless Local Area Networks (WLAN), Wireless Personal Area Networks (WPAN), Wireless Sensor Networks (WSN), and smart cards are examples of such technologies. Compared to software, significantly higher performance and lower power consumption can be achieved with dedicated hardware. As Advanced Encryption Standard (AES) [11] is a standardized encryption algorithm and considered secure, it has become the default choice in numerous applications, including the standard wireless technologies IEEE 802.11i [9], IEEE 802.15.4 [8], and ZigBee [15]. In this paper we present an Application Specific Integrated Circuit (ASIC) AES encryption core suited for low-cost and low-power devices.

Although pipelined and loop-unrolled hardware architectures enable very high-speed AES designs, they also imply large area and high power consumption. Furthermore, these architectures cannot be fully exploited in feedback modes of operation [6, 7]. Feedback modes are often used for encryption and/or Message Authentication Code (MAC) generation, e.g. as in the security schemes of [8, 9, 15]. Instead, iterative architectures, which consist of a single computational unit that is repeatedly fed with its own output [7, 14], enable low-resource implementations with full-speed utilization also in feedback modes. The width of the data path can be further reduced to decrease the logic area and power [2–7, 12, 14].

Our iterative AES core has a 8-bit (i.e. byte) data path and it is capable of encrypting with 128-bit keys. For example, the operation modes of [8, 9, 15] require only the 128-bit-key forward functionality of AES. This saves hardware resources since the forward and inverse functionalities of AES are partially different. The core derives from our previous work [10], which presents efficient structures for storing intermediate results during AES processing. According to our knowledge, only two 8-bit AES ASICs have previously been reported in the academic literature [4, 5]. These are also the most compact AES ASICs. The other 8-bit designs have been implemented in Field Programmable Gate Arrays (FPGA) [3, 6]. Compared to the previous 8-bit ASICs [4, 5], our core achieves considerably higher throughput and lower cycle count with corresponding area. Our cycle count is also lower than that of the 8-bit FPGA implementations [3, 6].

The paper is organized as follows. Section 2 presents an overview of AES. In Section 3 we review the previously reported compact AES implementations. Section 4 describes the architecture of our AES core. Section 5 presents the implementation results and compares the results with previous implementations. Section 6 concludes the paper.

2. AES Overview

AES [11] is a symmetric cipher that processes data in 128-bit blocks. It supports key sizes of 128, 192, and 256 bits and consists of 10, 12, or 14 iteration rounds, respectively. Each round mixes the data with a *roundkey*, which is

Proceedings of the 9th EUROMICRO Conference on Digital System Design (DSD'06)
0-7695-2609-8/06 $20.00 © 2006 IEEE

Authorized licensed use limited to: Purdue University. Downloaded on April 4, 2009 at 14:15 from IEEE Xplore. Restrictions apply.

generated from the encryption key. Decryption inverts the iterations resulting in a partially different data path.

The encryption round operations are presented in Fig. 1. The cipher maintains an internal, 4-by-4 matrix of bytes, called *State*, on which the operations are performed. Initially State is filled with the input data block and XORed with the encryption key. Regular rounds consist of operations called *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. The last round bypasses MixColumns.

SubBytes is an invertible, nonlinear transformation. It uses 16 identical 256-byte substitution tables (*S-box*) for independently mapping each byte of State into another byte. S-box entries are generated by computing multiplicative inverses in *Galois Field* GF(2^8) and applying an affine transformation. SubBytes can be implemented either by computing the substitution [1,4–6,12,14] or using table lookups [2, 3, 7, 12]. ShiftRows is a cyclic left shift of the second, third, and fourth row of State by one, two, and three bytes, respectively. MixColumns performs a modular polynomial multiplication in GF(2^8) on each column. Instead of computing separately, SubBytes and MixColumns can also be combined into large Look-Up-Tables (LUT), called *T-boxes* [6, 13]. During each round AddRoundKey performs XOR with State and the roundkey. Roundkey generation (*key expansion*) includes S-box substitutions, word rotations, and XOR operations performed on the encryption key. For more details on the AES algorithm we refer to [11].

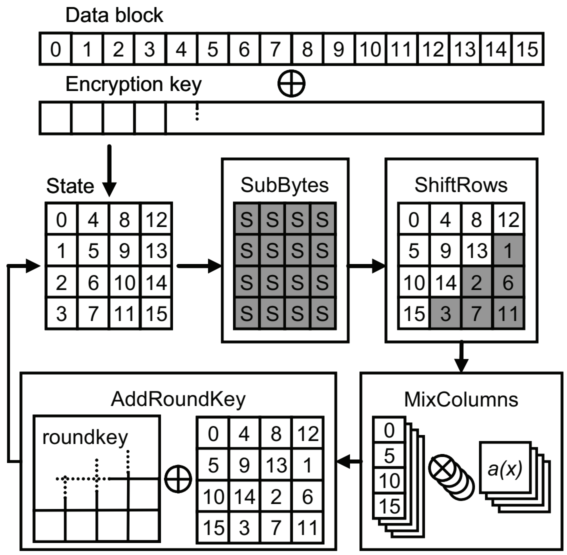


Figure 1. AES round operations. The indices show how State is permuted during ShiftRows. After AddRoundKey the indices are reallocated (no permutation is performed).

3. Related Work

After the ratification of AES in 2001, a large number of its hardware implementations has appeared. Whereas the earlier designs mainly focused on intensively pipelined, high-speed implementations, the more recent work has concentrated on compact and low-power architectures considering low-cost devices and feedback modes of operation.

A number of iterative AES ASIC designs with varying data path widths are reported in [14]. The designs are based on an efficient S-box architecture and include en/decryption for 128-bit keys. Roundkeys are generated on-the-fly, either by sharing S-boxes with the main data path or by dedicating separate S-boxes for the key expansion. The smallest version is a 32-bit AES architecture with four shared S-boxes.

A 32-bit AES architecture with a pre-computed key expansion is developed for FPGAs in [2]. The design takes advantage of the dedicated memory blocks of FPGAs by implementing S-box as a LUT. The paper proposes a method for arranging the bytes of State so that it can efficiently be stored into memory components or shift registers. The arrangement allows performing ShiftRows with addressing logic. The same method is proposed again in [12]. For decreasing the amount of storage space as well as supporting various data path widths, we have developed the idea further in [10] without an implementation. In [7] we removed the decryption functionality of [2] and used the core for implementing the security processing of [8] in a low-cost FPGA. Ref. [13] improves the FPGA resource consumption of [2] with the T-box method.

A resource-efficient ASIC design supporting en/decryption is presented in [12]. The on-the-fly key expansion shares S-boxes with the main data path. The design is based on a regular architecture that can be scaled for different speed and area requirements. The smallest version contains a 32-bit data path. Support for the Cipher Block Chaining (CBC) encryption mode is also included.

A low-power and compact ASIC core for 128-bit-key AES encryption is reported in [4]. The 8-bit data path is used for the round operations as well as for the on-the-fly key expansion. The data path contains one S-box implemented as combinatorial logic. State and the current roundkey are stored in a 32×8-bit RAM, which has been implemented with registers and multiplexers. The memory is intensively used by cycling each intermediate result through the RAM, increasing the total cycle count of the design. For MixColumns the design uses a shift-register-based approach, which is capable of computing the operation in 28 cycles. Decryption functionality is added to the design in [5], which also reports results from a manufactured chip.

Ref. [3] reports a 8-bit FPGA design for the AES encryption with 256-bit keys. Roundkeys are pre-computed. State is stored into a memory, allowing to perform ShiftRows

Proceedings of the 9th EUROMICRO Conference on Digital System Design (DSD'06)
0-7695-2609-8/06 $20.00 © 2006 IEEE

Authorized licensed use limited to: Purdue University. Downloaded on April 4, 2009 at 14:15 from IEEE Xplore. Restrictions apply.

with addressing logic. However, the memory consists of two sections used in turns, which consumes unnecessary resources [10]. Compared to [4, 5], the implementation of MixColumns is more efficient as the inputs need to be read only once from their storage. In [4,5] the first three bytes of each column are read twice from the RAM during a single MixColumns operation. The total cycle count of the AES core [3] is lower than in [4,5]. In this paper we use a similar approach for the MixColumns operation as in [3].

A 8-bit AES processor for FPGAs is designed in [6], capable of 128-bit-key en/decryption. The data path consists of an S-box and a GF multiplier/accumulator. The execution is controlled with a program stored in ROM. RAM is used as data memory. The cycle count of the design is significantly higher than in [3–5]. The FPGA resource consumption is lower than in [3].

4. AES Encryption Core Architecture

The most compact AES solutions both in ASICs [4, 5] and in FPGAs [6] have been realized with iterative architectures employing 8-bit data paths. We use the 8-bit data path width as well but adopt a completely different design approach. In [4–6] the AES round operations as well as the key expansion operations are performed sequentially. In our design the operations are performed in parallel (for different 8-bit pieces of data/key), which considerably decreases the total cycle count and increases the throughput. Still, we succeed in maintaining the hardware area and the power consumption low.

The high-level architecture of our AES encryption core is depicted in Fig. 2. The core supports 128-bit keys and computes one round at a time in 16 clock cycles. It consists of five components: *byte permutation unit*, *MixColumns multiplier*, *parallel-to-serial converter*, *S-box*, and *key expansion unit*. The S-box unit is instantiated twice. All the connections and registers in our design are 8 bits wide.

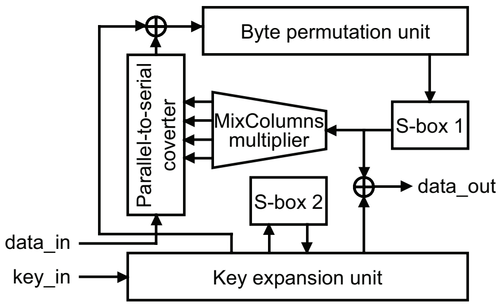


Figure 2. High-level architecture of the AES encryption core.

4.1. Byte Permutation Unit

The ShiftRows operation and partial storage of State are combined into the byte permutation unit depicted in Fig. 3. ShiftRows is performed by reordering the State bytes while they are shifted through the unit. The remaining four bytes of State are processed and maintained in the other data path registers of the AES core. For detailed operation of the unit we refer to [10]. The same structure can also be used for performing the inverse of ShiftRows [10].

4.2. MixColumns Multiplier

The MixColumns multiplier in Fig. 4(a) performs the matrix multiplication of MixColumns. One column of State is treated at a time in four clock cycles as presented in Fig. 4(b). Data are fed to the unit byte by byte and the intermediate results are maintained in the four registers. As the same multiplier coefficients are used for each row of a column [11], only in a cyclicly shifted order (i.e. the multiplier matrix is *circulant*), a 32-bit part of the MixColumns operation can be performed by adding and cyclicly shifting the intermediate results in the unit [3]. During inputting the first byte of a column (bytes 0, 4, 8, and 12 in Fig. 1) the contents of the registers are masked to zero with the *en* signal. Upon the completion, the 32-bit output is fed to the parallel-to-serial converter. The MixColumns multiplier performs a complete MixColumns operation in 16 cycles in parallel with the rest of the operations of the AES core.

4.3. S-box

Implementing S-box directly as a LUT, using either ROM or letting the synthesis tool to build combinatorial logic from the truth table, does not result in the optimal solution in terms of gate area in ASIC technologies [14]. This approach can efficiently be used in FPGAs [2, 3, 7, 12]. In our core we use the S-box design of [1], which presents the most compact ASIC S-box solution according to our knowledge. The author provides also a hand-tuned gate-level Verilog description of the S-box. In the design, the computation of the GF(2^8) inverse is broken up into computations in smaller subfields. The best choices for the subfields have been found after analyzing a large number of different alternatives. Our data path contains one forward S-box for performing SubBytes a byte at a time. Another identical S-box is used for the key expansion.

4.4. Key Expansion Unit

As roundkeys remain the same when the encryption key is not changed, the key expansion can be performed by pre-computing the roundkeys and storing them [2, 3, 7, 13].

Authorized licensed use limited to: Purdue University. Downloaded on April 4, 2009 at 14:15 from IEEE Xplore. Restrictions apply.

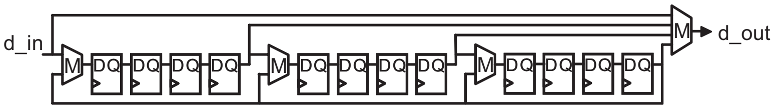


Figure 3. Data path of the byte permutation unit.

However, this requires a considerable amount of storage space. Instead, we calculate each roundkey from the previous one on-the-fly, which requires space only for a single roundkey [4, 5, 12]. It should be noted that when encrypting more than one block of data with the same encryption key, the on-the-fly approach continuously repeats the work already done. Therefore, the pre-computed approach can be more energy-efficient in the long run.

In both the approaches the S-boxes of the main data path can be utilized for the computations [2, 14], which results in equal amount of computational resources in both cases. However, in the on-the-fly method this requires interleaving the key expansion with the data block processing [14], which increases the total cycle count and implies more complex control logic. In our design we dedicate an S-box for the key expansion. This does not significantly increase the total area due to the compact S-box design.

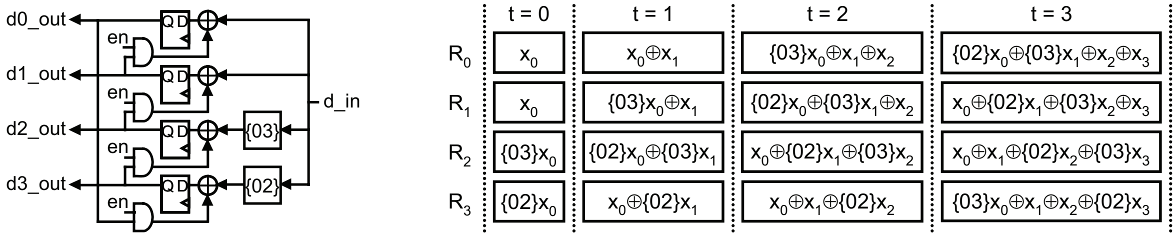
One round of the KeyExpansion transformation [11] is depicted in Fig. 5(a). The transformation treats roundkeys as 4-byte words. Each word of the new roundkey is formed by XORring the corresponding word of the previous roundkey with the preceding word. The last word of the previous roundkey is cyclicly shifted, processed through S-boxes, and XORed with a round-dependent constant (*Rcon*).

We have embedded the storage for the roundkey and the KeyExpansion transformation into the data path of the key expansion unit depicted in Fig. 5(b). The output *rk_last_out* is used for AddRoundKey during the final round. Since a 32-bit MixColumns operation takes four clock cycles, the key expansion unit has also another output *rk_delayed_out*, which is delayed by four clock cycles from *rk_last_out*. The output is used for AddRoundKey during the regular rounds and the initial encryption key addition. This AddroundKey operation is performed with the XOR in the input of the byte permutation unit in Fig. 2. The last AddRoundKey operation uses the other XOR gate. Rcon is formed from the round counter using combinatorial logic and masked to zero when not required. The operation of the key expansion unit is clarified in Table 1.

4.5. Top Level

Fig. 6 depicts the data path of the AES core. A plaintext block and the encryption key are simultaneously loaded to the core a byte at a time through the input ports *data_in* and *key_in*. The initial AES step, which is AddRoundKey between the block and the key, is performed during the loading with the XOR gate in the input of the byte permutation unit. After ten rounds of operation the ciphertext block can be unloaded byte by byte from the output port *data_out*. As roundkeys overwrite the encryption key, it must be resupplied to the core along with a new plaintext block.

The final round, consisting of SubBytes, ShiftRows, and AddRoundKey, is performed during the data unloading. Since State is stored into the register chain of the byte permutation unit and the roundkey into the register chain of the key expansion unit, a new plaintext block and an encryption key (the same or a new one) can be loaded to the core simultaneously with the unloading of the ciphertext block.



	t = 0	t = 1	t = 2	t = 3
R_0	x_0	$x_0 \oplus x_1$	$\{03\}x_0 \oplus x_1 \oplus x_2$	$\{02\}x_0 \oplus \{03\}x_1 \oplus x_2 \oplus x_3$
R_1	x_0	$\{03\}x_0 \oplus x_1$	$\{02\}x_0 \oplus \{03\}x_1 \oplus x_2$	$x_0 \oplus \{02\}x_1 \oplus \{03\}x_2 \oplus x_3$
R_2	$\{03\}x_0$	$\{02\}x_0 \oplus \{03\}x_1$	$x_0 \oplus \{02\}x_1 \oplus \{03\}x_2$	$x_0 \oplus x_1 \oplus \{02\}x_2 \oplus \{03\}x_3$
R_3	$\{02\}x_0$	$x_0 \oplus \{02\}x_1$	$x_0 \oplus x_1 \oplus \{02\}x_2$	$\{03\}x_0 \oplus x_1 \oplus x_2 \oplus \{02\}x_3$

(a)

(b)

Figure 4. MixColumns multiplier: (a) the data path and (b) the contents of the registers (R_i) during the four cycles (t). $\{02\}$ and $\{03\}$ denote the AES field multiplications with x and $x + 1$, respectively.

Authorized licensed use limited to: Purdue University. Downloaded on April 4, 2009 at 14:15 from IEEE Xplore. Restrictions apply.

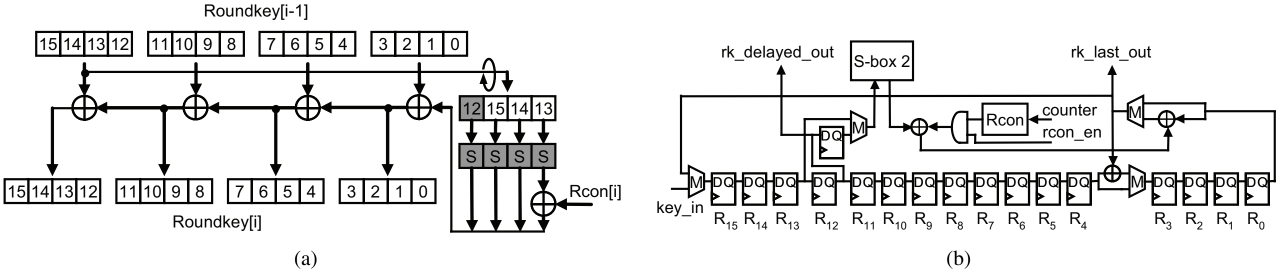


Figure 5. Key expansion: (a) a graphical representation of the KeyExpansion round and (b) the data path of the key expansion unit. The organization of bytes in (a) matches with the registers in (b) when a full roundkey has been generated.

This improves the performance as there are no dead cycles between consecutively processed blocks. The feature is convenient e.g. in the CBC-MAC mode, in which the ciphertext block is XORed with the next plaintext block before processing. Processing a single block in our core takes 176 clock cycles including loading and unloading. Since load/unload can be performed simultaneously, the effective cycle count is 160, also in feedback modes.

5. Results and Comparison

We described the AES core in VHDL at the register transfer level and synthesized it to the gate level using a 0.13 μm, 1.2 V, standard-cell CMOS technology. The area expressed as the number of gate equivalents, the power consumption, and the maximum clock frequency under typical operating conditions (1.2 V, 25 °C) are presented in Table 2. We have included the results from area, power, and speed optimized syntheses. For the power estimations we performed gate-level power analysis. The estimations and the power optimized synthesis were based on switching activities, which were captured with gate-level simulations using random test vectors as the stimulus. We used the Synopsys synthesis tools package version 2005.09-SP2 for the logic synthesis and the power analysis. Simulations were performed with Mentor Graphics ModelSim SE 6.0a.

The table compares our results with the previous compact AES ASICs. Whereas [4] does not report the power consumption (only the current is given), in [5] the power has been measured from the actual integrated circuit. For [12, 14] we have included the area optimized results.

Table 1. Operation of the key expansion unit

t	$\mathbf{R_{15}}$	$\mathbf{R_{14}}$	$\mathbf{R_{13}}$	$\mathbf{R_{12}}$	$\mathbf{R_{11}\ldots R_4}$	$\mathbf{R_3}$	$\mathbf{R_2}$	$\mathbf{R_1}$	$\mathbf{R_0}$	**operations**
0	a_{15}	a_{14}	a_{13}	a_{12}	$a_{11}\ldots a_4$	a_3	a_2	a_1	a_0	$b_0 = a_0 \oplus S(a_{13}) \oplus Rcon, b_4 = a_4 \oplus b_0$
1	$\boldsymbol{b_0}$	a_{15}	a_{14}	a_{13}	$a_{12}\ldots a_5$	$\boldsymbol{b_4}$	a_3	a_2	a_1	$b_1 = a_1 \oplus S(a_{14}), b_5 = a_5 \oplus b_1$
2	$\boldsymbol{b_1}$	b_0	a_{15}	a_{14}	$a_{13}\ldots a_6$	$\boldsymbol{b_5}$	b_4	a_3	a_2	$b_2 = a_2 \oplus S(a_{15}), b_6 = a_6 \oplus b_2$
3	$\boldsymbol{b_2}$	b_1	b_0	a_{15}	$a_{14}\ldots a_7$	$\boldsymbol{b_6}$	b_5	b_4	a_3	$b_3 = a_3 \oplus S(a_{12}), b_7 = a_7 \oplus b_3$
4	$\boldsymbol{b_3}$	b_2	b_1	b_0	$a_{15}\ldots a_8$	$\boldsymbol{b_7}$	b_6	b_5	b_4	$b_8 = a_8 \oplus b_4$
5	b_4	b_3	b_2	b_1	$b_0\ldots a_9$	$\boldsymbol{b_8}$	b_7	b_6	b_5	$b_9 = a_9 \oplus b_5$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14	b_{13}	b_{12}	b_{11}	b_{10}	$b_9\ldots b_2$	b_1	b_0	b_{15}	b_{14}	(no operations)
15	b_{14}	b_{13}	b_{12}	b_{11}	$b_{10}\ldots b_3$	b_2	b_1	b_0	b_{15}	(no operations)

One key expansion round, which takes 16 clock cycles (t), is shown. The columns R_{15}...R_0 represent the contents of the corresponding registers in Fig. 5(b), a_{15}...a_0 are the bytes of the previous roundkey, and b_{15}...b_0 are the bytes of the new roundkey. $S(a_i)$ is an S-box substitution of a_i. The bolded values have been updated by the operations of the previous cycle. During the last round, the byte b_t of the last roundkey is output via rk_last_out at cycle t. During the regular rounds, the output is delayed by four cycles and given via $rk\_delayed\_out$.

Authorized licensed use limited to: Purdue University. Downloaded on April 4, 2009 at 14:15 from IEEE Xplore. Restrictions apply.

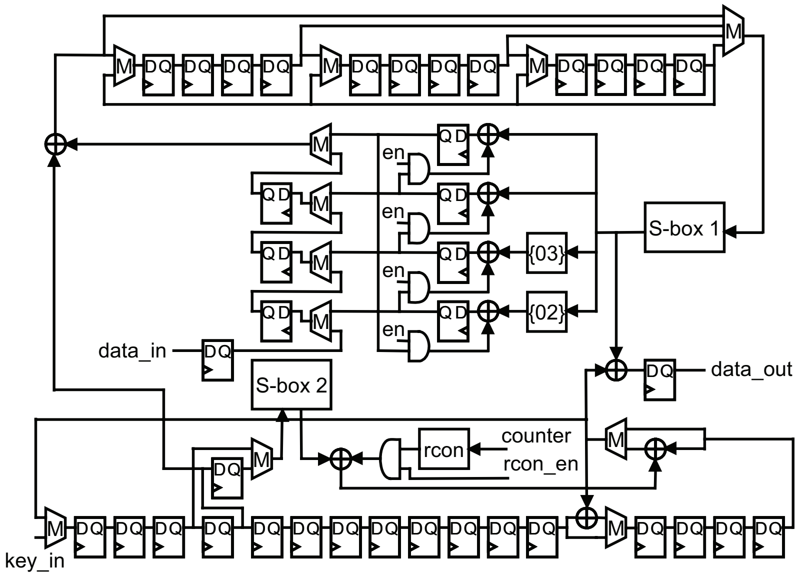


Figure 6. Data path of the AES encryption core.

Compared to the previous 8-bit implementations [4, 5], our cycle count is significantly lower and throughput higher because of the parallel operation. The gate area is at the same level. The cycle count we achieve (one cycle per byte per round, equalling the total of 160) can be seen as the minimum for an iterated 8-bit AES implementation. We have estimated that including the decryption functionality would add about 25% to our total area. Comparing power consumption with [5] is difficult as different technologies and power estimation levels have been used. However, because of the lower cycle count, the power estimations suggest that our energy per block can be lower than in [5]. Furthermore, for achieving equal throughput our design can be run at considerably lower clock frequency, which also decreases the total power consumption. Our cycle count is lower than those of the 8-bit FPGA implementations [3, 6] as well.

When a high throughput-area ratio is the main design goal, we believe that the best result can be achieved with a 128-bit data path, as demonstrated in [14]. Due to the lower parallelism, it is not possible to reach the cycle counts and/or throughputs of 32-bit, 64-bit, and 128-bit AES implementations with a 8-bit design. The minimum cycle counts for iterated architectures using those data path widths are 40, 20, and 10, respectively. Therefore, the throughput of [14] is higher than ours. Also, the throughput of [12] can be higher than ours when implemented in the same technology. However, our results as well as the previous results [4, 5] show that a 8-bit design can produce the lowest area with low power consumption. We have shown that the throughput can still be high. It has previously been believed that a 8-bit design results in an inefficient implementation [14].

Table 2. Implementation results and comparison

Implementation	**Width** [bits]	**Area** [kgates]	**Process** [μm]	**Max. freq.** [MHz]	**Cycles per block**	**Throughput** [Mbps]	**Power** [μW/MHz]	**Energy per block** [nJ]	**Decryption**
Ours (area)	8	3.1	0.13	152	160	121	37	5.9	no
Ours (power)	8	3.2	0.13	130	160	104	30	4.8	no
Ours (speed)	8	3.9	0.13	290	160	232	62	9.9	no
Ref. [4]	8	3.6	0.35	n/a	1016	n/a	n/a	n/a	no
Ref. [5] (3.3 V)	8	3.4	0.35	80	1032	10	n/a	n/a	yes
Ref. [5] (1.5 V)	8	3.4	0.35	n/a	1032	n/a	45	46.4	yes
Ref. [14]	32	5.4	0.11	131	54	311	n/a	n/a	yes
Ref. [12]	32	8.5	0.60	50	92	70	n/a	n/a	yes

Authorized licensed use limited to: Purdue University. Downloaded on April 4, 2009 at 14:15 from IEEE Xplore. Restrictions apply.

Table 3. Area and power consumption distributions of the AES encryption core

Component	**Gates** (%)	**Power** (%)
Key expansion unit	34.7	19.1
Byte permutation unit	24.0	12.0
Parallel-to-serial converter	9.4	3.6
MixColumns multiplier	8.8	22.6
S-box 1	8.0	29.0
S-box 2	8.0	6.7
Others/Control	7.1	7.0
Total	100.0	100.0

For additional analysis, Table 3 shows the area and power consumption distributions of the core. The registers consume the majority of the area. Especially, the byte permutation and the key expansion units account for 60% of the total area. S-box 1 is the most power consuming unit due to its high switching activity. Since S-box 2 is only utilized for one fourth of the time (see Table 1), its power consumption compared to S-box 1 is also one fourth. The second highest power consumption of the MixColumns multiplier is mainly due to the high switching activity at the input of the unit caused by glitches in S-box 1. To address this, we experimented by adding a register to the input of the MixColumns multiplier and taking the outputs before the four original registers. The modification did not affect the maximum clock frequency but reduced the total power consumption by 16%. The total area was increased by 4%.

6. Conclusions

In this paper we presented the design and implementation of a compact 8-bit AES ASIC encryption core suitable for low-cost and low-power devices. Compared to previous 8-bit designs, we achieved significantly higher throughput with corresponding area. The power estimations suggest that our energy consumption per processed block is also lower. In order to support further research and comparison of results on different technologies, we will publish the VHDL sources of the core as open source on our research group's website (http://www.tkt.cs.tut.fi/research/daci).

References

[1] D. Canright. A very compact S-box for AES. In *Proc. 7th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005)*, pages 441–455, Edinburgh, UK, Aug. 29–Sept. 1, 2005.

[2] P. Chodowiec and K. Gaj. Very compact FPGA implementation of the AES algorithm. In *Proc. 5th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003)*, pages 319–333, Cologne, Germany, Sept. 8–10, 2003.

[3] S. Farhan, S. Khan, and H. Jamal. Mapping of high-bit algorithm to low-bit for optimized hardware implementation. In *Proc. 16th IEEE Int. Conf. on Microelectronics (ICM 2004)*, pages 148–151, Tunis, Tunisia, Dec. 6–8, 2004.

[4] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In *Proc. 6th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, pages 357–370, Boston, MA, USA, Aug. 11–13, 2004.

[5] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *IEE Proc. Inf. Secur.*, 152(1):13–20, 2005.

[6] T. Good and M. Benaissa. AES on FPGA from the fastest to the smallest. In *Proc. 7th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005)*, pages 427–440, Edinburgh, UK, Aug. 29–Sept. 1, 2005.

[7] P. Hämäläinen, M. Hännikäinen, and T. Hämäläinen. Efficient hardware implementation of security processing for IEEE 802.15.4 wireless networks. In *Proc. 48th IEEE Int. Midwest Symp. on Circuits and Systems (MWSCAS 2005)*, pages 484–487, Cincinnati, OH, USA, Aug. 7–10, 2005.

[8] IEEE. *IEEE Standard for Local and Metropolitan Area Networks—Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPAN)*, 2003. IEEE Std 802.15.4.

[9] IEEE. *IEEE Standard for Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—Amendment 6: Medium Access Control (MAC) Security Enhancements*, 2004. IEEE Std 802.11i.

[10] T. Järvinen, P. Salmela, P. Hämäläinen, and J. Takala. Efficient byte permutation realizations for compact AES implementations. In *Proc. 13th European Signal Processing Conf. (EUSIPCO 2005)*, Antalya, Turkey, Sept. 4–8, 2005.

[11] National Institute of Standards and Technology (NIST). *Advanced Encryption Standard (AES)*, 2001. FIPS-197.

[12] N. Pramstaller, S. Mangard, S. Dominikus, and J. Wolkerstorfer. Efficient AES implementations on ASICs and FPGAs. In *Proc. 4th Conf. on the Advanced Encryption Standard (AES 2004)*, pages 98–112, Bonn, Germany, May 10–12, 2005.

[13] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat. Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications. In *Proc. IEEE Int. Conf. on Inf. Tech.: Coding and Computing (ITCC 2004)*, volume 2, pages 583–587, Las Vegas, NV, USA, Apr. 4–6, 2004.

[14] A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A compact Rijndael hardware architecture with S-box optimization. In *Proc. 7th Int. Conf. on Theory and Application of Cryptology and Inf. Secur., Advances in Cryptology (ASIACRYPT 2001)*, pages 239–254, Gold Coast, Australia, Dec. 9–13, 2001.

[15] ZigBee Alliance. *ZigBee Specification Version 1.0*, Dec. 2004.

0-7695-2609-8/06 $20.00 © 2006 IEEE

Authorized licensed use limited to: Purdue University. Downloaded on April 4, 2009 at 14:15 from IEEE Xplore. Restrictions apply.