

An Area Efficient Realisation of AES for Wireless Devices

Eine flächeneffiziente AES Hardwarerealisierung für drahtlose Geräte

Frank Vater, Peter Langendörfer, IHP Microelectronics GmbH, Frankfurt (Oder)

Summary In this paper we describe our own AES implementation, which supports encryption as well as decryption. Our major design goal was to reduce the area while still being capable to support high speed wireless networks such as IEEE 802.11a. Our AES solution provides a throughput of 54 MBit/s at 33 MHz and requires an area of 0.33 mm² in a 0.25 μm technology. This version may be run at up to 66 MHz which gives a throughput of 108 MBit/s. During the design we took into account global as well as local optimisations, i. e., optimisations which could be done inside an individual operation without affecting the rest of the design.

▶▶▶ Zusammenfassung Dieser Artikel diskutiert eine AES Realisierung in Hardware, die sowohl zur Ver- als auch zur Entschlüsselung verwendet werden kann. Die hier vorgestellte Lösung erreicht folgende Optimierungskriterien: Minimierung der Fläche unter Wahrung eines für breitbandige drahtlose Netzwerke notwendigen Durchsatzes. Die hier vorgestellte Lösung erlaubt einen Durchsatz von 54 MBit/s bei 33 MHz und hat eine Fläche von nur 0,33 mm² in einer 0,25 μm Technologie. Diese Version kann mit bis zu 66 MHz getaktet werden und bietet dann einen Durchsatz von 108 MBit/s. Bei der Realisierung wurden sowohl globale als auch lokale, d. h. operationsspezifische Optimierungen berücksichtigt.

KEYWORDS B.5.1 [Hardware: Register-Transfer-Level Implementation: Design] B.7.1 [Hardware: Integrated Circuits: Types and Design Styles], E.3 [Data: Data Encryption] symmetric key cryptography, AES, hardware accelerator, wireless systems / symmetrische Verschlüsselungsverfahren, Hardwarebeschleuniger, drahtlose Systeme

1 Introduction

The use of cipher mechanisms in wireless networks is a must. This holds true for all kinds of wireless networks such as MANETs, sensor networks mesh networks, WLANs etc. Mobile devices are still suffering from scarce resources. This concerns particularly memory, calculation power and energy. The extent to which a certain device suffers depends on its class. For example, a standard PDA has sufficient processing power to run symmetric and even asymmetric cipher mechanisms, but its up time is decreased by these operations. Even embedded devices or sensor nodes are capable to run cipher mechanisms, nevertheless their lifetime penalty is even

worse due to the fact that the batteries are smaller and that re-charging is much more complicated than for PDA like devices. A straightforward solution is the use of hardware accelerators for cryptographic operations, since they help to speed up the computation of cipher algorithms by two orders of magnitude while reducing the energy needed by three orders of magnitude. The additional hardware increases cost of devices. This might be acceptable for PDA like devices with a prize of 450 + €, but not for sensor nodes that will be deployed in high volume and therefore need to be low cost.

In this paper we present our own design of AES, which sup-

ports en- as well as decryption. It provides a throughput that is sufficient for high data rate, i. e., 54 MBit/s WLAN devices, and requires an area of less than 0.33 mm² in a 0.25 μm technology. Due to the small area the cost of such an accelerator is about 0.02 € if it is included into a system on chip solution, e. g., in our IEEE802.11a modem or into a sensor node.

The rest of this paper is structured as follows. We first provide a short description of AES and discuss related work. Section 3 provides details of our own implementation. Simulation results are presented in Section 4. There we compare our solution with other

hardware implementations. In Section 5 a short summary concludes this paper.

2 State of the Art

In this section we first provide a brief overview of AES, i.e., we describe its major functional blocks. Thereafter we give a short overview of hardware and software implementations of AES, which we later on will use as a basis for evaluating our own implementation.

2.1 Overview of AES

AES (Advanced Encryption Standard) was selected as DES successor by NIST in 2001. There are no known successful attacks against AES. It can be implemented very efficiently in software as well as in hardware. AES consists of a key generator, which creates the round keys from the initial key and a ciphering part, which does the en-/decryption. In order to complete an en-/decryption the algorithm has to be applied ten times to the incoming data. Each such sequence is called a round. Each round consists of the following four functions that are applied to the data in the order given here:

1. Add key operation: the incoming/outgoing data is “xored” with the current round key
2. S-BOX: The S-Box maps an 8-bit-input into an 8 bit output. For every direction (encryption and decryption) an own S-Box is required, since its result depends on the operation (en-/decryption) that is currently performed.
3. Shift Row: The shift row operation is performing a cyclic shift within a single row.
4. MixColumn: In the MixColumn operation each column is multiplied by a given polynomial, represented by a cyclic shifted matrix. The resulting vector becomes the new column. While decrypting data the columns are multiplied with the inverse polynomial of polynomial used for encryption.

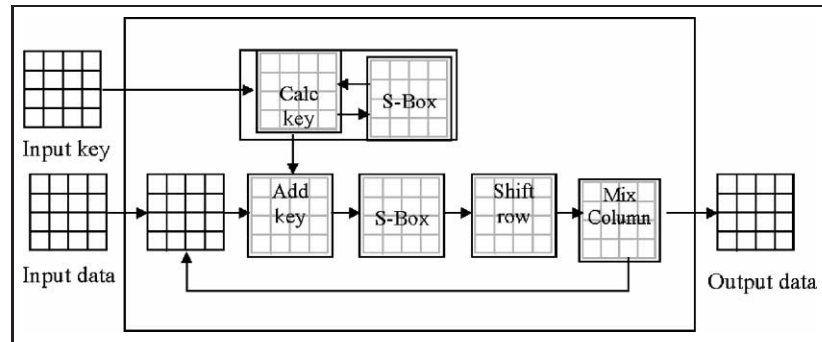


Figure 1 Functional blocks of AES.

5. To perform the next round a new round key is required. It is calculated from the initial/previous key using a S-Box and a round depending constant.

2.2 Related Work

In order to evaluate the performance and energy benefits a hardware implementation of AES can provide we have examined several software implementations. Table 1 shows the results. They clearly indicate that even embedded devices can run AES in software. Sophisticated assembler implementation can further reduce the number of required clock cycles. For instance, our own assembler implementation requires only 1,000 clock cycles to encrypt 128 bit on a 64 bit MIPS RISC processor. So, hardware accelerator is only justified if:

1. An extremely long lifetime is required.
2. The amount of data to be encrypted is significantly high.
3. The hardware accelerator is extremely cost efficient.

There is a lot of work done with respect to hardware implementations of AES. Most of these implementations are high-throughput designs

for up to 1 GBit/s and above, see e.g. [1; 11]. Since we are aiming at low cost and low energy consuming AES accelerators, high speed designs did not have serious impact on our own work. There are also several papers published that target at “low” throughput implementations [2; 3; 4]. References [2] and [3] have influenced our ideas on how to improve the AES performance. Also the optimisation of individual AES operations has attracted some attention. References [5] and [6] are investigating efficient S-Box implementations whereas [7] and [8] discuss the implementation of the AES mix-column operation. Especially [2] and [9] made an impact on our design with respect to the S-Box and MixColumn component.

3 Optimised Implementation of AES

In order to find an AES implementation that satisfies both optimisation goals, small area consumption and good performance, we investigated several potential designs for AES operations and put some effort in optimising the overall structure of the design. We

Table 1 Performance of AES software implementation in C, measured on simulated MIPS 4KEp [16].

Operation	# of clock cycles
Encrypt, 128 bit key (standard implementation)[14]	44,578
Decrypt, 128 bit key (standard implementation)[14]	55,899
Encrypt, 128 bit key (optimised version)[15]	5,228
Decrypt, 128 bit key (optimised version)[15]	6,857

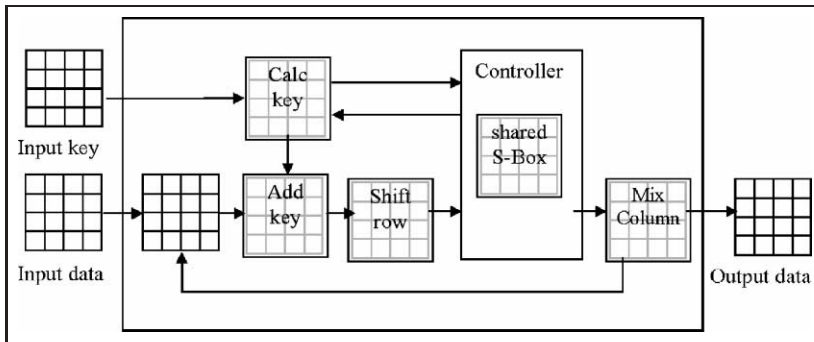


Figure 2 AES design with only one S-Box, whose use is scheduled by a specialized controller.

start regarding the global architecture, since it is the basis of all further improvements and optimisations of functional blocks. Fig. 2 shows the resulting architecture. The details of our modifications are discussed in the following subsections. The presented results have been generated by Synopsys Design-Vision [17].

3.1 Global Optimisations

In order to achieve an area efficient AES design we modified the AES structure shown in Fig. 1. We changed the following:

1. Number of S-Boxes
2. Order of AES operations, i.e., Shift Row and S-Box
3. Key management

3.1.1 Reduction of the Number of S-Boxes

A standard AES design needs at least two S-Boxes, see Fig. 1. The first one is used for key calculation, whereas the second one is used during de-/encryption. Since these two operations are never executed simultaneously, we decided to omit one of the S-Boxes. But this comes at some cost, i.e., additional control logic is needed to ensure that the S-Box is used for the correct operation. The area consumption of the control logic is $3,000 \mu\text{m}^2$ for 32 multiplexers. Thus, we reduced the area consumption by nearly $66,000 \mu\text{m}^2$ which corresponds to the area of an S-Box. The concrete value depends on the design of the S-Box that is used in the respective implementa-

tion, see Table 3 in Section 3.2.2. Fig. 2 shows the resulting structure of our AES design. Our optimisation has no performance penalty due the strict alternation in the use of the S-Box, i.e., none of the S-Box operations is blocking the other one.

3.1.2 Parallelisation of S-Box and MixColumn Operation

Due to the fact that all AES functions are realised as individual hardware blocks we decided to parallelise our design. In the current design we use a pipeline with 2 stages, consisting of the S-Box and MixColumn operations. In order to allow parallel processing of columns of input data by the S-Box and the MixColumn operation we changed the sequence, in which the shift row operation and the S-Box operation are applied. Inverting the order of these operations does not change the result. The shift row operation only changes the sequence of the bytes of the data under en-/decryption, whereas the S-Box is applied to single bytes, which are not affected by

the shift row operation. In other words the output of the S-Box operations only depends on its input, and not on the position of the input.

Our pipelined architecture needs 70 clock cycles, seven per round, to apply the whole AES algorithm to 128 bit of input data. Algorithm 1 (see Fig. 3) shows which operations are applied to the input data. All operations defined in the same line are executed in parallel, and each line requires a single clock cycle only to complete all specified operations. Table 2 shows the pipeline for S-Box and MixColumn. Please note that the S-Box and MixColumn operation are performed on a single column in one clock cycle and not on the full data array.

In some implementations, e.g. [2], the functions S-Box and MixColumn are performed in a single clock cycle on the full data array. But the S-Box is the most complex and time consuming operation of the AES algorithm. So splitting it into several steps reduces the timing constraints and allows operating the design at higher clock frequencies. Executing MixColumn and S-Box in one clock cycle reduces the maximal frequency clock to 66 MHz. Less complex S-Box and MixColumn implementations, that require more clock cycles, allow maximal clock frequencies of up to 125 MHz.

3.1.3 Optimisation of the Key Management to Support faster Decryption

The goal of our optimisation of the key management is to avoid the overhead needed to restore the original key after having en-/decrypted

```

data=addkey(data, key);
For i=1 to 10 do {
    data=shiftrw(data); key=Calc(key);
    data(0)=S-Box(data(0));
    data(1)=S-Box(data(1)); if (i<10) data(0)=MixCol(data(0));
    data(1)=S-Box(data(2)); if (i<10) data(0)=MixCol(data(1));
    data(1)=S-Box(data(3)); if (i<10) data(0)=MixCol(data(2));
    if (i<10) data(0)=MixCol(data(3));
    data=addkey(data, key);
    i++;
}
    
```

Figure 3 Algorithm 1: Pseudo code to perform the AES on a 128 bit data block. Each line displaying AES operations requires a clock cycle to be executed. If more than one function is specified all these functions are executed in parallel.

Table 2 Example of the AES pipeline processing 4 data blocks each 128 bit long. The number in brackets denotes the current row number.

Clock 1	Clock 2	Clock 3	Clock 4	Clock 5
S-Box(0)	S-Box(1) MixColumn(0)	S-Box(2) MixColumn(1)	S-Box(3) MixColumn(2)	MixColumn(3)

a 128 bit chunk. Please recall that the key is modified in each round, which means that it has to be recovered before starting to en-/decrypt the next chunk of data. The recovering needs ten clock cycles which is more than ten per cent of the time our AES design needs to complete the en-/decryption of 128 bit. Thus, we decided to use two 128 bit registers to store the initial keys. The first register contains the key for the encryption round and the second one contains the key for the decryption round. The major benefit of this solution is that a key setup time of ten clock cycles is required once per session only. Thereafter the direction, i. e., encryption or decryption, can be chosen and the algorithm will start immediately. Thus, more than ten per cent performance gain is achieved by using more area to store the keys, while it requires only seven per cent of the complete area of our design.

3.2 Optimized Operational Blocks

3.2.1 Optimisation of the Add Key Operation The add key operation is performed on 128 bit of input data in a single clock cycle. This is achieved by using 128 xor-gates to perform the add key operation. Thus, this design needs only 11 clock cycles (one cycle per round) to provide the add key operation for the complete en-/decryption of a 128 bit chunk. The size is about $10,200 \mu\text{m}^2$, which is about 3.1 per cent of the size of the complete design. A sequential design, e. g., using 32 xor-gates, would need 44 clock cycles to complete the add key operation (4 clock cycles per round). Since encrypting of a data block requires 70 clock cycles if the 128

xor-gates solution is used, the sequential solution would slow down the complete algorithm by up to 50 per cent. So, the performance penalty clearly out weights any area gain.

3.2.2 Optimisation of the S-Box

A full AES design (encryption and decryption direction) requires two different S-Boxes. Thus an area efficient design of the S-Box is of high importance, i. e., it helps to reduce the area consumption. In addition the design of the S-Box has a significant impact on the timing of the AES realisations, since it is the most complex operation.

There are several options to realise the S-Box, namely the use of ROM that stores the values, a look-up table in combinatorial logic, and the calculation in $\text{GF}(2^8)$ [6]. We have focused on the calculation in $\text{GF}(2^8)$, since we have no access to suitable ROM generator, and due to the fact that the combinatorial logic solution is quite expensive in area.

Every S-Box maps an input byte onto a specific output byte. If these mapping operations are executed sequentially, the complete S-Box operation on a 128 bit input stream takes 16 clock cycles. A straightforward way to improve the performance is to increase the number of S-Boxes. We integrated four single S-Boxes in our S-Box-component. Thus, it operates on a full column per clock cycles, which reduces the processing

time of a data block to four clock cycles.

Table 3 shows size and timing of S-Box implementations using a look-up table as well as calculating in $\text{GF}(2^8)$. For connecting the S-Box to the following operation MixColumn we evaluated two versions: a directly connected S-Box and a solution in which we use a register to buffer data when needed. The S-Box design using registers for buffering issues allows synthesizing the complete design for much higher frequencies than the version without registers. In the range of 100 MHz this difference becomes evident, i. e., the synthesis was not successful completed for the latter design, see Table 4. Thus, if high throughput is the design goal the version with registers has to be used.

3.2.3 Optimisation of the MixColumn Operation

In the MixColumn operation each column of the matrix representing the data under encryption is multiplied with a given polynomial $c(x)$, see Eq. (1). This operation can be described as a matrix multiplication, see Eq. (3). For decrypting data, a second polynomial $d(x)$ (see Eq. (2)) is used, $d(x)$ is the inverse of $c(x)$ in the field $\text{GF}((2^8)^4)$.

$$(1) c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

$$(2) d(x) = \{0B\}x^2 + \{0D\}x + \{09\}x + \{0E\}$$

$$(3) \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$(4) d^2(x) = \{04\}x^2 + \{05\}$$

The straightforward implementation is to use two independent hardware components for both multiplications. In order to reduce the area we applied the optimisation pro-

Table 3 Properties of different implementations of the S-Box.

	Size	Gates equivalent
S-Box (look-up table)	41,056 μm^2	1,032
S-Box (calc in $\text{GF}(2^8)$)	14,153 μm^2	356
S-Box (calc in $\text{GF}(2^8)$ with register)	16,512 μm^2	424

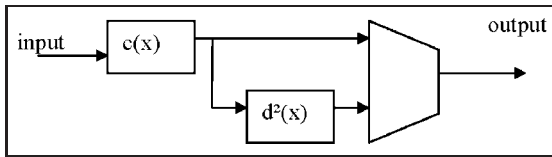


Figure 4 Reusing the MixColumn function for the inverse MixColumn function.

posed in [3]. The idea is to apply the MixColumn operation also for incoming data and afterwards multiplying it with $d^2(x)$ (Eq. (4)) to get the decrypted data. The point is that the area needed to realise the multiplication with $d^2(x)$ is much smaller than the one needed to implement the multiplication with $d(x)$. Fig. 4 shows the resulting structure of the MixColumn operation.

Some additional multiplexers are required to ensure the correct data flow for both operations, i. e. en-/decryption. The combined solution shown in Fig. 4 needs an area of about $16,500 \mu\text{m}^2$, whereas the use of a MixColumn and an InvMixColumn block requires $29,341 \mu\text{m}^2$ in silicon.

4 Measurements

In the following subsections we present simulation results concerning area and potential throughput of our AES design in $0.25 \mu\text{m}$ CMOS. We used Synopsys DesignVision [17] to synthesize the design and the simulations were executed using Cadence SimVision version 5.00-p001 [13]. All measurements presented in the rest of this section have also been done with these tools.

4.1 Throughput

The potential throughput depends on the number of clock cycles needed to en-/decrypt 128 bit, and on the maximal clock frequency which can be applied for a certain design. Table 4 shows the timing

constraints met by different S-Box designs. We are focusing here on the S-Box since it is the most complex function, i. e., its realisation determines the maximal frequency. The two S-Box designs that calculate the result are about 30 per cent smaller than the look-up table version, and are meeting the timing constraint down to 15 ns. The one using an additional register to relax the S-Box timing constraints can even satisfy timing constraints down to 8 ns. Please note that there is a significant larger area needed to meet the timing constraint of 8 ns than the one that is needed to meet the 10 ns constraint. The additional area is about 16 per cent of the 10 ns design. But this design is still about 14 per cent smaller than the look-up table version. Depending on the timing constraints met, clock frequencies up to 125 MHz can be supported. It results in a throughput of 205 Mbit/s, see Table 5. Including the input and output operations we need 78 clock cycles for a 128 bit data block. This is comparable to the results presented in [12] but our design requires about eight per cent less area.

Table 4 Cell area [μm^2] after synthesis and gate equivalents [GE] of complete AES core depending on different S-Box versions fulfilling different timing constraints.

	30 ns	15 ns	10 ns	8 ns
S-Box (Look-up table)	$0.472 \mu\text{m}^2$ 11,860 GE	$0.472 \mu\text{m}^2$ 11,860 GE	$0.472 \mu\text{m}^2$ 11,860 GE	$0.478 \mu\text{m}^2$ 12,000 GE
S-Box (Calc with reg)	$0.329 \mu\text{m}^2$ 8,266 GE	$0.332 \mu\text{m}^2$ 8,340 GE	$0.357 \mu\text{m}^2$ 8,967 GE	$0.415 \mu\text{m}^2$ 10,424 GE
S-Box (Calc without reg)	0.323mm^2 8,113 GE	0.335mm^2 8,414 GE	n/a	n/a

4.2 Area

In order to minimize the area required by our AES design we have evaluated the modules individually as well as the overall architecture of AES. The resulting design needs an area of about 0.329mm^2 to 0.415mm^2 . This is not the minimal area that could be achieved, but further reduction of the area significantly reduces the potential throughput. From our point of view the 30 ns design is an optimal tailor made solution to support the maximal raw data rate of an IEEE 802.11a WLAN system. If higher data rates are envisioned the 15 ns design is the better choice since it comes with double the throughput requiring only three per cent additional area. In addition this version provides a throughput comparable to those presented

Table 5 Throughput of the AES core at different clock frequencies.

	30 ns	15 ns	10 ns	8 ns
Clock frequency	33 MHz	66 MHz	100 MHz	125 MHz
Throughput	54 Mbit/s	108 Mbit/s	164 Mbit/s	205 Mbit/s

Table 6 Details of AES core with 30 ns clock cycle and S-Box with registers (¹: a row contains 4 S-Boxes.).

Module	Size	Gate equiv.	Per cent of total area
Alg w/o S-Box	$130,266 \mu\text{m}^2$	3,271	39.5
Key	$126,156 \mu\text{m}^2$	3,169	38.5
S-Box (row) ¹	$4 \times 16,532 \mu\text{m}^2$	4×415	20.5
Ctrl logic	$6,550 \mu\text{m}^2$	165	2.0
Total	$329,100 \mu\text{m}^2$	8,266	100.0

in [2] and [12], but requires about eight per cent less gate equivalents.

5 Conclusion

In this paper we have presented an area efficient and high performance hardware implementation of AES. In order to achieve such a small area we have analysed and optimised individual functional blocks of AES as well as the overall architecture. Our design allows a throughput up to 205 Mbit/s while requiring 0.415 mm² only. Thus, it is clearly well suited to be used in WLAN applications. It can easily support the required raw data rate of IEEE 802.11a, i. e. 54 Mbit/s, requiring only 0.33 mm². Due to its small area it does barely increase the bill of material of a standard WLAN enabled device. The power consumption of our AES implementation is about two orders of magnitude less than the one of the optimised software implementation presented in [15], i. e., it requires 1.04 mJ whereas the software implementation needs 95.3 mJ to encrypt 1 MByte.

An earlier version of our AES design was already manufactured and successfully presented at CeBIT 2005. There we used a streaming video application to show the applicability of our solution for real world problems as well as its interoperability with software implementations of AES.

Acknowledgements

The work presented in this paper was partially funded by the German Government under Grant 01AK060B.

References

- [1] A. Hodjat and I. Verbauwhede: High Performance Programmable Cryptocoprocessor. In: IEEE Micro, Vol. 24, No. 3, 2004.
- [2] St. Mangard, M. Aigner, and S. Dominiku: A Highly Regular and Scalable AES Hardware Architecture. In: IEEE Trans. on Computers, Vol. 52, No. 4, 2003.
- [3] P. Chodowicz and K. Gaj: Very Compact FPGA Implementation of the AES Algorithm. In: C.D. Walther et. al. (eds.) Ches 2003, LNCS 2779, 2003.
- [4] F. Bousse, M. Renaudin, and F. Germain: Asynchronous AES Crypto-Processor Including Secured and Optimized Blocks. In: Journal of Integrated Circuits and Systems, Vol. 1, No. 1, 2004.
- [5] N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede: A Systematic Evaluation of Compact Hardware Implementations of the Rijndael S-Box. In: A.J. Menezes (ed.) Proc. of CT-RSA, LNCS 3376, 2005.
- [6] J. Wolkerstorfer, E. Oswald, and M. Lamberger: An ASIC Implementation of the AES S-Boxes. In: B. Preneel (ed.) Proc. of CT-RSA, LNCS 2271, 2002.
- [7] J. Wolkerstorfer: An ASIC Implementation of the AES MixColumn-operation. In: Austrochip 2001.
- [8] E. Oswald: State of the Art in Hardware Architectures. September 2005.
- [9] A. Satoh, S. Morioka, K. Takano, and S. Munetoh: A Compact Rijndael Hardware Architecture with S-Box Optimisation. In: ASIACRYPT 2001, LNCS 2248.
- [10] Advanced Encryption Standard (AES), FIPS 197, NIST <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.
- [11] N. Kim, T. Mudge, and R. Brown: A 2.3Gb/s Fully Integrated and Synthesizable AES Rijndael Core. University of Michigan.
- [12] M. Aigner: Scalable AES HW-Module. Presentation, IAIK, 2002 <http://jce.iaik.tugraz.at/de/content/download/767/5659/file/AES%20Product%20Broch%20C3%BCre.pdf>.
- [13] www.cadence.com.
- [14] J. Daemon: AES implementation, Reference Implementation ANSI C v2.0 <http://www.iaik.tugraz.at/research/krypto/AES/old/~rijmen/rijndael/>.
- [15] J. Daemon: AES implementation, optimized ANSI C v2.0 <http://www.iaik.tugraz.at/research/krypto/AES/old/~rijmen/rijndael/>.
- [16] GreenHills Software Inc: Multi 2000, MIPS, v3.0.
- [17] www.synopsys.com.



1 B.Sc. Frank Vater has studied information and media technology at the Technical University in Cottbus (Germany) and received the Bachelor of Science in 2003. His research interests are security algorithm in hardware and protection systems against side channel attacks of hardware. Address: IHP GmbH, Im Technologiepark 25, 15236 Frankfurt (Oder), Germany, Tel.: +49-335-5625434, E-Mail: vater@ihp-microelectronics.com

2 Dr. Peter Langendörfer is leading the mobile middleware group at IHP microelectronics. He has published more than 55 refereed technical articles, filed six patents in the security/privacy area and worked as guest editor for several international journals including Wireless Communications and Mobile Computing (Wiley) and ACM Transactions on Internet Technology. He is a TPC member of many renowned international conferences such as Globecom, ICC, and WWIC. His current research interests include privacy and security issues in mobile computing. Address: IHP GmbH, Im Technologiepark 25, 15236 Frankfurt (Oder), Germany, Tel.: +49-335-5625350, E-Mail: langendoerfer@ihp-microelectronics.com, <http://www.ihp-microelectronics.com/~langend>