A Fully Pipelined Memoryless 17.8 Gbps AES-128 Encryptor

Kimmo U. Järvinen
Signal Processing Laboratory
Helsinki University of Technology
Otakaari 5 A
FIN-02150, Finland
Kimmo.Jarvinen@hut.fi

Matti T. Tommiska
Signal Processing Laboratory
Helsinki University of Technology
Otakaari 5 A
FIN-02150, Finland
Matti.Tommiska@hut.fi

Jorma O. Skyttä
Signal Processing Laboratory
Helsinki University of Technology
Otakaari 5 A
FIN-02150, Finland
Jorma.Skytta@hut.fi

ABSTRACT

A fully pipelined implementation of the Advanced Encryption Standard encryption algorithm with 128-bit input and key length (AES-128) was implemented on Xilinx' Virtex-E and Virtex-II devices. The design is called SIG-AES-E and it implements the S-boxes combinatorially and thus requires no internal memory. It is concluded, that SIG-AES-E is faster than other published FPGA-based implementations of the AES-128 encryption algorithm.

Categories and Subject Descriptors

E.3 [**Data Encryption**]: Standards; B.2.4 [**Arithmetic and Logic Structures**]: High-speed Arithmetic—*Algorithms*

General Terms

Algorithms, Performance, Design, Security

Keywords

Advanced Encryption Standard (AES), FPGA, pipelining

1. INTRODUCTION

The importance of cryptography is constantly increasing, since the amount of sensitive data being transmitted over open environments is growing at an unprecedented pace. Software-based implementations of cryptographic algorithms fall short of the required performance, as the transmission speeds of core networks reach the gigabits per second (Gbps) range. The significance and applicability of hardware-based implementations of cryptographic algorithms is therefore of interest also to the Field Programmable Gate Array (FPGA) design community.

FPGAs are nearly ideal candidates for high-speed cryptography for several reasons. The target market is generally low- to medium-sized, which makes the usage of Application Specific Integrated Circuits (ASIC) less attractive because of the large initial costs included in starting a ASIC manufacturing process. FPGA-designs also have a quicker time-to-market cycle than ASICs. A programmable platform has also applications in a multi-protocol environment, such as IPSec [5], since the cryptographic algorithm to be used can be configured on-the-fly to the target device in a fraction of a second.

The National Institute of Standards and Technology (NIST) of the United States announced in 1997 an Advanced Encryption Standard (AES) development effort to replace the Digital Encryption Standard (DES). There were five candidates in the last round of the AES algorithm selection process: MARS, RC6, Rijndael, Serpent and Twofish. In autumn 2000 the Rijndael algorithm, developed by Joan Daemen and Vincent Rijmen [4], was selected as the AES algorithm. AES was formally published on November 26 2001 in Federal Information Processing Standards' (FIPS) publication FIPS-PUB 197 [7]. The standard became effective on May 26, 2002.

The implementation of fully unrolled secret-key cryptographic algorithms is feasible on million-gate FPGAs. If the entire algorithm with full inner and outer loop pipelining fits on a single FPGA, the limiting factor for throughput is the achieved clock rate as follows [3]:

$$throughput = blocksize \times frequency \qquad (1)$$

Since the block size of AES is fixed at 128 bits, a 100 MHz clock rate implies a throughput of 12.8 Gbps. Clock rates above 100 MHz should be achieved in modern FPGAs by partitioning the design into stages and pipelining the entire system. For example, the International Data Encryption Algorithm (IDEA) with a fixed block size of 64 bits was recently implemented with a throughput of 6.78 Gbps on a Xilinx XCV1000E [11].

A typical feature of modern FPGAs is the inclusion of embedded internal memory within the device, for example BlockRAMs in Xilinx' Virtex devices [17, 18] and Embedded System Blocks (ESBs) in Altera's Apex devices [1]. This has several benefits, since lookup tables and conversion functions can be easily implemented as small RAMs within the device.

However, the amount of available internal memory may also become a bottleneck when implementing a heavily pipelined design where each stage of the pipeline requires its own unshared memory block. This may be the case with fully pipelined secret-key cryptographic algorithms, for example DES and AES, which implement non-linear substitutions with so-called S-boxes. In these cases, a smaller and less expensive target device requires implementing the design in an entirely combinatorial manner without resorting to memory accesses.

The implementation described in this paper is called SIG-AES-E, which reads as follows: SIG is the abbreviation for Signal Processing Laboratory at the Helsinki University of Technology, where the design was carried out, AES is the implemented cryptographic algorithm, and the final E means that the design performs only encryption. Implementations called SIG-AES-D (only decryption supported) and SIG-AES-ED (both encryption and decryption supported) were also designed. Design methods used in the implementations of SIG-AES-D and SIG-AES-ED were similar to the methods used in the design of SIG-AES-E. In this paper only the design of SIG-AES-E is considered in detail.

The paper is organized as follows: a summary of the AES encryption algorithm with a 128-bit key (AES-128 encryption) is presented in Section 2 and the mathematical details of mapping between different polynomial representations of GF(2^8) are described in Section 3. Section 4 contains a description of the design process with an emphasis on pipelining, and comparisons with other published FPGA-based implementations of AES-128 encryption are made in Section 5. The paper ends by drawing conclusions in Section 6 and expressing acknowledgements in Section 7.

2. THE AES-128 ALGORITHM

The Advanced Encryption Standard (AES) algorithm is a symmetric block cipher that processes data blocks of 128 bits using cipher keys with lengths of 128, 192 and 256 bits. The AES algorithm is also called the Rijndael algorithm named after its inventors, Joan Daemen and Vincent Rijmen. In this paper, only the 128 bit encryption version (AES-128 encryption) supported by SIG-AES-E is considered. A detailed specification of the AES algorithm, including AES-192 and AES-256, can be found in [7]. In the following chapters, the description generally concentrates on AES-128, but whenever the description is valid for all variants of AES, the generic abbreviation AES is used.

Data is handled mainly as bytes in the AES algorithm. One byte forms an element in a polynomial representation of Galois Field $GF(2^8)$. A byte can be represented in hexadecimal notation as $\{ab\}$, where a represents the four most significant bits (MSB) and b represents the four least significant bits (LSB) of the byte.

In this paper, the representation used in the official standard is called F_1, formally defined as $GF(2)[x]/m(x)$, where $m(x)$ is an irreducible polynomial

$$m(x) = x^8 + x^4 + x^3 + x + 1. \tag{2}$$

Additions are performed as bitwise XORs between operands in polynomial representations of F_1. Multiplications in F_1 are performed as a multiplication of the regular polynomials. The multiplication result can be a 14-degree polynomial which doesn't fit into a byte. Thus the final multiplication result in F_1 is the result of the polynomial multiplication modulo $m(x)$.

128-bit data block and key are considered as a byte array with four rows and four columns. AES-128 consists of ten rounds. One AES encryption round includes four transformations: *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*. The first and last round differ from other rounds in that there is an additional AddRoundKey transformation at the beginning of the first round and no MixColumns transformation is performed in the last round. *Key Expansion* in the AES algorithm calculates *RoundKeys* based on the original cipher key. The RoundKeys are needed in AddRoundKeys. In AES-128 encryption, the first RoundKey used in the additional AddRoundKey at the beginning of the first round is always the original key. Intermediate results after every transformation are called *States*.

The SubBytes transformation operates with every byte of the State separately. SubBytes consists of two transformations:

1. Multiplicative inverse. The zero element is mapped to itself.
2. Affine transformation which can be expressed in matrix form as:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \tag{3}$$

When it comes to FPGA-based implementations of the AES algorithm, SubBytes has usually been implemented by using a substitution table (S-Box) located in internal embedded memory, i.e. BlockRAMs in Xilinx' devices.

The ShiftRows transformation performs a cyclical left shift on the last three rows of the State. The first row is not shifted. The second row is shifted one byte, the third row is shifted two bytes and the fourth row is shifted three bytes. Thus, ShiftRows proceeds as follows:

$$s'_{r,c} = s_{r,(c+r)\,mod\,4}, \quad 0 \le r \le 3 \quad \text{and} \quad 0 \le c \le 3, \tag{4}$$

where $s_{r,c}$ is the byte (row r, column c) of the State.

The MixColumns transformation operates separately on every column of the State. A column is considered as a polynomial over F_1 and multiplied modulo $x^4 + x + 1$ with the polynomial

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}. \tag{5}$$

This results in replacing the four bytes of the column by the following equations:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \tag{6}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \tag{7}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \tag{8}$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}), \tag{9}$$

where $\bullet$ and $\oplus$ are multiplication and addition (bitwise XOR).

The AddRoundKey transformation performs an addition (bitwise XOR) of the State with the RoundKey.

The Key Expansion calculates RoundKeys for every AddRoundKey transformation. In AES-128 encryption, the original cipher key is the first RoundKey $rk[0]$ used in the additional AddRoundKey at the beginning of the first round. RoundKey $rk[i]$, where $i > 0$, is calculated from the previous RoundKey $rk[i-1]$. Let $p[j]$, where $0 \le j \le 3$, be the column j of the previous RoundKey $rk[i-1]$ and let $w[j]$ be the column j of the RoundKey being calculated. Then the new RoundKey $rk[i]$ is calculated as follows:

$$\begin{aligned} w[0] &= p[0] \oplus (\text{RotWord}(\text{SubWord}(p[3])) \oplus \text{rcon}[i]) \\ w[1] &= p[1] \oplus w[0] \\ w[2] &= p[2] \oplus w[1] \\ w[3] &= p[3] \oplus w[2]. \end{aligned}$$

`RotWord()` is a function that takes a four byte input $[a_0, a_1, a_2, a_3]$ and returns it rotated: $[a_1, a_2, a_3, a_0]$. The function `SubWord()` performs a SubBytes transformation for four bytes. The *Round constant* `rcon[i]` contains values $[x^{i-1}, \{00\}, \{00\}, \{00\}]$ where x^{i-1} are the powers of x (x is denoted as $\{02\}$) in F_1.

3. ISOMORPHISM BETWEEN F$_1$ AND F$_2$

As mentioned, the SubBytes transformation of the AES algorithm can be implemented with lookup tables located in BlockRAMs. This has obvious benefits, but in designing a fully pipelined design, the amount of available internal memory may become a bottleneck. Consequentially, a more expensive target device may be needed if every SubBytes transformation is implemented as a lookup table (See also Section 4.1).

Instead of the table implementation in F_1 it was decided to perform the SubBytes transformation by calculating the multiplicative inverse of the SubBytes in $F_2 := GF(2^4)[x]/(x^2+Ax+B)$ as described in [4] and [15]. To make this work a byte representing an element in F_1 must be transformed to a byte representing an element in F_2 [9]. All multiplications in $GF(2^4)$ are performed in $GF(2)[y]/(y^4+y+1)$. Constants A and B can be chosen freely as long as x^2+Ax+B is irreducible. In the implementation of SIG-AES-E, the constants are chosen as follows: $A = 0b0001 = \{1\}$ and $B = 0b1000 = \{8\}$. Thus the irreducible polynomial becomes x^2+x+y^3.

The problem is to find the isomorphism $\Phi : F_1 \mapsto F_2$. F_1 can also be considered as a vector space with the base $\{1, x, x^2, \ldots, x^7\}$, where x is considered as a root of $m(x)$. Thus Φ is also a linear transformation, and can be formed [8] by mapping the powers of roots in F_1 to the corresponding values in F_2. The irreducible polynomial $m(x)$ has a root $Z = \{20\}$. Calculating the powers Z^i in F_2 gives the following results:

$$\begin{aligned}
Z^0 &= 1 = \{01\}\\
Z^1 &= yx = \{20\}\\
Z^2 &= y^2x + (y^2+y) = \{46\}\\
Z^3 &= y^2x + (y^3+y^2) = \{4c\}\\
Z^4 &= (y+1)x + (y^3+y^2) = \{3c\}\\
Z^5 &= (y^3+y^2+1)x + (y^2+1) = \{d5\}\\
Z^6 &= (y+1)x + y^2 = \{34\}\\
Z^7 &= (y^3+y^2+y)x + y^2 + 1 = \{e5\}.
\end{aligned}$$

The transformation Φ in matrix form is given by:

$$\Phi = \begin{bmatrix} 1&0&0&0&0&1&0&1\\ 0&0&1&0&0&0&0&0\\ 0&0&1&1&1&1&1&1\\ 0&0&0&1&1&0&0&0\\ 0&0&0&0&1&1&1&0\\ 0&1&0&0&1&0&1&1\\ 0&0&1&1&0&1&0&1\\ 0&0&0&0&0&1&0&1 \end{bmatrix}. \qquad (10)$$

The inverse transformation $\Phi^{-1} : F_2 \mapsto F_1$ is also needed. Φ^{-1} can be defined by inverting the matrix Φ with the result as follows:

$$\Phi^{-1} = \begin{bmatrix} 1&0&0&0&0&0&0&1\\ 0&0&0&0&1&1&0&1\\ 0&1&0&0&0&0&0&0\\ 0&1&0&0&0&0&1&1\\ 0&1&0&1&0&0&1&1\\ 0&0&1&0&1&0&1&0\\ 0&1&1&1&0&0&0&1\\ 0&0&1&0&1&0&1&1 \end{bmatrix}. \qquad (11)$$

In addition to the multiplicative inverse also other transformations in the AES-128 encryption algorithm are calculated in F_2. This makes encryption faster and saves significant amounts of space since the transformations Φ and Φ^{-1} are performed only once. The transformation Φ is performed for both the key and data block at the beginning of the encryption and the inverse transformation Φ^{-1} is performed for the encrypted data block at the end of the last round. The following subsections describe how the mapping of SubBytes, MixColumns, AddRoundKey, ShiftRows and Key Expansion to F_2 was performed.

3.1 SubBytes in F$_2$

If a byte is mapped to F_2 with the transformation Φ, the multiplicative inverse can be calculated as follows [4, 15]:

$$\begin{aligned}(bx+c)^{-1} = b(b^2B+bcA+c^2)^{-1}x+\\ (c+bA)(b^2B+bcA+c^2)^{-1},\end{aligned} \qquad (12)$$

where b are the four most significant and c the four least significant bits of the byte. As already mentioned, it was chosen that $A = 0b0001 = \{1\}$ and $B = 0b1000 = \{8\}$. The multiplicative inverse $(b^2B+bcA+c^2)^{-1}$ can be calculated into a table.

Also the affine transformation defined by Equation (3) must be mapped to F_2. Since Φ is also a linear transformation, the affine transformation can be calculated as follows. Let

$$\mathbf{b}' = T\mathbf{b} + \mathbf{c} \qquad (13)$$

be the affine transformation in F_1 and let

$$\mathbf{b}'_\phi = T_\phi \mathbf{b}_\phi + \mathbf{c}_\phi \qquad (14)$$

be the affine transformation in F_2. Because

$$\mathbf{b}' = \Phi^{-1}\mathbf{b}'_\phi = \Phi^{-1}(T_\phi \mathbf{b}_\phi + \mathbf{c}_\phi) \qquad (15)$$

and $\mathbf{b}_\phi = \Phi\mathbf{b}$, Equation (13) can be expressed as

$$\mathbf{b}' = \Phi^{-1}(T_\phi(\Phi\mathbf{b}) + \mathbf{c}_\phi) = (\Phi^{-1}T_\phi\Phi)\mathbf{b} + \Phi^{-1}\mathbf{c}_\phi. \qquad (16)$$

Combining Equations (13) and (16) results in

$$T_\phi = \Phi T \Phi^{-1} \qquad (17)$$

and

$$\mathbf{c}_\phi = \Phi\mathbf{c}. \qquad (18)$$

The affine transformation in F_2 can now be expressed in matrix form:

$$\begin{bmatrix} b'_0\\ b'_1\\ b'_2\\ b'_3\\ b'_4\\ b'_5\\ b'_6\\ b'_7 \end{bmatrix} = \begin{bmatrix} 1&0&1&1&0&1&0&1\\ 1&0&0&1&0&1&1&0\\ 1&1&1&1&0&0&1&0\\ 0&1&1&1&1&0&0&0\\ 1&0&1&0&0&0&0&0\\ 0&1&0&0&1&0&1&1\\ 0&0&1&0&0&1&0&1\\ 0&0&0&1&0&1&1&1 \end{bmatrix} \begin{bmatrix} b_0\\ b_1\\ b_2\\ b_3\\ b_4\\ b_5\\ b_6\\ b_7 \end{bmatrix} + \begin{bmatrix} 0\\0\\0\\0\\0\\0\\1\\1 \end{bmatrix}. \qquad (19)$$

3.2 MixColumns in F$_2$

The MixColumns transformation of the AES-128 encryption algorithm must also be mapped to F_2. The addition in F_2 is calculated in a similar fashion as in F_1 (that is, by bitwise XORing the operands), and therefore only the multiplications must be mapped to F_2. Because Φ maps $\{01\}$ to $\{01\}$ it suffices to map only the multiplications with $\{02\}$ and $\{03\}$. Writing

$$\mathbf{a} = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \qquad (20)$$

multiplication $\{02\} \bullet \mathbf{a}$ in F_1 can be calculated as follows:

$$\begin{aligned}\{02\} \bullet \mathbf{a} &= x(a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + \\ &\quad a_1x + a_0) \\ &= a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + \\ &\quad a_1x^2 + a_0x \bmod x^8 + x^4 + x^3 + x + 1 \\ &= a_6x^7 + a_5x^6 + a_4x^5 + (a_3 + a_7)x^4 + \\ &\quad (a_2 + a_7)x^3 + a_1x^2 + (a_0 + a_7)x + a_7. \end{aligned} \tag{21}$$

Multiplication $\{03\} \bullet \mathbf{a}$ results in the following equation:

$$\begin{aligned}\{03\} \bullet \mathbf{a} &= (a_6 + a_7)x^7 + (a_5 + a_6)x^6 + (a_4 + a_5)x^5 + \\ &\quad (a_3 + a_4 + a_7)x^4 + (a_2 + a_3 + a_7)x^3 + \\ &\quad (a_1 + a_2)x^2 + (a_0 + a_1 + a_7)x + (a_0 + a_7). \end{aligned} \tag{22}$$

Equations (21) and (22) can be expressed as matrices M_2 and M_3 so that

$$\{02\} \bullet \mathbf{a} = M_2\mathbf{a} = \begin{bmatrix} 0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&1 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&1 \\ 0&0&0&1&0&0&0&1 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} \tag{23}$$

and

$$\{03\} \bullet \mathbf{a} = M_3\mathbf{a} = \begin{bmatrix} 1&0&0&0&0&0&0&1 \\ 1&1&0&0&0&0&0&1 \\ 0&1&1&0&0&0&0&0 \\ 0&0&1&1&0&0&0&1 \\ 0&0&0&1&1&0&0&1 \\ 0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&1&1&0 \\ 0&0&0&0&0&0&1&1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix}. \tag{24}$$

Matrices $M_{\phi2}$ and $M_{\phi3}$ for multiplication in F_2 can be calculated from M_2 and M_3 as follows:

$$M_{\phi2} = \Phi M_2 \Phi^{-1} = \begin{bmatrix} 0&0&0&0&1&0&0&1 \\ 0&0&0&0&1&1&0&1 \\ 0&0&0&0&0&1&1&0 \\ 0&0&0&0&0&0&1&1 \\ 0&0&0&1&0&0&0&1 \\ 1&0&0&1&1&0&0&1 \\ 0&1&0&0&0&1&0&0 \\ 0&0&1&0&0&0&1&0 \end{bmatrix} \tag{25}$$

and

$$M_{\phi3} = \Phi M_3 \Phi^{-1} = \begin{bmatrix} 1&0&0&0&1&0&0&1 \\ 0&1&0&0&1&1&0&1 \\ 0&0&1&0&0&1&1&0 \\ 0&0&0&1&0&0&1&1 \\ 0&0&0&1&1&0&0&1 \\ 1&0&0&1&1&1&0&1 \\ 0&1&0&0&0&1&1&0 \\ 0&0&1&0&0&0&1&1 \end{bmatrix}. \tag{26}$$

3.3 AddRoundKey and ShiftRows in F_2

Because addition is calculated as a bitwise XOR in both F_1 and in F_2 there is no need for changes in the AddRoundKey transformation. Also the ShiftRows transformation remains unchanged, because no calculations are required there.

3.4 Key Expansion in F_2

In the Key Expansion, the function `SubWord()` and the round constant `rcon[i]` must be mapped to F_2. `SubWord()`, which consists of four SubBytes, is mapped as described in Section 3.1. The `rcon[i]` values (powers of x) are mapped to F_2 by multipling them with the matrix Φ. The values of `rcon[i]` are presented in Table 1. All the transformations of the AES-128 encryption algorithm have now been mapped from F_1 to F_2. The encryption can be implemented as follows: first both the 128-bit data block and the 128-bit key are mapped to F_2 with the transformation Φ and then the encryption is carried out as described above. At the end of the last round the encrypted data is mapped back to F_1 with the inverse transformation Φ^{-1}.

F_1	F_2	F_1	F_2
01	01	20	d5
02	20	40	34
04	46	80	e5
08	4c	1b	51
10	3c	36	8f

Table 1: The values of `rcon[i]` in F_1 and F_2.

4. DESIGN AND IMPLEMENTATION

The AES-128 encryption implementation (SIG-AES-E) was designed fully pipelined so that a new data-key pair can be input at every clock cycle. The SIG-AES-E design has 128-bit inputs for data and key. A new data-key pair is loaded if `load` is high. Encryption of one data block requires 43 clock cycles. The output `done` is high when the encrypted data block is ready in `edata` (128-bit output).

The AES-128 consists of ten rounds. The transformation $\Phi : F_1 \mapsto F_2$ for both data and key and the additional AddRoundKey at the beginning of the first round are performed in the first block `round0`. After `round0` every block (`round1` ... `round10`) completes one round of the AES-128 encryption algorithm. Thus, SIG-AES-E consists of eleven separate blocks as presented in Figure 1. At the end of the last block the inverse transformation $\Phi^{-1} : F_2 \mapsto F_1$ is calculated.

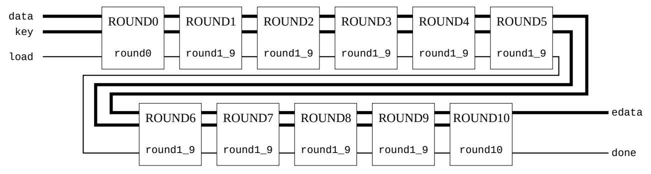

Figure 1: Block diagram of SIG-AES-E

4.1 The Target Device Families

Xilinx' Virtex-E device family [17] is an improved version of the older Virtex family. The flagship of Xilinx' Virtex series is Virtex-II [18], which has better performance and higher density than Virtex or Virtex-E. The basic unit of the Virtex devices is called *slice* and its structure is presented in Figure 2. The devices chosen as target devices for implementation were Virtex-E XCV1000E-8 with 12288 slices and Virtex-II XC2V2000-5 with 10752 slices. The area resources of the devices can be modelled so that XCV1000E has about 1.6 million and XC2V2000 about 2 million equivalent ASIC gates.

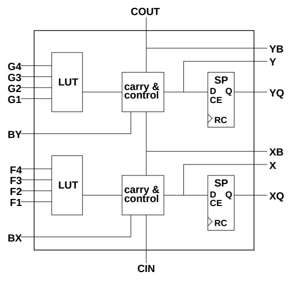

Figure 2: Virtex-E slice.

The internal memory cell in Virtex-E and Virtex-II devices is called BlockRAM, which consists of 4096 memory bits. There are varying amounts of BlockRAM in Xilinx' devices, for example, XCV1000E has 96 BlockRAM cells equalling 393216 memory bits [17].

If an AES S-box is implemented as a lookup table, $2^8 \times 8 = 2048$ memory bits are needed. This requires one half of a BlockRAM, because a BlockRAM cell can be shared between two S-boxes in dual-port mode.

If SIG-AES-E had been implemented with lookup tables, a single round would have required 8 BlockRAMs for data handling and 2 BlockRAMs for Key Expansion. Thus a single round would have required 10 BlockRAMs and the total number of required BlockRAMs for the entire ten-round pipelined design would have been 100. The smallest member of Xilinx' Virtex-E device family with enough BlockRAMs would have been XCV1600E, but because SIG-AES-E was implemented as a purely combinatorial design, the design fitted into an XCV1000E (See also Table 3).

4.2 Round0

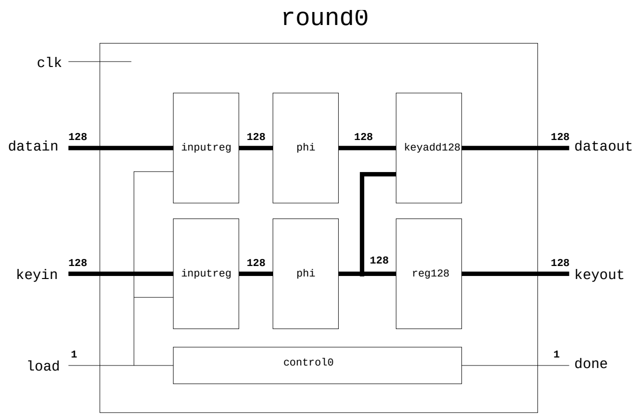

Figure 3: The inner structure of the first block.

The block diagram of the first block `round0` is presented in Figure 3. `Inputreg`s are 128-bit registers where new data and new key are loaded when `load` is high. The `phi` blocks map data and key from F_1 to F_2 as described in Section 3. The `keyadd128` is a 128-bit XOR which calculates the additional AddRoundKey of the first round of the AES-128 encryption algorithm.

`Reg128` (128-bit register) ensures that both data and key arrive to the block outputs during the same clock cycle. `Control0` includes 1-bit registers and therefore `done` follows `load` after a delay of three clock cycles. Each block in `round0`, excluding `control0`, requires one clock cycle, and `round0` is executed in three clock cycles.

4.3 Round1_9

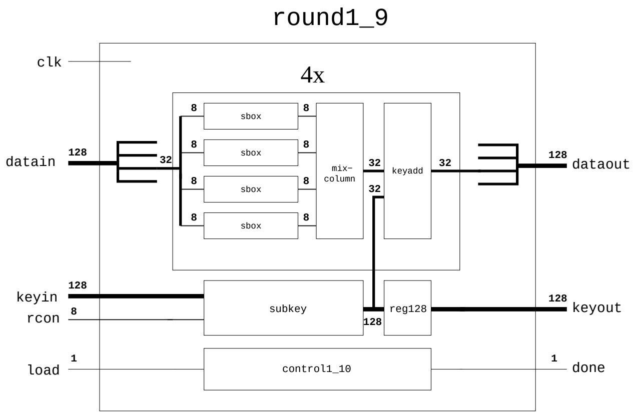

Figure 4: The inner structure of blocks 1–9.

Blocks 1–9 in Figure 1 are identical and the block diagram is presented in Figure 4. At the beginning of `round1_9` data is reorganized for the ShiftRows transformation. Each column of the data block is handled separately.

First, the SubBytes transformation is performed for every byte of the column in the `sbox` blocks. Two clock cycles are required to perform the transformation. During the first clock cycles the terms $(8b^2 + bc + c^2)^{-1}$ and $(c + b)$ in Equation (12) are calculated. The rest of Equation (12) with the affine transformation of SubBytes is calculated during the second clock cycle.

The MixColumns transformation for one column is performed in the `mixcolumn` block as presented in section 3.2. In the `keyadd` block one column of the data block is added with the corresponding column of the RoundKey with a 32-bit XOR operation.

The `subkey` block calculates new RoundKey based on the previous RoundKey (`keyin`). Details of the `subkey` operation are described in Section 4.5.

4.4 Round10

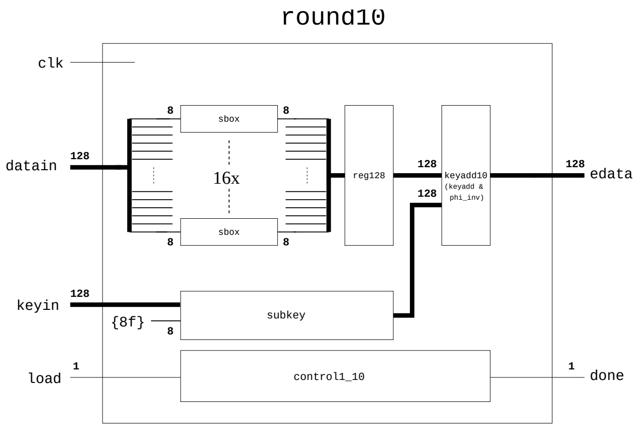

Figure 5: The inner structure of block 10.

The last block called `round10` differs slightly from `round1_9`. This can be seen in Figure 5. No MixColumns transformation is performed in the last round of the AES-128 encryption algorithm. In addition to the AddRoundKey operation the inverse transformation Φ^{-1} is also calculated in the `keyadd10` block.

The ShiftRows transformation is performed in the same way as in `round1_9`. `Reg128` must be inserted because the calculation of a new RoundKey in `subkey` requires three clock cycles (see Section 4.5) and it takes only two clock cycles to perform the SubBytes transformation in `sbox`es.

4.5 Subkey

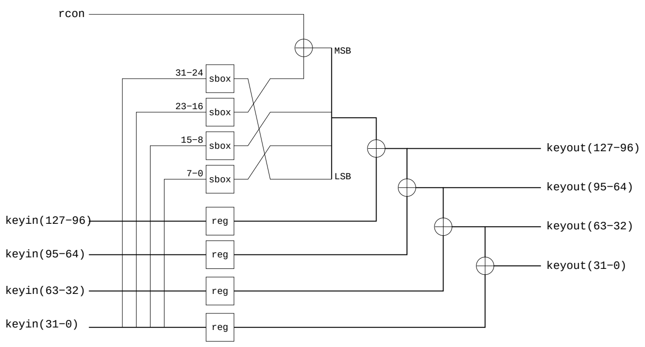

Figure 6: The calculation of a new RoundKey.

The `subkey` block performs Key Expansion of the AES-128 algorithm, which means that a new RoundKey is calculated from the RoundKey of the previous round. Details of the `subkey` calculation are presented in Figure 6.

The `SubWord()`-function of the Key Expansion is performed by four `sbox`es. They are similar to the `sbox`es described earlier, and thus two clock cycles are required to complete `SubWord()`. The rest of the `subkey` block is calculated in one clock cycle. In total, the calculation of a new RoundKey requires three clock cycles.

The `RotWord()`-function in Key Expansion is performed by reorganizing the bytes after `SubWord()`. Only the eight most significant bits of the round constant `rcon[i]` are passed to the `subkey` block because the rest of the bits are always zero. It also suffices to perform the XOR operation only with bits 16–23 because $a \oplus 0 = a$.

4.6 Synthesis and Place&Route

The implementation of SIG-AES-E was performed using VHDL as the design language and Aldec's Active-HDL as the main design tool. Synplicity's Synplify Pro 7.1 was used as the synthesis tool and Xilinx' ISE 4.1 was used as the place&route tool. The flow chart of the design process is presented in Figure 7. As mentioned in Section 4.1, Virtex-E XCV1000E-8 with 12288 slices and Virtex-II XC2V2000-5 with 10752 slices were chosen as the target devices.

As mentioned, synthesis was performed with Synplify Pro. Although the multiplicative inverses in $GF(2^4)$ were implemented as a 16x4 table (see Section 3.1), Synplify Pro was able to deduce entirely combinatorial functions for the multiplicative inverses, so that BlockRAMs were not needed. This is a substantial advantage, since the designer is not bounded by the amount of internal memory available in the target device.

The place&route was performed with Xilinx' ISE 4.1. The maximum clock frequency was 139.1 MHz for Virtex-II and the implementation required 10750 slices, which is 99% of the device's resources. The maximum clock frequency for Virtex-E was 129.2 MHz and the number of used slices was 11719 (95%). The throughput of a fully pipelined design can be calculated using Equation (1). Thus, the throughputs for the implementations are 17.80 Gbits/s for Virtex-II and 16.54 Gbits/s for Virtex-E. The main results of the implementation are presented in Table 2.

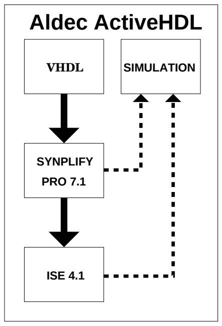

Figure 7: The flow chart of the design process.

	Virtex-II XC2V2000-5	Virtex-E XCV1000E-8
Throughput (Gbps)	17.8	16.5
Clock frequaency (MHz)	139.1	129.2
Clock cycle (ns)	7.19	7.74
Latency (ns)	318	337
Slices	10750	11719

Table 2: Summary of the implementation of SIG-AES-E.

It can be noticed from the values in Table 2 that the mapping $\Phi : F_1 \mapsto F_2$ has produced substantial benefits. Had an otherwise identical implementation with the SubBytes implemented in BlockRAMs been designed, the smallest available target device would have been a Virtex-E XCV1600E (See also Section 4.1), which is both bigger and more expensive than an XCV1000E.

As mentioned earlier, also an implementations called SIG-AES-D (supports only decryption) and SIG-AES-ED (supports both encryption and decryption) were designed. Same transformations $\Phi : F_1 \mapsto F_2$ and $\Phi^{-1} : F_2 \mapsto F_1$ were used also in the designs of SIG-AES-D and SIG-AES-ED. The matrices used in the decryption process were derived in the same way as the matrices used in SIG-AES-E.

SIG-AES-D fits into Xilinx Virtex-E XCV1000E and Virtex-II XC2V2000 devices. For Virtex-E maximum clock frequency is 124.8 MHz and for Virtex-II it is 132.4 MHz. The throughputs for SIG-AES-D implementations are 16.0 Gbits/s for Virtex-E and 16.9 Gbits/s for Virtex-II. The area requirements of SIG-AES-ED were 55% larger compared to SIG-AES-E. Thus, the smallest target device in Virtex-E family SIG-AES-ED fits in is Virtex-E XCV2000E.

5. COMPARISON

When comparing SIG-AES-E to other FPGA-based AES-128 encryption implementations, both academic and commercial designs were included. Helion Technology [12] and Amphion [2]

Design	Device	Throughput	BlockRAMs	Slices	B-RAMs/ Gbps	Slices/ Gbps
SIG-AES-E	Virtex-E XCV1000E-8	16.54 Gbps	0	11719	0	708
Weaver’s Rijndael	Virtex-E XCV600E-8	1.75 Gbps	10	770	5.71	440
GMU, Pipelined	Virtex-E XCV1000E-8	16.00 Gbps	80	9199	5.00	575
Amphion, High Speed	Virtex-E XCV50E-8*	1.06 Gbps	10	573	9.43	541
Amphion, Ultra High Speed	Virtex-E XCV1600E-8*	9.88 Gbps	100	2397	10.12	243
Helion, Fast	Virtex-E XCV400E-8*	1.19 Gbps	10	450	8.40	378
Helion, Pipelined	Virtex-E XCV????E-8	>10 Gbps	?	?	?	?
SIG-AES-E	Virtex-II XC2V2000-5	17.80 Gbps	0	10750	0	605
Amphion, High Speed	Virtex-II XC2V250-5*	1.32 Gbps	10	573	7.58	434
Amphion, Ultra High Speed	Virtex-II XC2V4000-5*	10.88 Gbps	100	2181	9.19	200
Helion, Fast	Virtex-II XC2V1000-5*	1.70 Gbps	10	450	5.88	265
Helion, Pipelined	Virtex-II XC2V????-5	>16 Gbps	?	?	?	?

Table 3: Throughput comparison of various FPGA-based AES-128 encryption implementations. * the size of the device is an estimate because Helion Technology and Amphion do not provide precise values.

	SIG-AES-E	Weaver’s Rijndael	GMU Pipelined	Amphion High Speed	Amphion Ultra High Speed	Helion Fast	Helion Pipelined
Key length							
128	✓	✓	✓	✓	✓	✓	✓
192			✓			✓	✓
256			✓			✓	✓
Modes							
ECB	✓	✓	✓	✓	✓	✓	✓
OFB				✓	✓		
CBC				✓			
CFB				✓			
Encrypt/Decrypt in the same design	(✓)		✓			(✓)	(✓)
Includes Key Expansion	✓	✓		✓	✓	✓	✓
I/O bits							
32				✓			
128	✓	✓	✓		✓	✓	✓

Table 4: Feature comparison of various FPGA-based AES(-128) implementations. (✓) there is also a version available including both encryption and decryption

sell commercial AES-128 implementations on Xilinx Virtex-E and Virtex-II devices. Both have several different cores with various features and speed grades. In this comparison only the two fastest cores from Helion and Amphion are concerned because it is not reasonable to compare cores of which the other is designed fast and the other compact-sized.

Nicholas Weaver’s Rijndael Core [19] and George Mason University’s Fully Pipelined AES implementation [10] are the academic implementations included in this comparison. It should be noticed, that the comparison list is not an exhaustive list of published FPGA-based AES-128 encryption implementations. For example, the implementation described in [14] has a throughput of 6.96 Gbps on XCV812E and the implementation described in [6] has a throughput of 1.94 Gbps on an XCV1000. However, according to the authors’ knowledge at the time of writing this paper, no other published FPGA-based implementation of AES-128 encryption exceeded the throughput of SIG-AES-E.

The fastest software implementation available at the time of writing this paper is probably Helger Lipmaa’s assembly language implementation [13]. The throughput of the Lipmaa’s implementation is about 1.65 Gbps on a Pentium IV processor running at 3.06 GHz.

5.1 Throughput Comparison

Information on throughputs and area requirements of FPGA-based AES-128 encryption implementations under comparison is presented in Table 3. The values B-RAMs/Gbps and Slices/Gbps in Table 3 illustrate the relationship between throughput and area requirements. The comparison of the area requirements of SIG-AES-E versus the other implementations is not straightforward. This is because the critical value determining the smallest device the implementation fits in is typically the number of BlockRAMs for the other FPGA-based AES-128 encryption implementation, whereas it is the number of slices for SIG-AES-E (See also Section 4.1). For example, it was estimated based on available datasheets, that Amphion Ultra High Speed requires a Virtex-E XCV1600E devices as it needs as many as 100 BlockRAMs. SIG-AES-E fits into a smaller

XCV1000E device although the value slices/Gbps is larger.

SIG-AES-E is the fastest FPGA-based AES-128 encryption implementation in the comparison and its area requirements are moderate. Amphion's fastest core, Amhion Ultra High Speed, is slower and requires a bigger target device. Helion Technology advertises that Helion Pipelined has over 16 Gbits/s throughput for Virtex-II devices, but a more detailed comparison cannot be done because Helion doesn't provide detailed information about their core. George Mason University's pipelined implementation is fast and fits in a relatively small target device. However, it requires an external Key Expansion unit, which means that the area requirements are not comparable.

The other implementations (Weaver's Rijndael, Amphion High Speed and Helion Fast) are significantly slower because of the lack of pipelining, but they also fit into a smaller target device.

5.2 Feature Comparison

Information on features of the FPGA-based AES implementations is collected in Table 4, and it can be noticed that there is a lot of variation between different implementations. SIG-AES-E supports only 128-bit key length, but at least at the present time, AES-128 is more popular than AES-192 or AES-256. Amphion's High Speed and Ultra High Speed cores also support only 128-bit key, but Amphion has cores (Amphion Standard) supporting also 192 and 256-bit key lengths.

George Mason University's implementation includes both encryption and decryption modes in the same device, and also Helion Technology has versions with the same feature. In this comparison the version of Helion Technology's AES cores supporting only encryption is considered. As mentioned at the end of Section 4.6, also SIG-AES-ED (both encryption and decryption supported in the same device), was designed, but the area requirements were 55% larger than in SIG-AES-E.

Every implementation naturally supports the ECB (Electronic Codebook) mode of operation [16]. Certain implementations support also other modes of operation, as can be seen in Table 4. Amphion High Speed has the most versatile mode support, as it supports ECB, OFB (Output-Feedback), CBC (Cipher Block Chaining) and CFB (Cipher-Feedback) modes.

CBC and CFB require previous cipher data to calculate the next cipher data, which makes it impossible to implement these modes of operation in a fully pipelined fashion. On the other hand, the OFB mode can be implemented in a fully pipelined fashion, and it is supported by Amphion Ultra High Speed, a fully pipelined implementation. If other modes than ECB and OFB are required, a slower alternative must be chosen.

Regarding key expansion, it has to be noted that GMU's implementation requires an external Key Expansion which can be regarded as a disadvantage. Amphion High Speed uses 32-bit inputs and outputs instead of 128-bit inputs and outputs used by other implementations in this comparison. The benefit of a smaller number of I/O-lines is obvious, since also smaller target devices with a limited number of input/output-pins can be used. As a disadvantage, encryption slows down significantly.

5.3 Summary of the Comparison

At the moment, SIG-AES-E appears to be the fastest available FPGA-based implementation, when very high-speed AES-128 encryption is needed. SIG-AES-E also fits into the smallest target device as compared to other fully pipelined designs (GMU also fits into a Virtex-E XCV1000E, but an external Key Expansion unit is also required).

If versatile key length support is needed, Helion Fast and Pipelined implementations are good choices. The comparison of the Helion Pipelined core was difficult, because Helion did not provide any detailed information about this core. As a general note, Helion's cores seem to provide fast encryption with versatile features.

Amphion's cores support various modes of operation, but the fastest two of them support only 128-bit key length. They are also slower than SIG-AES-E and Helion's cores. On the other hand, Amphion High Speed provides moderate throughput mixed with reasonable area requirements and a versatile mode support.

Nicholas Weaver's Rijndael Core is faster than Amphion High Speed and Helion Fast. Weaver's Rijndael Core also fits into a Virtex-E XCV600E device, which makes it a good alternative for the commercial cores.

GMU's implementation supports all key lengths but requires a 128-bit RoundKey from an external Key Expansion unit. In other words, supporting different key lengths is partially delegated to an external device.

6. CONCLUSIONS AND FUTURE WORK

A memoryless implementation called SIG-AES-E of the AES-128 encryption algorithm was designed for Xilinx' Virtex-E and Virtex-II devices. The implementation requires no embedded memory, which is typically a limiting factor in fitting fully pipelined secret-key cryptographic algorithms, because the S-boxes have traditionally been implemented as lookup tables within the programmable device.

The SIG-AES-E is a fully combinatorial implementation, because the computation of the multiplicative inverse in F_1 is transformed into F_2. This divides the 8-bit argument into 4-bit MSB and LSB parts, which enables the computation of the multiplicative inverse as described in Equation (12).

The SIG-AES-E has a throughput of 17.80 Gbps on a Virtex-II XC2V2000-5 with a clock frequency of 139.1 MHz and requires 10750 slices. On an XCV1000E-8, the corresponding numbers are 16.54 Gbps throughput with a clock frequency of 129.2 MHz and 11719 required slices. To the authors' knowledge, SIG-AES-E is the fastest published FPGA-based implementation of the AES-128 encryption algorithm.

Future work includes searching for an optimum transformation for both encryption and decryption. The area requirements might be slightly reduced by finding a transformation Φ' that minimizes the number of ones in the transformation matrices. Every one in a matrix requires one XOR-operation and therefore the number of ones should be kept as small as possible.

Additional future work involves research into the applicability of partial runtime reconfiguration with regard to block sharing between encryption and decryption modes. Also support for AES-192 and AES-256 is being considered.

7. ACKNOWLEDGEMENTS

The authors would like to express their gratitude to Mr. Joonas Pihlaja of University of Helsinki for his helpful comments on calculating multiplicative inverses in different representations of $\mathrm{GF}(2^8)$ and Mr. Jan Eriksson of the Signal Processing Laboratory at the Helsinki University of Technology for thoroughly reviewing the mathematical expressions and providing valuable comments on the theoretical background of Galois fields.

This research was performed within the GO project (see website: `http://go.cs.hut.fi`), a three-year multilaboratory project at the Helsinki University of Technology, financed by the National Technology Agency of Finland and several Finnish telecommunications companies.

8. REFERENCES

[1] Altera. APEX II Programmable Logic Device Family Data Sheet. `www.altera.com/literature/ds/ds_ap2.pdf`.

[2] Amphion. `www.amphion.com`.

[3] P. Chodowiec, P. Khuon, and K. Gaj. Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer-Round Pipelining. *Proceedings of the ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays, Monterey, California, USA*, pages 94–102, February 11-13 2001.

[4] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag Berlin Heidelberg, 2002.

[5] A. Dandalis and V. K. Prasama. An Adaptive Cryptographic Engine for IPSec Architectures. *in Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2000), Napa Valley, California, USA*, pages 132–131, 2000.

[6] A. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9:545–557, August 2001.

[7] FIPS. Advanced Encryption Standard (AES). *FIPS PUB 197*, November 26 2001. `csrc.nist.gov/publications/fips/ ... fips197/fips-197.pdf`.

[8] J. B. Fraleigh. *A First Course in Abstract Algebra*. Addison-Wesley Publishing Company, fourth edition, 1989.

[9] J. B. Fraleigh and R. A. Beauregard. *Linear Algebra*. Addison-Wesley Publishing Company, second edition, 1990.

[10] George Mason University. Hardware IP Cores of Advanced Encryption Standard AES-Rijndael. `ece.gmu.edu/crypto/rijndael.htm`.

[11] A. Hämäläinen, M. Tommiska, and J. Skyttä. 6.78 Gigabits per Second Implementation of the IDEA Cryptographic Algorithm. *in Proceddings of the 12th Conference on Field-Programmable Logic and Applications, FPL 2002, La Grande Motte, France*, pages 760–769, September 2002. Manfred Glesner, Peter Zipf and Michel Renovell (eds.).

[12] Helion Technology Limited. `www.heliontech.com`.

[13] H. Lipmaa. AES implementation speed comparison. `www.tcs.hut.fi/∼helger/aes/rijndael.html`.

[14] M. McLoone and J. V. McCanny. Single-Chip FPGA Implementation of the Advanced Encryption Standard Algorithm. *in Proceedings of the 11th Conference on Field-Programmable Logic and Applications, FPL 2001, Belfast, Northern Ireland, UK*, pages 152–161, August 2001. Gordon Brebner and Roger Woods (eds.).

[15] V. Rijmen. Efficient Implementation of Rijndael S-box. `www.esat.kuleuven.ac.be/∼rijmen/ ... rijndael/sbox.pdf`.

[16] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., second edition, 1996.

[17] Virtex-E. Xilinx' Virtex-E Datasheet. `www.xilinx.com/partinfo/ds022.pdf`.

[18] Virtex-II. Xilinx' Virtex-II Datasheet. `www.xilinx.com/partinfo/ds031.pdf`.

[19] N. Weaver. Rijndael core. `www.cs.berkeley.edu/∼nweaver/rijndael`.