HARDWARE IMPLEMENTATION OF AES ALGORITHM

Marko Mali — Franc Novak — Anton Biasizzo *

The paper presents a hardware implementation of the AES algorithm developed for an external data storage unit in a dependable application. The algorithm was implemented in FPGA using the development board Celoxica RC1000 and development suite Celoxica DK. The purpose of this prototype version was to test the correctness of the implemented algorithm and to gain experience in optimisation of algorithm structure for the prospective embedded implementation in the target application.

K e y w o r d s: AES algorithm, hardware implementation

1 INTRODUCTION

Cryptography plays an important role in the security of data. It enables us to store sensitive information or transmit it across insecure networks so that unauthorized persons cannot read it. The urgency for secure exchange of digital data resulted in large quantities of different encryption algorithms which can be classified into two groups: asymmetric encryption algorithms (with public key algorithms) and symmetric encryption algorithms (with private key algorithms) [1]. Symmetric key algorithms are in general much faster to execute electronically than asymmetric key algorithms.

In our case, an external data storage unit in a dependable application has been designed and different symmetric key algorithms have been considered for data encryption. We first considered Data Encryption Standard (DES) as possible candidate because the implementation is relatively simple [2]. Later, however, this option was rejected due to its vulnerability. As an alternative, AES algorithm was implemented and the results of the experimental feasibility study are reported in the paper.

The algorithm originates from the initiative of the National Institute of Standards and Technology (NIST) in 1997 to select a new symmetric key encryption algorithm. From the initial candidates [3], Rijndael algorithm was selected as the Advanced Encryption Standard (AES) [4] due to the combination of security, performance, efficiency, ease of implementation and flexibility.

Rijndael is a symmetric byte-oriented iterated (each iteration is called a round) block cipher that can process data blocks of 128 bits (4 words), using keys with length of 128, 192 and 256 bits. Rijndael is capable of processing additional block sizes (160, 192 and 244 bits) and key lengths (160 and 244 bits), however they are not adopted in AES. Our implementation refers to AES algorithm.

The algorithm is composed of three main parts: Cipher, Inverse Cipher and Key Expansion. Cipher converts data to an unintelligible form called ciphertext while Inverse Cipher converts data back into its original form called plaintext. Key Expansion generates a Key Schedule that is used in Cipher and Inverse Cipher procedure. Cipher and Inverse Cipher are composed of specific number of rounds (Table 1). For the AES algorithm, the number of rounds to be performed during the execution of the algorithm is dependent on the key length [4].

Table 1.

	Block Size N_b words	Key Length N_k words	Number of Rounds N_r
AES – 128-bits key	4	4	10
AES – 192-bits key	4	6	12
AES – 256-bits key	4	8	14

For both its Cipher and Inverse Cipher, the AES algorithm uses a round function that is composed of four different byte-oriented transformations: SubBytes, ShiftRows, MixColumns and AddRoundKey.

Inputs and outputs: The input and output for the AES algorithm each consists of sequences of 128 bits. The Cipher Key for the AES algorithm is a sequence of 128, 192 or 256 bits. The basic unit for processing in the AES algorithm is a byte (a sequence of eight bits), so the input bit sequence is first transformed into byte sequence. In the next step a two-dimensional array of bytes (called the State) is built. The State array consists of four rows of bytes, each containing N_b bytes, where N_b is the block size divided by 32 (number of words). All internal operations (Cipher and Inverse Cipher) of the AES algorithm's are then performed on the State array, after which its final value is copied to the output (State array is transformed back to the bit sequence).

Cipher: Using round function, which is composed of four different byte-oriented transformations, the Cipher converts input data (the input data is first copied to the

* Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

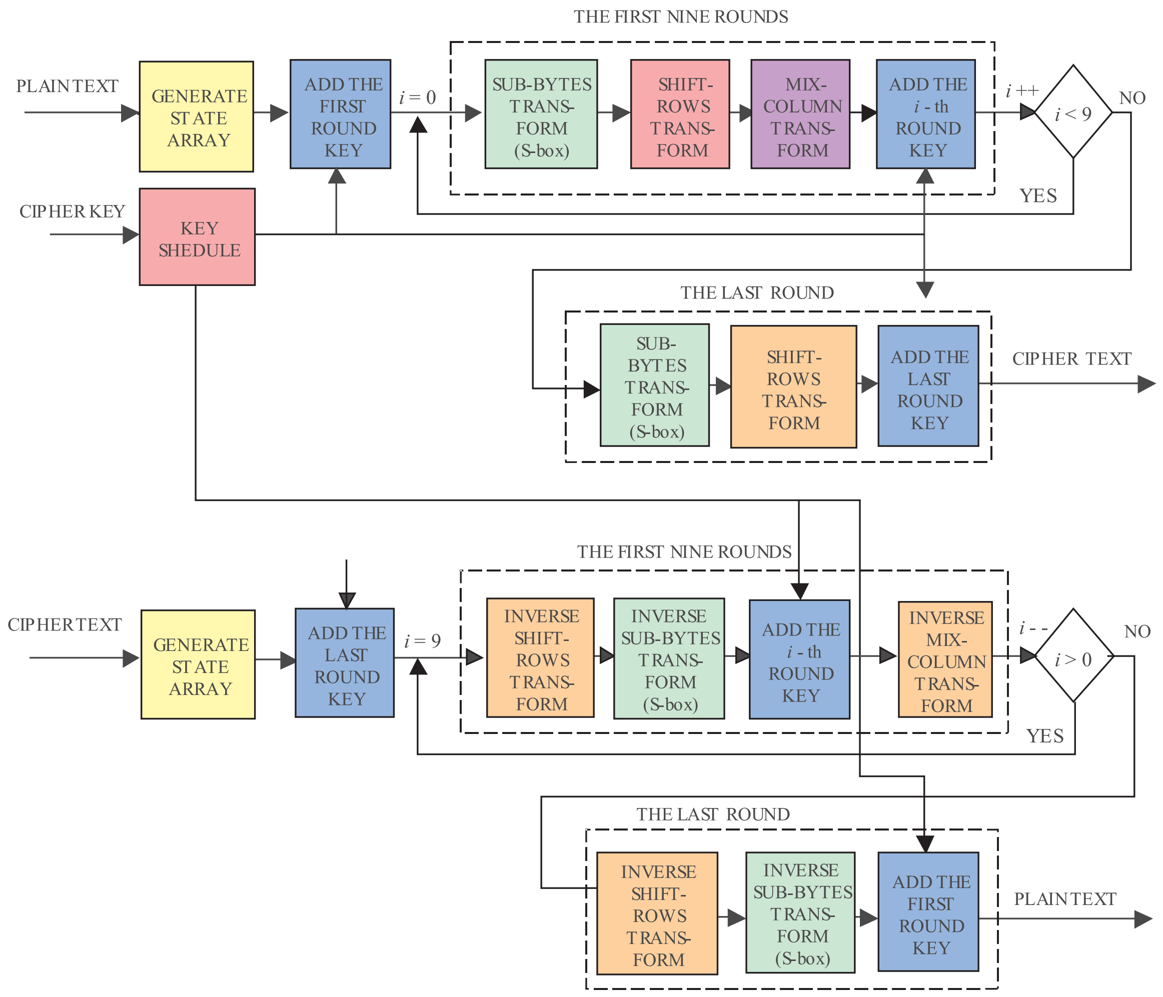

Fig. 1. Structure of AES algorithm

State array) to an unintelligible form called ciphertext. After an initial Round Key addition, the State array is transformed by implementing a round function with the final round differing slightly from the first $N_r - 1$ rounds. The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words (Round Key) derived using the Key Expansion routine.

All N_r rounds (see Table 1) are identical with the exception of the final round, which does not include the MixColumns transformation.

Key Schedule: The AES algorithm takes the Cipher Key and performs a Key Expansion routine to generate a Key Schedule. The Key Expansion generates a total $N_b(N_r + 1)$ words ($N_r + 1$ Round Keys).

Inverse Cipher: At the start of the Inverse Cipher, the input (ciphertext) is copied to the State array. After Round Key addition (the last Round Key is added), the State array is transformed by implementing a round function, that is composed of three different inverse transformations and AddRoundKey transformation (Round Keys are applied in the reverse order when decrypting), with the final round differing slightly from the first $N_r - 1$ rounds. So this procedure converts ciphertext back to its original form called plaintext.

All N_r rounds are identical with the exception of the final round, which does not include the Inverse Mix-Columns transformation.

The whole AES algorithm is sketched in Figure 1.

2 HARDWARE IMPLEMENTATION OF AES ALGORITHM

We have implemented AES algorithm in a field programmable device (FPD) and compared the speed of hardware implementation with the speed of Rijndael Reference Implementation in C++ [4]. The results and description of hardware and software equipment, which was used to implement AES algorithm in hardware, are presented in the following subsections.

2.1 Celoxica RC1000

We implemented the AES algorithm on Celoxica RC1000 hardware platform, which is a standard PCI bus card equipped with a XILINX$^{®}$ Virtex^{TM} family BG560

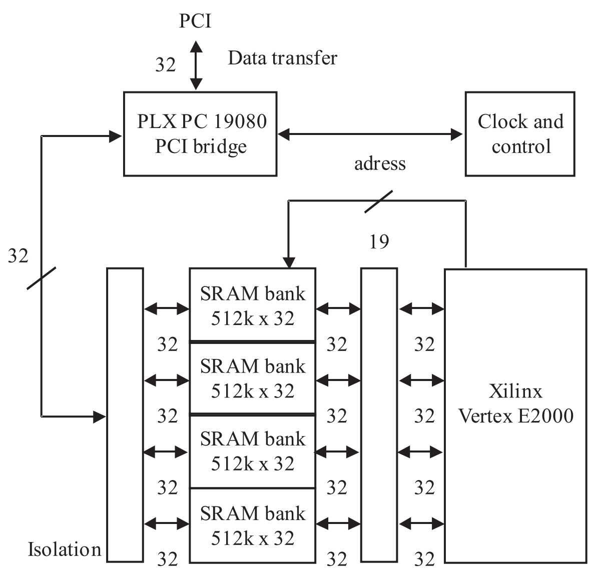

Fig. 2. Functional Block Diagram of RC1000

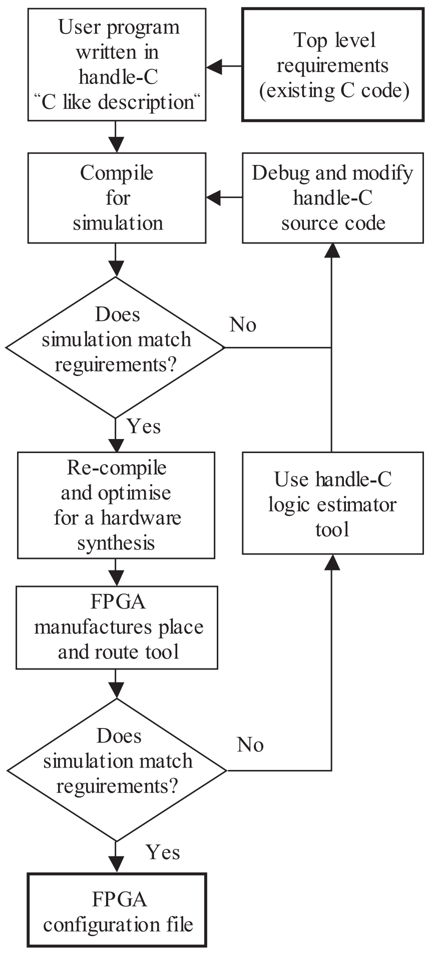
User program written in handle-C "C like description"
Top level requirements (existing C code)
Compile for simulation
Debug and modify handle-C source code
Does simulation match reguirements?
No
Yes
Re-compile and optimise for a hardware synthesis
Use handle-C logic estimator tool
FPGA manufactures place and route tool
Does simulation match reguirements?
No
Yes
FPGA configuration file

Fig. 3. Handle-C design flow

part with up to 1,000,000 system gates. It has 8 Mb of SRAM directly connected to the FPGA in four 32 bit wide memory banks. The memory is also visible to the host CPU across the PCI bus as if it was normal memory. Each of the 4 banks may be granted to either the host CPU or the FPGA at any time. Data can therefore be shared between the FPGA and host CPU by placing it in the SRAM on the board. It is then accessible to the FPGA directly and to the host CPU either by DMA transfers across the PCI bus or simply as a virtual address. High speed Direct Memory Access (DMA), data buffering and clock speed control make it suitable for high speed cryptographic applications.

The steps used to implement the AES algorithm on the RC1000 development board are as follows:

1. Download the Cipher design to the FPGA.
2. Write any constants or keys for the cipher from the host (PC) over the PCI to the SRAM banks.
3. Write the plaintext to the other available addresses of the SRAM banks.
4. Send an event signal to the FPGA to begin processing data.
5. Poll the FPGA until it signals that it has finished processing data.
6. Write the resulting ciphertext back over the PCI to the host.

2.2 Celoxica DK and Handel-C

The DK Design Suite software provides the environment necessary to implement the target software-compiled design described in a C-based design language Handel-C in FPGA [6]. It enables the partitioning, verification and implementation of complex algorithms in programmable logic. The Handel-C syntax is based on that of conventional C, so programmers familiar with conventional C can easily recognize the constructs in the Handel-C language [7]. Handel-C also includes parallel constructs that may considerably speed up the application. The compiler compiles and optimizes Handel-C source code into a file suitable for simulation (netlist). The resulting file can be placed and routed on a real FPGA using XILINX WEBISE Kit. There are, however, some restrictions that the designer must be aware of when writing the code in Handel-C or translating the source C code into Handel-C (*ie* , floating point variables are not supported, functions may not be recursive).

2.3 Performance comparison between hardware and software implementation

The hardware implementation of AES algorithm was realized in a way, that it was possible to perform each function of algorithm independently (Figure 4). An event command was transmitted (using status and control register of RC1000) from host to FPGA, which defined a particular function to be executed. Therefore we were capable to determine the executing time of individual function. In order to demonstrate the advantage of parallel implementation of AES algorithm, we implemented AES algorithm once using the parallelism (PAR) and once without using it (SEQ).

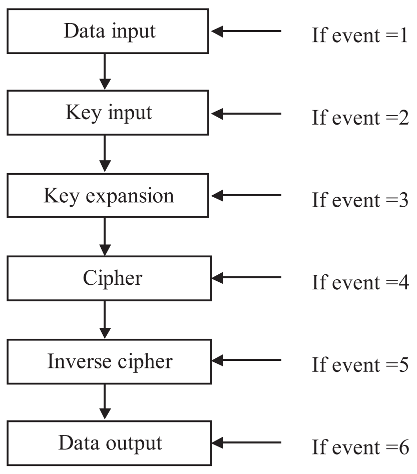

Fig. 4. Structure of hardware implementation of AES algorithm (event execution)

Table 2.

function	average execution times (μs)	throughput (Mbit/s)
sending control and status messages	64.70	–
DMA transfer	256.65	–
Key Expansion	8.86	14.44
Cipher SEQ	22.91	5.59
Cipher PAR	0.70	182.86
Inverse Cipher SEQ	22.91	5.59
Inverse Cipher PAR	0.70	182.86

Table 3.

function	average execution times (μs)	throughput (Mbit/s)
Key Expansion	4.71	27.18
Cipher	16.50	7.76
Inverse Cipher	20.56	6.23

The average execution times of different functions by hardware implementation of AES algorithm (by encoding/decoding one block-128bit of data) are presented in Table 2. The average execution times of different functions by software implementation (reference implementation by Brian Gladman in Visual C++) of AES algorithm (AMD Athlon 1.3 GHz processor was used) are presented in Table 3.

The data in Table 2 show the benefit of using parallelism. The Cipher is executed 110 % faster using parallel implementation then using sequential implementation. The parallel Inverse Cipher execution is also much faster (1.38 times) in comparison by sequential Inverse Cipher.

Comparison of data in Table 2 with data in Table 3 leads to some further conclusions:

- The Cipher, implemented in hardware (parallel implementation) is executed approximately 23 times faster than the Cipher implemented in software.
- The Inverse Cipher, implemented in hardware (parallel implementation) is executed approximately 29 times faster than the Inverse Cipher implemented in software.
- The Key Expansion function implemented in hardware is executed 1.88 times slower than the Key Expansion function implemented in software.
- The DMA transfer, which is used to transfer data to and from the SRAM banks respectively, consumes a large amount of time.

As described above, the Key Expansion function generates the Key Schedule using the input Cipher Key. In hardware implementation the Key Schedule was stored in a block RAM (BRAB) of FPGA. Because of such implementation, parallel execution of Key Expansion function was limited (this is the reason for slow execution in hardware implementation in comparison with software implementation). Notice, however, that the Key Schedule in the case of encrypting a large amount of data is generated only once. On the other hand, the Cipher and Inverse Cipher function are performed on separate blocks (128 bit) of data until the whole amount of data is encrypted. The final performance of the hardware implementation of AES algorithm is therefore practically not affected by the speed of the Key Expansion function execution.

As we can see from Table 2, the DMA transfer consumes large amount of time. Similar to the Key Expansion function, the DMA transfer is also performed only once for fixed amount of data. So in the case of encrypting small amount of data ($<$ 100 Kbytes) the final performance of software implementation is better than hardware implementation. The main reason for that is the low speed of DMA transfer. In the case of encrypting large amount of data ($>$ 100 Kbytes) the speed of Cipher and Inverse Cipher execution (that are performed many times) contribute the main part to the final performance of algorithm. In this case, the hardware implementation of AES algorithm is much faster than the software implementation.

With timing analysis and optimization of Handel-C design of the AES algorithm hardware implementation can be further improved. Currently our hardware implementation of the AES algorithm works with a clock rate of 74.4 MHz which meets the needs of the target application.

2.4 Performance comparison with other FPD implementations

Initial comparative study of AES final candidates using FPGAs is given in [7]. The study compares the performance of individual steps of algorithms for different architectural options which makes the comparison with our implementation rather difficult. On the other hand our design is inferior by a factor of 5 to the fast configuration with T-boxes reported by V. Fischer and M. Drutarovsky [8] and comparable to some other FPGA implementations referenced in the comparison study in the above paper, like for example [9]. The reason partially lies in the fact that different goals were pursued: instead of striving for a fast solution suitable for specific FPD structure we tried to fulfil the requirements of the given target application by direct translation of C++ code to Handel-C design of the FPGA supported by the available development tools. While the penalty of such an approach are lower execution times, it also has the advantages: it requires less effort and provides better portability among different FPDs.

3 CONCLUSION

Hardware implementation of the AES algorithm developed for an external data storage unit in a dependable application is described. The AES algorithm was implemented in FPGA using the development board Celoxica RC1000 and development suite Celoxica DK. Achieved performance proved to be satisfactory for the requirements of the target application so that the embedded data storage controller with cryptographic feature could be designed. The results presented in the paper might be a helpful initial assessment of the performance of FPGA implementations of algorithms described in a high level design language Handel-C.

References

[1] SCHNEIER, B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C, John Wiley & Sons, 1996.

[2] MYLONAS, M.—HOLDING, D. J.—BLOW, K. J.: DES Developed In Handel-C, 2002 (http://www.ee.ucl.ac.uk/lcs/papers2002/LCS057.pdf).

[3] FIPS PUB 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, U.S. Department of Commerce, November 2001 (http://csrc.nist.gov/ publications/fips/fips197/fips-197.pdf).

[4] DAEMEN, J.—RIJMEN, V.: AES Proposal: Rijndael, The Rijndael Block Cipher, AES Proposal, pp. 1–45, 1999 (http://csrc.nist.gov/CryptoToolkit/aes/).

[5] Celoxica, RC1000 Hardware Reference Manual, version 2.3, 2001.

[6] Celoxica, Handel-C Language Reference Manual, version 3.1, 2002.

[7] DANDALIS, A.—PRASANNA, V. K.—ROLIM, D. P.: A Comparative Study of Performance of AES Final Candidates Using FPGAs, the third AES conference, New York, April 2000 (http://csrc.nist.gov/encryption/aes/).

[8] FISCHER, V.—DRUTAROVSKY, M.: Two Nethods of Rijndael Implementation in Reconfigurable Hardware, Proc. of 3rd Int. Workshop on Cryptographic Hardware and Embedded Systems, May 2001, pp. 77–92.

[9] BORA, P.—CZAJKA, T.: Implementation of the Serpent Algorithm Using Altera FPGA Devices (http://csrc.nist.gov/ CryptoToolkit/aes/round2/comments/20000513-pbora.pdf).

Received 29 June 2005

Marko Mali gained the BSc degree from the University of Ljubljana, Faculty of electrical engineering. He is now working toward PhD degree at the Jozef Stefan International Postgraduate School.

Franc Novak gained the BSc, MSc, and PhD degrees in electrical engineering from the University in Ljubljana in 1975, 1977, and 1988, respectively. Since 1975 he has been with the Jozef Stefan Institute, where he is currently head of Computer Systems Department. Since 2001 he is also assoc. professor, at Faculty of Electrical Engineering and Computer Science, University of Maribor. His research interests are in the areas of electronic testing and diagnosis, and fault-tolerant computing. His most recent assignment has been on design for testability of analogue circuits.

Anton Biasizzo is a researcher at Joef Stefan Institute since 1991. He received the BSc, MSc, and PhD degrees from the University in Ljubljana in 1991, 1995, and 1998, respectively. His research interests include efficient algorithms for sequential diagnosis, constraint logic programming, model based diagnosis and automatic test pattern generation.