ICSES 2008 INTERNATIONAL CONFERENCE ON SIGNALS AND ELECTRONIC SYSTEMS
KRAKÓW, SEPTEMBER 14-17, 2008

AES hardware implementation in FPGA for algorithm acceleration purpose

Artur Gielata
AGH-UST. Dept. of Electronics
Mickiewicza 30, 30-059 Cracow, Poland
e-mail: gielata@agh.edu.pl

Pawel Russek
AGH-UST. Dept. of Electronics
ACC Cyfronet AGH
Cracow, Poland
e-mail: russek@agh.edu.pl

Kazimierz Wiatr
AGH-UST. Dept. of Electronics
ACC Cyfronet AGH
Cracow, Poland
e-mail: wiatr@agh.edu.pl

***Abstract*—In this paper we investigate hardware implementation of AES-128 cipher standard on FPGA technology. In many network applications software implementations of cryptographic algorithms are slow and inefficient. To solve that problems custom architecture in reconfigurable hardware was proposed to speed up the performance and flexibility of Rijndael algorithm implementation. We aimed to achieve the maximum speed and efficiency of cipher process, therefore pipeline architecture of AES module was proposed. The investigations involved simulations and synthesis of VHDL code utilizing Virtex4 series of Xilinx.**

***Index Terms*—cryptography, AES, FPGA, hardware acceleration,**

I. INTRODUCTION

At present a majority of computer and telecommunication systems requires data security when data is transmitted over network. Thus data encryption is performed to protect intrusion sensible data. Usually appropriate software algorithm is used for coding data at sender site and decode at receiver one. Such a solution is not adequate and too slow then high speed processing is necessary due to high transmission medium bandwidth and real time requirements for example. In such situation increase of computational platform processor performance is necessary, though usage of faster general purpose processor is not effective. That is why hardware acceleration of cryptographic algorithms is necessary. The very good solution of choice for such dedicated hardware are reconfigurable devices. For integer based data this technology guarantee better performance and lower power demand. Additionally because of the distinguished features of hardware solution it also provide better data security against crackers' attacks.

In this paper dedicated hardware is proposed for Rijndael algorithm [1]. The Rijndael is award winning algorithm for a successor of Data Encryption Stndard (DES) algorithm. The DES which was cracked in 1997 and considered as to be not safe enough. The Rijndael is categorized as a iterative, block based cipher algorithm what means that both input data and encryption key are processed in several rounds of transformations before output data is ready. Both encryption and decryption algorithms are very similar. Data is processed in blocks. The size of block and key length may vary but the standards are the 128-bit data block size and 128-bit, 192-bit or 256-bit key lengths. In presented work base version of Rijndael algorithm was chosen for implementation i.e.: both block and key sizes are 128-bit. This version is stated as AES-128 cryptographic standard.

II. RIJNDAEL ALGORITHM

The Rijndael cipher algorithm gained a reputation of not only fast but also very secure method. Despite several years of research no real algorithm security problems were reported [2]. According NIST (National Institute of Standard and Technology) organization, to be able to crack AES-128 coded data, computer which is able to crack 56-bit DES encrypted data in one second would need billion of years. The security level of AES algorithm when 192-bit and 256-bit keys are used is high enough to be approved by NSA (National Security Agency) to be used for protection of most confidential data [3]. Today AES and Rijndael are widely utilized in such areas like administration, diplomacy and military.

As to explain Rijndael algorithm operations [4] we have to start with assumption that for its execution both data and key are represented as a matrix first. Matrixes are constructed according the following rules:

1) matrix element is represented as 8-bit integer,
2) number of matrixes rows (Nb) is constant despite the data/key size and is equal to 4,
3) number of data matrix columns (Ns) depends on data sizes and is equal to data size divided by 32,
4) number of key matrix columns (Nk) depends on key sizes and is equal to key size divided by 32.

Three operational phases makes the whole Rijndael algorithm encryption procedure. The first phase executes XOR operation of input data and key origin data. The first phase is commonly called AddRoundKey. The second phase consist of several rounds. The number of rounds (Nr) depends on the version of algorithm, particularly depends on the key size. In

our implementation it is equal to 9. After execution of each round the new temporary encryption result is generated. It is called *State*. In second phase Nr-1 similar rounds are executed. Those rounds consist of *State* matrix transformations. Four basic transformation are performed in single round. If we consider encryption those transformations are:

1) SubBytes
2) ShiftRows
3) MixColumns
4) AddRoundKey

In the third phase final round is performed. The only difference between round in third phase and the rounds in second phase is that in third phase there is no MixColumns round.

Decryption process is performed according reversed encryption scheme although mentioned earlier transformations have different definition to be complementary to encryption transformations. Only AddRoundKey is identical for encryption and decryption. Also the same keys are used for rounds in decryption as they were use in encryption. Such for decryption round the following transformations are in use:

1) AddRoundKey
2) InvMixColumns
3) InvShiftRows
4) InvSubBytes

For decryption, in first phase the round derprived of InvMixColumn transformation is performed. The second phase consist of Nr-1 full rounds and in the last third phase only AddRoundKey transformation is performed. Table I presents Ns, Nk and Nr parameters of three different versions of encryption processes described for AES standard. In practice different combinations for data block size and key length are possible but only those presented in table I are considered in a USA government standard so any commercial versions of AES standard are also limited to above parameters.

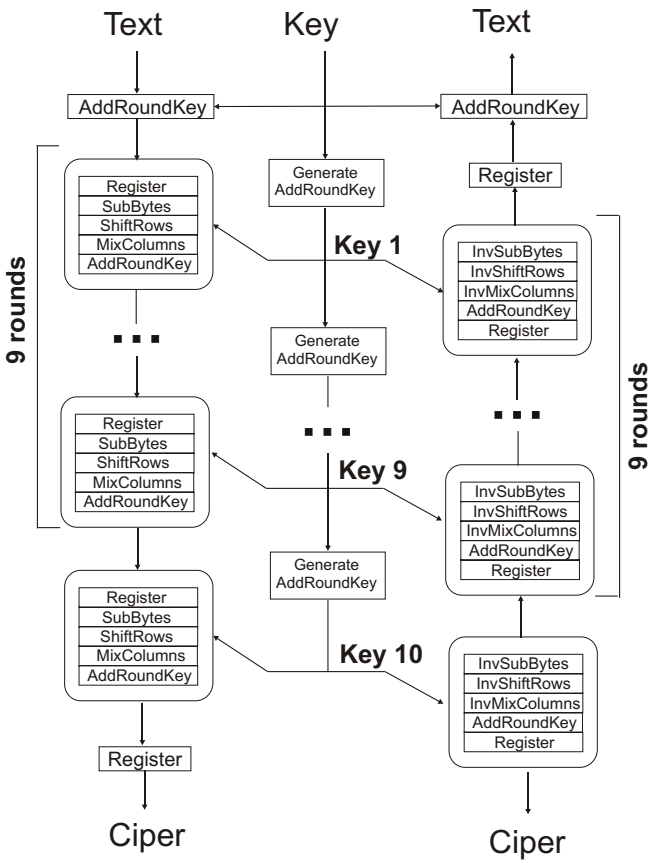

Fig. 1. Pipeline architecture of encrypt and decrypt module (8pt)

Data: byte key[4*Nk], Nk
Result: word w[Nb*(Nr+1)]
word temp;
i = 0;
while *i < Nk* **do**
  w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3]);
  i = i+1;
end
i = Nk;
while *i< Nb*(Nr+1)* **do**
  temp = w=[i-1];
  if *i mod Nk=0* **then**
    temp = SubWord(RotWord(temp)) xor Rcon[i/Nk];
  else
    if *Nk > 6 and i mod Nk=4* **then**
      temp = SubWord(temp);
    end
    w[i] = w[i-Nk] xor temp;
    i = i+1;
  end
end

Algorithm 1: KeyExpantion algorithm

TABLE I
PARAMETERS OF STANDARD AES ALGORITHM VERSIONS

Algorithm version	Parameters		
	Ns	Nk	Nr
AES-128	4	4	10
AES-192	6	6	12
AES-256	8	8	14

Full description of Rijndael algorithm requires notice about key extension procedure. Key extension procedure is performed to generate from the origin key a table of different keys. Each entry in that keys table is used in AddRoundKey transformation of subsequent rounds. For AES-256 version key extension procedure were slightly modified. For both coding and decoding round keys are the same and only the sequence of their usage is reversed. Keys matrix has the size adequate to *State* matrix despite the selected key length so for each AddRoundKey transformation extended key has 128-bit length. Key extension procedure for AES-128 and AES-192 follows according algorithm 1. Function SubWord performs LUT operation according which a 4-bajt word is selected from S-box table created in SubBytes transformation. RotWord function perform cyclic bytes rotation of single 4-bajt word. Rcon is a register which contain precisely predefined valued for subsequent phases of key generation.

III. IMPLEMENTATION

One of the main constrains considered than the winner of the competition for the best cipher algorithm was chosen was the simplicity. That meant that the best algorithm was easy to implement both in software and in hardware. Inventors of the AES algorithm [5] designed it with an idea in mind of ability for its efficient execution using different platform: for example on processors and microcontrollers or in custom hardware such as ASICs and reconfigurable logic [6]. Despite that Rijndael algorithm belongs to the category of iterative blocks algorithms, it is also suitable for parallel and pipeline implementation [7]. In order to achieve the best coding and decoding throughput for AES-128 implementation authors decided to design fully pipeline architecture. This pipeline architecture is presented on figure 1. In this solution before each processing step the temporary *State* value is latched into the 128-bit register. Because of that coded value appears on the coder output after 10 clock cycles. That makes coder latency for the architecture which is 11 clock cycle. In opposite key extension procedure is executed by purely combinatorial logic. It was possible to do that under assumption that extended key calculation for consecutive round is faster than single round execution time. All that means that data throughput of the designed circuits relys only on time necessary for single round completion time. Maximum clock frequency can be calculated as an inverse of single round completion time. In our implementation number of coded 128-bit words per second is equal to the value of clock frequency. Every transformation performed in designed architecture is executed in parallel for all 16 bytes in 128-bit *State* block. Each transformation was implemented as combinatorial logic with a goal to minimize propagation time in mind. Thanks to Rijndael algorithm simplicity it was possible to achieve very effective implementation of all transformations necessary to complete coding and decoding algorithm in FPGA. The SubBytes operation was implemented as 16 identical LUTs representing S-box table according which substitution of bytes in *State* blocks is performed. The ShiftRows transformation is very easy to implement in hardware because it can be hardwired as change of bytes sequence in *State* block. Thanks to that it requires no additional resources except proper connection routing between successive operations. The MixColumns operation requires multiplication of *State* matrix and appropriate polynomial a(x) in GF(2^8) (1).

$$a(x) = 03 * x^3 + 01 * x^2 + 01 * x + 02 \quad (1)$$

As it can be noticed this operation requires multiplication by 'two' and 'three' only which is relatively simple in hardware (multiplication by two is simple shift operation and multiplication by three is shift and add operation). Additionally XOR operations are performed for MixColumns operation. Similarly AddRoundKey transformation in FPGA requires XOR logic only to proceed with two 128-bit data.

Thanks to above we achieve speed optimized hardware implementation of full cipher round.

The last element necessary to complete AES-128 implementation was key extension procedure implementation which performs operations presented in algorithm 1. Each key is calculated based on the key from the previous round which allows for utilization of combinatorial logic. The GenerateRoundKey function was implemented by utilization of S-box table and XOR operation. The same GenerateRoundKey function and the AddRoundKey operation were used as well for decoder implementation. Other transformations necessary for decoder implementation was designed similar to its coder counterparts. In InvSubBytes operation S-box table was repleaced by InvS-box table. The InvShiftRows transformation like ShiftRows transformation requires no additional logic. The most major difference between coder and decoder was in InvMixColumns. In this operation decoder requires different polynomial b(x) in GF(2^8) (2).

$$a(x) = 0B * x^3 + 0D * x^2 + 09 * x + 0E \quad (2)$$

In decoder case multiplication is necessary for each of four coefficient and these coefficients have not so straightforward hardware implementation as those in coder. So if we compare decoder to coder we achieve worse timing parameters for decoder. This results in single round execution time increase and lower data throughput as well as bigger number of logic resources necessary for decoder implementation.

IV. RESULTS

Efficiency of hardware implementation in reconfigurable logic [8] can be measured according four parameters:

TABLE II
RESOURCES UTILIZATION OF VIRTEX4

	Resource type		
	Slices	Slices Flip-Flops	4 input LUTs
Coding round	1209 (1,4%)	128 (0,1%)	2403 (1,4%)
Decoding round	1934 (2,2%)	128 (0,1%)	3584 (2%)
Key extension	3296 (2,2%)	0 (0%)	6400 (2,2%)

TABLE III
TIME BASED PARAMETERS OF AES MODULE

	Coding	Decoding
Round max path delay	6ns	7,5 ns
Max clock freq.	165MHz	130MHz
Latency	11Clk	11Clk
Throughput	21,2Gbit/s	16,6Gbits

- number of used resources,
- maximum clock frequency,
- latency measured in number of clocks,
- througput measured in bytes per second.

The AES-128 coding module was designed in VHDL hardware description language [9] and first simulated to check its behavioral model in ALDEC Active-HDL. The next step was hardware synthesis performed by firmware Xilinx software: XST. For the reference we present generted by by XST reports concerned to the major design blocks. Coding and decoding modules were synthetised separately. Ten fully pipelined rounds were implemented according to figure 1. To analyse number of resources necessary for single round implementation we removed key expansion procedure from the design. Pipeline architecture allows for estimation of single round execution time i.e. time delay before data will be latched in the next *State* register. The maximum path delay time directly makes single round execution time and its inverese is the maximum clock frequency of the design. Detailed resource utilisation of the Virtex4 series 4vlx200ff1513-10 are presented in table II. Timing parameters of the designed circuits are summarised in table III. To perform coding/decoding operation 11 clock cycles are necessary. Each round is completed in one clock cycle. Additional clock cycle is used to read data from outpur register. Key extension algorithm block works in parallel to main algorithm block when coding is considered. During decoding execution, algorithm block can start its execution after ca. 46 ns (according post-synthesis simulation time) when key extension procedure is completed. It is because it is necessary to know all coding keys to start decoding procedure.

V. CONCLUSION

For evaluation of coding/decoding speed designed hardware implementation can be compared to the existing ANSI-C software implementation of Rijndael algorithm [10]. Fore example there were achieved coding and decoding throughputs of 77 Mbit/s and 74 Mbit/s accordingly on Pentium II 450MHz. So hardware acceleration in reconfigurable hardware allows for major speed-up of Rijndael algorithm. Parameters achieved for FPGA hardware module of the algorithm allow for its utilization in very fast computer networks such as GigaBitEthernet where high performance data coding and decoding is necessary. Additionally data security should be emphasized when hardware implementation of cipher algorithms is considered because it is much more complicated to crack cipher key when it is concealed in hardware. Another advantage of FPGA is its reconfigurability which allows for software like flexibility of algorithms change together with custom hardware like performance. Also many different cipher algorithms can be executed in single semiconductor structure.

Scholarly work financed through research funds by Polish Ministry of Science and Higher Education as a research project in 2008.

REFERENCES

[1] J. Foti, "Status of the advanced encryption standard (AES) development effort," in *Proc. 21st NIST-NCSC National Information Systems Security Conference*, 1998, pp. 549–554. [Online]. Available: citeseer.ist.psu.edu/foti98status.html
[2] N. Ferguson and B. Schneier, *Practical Cryptography*. New York, NY, USA: John Wiley & Sons, Inc., 2003.
[3] J. Daemen and V. Rijmen, "Aes proposal: Rijndael." [Online]. Available: citeseer.ist.psu.edu/daemen98aes.html
[4] E. Biham, "How to decrypt or even substitute des-encrypted messages in 228 steps," *Inf. Process. Lett.*, vol. 84, no. 3, pp. 117–124, 2022.
[5] J. Daemen and V. Rijmen, "Aes proposal: Rijndael, proceedings of the first advanced encryption standard," NIST, Ventura, California, August 1998.
[6] K. Gaj and P. Chodowiec, "Hardware performance of the aes finalists survey and analysis of results." [Online]. Available: citeseer.ist.psu.edu/460345.html
[7] ——, "Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays," *Lecture Notes in Computer Science*, vol. 2020, pp. 84–??, 2001. [Online]. Available: citeseer.ist.psu.edu/article/gaj01fast.html
[8] ——, "Comparison of the hardware performance of the aes candidates using reconfigurable hardware," in *AES Candidate Conference*, 2000, pp. 40–54.
[9] P. J. Ashenden, *The Designer's Guide to VHDL*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.
[10] L. E. B. III, "Efficiency testing of ANSI c implementations of round 2 candidate algorithms for the advanced encryption standard," in *AES Candidate Conference*, 2000, pp. 136–148. [Online]. Available: citeseer.ist.psu.edu/bassham00efficiency.html