Neural, Parallel, and Scientific Computations 18 (2010) 47 – 58

FastCrypto: Parallel AES Pipelines Extension for General-Purpose Processors

Mostafa I. Soliman and Ghada Y. Abozaid
Computer & System Section, Electrical Engineering Department, Faculty of Engineering, South Valley University, Aswan, Egypt

Abstract
In cryptography, the advanced encryption standard (AES) is an encryption standard issued as FIPS by NIST as a successor to data encryption standard (DES) algorithm. The applications of the AES are wide including any sensitive data that requires cryptographic protection before communication or storage. This paper proposes extending general-purpose processors with crypto coprocessor based on decoupled architectures. The extended coprocessor splits an encryption/decryption instruction into memory (load/store) and computation (encryption/decryption) portions (pseudo instructions). Loading/storing and encrypting/decrypting data are performed in parallel and communicated through architectural queues. The computational unit includes parallel AES pipelines for fast encrypting/decrypting data. On four parallel AES pipelines, our results show a performance of 222 Giga bits per second.
Keywords - Parallel processing, AES pipeline, cryptography, decoupled architectures, FPGA implementation.

1. INTRODUCTION

The basis of many technological solutions to computer and communication security problems is cryptography. Cryptography (secret (crypto) writing (graphy) that needs to be decoded) is an essential tool underlying virtually all computers and networking protection. It has long been used for espionage and military. However, it is now commonly used in protecting information within many kinds of civilian systems. Cryptographic processing has been brought to the forefront of system design due to the need for secure communication and storage data.

The Data Encryption Standard (DES) [1] had served as an important cryptographic algorithm for over two decades. However, the growth of computing power during that time had compromised the security of that algorithm. There was considerable evidence that it was time to replace DES with a new standard. In October 2000, the National Institute of Standards and Technology (NIST) selected the Rijndael algorithm [2] as the new encryption standard to replace the current DES algorithm. The Advanced Encryption

Received January 11, 2010

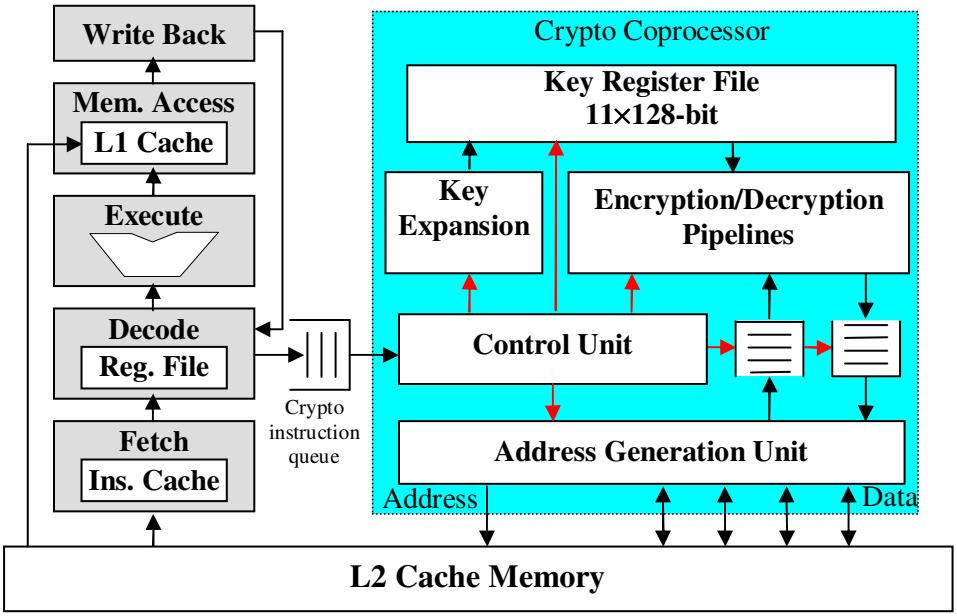
Standard (AES) specifies the Rijndael algorithm. The AES algorithm is a symmetric block cipher that can process (encrypt/decrypt) data blocks of 128-bit, using keys with lengths of 128, 192, and 256 bits. However, Rijndael was designed to handle additional block sizes and key lengths, which are not adopted in the AES.

According to NIST, AES is efficient, elegant, and secure. AES has the potential to maintain security well beyond twenty years due to the significantly larger key sizes than DES had [3]. Thus, AES was accepted as a FIPS (Federal Information Processing Standards) in November 2001 to protect electronic data [4]. The applications of the AES are wide; this standard can be used by agencies when an agency determines that sensitive information requires cryptographic protection. The AES algorithm may be implemented in software, firmware, hardware, or any combination thereof. The specific implementation may depend on several factors such as the application, environment, technology used, etc.

Since AES was accepted as a FIPS standard, there have been many different hardware implementations for ASIC (application specific integrated circuits) and FPGA (field-programmable gate array) to improve its performance. Some ASIC implementations are found in references [5-7]. These references mainly focus on area efficient implementation of the AES algorithm using Sbox (byte substitution) optimizations. Reference [7] (the first ASIC implementation of the Rijndael on silicon) presented an AES encryption chip architecture and discussed the design optimizations. References [8-10] presented different pipelined implementations of the AES algorithm as well as the design decisions and the area optimizations that lead to a low area and high throughput AES encryption processor. Some FPGA implementations of the AES algorithm are found in references [11-13]. References [14-15] presented another approach to improve the performance of AES on 32-bit processors using instruction set extensions.

This paper proposes another approach which is different from the above ones. It extends a general-purpose processor with an AES crypto coprocessor based on decoupled architectures, as shown in Figure 1. General-purpose processors can be simple in-order scalar or complex out-of-order superscalar processors. For simplicity, the well known 5-stage pipeline [16] is used as a scalar processor. Each instruction (scalar/crypto) is fetched from instruction cache and sent in-order to the decode stage. If the fetched instruction is scalar, it completes the remaining cycle of execution on the scalar pipeline stages (read operands, execute, memory access, and write-back result). However, crypto instructions are fetched from instruction cache and sent in-order to the extended crypto coprocessor during the decode stage for execution on the AES pipelines.

The extended coprocessor is based on decoupled architectures [17] to hid memory latency by letting the load/store and encryption/decryption operatrions to work in parallel. Note that crypto instructions do not block the issue stage of the scalar processor



eventhough the crypto coprocessor is busy. Crypto instructions are sent to the instruction queue freeing the issue stage to run ahead to execute scalar instructions latter in the instruction stream. In the proposed FastCrypto, latency is tolerated because the address unit is able to slip ahead of the computation unit and loads data that will be needed soon by the computation unit early in time. As Figure 1 shows, address generation unit and encyption/decryption piplines are communicated through architectural queues which are used to temporary keep the loaded/stored data from/to memory.

This paper is organized as follows. The specification of the AES algorithm is presented in Section 2. Section 3 describes microarchitecture of FastCrypto, which is based on decoupled architectures. Parallel pipelined implementation for accelerating AES is presented in Section 4. Finally, Section 5 concludes this paper.

2. THE SPECIFICATION OF THE AES ALGORITHM

The AES algorithm is a symmetric block cipher that can convert data to an unintelligible form (encryption) and convert the data back into its original form (decryption). The input and output for the AES algorithm each consist of sequences of 128-bit. Each 128-bit is called a block, which can be represented as 16-byte; in_0 through in_{15} for input and out_0 through out_{15} for output. Moreover, the cipher key for the AES algorithm is a sequence of 128, 192 or 256 bits. Internally, the AES algorithm's operations are performed on a two-dimensional (2-D) array of bytes called the *State* array. The *State* array consists of four rows of bytes, each containing *Nb* bytes, where *Nb*

is the block length divided by 32 (the word size). At the start of encrypting/decrypting a bock of data, the input (the 1-D array of bytes; in_0, in_1, … in_{15}) is copied into the *State* array (2-D array of bytes) according to the following scheme:

$$State_{row, col} = in_{row + 4*col} \text{ for } 0 \leq row < 4 \text{ and } 0 \leq col < Nb.$$

Then all encryption/decryption operations are performed on *State* array. The final value of the *State* array is copied to the output (1-D array of bytes; out_0, out_1, … out_{15}) as follows:

$$out_{row + 4*col} = State_{row, col} \text{ for } 0 \leq row < 4 \text{ and } 0 \leq col < Nb.$$

The four bytes in each column of the *State* array form 32-bit words, where the row number (*row*) provides an index for the four bytes within each word.

The AES algorithm consists of three distinct phases, as shown in Figure 2. In the first phase, an initial addition (XORing) is performed between the input data (plaintext) and the given key (cipher key). A number of standard rounds (N_r-1) are performed in the second phase, which represents the kernel of the algorithm and consumes most of the execution time. The number of these standard rounds depends on the key size; nine for 128-bits, eleven for 192-bits, or thirteen for 256-bits. Each standard round includes four fundamental algebraic function transformations on arrays of bytes.

1- byte substitution using a substitution table (Sbox),

2- shifting rows of the *State* array by different offsets (ShiftRows),

3- mixing the data within each column of the *State* array (MixColumns), and

4- adding a round key to the *State* array (KeyAddition).

The third phase of the AES algorithm represents the final round of the algorithm, which is similar to the standard round, except that it does not have MixColumns operation. In this paper, 128-bit key length AES algorithm is implemented, which has 10 rounds (nine

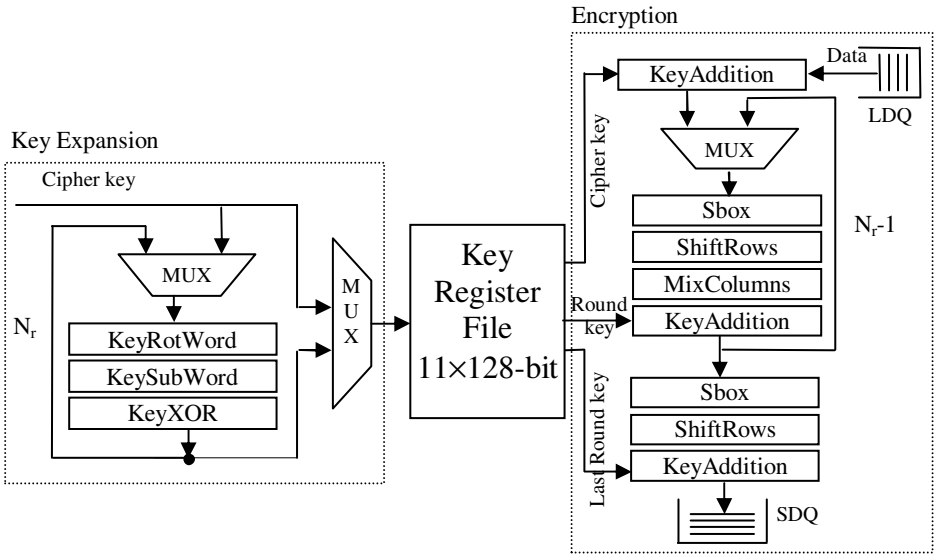


Figure 2: Functions needed for AES algorithm (rolled version)

standard rounds plus a final round) executed on parallel AES pipelines (see Figure 1). Moreover, the initial key is expanded on the key expantion unit of the crypto coprocessor to generate the round keys, each of size equals 128-bit. Each of the ten rounds of the algorithm receives a new round key from the key expantion unit. Decryption is performed by applying of the inverse transformations of the round functions (see [4] for more detail).

Sbox is a non-linear substitution table used in several byte substitution transformations and in the key expansion routine to perform a one-to-one substitution of a byte value. It is the primary source of nonlinearity in the AES algorithm. It takes each byte in the 128-bit *State* and computes its multiplicative inverse in GF(2^8), followed by a single stage of systematic bit mixing. The multiplicative inverse in GF(2^8) is computed using the extended Euclidean algorithm, which is essentially the Euclidean algorithm for integers, applied to polynomials in GF(2^8). The computational complexity of the Euclidean algorithm and the non-constant number of iterations required to compute the multiplicative inverse leads to use a lookup table for computing the multiplicative inverses. Thus, the lookup table technique is the optimum software/hardware approach for implementing Sbox.

ShiftRows transformation function processes the *State* array by cyclically shifting the last three rows over different numbers of offsets. In other words, for rows number one, two, and three, the offsets used are one, two, and three bytes respectively, however, the first row is not shifted.

MixColumns transformation function takes the columns of the *State* array and mixes their data (independently of one another) to produce new columns. The MixColumns transformation for column *c* can be written as follows:

$$State`_{0,c} = (02H \bullet State_{0,c}) \oplus (03H \bullet State_{1,c}) \oplus (01H \bullet State_{2,c}) \oplus (01H \bullet State_{3,c})$$

$$State`_{1,c} = (01H \bullet State_{0,c}) \oplus (02H \bullet State_{1,c}) \oplus (03H \bullet State_{2,c}) \oplus (01H \bullet State_{3,c})$$

$$State`_{2,c} = (01H \bullet State_{0,c}) \oplus (01H \bullet State_{1,c}) \oplus (02H \bullet State_{2,c}) \oplus (03H \bullet State_{3,c})$$

$$State`_{3,c} = (03H \bullet State_{0,c}) \oplus (01H \bullet State_{1,c}) \oplus (01H \bullet State_{2,c}) \oplus (02H \bullet State_{3,c})$$

The above polynomials include three types of multiplication ($01H \bullet State_{0,c}$, $02H \bullet State_{0,c}$, and $03H \bullet State_{0,c}$). Obviously, any byte (binary polynomial $b_7b_6b_5b_4b_3b_2b_1b_0$) times 01H results in the same value ($b_7b_6b_5b_4b_3b_2b_1b_0$). Multiplying a byte by 02H can be implemented at the byte level as a left shift and a subsequent conditional bitwise XOR with 1bH. After shifting by one bit, if $b_7 = 0$, the result ($b_6b_5b_4b_3b_2b_1b_00$) is already in reduced form. Otherwise, the reduction is accomplished by XORing the shifted byte with the polynomial 1bH. Multiply by 03H can be performed by XORing the results from multiplying by 01H and 02H.

Table 1: The content of the Rcon[round]

Round	1	2	3	4	5	6	7	8	9	10
Rcon[]	01	02	04	08	10	20	40	80	1b	36
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00

The addition in the AES algorithm is performed with parallel XOR operations. KeyAddition can be implemented by performing a bit-wise XORing between a column in the *State* array and corresponding column of the round key.

The key expansion round starts with KeyRotWord function, which cyclically shifts (rotates) the last four bytes of the round key by one byte (see Figure 2). After cyclic shifting, KeySubWord function is performed by applying Sbox on this last four bytes of the round key. The resulted column is XORed in KeyXOR stage with the round constant (Rcon[round]), as shown in Table 1.

3. THE MICROARCHITECTURE OF FASTCRYPTO

FastCrypto is a general-purpose processor extended with crypto coprocessor for fast encrypting/decrypting data, as Figure 1 shows. The extended part is based on decoupled architectures to execute encryption/decryption instructions with the following format, which is similar to MIPS ISA [18].

6-bit	5-bit	5-bit	10-bit	6-bit
010010	rs	rd	length	Enc/Dec

Any crypto instruction has a prefix of 010010 to tell the decode stage of the scalar part that the execution of this instruction is on the crypto coprocessor. Crypto instructions are fetched, decoded, and then dispatched in-order by the scalar core to a queue called crypto instruction queue (CIQ), as shown in Figure 3. Note that *rs* and *rd* are scalar registers holding the starting source and destination addresses for encryption and decryption and $1 \leq length < 1024$ is the number of blocks (128-bit) needed to be encrypted/decrypted. During the decode stage of the scalar unit, the contents of *rs* and *rd* are read from the scalar register file and pushed into CIQ queue. Thus, 80-bit (6-bit Enc/Dec, 10-bit *length*, 32-bit the content of *rs*, and 32-bit the content of *rd*) are sent from scalar unit to the extended crypto coprocessor though CIQ queue by the end of decode stage of the scalar pipeline.

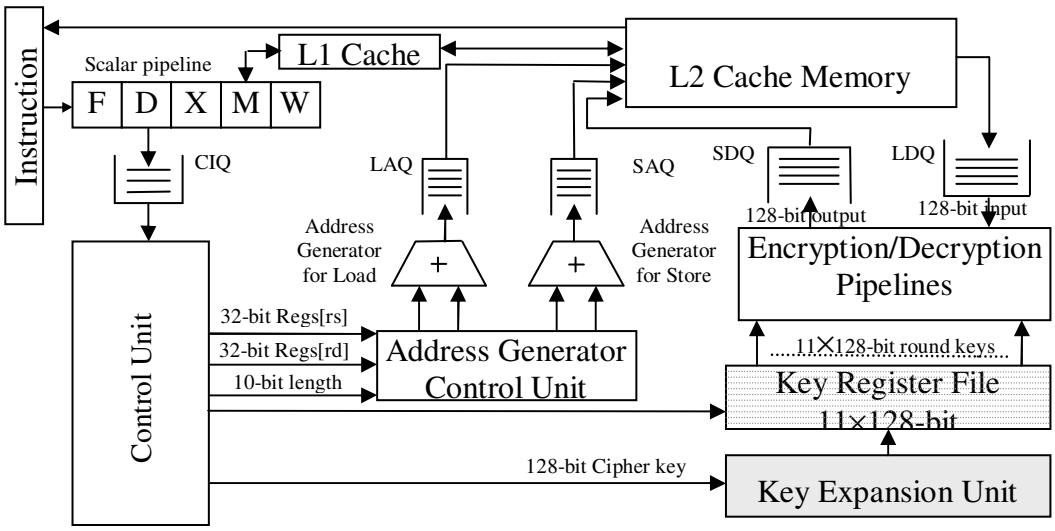


Figure 3: Organization of Decoupled FastCrypto

The control unit of the crypto coprocessor fetches crypto instructions in-order from the head of CIQ (see Figure 3). Then the control unit sends *rs*, *rd*, and *length* to the address generation control unit for generating the addresses to load data from memory and to store the encrypted/decrypted data back into the memory. As Figure 3 shows, the address generation unit consists of two address generators and two queues working under control of address generator control unit. Load address generator works in parallel with store address generator, where the generated addresses are stored into LAQ (load address queue) and SAQ (store address queue).

For the key expansion process, the following instruction format is used for sending 128-bit key from the scalar processor to the crypto coprocessor through CIQ queue.

6-bit	5-bit	5-bit	10-bit	6-bit
010010	rs	rd	---	Key1/ Key2

The 128-bit key are sent to the crypto coprocessor in two steps because during the decode stage only two register (2×32-bit) can be read from the scalar register file. Thus, the main control unit of the crypto coprocessor stores 64-bit comes with Key1 until the other 64-bit comes with Key2. After 128-bit key is sent from the scalar unit, the main control unit passes it to the key expansion unit. As Figure 2 shows, the key expansion unit iterates the functions KeyRotWord, KeySubWord, and KeyXOR ten times (N_r = 10) and stores the generated keys (round keys) into key register file. Thus, the size of the key register file is 11×128-bit (the original cipher key plus 10 generated round keys).

4. PARALLEL IMPLEMENTATION OF AES PIPELINE

Various architectures exist to realize the AES encryption/decryption algorithm. Among them, rolling and unrolling are the two basic architectures. The rolled AES pipeline uses a feedback structure (see Figure 2) where the data is iteratively transformed by round functions. This approach occupies small area, but achieves low throughput. Some existing rolled implementations are presented in [19-20]. In the unrolled AES pipeline, the round functions are pipelined furthermore. Thus pipeline registers should be inserted between rounds allowing simultaneous operations of all 11 round stages. Pipelined implementation of AES achieves higher throughput, however, requires larger area. Some Existing unrolling implementations are appeared in [21-24].

To further improve the performance (throughput) of unrolling implementations, each round would be pipelined. Thus pipelining technique can be applied both for inside (inner) and around (outer) each round. Our implementation of the outer pipeline (one pipeline stage per round) achieves a throughput of 45 Gbps at frequency 360 MHz on Xilinx Virtex V FPGA and Xilinx ISE 10.1 synthesis tool. Since the look-up tables of Sbox are the main critical path in the pipeline design, the implementation of inner pipeline with two, three or four pipeline stages achieves 70 Gbps at maximum frequency 557 MHz. This results in two stages per round is the best choice for implementation. This choice gives maximum throughput-area tread-off.

This section proposes the use of parallel AES pipelines to further improve the throughput of encryption/decryption pipelines unit shown in Figure 1 and 3. Figure 4 shows the proposed implementation, which is based on using parallel AES pipelines architecture. Each round has two pipeline stages, which achieves a throughput of 70 Gbps. The first stage performs Sbox and ShiftRows transformations and the second stage performs MixColumns and keyAddition transformations. Four parallel AES pipelines for each encryption and decryption, key expansion unit, and key register file consume up to 15840 slices. In Xilinx terms, two logic cells are grouped to form a slice, where each logic cell contains a four-input look-up table and a D flip-flop. The maximum frequency for this architecture on Xilinx Virtex V FPGA is 444 MHz, which provides a throughput of 222 Gbps. As Figure 4 shows, the input data for these parallel pipelines is loaded from four 128-bit queues called LDQ. Each queue is implemented as four words of 128-bit wide, which consumes 144 slices. A single data bus feeds the four queues of LDQ in round-robin fashion. In addition, four 128-bit output ciphertext/deciphertext are stored into four 128-bit queues called SDQ. The implementation of SDQ is the same as LDQ. The ICQ is implemented as four 80-bit word with area consumption equals to 72 slices.

To keep a balance between data encryption paths and key expansion unit, the key expansion round implemented as two stages per round. The unrolled key expansion,

which consumes 864 slices calculates the round keys at first 20 cycles and remains constant during the execution of encryption/decryption instructions. Thus, the total area would be minimized using the offline technique and rolled key expansion unit. The round keys in FastCrypto are generated using rolled key expansion unit, which consumes 144 slices. The key expansion unit calculates the round keys offline and stores them in the key register file. As Figure 4 shows, each data encryption path needs eleven 128-bit round keys. Instead of adding key expansion unit for each data encryption path, our implementation is based on the use of a common key register file for all parallel pipelines. By broadcasting the round keys, the key register file feeds parallel AES pipelines at once.

5. CONCLUSION

Encryption has long been used by militaries and governments to facilitate secret communication. However, encryption is now commonly used in protecting information within many kinds of civilian systems. This paper proposes extending general-purpose processors with crypto coprocessor for fast encrypting/decrypting data. The extended unit is based on decoupled architectures, which splits each crypto instruction into memory (load/store) and computation (encryption/decryption) portions (pseudo instructions).

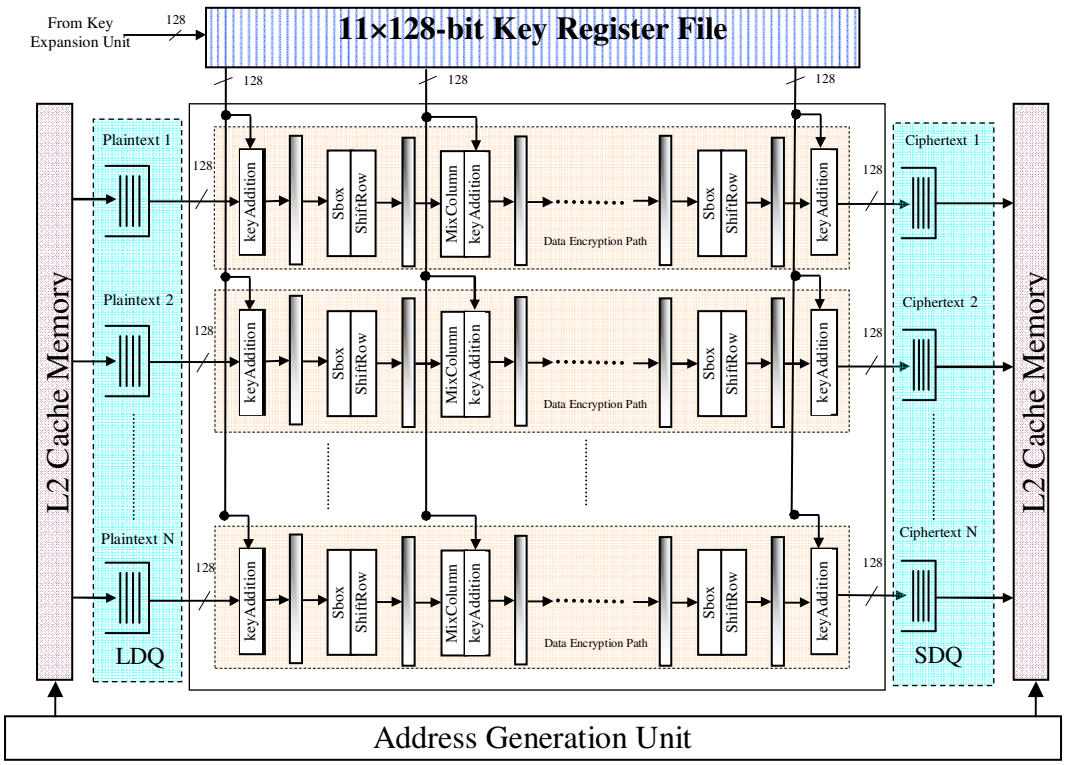


Figure 4: Parallel pipeline implementation of AES algorithm

Loading/storing and encrypting/decrypting data are performed in parallel and communicated through architectural queues. The computational unit includes parallel AES pipelines for fast encrypting/decrypting data. Each pipeline is based on unrolling architectures. In addition, each round has two pipeline stages. On four parallel AES pipelines, our results show a performance of 222 Gbps.

REFERENCES

[1] Data Encryption Standard (DES), Available at http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, October 1999.

[2] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," Available at http://csrc.nist.gov/archive/aes/index.html, September 1999.

[3] http://www.nist.gov/public_affairs/releases/aesq&a.htm.

[4] Advanced Encryption Standard (AES), US National Institute of Standards and Technology, FIPS 197, Available at, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.

[5] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," *Proc. 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT2001)*, Australia, LNCS2248, ISBN 3-540-42987-5, pp. 239-254, 2001.

[6] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC Implementation of the AES Sboxes," *Proc. The Cryptographer's Track at the RSA Conference on Topics in Cryptology (CT-RSA2002)*, San Jose, CA, USA, LNCS2271, pp. 29-52, 2002.

[7] I. Verbauwhede, P. Schaumont, and H. Kuo, "Design and Performance Testing of a 2.29 Gb/s Rijndael Processor," *IEEE Journal of Solid-State Circuits*, Vol. 38, No.3, pp. 569-572, 2003.

[8] A. Hodjat and I. Verbauwhede, "Speed-Area Trade-off for 10 to 100 Gbits/s Throughput AES Processor," *Proc. 37th Asilomar Conference on Signals, Systems & Computers, Monterey*, USA, Vol. 2 , pp. 2147-2150, 2003.

[9] A. Hodjat and I. verbauwhede, "A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA," *Proc. 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004)*, ISBN 0-7695-2230-0, pp. 308-309, 2004.

[10] A. Hodjat and I. Verbauwhede, "Area-Throughput Trade-offs for Fully Pipelined 30 to 70 Gbits/s AES Processors," *IEEE Transactions on Computers*, Vol. 55, No. 4, pp. 366-372, 2006.

[11] M. McLoone and J. McCanny, "High Performance Single Chip FPGA Rijndael Algorithm Implementations," *Proc. 3rd International Workshop on Cryptographic Hardware and Embedded Systems*, Paris, France, LNCS2162, ISBN 3-540-42521-7, pp. 65-76, 2001.

[12] 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*, Vol. 9, No. 4, pp. 545-557, 2001.

[13] K. Gaj and P. Chodowiec, "Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays," *Proc. 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA (CT-RSA2001)*, San Francisco, CA, USA, LNCS2020, ISBN 3-540-41898-9, pp. 84-99, 2001.

[14] S. Tillich and J. Großschädl, "Instruction Set Extensions for Efficient AES Implementation on 32-bit Processors," *Proc. 8th International Workshop in Cryptographic Hardware and Embedded Systems*, Yokohama, Japan, LNCS4249, ISBN 978-3-540-46559-1, pp. 270-284, 2006.

[15] S. Tillich and J. Großschädl, "Accelerating AES Using Instruction Set Extensions for Elliptic Curve Cryptography. *Proc. International Conference on Computational Science and Its Applications (ICCSA 2005)*, LNCS3483, ISBN: 978-3-540-25863-6, Singapore, pp. 665-675, 2005.

[16] D. Patterson and J. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 4th Edition, Elsevier Publisher, 2009.

[17] J. Smith, "Decoupled Access/Execute Computer Architectures," *ACM Transactions on Computer Systems*, Vol. 2, No. 4, pp. 289-308, 1984.

[18] MIPS32 Architecture For Programmers, *Volume I: Introduction to the MIPS32 Architecture*, Available at http://www.mips.com/products/product-materials/processor/mips-architecture/, 2008.

[19] J. Zambreno, D. Nguyen, and A. Choudhary, "Exploring Area/Delay Tradeoffs in an AES FPGA Implementation," *Proc. 14th International Conference on Field Programmable Logic and Application*, Leuven, Belgium, LNCS3203, ISBN 3-540-22989-2, pp. 575-585, 2004.

[20] HELION Technology Limited, *High Performance AES (Rijndael) Cores for Altera FPGA*, Available at http://www.heliontech.com/core2.htm .

[21] G. Saggese, A. Mazzeo, N. Mazzocca, and A. Strollo, "An FPGA-Based Performance Analysis of the Unrolling, Tiling, And Pipelining of the AES Algorithm," *Proc. 13th International Conference on Field-Programmable Logic*

And Applications (FPL2003), Lisbon, Portugal, LNCS 2778, ISBN 3-540-40822-3, pp. 292-302, 2003.

[22] F. Standaert, G. Rouvroy, J. Quisquater, and J. Legat, "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," *Proc. Cryptographic Hardware and Embedded Systems (CHES2003)*, Cologne, Germany, LNCS2779, ISBN 3-540-40833-9, pp. 334-350, 2003.

[23] F. Charot, E. Yahya, and C. Wagner, "Efficient Modular-Pipelined AES Implementation in Counter Mode on ALTERA FPGA," *Proc. 13th International Conference on Field-Programmable Logic And Applications (FPL2003)*, Lisbon, Portugal, LNCS2778, ISBN 3-540-40822-3, pp. 282-291, 2003.

[24] N. Pramstaller and J. Wolkerstorfer, "A Universal and Efficient AES Co-Processor for Field Programmable Logic Arrays," *Proc. 14th International Conference on Field-Programmable Logic And Applications (FPL2004)*, Leuven, Belgium, LNCS3203, ISBN 3-540-22989-2, pp. 565-574, 2004.