Minimal Instruction Set FPGA AES Processor using Handel - C

J. H. Kong, L.-M. Ang, K. P. Seng and Achonu Oluwole Adejo.
University Of Nottingham Malaysia Campus
Jalan Broga, Semenyih, 43500 Selangor, Malaysia.
E-mail: {keyx9kjh, kezklma, kezkps, kecx8aoa}@nottingham.edu.my

***Abstract*—This paper presents an FPGA implementation of the Advanced Encryption Standard (AES), using a Minimal Instruction Set Computer (MISC) architecture. The MISC's architecture is simple and reconfigurable to execute fundamental instructions with just simple hardware logic components. Due to the MISC's simplicity, it can be further extended to data encryption systems for certain applications like wireless sensor networks and other low complexity systems which may have severely constrained physical memory requirements. With the availability of the FPGA technology, aids practical implementation of the data encryption purpose processor.**

***Keywords*—Minimal Instruction Set Computer; AES**

I. INTRODUCTION

The original motivation for the OISC (One Instruction Set Computer) was meant for educational purposes. It is the simplicity which gives greater insight into the CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer) architectures. It has been a useful educational tool for teaching the basics of computer organization concepts (such as the Control Unit Design and Micro-programming) to students, and as an objective and independent way of comparing and contrasting the existing instruction sets. [1], [2]

An instruction set constitutes the language that describes a computer's functionality and it is a function of the computer's organization. The one instruction set computer (OISC), also called as the Ultimate Reduced Instruction Set Computer (URISC), is a single-instruction universal computing machine. In OISC, the instruction set consists of one instruction, and then by the orthogonality of the instruction along with composition, a complete set of operations is synthesized. It is the most flexible computer architecture with a simplified hardware level functionality implemented around a single instruction, without any instruction decoder circuitry.

With OISC as the simplest form of a one instruction set computer architecture, several instructions can be developed to design a new purpose computer. Micro-operations are be redefined for the sole purpose computer and it can be further expanded into an architecture that processes not only a single instruction. A Minimal Instruction Set Computer (MISC) is a computer similar to an OISC with the difference of, the MISC incorporates not only one instruction but several simple instructions. With a sufficient amount of simple instructions, a new purpose-designed computer can be built and the possibility of application is vast.

This paper seeks to further present an application of the MISC, using FPGAs in data encryption specifically for low complexity processing e.g wireless sensor networks, embedded systems and mobile communication. Security is a crucial part of the design and function of data transmission and storage in these applications which usually have restrictions like small physical and memory sizes, low energy usage and limited communication bandwidth. [8] FPGAs have been useful tools in the practical implementation of encryption algorithms especially the advanced encryption standard (AES) as can be seen in [9], [10].

II. THE REVIEW OF OISC

The original motivation for the OISC was meant for educational purposes. It is the simplicity which gives greater insight into the CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer) architectures. It has been a useful educational tool for teaching the basics of computer organization concepts (such as the Control Unit Design and Micro-programming) to students, and as an objective and independent way of comparing and contrasting the existing instruction sets. [1], [2]

The most important theoretical models of the OISC is, subtract and branch if negative (SBN) architecture. This processor was originally proposed by van der Poel, and the format of the instruction is shown in Fig. 1.

The 1st operand is subtracted from the 2nd operand, and if the result is negative, the program execution proceeds to the next address. In [2], an implementation of the OISC using subtract and branch if negative, and employing a 64k X 16 memory was presented. In this paper, the AES MISC Architecture is designed based on this OISC SBN Architecture.

SBN	Operand Address 1 (a)	Operand Address 2 (b)	Target Address (c)

SBN a,b,c ;

Mem[b] = Mem[b] – Mem[a];

if (Mem[b] < 0), goto c;

Figure 1. The simplified SBN Instruction format and pseudo code

III. THE REVIEW OF AES

The AES is a symmetric block cipher that can process data blocks of 128 bits, using cipher keys that can be of lengths 128, 192 or 256 bits. The basic unit for processing in the AES Algorithm is a byte and the input, output and cipher key bit sequences are processed as array of bytes. The state is a two-dimensional array of bytes where the AES Algorithm's operations are performed internally. It consists of four rows of bytes containing Nb bytes, where Nb is the block length divided by 32. The input is copied into the state array at the start of the Cipher and Inverse Cipher and the algorithm is executed over a number of rounds determined by the key size. The round function is composed of four different byte-oriented transformations: the Sub Bytes, Shift Rows, Mix Columns and the Add Round Key. The flowchart showing the round function and the transformations in each round is shown below: [10], [11].

For a complete AES encryption process, a total of 10 rounds (Nr = 10) of substitution and permutation are required. All four byte-oriented transformation together with a 128 cipher key are applied onto the input plaintext. And for each round, a unique key will be generated by the Key Expansion Algorithm. In every round, the Round Key will be added to the plaintext and also, going through rounds of S - Box Substitution, Mix Column, and Shift Row [11].

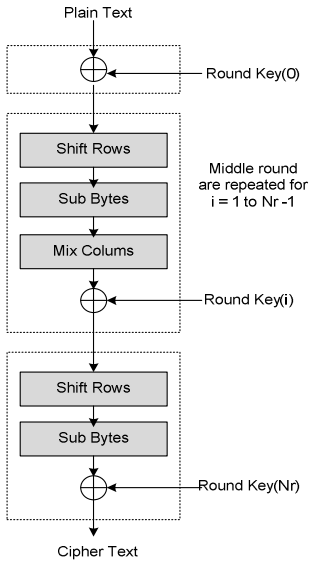


Figure 2. The byte-oriented transformation process in AES (Encryption)

IV. THE MISC AES PROCESSOR

A. Instruction Set

In order to perform AES computations on to the plaintext, byte-oriented methods are adapted from the AES encryption method. To perform tasks such as, Sub Bytes and Mix Columns, a new series of instructions have to be developed, in order to perform these operations.

The MISC Instructions in the Figure 3 above are differentiated using the two MSB of each of the instructions. Based on the operation required for each byte-oriented transformation in the AES algorithm, the four instruction sets used to perform separate operations is showed in the table below:

The AES Round Functions will be described in section 4.4 in more detail. These instruction sets are the fundamental commands to the processor to operate accordingly to the AES Algorithm.

TABLE I.
THE MISC INSTRUCTION FORMATS

Operation	Op Code (2 MSB)	Instruction Format
SBN	0 (00)	(00 @ Mem_A), (Mem_B), Target
XOR	1 (01)	(01 @ Mem_A), (Mem_B), Target
XTime	2 (10)	(10 @ 00000000), (Mem_B), Target
Sub Bytes	3 (11)	(11 @ 00000000), (Mem_B), Target

SBN (Subtract and Branch If Negative)
Mem_B = Mem_B – Mem_A
If Mem_B < 0 Goto (PC + C)
Else Goto (PC + 1)

XOR
Mem_B = Mem_B XOR Mem_A

xTime
Mem_B = xTime(Mem_B)

Sub Bytes
Mem_B = Sub_Bytes(Mem_B)

Figure 3. The MISC Instruction Sets

B. Architecture

In the MISC AES architecture, 7 registers, 4 multiplexers (MUX), an Adder, an XOR Block, and xTime Block and a Sub-Bytes Block is used. At the top of the architecture design, the PC register stores the Program Counter (PC) value, which holds the subsequent program code in the memory to be read. The R register will store the first data read (Mem_A) from the memory. Memory Address Register (MAR) will stores the address of the memory, providing a reference to which memory location to be written or read. The Memory Data Register (MDR) is used to store the output computed by the Core ALU. The computation results can be either the results of the Adder, XOR, xTime or the Sub Bytes. The result will then be written back to the memory, replacing the value of the second data read (Mem_B). The Z and N registers indicate the arithmetic result performed by the Adder, is either a zero or a negative respectively. The Operation Code Register (OP) stores the Op Code of the current operation to be performed and it is used to select the output ALU.

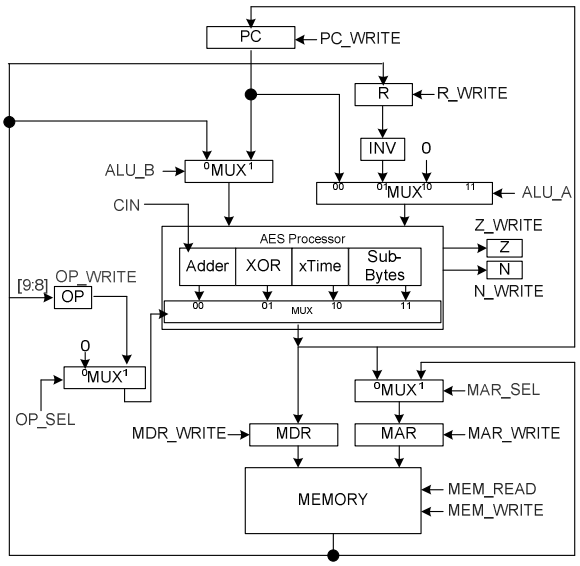


Figure 4. The MISC AES Architecture

C. Memory Map

This architecture includes a 1024 x 10-bit Memory. The memory used in the MISC architecture is based on the Von Neumann Architecture. The total available memory is 1024 x 8-bit (512 bytes), which accommodates both the data and program codes. The data section lies at the address location of 0 to 127, whereas the program section takes the location of 128 to 1024. Fig. 5 shows the implemented memory in the MISC architecture. Each line of the Program Code section contains the Target Memory Addresses of the Data and the Jump Address for the SBN instructions. The memory used in this architecture can only be read or written at a particular clock cycle.

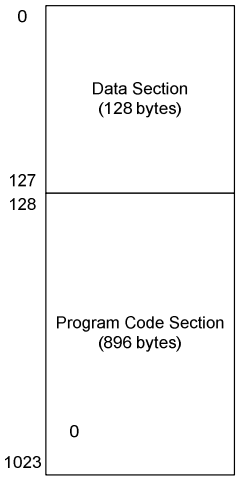


Figure 5. The Memory allocation for Data and Program Code sections

D. MISC ALU

The AES ALU consists of the 4 main logic circuits: the Adder, XOR, xTime, and the Sub Bytes.

From Fig. 6, the AES Core Processor consists of 4 main circuit blocks. The Adder circuit takes in 2 inputs and adds them together. This Adder is essential in performing the adding operation of the SBN instruction. In the XOR Block, the circuit takes in 1 input and performs an XOR operation on to the data.

The xTime Block is a part of the Mix-Column Transformation. From the Fig. 7, the Mix Column Transformation is a process involving several XOR processes and xTime processes. This transformation involves only 4 bytes every it is used.

The Sub Bytes Transformation is implemented using a non-linear substitution cipher. The input byte is mapped to the multiplicative inverse in the Galois field GF (2^8). And lastly, an Affine Transformation is applied.

The input byte is transferred to the various sub blocks. Their combinational logic components are showed in the series of figures below.

The Sub Block, Multiplication in GF (2^4), has 3 operations of Multiplication in GF (2^2), and the circuit is showed in Fig. 13.

In the final process, in the input bytes are mapped to their Multiplicative Inverse in the Galois field GF (2^8). And the Sub Bytes transformation is complete.

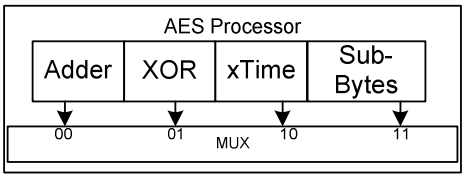


Figure 6. The AES Processor ALU

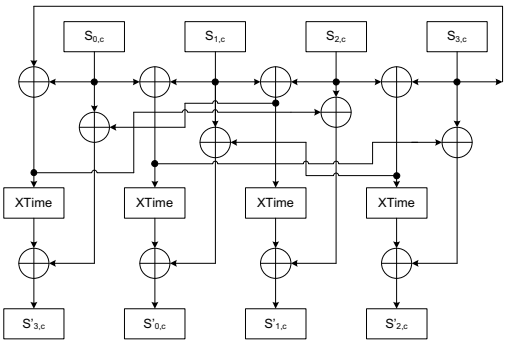


Figure 7. The Mix Column Transformation Process

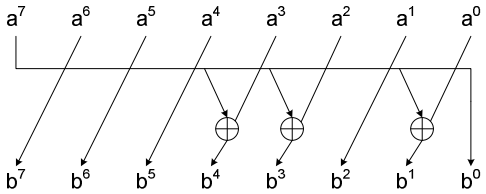


Figure 8. The xTime Circuit

\

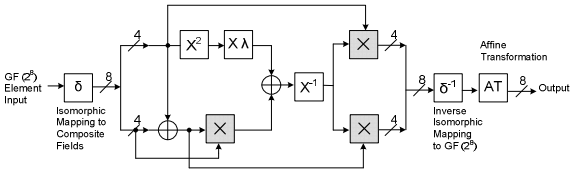
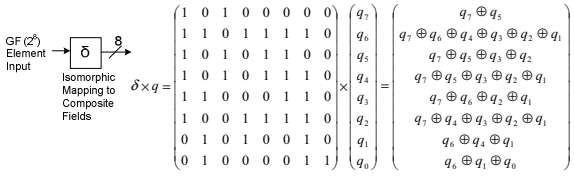


Figure 9. The combinational circuit for Sub Bytes (S-Box)



$$\begin{pmatrix} 1&0&1&0&0&0&0&0\\ 1&1&0&1&1&1&1&0\\ 1&0&1&0&1&1&0&0\\ 1&0&1&0&1&1&1&0\\ 1&1&0&0&0&1&1&0\\ 1&0&0&1&1&1&1&0\\ 0&1&0&1&0&0&1&0\\ 0&1&0&0&0&0&1&1 \end{pmatrix} \times \begin{pmatrix} q_7\\ q_6\\ q_5\\ q_4\\ q_3\\ q_2\\ q_1\\ q_0 \end{pmatrix} = \begin{pmatrix} q_7 \oplus q_5\\ q_7 \oplus q_6 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1\\ q_7 \oplus q_5 \oplus q_3 \oplus q_2\\ q_7 \oplus q_5 \oplus q_3 \oplus q_2 \oplus q_1\\ q_7 \oplus q_6 \oplus q_2 \oplus q_1\\ q_7 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1\\ q_6 \oplus q_4 \oplus q_1\\ q_6 \oplus q_1 \oplus q_0 \end{pmatrix}$$

Figure 10. The Isomorphic Mapping to Composite Fields

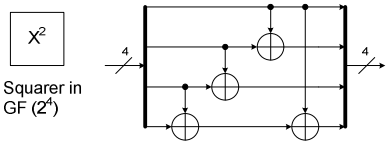


Figure 11. The Square in GF (2^4)

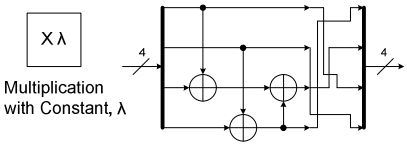


Figure 12. The Multiplication with Constant, λ

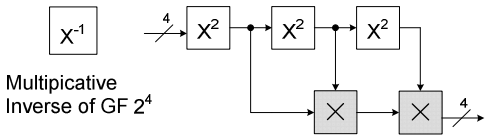


Figure 13. The Multiplicative Inverse of GF 2^4

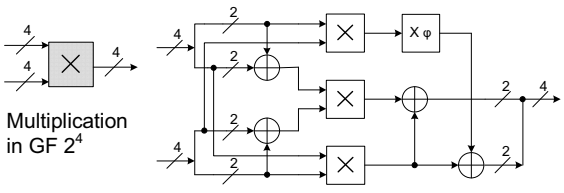


Figure 14. The Multiplication in GF 2^4

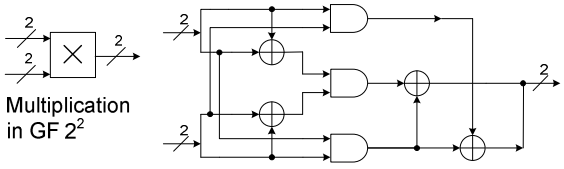


Figure 15. The Multiplication in GF 2^2

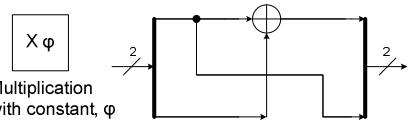
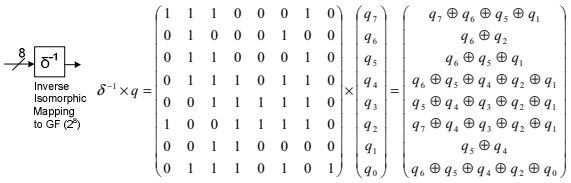
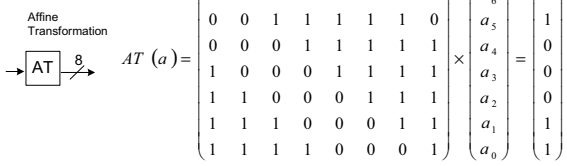


Figure 16. The Multiplication with Constant, φ



$$\begin{pmatrix} 1&1&1&0&0&0&1&0\\ 0&1&0&0&0&1&0&0\\ 0&1&1&0&0&0&1&0\\ 0&1&1&1&0&1&1&0\\ 0&0&1&1&1&1&1&0\\ 1&0&0&1&1&1&1&0\\ 0&0&1&1&0&0&0&0\\ 0&1&1&1&0&1&0&1 \end{pmatrix} \times \begin{pmatrix} q_7\\ q_6\\ q_5\\ q_4\\ q_3\\ q_2\\ q_1\\ q_0 \end{pmatrix} = \begin{pmatrix} q_7 \oplus q_6 \oplus q_5 \oplus q_1\\ q_6 \oplus q_2\\ q_6 \oplus q_5 \oplus q_1\\ q_6 \oplus q_5 \oplus q_4 \oplus q_2 \oplus q_1\\ q_5 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1\\ q_7 \oplus q_4 \oplus q_3 \oplus q_2 \oplus q_1\\ q_5 \oplus q_4\\ q_6 \oplus q_5 \oplus q_4 \oplus q_2 \oplus q_0 \end{pmatrix}$$

Figure 17. The Inverse Isomorphic Mapping to GF (2^8)



$$\begin{pmatrix} 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1\\ 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} a_7\\ a_6\\ a_5\\ a_4\\ a_3\\ a_2\\ a_1\\ a_0 \end{pmatrix} = \begin{pmatrix} 0\\ 1\\ 1\\ 0\\ 0\\ 0\\ 1\\ 1 \end{pmatrix}$$

Figure 18. The Affine Transformation

E. Control Circuit

The Control Circuit is driven by a 4-bit counter (C3C2C1C0). At each clock cycle, the control signals for particular control inputs are different. They are required to control the registers and store memory at any particular clock cycle.

During clock cycle 0, the value of the program counter (PC), is loaded into the MAR and at the same clock cycle, the Z register will be set accordingly by the ALU Output to determine whether the PC has restart at 0x00. In the following clock cycles, the data is read from the memory location, addressed by the MAR which stored the value of PC. Subsequently, the read data is written back to MAR, storing the address of the data to be used for computation. These processes are repeated for a second data. The PC value will be increased by 1 after each data loading operation is done.

At clock cycle 1, the address for Mem_A is stored in the MAR, at the same time, the Op Code for the instruction is written to the OP Register. At clock cycle 2, the value of Mem_A is read and stored into the R register.

At the clock cycle 6, the value of Mem_B is read and sent to the ALU for computation. The Adder and the other hardware blocks will perform their individual operations from the two given inputs (Mem_A and Mem_B). At that particular clock cycle, depending on the value of the OP Register, the desired output will be chosen via a ALU MUX, and the output is sent to the MDR Register for storage.

With the arithmetic operations performed, clock cycle 7 will load the jump address from memory. Then the jump address will be added to the PC value at the same clock cycle. The jump address value will only be added to the

PC value, provided that the OP value is corresponding to the SBN instruction and a negative result is found at the output.

The Boolean expressions in Fig. 19 are the control signals to each bit of the 4-bit counter. If a branching off occurs, the N register would have a value of 0. So, the PC register would just would take in take in the value of the jump address and increase by 1. Then the following instruction in the written program code will be performed.

$$ALU\_B = \overline{C_3}\overline{C_2} + \overline{C_2}\overline{C_1}\overline{C_0} + \overline{C_3}\overline{C_0}$$
$$ALU\_A1 = \overline{C_3}\overline{C_2} + \overline{C_2}\overline{C_1}\overline{C_0} + \overline{C_3}\overline{C_0}$$
$$ALU\_A0 = \overline{C_3}C_2\overline{C_1}C_0$$
$$CIN = \overline{C_3}C_2\overline{C_1} + \overline{C_3}C_1C_0 + C_3\overline{C_2}\overline{C_1}\overline{C_0}$$
$$MAR\_SEL = \overline{C_3}C_2\overline{C_1} + \overline{C_3}C_1C_0$$
$$PC\_Write = \overline{C_3}C_2C_1C_0N + \overline{C_3}C_2\overline{C_1}C_0 + \overline{C_3}C_2\overline{C_1}\overline{C_0} + C_3\overline{C_2}C_1\overline{C_0}$$
$$R\_Write = \overline{C_3}\overline{C_2}C_1\overline{C_0}$$
$$Z\_Write = \overline{C_3}\overline{C_2}C_1\overline{C_0}$$
$$N\_Write = \overline{C_3}C_2\overline{C_1}C_0$$
$$MAR\_Write = \overline{C_3}\overline{C_2}C_0 + \overline{C_3}C_2\overline{C_0} + \overline{C_3}C_1\overline{C_0}$$
$$MDR\_Write = \overline{C_3}C_2\overline{C_1}C_0$$
$$Mem\_Read = \overline{C_3}C_2\overline{C_1} + \overline{C_3}C_2C_0 + \overline{C_3}\overline{C_1}C_0 + \overline{C_3}C_2C_1\overline{C_0}$$
$$Mem\_Write = \overline{C_3}C_2C_1\overline{C_0}$$
$$Op\_Write = \overline{C_3}\overline{C_2}\overline{C_1}C_0$$
$$Op\_SEL = \overline{C_3}C_2\overline{C_1}C_0$$

Figure 19. Control Signals

V. RESULTS

The result simulated showed a successful encryption process with a cipher text, corresponding to its original plaint text.

A small amount of hardware is used in the MISC AES implementations. Only 1% of available flip-flop and LUTs in the hardware is used. Since there is only a total of 1024 kilobytes memory used for the MSIC architecture implementation, only 3% of the available Block RAMs in the RC10 board is occupied. The hardware usage can be found in the Table 3.

TABLE II.
THE MASTER KEY AND THE CIPHER TEXT OF THE AES

Plaintext	00112233445566778899AABBCCDDEE
Master Key	000102030405060708090A0B0C0D0E
Cipher Text	69C4E0D86A7B0430D8CDD78070B4C55A
No. of Rounds (N_R)	10

TABLE III.
THE MASTER KEY AND THE CIPHER TEXT OF THE AES

Component	Quantity	Total	Usage
Flip Flop	188 slice	13,312	1%
4 input LUTs	317	26,624	1%
- Logic	278	-	-
- Route-thru	37	-	-
- Dual Port Rams	0	-	-
- Shift registers	2	-	-
Bonded IOB	46	221	20%
Block RAMs	1	32	3%
BUFGMUX	3	8	37%
DCMs	1	4	25%

VI. CONCLUSION

With the aid of MISC, a small and compact AES Processor can be realized. With this simple MISC architecture and a series of simple instructions, the total memory required is less than 1k bytes. With this small amount of memory requirement, the encryption process would take a small portion of the memory, and nonetheless, the simplified MISC Architecture implements on simple hardware add value to the MISC's compactibility.

REFERENCES

[1] William F. Gilreath, Phillip A. Laplante, "Subtract and Branch if Negative (SBN)," Computer Architecture: A Minimalist Perspective, Springer, United States of America, pp. 41-42, 2003.

[2] Farhad Mavaddat (Department of Computer Science, University of Waterloo) and Behrooz Parhami (School of Computer Science, Carleton University Ottawa), "URISC: The Ultimate Reduced Instruction Set Computer". Int. J. Elect. Enging EDUC, VOl 25. pp. 327 - 334, Manchester U.P, 1988.

[3] Richard R. Oehler, Reduced instruction set computer (RISC), Encyclopedia of Computer Science, 4th edition, John Wiley and Sons Ltd., Chichester, 2003, Pages 1510 – 1511.

[4] http://nc.kau.ac.kr/index.php/URiSC_backup.

[5] Renner, R, Ada on reduced instruction set computers, for real-time embedded systems, Digital Avionics Systems Conference, 1990. Proceedings., IEEE/AIAA/NASA 9th, 1990, pages 171-176.

[6] Phillip Laplante and William Gilreath, One Instruction Set Computers for Image Processing, Journal of VLSI Signal Processing v 38, 2004, Pages 45 – 61.

[7] William Gilreath and Phillip Laplante, A one instruction set architecture for genetic algorithms, Biocomputing, Nova Science Publishers, Inc., Commack, NY, 2004.

[8] Yee Wei Law, Jeroen Doumen and Pieter Hartel, Survey and Benchmark of Block Ciphers for Wireless Sensor Networks, ACM transactions on Sensor Networks, Vol 2, No. 1, February 2006. (Pages 65 – 93).

[9] Xinmiao Zhang and Keshab K. Parhi, High-Speed VLSI Architectures for the AES Algorithm, IEEE transactions on very large scale integration (VLSI) systems, vol. 12, no. 9, 2004

[10] Tim Good and Mohammed Benaissa, Very Small FPGA Application-Specific Instruction Processor for AES, IEEE transactions on circuits and systems, vol. 53, no. 7, 2006.

[11] Federal Information Processing Standards Publication 197, November 26, 2001: Announcing the Advanced Encryption Standard (AES).