Noname manuscript No.
(will be inserted by the editor)

CIARP: A RISC Processor For Cryptography Applications

Nima Karimpour Darav · Reza Ebrahimi Atani · Erfan Aghaei · Ahmad Tahmasivand · Mahsa Rahmani · Mina Moazzam Jazi

the date of receipt and acceptance should be inserted later

Abstract Security is one of the most important features of industrial products. Cryptographic algorithms are mainly used for this purpose to obtain confidentiality and integrity of data in industry. One of the main concerns of researchers in designing cryptographic algorithms is efficiency in either software implementation or hardware implementation. However, the efficiency of some well-known algorithms is highly questionable. The main goal of this paper is to present a novel processor architecture called CIARP (stands for Crypto Instruction-Aware RISC Processor) being feasible for high speed implementation of low throughput cryptographic algorithms. CIARP has been designed based on a proposed instruction set named Crypto Specific Instruction Set (CSIS), that can speed up encryption and decryption processes of data.

Keywords Co-design cryptography · CIARP processor · Crypto-Purpose Instruction

1 Introduction

Old-time necessity for security and data protection against unauthorized access to classified information in many industries especially in military application is undeniably sobering. Hence, Cryptography plays a significantly important role in the security of data transmission.

On one hand, with developing computing technology, implementation of sophisticated cryptographic algorithms has become feasible. On the other hand, stronger cryptographic specifications are needed in order to be reluctant to possible threats. Some well-known examples of cryptographic algorithms are DES and AES. The Data Encryption Standard (DES) has been the U.S. government standard since 1977. However, now, it can be easily cracked inexpensively. In 2000, the Advanced

N. Karimpour. D
Lahijan Branch, Islamic Azad University, Guilan, Iran
E-mail: karimpour@alum.sharif.edu

Reza Ebrahimi Atani · Erfan Aghaei · Ahmad Tahmasivand · Mahsa Rahmani · Mina Moazam
Department of Engineering, University of Guilan, Guilan, Iran

Encryption Standard (AES) was substituted for the DES to meet ever-increasing requirements for security.

General purpose processors are mostly used to speed up data manipulation and information processing in systems. Nevertheless, these processors are not performance efficient when they are utilized for data encryption and decryption, mainly because many cryptographic algorithms need bit-oriented operations in contrast with what general purpose processors work based on. Data are manipulated by word-oriented operations in processors. Consequenty, bit-oriented manipulating by means of them poses undesirable time overhead although some approaches like hardware/software co-design can mitigate the problem. By the way, the most well-known approaches which are used for implementations of cryptographic algorithms can be enumerated as:

- General purpose processors and DSP.
- Application Specific IC (ASIC).
- Reconfigurable hardware or encryption systems (FPGAs).
- Crypto processor and Crypto-Coprocessor.

The main goal of this paper is to present an especial purpose instruction set called CSIS (Crypto Specific Instruction Set) that can be utilized for cryptographic algorithms. This instruction set is structured based on some bit and byte-oriented operations. In order to show their appropriateness, a cryptographic processor named CIARP (Crypto Instruction-Aware RISC Processor) has been designed. By this novel processor, the time order of the standard cryptographic algorithms has been reduced compared to those run on general purpose processors. We believe that CSIS and CIARP open a new field in implementation of cryptographic algorithms.

The rest of the paper is organized as follows:Related Work is presented in section 2. a general description of CIARP is presented in section 3. Afterwards, The assembler of CIARP is presented in section 4. Section 5 describes the syntehsis result of the CIARP processor. Section 6 describes the simulation and implementation results of the CIARP processor. Finally, the paper is concluded in section 7.

2 Related Work

Research to develop a device which can support high efficient implementation of multiple cryptographic algorithms to reduced the time order of the execution has gained considerable momentum in the last decade, since it reduces efforts of the chip makers to come up with a new chip as new cryptographic algorithms continue to emerge.So we can divide implementation of cryptographic algorithms to three basic techniques:

- **Fully Hardware Cryptographic algorithms Implementation-** There have been many different hardware realization and implementation of Cryptographic algorithms on FPGA and ASIC platform. Refs. [1]through [17] present the FPGA implementations of the Cryptographic algorithms. All of the architectures used in those works can achieve the throughput rate of several Gbps [18].ASIC and FPGA technologies provide the opportunity to augment the existing datapath of a processor implemented via an IP core to add acceleration modules

supported through newly defined instruction set extensions targeting performance critical functions [19], [20], [21].Though it is an energy efficient solution, the downside of it is lack of flexibility and scalability to support new Cryptographic algorithms.

– **Fully Software Cryptographic algorithms Implementation-** The advantages of a software implementation include ease of use, ease of upgrade, ease of design, portability, and flexibility. However, a software implementation offers only limited physical security, especially with respect to key storage [22]. The most significant disadvantage of software based solutions is that the speed performance is significantly lower than that based on hardware [23]. Most traditional methods for improving the throughput of pure software implementations of symmetric-key algorithms fall into one of two categories. One option is to construct memorybased look-up tables where results of some of the basic operations of the algorithm have been pre-computed and stored.Another method for speeding up software implementations of cryptographic algorithms involves taking advantage of mathematical or structural properties of the particular algorithm. The software implementation requires very high time to execute. The hardware implementation is better than software implementation in terms of processing speed and power consumption. The combination of both the implementations constitutes a co-design, which offers flexibility to implement complex systems.
– **Co-design Cryptographic algorithms Implementation-** The hardware / software co-design methodology is adopted to implement one of the functional modules in hardware and subsequent remaining modules in software .For example, An AES has four modules in the algorithm, each one of them is performing a specific function.The modules are addroundkey, subbyte, shiftrow, and mixcolumn. At first, the complete algorithm is implemented in software. The timing profile is done. The mixcolumn, block with more computational complexity is shifted in hardware. The remaining three blocks are kept in software only [24]. The software (SW) critical part i.e., a mixcolumn, module of AES is implemented inhardware (HW) in VHDL language [25]. The main idea of AES implementation is to achieve advantages of the parallel structures, which can be efficiently implemented in hardware [26].By using parallel hardware structure, there is a reduction in the number of operations and also parallel operation is possible. An implementation result shows a considerable improvement in speed as compared to software only approach. On the other hand, the significant reduction in area is achieved as compared to hardware only approach.Further, the software and hardware blocks are combined together and co-design implementation is done. By applying the suggested co-design approach for an AES encryption algorithm the execution time is accelerated and power consumption is reduced as compared to that of software implementation. By incorporating the hardware / software codesign methodology, a significant reduction in area usage and thermal power dissipation is achieved as compared to custom hardware. The considerable reduction in time further allows the preference ofsuggested methodology over software. The hardware /software co-design methodology can be easily extended to any application. The application under consideration in the present work is Rijindaels Encryption Algorithm (AES) [35].

The main open problem is:

Can the techniques that noted above implement a wide range of Cryptography algorithms efficiently ? In this paper, We introduce CSIS concept (i.e. we have designed innovative instruction based on Shannon's principleregarding the confusion and diffusion principles [27] , Bit-oriented , and Byte-oriented operations). So we use the concept of CSIS to implement a wide range of cryptographic algorithms efficiently to reduce the time order of the execution.
On one hand, The CSIS concept uses the advantages of software implementation such as ease to use, ease of upgrade, ease of design, portability, and flexibility and eliminates the disadvantage of software implementation "very high time to execute". on the other hand, It's more scalable and more flexible than co-design implementation.

3 CIARP Processor

CIARP is a 32-bit processor that its architecture has been designed in a way to be modular. In other words it is composed of some basic building blocks such as multiplexers, decoders, registers and counters and whole processor has been implemented by interconnecting these modules that makes it a fully-synthesizable processor.

Each instruction cycle in CIARP is composed of four T-states. The CIARP instruction set includes two categories of computing instructions. One group contains word-oriented instructions like those can be found in general purpose processors such as arithmetic and logical operations. Another class of instructions is bit and byte oriented so that they can be exploited in implementing algorithms needing such features.

CIARP reaps the benefits of RISC architectures to speed up running programs by utilizing fewer instructions and addressing modes and more registers as well as pipeline technique and register windowing in its structure. The memory is accessible via only two instructions (*load* and *store*). Furthermore, this processor has some control instructions like unconditional and conditional branch instructions.

The main part of CIARP is IR-Decoder that has been designed to decode the instructions. In other words, according to the given instruction, it determines which unit in the next stage should be activated. One of the distinct features of CIARP is the barrel shifter that is able to shift a word as many as it is desired in one clock pulse. The most important instructions of CIARP are BITP, BYTP and RXOR that will be disscussed further.

3.1 The Register File

Here, CIARP registers are presented. In all registers, the bit with number zero is assessed as the LSB (Least Significant Bit) while the MSB (Most Significant Bit) is recognized as the bit with number 31.

3.1.1 General Purpose Registers (GPR)

These 32-bit registers are used as the source or the destination in most of instructions.

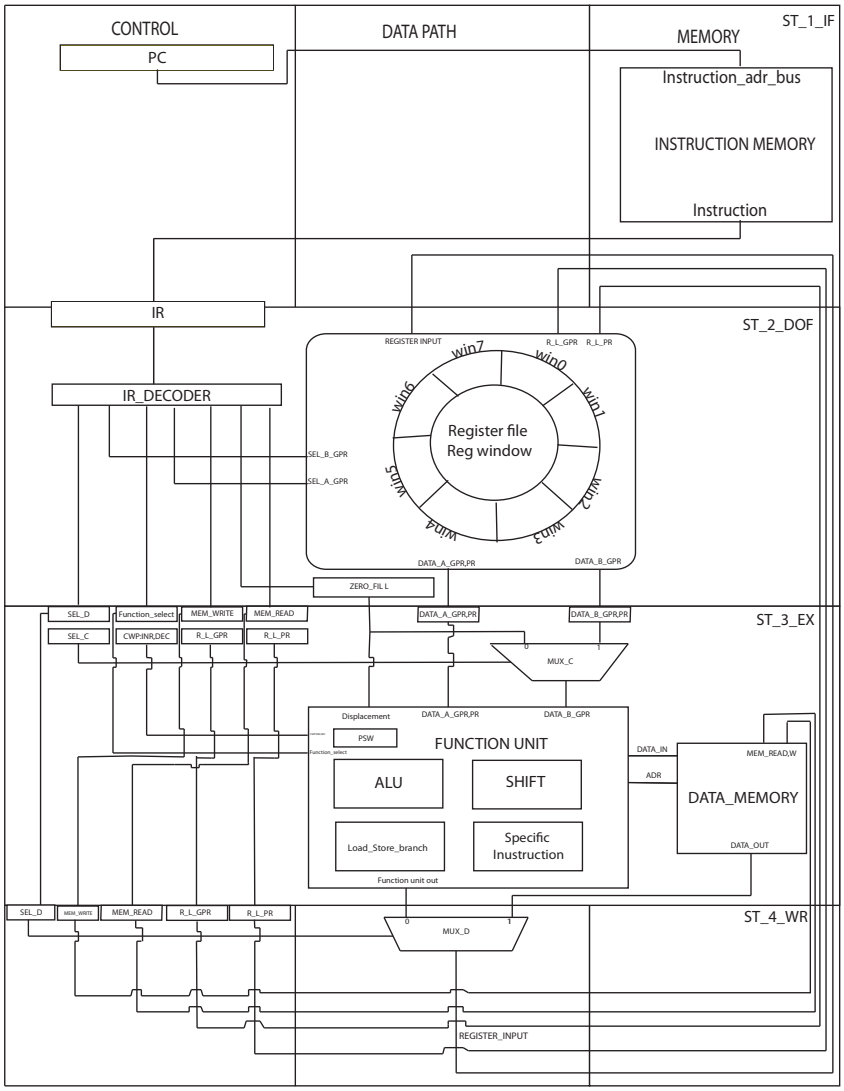

Fig. 1 The internal structure of CIARP.

3.1.2 Permutation Registers (PR)

PRs are 32-bit registers that specify the pattern of permutation in crypto-specific instructions. They determine the number of bits or bytes in bit-oriented or byte oriented instructions.

3.1.3 Program Counter (PC)

It is a 32-bit counter that ascertains the sequence of the program.

3.1.4 Address Register (AR)

A 32-bit register that is used for memory addressing.

3.1.5 Program Status Word (PSW)

It is 32-bit register that each bit of it determines a status except some bits that have been considered for future use. Fig. 2 shows its structure and the functionality of each bit.

31 10	9 7	6	5	4	3	2	1	0
Reserved	CWP	IE	PF	CF	OF	SF	EQ	GT

Fig. 2 The functionality of each bit in PSW.

GT and *EQ* are used in the comparison instructions and refreshed when a comparison instruction is issued. If the first register is greater than the second, *GT* bit is set to 1 while EQ bit is set to 1 if both registers are equal. In other cases, both of them are filled by zero. *SF* is known as the Sign Flag and is set to 1 when the result of an arithmetic instruction is negative. *CF* is used to save the carry generated by arithmetic operations and by RXOR as well. When overflow occurs, *OF* is set to 1. In arithmetic and logical instructions, if the result has even parity, *PF* is set to 1. *IE* is a writable bit to enable maskable interrupts. Finally, *CWP* known as the Current Window Pointer of CIARP is a 3-bit field to keep the sequence of window switching in the 8-window register file of CIARP.

3.2 Addressing Mode

Most instructions work directly with registers and the memory is accessible through *load* or *store* instructions like what are designed in RISC processors. There are four addressing modes in CIARP: Immediate addressing, Register Direct Addressing, Register Indirect Addressing, and Index addressing.

3.2.1 Immediate Addressing Mode

It can be used for all instructions except for *load* and *store* instructions. This kind of addressing mode is shown in Fig. 3.

3.2.2 Register Direct Addressing Mode

As the former addressing mode, It can be used for all instructions except for *load* and *store* instructions. This kind of addressing mode is shown in Fig. 4.

3.2.3 Register Indirect Addressing Mode

It is utilized in the *load* and *store* instructions. For the *load* instruction, the source field refers to a register containing the address of the desired operand. By contrast, in *store* instruction, the destination field uses this addressing mode.

(i)

AM		29 24	23 0
0	0	Opcode	Immediate

(ii)

AM		29 24	23 18	17 12	11 0
0	0	Opcode	Op3	Unused	Immediate(Op1)

(iii)

AM		29 24	23 18	17 12	11 0
0	0	Opcode	Op3	Op2	Immediate(Op1)

Fig. 3 Immediate addressing mode for (i) Single operand instructions (ii) two operand instructions and (iii) three operand instructions.

(i)

AM		29 24	23 18	17 12	11 6	5 0
0	1	Opcode	Op3	Op2	Op1	Unused

(ii)

AM		29 24	23 18	17 12	11 6	5 0
0	1	Opcode	Op3	Op2	Op1	Op0

(iii)

AM		29 24	23 18	17 0
0	1	Opcode	Destination	Unused

Fig. 4 Register direct addressing mode for (i) three operand instructions (ii) four operand instructions and (iii) branch statements.

3.2.4 Index Addressing Mode

As the previous addressing mode, it is used in load and store instructions. There are two fields to indicate this kind of addressing mode. One field is reserved for *displacement* while the other field refers to the index register. The effective address in this mode is as follows:EA=$index$+$displacement$. This kind of addressing mode is shown in Fig. 5.

AM		29 24	23 18	17 12	11 0
1	1	Opcode	Destination	Source	Displacement

Fig. 5 Index Addressing Mode.

3.3 ISA(Instruction Set Architecture)

CIARP's ALU support many instructions like arithmatical, logical, branch, load and store instructions that define in RISC processors. CIARP's ISA consist of some insrtuction that specialize CIARP from othere processors. These instructions named CSIS and will be explained in upcoming sections. Instruction set of CIARP and explanations shown in Table. 2. Instructions divided to three parts depend on operand numbers: one, two and three operand instructions as they are mentioned in previous section and shown in Table. 2. Just the third operand in three operand instructions and the second operand in two operand instructions and the first in one operand instructions can be point to an immediate value in immediate and index addressing modes. Otherwise, the operands must be point to a register.

Table 1 Instruction set architecture of CIARP (IM means IMmediate, IN means INdex, RD means Rgister Direct and RI means Register Indirect).

Instruction	Addressing modes	Description
ADD Op1,Op2,Op3	RD,IM	Op1=Op2+Op3
SUB Op1,Op2,Op3	RD,IM	Op1=Op2-Op3
MUL Op1,Op2,Op3	RD,IM	Op1=Op2*Op3
LOAD Op1,Op2	RI	Op1=[Op2]
LOAD Op1,Op2,OP3	IN	Op1=[Op2+Op3]
STORE Op1,Op2	RI	[Op1]=Op2
STORE Op1,Op2,OP3	IN	[Op1]=Op2+Op3
AND Op1,Op2,Op3	RD,IM	Op1=Op2 (and) Op3
OR Op1,Op2,Op3	RD,IM	Op1=Op2 (or) Op3
XOR Op1,Op2,Op3	RD,IM	Op1=Op2 (xor) Op3
SHR Op1,Op2,Op3	RD,IM	Op1=Rigth shifting of Op2 that Op3 specify number of shifting
SHL Op1,Op2,Op3	RD,IM	Op1=Left shifting of Op2 that Op3 specify number of shifting
ROR Op1,Op2,Op3	RD,IM	Op1=Rigth rotating of Op2 that Op3 specify number of shifting
ROL Op1,Op2,Op3	RD,IM	Op1=Left rotating of Op2 that Op3 specify number of shifting
RORC Op1,Op2,Op3	RD,IM	Op1=Rigth rotating of Op2 with Carry Bit that Op3 specify number of shifting
ROLC Op1,Op2,Op3	RD,IM	Op1=Left rotating of Op2 with Carry Bit that Op3 specify number of shifting
CMG Op1,Op2	RD,IM	GT= 1 If Op1 Greater than Op2
CML Op1,Op2	RD,IM	LT= 1 If Op1 Less than Op2
CME Op1,Op2	RD,IM	EQ= 1 If Op1 Equals with Op2
JMG Op1	RD,IM	If GT==1, PC=Op1
JML Op1	RD,IM	If LT==1, PC=Op1
JME Op1	RD,IM	If EQ==1, PC=Op1
JMP Op1	RD,IM	PC=Op1
MGG Op1,Op2	RD	Op1=Op2
MGP Op1,Op2	RD	Op1=Op2
MPG Op1,Op2	RD	Op1=Op2
MPP Op1,Op2	RD	Op1=Op2
MIP Op1,Op2	IM	Op1=Op2
MIG Op1,Op2	IM	Op1=Op2
CALL Op1	RD,IM	Push PC then PC=Op1
RET	Implicit	POP PC

3.4 The Pipeline Structure of CIARP

Pipelining is an implementation technique in which the phases of the instruction cycle are executed simultaneously. While an instruction is being fetched from the memory, the former instructions are being processed in following stages.CIARP takes the advantages of a 4-stage pipeline structure [28]. Each of fetch cycle, decode cycle, execute cycle, and write back cycle has been considered one stage in the CIARP pipeline architecture. Hence, all instructions run in four clock periods. Fig. 1 illustrates the pipeline architecture of CIARP.

3.5 Register Windows

Register Files represent a substantial portion of the energy budget in modern microprocessors [29] and [30]. The techniques for reducing the size include sharing an entry among several operands with the same value [31] and [32], and dividing the register storage hierarchically [33] and [34]. The register model of CIARP is the same as the SPARC architecture and uses the same register windowing mechanism that is described in this section.

The SPARC architecture uses a windowed register file model in which the file is divided up into groups of registers called windows [35]. This windowed register model simplifies compiler design and accelerates procedure calls.

The implementation of the proposed CIARP processor contains 68 GPR registers and 64 PR registers that are 32-bit wide and are classified into a set of 128 window registers and a set of 4 global registers. The 128 window registers are grouped into 8 sets of 12 GPR registers and 12 PR registers called windows. Thus, the register file consists of 8 register windows where each window includes of a set of 24 registers. While a program is running, it has access to 28 32-bit processor registers which include 4 global registers plus 24 registers that belong to the current register window. As it is shown in Fig. 6, The first 8 registers (GPR and PR) in the window are called the *in* registers (in4-in7 (GPR) and in4-in7 (PR)). When a function is called, these registers may contain arguments that can be used. The next 8 registers are the *local* registers (local8-loca11 (GPR) and local8-local11 (PR)) which are scratch registers that can be used for anything while the function executes. The last 8 registers are the *out* registers (out12-out15 (GPR) and out12-out15 (PR)) which the function uses to pass arguments to functions that it calls. At any given time, a program can access an active 28-register window (24 window registers) and the four global registers. The current active window (the window visible to the programmer) is identified by the Current Window Pointer (CWP). Either incrementing or decrementing the CWP results in an eight register overlap between windows. This overlap of window registers is used to pass parameters from one window to the next.

3.6 The Structure of Barrel Shifter in CIARP

The key objective of today's circuit design is to increase the performance without the proportional increase in power consumption. Shifting and rotating are required in many operations such as arithmetic and logical operations, address decoding

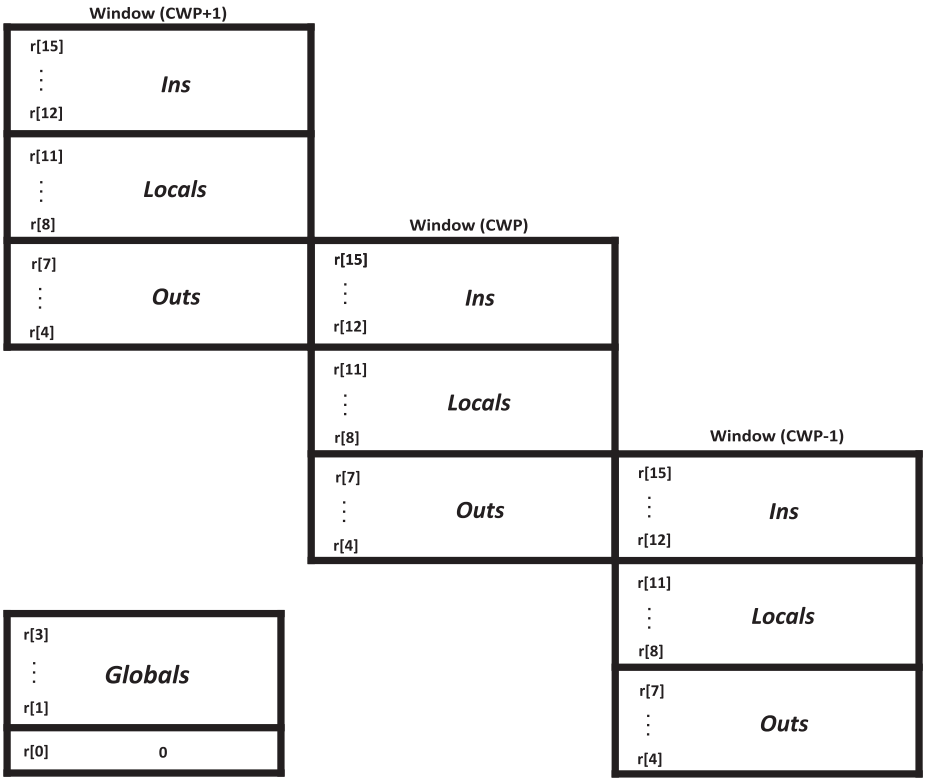

Fig. 6 The functionality of the designed register window.

and indexing etc. Barrel shifters, which can shift and rotate multiple bits in a single cycle, have become a common design choice for high speed applications. For this reason, CIARP reap the benefits of the method that uses multipliers in its barrel shifter [36] and [37]. A 32-bit barrel shifter requires thirty-two, 32-to-1 multiplexers. A 32-to-1 multiplexer can be implemented in a Spartan-3a device using two CLBs. Only sixty four CLBs are required to accomplish all the required multiplexing. By using a multiplier-based barrel shifter, a 32-bit barrel shifter is built using four 8-bit barrel shifters and thirty two 4-to-1 multiplexers.

The diagram on the left side of Fig. 7 is a single-cycle, 32-bit barrel shifter. The input bus is broken down into four 8-bit words. The data is processed in two stages. The first stage is constructed of the 8-bit barrel shifters. This stage provides the fine shifting, moving the bits from adjoining bytes. Passing the first stage, the appropriate bits are stored in a byte; however the bytes need to be reordered. The reordering of the bytes, or bulk shifting, is applied in the second stage, as shown on the right in Fig. 7. The 8-bit barrel shifter requires the shift amount being in the one-hot encoded format where three LSBs are used to control the fine shifting, and the two MSBs are used to control the bulk shifting [9].

CIARP supports six kinds of shift operations (SHL, SHR, ROR, ROL, ROLC, and RORC) and uses Hardware sharing technique to reduce the number of slices

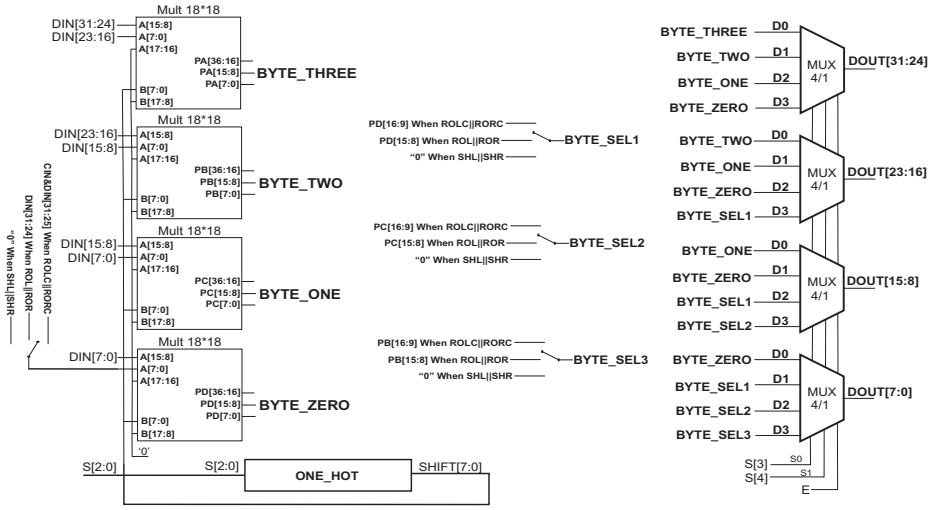

Fig. 7 The barrel shifter based on multipliers.

Table 2 The synthesis results of the barrel shifter in CIARP.

	Used	Available	Utilization
Number of slices	117	11264	1%
Number of occupies	107	11264	1%

and the hardware cost without missing performance. Table 2 shows the synthesis results of the barrel shifter of CIARP.

3.7 Interrupt

When an interrupt occurs by peripheral device, an 8-bit vector is placed on the bus Interrupt Vector and the CPU in interrupt cycle jump to the location of the memory which interrupts vector indicates.

3.8 The Crypto-Purpose Instructions

3.8.1 The BITP Instruction

It is a bit-oriented instruction with four operands that is demonstrated in Fig. 8. A sample use of BITP is shown in 1.

$$BITP\ GPR1,\ GPR2,\ PR1,\ PR2 \tag{1}$$

It is a bit permutation on lower 16 bits of GPR2 (as the source register) based on the pattern determined by PR1 and PR2 registers, and the result is stored in GPR1 as the target register.

The PR registers are divided into eight parts containing 4 bits. Each part is able to indicate one bit out of lower 16 bits in GPRs. In the BITP instruction,

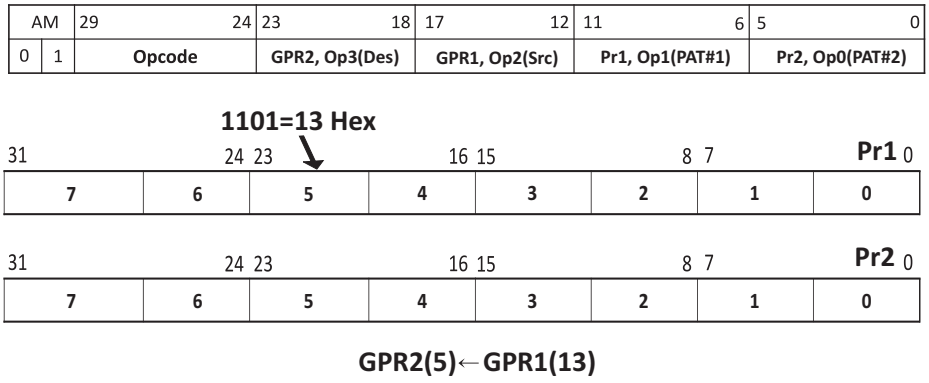

Fig. 8 The functionality of the BITP instruction.

PR2 specifies the bits of GPR2 (the source register) to be copied into those bits of GPR1 (the target register) that are designated by PR1. For example if the value of part 5 of PR2 and PR1 is 13 and 5 respectively, the content of the bit 13 of GPR2 will be copied into the bit 5 of GPR1. Any GPR can be substituted for GPR1 and GPR2 as well as PR1 and PR2 which can be replaced by any PR.

3.8.2 The BYTP Instruction

It is a byte-oriented instruction with three operands that is demonstrated in Fig. 9. A sample use of BYTP is shown in 2.

$$BYTP\ GPR1,\ GPR2,\ PR1 \tag{2}$$

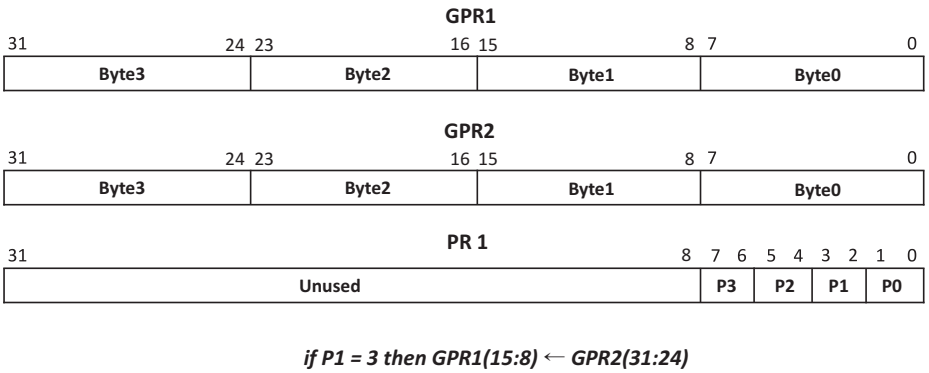

Fig. 9 The functionality of the BYTP instruction.

It is a byte permutation on the four bytes of GPR2 (as the source register) based on the pattern determined by PR1, and the result is stored in GPR1 as the target register.

The PR registers are divided into four parts containing 2 bits. Each part is able to indicate one byte out of the four bytes in GPRs. In the BYTP instruction,

PR1 specifies the bytes of GPR2 (the source register) to be copied into relative bytes of GPR1 (the target register). For example if the value of part 1 of PR1 is 3, the contents of byte 3 of GPR2 will be copied into byte 1 of GPR1. Any GPR can be substituted for GPR1 and GPR2 as well as PR1 and PR2 which can be replaced by any PR.

3.8.3 The RXOR Instruction

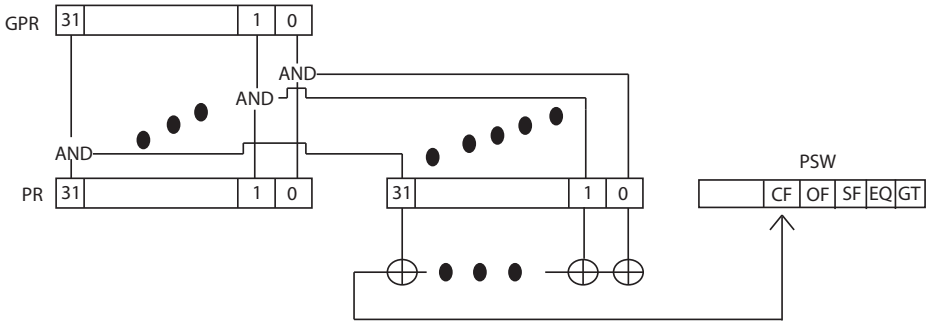

Fig. 10 The functionality of the RXOR instruction.

A sample use of BYTP is shown below:

RXOR GPR1, PR1

As it is shown in Fig. 10, this instruction applies the XOR operation on all bits of GPR2 according to contents of PR1 and stores the result into CF which is a flag bit in PSW.

4 CIARP Assembeler

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions. It is usually embedded as part of a complete device including hardware and mechanical parts. Since embedded systems usually use processors with limited computational power and few hardware resources, it is important to write efficient programs controlling the systems [38]. In order to automate generating executable code for CIARP, we have designed an assembler that converts assembly instructions written in a file into machine language instructions [39]. Assembler provided in java programming language and support the instructions in Table 2 and CSIS.

5 Synthesis Results

Having been completed the design process, in order to verify its performance, the processor was implemented on hardware using VHDL and Xilinx ISE synthesis

tool. The target FPGA used in the implementation was Spartan3a xc3s1400an device from Xilinx. The result of the synthesis showed that the full implementation needs 5,127 (45% of 11264) slices and its critical path limited the maximum clock frequency on 69 MHz. The post-synthesis simulation verified the design goals of CIARP. Synthesis results detail shown in Table 3.

Table 3 The synthesis results of CIARP.

Components	Number of occupies Slice	Utilization	Critical path Delay
CIARP	5127	45%	14.299ns
Data path	4956	43%	12.614ns
Control unit	162	1%	5.512ns
Register file	5162	45%	6.081ns
Stage3 - data path	644	5%	10.896ns
Function unit	618	5%	-
Special order	160	1%	-
ALU	156	1%	-
Load store branch	132	1%	-
Shift	107	1%	-

6 Experiments, Results and discussions

To evaluate our specific instructions, in cryptography algorithms, one issue should be discussed: The number of cycles for implementation (Ordering). Notice that, using specific instructions can reduce number of cycles, so an accuracy measure number of cycles should be used. Therefore, for these objectives, the number of cycles should be evaluated with and without specific instructions. So, to achieve this purpose the AES-128bit algorithm as Block Cipher and the A5/1 and Salsa20 as Stream Ciphers implement on CIARP, MICROBLAZE and LEON3 [40] [41] processors.

6.1 Implementation of AES algorithm on Leon3

The first step for implementation of AES on LEON3 is the configuration of LEON3 processor according to our requirements. For the configuration of LEON3 Processor, first enter into the grlib-gpl-1.2.0-b4121/designs/leon3-gr-xc3s-1500 and then give the command, make xconfig[] Using GUI interface configure various aspects of LEON3 processor. Click the 'Processor' button and it will give various options for configuring integer unit, floating-point unit, cache system, memory management unit, debug support unit etc. For this project we have disabled the floating-point unit and memory management unit and enabled debug support unit and Accelerated UART tracing. To copy the configuration to leon3-gr-xc3s-1500/config.vhd file click on 'save and exit'.

To compile the AES program written in C, we have to first copy the program in systest.c of directory grlib-gpl-1.2.0-b4121/designs/leon3-gr-xc3s-1500 and then

give command, make soft. The design can be compiled by giving the following command in grlib-gpl-1.2.0-b4121/designs/leon3-gr-xc3s-1500 directory, make vsim. It compiles all .vhd files [42].

To simulate the design, first enter into the grlib-gpl-1.2.0-b4121/designs/leon3-gr-xc3s-1500 directory and then give command, vsim testbench. The subdirectory 'software' contains all the test files for the processor. Each test has been described in a separate file. These tests are compiled into an sdram.srec file which is loaded into the memory of the processor while simulation. Finally to start the simulation give command. run -all. It runs the simulation completely. Simulation is halted by generating a failure.

6.2 Implementation of AES algorithm on CIARP

The first step for implementation of AES on CIARP is the AES program written in CIARP assembly, we have to first copy the program in Assembler(CIARP) and then click the 'Open and Save Directory Of instructions' button it will give instruction.txt file that should be copied in /CIARP-IP directory. /CIARP-IP directory contains of two different instruction .txt and data .txt files which consequently loaded into the instruction memory and the data memory of the processor while simulation, and .vhd files which are CIARP core. To simulation the design, first enter into /CIARP-IP directory and then give command, vsim testbench. Finally to start the simulation give command. run -all. It runs the simulation completely.

6.3 Implementation of A5/1 algorithm

The A5/1 is a stream cipher and divided to six step for implementation. The fifth step of A5/1 consist of six basic function [43]:

- bit parity(word x).
- word clockone(word reg, word mask, word taps).
- bit majority().
- void clock().
- void clockallthree().
- void getbit().

We imeplement the fifth step of A5/1 on three different processors CIARP, MICROBLAZE, and LEON3.

6.4 Implementation of Salsa20 algorithm

Salsa20 is a stream cipher submitted to eSTREAM by Daniel J. Bernstein. It is built on a pseudorandom function based on 32-bit addition, bitwise addition (XOR) and rotation operations. The one round of Salsa20 for implementation on CIARP, MicroBlaze, and LEON3 shown at Fig. 11.

$$z1 = y1 \text{ xor } ((y0 + y3) << 7)$$
$$z2 = y2 \text{ xor } ((z1 + y0) << 9)$$
$$z3 = y3 \text{ xor } ((z2 + z1) << 13)$$
$$z0 = y0 \text{ xor } ((z3 + z2) << 18)$$

Fig. 11 The one round of Salsa20.

Table 4 Simulation Report

CPU core	AES-128	A5/1	Salsa20	Frequency Mhz
CIARP	5426	41	14	50
MICROBLAZE	8378	420	14	50
LEON3	8888	2888	41	50

Table 5 Memory Usage

CPU core	AES (byte)	A5/1 (byte)	Salsa20 (byte)
CIARP	1292	336	48
MICROBLAZE	2388	552	48
LEON3	15224	4656	48

6.5 Memory-Usage of algorithms

As shown in Table 5 The CIARP Processor not only reduce number of time order for implementing cryptography algorithms but also has better results for memory usage comparing to other processors.

6.6 Results

Following Implementation AES-128bit, A5/1 and Salsa20 algorithms on three different processors CIARP, MICROBLAZE, and LEON3 Results shown at Table 4 and Table 6.

6.7 discussions

The basic reason of reduce number of ordering in CIARP processor for implementation A5/1 is using RXOR instruction while other processors lack of this instruction and use parity function. Fig. 12 show how RXOR instruction decrease number of ordering.

On other hand, The basic reason of decrease number of ordering in CIARP processor for implementation AES is using BYTP instruction while other processors

Table 6 CPU Soft Cores

CPU core	License	Pipeline depth	Cycles per instruction	Area (LEs)	Comments
CIARP	-	4	1	5000	Crypto Specific Instruction Set (CSIS)
MICROBLAZE	Proprietary	3, 5	1	3500	Limited to Xilinx devices
LEON3	Open-source (GPL)	7	1	1300	-

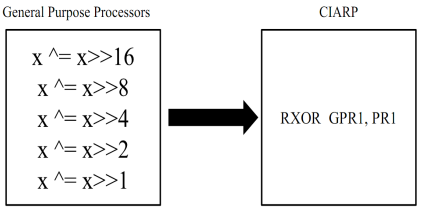

Fig. 12 The functionality of the RXOR instruction.

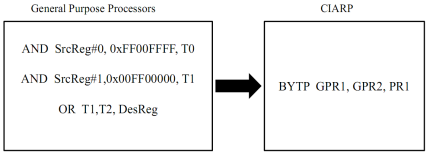

Fig. 13 The functionality of the BYTP instruction.

lack of this instruction. Fig. 13 show how BYTP instruction decrease number of ordering.

7 Conclusion

In this paper a novel architecture for crypto processors were presented. The proposed idea of CSIS was implemented by means of the architecture in a way to reduce the time order of the execution of the cryptographic-related instructions. Three instructions called BYTP, BITP and RXOR are introduced. The simulation results show that they are able to speed up execution of stream ciphers and block ciphers.

References

1. W. K. Gielata A, Russek P, "Aes hardware implementation in fpga for algorithm acceleration purpose." in *International Conference on Signals and Electronic Systems*, 2008, pp. 40–137.

2. M. J. R. McLoone M, "Fpga implementations utilizing lookup tables," *VLSI Signal Processing*, vol. 34, no. 2, pp. 75–261, 2003.
3. S. J. Jrvinen KU, Tommiska MT, "A fully pipelined memoryless 17.8 gbps aes-128 encryptor," in *the International Symposium on Field Programmable Gate Arrays*, 2003, pp. 15–207.
4. J. K., *Study on high-speed hardware implementation of cryptographic algorithms*. Department of Signal Processing and Acoustics Helsinki University of Technolog, 2009.
5. Q. J. L. J. Standaert F, Rouvroy G, "Efficient implementation of rijndael encryption in reconfigurable hardware: improvements and design tradeoffs," in *CHES 2003*.
6. M. N. S. A. Saggese GP, Mazzeo A, "An fpga-based performance analysis of the unrolling, tiling, and pipelining of the aes algorithm." in *FPL 2003*, 2003, pp. 292–302.
7. V. I. Hodjat A, "A 21.54 gbits/s fully pipelined aes processor on fpga." in *the 12th IEEE symposium on field-programmable custom computing machines*, 2005, pp. 9–308.
8. P. K. Zhang X, "High-speed vlsi architectures for the aes algorithm." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, pp. 67–957, 2004.
9. S. P. J. M. G.-P. J. A. Granado-Criado JM, Vega-Rodriguez Miguel A, "A new methodology to implement the aes algorithm using partial and dynamic reconfiguration," *Department Technologies of Computers and Communications,University of Extremadura, Spain*, 2009.
10. P. D. B. J. Yoo S-M, Kotturi D, "An aes crypto chip using a high-speed parallel pipelined architecture." *Microprocessors and Microsystems Elsevier Journal*, vol. 29, pp. 26–317, 2005.
11. A. S. Saleh AH, "High performance aes design using pipelining structure over gf ((24)2)," in *IEEE International Conference on Signal Processing and Communications*, 2007, pp. 9–716.
12. E. A. Rady A, ElSehely E, "Design and implementation of area optimized aes algorithm on reconfigurable fpga." in *international conference on microelectronics*, 2007, pp. 8–35.
13. Q. S. Rais MH, "A novel fpga implementation of aes-128 using reduced residue of prime numbers based s-box," *IJCSNS international journal of computer science and network security*, vol. 9, p. 305, 2009.
14. M. A. Rohiem AE, Ahmen FM, "Fpga implementation of reconfigurable parameters aes algorithm." in *13th international conference on aerospace sciences and aviation technology, ASAT*, 2009, pp. 8–26.
15. P. A. Henriquez FR, Saqib NA, "4.2 gbit/s single-chip fpga implementation of aes algorithm." *IEEE Journal Electron Lett*, vol. 39, pp. 6–1115, 2003.
16. C. P. Gaj K, "Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays." in *CT-RSA 2001, LNCS 2020*, 2001, pp. 84–99.
17. D.-F. J. Dyken JV, "Fpga schemes for minimizing the power-throughput trade-off in executing the advanced encryption standard algorithm," *Journal of Systems Architecture: The EUROMICRO Journal*, vol. 56, 2010.
18. F. S. H. N. R. Liakot Ali a, IshakArisb, "Design of an ultra high speed aes processor for next generation it security," *Institute of Information and Communication Technology, Bangladesh University of Engineering and Technology,Dhaka, Bangladesh bInstitute of Advanced Technology (ITMA), Universiti Putra Malaysia, Malaysia*, 2011.
19. L. L. M. Gschwind, A. Jerraya and F. Vahid, "Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays." in *CT-RSA 2001, LNCS 2020*, 2001, pp. 84–99.
20. K. K. ukcakar., "An asip design methodology for embedded systems." in *Seventh International Symposiumon Hardware/Software Codesign*, 1999, pp. 17–21.
21. D. E. M. A. Wang, E. Killian and C. Rowen., "Hardware/ software instruction set configurability for system- on-chip processors." in *the 38th Design AutomationConference DAC 2001*, 2001, pp. 18–22.
22. B. Schneier, *Applied Cryptography.*, second edition ed. John Wiley and Sons Inc., 1996.
23. F. S. H. N. R. a. Liakot Ali a, IshakArisb, "Design of an ultra high speed aes processor for next generation it," in *Elsevier Journal*, 2011.
24. D. E. M. A. Wang, E. Killian and C. Rowen., "Design and implementation of rijindaels encryption algorithm with hardware / software codesign using nios ii processor." in *Meghana A. Hasamnis Associate Professor, Electronics Engineering ShriRamdeobaba College of Engineering and Management, Nagpur, India , S. S. Limaye Professor,Electronics Engineering Jhulelal Institute of Technology Nagpur, India.*, 2001.
25. M. L. W. Chen, P. Kosmas and C. Rappaport., "An fpga implementation of the two-dimensional finite-difference time- domain (fdtd) algorithm," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2004, pp. 97–105.

26. W. F. A. Balboni and D. Sciuto., "Partitioning and exploration in the tosca co-design flow," in *International Workshop on Hardware / Software Codesign (CODES)*, 1996, pp. 62–69.
27. C. Shannon, "Communication theory of secrecy systems," in *Bell System Technical Journal*, 1949, pp. 656–715.
28. D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 3rd ed. Morgan Kaufmann, 2005.
29. D. R. Gonzales, "Micro-risc architecture for the wireless market." in *IEEE Micro*, 1999, pp. 30–37.
30. A. Kalambur and M. J. Irwin, "An extended addressing mode for low power," in *the IEEE Symposium on Low Power Electronicsl*, 1999, pp. 213–208.
31. M. B. B. S. S. Jourdan, R. Ronen and A.Yoaz, "A novel renaming scheme to exploit value temporal locality through physical register reuse and unification," in *the 31st MICRO*, 1998, pp. 216–225.
32. S. Balakrishnan and G. Sohi, "Exploiting value locality in physical register files," in *the 36th MICRO*, 2003, pp. 265–276.
33. M. V. N. T. J.-L. Cruz, A. Gonzalez, "Multiple-banked register file architectures," in *the 27th ISCA*, 2000, pp. 316–325.
34. S. D. R. Balasubramonian and D. Albonesi, "Reducing the complexity of the register file in dynamic superscalar processorss," in *the 34th MICRO*, 2001, pp. 237–248.
35. P. Hall, *The SPARC Architecture Manual*, 8th ed. SPARC International, 1992.
36. H. B. I. Hashmi, "An efficient design of a reversible barrel shifter," in *' International Conference on VLSI design*, 2010, pp. 93–98.
37. P. Gigliottr, "Implementing barrel shifters using multipliers," August 2004, xilinx Corp.
38. Y. I. K. Nakano, "Processor, assembler, and compiler design education using an fpga," in *IEEE International Conference on Parallel and Distributed Systems*, 2008, pp. 723–728.
39. M. M. Manl, *Computer System Architecture, publisher = , Prentice-Hall, year = 1993,.*
40. J. Gaisler, "Grlib ip library users manual," January 2012, version 1.1.0.
41. "Grlib ip core users manual," January 2012, version 1.1.0.
42. "Bcc - bare-c cross-compiler users manual," April 2011, version 1.0.36.
43. "implementation of a5/1." [Online]. Available: http://www.scard.org/gsm/a51.html