Journal of Advanced Research 12 (2018) 67–78

Contents lists available at ScienceDirect

Journal of Advanced Research

journal homepage: www.elsevier.com/locate/jare

Original Article

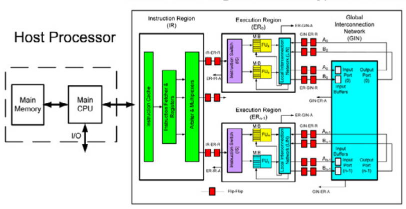
Optimization of a novel programmable data-flow crypto processor using NSGA-II algorithm

Mahmoud T. El-Hadidi [a,*], Hany M. Elsayed [a], Karim Osama [a], Mohamed Bakr [b], Heba K. Aslan [c]

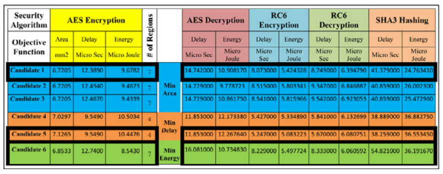
[a] Department of Electronics and Electrical Communications Engineering, Faculty of Engineering, Cairo University, Giza 12613, Egypt
[b] College of Computing and Information Technology, Arab Academy of Science and Technology and Maritime Transport, Cairo, Egypt
[c] Informatics Department, Electronics Research Institute, Cairo, Egypt

GRAPHICAL ABSTRACT

Data-Flow Programmable Crypto Processor

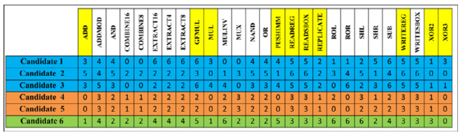


Six candidate solutions on the Pareto front obtained by applying modified NSGA-II optimization algorithm to the programmable data-flow crypto processor.



Security Algorithm	AES Encryption			# of Regions		AES Decryption		RC6 Encryption		RC6 Decryption		SHA3 Hashing	
Objective Function	Area	Delay	Energy			Delay	Energy	Delay	Energy	Delay	Energy	Delay	Energy
	mm2	Micro Sec	Micro Joule			Micro Sec	Micro Joule	Micro Sec	Micro Joule	Micro Sec	Micro Joule	Micro Sec	Micro Joule
Candidate 1	6.7205	12.3890	9.6782	7	Min Area	14.742000	10.908170	8.073000	5.424328	8.749000	6.394790	41.379000	24.763410
Candidate 2	6.7205	12.4540	9.4673	7		14.729000	9.778723	8.515000	5.803341	9.347000	6.846887	40.859000	26.002300
Candidate 3	6.7205	12.4670	9.4339	7		14.729000	10.861750	8.541000	5.815966	9.542000	6.923053	40.859000	25.472960
Candidate 4	7.0297	9.5490	10.5034	4	Min Delay	11.853000	12.173380	5.427000	5.334890	5.841000	6.132699	38.889000	36.882750
Candidate 5	7.1265	9.5490	10.4476	4		11.853000	12.267640	5.247000	5.083223	5.670000	6.080751	38.259000	36.553450
Candidate 6	6.8533	12.7400	8.5430	7	Min Energy	16.081000	10.734830	8.229000	5.497724	8.333000	6.060592	54.821000	36.191670

Mapping of Function Units among execution regions for the candidate solutions obtained using modified NSGA-II algorithm



	ADD	ADDMOD	AND	COMBINE16	COMBINE8	EXTRACT16	EXTRACT4	EXTRACT8	GFMUL	MUL	MULINV	MUX	NAND	OR	PUSHIMM	READREG	READSBOX	REPLICATE	ROL	ROR	SHL	SHR	SUB	WRITEREG	WRITESBOX	XOR2	XOR3
Candidate 1	3	4	4	0	0	6	6	6	6	3	0	0	4	4	4	5	5	2	1	1	2	5	6	5	5	1	3
Candidate 2	5	4	5	2	2	2	2	2	3	0	1	3	5	5	1	6	6	2	3	4	5	1	4	6	6	0	0
Candidate 3	3	5	3	0	0	2	2	2	6	4	4	0	3	3	4	5	5	2	0	6	2	3	6	5	5	1	1
Candidate 4	0	3	2	1	1	2	2	2	2	0	2	3	2	2	0	3	3	1	2	0	3	1	2	3	3	1	0
Candidate 5	0	3	2	1	1	2	2	2	2	0	2	3	2	2	0	3	3	1	0	0	2	2	2	3	3	1	0
Candidate 6	1	4	2	2	2	4	4	4	5	1	6	2	2	2	5	3	3	3	6	6	6	2	4	3	3	3	0

ARTICLE INFO

Article history:
Received 22 July 2017
Revised 30 October 2017
Accepted 3 November 2017
Available online 4 November 2017

Keywords:
Programmable crypto processor
Data-flow crypto processor
NSGA-II Genetic Algorithm

ABSTRACT

The optimization of a novel programmable data-flow crypto processor dedicated to security applications is considered. An architecture based on assigning basic functional units to four synchronous regions was proposed in a previous work. In this paper, the problem of selecting the number of synchronous regions and the distribution of functional units among these regions is formulated as a combinatorial multi-objective optimization problem. The objective functions are chosen as: the implementation area, the execution delay, and the consumed energy when running the well-known AES algorithm. To solve this problem, a modified version of the Genetic Algorithm - known as NSGA-II - linked to a component database and a processor emulator, has been invoked. It is found that the performance improvement introduced by operating the processor regions at different clocks is offset by the necessary delay

Peer review under responsibility of Cairo University.
* Corresponding author.
E-mail addresses: hadidi@eun.eg, mahmoud.hadidi@gmail.com (M.T. El-Hadidi).

https://doi.org/10.1016/j.jare.2017.11.002

Multi-objective optimization
FPGA implementation
Design space exploration

introduced by wrappers needed to communicate between the asynchronous regions. With a two clock-periods delay, the minimum processor delay of the asynchronous case is 311% of the delay obtained in the synchronous case, and the minimum consumed energy is 308% more in the asynchronous design when compared to its synchronous counterpart. This research also identifies the Instruction Region as the main design bottleneck. For the synchronous case, the Pareto front contains solutions with 4 regions that minimize delay and solutions with 7 regions that minimize area or energy. A minimum-delay design is selected for hardware implementation, and the FPGA version of the optimized processor is tested and correct operation is verified for AES and RC6 encryption/decryption algorithms.

Introduction

As standard cryptographic algorithms – such as DES [1], RSA [2], ECC [3], and AES [4] – were adopted, researchers and technology firms devoted considerable effort and time to develop efficient implementations in software and hardware. Initially, attention was directed to achieving high throughput as well as low cost and/or low power consumption for specific algorithms such as AES (see [5] and the references therein). Considering the fact that cryptography – by its very nature – is always ever-changing, the need for a flexible platform that can implement a wide range of cryptographic primitives, algorithms, and protocols was soon recognized. Since the late 90s, activities concerning the implementation of multiple security algorithms have centred around three main approaches: Customized General Purpose Processor (GPP) [6–12], Crypto Co-processor [13–16], and Crypto Processor [17–21]. While throughput was almost always the prime metric, other figures of merit were sought such as flexibility and security. This trilogy was used for evaluating various proposals, along with the usual design considerations of surface area, cost, and power consumption, [5].

A novel data flow-oriented Crypto Processor based on the Transport Triggered Architecture (TTA) was proposed [22,23]. The architecture comprised Function Units (FUs) which were selected to cover all arithmetic/logic functions typically encountered in encryption/decryption algorithms. A FU would not store its output in a common memory (as is typical with the von Neumann Architecture), but rather it would feed its output directly to one (or two) FUs waiting for such output as an operand. To allow execution of security algorithms in a parallel mode, the FUs are distributed among several Execution Regions (ERs). Each of the ERs, as well as an Instruction Region (IR) and an interconnection region (called Global Interconnection Network – GIN) operates synchronously at its own clock frequency, while regions communicate asynchronously. This gives rise to a Globally Asynchronous Locally Synchronous (GALS) architecture. This architecture allows higher throughput, and the decoupled structure of the GALS units makes it possible to clock gate idle regions, thereby reducing the amount of dissipated power. Finally, the asynchrony of regions, in addition to a novel data scrambling technique can render the processor more immunity against side channel attacks.

In this paper, the above architecture is enhanced and extended by investigating the effect of assigning FUs among arbitrary number of synchronous ERs in order to optimize the processor performance. The optimum design will be based on a trade-off between the primary objective functions of implementation area, execution delay and consumed energy. Thus the problem will essentially be one of multi-objective optimization.

The main contributions of this research is the proposal of a modified version of the Genetic Algorithm - known as NSGA-II - and linking it to a component database to perform design space exploration, building a processor emulator that is invoked to calculate the solutions cost and building estimation models for the design metrics used in the optimization process.

The rest of the paper is organized as follows. Section 2 presents a brief overview of recent work regarding the three main categories of security processors. Section 3 provides an overview of the design of the data-flow crypto processor, while section 4 presents the design objective functions and performance metrics. Then section 5 presents the proposed optimization algorithm, and section 6 gives the optimization results and analyzes their significance. In section 7, the implementation of the optimized processor on FPGA is presented, and finally section 8 concludes the paper.

Security processors: An overview

Historically, security processors were designed as Customized GPPs. These are based on using standard processors (whether CISC or RISC) and adding special functional units to its Arithmetic and Logic Unit (ALU) that cater for cryptographic operations, such as "bit shifting", "bit rotation", "modular addition", "modular multiplication", "substitution", and the like. Since new instructions are introduced to deploy these additional functions, Customized GPP's are also called Instruction Set Extension (ISE) processors. While they offer the highest flexibility – because of their reliance on easily modifiable software instructions – they require modifications in the existing processor hardware which comes at the expense of increased chip area, increased cost, and reduced throughput. To further enhance the flexibility of Customized GPP, reconfigurable designs have been proposed [24,25] and arrays of GPPs have been deployed [26].

On the other hand, Crypto Co-processors attempt to avoid the shortcomings of Customized GPP's by detaching the special cryptographic function units from the ALU of the main processor. They execute cryptographic algorithms or cryptographic protocols on a completely separate co-processor which is either tightly-coupled or loosely coupled to the main-processor. Such co-processors provide hardware implementation of selected cryptographic algorithms or protocols and hence can exhibit a higher throughput than Customized GPP (especially for tightly-coupled implementations). However, they lack the flexibility of Customized GPP's. To partially circumvent this drawback, reconfigurable designs - such as reconfigurable FPGA cores - have been proposed [27–31]. Further enhancements of Crypto Co-processors were realized by deploying several such engines within an array (called Crypto Array) [32,33]. Multi-core versions of Crypto Co-processors were also proposed [34,35]. Still, Crypto Co-processors cannot be easily adapted to the wide range of existing and expected future cryptographic algorithms and protocols.

Consequently, Crypto Processors have been proposed to retain the flexibility of Customized GPP and still approach the higher throughput exhibited by Crypto Co-processors, by implementing the additional cryptographic functional units in a separate tightly-coupled or loosely-coupled processor. The function units are either realized using customized ALUs or systolic arrays. Other designs that exhibited flexible implementation of cryptographic algorithms using Crypto Processors have been also proposed. In one instance, the ALU is replaced by a set of Function Units (FUs) connected to a common bus so as to allow data flow implementations of a cryptographic algorithm [36,37]. It is called Transport Triggered Architecture (TTA), deploys MOVE instructions, and allows improved throughput performance when compared to the classical von Neumann approach. By using a reconfigurable FPGA engine, the type and number of deployed FUs are adapted to various security algorithms.

In a second instance, use of multiple ALUs in the form of an array has been proposed. Specifically, [38] proposed an architecture - called Celator – which is based on using an array of Processing Elements (PEs) where each is capable of performing basic arithmetic operations, logic operations, modular operations, shifting operations and rotation operations. These PEs are controlled by instructions stored in a memory (CRAM), and would be changed depending on the desired cryptographic algorithm. Another architecture – called Cryptoraptor - is proposed which comprises 3 main blocks: State Engine, Register File, and Execution Tile [39]. The Execution Tile consists of 20 stages of Processing Elements (PEs) rows and Connection Rows (CR). Each PE row consists of 4 independent PE's. (The values 20 and 4 are based on intuitive judgment). Memory blocks are located in the State Engine, in each PE and next to each CR. Cryptoraptor has no fetch and decode cycles, and provides a reconfigurable substrate on a non-configurable platform. However, the State Engine is controlled by HW State Machine which is configured at initial step (and remains fixed for each algorithm). This limits the flexibility and ease of use of this design. Moreover, no public-key cryptographic algorithms could be accommodated by Cryptoraptor. In order to improve the performance of Crypto Processors, several cores are grouped together to form Multi-Core-Crypto-Processor (MCCP). This approach seemed to be promising and reconfigurable versions have been developed for it [40,41].

Design background of the novel crypto processor

The architecture of the novel crypto processor proposed previously [22,23] inherits features both from the transport-triggered paradigm and the dataflow paradigm. Thus, instructions control the bypass circuits rather than the FUs; the FU operation is triggered by the presence of its operands; and the results are passed between the FUs instead of returning back to a register file. All FUs can work in parallel and fast FUs do not need to run at the same speed as the slow FUs, which can lead to further improvement in the processor performance.

Fig. 1 shows a block diagram of the processor. The design includes 27 FUs needed in the execution of major encryption/decryption algorithms. Table 1 shows the 27 FUs included in the processor together with the clock period of their ASIC implementation. The FUs are grouped into a number of ERs, each operating at a given clock frequency. Normally an ER will include FUs with close latencies and will operate at a clock period dictated by the slowest FU it includes. Within the ER, an Instruction Switch (IS) is used to forward an incoming instruction to the designated FU. Matched Input Buffers (MIBs) are provided inside an FU to store incoming instructions and operands. Meanwhile, results from an FU can be either sent to FUs inside the ER using a Local Interconnection Network (LIN), or else sent to other FUs in other ERs through the GIN.

The function of the IR is to hold the instructions of the algorithm to be executed, and to dispatch instructions to appropriate ERs. An Instruction Fetcher within IR fetches instructions from the Instruction Cache. To allow regions to work in parallel, each fetch operation can get up to *n* instructions, where *n* is the number of ERs. An Arbiter unit then forwards the fetched instructions to the appropriate ERs, after checking that such ERs are ready to handle a new instruction. To enhance the processor's ability to combat side-channel attacks, the order by which instructions are executed is

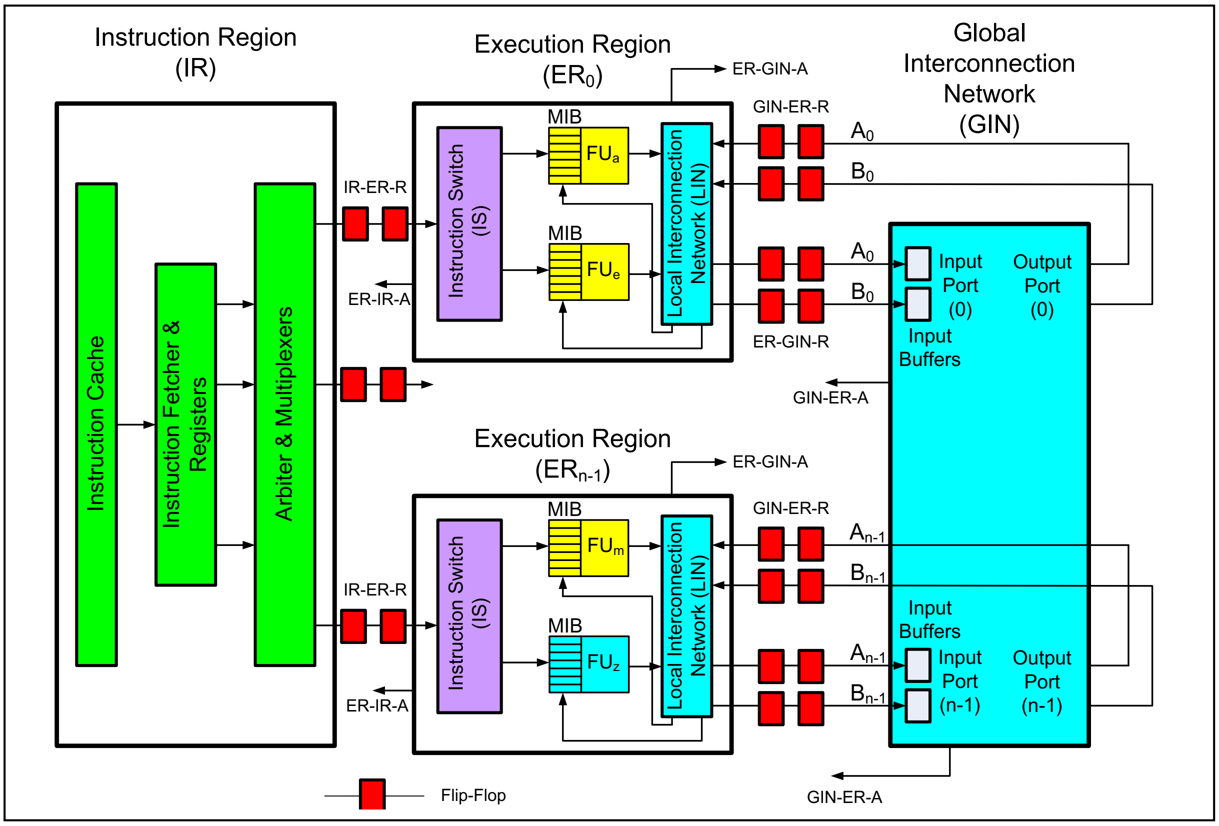


Fig. 1. Block diagram of the security processor [42].

Table 1
Functional units used in the crypto processor with the clock period of their ASIC implementation.

FU no	FU short name	FU operation	Clock Period (ns)
0	ADD	Addition	4
1	ADDMOD	Modular Addition	4
2	AND	Logical AND	3
3	COMBINE 16	Combining two 16-bit numbers	3
4	COMBINE 8	Combining two 8-bit numbers	3
5	EXTRACT 16	Extracting 16-bits from a 32-bit number	4
6	EXTRACT 4	Extracting 4-bits from a 32-bit number	4
7	EXTRACT 8	Extracting 8-bits from a 32-bit number	4
8	GFMUL	Galois Field Multiplication	5
9	MUL	Multiplication	8
10	MULINV	Multiplicative Inverse	3
11	MUX	Multiplexing	3
12	NAND	Logical NAND	3
13	OR	Logical OR	3
14	PUSHIMM	Pushing a number into a register	3
15	READREG	Reading a value from a register	10
16	READSBOX	Reading a value from SBox	10
17	REPLICATE	Replication of a value	3
18	ROL	Rotation to the left	4
19	ROR	Rotation to the right	4
20	SHL	Shifting to the left	4
21	SHR	Shifting to the right	4
22	SUB	Subtraction	4
23	WRITEREG	Writing a value into a register	10
24	WRITESBOX	Writing a value into SBox (not used)	10
25	XOR2	Logical XOR (two inputs)	3
26	XOR3	Logical XOR (three inputs)	3

randomly altered, and a random selection is made among internal signals targeting the same FU.

The GIN is a high speed interconnection network that allows exchange of intermediate results between the ERs. Incoming data to the GIN wait in input buffers and are forwarded through the proper output port to other ERs.

Both the IR and GIN are allowed to have clocks that best suit their operation. According to the GALS paradigm, signalling inside each ER is controlled by a single clock (synchronous operation), but the different ERs may have independent clocks and therefore operate asynchronously. This necessitates the use of wrappers between the various regions. In Fig.1, asynchronous communication uses Request (R)/Acknowledge (A) signals through a double FF mechanism.

Instruction set

Any security algorithm can be encoded in an assembly code written using instructions that have a standardized format. As described in [22,23], each instruction specifies the FU responsible for executing the operation and the FU(s) to which the outcome should be forwarded. One instruction can identify one or two destination FUs. The instruction also specifies the “tag” of the FU outcome. Tags help an FU to link instructions dispatched by IR with operands received from other FUs. Fig. 2 depicts an example for a typical instruction. (Due to lack of space, full details regarding the meaning of fields in Fig. 2, the use of “tags” to bind data to instructions and the generation of instructions for a specific security algorithm cannot be provided here. However, a concise description of some of these concepts is given in Appendices B and C).

The encryption/decryption program instructions are stored in the Instruction Cache of the IR, while the input data (such as plaintext, ciphertext, encryption keys, decryption keys, and Substitution Box values) are stored in the WRITEREG and READSBOX FUs.

Each FU stores data and instructions in its MIBs, where each instruction is matched to corresponding operands according to their tags. When an FU receives the appropriate operands, along with the associated instruction, the FU executes its operation and forwards its outcome to the destination FUs. MIBs allow the results to wait for their instructions and also the instructions to wait for their data to arrive. Compared to the common register file used in TTA architecture, this requires much less address decoding time and buffers can be read and written back in parallel. Typically, the size of FU buffers, specified at the design stage, is limited. It is to be noted that if the FU MIB size is too small, some instructions may be lost, and if the FU MIB size is too large, there will be wasted area in the processor.

Design space

Within the guiding design principles mentioned above, still many design variations are possible. These variations may affect the different performance aspects of the processor and hence need to be considered within the Design Space Exploration (DSE) process.

The number *n* of ERs in the processor is a major design parameter. It will determine the degree to which programs can benefit from parallelism and asynchrony. The number *n* will also determine how many instructions are fetched within the IR which – in turn - determines the complexity of the Instruction Fetcher and the Instruction Arbiter. It will also determine the width of

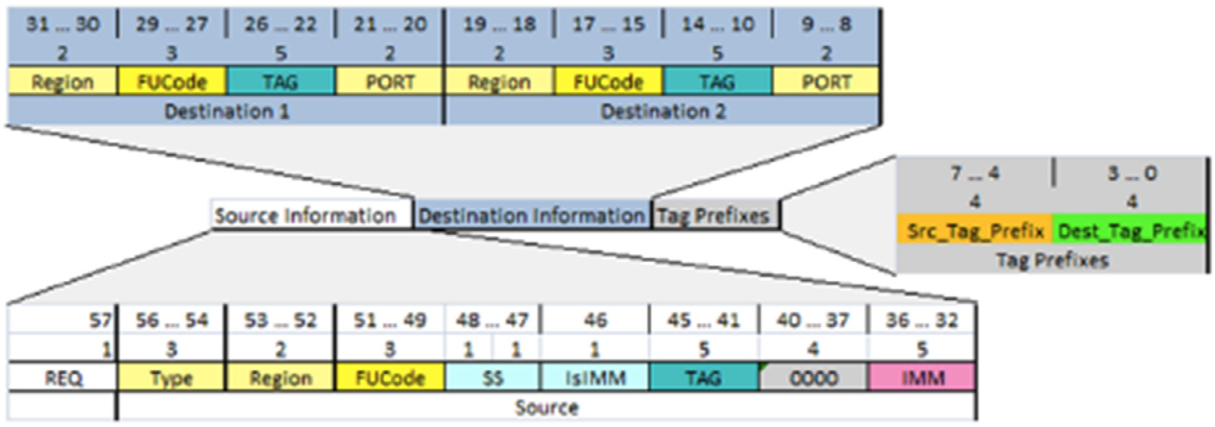


Fig. 2. Format for a typical instruction of the programmable crypto processor [23].

buses for data and control interconnecting IR and ERs, as well as how many ports and input buffers the GIN should deploy. This will have an important impact on processor area and consumed energy.

The manner in which FUs are distributed among ERs is another major design option. The FUs mapped to an ER will operate at the clock frequency dictated by the slowest FU in the ER. This may slow down some FUs and increase the delay of executing some algorithms. Also, the number of FUs within an ER will determine the number of ports of this region's IS and LIN, hence affecting their complexity, area, and energy consumption.

In [23] the distribution of FUs over four regions was heuristically selected based on execution time of FUs, the communication frequency between the execution regions, and the communication frequency within the Instruction Region. However, other performance issues, such as area and energy consumption were not considered. Considering all these performance aspects will result in a multi-objective combinatorial optimization problem with a huge search space (= 20,971,522 points).

Other design options, such as the possible duplication of some FUs, and changing the number of instruction busses and data busses may also be considered but will not be addressed in this paper.

Design objective functions and metrics

The design of embedded processors - such as the considered security processor - needs to comprehensively address different performance aspects [43,44]. Some of these performance measures are major quantifiable properties of the system that are typically used as optimization goals. These are termed the primary objectives of the design. Other performance aspects are domain-specific features that are not easily quantifiable but cause the designer to prefer one of otherwise equal designs. These are termed the secondary objectives of the design.

Speed is the first important primary objective of the design. The speed of a design can be expressed by different metrics such as the throughput achieved for computations or the latency/response time for certain events. The average and/or worst-case latency needed to perform the encryption or decryption algorithm on a block of data is an appropriate performance measure for the crypto processor. This follows since these operations will have to be performed repeatedly in a typical usage of the processor and will affect the overall system performance. Since the Advanced Encryption Standard (AES) – [4] – is the most popular algorithm among all symmetric key cryptographic techniques, it has been chosen as the reference security algorithm for optimizing the programmable data-flow crypto processor.

The latency of operations will be affected by all the design choices mentioned in the previous section. For example, placing a frequently used FU in certain region may force this unit to operate at a clock frequency less than the maximum possible value. This in turn delays the execution of instructions and causes the block encryption process to take a longer time.

The energy consumed by the processor to perform the encryption or decryption process on a block of data must also be emphasized in the design. This is important for embedded battery-operated systems to prolong the battery life-time. Even for high-end systems optimized for speed, one needs to consider the generation of heat within the system that degrades the life-time of the components. Again, the consumed energy will be affected by all the considered design choices.

The third quantifiable primary objective function is the cost. The cost of the design is essentially determined by the area consumption in the target technology and the packaging costs. Fixed costs, such as the fabrication costs of mask sets cannot drive the optimization process and are often not included [43].

Secondary objectives that are typically used to compare designs include utilization of computation and communication resources, I/O and memory specific metrics, and testability of design. All these secondary objectives can be used to evaluate different designs of the security processor, but a major objective will be the immunity to side channel attacks which target the hardware implementation of a cryptosystem.

Area, energy, and delay calculations

The objective functions of processor area, energy consumed in executing the AES encryption algorithm and total delay for executing this algorithm, will be used to compare between solutions in the design space. For a given design, the total area of the processor can be estimated as:

$$\text{Total Area} = A_{IR}(n) + A_{GIN}(n) + \sum_{i=1}^{n} A_{IS}(f_i) + \sum_{i=1}^{n} A_{LIN}(f_i) + \sum_{j=1}^{f} A_{FU}(j). \tag{1}$$

where is the number of ERs, f_i is the number of FUs in ER_i, f is the total number of FUs in the processor, $A_{IR}(n)$ is the area of the IR as function of the number of ERs, A_{GIN} (n) is the area of GIN as function of the number of ERs, A_{IS} (f_i) is the area of IS in ER_i as function of the number of FUs in ER_i, A_{LIN} (f_i) is the area of LIN in ER_i as function of the number of FUs in ER_i, and A_{FU} (j) is the area of FU number j.

The consumed energy can be estimated as:

$$\text{Total Energy} = [P_{IR}(n) + P_{GIN}(n) + \sum_{i=1}^{n} P_{IS}(f_i) + \sum_{i=1}^{n} P_{LIN}(f_i) + \sum_{j=1}^{f} P_{FU}(j)] \times D. \tag{2}$$

where $P_{IR}(n)$ is the power consumed by the IR as function of the number of ERs, $P_{GIN}(n)$ is the power consumed by the GIN as function of the number ERs, P_{IS} (f_i) is the power consumed by the IS in ER_i as function of the number of FUs in ER_i, P_{LIN} (f_i) is the power consumed by the LIN in ER_i as function of the number of FUs in ER_i, P_{FU} (j) is the power consumed by the FU number j, and *D* is the total delay for executing the program.

The values for the power terms in Eq (2) can be expressed in terms of a dynamic component and a static (idle) component, along with the busy period and idle period for each term. Specifically, each power term can be expressed as:

$$P = [(P_d * T_d) + (P_i * T_i)]/D. \tag{3}$$

where P_d. the dynamic power of the term, P_i. the static (idle) power of the term, T_d is the busy period of the term during program execution, and T_i is the idle period of the term during program execution.

The third objective function is:

$$\text{Execution Delay} = D = T_d + T_i. \tag{4}$$

Constructing components database

In order to evaluate the objective functions, it is necessary to know the area, dynamic power, and static power of different components as function of corresponding parameters. These depend on the used circuit designs as well as the technology used to build these circuits. For optimization purposes, a database of the characteristics of different modules using real ASIC technology data has been constructed. With the help of synthesis tools for 130 nm ASIC technology, the VHDL files for the design of the IR, FUs, LIN, IS, and GIN are used to obtain the required data. The clock periods, area,

Table 2
Area of ASIC implementation of IR as function of number of regions.

Number of regions n	$A_{IR}(n)$ (in μm^2)
2	1265264.644446
3	1412147.201183
4	1568757.75622
5	1749327.356845
6	1872148.475496
7	2002606.080944
8	2178954.240621
9	2332172.799922
10	2484849.920043
11	2677141.759073
12	2861539.837971
13	3014935.036752
14	3182263.037729
15	3339948.796589

and power consumption of different components are then used to evaluate equations (1)-(3) for every candidate design.

For example, Table 1 shows the clock periods for the FUs used in the database, while Table 2 shows the area of the IR for different values of n.

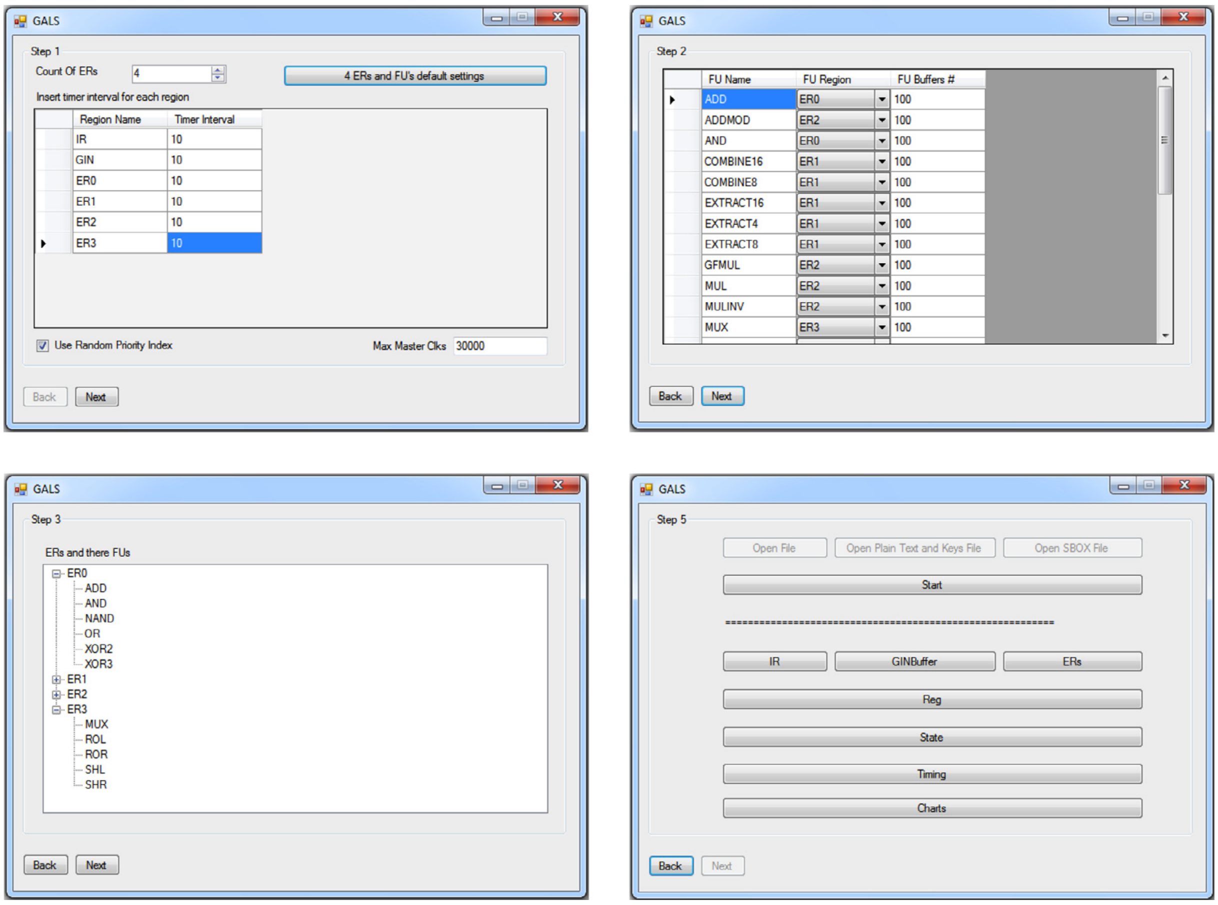
Processor emulator

In order to estimate the execution times, as well as the busy and idle periods of different components in a given configuration, the programmable data-flow crypto processor is emulated using the C# programming language. The following specifications have been targeted during the development of the emulator:

(a) A user friendly interface that facilitates the selection of the processor's design parameters, namely: the number of Execution Regions, the operating frequency for each Execution Region, the mapping of Function Units to Execution Regions, the number of Matching Input Buffers for each Function Unit, and whether the crypto processor runs in the randomized mode or not.
(b) Ability to read in the set of instructions for the security algorithm under test.
(c) Ability to read in the input message and the key chosen for the security algorithm.
(d) Ability to read in S-Box values.
(e) Ability to specify the link delay between any two regions to emulate the effect of wrappers.
(f) Running the set of instructions and keeping the output of execution in a file with suitable format.
(g) Keeping record of execution delay and percentage utilization for all components appearing in Eqs. (1) and (2).

Fig. 3 shows several screen captures of the GUI for the developed software emulator. Meanwhile, sample results are depicted in Fig. 4.

It is to be observed that the logic used by the various hardware modules of the crypto processor has been mimicked when developing the software for the emulator. For example, the determination of the Program Counter values follows the exact logic used by the hardware circuit. Moreover, in order to have a cycle accurate emulation of the crypto processor, the delays experienced by all signals as they propagate through the IR, ER, and the GIN have been inserted into the program.



Count Of ERs: 4

Insert timer interval for each region

Region Name	Timer Interval
IR	10
GIN	10
ER0	10
ER1	10
ER2	10
ER3	10

[✓] Use Random Priority Index

Max Master Clks: 30000

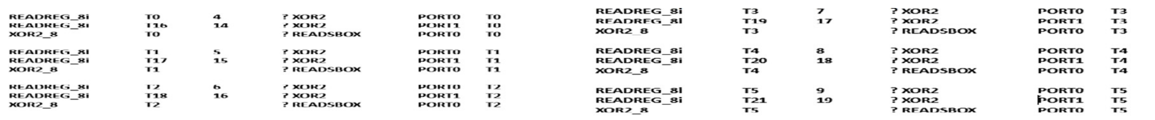
FU Name	FU Region	FU Buffers #
ADD	ER0	100
ADDMOD	ER2	100
AND	ER0	100
COMBINE16	ER1	100
COMBINE8	ER1	100
EXTRACT16	ER1	100
EXTRACT4	ER1	100
EXTRACT8	ER1	100
GFMUL	ER2	100
MUL	ER2	100
MULINV	ER2	100
MUX	ER3	100

ERs and there FUs

```
⊟ ER0
    ADD
    AND
    NAND
    OR
    XOR2
    XOR3
⊞ ER1
⊞ ER2
⊟ ER3
    MUX
    ROL
    ROR
    SHL
    SHR
```

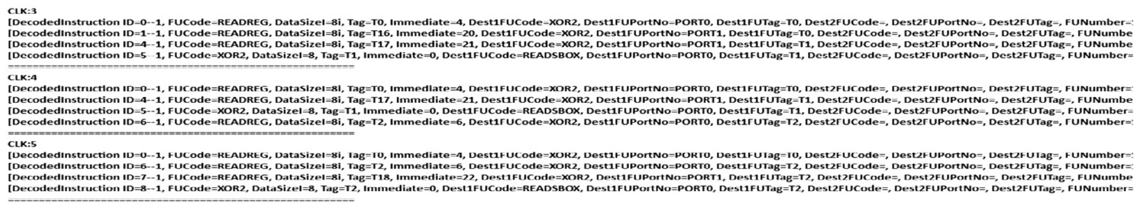
Fig. 3. Screen captures from the GUI of the software emulator developed for the crypto processor.

(a)



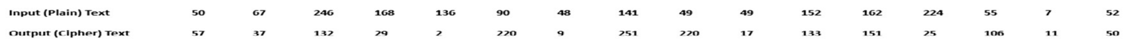
```
READREG_8i    T0     4     ? XOR2        PORT0   T0        READREG_8i    T3     7     ? XOR2        PORT0   T3
READREG_8i    T16    14    ? XOR2        PORT1   T0        READREG_8i    T19    17    ? XOR2        PORT1   T3
XOR2_8        T0           ? READSBOX    PORT0   T0        XOR2_8        T3           ? READSBOX    PORT0   T3

READREG_8i    T1     5     ? XOR2        PORT0   T1        READREG_8i    T4     8     ? XOR2        PORT0   T4
READREG_8i    T17    15    ? XOR2        PORT1   T1        READREG_8i    T20    18    ? XOR2        PORT1   T4
XOR2_8        T1           ? READSBOX    PORT0   T1        XOR2_8        T4           ? READSBOX    PORT0   T4

READREG_8i    T2     6     ? XOR2        PORT0   T2        READREG_8i    T5     9     ? XOR2        PORT0   T5
READREG_8i    T18    16    ? XOR2        PORT1   T2        READREG_8i    T21    19    ? XOR2        PORT1   T5
XOR2_8        T2           ? READSBOX    PORT0   T2        XOR2_8        T5           ? READSBOX    PORT0   T5
```

(b)



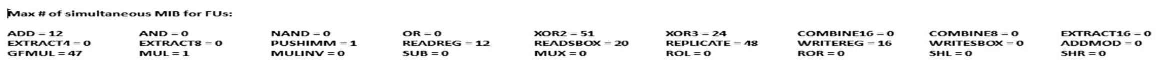
```
CLK:3
[DecodedInstruction ID=0--1, FUCode=READREG, DataSizeI=8i, Tag=T0, Immediate=4, Dest1FUCode=XOR2, Dest1FUPortNo=PORT0, Dest1FUTag=T0, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumber=
[DecodedInstruction ID=1--1, FUCode=READREG, DataSizeI=8i, Tag=T16, Immediate=20, Dest1FUCode=XOR2, Dest1FUPortNo=PORT1, Dest1FUTag=T0, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumbe
[DecodedInstruction ID=4--1, FUCode=READREG, DataSizeI=8i, Tag=T17, Immediate=21, Dest1FUCode=XOR2, Dest1FUPortNo=PORT1, Dest1FUTag=T1, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumbe
[DecodedInstruction ID=5--1, FUCode=XOR2, DataSizeI=8, Tag=T1, Immediate=0, Dest1FUCode=READSBOX, Dest1FUPortNo=PORT0, Dest1FUTag=T1, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumber=
==========================================================
CLK:4
[DecodedInstruction ID=0--1, FUCode=READREG, DataSizeI=8i, Tag=T0, Immediate=4, Dest1FUCode=XOR2, Dest1FUPortNo=PORT0, Dest1FUTag=T0, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumber=
[DecodedInstruction ID=4--1, FUCode=READREG, DataSizeI=8i, Tag=T17, Immediate=21, Dest1FUCode=XOR2, Dest1FUPortNo=PORT1, Dest1FUTag=T1, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumbe
[DecodedInstruction ID=5--1, FUCode=XOR2, DataSizeI=8, Tag=T1, Immediate=0, Dest1FUCode=READSBOX, Dest1FUPortNo=PORT0, Dest1FUTag=T1, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumber=
[DecodedInstruction ID=6--1, FUCode=READREG, DataSizeI=8i, Tag=T2, Immediate=6, Dest1FUCode=XOR2, Dest1FUPortNo=PORT0, Dest1FUTag=T2, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumber=
==========================================================
CLK:5
[DecodedInstruction ID=0--1, FUCode=READREG, DataSizeI=8i, Tag=T0, Immediate=4, Dest1FUCode=XOR2, Dest1FUPortNo=PORT0, Dest1FUTag=T0, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumber=
[DecodedInstruction ID=6--1, FUCode=READREG, DataSizeI=8i, Tag=T2, Immediate=6, Dest1FUCode=XOR2, Dest1FUPortNo=PORT0, Dest1FUTag=T2, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumber=
[DecodedInstruction ID=7--1, FUCode=READREG, DataSizeI=8i, Tag=T18, Immediate=22, Dest1FUCode=XOR2, Dest1FUPortNo=PORT1, Dest1FUTag=T2, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumbe
[DecodedInstruction ID=8--1, FUCode=XOR2, DataSizeI=8, Tag=T2, Immediate=0, Dest1FUCode=READSBOX, Dest1FUPortNo=PORT0, Dest1FUTag=T2, Dest2FUCode=, Dest2FUPortNo=, Dest2FUTag=, FUNumber=
==========================================================
```

(c)



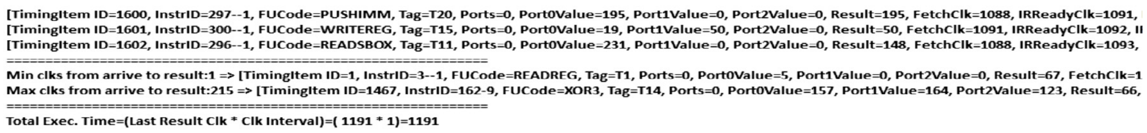
```
Input (Plain) Text      50   67   246   168   136   90    48   141   49    49   152   162   224   55    7    52

Output (Cipher) Text    57   37   132   29    2     220   9    251   220   17   133   151   25    106   11   50
```

(d)



```
Max # of simultaneous MIB for FUs:

ADD = 12        AND = 0         NAND = 0        OR = 0          XOR2 = 51       XOR3 = 24       COMBINE16 = 0   COMBINE8 = 0    EXTRACT16 = 0
EXTRACT4 = 0    EXTRACT8 = 0    PUSHIMM = 1     READREG = 12    READSBOX = 20   REPLICATE = 48  WRITEREG = 16   WRITESBOX = 0   ADDMOD = 0
GFMUL = 47      MUL = 1         MULINV = 0      SUB = 0         MUX = 0         ROL = 0         ROR = 0         SHL = 0         SHR = 0
```

(e)



```
[TimingItem ID=1600, InstrID=297--1, FUCode=PUSHIMM, Tag=T20, Ports=0, Port0Value=195, Port1Value=0, Port2Value=0, Result=195, FetchClk=1088, IRReadyClk=1091,
[TimingItem ID=1601, InstrID=300--1, FUCode=WRITEREG, Tag=T15, Ports=0, Port0Value=19, Port1Value=50, Port2Value=0, Result=50, FetchClk=1091, IRReadyClk=1092, II
[TimingItem ID=1602, InstrID=296--1, FUCode=READSBOX, Tag=T11, Ports=0, Port0Value=231, Port1Value=0, Port2Value=0, Result=148, FetchClk=1088, IRReadyClk=1093,
==============================================================
Min clks from arrive to result:1 => [TimingItem ID=1, InstrID=3--1, FUCode=READREG, Tag=T1, Ports=0, Port0Value=5, Port1Value=0, Port2Value=0, Result=67, FetchClk=1
Max clks from arrive to result:215 => [TimingItem ID=1467, InstrID=162-9, FUCode=XOR3, Tag=T14, Ports=0, Port0Value=157, Port1Value=164, Port2Value=123, Result=66,
==============================================================
Total Exec. Time=(Last Result Clk * Clk Interval)=( 1191 * 1)=1191
```

(f)

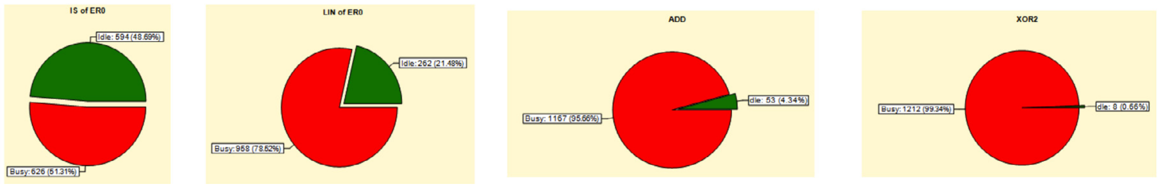


Fig. 4. Sample results from the software emulator developed for the crypto processor: (a) Part of the Assembly Code for the AES Encryption Algorithm, (b) Fetched instructions (4 per clock), (c) Input "Plain" text and Output "Cipher" text, (d) Max number of simultaneously used FUs, (e) Computation of the "Total Delay Time", (f) Percentage Utilization for some modules of the processor.

It is to be noticed that the emulator software has been modified to allow its integration with the NSGA-II algorithm. This necessitates that the emulator reads text files with specified formats describing the input parameters, and then outputs the results in text files of appropriate formats.

Optimization algorithm

The multi-objective optimization problem

Generally, it is not expected that a single design minimizes all the objective functions simultaneously. In fact, these objectives may conflict with each other and a solution that minimizes one objective may result in unacceptable values for other objectives. A trade-off between objectives is thus necessary. One way to handle multi-objective optimization is to combine objective functions into a single composite function, e.g. a weighted sum or product. The problem with this approach is that a designer has to determine a priori the weight given to each objective. It is usually very difficult to quantify the desired trade-off, in terms of weights that are assigned to objectives having different dimensions and units. Also, in this approach a single solution is obtained without giving the designer a view about the range of possible trade-off solutions.

Another approach is to handle one objective function and to treat the value of other objectives as constraints in the optimization problem, e.g. minimize the power consumption under a delay constraint or vice versa. Again, it is usually difficult to determine the bounds a priori and repeating the solution for different values is often necessary.

An alternative approach used in most recent works involving DSE is to use the concepts of dominance and Pareto solutions, which originate from the area of economics and game theory. In this approach, a set of solutions representing the possible range of trade-offs is obtained. In the following, we give a definition of Pareto solutions [43,45].

Given k objective functions to be minimized (e.g. in our design problem area, delay and energy) and two design points A and B with objective values $(a_1, a_2, \ldots a_k)$ and $(b_1, b_2, \ldots b_k)$, solution A is said to dominate B (denoted by A≺B) if and only if

$$\begin{aligned} & a_i \leqslant b_i \quad \textit{for } 1 \leqslant i \leqslant k \\ \text{and} \quad & \exists\, j \textit{ such that } a_j < b_j \end{aligned} \tag{5}$$

That is, solution A is better for at least one objective, while being at least the same for all other objectives. A solution which is not dominated by any other solution is said to be a Pareto optimal solution. A Pareto solution, which represents one possible compromise solution, is a point where one cannot improve one objective without degrading at least one other objective. One seeks to obtain the set of Pareto solutions (or the Pareto front) for the problem of crypto processor design.

Design space exploration using Genetic Algorithms

Exhaustive evaluation of every design point is prohibitive in problems with huge design spaces such as the design problem under consideration. Different approaches are used for multi-objective optimization in the context of DSE [43–45]. One particularly successful approach is to use evolutionary approaches and in particular the Genetic Algorithm (GA) [46].

The GA method is a general method that can be applied without particular requirements in the characteristics of the search space. It incorporates knowledge of the design space acquired gradually through iterations, which results in faster convergence toward desired solutions. GA method evaluates a number of solutions rather than a single solution - in each iteration - and thus it could be readily modified to obtain Pareto fronts for multi-objective problems. In this paper, the solution of the optimization problem is based on the Non-dominated Sorting Genetic Algorithm-Type II (NSGA-II) [47].

Multi-objective GA and NSGA-II algorithm

The aim of the NSGA-II algorithm is to obtain a good estimation of the Pareto front of a multi-objective problem through a genetic optimization process. It finds an evenly distributed range of solutions along the Pareto front by combining GA with the Non-dominated Sorting algorithm and the Crowding Distance calculations.

NSGA-II sorts a given population based on non-domination into a number of fronts. The first front is the non-dominated set in the current population and the second front is dominated only by individuals in the first front and so on. Each individual is assigned a rank (fitness) value based on front on which it lies.

The crowding distance is a measure of how an individual is close to its neighbours in the objective functions space. It assigns a further fitness value for each solution, which is higher as the solution is farther from its neighbours. Selecting solutions with larger crowding distance results in a better diversity of outcomes, and thus the obtained solutions are evenly spaced along the Pareto front.

Offspring individuals are obtained by first selecting parents from the current population. Parents are chosen using binary tournament selection based on both the rank and the crowding distance. Thus, two random individuals are compared and the one with lower rank is selected. If ranks of the two solutions are the same, the one with higher crowding distance is selected. Traditional crossover and mutation are next applied on the selected individuals to generate a child population. Giving better solutions a higher probability of being selected for breeding allows keeping good solution attributes and results in faster convergence of the algorithm.

```
Set algorithm parameters
Generate a random population of size N
for each individual in population do
        Assign rank based on Pareto fast non-dominated sort
        Calculate crowding distance
end for
for (i=1 to Max_generations) do
        Select parents from population
        Apply crossover and mutation to obtain child population of size N
        Combine Parent and Child population into population of size 2N
        for each individual in Combined population do
                Assign rank based on Pareto fast non-dominated sort
                Calculate crowding distance
        end for
        Generate population of size N from best ranked solutions
end for
Present results
```

Fig. 5. Pseudo-code of the NSGA-II algorithm.

Individuals of the current and child population are combined and sorted again based on non-domination and crowding distance. Best solutions from both populations are chosen as the next generation. Thus, best individuals in a population are given a chance to be directly carried over to the next generation (in GA literature this is referred to as *elitism*). Thus, a good solution found early on in the run will never be lost unless a better solution is discovered. This is repeated until some stopping criterion is attained. Fig. 5 shows the pseudo-code of the NSGA-II algorithm. Further details can be found in [47].

Modifications of the NSGA-II algorithm

The problem of selecting the best distribution of FUs among ERs falls within the class of grouping problems, for which special solution encodings and genetic operators are needed [48]. The encoding and genetic operators used in the processor optimization programs are based on a modified version of those used in Refs [49,50].

A solution is represented essentially by a binary string with f fields of f bits each, where f is the number of functional units in the design (currently = 27). Each field corresponds to one possible region, implies a maximum of f regions. The ith bit in the jth field is set to 1 if the ith FU is placed in the jth region and is 0 otherwise (Fig. (6)).

For efficiency, this string is actually stored and handled in program as a string of integers. The initial population is generated randomly, i.e. individual solutions have random distribution of FUs on regions. Generation of initial population thus takes relatively insignificant time. When a random initial population is generated, a minimum and maximum number of regions are specified (e.g. 3 to 10 regions). A number of individuals is generated for each size between these limits to cover a large range of the solution space. For a random solution with n regions, the region in which each FU is placed is selected at random in the range from 0 to n−1.

The genetic operator applied on selected solutions is either:

- Move a functional unit selected at random from its region to another random region.
- Swap two random functional units among two different regions.

Further, one of these operators is chosen to generate an offspring from each selected solution with a specific user selection probability. Thus, GA is used in combination with local search, which has been reported to be efficient in a number of similar problems.

Suppl. Fig. 1 shows the pseudo-code for the modified NSGA-II algorithm, while Suppl. Fig. 2 shows a Flowchart for the evaluation of the objective functions for a generation of solutions.

Optimization results

Each run of the optimization algorithm starts with a random population and uses a population size of 100 individuals and runs for 200 generations. It is also to be noted that efficient hardware design dictates the placement of some related FUs within the same region. These constraints are added to the optimization process. In particular, FU for logic functions (2, 12, and 13) should be in the same region. Similarly, Memory-related FUs (15, 16, 23, and 24),

FU0	FU1	FU2	FU3		FU0	FU1	FU2	FU3			FU0	FU1	FU2	FU3	
1	0	1	0	......	0	0	0	1	......		0	0	0	0	.....
Region 0					Region 1					....	Region 26				

Fig. 6. Binary encoding of solution.

COMBINE FUs (3 and 4), and EXTRACT FUs (5, 6, and 7) are placed in the same regions. These constraints have been enforced by the program in all individuals of the generated populations.

Optimization results for AES on synchronous processor

The optimization algorithm is applied on the processor architecture executing the AES encryption algorithm, with all the regions running at the same clock frequency. Synchronous operation simplifies design as it does not require special communication mechanisms between regions operating at different clocks. Suppl. Figs. 3 and 4 show sample results for the simulation at three stages: Generation # 1, Generation # 50, and Generation # 200.

It is clear from Suppl. Figs. 3 and 4 that:

- At Generation # 1, processor area ranges between 6.7 mm^2 and 8.0 mm^2, its delay performance ranges between 10.0 μs and 22.5 μs, and its energy consumption ranges between 9.0 μJ and 18.0 μJ. When reaching Generation # 50, range of variation narrows for all objective functions. Starting from Generation #200 and thereafter, the range of variation stabilizes at a still narrower window, where area ranges between 6.7 mm^2 and 7.5 mm^2, delay performance ranges between 9.5 μs and13.0 μs, and energy consumption ranges between 8.0 μJ and11.0 μJ.
- At Generation # 1, the population has number of execution region ranging between 3 and 10. When reaching Generation # 50, the range for the number of execution regions is reduced and becomes between 4 and 8. Starting from Generation #200 and thereafter, the range for the number of execution regions stabilizes at four values, namely: 4 (delay is minimized), 7 (area is minimized or energy is minimized), and 5 or 8 (neither area nor delay nor energy are minimized, but these candidate solutions are on the Pareto Front implying no other points in the design space dominate them).

Synchronous vs. asynchronous operation of the processor

In this section, the asynchronous operation of the proposed architecture is compared with the synchronous operation, [50]. In a number of previous works [51,52], the advantages of GALS architectures including the avoidance of clock distribution problems and the possible reuse of IP components that have independent clock requirements were shown. However, the asynchronous communication between regions using mechanisms such as FIFOs or wrappers could incur delay and power overheads that may offset the benefits obtained by using various clocks. This can cause the synchronous design to outperform the asynchronous design in delay and power consumption. In this section, this phenomenon is studied as applied to the proposed architecture by comparing the optimal performance measures in the cases of asynchronous and synchronous operation when executing the AES encryption algorithm.

To evaluate the effect of the delay of the links between the processor regions through the wrappers, separate optimization runs are made for the cases of zero, one, and two clock link delay. Signals for which link delay is introduced are request/acknowledge signals between IR and ERs, and request/acknowledge signals between ERs and ports of GIN. Each of these delays has been put as 0, 1 or 2 clock periods.

Also, since the Instruction Region (IR) could represent a bottleneck for the processor operation, the effect of speeding up the IR is considered. Thus, for each of the above cases, the optimization process is repeated assuming that the IR clock period is reduced to nearly 50% and 30% of its original value (integer values are used).

Suppl. Fig. 5 shows the minimum delay obtained on the Pareto front in each of the considered cases, compared with that obtained with synchronous operation. Optimization results show that if asynchronous design uses zero link delay (as in the synchronous case), minimum delay for both asynchronous design with slow IR and synchronous design is obtained by arranging the units into 4 ERs. The minimum delay for asynchronous design is slightly lower than the synchronous case. By speeding up the IR however, asynchronous design benefits from regions that can run at higher clocks so that when IR clock period is reduced to 30% of its original value, delay becomes 56% of the execution delay in the synchronous case. However, as link delay increases, its overhead causes the delay of the asynchronous design to increase by a large factor. Thus, with two clock periods link delay and original IR speed, execution delay increases to 311% of the delay in the synchronous case. Speeding up the IR improves the asynchronous processor delay, but it remains higher than the synchronous processor.

Suppl. Fig. 6 shows the minimum energy of solutions on the Pareto front for different values of link delay and IR speed. Again, asynchronous case is better than the synchronous case for zero link delay and faster IR. For 50% IR clock period the energy is 83% of that in the synchronous case, and for 30% clock period it becomes 81% of the energy in the synchronous case. As link delays increase, the asynchronous case consumes more energy as a result of increased delay and region activities. At original IR speed, for 1 and 2 clocks link delay, the energy becomes 201% and 308% - respectively - compared to the energy in the synchronous case.

Selecting a processor configuration for implementation

Multi-objective optimization problems typically have multiple solutions, where each would behave well for one or more performance measures, but none would behave well for all measures. In order to reduce the candidate solutions to a small set, the AES algorithm has been chosen to be the basis for optimization of the programmable data-flow crypto processor. To further reduce the set of candidate optimal solutions, it is proposed to differentiate between members of the optimal solutions for the AES algorithm, based on the metrics obtained when executing other security algorithms. Earlier results following this approach have been reported before [53]. However, more refined results are presented in this section by using an updated component database.

In general, the cardinality of the obtained Pareto front gave a wide range of design choices. For example, in the case of synchronous design the final population of 100 solutions contained 75 distinct Pareto non-dominated solutions. However, the following method is used to identify the major candidate solutions considering each of the objective functions.

Step 1: Identify the set of points yielding minimum area or minimum delay or minimum energy when applying the NSGA-II and using the instruction sets for AES.

Step 2: Remove candidate solutions that appear more than once. Table 3 depicts the outcome of this step which shows three candidates with minimum area (1, 2 and 3), two candidates with minimum delay (4 and 5), and one candidate with minimum energy (6). The number of regions for these candidates is, respectively; 7, 4, and 7. Table 4 depicts the mapping of FUs for the six candidates.

Step 3: Deduce the metrics for the candidates obtained in Step 2 when running security algorithms other than AES encryption. The following four algorithms have been considered using the FUs' mapping shown in Table 4: AES Decryption, RC6 Encryption, RC6 Decryption, and SHA3 Hashing. Using Eqs. (1)-(4), along with the emulator output, values for area, delay and energy have been deduced. The resulting metrics for the six candidates when using the aforementioned security algorithms are listed in Table 3.

Step 4: Analyze the metrics of the six candidates. The following observations can be made:

Table 3
Results for the six candidate solutions on the Pareto front for the synchronous processor design.

Security Algorithm	AES Encryption			# of Regions		AES Decryption		RC6 Encryption		RC6 Decryption		SHA3 Hashing	
Objective Function	Area	Delay	Energy			Delay	Energy	Delay	Energy	Delay	Energy	Delay	Energy
	mm2	Micro Sec	Micro Joule			Micro Sec	Micro Joule	Micro Sec	Micro Joule	Micro Sec	Micro Joule	Micro Sec	Micro Joule
Candidate 1	6.7205	12.3890	9.6782	7	**Min Area**	14.742000	10.908170	8.073000	5.424328	8.749000	6.394790	41.379000	24.763410
Candidate 2	6.7205	12.4540	9.4673	7		14.729000	9.778723	8.515000	5.803341	9.347000	6.846887	40.859000	26.002300
Candidate 3	6.7205	12.4670	9.4339	7		14.729000	10.861750	8.541000	5.815966	9.542000	6.923053	40.859000	25.472960
Candidate 4	7.0297	9.5490	10.5034	4	**Min Delay**	11.853000	12.173380	5.427000	5.334890	5.841000	6.132699	38.889000	36.882750
Candidate 5	7.1265	9.5490	10.4476	4		11.853000	12.267640	5.247000	5.083223	5.670000	6.080751	38.259000	36.553450
Candidate 6	6.8533	12.7400	8.5430	7	**Min Energy**	16.081000	10.734830	8.229000	5.497724	8.333000	6.060592	54.821000	36.191670

Table 4
Assignment of FUs to various regions for the 6 candidates points in the search space.

	ADD	ADDMOD	AND	COMBINE16	COMBINE8	EXTRACT16	EXTRACT4	EXTRACT8	GFMUL	MUL	MULINV	MUX	NAND	OR	PUSHIMM	READREG	READSBOX	REPLICATE	ROL	ROR	SHL	SHR	SUB	WRITEREG	WRITESBOX	XOR2	XOR3
Candidate 1	3	4	4	0	0	6	6	6	6	3	0	0	4	4	4	5	5	2	1	1	2	5	6	5	5	1	3
Candidate 2	5	4	5	2	2	2	2	2	3	0	1	3	5	5	1	6	6	2	3	4	5	1	4	6	6	0	0
Candidate 3	3	5	3	0	0	2	2	2	6	4	4	0	3	3	4	5	5	2	0	6	2	3	6	5	5	1	1
Candidate 4	0	3	2	1	1	2	2	2	2	0	2	3	2	2	0	3	3	1	2	0	3	1	2	3	3	1	0
Candidate 5	0	3	2	1	1	2	2	2	2	0	2	3	2	2	0	3	3	1	0	0	2	2	2	3	3	1	0
Candidate 6	1	4	2	2	2	4	4	4	5	1	6	2	2	2	5	3	3	3	6	6	6	2	4	3	3	3	0

- Among the three candidates yielding minimum area for AES Encryption, Candidate 1 has smallest delay for AES Encryption, smallest delay and energy for RC6 Encryption, smallest energy for RC6 Decryption, and smallest energy for SHA3. Meanwhile, Candidate 2 has smallest delay and energy for AES Decryption and smallest delay for SHA3. On the other hand, Candidate 3 has smallest energy for AES Encryption, smallest delay for AES Decryption, and smallest delay for SHA3. Consequently, one may rank Candidate 1 as the best - followed by both Candidates 2 and 3 - among the three candidates exhibiting minimum area for AES Encryption.
- Among the two candidates yielding minimum delay, Candidate 5 has smaller energy for AES Encryption, smaller delay and energy for RC6 Encryption, smaller delay and energy for RC6 Decryption, and smaller delay and energy for SHA3. Candidate 4 has smaller area for AES Encryption and smaller energy for AES Decryption. Consequently, one may rank Candidate 5 as the best – followed by Candidate 4 – among the two candidates exhibiting minimum delay for AES Encryption.
- Candidate 6 has lowest energy among all 6 Candidates for AES Encryption, AES Decryption and RC6 Decryption.

For the purpose of FPGA implementation and testing – presented in the next section - it is decided to give priority to delay, and hence candidate 5 has beens selected.

FPGA implementation

To validate the functionality of the deduced architecture in hardware, the entire design is written in VHDL and is implemented on a Xilinx Virtex 6 FPGA. The FPGA of choice is XC6VLX240T package FF1156 speed grade −1, as found in the Xilinx evaluation board ML605. Verification of functionality and performance is ensured in a variety of ways. First, bit matching is confirmed between the software emulator and the results of behavioral simulation for the VHDL. Second, bit and cycle matching is confirmed between behavioral and post-synthesis simulation using the clock period indicated by the critical path in the synthesis report. This ensures that the VHDL is properly written and free of synthesis unfriendly syntax. It also confirms that the critical path produced is true.

As shown in Table 5, the highest resource utilization is in slices used as logic, which is a result of optimization of MIB sizes. Slices used as registers are almost exclusively used to provide storage in the MIBs, as well as a minor component going to pipelining and state registers. RAM/FIFO unit utilization is very small and is used entirely as BRAM to store the program, data, and SBOX tables. DSP48E1 slices provide the ability to use high speed multipliers while at the same time freeing slice logic to be used for other operations. The design uses only three DSP units in the multiplicative FUs.

Table 5
Resource utilization results obtained for XC6VLX240T package FF1156 speed grade -1using Xilinx ISE Design Suite 14.5.

Resource	Utilization
Number of slices used as registers	66,000
Number of slices used as logic	153,000
Number of RAM/FIFO units used	10
Number of DSP48E1	3

The overall design includes the crypto processor plus a Xilinx clock manager unit used to generate the appropriate clock from the on-board 66 MHz clock. The design includes all functional units, whether or not they are utilized by the algorithms under investigation, thus inflating the resource utilization but ensuring full flexibility. To ensure the design fits on the target FPGA, the size of matched input buffers (the memory component of the content addressable memory) is optimized per functional unit, ensuring enough but not excessive entries are included.

Translation, mapping, and placement and routing are performed on the synthesis results with the addition of a chipscope component used to probe certain signals from the design for verification purposes. The constraint file demands a clock that fits the critical path and assigns pin locations so they can be probed in the hardware setup. Although the FPGA is almost entirely utilized by the design, judicious choices of constraints and optimization effort lead to a timing closure and successful generation of bit file. A clock speed of 13 MHz is attained with moderate or high optimization effort. This translates into a bit rate of 1.66 Mbps for a single processor running AES encryption.

The bit file is programmed to the target FPGA. The program and input data are first fed to the storage memories through a test mode setting. Functionality is confirmed in two ways. First, on-chip assertion is performed through a hardware test bench that performs bit matching on the plain text and encryption results. Secondly, a logic analyzer attached to specialty software and interface allows real-time display of signals from the chipscope.

Suppl. Fig. 7 shows the results of running the AES encryption program on the FPGA-implemented crypto processor. The resulting waveforms are screen captured from logic analyzer Model Agilent 16851A as fed from the hardware through the chipscope module. In Suppl. Fig. 7, signals ConfigDataOut_0_OBUF through ConfigDataOut_10_OBUF show part of the data memory output bus carrying the results of encryption. This result is for a case where plain text is 32 43 F6 A8 88 5A 30 8D 31 31 98 A2 E0 37 07 34, the key is 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C and the expected (and obtained) result is 39 25 84 1D 02 DC 09 FB DC 11 85 97 19 6A 0B 32. AES encryption bits match the results of the software emulator and – in addition - the number of cycles in the FPGA implementation matches those obtained from the emulator. Another test is carried out which involves the encryption/decryption of a stream of blocks, and this passed successfully.

Suppl. Figs. 8 and 9 show the results for RC6 encryption and decryption, respectively. Running conditions are identical to those for testing the AES program. Again, bit-matching is achieved with the software emulator and cycle-matching agrees with the behavioral simulation. Functionality is confirmed in processing several consecutive blocks and when switching between encryption and decryption.

Conclusions

In this paper, the optimization of a novel programmable data-flow crypto processor dedicated to security applications has been considered. Its architecture is based on distributing function units needed for executing security algorithms over a number of execution regions that operate in parallel. Processor optimization is formulated as a combinatorial multi-objective optimization problem with the objective functions being area, delay and consumed energy. The evaluation of objective functions relies on a database of component specifications as well as a cycle-accurate emulation of the processor. The optimization problem is solved using a modified version of the NSGA-II algorithm.

The optimization program, coupled with the emulator and the component database provides a tool that allows the exploration of the design space and the study of the impact of different architectural choices and parameters. It is found that the performance improvement introduced by operating the processor regions at different clocks is offset by the necessary delay introduced by wrappers needed to communicate between the asynchronous regions. With a two clock-periods delay, the minimum processor delay of the asynchronous case is 311% of the delay obtained in the synchronous case, and the minimum consumed energy is 308% more in the asynchronous design when compared to its synchronous counterpart. The Instruction Region has been also identified as a major design bottleneck. For the synchronous case, the Pareto front contains solutions with 4 regions that minimize delay and solutions with 7 regions that minimize area or energy. A minimum-delay design is selected for hardware implementation, and the FPGA version of the optimized processor is tested and correct operation is verified for AES and RC6 encryption/decryption algorithms.

The ASIC implementation of the optimized programmable data-flow crypto processor, as well as the redesign of different performance bottlenecks, are the subject of future extensions of this work.

Conflict of interest

The authors have declared no conflict of interest.

Compliance with Ethics Requirements

This article does not contain any studies with human or animal subjects.

Acknowledgments

This work has been funded by the National Telecommunication Regulatory Authority (NTRA) of Egypt. The authors would like to express their deep appreciation for the kind support and encouragement extended to them by NTRA.

Appendix A–C. Supplementary material

Supplementary data associated with this article can be found, in the online version, at https://doi.org/10.1016/j.jare.2017.11.002.

References

[1] National Bureau of Standards. Data Encryption Standard, FIPS-Pub. 46. U.S. Department of Commerce January 1977.
[2] Rivest R, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM February 1978;21(2): 120–6.
[3] Koblitz N. Elliptic curve cryptosystems. Mathematics of Computation January 1987;48(177): 203–9.
[4] NIST. Advanced Encryption Standard, FIPS-Pub 197 26 November 26 2001.
[5] Bossuet L, Grand M, Gaspar L, Fischer V, Gogniat G. Architectures of flexible symmetric key crypto engines - A survey from hardware coprocessor to multi-crypto-processor system on chip. ACM Computing Surveys August 2013; 45(4) Article 41.
[6] Shi Z, Lee R. Bit permutation instructions for accelerating software cryptography. Proc IEEE Int Conf Appl-specific Syst, Arch Processors July 2000;10–12:138–48.
[7] Ravi S, Raghunathan A, Potlapally N, Sankardass M. System design methodologies for a wireless security processing platform. In: Proceedings 39th Annual Design Automation Conference (DAC'02) 2002: 777–82.
[8] Tillich S, Großschädl J, Szekely A. An instruction set extension for fast and memory-efficient AES implementation. CMS 2005, LNCS 3677 2005: 11–21.
[9] Großschädl J, Tillich S, Szekely A. Performance evaluation of instruction set extensions for long integer modular arithmetic on a SPARC V8 processor. Proceedings 10th Euromicro Conference on Digital System Design (DSD 2007) 2007: 680–9.
[10] Jenkins C, Mamidi S, Schulte M, Glossner J. Instruction set extensions for the advanced encryption standard on a multithreaded software defined radio platform. Int J High Perform Syst Arch 2010;2(2–3):203–14.

[11] Gueron S. Intel® advanced encryption standard (AES) new instructions set. White paper: Intel Mobility group. Israel Development Centre, Intel Corporation; 2012.
[12] Benhadjyoussef N, Elhadjyoussef W, Machhout M, Tourki R. Enhancing a 32-bit processor core with efficient cryptographic instructions. J Circ, Syst, Comp 2015;24(10).
[13] Kim HW, Lee S. Design and implementation of a private and public key crypto processor and its application to a security system. IEEE Trans Consumer Electronics February 2004;50(1): 214–24.
[14] Hodjat A, Verbauwhede I. High-throughput programmable cryptocoprocessor. IEEE Micro May-June 2004:34–45.
[15] IBM 4765 PCIe cryptographic coprocessor data sheet, IBM Corporation, 2011.
[16] IBM 4767–002 PCIe cryptographic coprocessor (HSM) data sheet, IBM Corporation, 2016.
[17] Wu L, Weaver C, Austin T. CryptoManiac- a fast flexible architecture for secure communication. In: Proceedings 28th Annual International Symposium on Computer Architecture 30 June – 4 July 2001; 110–9.
[18] Buchty R, Heintze N, Oliva D. Cryptonite – a programmable crypto processor architecture for high-bandwidth applications. In: Proceedings International Conference on Architecture of Computing Systems (ARCS 2004) 2004; 184–98.
[19] Arora D, Raghunathan A, Ravi S, Sankaradass M, Jha NK, Chakradhar ST. Software architecture exploration for high-performance security processing on a multiprocessor mobile SoC. In: Proceedings 43rd Annual Design Automation Conference July 24–28 2006; 496–501.
[20] Han L, Han J, Zeng X, Lu R, Zhao J. A programmable security processor for cryptography algorithms. In: Proceedings 9th International Conference on Solid-State and Integrated-Circuit Technology (ICSICT 2008) 20–23 Oct. 2008; 2144–7.
[21] Li C, Jiang Y, Su D, Xu Y, Luo Z. A new design of low cost security coprocessor for portable electronic devices. In: Proceedings 2010 International Conference on Communications and Mobile Computing (CMC2010) April 2010; 12–4.
[22] Farouk H, El-Hadidi MT, Abou El Farag A. GALS-based LPSP: Implementation of a novel architecture for low power high performance security processors. In: Proceedings 25th IEEE International Parallel and Distributed Processing Symposium 16–20 May 2011; 542–50.
[23] Farouk H, El-Hadidi M, Abou El-Farag A. GALS-based LPSP: performance analysis of a novel architecture for low power high performance security processors. Int J Netw Comput 2012;2(1):56–78.
[24] Barat F, Lauwereins R. Reconfigurable instruction set processors: a survey. In: Proceedings 11th International Workshop on Rapid System Prototyping (RSP 2000) 21–23 June 2000; 168–73.
[25] Grabher P, Großschädl J, Hoerder S, Järvinen K, Page D, Tillich S, Wójcik M. An exploration of mechanisms for dynamic cryptographic instruction set extension. J Cryptogr Eng 2010;2:1–18.
[26] Elbirt AJ, Paar C. Instruction-level distributed processing for symmetric-key cryptography. In: IEEE Transactions on Parallel and Distributed Systems May 2005; 16(5); 468–80.
[27] Taylor RR, Goldstein SC. A high-performance flexible architecture for cryptography. In: Proceedings Workshop on Cryptographic Hardware and Embedded Systems (CHES1999) August 1999; 231–45.
[28] Dandalis A, Prasanna VK. An adaptive cryptographic engine for internet protocol security architectures. ACM Trans Des Automation Electronic Syst. July 2004;9(3); 333–53.
[29] Gonzalez I, Gomez-Arribas FJ. Ciphering algorithms in Microblaze-based embedded systems. IEE Proc-Comput. Digit Tech March 2006;153(2); 87–92.
[30] Sun K, Pan X, Wang J, Wang J. Design of a novel asynchronous reconfigurable architecture for cryptographic applications. In: Proceedings First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06) 20–24 June 2006; 751–7.
[31] Ni S, Dou Y, Chen K, Deng L. A novel design of flexible crypto coprocessor and its application. In: Wu J, Chen H, Wang X, editors. Advanced computer architecture. communications in computer and information science 2014; 451: 128–139.
[32] Lomonaco MJ. Cryptarray: a scalable and reconfigurable architecture for cryptographic applications. M.Sc. Thesis, University of Central Florida, 2004.
[33] Majzoub S, Diab H. MorphoSys reconfigurable hardware for cryptography: the twofish case. J Supercomput 2012;59(1):22–41.
[34] Theodoropolous D, Siskosy A, Pnevmatikatosy D. CCproc- a custom VLIW cryptography co-processor for symmetric-key ciphers. In: J. Becker et al., editors. Proceedings International Workshop on Applied Reconfigurable Computing (ARC2009): Reconfigurable Computing: Architectures, Tools and Applications, - LNCS 5453–2009; 318–23.
[35] Niu Y, Wu L, Liu Y, Zhang X, Chen H. A 10 gbps in-line network security processor based on configurable hetero-multi-cores. J Zhejiang Univ-SCIENCE C (Comp Electron) 2013;14(8):642–51.
[36] Hämäläinen P, Hännikäinen M, Hämäläinen T, Corporaal T, Saarvinen J. Implementation of encryption algorithms on transport triggered architecture. In: Proceedings International Symposium on Circuits and Systems (ISCAS 2001) May 2001: 6–9.
[37] Hämäläinen P, Heikkinen J, Hännikäinen M, Hämäläinen TD. Design of transport triggered architecture processors for wireless encryption. In: Proceedings 8th Euromicro Conference on Digital System Design (DSD'05) 2005; 144–52.
[38] Fronte D, Perez A, Payrat E. Celator: A multi-algorithm cryptographic co-processor. In: Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig'08) 2008; 438–43.
[39] Sayilar G, Chiou D. Cryptoraptor: high throughput reconfigurable cryptographic processor. In: Proceedings Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on, 2–6 Nov. 2014; 155–61.
[40] Grand M, Bossuet L, Gogniat G, Le Gal B, Delahaye JP, Dallet D. A reconfigurable multi-core cryptoprocessor for multi-channel communication systems. In: Proceedings 25th IEEE International Parallel and Distributed Processing Symposium 16–20 May 2011; 199–206.
[41] Wa B, Liu L. A flexible and energy-efficient reconfigurable architecture for symmetric cipher processing. 2015 IEEE International Symposium on Circuits and Systems (ISCAS) 24–27 May 2015; 1182–5.
[42] Elsayed HM, El-Hadidi MT, Osama K, Aslan H. Multi-objective genetic algorithm-based optimization of an asynchronous data-flow security processor. In: Proceedings 2016 33rd National Radio Science Conference (NRSC2016) 2016; 168–77.
[43] Gries M. Methods for evaluating and covering the design space during early design development. VLSI J Dec 2004;38(2):131–83.
[44] Kempf T. Principles of design space exploration. Multiprocessor Systems on Chip: Design Space Exploration; 2011; 23–47.
[45] Deb K. Multi-objective optimization using evolutionary algorithms. John Wiley; 2001. Chapter 2.
[46] Konak A, Coitb D, Smith A. Multi-objective optimization using genetic algorithms: a tutorial. Reliab Eng Syst Saf Sept 2006;91(9):992–1007.
[47] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Trans. Evolutionary Computing April 2002; 6 (2); 182–97.
[48] Reeves C. Hybrid genetic algorithms for bin-packing and related problems. Ann Oper Res 1996;63(3):371–96.
[49] Pargas R, Jain R. A parallel stochastic optimization algorithm for solving 2D bin packing problems. In: Proceedings 9th Conference on Artificial Intelligence for Applications 1993; 18–25.
[50] Mohamadi N. Application of genetic algorithm for the bin packing problem with a new representation scheme. Math Sci 2010;4(3):253–66.
[51] Iyer A, Marculescu D. Power and performance evaluation of globally asynchronous locally synchronous processors. In: Proceedings 29th Annual International Symposium on Computer Architecture 2002; 158–68.
[52] Stevens KS, Golani P, Beerel PA. Energy and performance models for synchronous and asynchronous communication. In: IEEE Transactions on VLSI Systems March 2011; 19(3); 369–382.
[53] El-Hadidi MT, Elsayed HM, Aslan H, Osama K. Structured design approach for an optimal programmable synchronous security processor. In: Kim H, Choi D, editors. Information Security Applications 2015; 313–25.