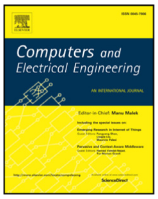
Computers and Electrical Engineering 000 (2017) 1–13

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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



An analysis framework for hardware and software implementations with applications from cryptography[☆]

Issam Damaj[a,*], Safaa Kasbah[b]

[a] Electrical and Computer Engineering Department, American University of Kuwait, Salmiya, Kuwait
[b] Computer Science and Mathematics Department, Lebanese American University, Beirut, Lebanon

ARTICLE INFO

Article history:
Received 1 June 2017
Revised 5 June 2017
Accepted 5 June 2017
Available online xxx

Keywords:
Analysis
Hardware
Software
Gate arrays
Algorithms
Cryptography

ABSTRACT

With the richness of present-day hardware architectures, tightening the synergy between hardware and software has attracted a great attention. The interest in unified approaches paved the way for newborn frameworks that target hardware and software co-design. This paper confirms that a unified statistical framework can successfully classify algorithms based on a combination of the heterogeneous characteristics of their hardware and software implementations. The proposed framework produces customizable indicators for any hybridization of processing systems and can be contextualized for any area of application. The framework is used to develop the Lightness Indicator System (*LIS*) as a case-study that targets a set of cryptographic algorithms that are known in the literature to be tiny and light. The *LIS* targets state-of-the-art multi-core processors and high-end Field Programmable Gate Arrays (*FPGAs*). The presented work includes a generic benchmark model that aids the clear presentation of the framework and extensive performance analysis and evaluation.

1. Introduction

With the advancements in high-performance computing, algorithms have a wide range of efficient implementation options. Current computers can be equipped with multi-core processors, Graphics Processing Units (*GPUs*), and high-end programmable devices, such as, *FPGAs*. The variety of processing options are supported by a wealth of co-design tools that facilitates hardware and software implementations [1,2]. Nevertheless, several questions remain on what algorithm is the best to suite an implementation option, and vice-versa. How would an algorithm perform within hybrid processing systems, and how to make an evaluation based on heterogeneous performance measurements?

The core of any performance measurement includes measures, metrics, and indicators. Indicators are defined as qualitative or quantitative factors, or variables that provide simple and reliable means to measure achievement. A qualitative performance indicator is a descriptive characteristic, an opinion, a property or a trait. However, a quantitative performance indicator is a specific numerical measurements resulted by counting, adding, averaging numbers or other computations [3]. Qualitative and quantitative measurements can be combined to define measurement frameworks and benchmarks [4]. There

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Area Editor Dr. G. Martinez Perez.
* Corresponding author.
E-mail addresses: idamaj@auk.edu.kw (I. Damaj), safaakasbah@gmail.com (S. Kasbah).

http://dx.doi.org/10.1016/j.compeleceng.2017.06.008
0045-7906/© 2017 Elsevier Ltd. All rights reserved.

is a large number of hardware and software benchmarks in the literature. Yet, limited research work is reported to address developing analysis frameworks for heterogeneous hardware and software implementations.

In this paper, we present a statistical analysis framework for performance profiling of related algorithms running under different hardware and software subsystems. The framework comprises criteria, indicators, and measurements obtained from heterogeneous sources. The measurements are statistically combined to produce indicators that capture the algorithmic, software, and hardware characteristics of the assessed algorithms. The developed framework enables the deep and thorough reasoning about each hardware and software subsystem, and combines heterogeneous characteristics to provide overall ratings, rankings, and classifications. The proposed framework is customizable for any hybridization of processing systems and can target any model of computation or area of application.

The paper includes the development of a generic benchmark model that serves as a specification pattern of analysis and evaluation frameworks. The model captures the activities, resources, implementation, mathematical formulation, and intended measurements of an analysis framework or a benchmark. The developed model can be used to describe any benchmark with simplicity and clarity. The model is adopted to present the proposed analysis framework.

To validate the proposed framework in application, a case-study is carried out with application from cryptography. The case-study enables the development of the*LIS* with its bouquet of statistical indicators. The *LIS* formulates the proposed framework within the context of lightweight cryptographic algorithms. The proposed performance analysis classifies the investigated algorithms into a combination of their mathematical, software, and hardware characteristics. The two main targeted high performance computing devices are multi-core processors for software implementations and *FPGAs* for hardware implementations.

The rest of the paper is organized so that Section 2 surveys the literature. In Section 3, the motivation, research questions, and the paper contribution are presented. In Section 4, the generic benchmark model and the analysis framework are presented. Section 5 introduces the *LIS* according to the generic model. A thorough performance analysis and evaluation is presented in Section 6. Section 7 concludes the paper and sets the ground for future work.

2. Related work

2.1. Benchmarks

Benchmarks are widely addressed in the literature. Famous benchmarks include Whetstone, LINPAC, Dhrystone, Standard Performance Evaluation Corporation (SPEC), etc [5–7]. Several developments of embedded systems Benchmarks are lead by the Embedded Microprocessor Benchmark Consortium (*EEMBC*). *EEMBC* helps system designers in selecting the optimal processors, smartphones, tablets, and networking appliances. *EEMBC* mainly targets embedded system's hardware and software. *EEMBC* organizes its benchmark suites targeting automotive, digital media, multi-core processors, networking, automation, signal processing, hand-held devices, and browsers. The benchmarks developed by *EEMBC* include *AutoBench, BrowsingBench, AndBench*, and *MiBench* [8]. Cryptography benchmarks are designed to measure the performance of different cryptographic algorithms running under different systems, such as, *GPUs* or other processors. Rukhin et al. in [9] presents a statistical test suite for random and pseudorandom number generators for cryptographic applications. Yue et al. in [10] presents a cryptographic benchmark suite for network processors (NPCryptBench).

2.2. Hardware/software co-design evaluation frameworks

Performance analysis and evaluation within hardware/software (HW/SW) co-design investigations are usually based on a variety of metrics. Besides standard metrics, such as execution time, maximum frequency, throughput, hardware resource utilization, power consumption, etc., several metrics are identified within the context of application. Jain–Mendon and Sass in [11] proposed a HW/SW co-design approach for implementing sparse matrix vector multiplication on *FPGAs*. Within the context of application, the authors evaluated their approach by analyzing the hardware and software implementations in terms of the speed of processing floating point operations, bandwidth efficiency, data block size, communication time, etc. Lumbiarres–Lopez et al. [12] implemented, within a co-design environment, a countermeasure against side-channel analysis attacks. The used application-specific metrics comprise the difference in change of input current over time and correlations between data and power consumption. All the aforementioned investigations employed the standard co-design metrics. In [13], the performance of block tridiagonal solvers was evaluated under heterogeneous multi-core processors and *GPUs*. The evaluation was mainly based on analyzing memory performance and measuring the total execution times of different scenarios.

The standard metrics of co-design applies to partitioned hardware and software implementations. The focus in partitioned implementation is the analysis and evaluation of the developed subsystems with an aim to find the best possible partitioning strategy. Wu et al. [14] studied the performance and algorithmic aspects of a proposed heuristic partitioning algorithm. The produced implementations were analyzed with-respect-to execution time, resource utilization, and the attained solution quality as related to the smallest possible error. In [15], Jemai and Ouni proposed a partitioning strategy based on control data graphs. The partitioning algorithm was deployed within three different case-studies. The metrics analyzed across the three studies comprised the number of partitions, software execution time, hardware resource utilization, software resource utilization, etc. The authors adopted a pattern chart that summarizes the performance analysis results and aids the evaluational of the algorithm.

Nevertheless the evaluation approaches for HW/SW co-design considered various aspects, limited attempts were made to combine multiple measurements and characteristics in unified indicators. Spacy et al. in [16] investigated the automatic quantification of acceleration opportunities for programs across a wide range of heterogeneous architectures. The investigation focused on allowing designers to identify promising implementation platforms before investing in a particular HW/SW co-design and a specific partitioning scenario. The authors unified many hardware and software characteristics into a single execution time estimate. The incorporated hardware characteristics included cycle time, number of parallel execution units, execution efficiency, bus latency, bus width, and hardware size capacity. The employed software characteristics included the execution time, the number of parallel execution slots, program code unit iterations, control flows, data flows, and size of codes. Additional composite indicators were developed to calculate speedup factors. The combined characteristics were used to calculate co-design performance estimates and evaluating opportunities for hardware acceleration.

3. Research objectives

The modern trend in computer systems is clearly in the direction of further hybridization using high-end co-processing systems. Hybrid systems are mainly studied within HW/SW co-design and co-analysis frameworks. To increase the effectiveness of co-analysis and accordingly co-design, the following research opportunities are highlighted:

- The identification of commonly-used metrics in HW/SW co-design
- Addressing the need for the contextualization of application-specific metrics, properties, and key indicators in HW/SW co-design applications
- Evaluating implementations–given the heterogeneous characteristics of the targeted systems
- The identification of optimized combinations of hardware and/or software implementations based on co-analysis
- The limited attempts in the literature to combine multiple measurements and characteristics in unified indicators that can rank, rate, classify, and evaluate algorithms for hardware and software implementations
- The limited work in the literature to develop co-analysis frameworks that target cryptographic algorithms

The research objective of this paper is mainly to develop a statistical framework that can combine heterogeneous characteristics of algorithms and their implementations in hardware and software. The framework aims at being portable across different hardware and software systems, customizable, scalable, and able to target any area of application. In addition, the framework aids the composition of a bouquet of indicators to capture specific desirable properties and enable classifying, ranking, rating, and evaluating algorithms and implementations. In addition, the objectives include the follows:

- Provide a generic benchmark model that serves as a specification pattern of analysis and evaluation frameworks. The developed model aims at being clear, simple, and highly reusable. The model is used to present the developed analysis framework.
- Validate the proposed framework by developing the *Lightness Indicator System* for cryptographic ciphers.
- Perform a thorough analysis and evaluation based on the *LIS* system for a set of cryptographic ciphers.
- Studying the integration of the developed framework within and Integrated Development Environment (*IDE*) that can connect to various hardware and software implementation and analysis tools.

The paper includes a thorough evaluation of the framework and a discussion on its usefulness.

4. The generic model and the analysis framework

4.1. The generic benchmark model

The proposed generic model diagrams the continuum of important elements of benchmarks and analysis frameworks. The generic model defines the goal, inputs, activities, output, outcomes, and the desired performance profile of a benchmark. The model captures the relationships among the resources, implementation, mathematical formulation, and the obtained results. Moreover, it standardizes the evaluation process that can be applied to any benchmark. The proposed model consists of the following six elements; the model elements are diagrammed in Fig. 1:

1. **Goal**: is the definition of the aim of the benchmark, or the analysis framework, and what does it mainly provide.
2. **Input**: is the identification of the algorithms under study, implementation environments, reference algorithm, performance metrics, etc.
3. **Activities**: are the implementations of the algorithms under the identified environments and collection of results.
4. **Output**: is the formulation of the key indicators and development of their rubrics–if any.
5. **Outcomes**: are the formulations of the statistical assessment as combinations of the Output.
6. **Performance**: is the application of the developed assessment framework to profile and classify algorithms according to the obtained results.

The proposed model provides a generic profiling pattern than can be used for any benchmark or analysis framework.

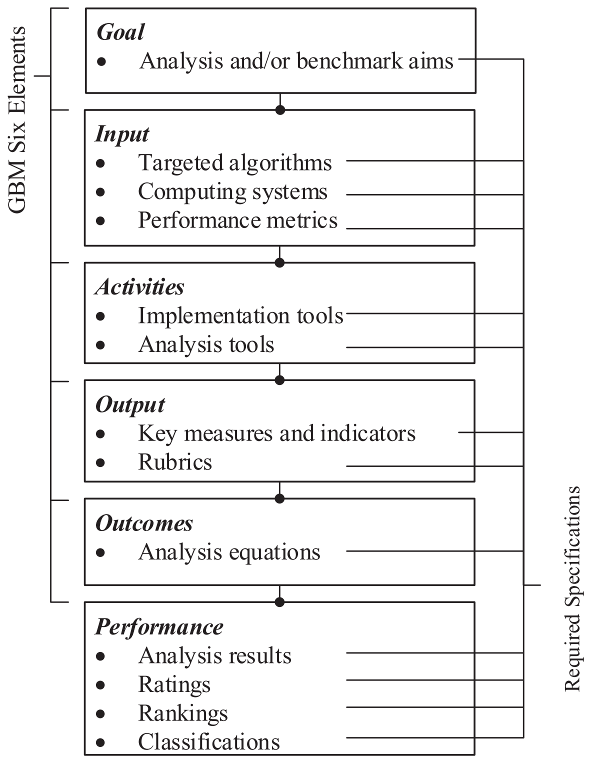


Fig. 1. The six elements diagram of the Generic Benchmark Model with the required specifications.

4.2. The analysis framework

The proposed analysis framework classifies the heterogeneous sources of measurements into analysis profiles (APs), such as, general algorithmic, hardware, and/or software. The development of each profile includes the identification of a set of Key Indicators (*KIs*). The indicators are the most extensive part of the measurement framework and should be carefully developed within the context of application. The measurements associated with the identified indicators may include quantities, scores from scale-rubrics, etc. The measured indicators are then each divided by a measurement from a reference algorithm for normalization and for producing performance ratios. Accordingly, Combined Measurement Indicators (*CMIs*) are calculated using the Geometric Mean of *KI* ratios. The generic equation of *CMIs* is as follows:

$$CMI = \sqrt[n]{ratio_1 \times ratio_2 \times \ldots ratio_n}$$

Where $ratio_i = \dfrac{KI_{i.j}}{KI_{i.j}^{ref}}$

$KI_{i,j}$ is the *i*th *KI* of the *j*th *AP*,

and $KI_{i.j}^{ref}$ is the reference measurement of the indicator $KI_{i,j}$

The Geometric Mean is used, in the *CMI* equation, as it is able to measure the central tendency of data values that are obtained from ratios. Using the Geometric Mean insures the following two important properties [17–19]:

- Geometric Mean of the ratios is the same as the ratio of Geometric Means.
- Ratio of the Geometric Means is equal to the Geometric Mean of performance ratios; which implies that when comparing two different implementations' performance, the choice of the reference implementation is irrelevant [19].

The developed statistical framework can be applied in different areas of applications and using different *APs*. The application includes contextualizing the *KIs* of every *AP* according to the characteristics of the targeted area.

5. The application of the LIS system to cryptographic algorithms

In the following sections, we present the *LIS* based on the generic model.

Table 1
The rubric of the complexity analysis indicator.

General	Scale				
Indicator	Logarithmic low	Logarithmic high	Linear	Almost quadratic	Higher than quadratic
Complexity analysis	$O(logn)$	$\omega(logn)$ but better than Linear	$\theta(n)$	$O(n^2)$ but worse than Linear	$\omega(n^2)$

5.1. Goal

The *LIS* is a HW/SW statistical framework that provides performance profiling for lightweight cryptographic algorithms running under different hardware and software systems. The *LIS* combines a bouquet of performance metrics that include speed, algorithmic complexity, memory efficiency, algorithmic strength, hardware size, etc.

5.2. Input

The input identifies the targeted algorithms, computing systems, and the performance metrics. The *LIS* targets a set of cryptographic algorithms that are specific for applications running on low resources. The selected ciphers are classified, in the literature, as tiny, small, lightweight, or ultra-lightweight ciphers. The targeted ciphers are *Skipjack, 3-WAY, XTEA, KATAN* and *KTANTAN*, and *Hight*. The reference cipher is the *AES* [20].

The two targeted high performance computing devices are the *Dell Precision T7500* with its dual quad-core *Xeon processor* and *24 GB* of *RAM*. The targeted *FPGA* is *Altera Stratix-IV*.

The identified performance metrics of the *LIS* are classified into general algorithmic, hardware, and software profiles. The general algorithmic profile includes the complexity of the algorithms and their security strength. Within the multi-core environment, the software profile includes execution times, clocks per instruction, throughput, and a cache analysis. Within the *FPGA* environment, the hardware profile includes the resource utilization, propagation delay, throughput, power consumption, etc.

5.3. Activities

The activities includes software implementations under *C* and hardware implementations under *VHDL*. The Software tools used for software and hardware implementations and profiling are *Quartus, ModelSim*, and *Intel VTune Amplifier* under Visual Studio.

5.4. Output

The outputs of the analysis framework are measures and indicators. The measures are the general algorithmic, hardware, and software profiles. The indicators of the general algorithmic profile intend to capture the complexity and ciphering strength of the algorithm; the *KIs* include the following in bold:

- **Algorithm Complexity** (*AC*): Asymptotic complexity analysis using the Big-*O*, small-ω, and θ notations
- Cipher Strength: based on **Key Size** (*KS*), **Number of Rounds** (*NR*), and the text **Block Size** (*BS*)

Complexity analysis of algorithms is the determination of the amount of resources necessary to execute them. To analyze the complexity of studied algorithms, we study their asymptotic behavior. The asymptotic behavior classifies algorithms according to their rate of growth with respect to the increase in input size. The following standard complexity analysis classification is adopted [21]:

- $O(f(n))$: The rate of growth of an algorithm is asymptotically no worse than the function $f(n)$ but can be equal to it.
- $\omega(f(n))$: The rate of growth of an algorithm is asymptotically no better than the function $f(n)$.
- $\theta(f(n))$: The rate of growth of an algorithm is asymptotically equal to the function $f(n)$.

Here, n is the size of input.

To facilitate the assessment of the studied ciphers, a rubric is created. The rubric scale points are logarithmic low (*LL*), logarithmic high (*LH*), Linear (L), Almost Quadratic (*AQ*), and Higher than Quadratic (*HQ*). For instance, *LL* describes the case when the complexity is asymptotically no worse than *logn* but can be equal to it; such a complexity is formulated as $O(logn)$. The complete description of the rubric is shown in Table 1. In preparation for the statistical formulation, we map this qualitative properties onto quantities. For every point in the scale, we map it onto a fixed number. Hence, each point in the scale is mapped onto the values 20%, 40%, 60%, 80%, and 100%.

Cipher Strength is an assessment of the algorithm based on a variety of aspects that can include Key Size, the Number of Rounds, and the Block Size [22]. Key size or key length is the size measured in bits of the key used in cryptographic algorithms. The security of the cryptographic algorithms is function of the length of its key. For some algorithms, such as those targeted in this investigation, the longer the key, the more resistant is the algorithm [23]. However, in the broader context, the relation between key lengths and security could be more delicate [24]. For example, key sizes of 80, 160, and 1024 bits, nevertheless different, they imply comparable security when 80 is for a symmetric cipher, 160 is for a hash length, and 1024 is for RSA modulus [24]. In addition, Elliptic Curve Cryptography (ECC) is famous for its strength that can be attained at relatively small key sizes. For instance, a comparable security level can be achieved using RSA with a key size of 15,360 bits and ECC with a key size of only 512 bits [25]. Investigations relating the level of security, and the strength of the algorithm, to the key size are given wide and careful attention in the literature [24–26]. The most recent standardized key size requirements for security are published at [26].

Furthermore, block ciphers transform a plain-text block of several bits into an encrypted block. The block size cannot be too short in order to secure the cryptographic scheme. In other words, the larger the block size is, the greater the cipher strength [23]. In addition, rounds are important to the strength of ciphers; a single round is usually responsible for mixes, permutations, substitutions, and shifts in the text being encrypted. Mostly, more rounds lead to greater confusion and diffusion and hence stronger security. Indeed, indicators like the Key Size, Number of Rounds, and Block Size should be carefully adopted and specified within the scope of the targeted cryptographic algorithms. The proposed indicators are not necessarily applicable to all cryptographic algorithms.

The software profile includes the following indicators[27]:

- **Execution Time** (*ET*): the time between the start and the completion of a task.
- **Throughput** (*TH*): the total amount of work done in a given time.
- **Clock Cycle per Instruction** (*CPI*): the average number of clock cycles each instruction takes to execute.
- **Cache Miss Ratio** (*CMR*): the ratio of memory accesses cache miss.

The hardware profile includes *ET, TH* and the following indicators:

- **Propagation Delay** (*PD*): the time required for a signal from an input pin to propagate through combinational logic and appear at an external output pin.
- **Look-Up Table** (*LUT*): the number of combinational adaptive lookup tables required to implement an algorithm in hardware. The number of *LUTs* is an indicator of the size of hardware in Altera devices. In other devices, the area could be measured in terms the total number of gates, logic elements, slices, etc.
- **Logic Register** (*LR*): the total number of logic registers in the design.
- **Power Consumption** (*PC)*: the power consumption of the developed hardware in Watts.

5.5. Outcomes

The Outcomes element is the formulation of *CMIs* as function of *KIs*. The Lightness Indicator *LI* is the main *CMI* calculation in the presented statistical analysis framework. The *LI* is calculated in terms of several *AP*s; three for the current study, namely the General Algorithmic Profile (*GAP*), Software Profile (*SWP*), and Hardware Profile (*HWP*). The simplified form of *LI* is shown in Eq. (1):

$$LI = \sqrt[l]{ratio_1 \cdot ratio_2 \cdot ratio_3 \ldots ratio_l} \tag{1}$$

and hence

$$LI = (\prod_{i=1}^{l} ratio_i)^{\frac{1}{l}}$$

Where l is the number of key indicators.

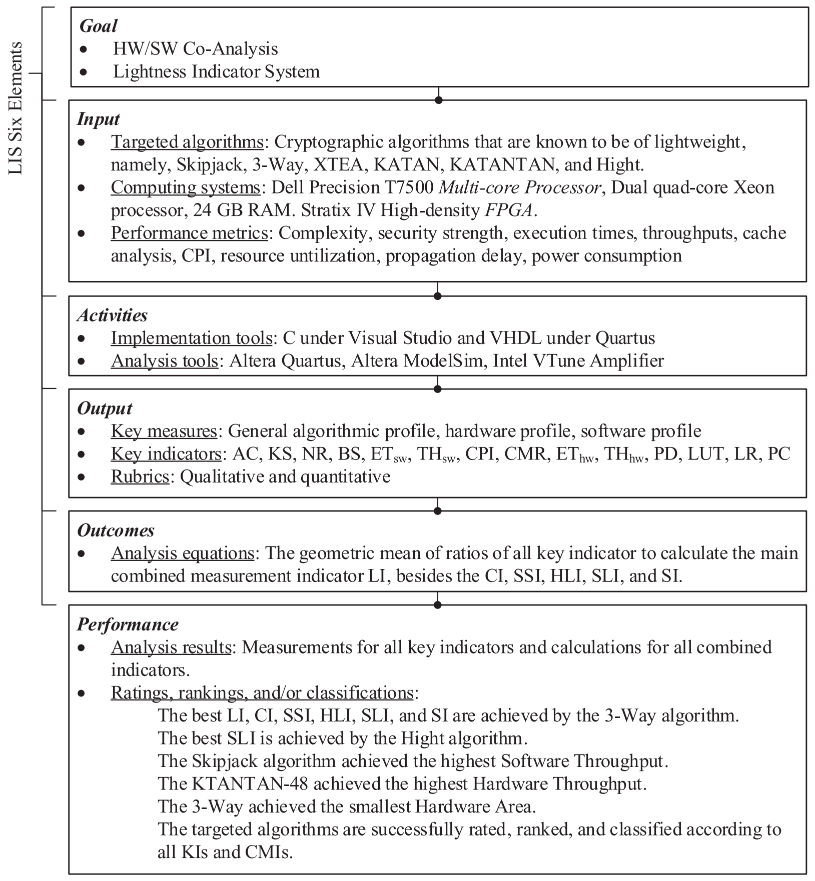
The weighted version of *LI* is denoted by *wLI* in Eq. (2). The weighted version enables the emphasis of specific indicators. If all the assigned weights are equal, the *wLI* is the same as *LI*.

$$wLI = \left(\prod_{k=1}^{l} ratio_k^{w_k}\right)^{\frac{1}{\sum_{k=1}^{l} w_k}} \tag{2}$$

Where w_k is the weight of the kth ratio.

The *LIS* enables the classification of cryptographic algorithms according to their lightness. A higher *LI* is achieved through a higher throughput, a more efficient memory performance, more compact size, less complexity, less power consumption, and less resource utilization. The *LI* is either directly or inversely proportional to the indicators. The aim of the chosen proportion is to emphasize lightness; the proportions could be modified to capture other properties. The master *LIS* formula using the developed indicators is shown in Eq. (3). The indicators that are common to the Software (sw) and Hardware (hw) profiles are labeled with the profile name.

$$\begin{aligned} LI &= \sqrt[14]{GAP \cdot SWP \cdot HWP} \\ GAP &= \frac{AC_{ref}}{AC} \cdot \frac{KS_{ref}}{KS} \cdot \frac{NR_{ref}}{NR} \cdot \frac{BS_{ref}}{BS} \\ SWP &= \frac{ET_{sw,ref}}{ET_{sw}} \cdot \frac{TH_{sw}}{TH_{sw,ref}} \cdot \frac{CPI_{ref}}{CPI} \cdot \frac{CMR_{ref}}{CMR} \end{aligned}$$



Goal
- HW/SW Co-Analysis
- Lightness Indicator System

Input
- Targeted algorithms: Cryptographic algorithms that are known to be of lightweight, namely, Skipjack, 3-Way, XTEA, KATAN, KATANTAN, and Hight.
- Computing systems: Dell Precision T7500 *Multi-core Processor*, Dual quad-core Xeon processor, 24 GB RAM. Stratix IV High-density *FPGA*.
- Performance metrics: Complexity, security strength, execution times, throughputs, cache analysis, CPI, resource untilization, propagation delay, power consumption

Activities
- Implementation tools: C under Visual Studio and VHDL under Quartus
- Analysis tools: Altera Quartus, Altera ModelSim, Intel VTune Amplifier

Output
- Key measures: General algorithmic profile, hardware profile, software profile
- Key indicators: AC, KS, NR, BS, ET_{sw}, TH_{sw}, CPI, CMR, ET_{hw}, TH_{hw}, PD, LUT, LR, PC
- Rubrics: Qualitative and quantitative

Outcomes
- Analysis equations: The geometric mean of ratios of all key indicator to calculate the main combined measurement indicator LI, besides the CI, SSI, HLI, SLI, and SI.

Performance
- Analysis results: Measurements for all key indicators and calculations for all combined indicators.
- Ratings, rankings, and/or classifications:
  The best LI, CI, SSI, HLI, SLI, and SI are achieved by the 3-Way algorithm.
  The best SLI is achieved by the Hight algorithm.
  The Skipjack algorithm achieved the highest Software Throughput.
  The KTANTAN-48 achieved the highest Hardware Throughput.
  The 3-Way achieved the smallest Hardware Area.
  The targeted algorithms are successfully rated, ranked, and classified according to all KIs and CMIs.

Fig. 2. The six elements diagram of the *LIS*.

$$HWP = \frac{ET_{hw,ref}}{ET_{hw}} \cdot \frac{TH_{hw}}{TH_{hw,ref}} \cdot \frac{PD_{ref}}{PD} \cdot \frac{LUT_{ref}}{LUT} \cdot \frac{LR_{ref}}{LR} \cdot \frac{PC_{ref}}{PC} \tag{3}$$

The *LIS* provides the following set of combined statistical indicators:
Complexity Indicator (*CI*):

$$CI = \sqrt[5]{\frac{AC}{AC_{ref}} \cdot \frac{KS}{KS_{ref}} \cdot \frac{NR}{NR_{ref}} \cdot \frac{BS}{BS_{ref}} \cdot \frac{CPI}{CPI_{ref}}}$$

Security Strength Indicator (*SSI*):

$$SSI = \sqrt[4]{\frac{AC}{AC_{ref}} \cdot \frac{KS}{KS_{ref}} \cdot \frac{NR}{NR_{ref}} \cdot \frac{BS}{BS_{ref}}}$$

Hardware Lightness Indicator (*HLI*):

$$HLI = \sqrt[6]{\frac{ET_{hw,ref}}{ET_{hw}} \cdot \frac{TH_{hw}}{TH_{hw,ref}} \cdot \frac{PD_{ref}}{PD} \cdot \frac{LUT_{ref}}{LUT} \cdot \frac{LR_{ref}}{LR} \cdot \frac{PC_{ref}}{PC}}$$

Software Lightness Indicator (*SLI*):

$$SLI = \sqrt[4]{\frac{ET_{sw,ref}}{ET_{sw}} \cdot \frac{TH_{sw}}{TH_{sw,ref}} \cdot \frac{CPI_{ref}}{CPI} \cdot \frac{CMR_{ref}}{CMR}}$$

Speed Indicator (*SI*):

$$SI = \sqrt[5]{\frac{ET_{sw,ref}}{ET_{sw}} \cdot \frac{TH_{sw}}{TH_{sw,ref}} \cdot \frac{ET_{hw,ref}}{ET_{hw}} \cdot \frac{TH_{hw}}{TH_{hw,ref}} \cdot \frac{PD_{ref}}{PD}}$$

5.6. Performance

The analysis based on the *LIS Output* and *Outcomes* provides measurements for all *KIs* and enables the calculation of the defined *CMIs*. The results include rating, ranking, and classifying the targeted algorithms. The analysis and evaluation of results are presented in Section 6. The six elements of the *LIS* are summarized in Fig. 2.

Table 2
General Algorithmic profile.

Algorithm name	AC	Mapped AC	KS	NR	BS
Skipjack	AQ	0.8	80	32	64
XTEA	AQ	0.8	96	64	64
3-WAY	AQ	0.8	128	11	96
HIGHT	AQ	0.8	128	32	64
KATAN-32	AQ	0.8	80	254	32
KATAN-48	AQ	0.8	80	254	48
KATAN-64	AQ	0.8	80	254	64
KTANTAN-32	AQ	0.8	80	254	32
KTANTAN-48	AQ	0.8	80	254	48
KTANTAN-64	AQ	0.8	80	254	64
AES	**AC**	**0.8**	**192**	**12**	**128**

5.7. *Programming interface*

The developed statistical framework is embedded in a sample co-design *IDE*. The *IDE* is implemented using *Java* under *Netbeans*. Moreover, the code editor is implemented using *RSyntaxTextArea* Java framework, while the *IDE* theme is implemented using *JTattoo* Java framework. The used implementation and performance evaluation tools comprise *Altera Quartus* and *Altera ModelSim* for Hardware implementation and analysis, and In*tel vTune Amplifier* under *Visual Studio* for Software analysis. The developed IDE connects to *Altera Quartus* using *TCL* commands to synthesize and generate timing analyses, pin assignments for *FPGA* boards, and generate bit files to program the targeted *FPGAs*. The *IDE* connects to *Intel vTune Amplifier*, using Command Line and Batch Files, to perform the software analysis and calculating the total execution time, CPI, etc. The generated hardware and software analysis files are exported to *MS Excel* to produce the complete analysis profile and charts.

6. Analysis and evaluation

6.1. *Performance analysis*

The *LIS* is an application of an analysis framework that can be the core part of a benchmark within HW/SW co-design. The developed framework is applied by developing the *LIS* on several cryptographic ciphers that are presented in the literature as lightweight, tiny, small, or minute. The developed system enables the validation of the lightness of the algorithms through measurements and statistical analysis.

In accordance with our generic benchmark model, and upon the identification of the system *Goal* and *Input*, the *Activities* are done according to the following procedure:

1. Implement hardware using VHDL under *Quartus*
2. Analyze the hardware profile using *Quartus* and *ModelSim*
3. Implement software using C-language under Visual Studio and its integrated *Intel VTune Amplifier*
4. Analyze the software profile using *Intel VTune Amplifier*
5. Derive and analyze the general algorithmic profile
6. Combine and analyze the results from all profiles using a statistical software tool

With the finalization of *Activities*, the following steps complete the elements of the framework:

7. Produce the *Output* key indicators
8. Calculate the combined indicators of the *Outcomes*
9. Build the overall *Performance* report

Tables 2–4 present the derivation and implementation results of the general algorithmic, software, and hardware profiles. On the simple indicators level, the *Skipjack* algorithm achieved the highest software execution throughput of 156.098Mbps, while the highest hardware execution throughput, 480 Mpbs is achieved by the *KTANTAN-48* algorithm. The *3-Way* algorithm attained the smallest hardware area of 77*ALUTs* and 167*LRs*.

The *Lightness, Complexity, Security Strength, Hardware Lightness, Software Lightness*, and *Speed* indicators are shown in Table 5. The algorithm that attained a larger indicator value is lighter, of a less complexity, of a higher security strength, or faster than the algorithm with a lower indicator value. Figs. 3–6 present a per-*CMI* comparison. The *3-Way* got the best lightness index of 2.52, while $KATAN - 64$ attained the lowest with an index of 0.76. The best *CI* index is 1.19; attained by the *3-Way* algorithm. The best *SSI, HLI*, and *SI* indices are 1.16, 5.24, and 5.08; all attained by the *3-Way* algorithm. The HIGHT algorithm achieved the highest *SLI* index of 3.4.

Table 3
Software profile.

Algorithm name	BS	ET(μsec)	TH(Mbps)	CPI	CMR
Skipjack	64.000	0.410	156.098	1.327	0.164
XTEA	64.000	2.570	24.903	0.729	0.033
3-WAY	96.000	2.320	41.379	1.107	0.036
HIGHT	64.000	8.640	7.407	1.330	0.000
KATAN-32	32.000	27.460	1.165	0.634	0.006
KATAN-48	48.000	40.330	1.190	0.634	0.004
KATAN-64	64.000	52.830	1.211	0.627	0.003
KTANTAN-32	32.000	791.080	0.040	0.986	0.001
KTANTAN-48	48.000	803.320	0.060	0.975	0.001
KTANTAN-64	64.000	821.830	0.078	0.965	0.001
AES	**128.000**	**23.210**	**5.515**	**1.235**	**0.004**

Table 4
Hardware profile.

Algorithm name	ET(μsec)	TH(Mpbs)	PD(nsec)	ALUT	LR	PC(mW)
Skipjack	7.49	8.55	11.90	554.00	142.00	331.01
XTEA	6.18	10.35	11.10	2799.00	135.00	332.77
3-WAY	0.80	120.00	3.82	77.00	167.00	331.01
HIGHT	1.85	34.59	127.78	2036.00	72.00	332.66
KATAN-32	1.47	21.77	43.57	2145.00	540.00	328.63
KATAN-48	1.89	25.40	79.94	3982.00	556.00	329.95
KATAN-64	2.38	26.89	78.31	4315.00	572.00	330.94
KTANTAN-32	0.09	372.09	40.03	1947.00	112.00	328.58
KTANTAN-48	0.10	480.00	72.78	3662.00	128.00	329.81
KTANTAN-64	0.15	438.36	79.30	4075.00	144.00	331.00
AES	**1.46**	**6.83**	**5.35**	**3998.00**	**750.00**	**654.87**

Table 5
Indicators.

Algorithm name	LI	CI	SSI	HLI	SLI	SI
Skipjack	1.57	1.11	1.16	1.42	2.46	2.81
XTEA	1.22	1.05	0.93	1.18	1.70	1.48
3-WAY	2.52	1.19	1.22	5.24	1.75	5.08
HIGHT	1.93	1.01	1.03	2.02	3.40	1.43
KATAN-32	0.89	0.98	0.82	1.12	0.69	0.59
KATAN-48	0.79	0.90	0.74	0.90	0.70	0.47
KATAN-64	0.76	0.85	0.69	0.86	0.71	0.44
KTANTAN-32	1.04	0.89	0.82	3.88	0.18	0.48
KTANTAN-48	0.95	0.83	0.74	3.14	0.20	0.47
KTANTAN-64	0.89	0.78	0.69	2.76	0.21	0.44
AES	1.00	1.00	1.00	1.00	1.00	1.00

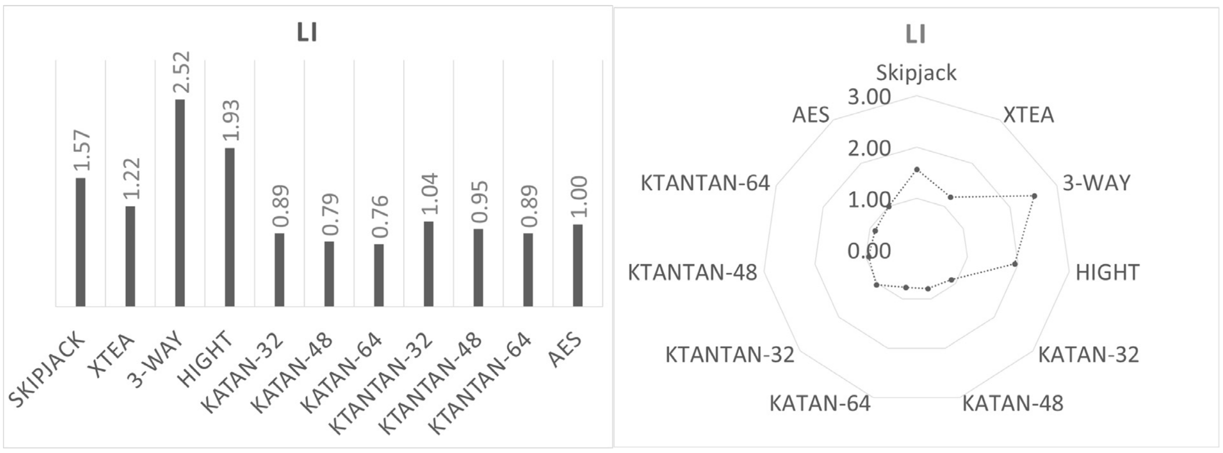


Fig. 3. The lightness indicator classification.

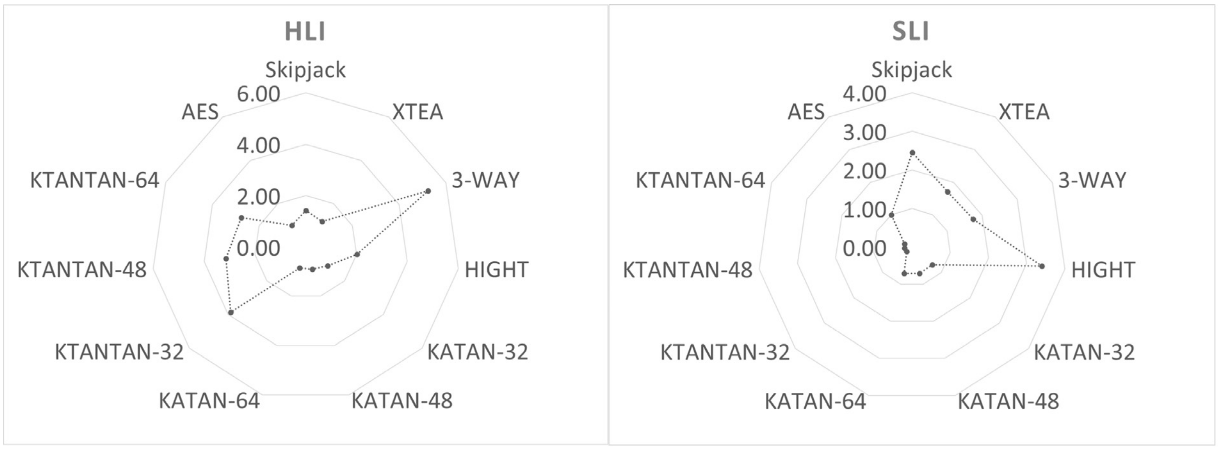


Fig. 4. The software and hardware lightness indicator classifications.

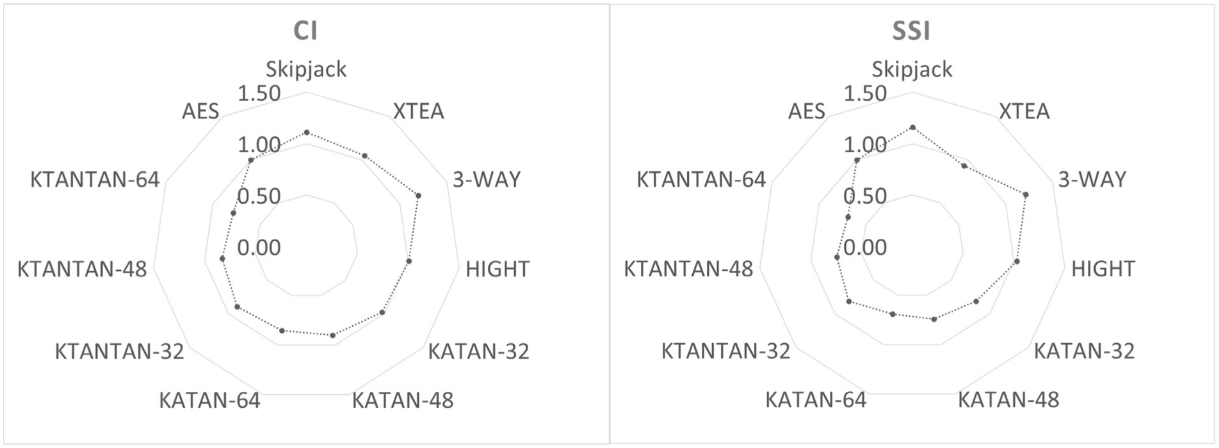


Fig. 5. The complexity and security strength indicator classifications.

6.2. General evaluation

The current investigation can be evaluated at the levels of the framework development, application, and contextualization. The developed framework is unique in combining algorithmic, hardware, and software characteristics to provide unified performance evaluation criteria and useful performance indicators. The framework addresses the need for methods that can analyze the performance and deal with the hybrid nature of modern computing systems. The investigation proposes the creation of unified indexes/indicators that can capture specific qualities in terms of a wide range of heterogeneous key performance indicators, such as the *LIS*. The *LI* served as a master *CMI* while an indicator like the *SI* is developed with focus on speed. Indeed, the framework is scalable and upgradeable without changing the statistical computation or the structure of the measurement. For instance, an additional *AP* can be incorporated into the calculations of the *LIS* to include the performance characteristics of *GPUs*.

At the application level, the developed framework can be used to examine qualities of importance and interest to developers or users. For example, the presented *LIS* enables the indexing and classifying of cryptographic algorithms. Here, the qualities of importance are the lightness, speed, complexity, and security strength. The *LIS* can be applied to examine the same qualities for a similar area of application, such as, signal processing. In signal processing, the *SLI* and *HLI* can be reused. However, the *LI* and *CI* needs to be redefined within the context of signal processing, and the *SSI* is not applicable. Signal processors are usually embedded within real-time application and characterized by their numerical accuracy, acceleration schemes, and the ability to perform fast computations and data access. A *Reliability and Accuracy Indicator* (*RAI*) *CMI* can be created to combine the desired characteristics and capture, index, and aid in the classification of signal processing algorithms.

The contextualization of the framework in relation to the targeted area of application produces a rich and comprehensive set of reference *KIs. KIs*, such as *ET, TH, CPI, CMR, PD, LUT, LR*, and *PC*, are independent of the context of application and thus highly reusable. Other *KIs*, such as *KS, NR*, and *BS* are specific to cryptographic algorithms. *KIs* can measure quantities or

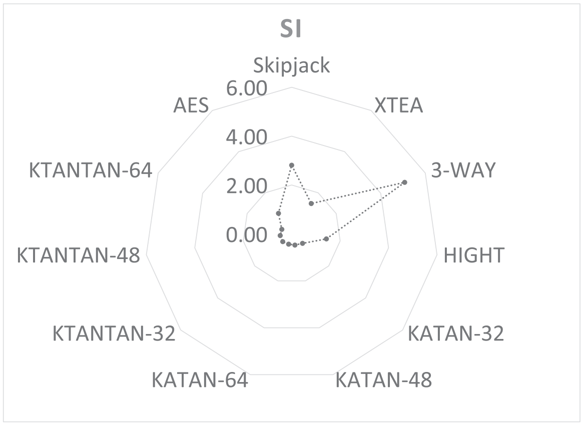


Fig. 6. The speed indicator classification.

describe qualities. Qualities can be easily specified using rubrics and mapped onto quantities that can be substituted into the indicator equations. *KIs* should be carefully identified and developed by experts in the targeted area of application, and supported by evident applicability. The contextualization of the application for the proposed *RAI*, in signal processing, can comprise *KIs*, such as, *Memory Access Time* as a quantitative measurement. The availability of *Specialized Addressing Modes* can be captured as a qualitative indicator.

This paper presents a generic model that specifies the elements of benchmarks and/or analysis frameworks. The benchmark model is used to present the *LIS*, nevertheless the model can be used to describe any benchmark. The developed model is generic, simple, concise, and aids the clear description of benchmarks using a unified pattern.

The developed statistical framework is applied through a case-study that targets a class of cryptographic algorithms. The selected algorithms are presented in the literature as tiny, small, minute, and light. The case-study provided a unified classification criteria that include *LI*. The proposed framework successfully classified the targeted algorithms according to their hardware, software, and algorithmic characteristics. The addressed algorithms are widely implemented, analyzed, and evaluated in the literature. The work presented in the literature is limited to single algorithm evaluation, single system implementation, such as either hardware or software, and still make holistic claims of lightness based on limited number of indicators. The used reference implementation is the *AES* cipher with a key size of 192 bits. The use of other key sizes, such as 256 bits, doesn't change the algorithm classifications or falsify the analysis as it is consistently applied for all the targeted algorithms. However, different indicator values are expected.

The developed framework is intended to capture hardware and software properties. The current investigation is limited to non-partitioned implementations, where the whole computation is delegated to a co-processor. Partitioned implementations can be evaluated, based on the proposed framework, by analyzing the *KIs* of the hardware and software subsystems. The obtained *KI* measurements capture the subsystem characteristics. In addition, carefully defined *CMIs* can rank, rate, and classify different partitioning strategies per optimization target, such as, area, speed, power consumption, etc.

An example similar investigation is presented by Spacey et al. in [16]. The authors combined several hardware and software characteristics within a heuristic to produce a single execution time estimate. The best time estimate is identified based on heterogeneous performance and architectural characteristics of different hardware and software partitions. Our proposed framework would, with no doubt, enrich such investigations and provide versatile estimates with *CMIs* such as *HLI, SLI, CI, SI*, and/or other customized indicators.

7. Conclusion

Modern high-performance computers are hybrids of multi-core processors, *GPUs, FPGAs*, etc. In this paper, a statistical framework is developed to provide thorough analysis and evaluation of algorithms and their implementations on different processing systems. A generic benchmark model is created to present the framework with clarity. The framework categorizes processing subsystems into profiles, where each can be contextualized according to a specific application. The statistical framework is adopted to analyze and evaluate a set of cryptographic algorithms that are claimed to be small in size, tiny, and efficient. The proposed framework enabled the creation of several key indicators including the lightness, complexity, security strength, and speed indicators. The two main targeted high-performance computing devices are multi-core processors for software implementations and high-end *FPGAs* for hardware implementations. The developed lightness indicator ranks the *3-Way* algorithm as the lightest among all with an *LI* of 2.52. *Hight* achieves the second best lightness with a score of 1.93. The lowest score of 0.79 was attained by *KATAN-64*. The case-study validates the statistical framework and leads to a successful classification of the targeted algorithms. The obtained results are based on a combination of three profiles including the algorithmic, software, and hardware profiles. The presented framework enjoys being scalable, upgradeable, and portable across-applications. Future work includes incorporating additional processing systems, targeting other areas of application, and embedding the framework within a co-design *IDE* and target partitioned implementations.

References

[1] Damaj I. Parallel algorithms development for programmable devices with application from cryptography. Int J Parallel Program 2007;35(6):529–72.
[2] Kasbah S, Damaj I, Haraty R. Multigrid solvers in reconfigurable hardware. Comput Appl Math Els 2008;213:79–94. Issue 1.
[3] Rolstadås A, for Information Processing IF. Benchmarking: theory and practice, 148. Chapman & Hall; 1995.
[4] Damaj I, Kranov AA. The sustainability of technical education: A measurement framework. In: The american society of engineering education mid-atlantic conference. New York, US: ASEE; 2013. p. 47–59.
[5] Dongarra J., Luszczek P. Encyclopedia of parallel computing; chap. LINPACK benchmark. Springer US; 2011, p. 1033–1036.
[6] Weicker RP. Dhrystone: a synthetic systems programming benchmark. Commun ACM 1984;27(10):1013–30.
[7] Henning JL. SPEC CPU2000: measuring CPU performance in the new millennium. Computer 2000;33(7):28–35.
[8] EEMBC. Website; 2017 http://www.eembc.org/.
[9] Rukhin A, Soto J, Nechvatal J, Smid M, Barker E. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Tech. Rep. DTIC Document; 2001.
[10] Yue Y, Lin C, Tan Z. Npcryptbench: a cryptographic benchmark suite for network processors. ACM SIGARCH Comput Archit News 2006;34(1):49–56.
[11] Jain-Mendon S, Sass R. A hardware-software co-design approach for implementing sparse matrix vector multiplication on FPGAs. Microprocess Microsyst 2014;38(8):873–88. doi:10.1016/j.micpro.2014.02.004.
[12] Lumbiarres-Lopez R, Lopez-Garcia M, Canto-Navarro E. A new countermeasure against side-channel attacks based on hardware-software co-design. Microprocess Microsyst 2016;45, Part B:324–38. doi:10.1016/j.micpro.2016.06.009.
[13] Park AJ, Perumalla KS. Efficient heterogeneous execution on large multicore and accelerator platforms: case study using a block tridiagonal solver. J Parallel Distrib Comput 2013;73(12):1578–91. doi:10.1016/j.jpdc.2013.07.012. Heterogeneity in Parallel and Distributed Computing.
[14] Shi W, Wu J, Lam S-K, Srikanthan T. Algorithmic aspects for bi-objective multiple-choice hardware/software partitioning; 2014. p. 7–12. doi:10.1109/PAAP.2014.42.
[15] Jemai M, Ouni B. Hardware software partitioning of control data flow graph on system on programmable chip. Microprocess Microsyst 2015;39(4–5):259–70. doi:10.1016/j.micpro.2015.04.006.
[16] Spacey S, Luk W, Kelly P, Kuhn D. Rapid design space visualisation through hardware/software partitioning; 2009. p. 159–64. doi:10.1109/SPL.2009.4914913.
[17] Bullen P, Mitrinovic D, Vasic M. Handbook of means and theirs inequality; 2003.
[18] Denscombe M. The good research guide: for small-scale social research projects. McGraw-Hill Education (UK); 2014.
[19] Hennessy JL, Patterson DA. Computer architecture: a quantitative approach. Elsevier; 2011.
[20] Daemen J, Rijmen V. The design of rijndael: AES-the advanced encryption standard. Springer; 2002.
[21] Cormen TH, Leiserson CE, Rivest RL, Stein C, et al. Introduction to algorithms, 2. MIT press Cambridge; 2001.
[22] Jorstad ND, Landgrave T. Cryptographic algorithm metrics. 20th national information systems security conference; 1997.
[23] Menezes AJ, Van Oorschot PC, Vanstone SA. Handbook of applied cryptography. CRC Press; 2010.
[24] Lenstra AK. Key length. contribution to the handbook of information security; 2004.
[25] Maletsky K. RSA Vs ECC comparison for embedded systems. White Paper, Atmel 2015:5.
[26] BlueKrypt. Website; 2017 https://www.keylength.com.
[27] Patterson DA, Hennessy JL. Computer organization and design: the hardware/software interface. 5th ed. Morgan Kaufmann; 2013.

Issam Damaj, PhD ME BE, is an Associate Professor of Computer Engineering at the American University of Kuwait. His research interests include hardware/software co-design, embedded system design, automation, Internet-of-things, and engineering education. He is a Senior Member of the IEEE and a Professional Member of the ASEE. He maintains an academic website at www.idamaj.net.

Safaa Kasbah, MSc BSc, received a Master Degree in Computer Science, in 2006, from the Lebanese American University. She received a Bachelor Degree in Computer Science and a minor in Physics from the American University of Beirut in 2004. Her main research interests are iterative methods, hardware/software co-design, reconfigurable computing, quantum computing and Information and Knowledge Management.