Energy-Aware Design of Secure Multi-Mode Real-Time Embedded Systems with FPGA Co-Processors

Ke Jiang* Adrian Lifa* Petru Eles* Zebo Peng* Wei Jiang†
*Linköping University, Sweden †University of Electronic Science and Technology of China, China
{ke.jiang, adrian.alin.lifa, petru.eles, zebo.peng}@liu.se weijiang@uestc.edu.cn

ABSTRACT

We approach the emerging area of energy efficient, secure real-time embedded systems design. Many modern embedded systems have to fulfill strict security constraints and are often required to meet stringent deadlines in different operation modes, where the number and nature of active tasks vary (dynamic task sets). In this context, the use of dynamic voltage/frequency scaling (DVFS) techniques and onboard field-programmable gate array (FPGA) co-processors offer new dimensions for energy savings and performance enhancement. We propose a novel design framework that provides the best security protection consuming the minimal energy for all operation modes of a system. Extensive experiments demonstrate the efficiency of our techniques.

1. INTRODUCTION AND RELATED WORK

Security is becoming an important dimension for embedded systems design, since many safety- and reliability-critical applications are now controlled by embedded systems. As the systems become more and more connected, and the number of threats continues to increase, it becomes more and more important to provide appropriate levels of protection [19]. One key aspect of information security is confidentiality. The messages generated, especially those in critical applications, often contain sensitive information that is sent over the network and should not be disclosed to unauthorized parties. Thus, in this paper, we focus on providing confidentiality protection for the system communication.

For modern embedded systems, energy consumption is also a major issue, and energy-efficient design is indispensable, especially for battery powered systems. Dynamic voltage/frequency scaling (DVFS) is one popular technique for achieving better energy-efficiency: lowering the supply voltage in conjunction with the clock frequency of a processor is used for minimizing the overall energy consumption [1]. Unfortunately, this could lead to violation of time constraints. For real-time systems, both energy consumption and performance are important design considerations. Furthermore, many embedded systems are functioning under a dynamic load, with the number and nature of active tasks varying over time (multi-mode systems). As a result, in order to meet both the security and timing constrains, and at the same time minimize the energy consumption, it is important to perform careful system optimization, taking all these aspects into account at early design phases.

RTNS 2013, October 16 - 18 2013, Sophia Antipolis, France

http://dx.doi.org/10.1145/2516821.2516830.

In the past decade, FPGA-based reconfigurable platforms have been widely used in embedded systems for pursuing both higher performance and flexibility [18]. Modern FPGAs provide support for partial dynamic reconfiguration [26], i.e., parts of the device may be reconfigured at run time, while the other parts remain fully functional. Considering these advantages, FPGAs have been applied in a variety of applications: e.g., performance enhancement [3, 11] and, more recently, low-power and energy-efficient applications [22, 21, 15]. In this paper, we will utilize the available FPGA in the design of energy-efficient secure embedded systems.

Due to the difficulties of designing secure embedded systems under tight resource and timing constraints, only few pioneering papers discussed the security related issues of real-time embedded systems in previous years. In [12], delivering sound security protection under real-time constraints has been studied and validated. But the aspect of multi-mode systems and its implications were not addressed by these contributions. The paper [16] described an automatic hardware-software design flow for secure MPSoCs, but ignored the energy and real-time requirements. The authors in [8] presented a design framework for secure multi-mode embedded systems without considering the actual energy constraints and without using FPGA co-processors. The paper [7] proposed a codesign technique for distributed embedded systems under tight security and real-time constraints using FPGA acceleration for cryptographic operations.

Topics of energy efficiency and dynamic task sets in FPGA-accelerated systems started to attract more attention. The authors in [4] presented an approximation algorithm for energy-efficient task scheduling in heterogeneous systems having two processing units, i.e. a DVFS enabled and a non-DVFS unit. Researches in [14] proposed a warp processor architecture having a main processor and an FPGA for substantially reducing power consumption. However, the multi-modes and security related design aspects were not addressed in either of these two works. A task relocation strategy for FPGA-based systems running dynamic task sets was discussed in [17]. But energy efficiency and security issues were missing. To the best of our knowledge, this is the first work that addresses the design optimization of secure and energy-efficient real-time embedded systems with FPGA co-processors, running dynamic task sets.

2. PRELIMINARIES

2.1 Confidentiality

Confidentiality is the key concept of information security

Table 1: The strength and encryption/decryption time of selected ciphers

RC6 rounds	4	6	8	10	12	14	16
Strength	29	45	61	78	94	110	118
Time/block [μs]	17	26	35	44	52	61	70

that refers to preventing the disclosure of information to unauthorized parties. In the context of communication, it is most often achieved by encrypting/decrypting the sensitive messages using either public-key cryptography or symmetric-key cryptography. In this paper we have chosen the iterated block ciphers (IBCs), a type of symmetric-key cryptography and arguably the most widely used cryptosystems for message encryption/decryption [9].

IBCs transform fixed-size blocks of plaintext into ciphertext blocks of the same size, by repeatedly applying an invertible transformation, with each iteration being referred to as a round. The decryption procedure of IBCs is similar but in reverse order. In this paper, we need to explore the trade-off between protection strength and message encryption/decryption time. We quantify the protection strength of an IBC as the logarithm of the amount of plaintext-ciphertext pairs required to break the IBC using the best known cryptanalysis attack.

For the current work, we studied several IBC algorithms, i.e., RC6, Rijndael, and Twofish, and we selected RC6 for its flexibility and efficiency of providing good confidentiality protection, as presented in [5]. Since the number of rounds can be customized, RC6 is able to provide different levels of confidentiality protection using corresponding amount of execution time. Table 1 illustrates the protection_strength/execution_time trade-off for seven selected variants of RC6, running on a processor at the highest available frequency (see Section 2.2.1). The first row presents the number of employed encryption/decryption rounds; the second row lists the protection strength (as defined above); and the last row gives the corresponding execution time to encrypt/decrypt a block, for each RC6 variant considered. Note that the design framework proposed in this paper is general enough to be applied to other cryptographic algorithms and quantification methods, if similar protection_strength/execution_time relations can be derived.

2.2 Power Model

In this paper, one of our optimization goals is to reduce the energy consumption of a uniprocessor embedded system having an attached FPGA co-processor. We will next present the power models used in the rest of this paper.

2.2.1 Processor

The power consumption of a processor μP (designed with CMOS technology) depends on the processor's state and consists of several components: dynamic ($P_{\mu P}^{Dyn}$), static ($P_{\mu P}^{Stat}$), short circuit and inherent ($P_{\mu P}^{On}$) power consumption. The short circuit power consumption occurs only during signal transitions and is negligible [24]. $P_{\mu P}^{On}$ represents the inherent power cost incurred by keeping the processor on, and is a constant value.

The dynamic power consumed by the processor can be calculated as

$$P_{\mu P}^{Dyn} = C_{eff} V_{dd}^2 f, \tag{1}$$

where C_{eff}, V_{dd} and f denote the effective switching capacitance, supply voltage and clock frequency of the processor, respectively. The dynamic power consumption occurs only when the processor is active (i.e., executing tasks).

The static power does not depend on switching activity, and is consumed due to leakage current, which is mainly a combination of subthreshold conduction (I_{sub}) and reverse bias junction current (I_{ju}) [13]. The static power (consumed both when the processor is active and idle), is given by

$$P_{\mu P}^{Stat} = L_g(V_{dd} I_{sub} + |V_{bs}| I_{ju}), \tag{2}$$

where L_g is the number of logic gates in the circuit and V_{bs} is the voltage applied between the body and the source of a transistor. As shown in [13], the subthreshold leakage current can be approximated with the following expression

$$I_{sub} \approx K_1 e^{K_2 V_{dd}} e^{K_3 V_{bs}}, \tag{3}$$

where K_1, K_2 and K_3 are constant fitting technology dependent parameters.

The V_{dd}^2 in Eq. 1 and V_{dd} in Eq. 2 indicate that reducing the supply voltage (while proportionately reducing the clock frequency f) is the most effective way to decrease the energy consumption (for processors that are not switched off or to low power states during idling periods). This method is known as dynamic voltage/frequency scaling (DVFS). The dependency of maximum operating frequency on supply voltage [13] is given by

$$f = \frac{((K_4 + 1)V_{dd} + K_5 V_{bs} - v_{th1})^{\alpha}}{K_6 L_d}, \tag{4}$$

where K_4, K_5, K_6 and v_{th1} are technology dependent coefficients, L_d is the logic depth and α is a measure of velocity saturation. In this paper, we consider that the discrete pairs (V_{dd}, f) at which the processor can run are given.

2.2.2 FPGA

The power consumption of an FPGA device is composed of dynamic power (P_{FPGA}^{Dyn}), due to switching activity when the FPGA is executing the hardware modules mapped on it, and static power (P_{FPGA}^{Stat}), independent of switching activity. The static power consists of device static power (P_{FPGA}^{Dev}), which represents the leakage power when the device is powered but not configured, and design static power (P_{FPGA}^{Des}), representing the additional power when the device is configured but there is no switching activity [25].

Power estimation for designs on FPGAs, e.g., the Xilinx Virtex families, can be done in several ways. The XPower tool [25] uses a corresponding capacitance model for every element in a design (e.g., LUT, RAM, I/O, wire, etc.), and then derives the dynamic power consumption using information about the circuit's switching activity, which can be generated by timing simulation. Another method is to use spread-sheet-table based methods [25]. In this case, the designer provides information about the number and type of resources used by a certain design, e.g., CLBs, RAM, I/O etc., and obtains a raw estimation of the power consumption.

3. SYSTEM MODEL

3.1 Architecture Model

An example architecture is depicted in Fig. 1. We consider a uniprocessor platform that has an FPGA co-processor with shared memory. The system uses a communication module to communicate (by wire or wireless) with other peers or service centers. The processor supports DVFS, i.e., the supply voltage (and implicitly the frequency) of the processor can be selected at run time from a discrete set, depending on the concrete actual requirements. If the

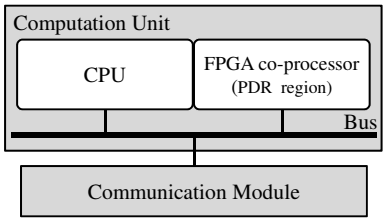

Figure 1: The architecture model

real-time constraints are tight, the voltage (and frequency) could be increased in order to reduce the execution time of the tasks. If, on the contrary, the real-time requirements are relaxed, the voltage (and frequency) could be reduced in order to lower the energy consumption (see Section 2.2.1). In most cases, the tasks implemented on FPGA are faster, and consume less energy compared to their software implementations running on a general purpose processor [23]. Moreover, new techniques, e.g., [20], can be used to further reduce the power consumption on FPGAs. So the FPGA co-processor is used for both accelerating the execution of real-time tasks and minimizing the total energy consumption.

Modern FPGA families, like the Xilinx Virtex or Altera Stratix, provide support for partial dynamic reconfiguration: i.e., parts of the device may be reconfigured at run time, while the other parts remain fully functional. This offers great flexibility, allowing customization of the hardware platform according to the system requirements. One scenario often employed for current reconfigurable platforms is that the FPGA is partitioned into a static and a partially dynamically reconfigurable (PDR) region. The static region hosts a microprocessor, a reconfiguration controller (that takes care of reconfiguring the PDR region), and potentially other peripheral modules that need not change at run time. The PDR region is organized as reconfigurable slots (composed of heterogeneous configurable tiles), where hardware modules can be reconfigured at run time [10]. We refer to the processor and the PDR region (where the co-processor resides) as the computation unit (see Fig. 1).

3.2 Application Model

The set of all tasks that might occur in the system, $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$, is given, and the set of active tasks is dynamically changing at run time, defining the current *mode* $M \subseteq \mathcal{T}$. The complete set of modes for a given system is the power set of $\mathcal{T}$, denoted with $\mathcal{M}$, having cardinality $|\mathcal{M}| = 2^{|\mathcal{T}|}$. However, certain modes can be excluded due to functionality constraints. In the rest of the paper, we are interested only in the modes that can occur at run time, and we shall refer to them as *functional* modes $\mathcal{M}^{func} \subseteq \mathcal{M}$. Mode M is called a *supermode* of M' if $M, M' \in \mathcal{M}$ and $M' \subset M$. Similarly, M' is called a *submode* of M. The sets of all supermodes and submodes of M are denoted with $\overline{\mathcal{M}}(M)$ and $\underline{\mathcal{M}}(M)$, respectively. The mode containing all the tasks in $\mathcal{T}$ is referred to as the *root* mode. Functional modes that do not have any functional supermodes are called *top functional* modes, denoted by $\mathcal{M}_{\uparrow}^{func}$.

We assume that the tasks in $\mathcal{T}$ are preemptable and periodic, and their executions are independent (i.e., they do not have any precedence constraints or data dependencies). A task τ_i in mode M has a set of design attributes ($\mathcal{W}_i$, $\mathcal{P}_i$, $\mathcal{L}_i$, $\mathcal{F}_i$). For any task τ_i, we know its worst-case execution time (WCET) $\mathcal{W}_i$ at the highest available processor frequency f_{MAX}. Thus, the WCET of τ_i at the current frequency f can be obtained from

$$\mathcal{W}_i^{\mu P}(f) = \frac{\mathcal{W}_i \cdot f_{MAX}}{f}. \tag{5}$$

$\mathcal{P}_i$ is the release period of τ_i and also its relative deadline. $\mathcal{L}_i$ is the set of messages via which task τ_i interacts with the outside world. Each message $m_{ij} \in \mathcal{L}_i$ is associated with a length l_{ij} (in number of RC6 blocks), a weight w_{ij} representing its relative importance (criticality) and a minimal level of confidentiality requirement, QoC_{ij}^{MIN} (see Section 3.3). The quadruple $\mathcal{F}_i = (\mathcal{F}_i^a, \mathcal{F}_i^s, \mathcal{F}_i^{dp}, \mathcal{F}_i^{ap})$ represents the FPGA related properties, denoting the area consumption (expressed in number of reconfigurable slots), relative execution speedup (over $\mathcal{W}_i$), design static power, and active (dynamic) power of τ_i, respectively, if it is implemented on the FPGA co-processor.

A task can be mapped to the processor or the FPGA co-processor. The task mapping of a task is given by

$$Map(\tau_i) : \tau_i \rightarrow \{\mu P, FPGA\}. \tag{6}$$

If τ_i is mapped on the processor, then the processor utilization due to the execution of τ_i, together with the potential encryption/decryption of communication messages, is

$$U_{\tau_i}(f) = \frac{\mathcal{W}_i^{\mu P}(f)}{\mathcal{P}_i} + \sum_{m_{ij} \in \mathcal{L}_i} \frac{K_{ij}(f)}{\mathcal{P}_i}, \tag{7}$$

where $K_{ij}(f)$ is the encryption/decryption time (at the given processor frequency f) for message m_{ij} using the chosen RC6 variant C_{ij}

$$K_{ij}(f) = \frac{l_{ij} \cdot \mathcal{W}_{C_{ij}}^{\mu P} \cdot f_{MAX}}{f}. \tag{8}$$

$\mathcal{W}_{C_{ij}}^{\mu P}$ is the corresponding WCET (measured at the highest processor frequency f_{MAX}) of the selected IBC variant C_{ij} (retrieved from Table 1) for encrypting/decrypting one block of message m_{ij}. Note that l_{ij} represents the length of message m_{ij}, in number of RC6 blocks.

If τ_i is mapped on FPGA, then the active load $ACT(\tau_i)$ of the FPGA module implementing τ_i can be calculated from

$$ACT(\tau_i) = \frac{U_{\tau_i}(f_{MAX})}{\mathcal{F}_i^s}. \tag{9}$$

$ACT(\tau_i)$ indicates the fraction of time when the FPGA module used by τ_i is in the active state. Note that $U_{\tau_i}(f_{MAX})$ represents the processor utilization measured at the highest available frequency f_{MAX}, and $\mathcal{F}_i^s$ represents the relative execution speedup obtained by implementing task τ_i on the FPGA.

3.3 Quality of Confidentiality

We define the quality of confidentiality (QoC) protection for message m_{ij}, encrypted/decrypted with RC6 variant C_{ij}, as

$$QoC_{m_{ij}} = \frac{e^{\text{Strength}(C_{ij})/MAX} - 1}{e - 1} \tag{10}$$

where MAX is the highest protection strength value available for a system, e.g., 118 in Table 1. Then the QoC delivered by the whole system in mode M is defined as

$$QoC_M = \frac{\sum_{\tau_i \in M} \sum_{m_{ij} \in \mathcal{L}_i} w_{ij} \cdot QoC_{m_{ij}}}{\sum_{\tau_i \in M} \sum_{m_{ij} \in \mathcal{L}_i} w_{ij}}, \tag{11}$$

where $\mathcal{L}_i$ is the set of all messages over which task τ_i interacts with the environment, and w_{ij} is the importance (criticality) weight of m_{ij} as described in the previous section.

In addition, we assume a security monitor that determines at run time the system-wide security requirement, QoC^R, based on the current status of the system and the threat level from the environment.

3.4 Scheduling

Let us denote the tasks mapped on the processor in mode M with $\mathcal{T}_M^{\mu P} = \{\tau_i \in M | Map(\tau_i) = \mu P\}$, and similarly the tasked mapped on the FPGA with $\mathcal{T}_M^{FPGA} = \{\tau_i \in M | Map(\tau_i) = FPGA\}$. The tasks mapped on the processor are scheduled using the earliest-deadline-first (EDF) policy: a set of tasks is schedulable by EDF if and only if the total utilization of the tasks is no more than 100%. The utilization $U_{\tau_i}(f)$ of a task τ_i at a certain processor frequency f was defined in Eq. 7. Thereby, at frequency f, the schedulability of a certain mode M can be examined with

$$U_M(f) = \sum_{\tau_i \in \mathcal{T}_M^{\mu P}} U_{\tau_i}(f) \leq 1, \tag{12}$$

The tasks mapped on the FPGA co-processor can run in parallel. Thus, the schedulability condition for the FPGA co-processor reduces to the requirement that the active load of each FPGA module should be smaller or equal to 1, namely

$$ACT(\tau_i) \leq 1, \forall \tau_i \in \mathcal{T}_M^{FPGA}, \tag{13}$$

and the hardware modules mapped on the FPGA should fit in the number of available PDR slots, namely

$$\sum_{\tau_i \in \mathcal{T}_M^{FPGA}} \mathcal{F}_i^a \leq \mathcal{F}_{total}^a, \tag{14}$$

where $\mathcal{F}_i^a$ is the area consumption of τ_i and $\mathcal{F}_{total}^a$ is the total amount of available FPGA area, expressed in number of reconfigurable slots (see Section 3).

3.5 Average Power Consumption

The power consumed by the processor in a certain mode M is calculated as follows,

$$\begin{aligned} P_M^{\mu P} &= U_M(f) \cdot P_{\mu P}^{Dyn} + P_{\mu P}^{Stat} + P_{\mu P}^{On} \\ &= U_M(f) \cdot C_{eff}^M V_{dd}^2 f + P_{\mu P}^{On} + \\ &+ L_g \left(V_{dd} K_1 e^{K_2 V_{dd}} e^{K_3 V_{bs}} + |V_{bs}| I_{ju} \right) \end{aligned} \tag{15}$$

Note that the dynamic power is consumed by the processor in mode M only when it is actively executing tasks, i.e., in the fraction of time given by its utilization $U_M(f)$ at frequency f (Eq. 12). Thus, this directly reflects the energy consumed by the processor in mode M.

In a mode M, the FPGA area is occupied by tasks (implemented as hardware modules) $\mathcal{T}_M^{FPGA}$, as defined in Section 3.4. If the FPGA co-processor is switched on, then it consumes the device static power P_{FPGA}^{Dev} (see Section 2.2.2) all the time, regardless of the hardware modules present on the FPGA and their switching activity. The hardware modules (implementing tasks $\tau_i \in \mathcal{T}_M^{FPGA}$) configured on the FPGA in mode M generate additional design static power consumption $P_{FPGA}^{Des}(\tau_i)$. The extra power consumption from the user logic utilization and switching activity is captured by $P_{FPGA}^{Dyn}(\tau_i)$, and it is consumed only in the fraction of time when τ_i is operating, i.e., $ACT(\tau_i)$ (Eq. 9). Thus, the long term average power consumption (also reflecting the energy consumption) of the FPGA in mode M is given by

$$\begin{aligned} P_M^{FPGA} &= P_{FPGA}^{Stat} + P_{FPGA}^{Dyn} \\ &= P_{FPGA}^{Dev} + \sum_{\tau_i \in \mathcal{T}_M^{FPGA}} P_{FPGA}^{Des}(\tau_i) + \\ &+ \sum_{\tau_i \in \mathcal{T}_M^{FPGA}} ACT(\tau_i) \cdot P_{FPGA}^{Dyn}(\tau_i) \\ &= P_{FPGA}^{Dev} + \sum_{\tau_i \in \mathcal{T}_M^{FPGA}} \left(\mathcal{F}_i^{dp} + ACT(\tau_i) \cdot \mathcal{F}_i^{ap} \right) \end{aligned} \tag{16}$$

The total average power consumption (which directly reflects the energy consumption) of the system is

$$P_M = P_M^{\mu P} + P_M^{FPGA}. \tag{17}$$

One of our design optimization objectives is to *minimize* P_M. For better illustration purposes in later sections, we convert this objective into *maximizing* the system power saving (P_M^{save}), with respect to the maximal average power budget of the system, P^{MAX}. The objective becomes

$$\max P_M^{save} = \max(P^{MAX} - P_M). \tag{18}$$

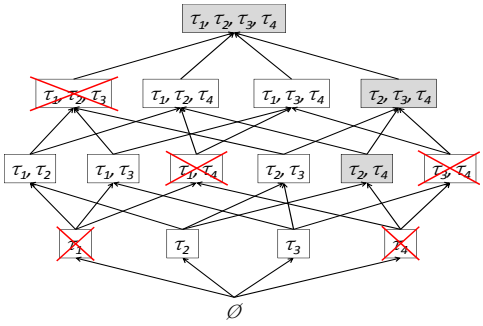

Figure 2: The Hasse diagram of all potential modes

Table 2: Task attributes for the example

τ_i	$\mathcal{W}_i$	$\mathcal{P}_i$	$\mathcal{L}_i$	$\mathcal{F}_i^a$	$\mathcal{F}_i^s$	$\mathcal{F}_i^{dp}$	$\mathcal{F}_i^{ap}$
τ_1	800	3200	$\{m_{11}\}$	18	5	0.4	0.9
τ_2	1100	3000	$\{m_{21}\}$	15	5	0.3	0.7
τ_3	400	2000	$\emptyset$	6	4	0.1	0.3
τ_4	600	1900	$\{m_{41}\}$	11	3	0.3	0.4

4. MOTIVATIONAL EXAMPLE

Let us now consider the architecture model depicted in Fig. 1, composed of a computation unit responsible for executing the tasks, and a communication module that handles all incoming and outgoing messages (as described in Section 3.1). The maximal power budget for the system is $P^{MAX} = 3W$. Four application tasks, $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$, may occur in the system at run time. The corresponding partial order capturing the relations of all possible modes is presented as the Hasse diagram in Fig. 2. The functionally excluded modes, e.g., $M^{123} = \{\tau_1, \tau_2, \tau_3\}$, $M^{14} = \{\tau_1, \tau_4\}$, $M^{34} = \{\tau_3, \tau_4\}$, $M^1 = \{\tau_1\}$ and $M^4 = \{\tau_4\}$, are marked with crosses. All the other modes may occur during system execution. The task attributes are listed in Table 2. The messages m_{11}, m_{21}, m_{41} have lengths (expressed in number of RC6 blocks) $l_{11} = 16$, $l_{21} = 8$, $l_{41} = 8$ and criticality weights $w_{11} = 0.6$, $w_{21} = 0.7$, $w_{41} = 0.4$, respectively.

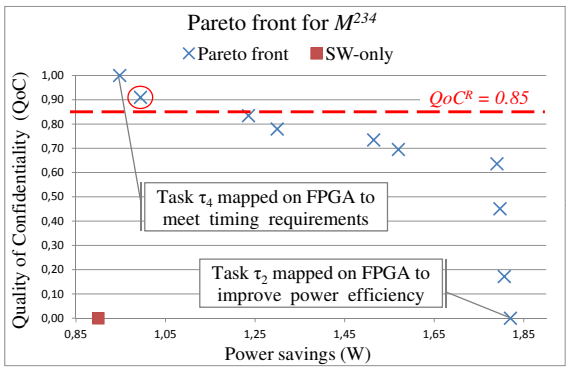

Figure 3: Pareto front for mode M^{234}

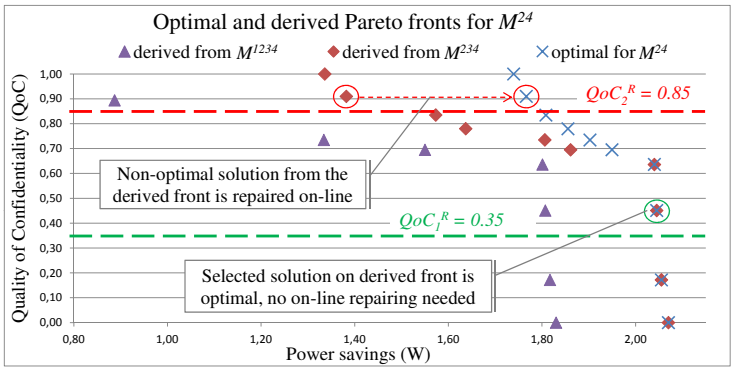

Figure 4: The derived Pareto fronts for mode M^{24}

Let us assume that the system is currently in mode $M^{234} = \{\tau_2, \tau_3, \tau_4\}$. Assuming that no security protection is needed, and considering a software-only solution (i.e., all tasks mapped on the processor), the system utilization at the highest available frequency ($f_{MAX} = 762MHz, V_{dd} = 1.8V$) is $U_{M^{234}}(f_{MAX}) = \sum_{\tau \in M^{234}} U_\tau(f_{MAX}) = 0.88$. Thus, we can scale down the frequency to $f = 650MHz$ ($V_{dd} = 1.6V$), yielding a processor utilization of $U_{M^{234}}(f) = 0.99$ and power savings $P^{save}_{M^{234}} = 0.9W$ (with respect to the maximal power budget $P^{MAX} = 3W$). In Fig. 3, this point is represented with a square marker while the Pareto front for mode M^{234} is represented with crosses. All the solutions on the Pareto front have at least one task mapped to hardware. It is interesting to note that, for the case with no security requirements, although the system is schedulable purely on the processor, the use of FPGA co-processor gives more energy-efficient solutions. In this particular case, task τ_2 is mapped on the FPGA, while tasks τ_3 and τ_4 run on the processor at the lowest frequency available, i.e., $f = 427MHz (V_{dd} = 1.2V)$, yielding a power saving $P^{save}_{M^{234}} = 1.82W$ (more than twice the saving obtained with the software only solution).

Let us now consider the case when maximum security protection is needed. In this case, a software-only solution would yield a processor utilization at the highest possible frequency of $U_{M^{234}}(f_{MAX}) = 1.36$. Thus, it is impossible to schedule the system on the processor only, and the FPGA acceleration is needed in order to fulfill the timing requirements. By mapping task τ_4 on the FPGA, and running tasks τ_2 and τ_3 on the processor at a frequency $f = 595MHz (V_{dd} = 1.5V)$, the system will satisfy all deadlines, and the power consumption will be minimal. The two scenarios outlined above show that the FPGA acceleration is essential both for accelerating the applications with high security requirements (increased message encryption/decryption load) under tight deadlines, and for obtaining more energy-efficient solutions.

As also shown in Fig. 3, the two cases discussed above represent two extreme scenarios: confidentiality is delivered either at the lowest or at the highest possible level. Thus, when the system is minimally loaded, bigger power savings can be obtained; at the other extreme, for a maximally loaded system, there is little room to optimize the energy consumption. In reality, there exist many different scenarios in between, and requirements for the system vary at run time, depending on the current threat level from the environment. Thus, we are facing a multi-objective optimization problem, that tries to provide maximal security protection with minimal energy consumption and, at the same time, satisfy the schedulability constraints. The solutions to this problem are captured by a Pareto front.

By inspecting the Pareto front for mode M^{234} (Fig. 3), the trade-off between security protection and power consumption can be observed: solutions with low security requirements consume less power (thus the savings are bigger), and, as the security requirements increase, the power consumption also increases (thus the savings are smaller). It is interesting to note that for low security requirements, up to a point ($QoC \approx 0.6$ in Fig. 3) we can obtain significant increases in quality of confidentiality with small power losses. This is due to the fact that we can increase the encryption/decryption strength for the tasks mapped on the FPGA, and this is very efficient. For higher security requirements, we need to increase the encryption/decryption strength for the tasks mapped on the processor as well, and this is done with higher power expenses.

At run time, depending on the threat level, the security monitor will set a security requirement for the system, e.g., $QoC^R = 0.85$. Assuming that the Pareto front for the current mode is stored in memory, the solution satisfying the security requirement that generates the biggest power savings can directly be chosen. For our example, we would choose the solution marked with a circle in Fig. 3, with $QoC = 0.91 \geq QoC^R$ and power savings $P^{save}_{M^{234}} = 1W$.

Since we assume dynamic task sets, the application might change mode at run time. Let us consider the situation when the system switches to mode $M^{24} = \{\tau_2, \tau_4\}$. If the Pareto front for the new mode is saved in memory, the procedure described in the above paragraph would be applied. Unfortunately, not all Pareto fronts for all the possible modes are available. There are two reasons for this: 1) the run time memory constraints only allow the storage of a limited number of Pareto fronts; 2) the number of modes is exponential in the number of tasks, so it is impossible to explore and generate at design time the Pareto fronts for all the modes, for large designs. Because of these limitations, we will discuss next how to extrapolate a good solution for a mode, based on the Pareto fronts of its implemented supermodes. Let us assume that modes M^{234} and M^{1234} are implemented, i.e., they have their Pareto fronts stored in memory. Thus, for our example, the implemented supermodes of M^{24} are $\overline{\mathcal{M}}(M^{24}) \cap \mathcal{M}^{impl} = \{M^{234}, M^{1234}\}$. The current mode M^{24} and its implemented supermodes are illustrated with shading in Fig. 2.

We obtain a derived Pareto front by freeing the resources occupied by the tasks that are not active in the current mode. For example, for mode M^{1234}, we disregard tasks $M^{1234} \backslash M^{24} = \{\tau_1, \tau_3\}$. Fig. 4 presents the Pareto fronts for the considered mode (not saved in memory), as well as the derived fronts from its two implemented supermodes. Once we derive on-line the fronts from both supermodes of M^{24},

we pick the front derived from M^{234} because it provides higher quality solutions. As can be seen in Fig. 4, the solutions derived from M^{234} (marked with red rhombuses) dominate the ones derived from M^{1234} (marked with purple triangles).

Once we selected one derived front, we need to select a solution for a particular security requirement. Considering a requirement $QoC_1^R = 0.35$, we would choose the solution with $QoC = 0.45$ and power savings $P_{M^{24}}^{save} = 2.05W$ (the overlapping rhombus and cross, circled with green), which is identical to the optimal solution from the Pareto front of mode M^{24} (task τ_2 mapped on FPGA, processor frequency $f = 427MHz$ and supply voltage level $V_{dd} = 1.2V$). For a security requirement $QoC_2^R = 0.85$, as can be seen from the figure, the solution on the derived front (the rhombus circled with red) is not optimal. This is due to the fact that the extra resources freed (occupied by task τ_3 in mode M^{234}), are not optimally used. Let us elaborate more on this: since τ_3 is mapped on the processor in mode M^{234}, the solution on the derived front has processor utilization lower than 1 (in this case $U = 0.71$), at a frequency $f = 595MHz(V_{dd} = 1.5V)$ which is unnecessarily high, and generates a power saving of only $P_{M^{24}}^{save} = 1.38W$. Thus, we apply a quick online procedure to improve this solution obtained from the derived front. We scale down the frequency of the processor to the minimum value available $f = 427MHz(V_{dd} = 1.2V)$, bringing the utilization as close as possible to 1 ($U = 0.96$), in order to reduce the power consumption. In the example discussed above, we manage to recover the optimal solution (the cross circled with red), yielding $P_{M^{24}}^{save} = 1.77W$ (task τ_4 mapped on FPGA, processor frequency $f = 427MHz$ and supply voltage level $V_{dd} = 1.2V$).

The methods to obtain the Pareto fronts at design time, as well as the methods to select, at run time, an efficient solution that satisfies the security requirements, will be presented in Section 6.

5. PROBLEM FORMULATION

Our global optimization goal is that, whenever a new mode is entered at run time, or the security requirements for a particular mode change, the system adapts to a new energy-efficient configuration that is schedulable and satisfies the current security constraints. The actual configuration is characterized by the tasks mapped on the FPGA and processor, the voltage/frequency level on the processor, and the security protection level for each message. The problem is decomposed into two sub-problems, namely, design time and run time optimization. At design time, we want to find the optimal solutions to the multi-objective optimization problem for each mode, such that we have solutions satisfying different requirements. Thus, we need to prepare solutions for all functional modes $\mathcal{M}^{func}$ that may occur at run time. However, since there exist $O(2^{|\mathcal{T}|})$ potential modes in $\mathcal{M}^{func}$, we cannot afford to explore all $M \in \mathcal{M}^{func}$ when $|\mathcal{T}|$ becomes large. Therefore, we need to find an efficient method to explore the Hasse diagram, covering only a subset of $\mathcal{M}^{func}$ (depending on the available design time and memory limitation of the hardware platform for storing the generated solutions), and still yielding high quality results. At run time, a new mode $M \in \mathcal{M}^{func}$ can occur randomly, and the system is required to find a energy-efficient configuration that satisfies the QoC requirements quickly.

5.1 Design Time Optimization

At design time, there are two sub-problems to consider. The first is to solve the multi-objective optimization problem for one mode, i.e., maximizing the confidentiality protection of the system (Eq.11) and the long term average power saving (Eq. 18), while meeting the schedulability constraints. The second sub-problem is to explore $\mathcal{M}^{func}$ efficiently depending on the available design time and system memory, and to apply the approach for the first sub-problem on each explored mode.

5.1.1 Optimization for one mode

The optimization problem is over two objectives: QoC (Eq. 11) and long term average power saving (Eq. 18). The optimal solutions to this problem form a Pareto curve on which no solution is dominated by any other[1]. The Pareto solutions are considered to be equally good, but with different emphases. An implementation $\mathcal{I}_M$ for mode M is a subset of the Pareto solutions, that are saved in the system memory.

A solution $s \in \mathcal{I}_M$ contains three design decisions:

- cipher selections C_{ij} for all messages;
- the assigned supply voltage V_{dd} and corresponding frequency f of the processor for mode M; and
- a task partitioning of all tasks in M between the processor and the FPGA co-processor.

A solution is *feasible* if the assigned V_{dd} is available in the system, no task misses its deadline, i.e., Eq. 12 and 13 must be satisfied, and the FPGA area constraint is not violated, i.e., Eq. 14 must be satisfied.

The inputs for this problem are the active tasks in mode M and their attributes $\tau_i(\mathcal{W}_i, \mathcal{P}_i, \mathcal{L}_i, \mathcal{F}_i)$, the message attributes $m_{ij}(l_{ij}, w_{ij}, QoC_{ij}^{MIN})$ for all $m_{ij} \in \mathcal{L}_i$, and the FPGA related properties ($\mathcal{F}_i^a$, $\mathcal{F}_i^s$, $\mathcal{F}_i^{dp}$, $\mathcal{F}_i^{ap}$) for all $\mathcal{F}_i$ (see Section 3.2). A designer provided protection_strength/ execution_time trade-off table for selected cryptographic algorithms (similar to Table 1) is also required. The desired output is the implementation $\mathcal{I}_M$, consisting of a set of solutions from the Pareto front for M, with respect to the two optimization objectives, i.e., maximization of QoC and average power saving.

5.1.2 Optimization for the whole system

There can be up to $O(2^{|\mathcal{T}|})$ different modes that may occur at run time. Our concrete optimization objective is to solve the aforementioned problem, i.e., find the Pareto fronts, for all $M \in \mathcal{M}^{func}$. Thereby, whenever the system switches into a new mode, or the security requirement changes for a particular mode, the system can adapt to the best solution selected from the saved Pareto fronts depending on the run time requirements. The ideal scenario is that we can prepare at design time the Pareto fronts for all functional modes $\mathcal{M}^{func}$. Unfortunately, due to both time and memory constraints, this might not be possible for large systems. In such cases, our run time policy will use the best Pareto front derived from the supermodes of the current mode in order to choose a solution (see Sec. 5.2).

Before going further, let us introduce a relation between two implementations $\mathcal{I}_M$ and $\mathcal{I}'_M$ of mode M: we say that $\mathcal{I}_M$ *outperforms* $\mathcal{I}'_M$ if and only if $\mathcal{H}(\mathcal{I}_M) > \mathcal{H}(\mathcal{I}'_M)$, where $\mathcal{H}(\mathcal{I})$ represents the hypervolume metric for $\mathcal{I}$, computed as shown in [27]. For a mode $M \notin \mathcal{M}^{impl}$, there is no implementation $\mathcal{I}_M$ saved in memory. Then we refer to the derived implementation obtained from $M' \in (\overline{\mathcal{M}}(M) \cap \mathcal{M}^{impl})$

[1]A solution is dominated if there exists at least one other (dominating) solution that performs better in both optimization objectives.

that gives the highest hypervolume after removing the resources occupied by the tasks $\tau_i \in M' \backslash M$ as the *derived implementation* $\mathcal{I}_M^{M'}$ for mode M. $\mathcal{I}_M^{M'}$ has the following properties,

$$\mathcal{H}(\mathcal{I}_M^{M'}) \geq \mathcal{H}(\mathcal{I}_M^{M''}), \forall M'' \in (\overline{\mathcal{M}}(M) \cap \mathcal{M}^{impl}). \quad (19)$$

Now let us define the characteristic hypervolume of a mode M, denoted with $\mathcal{H}_M$:

$$\mathcal{H}_M = \begin{cases} \mathcal{H}(\mathcal{I}_M) & \text{if } M \in \mathcal{M}^{impl} \\ \mathcal{H}(\mathcal{I}_M^{M'}) & \text{otherwise} \end{cases} \quad (20)$$

The inputs for this second problem are the set of functional modes $\mathcal{M}^{func}$ and the top functional modes $\mathcal{M}_{\uparrow}^{func}$. The top functional modes must be implemented, since they have no supermodes which could be used for deriving an implementation. The output is represented by implementations of selected modes, denoted with $\mathcal{M}^{impl} \subseteq \mathcal{M}^{func}$. The objective for this second step is to generate $\mathcal{M}^{impl}$, under the given run time memory and available optimization time constraints, such that the total hypervolume H of all functional nodes is maximized, i.e.,

$$\max H = \sum_{M \in \mathcal{M}^{func}} \mathcal{H}_M \quad (21)$$

5.2 Run Time Optimization

At run time, we need to find an appropriate solution for the current mode M, which satisfies the confidentiality requirement QoC^R imposed by the security monitor, and maximizes the long term average power saving (thus implicitly minimizing the energy). More precisely, at the stage of a mode change, or when the security requirement changes, we want to quickly adapt the system with a solution s, based on the available implementations $\{\mathcal{I}_M | M \in \mathcal{M}^{impl}\}$ stored in memory. The selected solution s is desired to deliver a confidentiality protection QoC_s no less than the security constraints QoC^R received from the security monitor, while maximizing the long term average power saving.

If $M \in \mathcal{M}^{impl}$, then s can be directly selected from $\mathcal{I}_M$ that is available in memory. Otherwise, we need to find a good solution s, derived from that implemented supermode of M giving the highest hypervolume on the derived solution front. However, due to the sub-optimality of the derived solutions, the delivered QoC_s and power saving P_s^{save} may, potentially, be improved. So further optimization needs to be performed in order to improve the efficiency of the solution.

6. PROPOSED TECHNIQUES

6.1 Design Time Optimization

Due to the huge computational complexity of the first sub-problem (Section 5.1.1) for even one mode, it is not affordable to find the whole optimal Pareto front. Thus, we choose the genetic algorithm based multi-objective optimization framework NSGA-II [2] for generating a close-to-optimal Pareto curve for each explored mode. The obtained solutions have to satisfy the schedulability constraints (Eq. 12 and Eq. 13) and the FPGA area constraint (Eq. 14). The optimization parameters of NSGA-II, e.g., population size, number of generations, and mutation rates, are fine-tuned for different problem sizes.

The number of possible functional modes grows exponentially as the number of tasks in the root mode $|\mathcal{T}|$ increases. Therefore, it is indispensable to explore the Hasse diagram

Algorithm 1 Hasse diagram exploration algorithm

```
1: Initialize M^wait := empty, and M^impl, M^skip ← ∅
2: for all M_t ∈ M_↑^func do
3:    I_Mt = NSGA(M_t), and M^impl ← M^impl ∪ {M_t}
4:    Insert all M ∈ M^-(M_t) into M^wait
5: while M^wait ≠ empty do
6:    Pop out M' = head(M^wait)
7:    if M' ∈ M^func \ (M^impl ∪ M^skip) then
8:       for each I_M'' of M'' ∈ (M̄(M') ∩ M^impl) do
9:          Calculate H(I_M'^M'') of derived front I_M'^M''
10:         H^D_MAX = MAX(H(I_M'^M''), H^D_MAX)
11:      I_M' = NSGA(M')
12:      if H(I_M') ≥ H^D_MAX · (1 + λ) then
13:         M^impl ← M^impl ∪ {M'}
14:         Insert all M'' ∈ M^-(M') \ (M^impl ∪ M^skip) into
            M^wait
15:      else
16:         M^skip ← M^skip ∪ {M'} ∪ M_(M')
17:         Remove all M'' ∈ M_(M') from M^wait
```

in an efficient and tunable way. We introduce an improvement factor λ for limiting the depth of exploration. An obtained Pareto front $\mathcal{I}_M$ is saved in memory if it gives more than λ gain over the best derived curve from its implemented supermodes, i.e., $\mathcal{H}(\mathcal{I}_M) \geq \mathcal{H}^D_{MAX} \cdot (1 + \lambda)$, where $\mathcal{H}^D_{MAX} = max(\mathcal{H}(\mathcal{I}_M^{M'}))$, $\forall M' \in (\overline{\mathcal{M}}(M) \cap \mathcal{M}^{impl})$. Otherwise, $\mathcal{I}_M$ is discarded, and all its submodes $\underline{\mathcal{M}}(M)$ will be skipped in the succeeding exploration. The detailed procedure is presented as pseudo-code in Algorithm 1.

Before going further, let us introduce the notation of an *immediate submode* $M' \in \mathcal{M}^-(M)$ of mode M that is a submode of M having strictly one less task, i.e.,

$$M' \in \mathcal{M}^-(M), \text{iff.} M' \in \underline{\mathcal{M}}(M), \text{and} |M'| = |M| - 1 \quad (22)$$

First, we initialize the variables used in the algorithm in Line 1. Then we start the optimization by implementing all top functional modes $M_t \in \mathcal{M}_{\uparrow}^{func}$ using NSGA-II, and insert their immediate submodes $\mathcal{M}^-(M_t)$ into the list to be processed $\mathcal{M}^{wait}$ (Line 2-4). We then process the waiting list $\mathcal{M}^{wait}$ from the first element $M' = head(\mathcal{M}^{wait})$ (Line 5-6). If M' is a functional mode, and is not implemented or skipped, then the algorithm tries to find the best derived curve $\mathcal{I}_{M'}^{M''}$ from its implemented supermodes in Line 7-10. After obtaining $\mathcal{I}_{M'}^{M''}$, the algorithm checks whether saving the Pareto curve returned by NSGA-II (Line 11) gains enough with respect to the improvement factor λ. If yes, M' is implemented, and its immediate submodes $\mathcal{M}^-(M')$ are inserted into $\mathcal{M}^{wait}$. Otherwise, M' and its submodes $\underline{\mathcal{M}}(M')$ are ignored in later mode exploration (Line 12-17). After the algorithm terminates, we obtain a set of implementations $\{\mathcal{I}_M | M \in \mathcal{M}^{impl}\}$ which later is saved in the system memory.

6.2 Run Time Optimization

Let us assume that, at a certain moment during run time, the system is required to switch into a mode M, and the current system-wide confidentiality requirement received from the security monitor is QoC^R. Then the system must be able to adapt to mode M with a good configuration, that is robust against the current security threats and uses the lowest possible power. In fact, two different scenarios can appear at run time, i.e., M is implemented ($M \in \mathcal{M}^{impl}$) and M is not implemented ($M \notin \mathcal{M}^{impl}$), as result of the offline design phase discussed in the previous section.

Algorithm 2 Run time optimization if $M \notin \mathcal{M}^{impl}$ and $s \in \mathcal{I}_M^{M'}$ is found

1: Initialize $\mathcal{T}^{skip} \leftarrow \emptyset$
2: **while** $\mathcal{T}_s^{\mu P} \neq \mathcal{T}^{skip}$ **do**
3: Find $\tau_i \in \mathcal{T}_s^{\mu P} \backslash \mathcal{T}^{skip}$ with the lowest PPAU
4: **if** $\sum_{\tau_j \in \mathcal{T}_s^{FPGA} \cup \{\tau_i\}} \mathcal{F}_j^a > \mathcal{F}_{total}^a$ or Eq. 13 can be violated **then**
5: $\mathcal{T}^{skip} \leftarrow \mathcal{T}^{skip} \cup \{\tau_i\}$
6: **else**
7: $Map(\tau_i) = FPGA$, and reduce V_{dd} until violating Eq. 12
8: **if** $P_{current} > P_s$ **then**
9: $Map(\tau_i) = \mu P$, and **break**
10: **return** the current solution

If $M \in \mathcal{M}^{impl}$, then a set of Pareto solutions $\mathcal{I}_M$ is available in memory. Therefore, we can directly select an operation point $s \in \mathcal{I}_M$ for M that satisfies

$$QoC_s \geq QoC^R \text{ and } P_s^{save} \geq P_{s'}^{save}, \forall s' \in \{k \in \mathcal{I}_M | QoC_k \geq QoC^R\}. \quad (23)$$

However, it is possible that there exists no solution that can satisfy all constraints, i.e., confidentiality constraint QoC^R, schedulability constraints (Eq. 12 and 13), and FPGA area constraint (Eq. 14), at the same time even with the highest supply voltage of the processor and the best mapping of tasks. In such cases, the security monitor will be notified and emergency measures must be taken.

If M is not implemented after the offline design phase, i.e., $M \notin \mathcal{M}^{impl}$, there is no direct solution point available for M in memory. Thus, we need to make use of the existing Pareto curves to quickly adapt the system with a good resource allocation decision. The first step that we do is to find the supermode $M' \in (\overline{\mathcal{M}}(M) \cap \mathcal{M}^{impl})$ that gives the highest hypervolume of the derived curve $\mathcal{I}_M^{M'}$. After that, we try to select an operation point $s \in \mathcal{I}_M^{M'}$ that satisfies $QoC_s \geq QoC^R$, and uses the lowest power. At this point, we face two alternatives:

Case 1

A solution s, satisfying all constraints (confidentiality constraint QoC^R, schedulability constraints Eq. 12 and 13, and FPGA area constraint Eq. 14), is found on the derived implementation. Our proposed technique for this case is described in Alg. 2. It is important to notice that, due to the sub-optimality of $\mathcal{I}_M^{M'}$, s may not deliver the best supply voltage and task mapping strategy, as compared with the one produced using the (unavailable) implementation $\mathcal{I}_M$. So we need to further improve the power consumption, if possible. However, the problem of recovering the real optimal solution from the operational point s on the derived curve is actually a knapsack problem, which is too complex to solve optimally on-line. So we propose a fast greedy method to efficiently find a good implementation at run time.

The problem with the solution s is that there still might be unused resources on the FPGA and the voltage/frequency level of the processor might be unnecessarily high. Exploiting this observation, together with the fact that running a task on the FPGA is more energy-efficient than running it on the processor, we first find the task $\tau_i \in \mathcal{T}_s^{\mu P} = \{\tau \in s | Map(\tau) = \mu P\}$ mapped on the processor that gives the lowest power per area and utilization value $PPAU$ (Line 3 in Alg. 2)

$$PPAU(\tau_i) = \frac{\mathcal{F}_i^{dp} + ACT(\tau_i) \cdot \mathcal{F}_i^{ap}}{\mathcal{F}_i^a \cdot U_{\tau_i}(f)}. \quad (24)$$

If it is possible to migrate τ_i to FPGA without violating the FPGA area constraint Eq. 14, then the processor will be offloaded. Thus, we can now reduce the supply voltage V_{dd} of the processor as much as possible while satisfying all constraints (Lines 6-7). This procedure terminates when no further improvement in the power consumption can be achieved (Lines 8-9).

Algorithm 3 Run time optimization if $M \notin \mathcal{M}^{impl}$ and no s is found on the derived implementation $\mathcal{I}_M^{M'}$

1: Initialize $\mathcal{T}^{skip} \leftarrow \emptyset$
2: Select $s \in \mathcal{I}_M^{M'}$ with the highest QoC_s
3: **while** $M \neq \mathcal{T}^{skip}$ **do**
4: Find unprocessed m_{ij} from $\tau_i \in M \backslash \mathcal{T}^{skip}$ with the highest w_{ij}, set C_{ij} to C_{MAX}
5: **if** $QoC_s \geq QoC^R$ and Eq. 12 & 13 **then**
6: $\mathcal{T}^{skip} \leftarrow \emptyset$, **goto** Alg. 2
7: **if** Eq. 12 is violated **then**
8: **if** $\sum_{\tau_j \in \mathcal{T}_s^{FPGA} \cup \{\tau_i\}} \mathcal{F}_j^a > \mathcal{F}_{total}^a$ **then**
9: $\mathcal{T}^{skip} \leftarrow \mathcal{T}^{skip} \cup \{\tau_i\}$, and restore C_{ij}
10: **else**
11: $Map(\tau_i) = FPGA$
12: **if** Eq. 13 is violated **then**
13: $\mathcal{T}^{skip} \leftarrow \mathcal{T}^{skip} \cup \{\tau_i\}$, restore C_{ij}, and $Map(\tau_i) = \mu P$
14: **if** Eq. 13 is violated **then**
15: $\mathcal{T}^{skip} \leftarrow \mathcal{T}^{skip} \cup \{\tau_i\}$, and restore C_{ij}
16: **if** $\forall m_{ik} \in \tau_i, C_{ik} = C_{MAX}$ **then**
17: $\mathcal{T}^{skip} \leftarrow \mathcal{T}^{skip} \cup \{\tau_i\}$
18: Notify run time security monitor

Case 2

We may face the possibility that there exists no solution s on the derived front $\mathcal{I}_M^{M'}$ satisfying all constraints (confidentiality constraint QoC^R, schedulability constraints Eq. 12 and 13, and FPGA area constraint Eq. 14). Our strategy to address this second scenario is illustrated in Alg. 3. Since $\mathcal{I}_M^{M'}$ is not the real Pareto front for M, but obtained by ignoring the unnecessary tasks in its parent M', it is possible that the constraints can, in fact, be satisfied. Therefore, we try to satisfy the QoC requirement first by selecting the operation point $s \in \mathcal{I}_M^{M'}$ with the highest QoC (Line 2). After that, we find the message m_{ij} having the highest importance factor w_{ij}, and set its encryption or decryption rounds C_{ij} to C_{MAX} (Line 4). If QoC^R is achieved, then we try to reduce the power consumption by reusing the method defined in Alg. 2 (Lines 5-6).

If the above increment of C_{ij} leads to a schedulability violation (Eq. 12) on the processor (Line 7), we check whether we can map τ_i to FPGA instead (Lines 10-12). If this results in a violation of Eq. 13, τ_i is moved back to the processor and its C_{ij} is restored (Lines 12-13). If the increment of C_{ij} is not possible, neither on the processor, nor on the FPGA (Line 14), then we skip processing τ_i (Line 15). The process continues until it satisfies the security requirement, or all tasks are fixed in their mapping but QoC^R is still not satisfied. In the second case, the security monitor is notified (Line 18) and emergency measures must be taken.

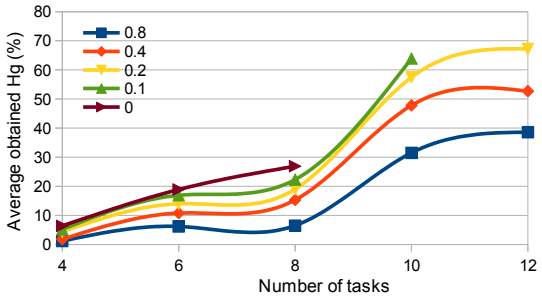

Figure 5: Performance improvement of off-line phase

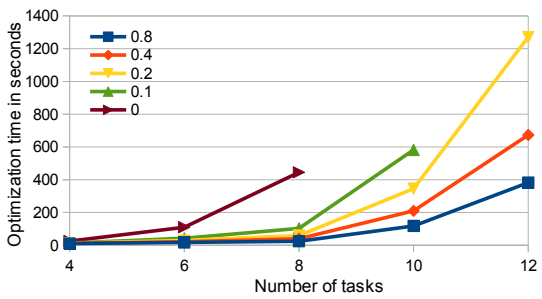

Figure 6: Optimization time of off-line phase

7. EXPERIMENTAL RESULTS

We have carried out experiments on a Linux machine having a quad-core Intel Xeon 2.66GHz CPU and 8GB RAM. We evaluated our proposed design framework on five problem sizes with $|\mathcal{T}| = 4, 6, 8, 10,$ and 12 tasks, respectively. The parameters for NSGA-II were fine tuned for each problem size. On each level, 20 test applications with different task attributes were randomly generated, and each application was designed separately using five improvement factors $\lambda = 0, 0.1, 0.2, 0.4,$ and 0.8, respectively. In the case $\lambda = 0$, the optimal solutions were always guaranteed. However, this requires to explore and implement all modes, thus consuming the longest optimization time and largest run time memory space.

The task execution times and message lengths were generated randomly from the intervals $[500, 1400]$ time units and $[4, 16]$ blocks, respectively. The FPGA area $\mathcal{F}_i^a$ and design static FPGA power $\mathcal{F}_i^{dp}$ of each task were generated based on their execution times with a uniform distribution. The FPGA speed-up $\mathcal{F}_i^s$ and active FPGA power consumption $\mathcal{F}_i^{ap}$ of each task were generated using a normal distribution. The supply voltage V_{dd} to the processor was restricted to seven discrete levels, i.e., $V_{dd} \in \{1.2, 1.3, ..., 1.8\}$. The coefficients used in the power model (see Section 2.2) were set based on the values given in [13]. We set $P_{\mu P}^{On}$ and V_{bs} with conservative values 0.1W and -0.7V, respectively, similar to [6].

We used the same strength/time trade-off table as Table 1. The average power budget (P^{MAX} in Eq. 18) for the five problem sizes were set to 4W, 4W, 5W, 5W, and 5W, respectively.

7.1 Design Time Optimization

We first analyzed our proposed design time optimization techniques on the five problem sizes as mentioned above. For evaluation purposes, we also conducted a baseline study, in which only the top functional modes were implemented, i.e., $\mathcal{M}^{impl} = \mathcal{M}_{\uparrow}^{func}$. In this scenario, only the minimal amount of necessary Pareto curves were saved in memory. Thereby, it delivers the lowest global optimization performance. After that, we evaluated the impact of different λ on the result quality by calculating the obtained total hypervolume gain (Hg^{λ}) of all functional modes $H(\mathcal{M}^{impl})$ over the baseline $H(\mathcal{M}_{\uparrow}^{func})$, i.e.,

$$Hg^{\lambda} = \frac{H(\mathcal{M}^{impl}) - H(\mathcal{M}_{\uparrow}^{func})}{H(\mathcal{M}_{\uparrow}^{func})}, \tag{25}$$

where $H(\mathcal{M}^{impl})$ is the hypervolume of all functional modes, given $\mathcal{M}^{impl}$ obtained using the improvement factor λ. If $M \in \mathcal{M}^{impl}$, we add its hypervolume, otherwise, we add the hypervolume of its best derived supermode. Similarly, we compute $H(\mathcal{M}_{\uparrow}^{func})$.

The average obtained Hg (in percentage) and optimization times of the five problem sizes with different λ are depicted in Fig. 5 and 6, respectively. The x-axis in both figures indicates the number of tasks. The y-axis in Fig 5 shows the average Hg of each experimental setup, while the y-axis in Fig. 6 is the optimization time in seconds. The case $\lambda = 0$ is the most extensive experiment, in which the algorithm goes to the deepest level of the Hasse diagram. Thus, the best solutions are always produced in all modes. However, it does not scale for medium or large designs. So, as can be noticed in both figures, no result is shown for $\lambda = 0$ on levels $|\mathcal{T}| = 10$ and 12, because no experiment terminated within our time-out restriction, i.e., 1500 seconds. For the same reason, the result for $\lambda = 0.1$ for $|\mathcal{T}| = 12$ is not presented either.

It is observable from the two figures that bigger improvement factor λ leads to smaller performance gains over the baseline, but using much less optimization time. In addition, higher performance gains are achieved on bigger problem sizes with the same λ, e.g., $Hg^{0.8}(12) = 38.6\%$ and $Hg^{0.8}(10) = 31.5\%$. Fig. 5 and 6 reveal that the designer can trade-off the desired optimization quality with his tolerable optimization time. However, for large system designs, it is not affordable to explore the possible modes with a small λ. Nevertheless, good solutions can still be obtained with a larger λ using much less optimization time.

7.2 Run Time Optimization

In order to simulate the run time behavior of dynamic systems, we conducted experiments on the five aforementioned problem sizes with the same λ settings. For each experimental setup, namely each test application with a given λ, 20 functional modes were randomly generated for simulating the run time occurrence of modes, together with random security requirements QoC^R for emulating the uncertainty of security constraints. For evaluation purposes, we computed the extra power, consumed in the current mode using the off-line design decisions with respect to λ and the run time security requirement QoC^R, over the optimal power when the Pareto curve of the mode is available in memory.

Fig. 7 presents the average extra power for all experimental settings. In this figure, the results for $\lambda = 0$ are not shown, because the optimal power consumptions are always guaranteed (this corresponds to bars of length 0). As can be seen, the system consumes only 6.3% more power than the optimal solutions on average for all problem sizes if it is designed with $\lambda = 0.1$. While, the average power consumption of $\lambda = 0.8$ for all levels is 18.5% more than the optimum. The smaller λ is used in the design time optimization, the less power is consumed by the system at run time, as can be noticed in Fig. 7. But smaller λ also implies longer optimization time and larger memory space. This also demonstrates how our proposed design framework can be tuned for pursu-

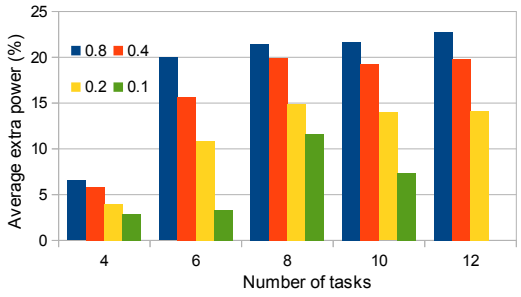

Figure 7: Performance of on-line solution selection

ing better design quality or faster optimization convergence and smaller memory consumption.

8. CONCLUSION

This paper addressed a novel design optimization problem for modern multi-mode embedded systems (running dynamic task sets), in which both energy-efficiency and security are critical. Our design goal is that, no matter what mode or security requirement the system is running in, the minimal energy consumption is ensured. We have used DVFS techniques and on-board FPGA co-processor to obtain significant power savings and, at the same time, meet strict security and timing constraints. Due to the huge complexity of the problem, we proposed an efficient design framework, which is tunable for better design quality or short optimization time, to approach the optimal solutions.

9. REFERENCES

[1] M. Bao et al. On-line Thermal Aware Dynamic Voltage Scaling for Energy Optimization with Frequency/Temperature Dependency Consideration. *Design Automation Conference*, 2009.

[2] K. Deb et al. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *Evolutionary Computation*, 6:182–197, 2002.

[3] C. Huang and F. Vahid. Dynamic Coprocessor Management for FPGA-Enhanced Compute Platforms. *Intl. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, 2008.

[4] C.-M. Hung et al. Energy-Efficient Real-Time Task Scheduling for a DVS System with a Non-DVS Processing Element. *Real-Time Systems Symposium*, 2006.

[5] O. Hyncica et al. Performance Evaluation of Symmetric Cryptography in Embedded Systems. *International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, 2011.

[6] R. Jejurikar et al. Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems. *Design Automation Conference*, 2004.

[7] K. Jiang et al. Co-Design Techniques for Distributed Real-Time Embedded Systems with Communication Security Constraints. *Design, Automation and Test in Europe*, 2012.

[8] K. Jiang et al. Optimization of Secure Embedded Systems with Dynamic Task Sets. *Design, Automation and Test in Europe*, 2013.

[9] L. Knudsen and W. Meier. Correlations in RC6 with a Reduced Number of Rounds. *Fast Software Encryption*, 2001.

[10] M. Koester et al. Design Optimizations for Tiled Partially Reconfigurable Systems. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 19(6): 1048–1061, 2011.

[11] A. Lifa et al. Dynamic Configuration Prefetching Based on Piecewise Linear Prediction. *Design, Automation and Test in Europe*, 2013.

[12] M. Lin et al. Static Security Optimization for Real-Time Systems. *IEEE Trans. on Industrial Informatics (II)*, 22–37, 2009.

[13] S. Martin et al. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads. *Intl. Conf. on Computer-Aided Design*, 2002.

[14] J. Mu and R. Lysecky. Autonomous Hardware/ Software Partitioning and Voltage/Frequency Scaling for Low-Power Embedded Systems. *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, 15(1):2:1–2:20, 2009.

[15] A. Nabina and J. L. Nunez-Yanez. Adaptive voltage scaling in a dynamically reconfigurable fpga-based platform. *ACM Trans. Reconfigurable Technol. Syst.*, 5 (4):20:1–20:22, Dec. 2012.

[16] K. Patel and S. Parameswaran. SHIELD: a Software Hardware Design Methodology for Security and Reliability of MPSoCs. *Design Automation Conference*, 2008.

[17] R. Pellizzoni and M. Caccamo. Adaptive Allocation of Software and Hardware Real-Time Tasks for FPGA-based Embedded Systems. *Real-Time and Embedded Technology and Applications Symposium*, 2006.

[18] M. Platzner et al. *Dynamically Reconfigurable Systems*. Springer, 2010.

[19] S. Ravi et al. Security in Embedded Systems: Design Challenges. *ACM Trans. on Embedded Computing Systems (TECS)*, 3:461–491, 2004.

[20] C. Ravishankar et al. FPGA Power Reduction by Guarded Evaluation Considering Logic Architecture. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (CAD)*, 31(9):1305–1318, 2012.

[21] M. Shafique et al. REMiS: Run-time Energy Minimization Scheme in a Reconfigurable Processor with Dynamic Power-Gated Instruction Set. *International Conference on Computer-Aided Design*, 2009.

[22] L. Shang et al. SLOPES: Hardware/Software Cosynthesis of Low-Power Real-Time Distributed Embedded Systems With Dynamically Reconfigurable FPGAs. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (CAD)*, 26(3): 508–526, 2007.

[23] T. v. Sydow et al. Quantitative Analysis of Embedded FPGA-Architectures for Arithmetic. *Intl. Conf. on Application-specific Systems, Architectures and Processors*, 2006.

[24] H. Veendrick. Short-Circuit Dissipation of Static CMOS Circuitry and its Impact on the Design of Buffer Circuits. *IEEE Journal of Solid-State Circuits (SSC)*, 19(4):468–473, 1984.

[25] Xilinx. XPower Estimator User Guide UG440. 2012.

[26] Xilinx. Partial Reconfiguration User Guide UG702. 2012.

[27] E. Zitzler and L. Thiele. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. *Conference on Parallel Problem Solving from Nature (PPSN V)*, 1998.