Co-Design Techniques for Distributed Real-Time Embedded Systems with Communication Security Constraints

Ke Jiang, Petru Eles, Zebo Peng
Department of Computer and Information Science, Linköping University
{ke.jiang, petru.eles, zebo.peng}@liu.se

***Abstract*—In this paper we consider distributed real-time embedded systems in which confidentiality of the internal communication is critical. We present an approach to efficiently implement cryptographic algorithms by using hardware/software co-design techniques. The objective is to find the minimal hardware overhead and corresponding process mapping for encryption and decryption tasks of the system, so that the confidentiality requirements for the messages transmitted over the internal communication bus are fulfilled, and time constraints are satisfied. Towards this, we formulate the optimization problems using Constraint Logic Programming (CLP), which returns optimal solutions. However, CLP executions are computationally expensive and, hence, efficient heuristics are proposed as an alternative. Extensive experiments demonstrate the efficiency of the proposed heuristic approaches.**

I. Introduction

In the last years, more and more distributed embedded systems (DiES) have been shipped and deployed, especially in safety or reliability critical areas, e.g., automotive electronics and medical control. For example, it is common to find tens of embedded processors (Electronic Control Units or ECUs) in modern vehicles, which are connected by a set of communication infrastructure, e.g., CAN [1] or Flexray [2].

Security is not only a concern for the engineers developing general purpose systems, but is also coming into the scope of the embedded system designers. Although seriously neglected, the need for providing secure embedded systems, especially secure DiES, is emerging rapidly. Moreover, the adoption of new interfaces, e.g., Wi-Fi, dramatically increases potential security threats [3], [4]. Therefore, designing robust DiES against malicious snooping on internal communication becomes a pressing topic. Nevertheless, currently the internal communication of automotive electronic systems is completely unencrypted [5].

The most important component of security in the context of DiES is confidentiality. The internal communication of DiES needs to be protected against malicious eavesdropping due to e.g. privacy concerns [6]. In this paper, we will focus on achieving confidentiality protection of the internal communication of DiES under real-time constraints using reconfigurable hardware, i.e. Field-Programmable Gate Arrays (FPGAs).

Previous work on DiES networks mainly focused on protocols and applications, while potential security risks were seriously overlooked. Take the automotive area, for example. Most of the works, e.g., [7], [8], [9], on communication security dealt with external communication, e.g., vehicle to vehicle (V2V) and vehicle to infrastructure (V2I) communication. Works considering security of internal communication, e.g., [5], [6], [10], are rare. Wolf et al [5] presented feasible attacks and an abstract cryptographic architecture for automotive networks without studying the actual resource constraints. In [6], the authors described four practically implemented attack scenarios, and bring forward the necessity of applying cryptography to protect the internal bus communication. In [10], the authors proposed a software technique providing the best system-affordable security protection for internal communication, which might be lower than the required level if the system does not have sufficient resources. Schaumont et al [11] presented a cryptographic coprocessor architecture, but they did not analyze internal communication security and real-time requirements.

Researchers have widely explored the use of reconfigurable hardware in computer systems, especially in embedded systems, to realize hard-to-implement applications, e.g., the use of FPGA in fault-tolerance [12] and cryptography implementation [13], [14]. But to the best of our knowledge, this is the first work that deals with real-time DiES design problems with an extra dimension of security using FPGA.

The rest of the paper is organized as follows. Section II introduces related background. Sections III and IV present our system model and an illustrative example, respectively. We formulate the problems in Section V. The proposed techniques and experimental results are presented in Section VI and VII respectively. We conclude the work in Section VIII.

II. Preliminaries

A. Confidentiality requirement of internal communication

The internal communication of DiES is usually carrying important system status or control messages that are critical to the functional correctness. Moreover, the new wireless interfaces, e.g., Wi-Fi and GSM which are usually connected to, and cooperate with the internal communication infrastructures to achieve better reliability and performance, do open up the systems to the outside world and, thus, create the window for potential malicious attacks. Therefore, protecting the internal communication is indispensable for securing the whole system.

Cryptography can be utilized to protect security attributes. However, it requires extra computational overhead. This makes securing the DiES difficult, since the system very often has limited computational capacity and have to function under stringent timing constraints. In order to keep the internal communication secret to unauthorized parties, even if the messages were captured, we need to introduce confidentiality protection for the communication by message encryption.

B. Iterated block ciphers

Cryptography can, by its nature, be divided into two categories, public-key and symmetric-key cryptography. Block ciphers, one type of symmetric-key cryptography, are widely used for encrypting critical information. In this paper, we will focus on arguably the most widely adopted branch of block ciphers, i.e. the iterated block ciphers (IBCs). IBCs are suitable for use in embedded systems because of their high throughput rate and suitability for hardware implementations. In this paper, we assume the use of Advanced Encryption Standard (AES), which is a standardized IBC originally known as Rijndael [15], for encrypting and decrypting the messages, but the techniques are also applicable if other IBCs are required explicitly.

The WCET of an encryption or decryption task of message m_i in IBCs is a linear function of the number of rounds, as in Eq. 1.

$$WCET_{ed_i} = w_{I(ed_i)} + r_{I(ed_i)} * x_i \quad (1)$$

where, ed_i is the encryption or decryption task for message m_i, and I_{ed_i} is the implementation of ed_i that can be software or FPGA. x_i is the number of rounds, and ($w_{I(ed_i)}$, $r_{I(ed_i)}$) depend on I_{ed_i}. The more rounds are used, the longer time the procedure takes and the closer the output is to a random bitstream, which implies more difficult to break. For correct message transmission, the same number of rounds is used by an encryption and decryption (E/D) process pair.

C. Reconfigurable hardware

It is common to implement some functions or part of the system in hardware when designing embedded systems to enhance performance or meet constraints. FPGAs can be reconfigured for different purposes, and are rapidly stepping into embedded system design because of their flexibility. The characteristics of FPGAs seem to match well with the core operations of iterated block ciphers, i.e. bit-substitution, XOR and lookup table operations. And due to the fact that the life cycle of cryptographic algorithms is significantly lower than the life time of most DiESs (which might easily be used for 20 years), we can always flash the FPGAs with up-to-date algorithms when needed without changing the hardware. Furthermore, FPGA implementations can exploit the potential parallelism of IBCs.

III. Application and System model

In this work, we capture an application S as a directed acyclic process graph $G(V, E, M)$ that is mapped to an execution platform constituted of a set of computation nodes interconnected by a communication bus. V is the set of non-preemptible processes, and E contains the edges of the graph. An edge $e_i \in E$ from v_s to v_t indicates that v_t depends on v_s in the execution flow. The mapping from the processes to processing resources is given by a function $F : V \to P$, where $P = \{p_1, p_2, ..., p_n\}$ is the set of computation nodes (the bus is also considered as a computation node). The node that process v_i is mapped to for execution can be traced by $F(v_i)$. A processor, which runs software, can cooperate and share memory with FPGA coprocessors. They together form a computation node. $m_i \in M$ indicates a message that is to be sent over the bus. The messages in M need to be protected by encryption, and are depicted as black dots on the edges in the process graphs. The application must complete its execution before an end-to-end delay D, known as the global deadline.

Fig. 1 depicts an application graph and its mapping on the execution platform. The hatched area beside each embedded processor represents the attached FPGA unit. Additionally, Xilinx [16] has shipped FPGA devices that allow modification of only a part of the gate array (called partial dynamic reconfiguration capability, abbr. as PDR). This capability offers even more flexibilities to the system designers, since such PDR enabled FPGAs can have part of the device reconfigured at runtime, while the rest keeps operational. Meanwhile, the FPGAs without PDR capability can only be reconfigured offline. We will consider both kinds of FPGAs in this work.

Since the AES decryption performs similar operations as the encryption, but in reverse order, (and replaces several inner operations, e.g., *SHIFTROWS*, with their inverse operations), the required amount of FPGA area and the execution times of AES E/D processes are roughly the same. Therefore, we assume that the area overhead of implementing one E/D process in FPGA is 1 unit. Consequently, the extra hardware overhead introduced into the whole system is the total FPGA units added.

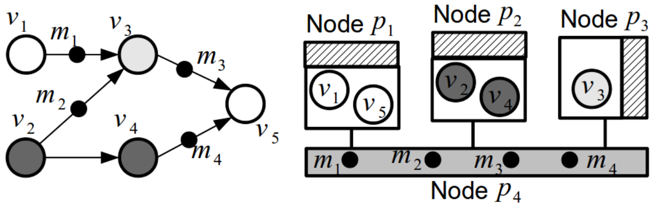

Fig. 1. A simple application

In this example, $V = \{v_1, v_2, v_3, v_4, v_5\}$, $E = \{e_1, e_2, e_3, e_4, e_5\}$, and $M = \{m_1, m_2, m_3, m_4\}$. Process $\{v_1, v_5\}$, $\{v_2, v_4\}$ and $\{v_3\}$ are mapped to the ECU on computation nodes p_1, p_2 and p_3 respectively. The communication messages $\{m_1, m_2, m_3, m_4\}$ are transferred over the bus, denoted as p_4. The worst case execution times (WCETs) of the processes are {60, 180, 50, 140, 150}, and the worst case transmission times (WCTTs) of the messages are assumed to be 20 time units. The system is constrained by a global deadline D of 600 time units.

IV. Motivational Examples

In order to make the internal communication confidential, we need to encrypt the messages before sending, and decrypt them after receiving on another node. The E/D processes can be implemented in hardware or software. However, implementing all cryptographic operations in software very likely cannot provide sufficient protection and at the same time, satisfy the imposed time constraints.

Let us consider the system in Fig. 1. Assuming that all the messages are plaintext (no E/D operations were performed), the corresponding schedule is shown in Fig. 2(a). The end to end delay is 490 units and, as can be observed, we have some time slacks that can be used to perform E/D operations on the messages. The WCETs of E/D processes are determined by the function in Eq. 1. In this system, the designated rounds x_i is 10 for all messages, and ($w_{I(ed_i)}$, $r_{I(ed_i)}$) of software and FPGA implemented E/D processes are (4, 10) and (2, 4) respectively.

If we do not introduce FPGA units into the system, we have to carry out all E/D tasks in software on the ECUs. In this case, in order to satisfy the deadline of 600, we can only encrypt the messages with 3 rounds, which is not a satisfactory solution. The schedule can be found in Fig. 2(b). A simple approach that tries to reach the required number of 10 rounds, while respecting all constraints, is to add new FPGA units for all E/D tasks. This leads to an end-to-end delay of 594, and hardware overhead of 8. Fig. 2(c) illustrates the corresponding schedule.

If we look closer at the schedule in Fig. 2(c), we can find that the encryption operations of m_2 and m_4 can share the same FPGA unit. The same goes for decryption operations of m_1 and m_2. In addition, we can save one extra FPGA unit by implementing the encryption operation of m_1 in software. By this, we save three FPGA units without sacrificing the protection level, as can be observed in Fig. 2(d). With this solution, the system is able to encrypt all messages with the required 10 rounds, assisted by 5 FPGA units. In fact, this is the optimal solution, and is the solution that we are interested in, if FPGAs with static configuration are used.

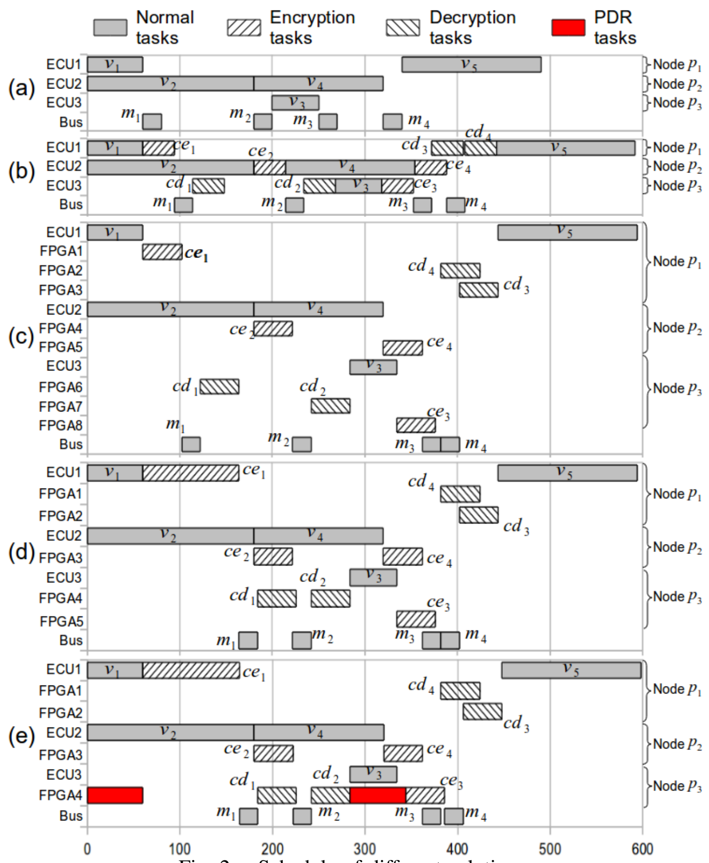
Fig. 2. Schedule of different solutions

Solutions	Rounds	HW overhead	Schedule
No E/D	0	0	Fig.2(a)
Software	3	0	Fig.2(b)
Straight-forward	10	8	Fig.2(c)
Static configuration	10	5	Fig.2(d)
PDR enabled	10	4	Fig.2(e)

TABLE I
RESULTS OF DIFFERENT SOLUTIONS

Furthermore, if we use PDR enabled FPGA, we can also achieve the same level of security using even less hardware area. The reconfiguration time ρ is assumed to be 60 units, and the schedule is illustrated in Fig. 2(e). This solution is what we want to achieve if PDR-enabled FPGAs are used. The FPGAs on p_1 and p_2 only undertake E/D processes of the same kind (all are encryptions, or all are decryptions), so they do not need to be reconfigured at run time. While, the FPGA unit on p_3 must be reconfigured dynamically (depicted as red rectangles), since it is shared by both kinds of cryptographic operations. The numbers of achieved rounds and hardware overheads of the above solutions are listed in Table I.

V. Problem Formulation

The application is modeled as a process graph $G(V, E, M)$ that is mapped to an execution platform as stated in Section III. The WCETs and WCTTs of ordinary processes and messages are known. The values of software and FPGA implemented $w_{I(ed_i)}$ and $r_{I(ed_i)}$ are given. The hardware cost implied by an E/D process is considered to be 1 unit. The application is constrained by an end-to-end delay D, and is required to meet a specific confidentiality protection level, that is given by the set $X = \{x_1, x_2, ..., x_n\}$. As previously mentioned, x_i is the number of rounds the chosen IBC uses for encrypting message

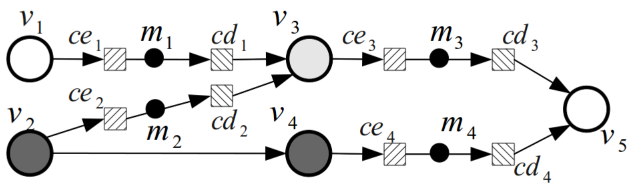
Fig. 3. The updated process graph

m_i. Different messages can have different security demands reflecting their importance.

We are interested in finding the minimal total number of FPGA units N which need to be added into the system, so that the system reaches the designated confidentiality protection, and, at the same time, the real-time and resource constraints are satisfied. Therefore, the objective is to find the appropriate implementation for all E/D processes and the corresponding system schedule that lead to the minimal hardware overhead. In this paper, we consider two different FPGA resources, i.e. FPGA with static configuration and with PDR capability. Therefore, two solutions to the above problem are proposed in the next section.

VI. Proposed Techniques

In order to make the system more flexible to be scheduled and analyzed, we explicitly represent the message E/D operations as independent processes. Such new processes are mapped to the same computation nodes as their parents (the predecessor of an encryption task, or the successor of a decryption task), and can both be allocated on the ECU or an FPGA unit. Fig. 3 represents the updated application corresponding to the one in Fig. 1. For example, the encryption operation of message m_1 is explicitly captured as ce_1, and can be mapped to the ECU or an FPGA unit on p_1.

We first formulate the design optimization problem using constraint logic programming (CLP). CLP allows a user to formulate the problem as a process of constraint satisfaction using a set of clauses containing the constraints. Then the program is passed to the solver that tries to find the optimal solution using various methods, such as branch and bound search. However, solving CLP problems is computationally expensive. Therefore, we also propose heuristic approaches as an alternative, which can be employed for large designs. As mentioned previously, we consider two FPGA techniques, FPGA with static configuration and FPGA with PDR capability.

A. *FPGAs with Static Configuration*

1) *CLP formulation:*

a) Dependency constraints: This set of constraints captures the structure of the process graph, i.e., a process can only start its execution after all its predecessors terminate.

$$\forall v_i \in V \text{ and } \forall v_j \in Predecessors(v_i),$$
$$StartTime(v_i) \geq StartTime(v_j) + WCET(v_j) \quad (2)$$

b) Implementation constraints: An E/D process is implemented either in software, or in FPGA.

$$sw_{ed_i} + \sum_{j=1}^{U} hw^j_{ed_i} = 1 \quad (3)$$

where, $sw_{ed_i}, hw^j_{ed_i} \in \{0, 1\}$. ed_i is implemented in software if $sw^j_{ed_i} = 1$, or on the jth FPGA unit of $F(ed_i)$ if $hw^j_{ed_i} = 1$. U is a constant number that gives finite number of constraints.

c) Execution time constraints: The non-E/D processes and messages have fixed WCETs and WCTTs. The WCETs of E/D processes depend on their implementations and number of encryption rounds as in Eq. 1.

d) Resource sharing constraints: The encryption and decryption processes on the same computation node cannot share the same FPGA unit. This is formulated as

$$\forall p_i \in P \text{ and } \forall j \in \{1, ..., U\},$$

$$\sum_{ce_s \in \{\text{All E on } p_i\}} hw^j_{ce_s} * \sum_{cd_t \in \{\text{All D on } p_i\}} hw^j_{cd_t} = 0 \tag{4}$$

e) Schedulability constraints: This set includes the schedulability related constraints. First, processes on different ECUs or FPGA units can run in parallel, while the execution of those on the same ECU or FPGA unit must not overlap with each other. And second, as all dependencies have been successfully defined, the deadline constraint can be directly formulated as follows.

$$StartTime(v_{last}) + WCET(v_{last}) \leq D \tag{5}$$

Optimization objective: If a potential FPGA unit on p_i is assigned with at least one E/D process, the corresponding FPGA areas have to be added into the system. Therefore, the optimization objective is to find the minimal number N of FPGA units that need to be added into the system, so that all constraints are satisfied. This can be represented as Eq. 6.

$$N = \sum_{\forall p_i \in P} \sum_{j=1}^{U} N'_j \tag{6}$$

where, $N'_j = \begin{cases} 0, \text{ if } \sum_{\forall ed_k \text{ on } p_i} hw^j_{ed_k} = 0 \\ 1, \text{ otherwise.} \end{cases}$

2) Heuristic:

The CLP solver returns the optimal solution satisfying the constraints outlined above, if a solution exists. In this section, we propose a heuristic approach which solves the problem efficiently, and handles large designs. Our heuristic can be divided into two parts, resource allocation and list scheduling [17]. The former controls the outer co-design problem of FPGA allocation and resource sharing. The latter handles the system scheduling. The pseudocode is shown in Alg. 1.

We first detect the tasks on the critical path of application S (line 2 of Alg. 1). Then FindBestECU(S) returns the ECU carrying the most number of critical (on the critical path) software implemented encryption tasks (or decryption tasks) (line 3). If two ECUs have the same number of critical encryption or decryption tasks, the one with the longest total partial critical path (PCP) length [17] of the critical tasks is chosen. After that, all encryption or decryption tasks on ECU $best$ are moved to a new FPGA unit (line 5-7). As, in this case, an FPGA unit can only be configured for encryption or decryption processes, the function MoveToFPGA(ed) checks whether ed and each FPGA unit belong to the same computation node, and whether the unit is for the same type of operation, i.e. for encryption, or for decryption.

If there is no software implemented critical E/D task, the algorithm tries to find the critical E/Ds on FPGAs, and save them into a list $edList$ (line 9). IF $edList$ is empty, it means that no E/D task is left on the critical path. Hence, the application cannot further be accelerated by hardware implementation of E/D tasks and, thus, the heuristic is terminated. Otherwise, we sort $edList$ in descending order of their PCP lengths, and try to allocate each E/D task $ed \in edList$ in an appropriate implementation (line 14-27). Fig. 4 is an illustrative example for line 14-27.

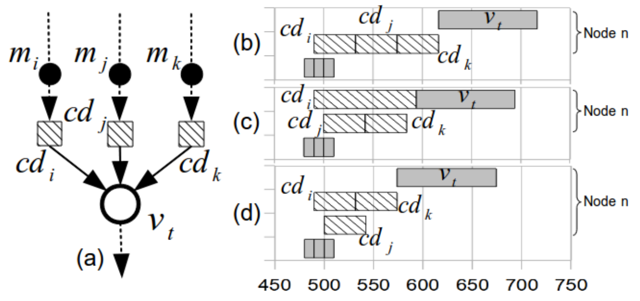

Fig. 4. An illustrative example

The task graph in Fig. 4(a) is part of a bigger system. The WCET of v_t is 100 time units, and WCTT of the messages is 10 time units. $(w_{I(ed_i)}, r_{I(ed_i)})$ of software and FPGA implemented E/D processes are set as (4, 10) and (2, 4) respectively. m_i, m_j and m_k are received at time 490, 500 and 510 respectively. In order for the system to meet the global deadline, v_t must finish its execution before 700 time units. Currently, all the three decryption processes cd_i, cd_j and cd_k are on the critical path, and are allocated on the same FPGA unit leading to the schedule in Fig. 4(b). As can be noticed, v_t finishes at time 716, which means that the global deadline will be violated. The algorithm tries to move cd_i back to the ECU that it originally belongs to. This will lead to an end-to-end delay no longer than the delay sl of the previous system, so the algorithm keeps the current system, and breaks from the inner loop (line 15-17). The schedule can be found in Fig. 4(c). In another case, if v_t must finish before 680 time units, reallocating cd_i will not give a feasible solution, so the algorithm tries to allocate the next process cd_j in $edList$ on an FPGA unit that it was not be mapped to before (line 19), e.g., a new FPGA unit possibly or another existing FPGA unit on the same computation node that is free for cd_j. This leads to the schedule in Fig. 4(d). Then, the algorithm breaks from the inner loop (line 21). Otherwise, it moves cd_j back to the FPGA unit which it was mapped to

Algorithm 1 Heuristic for FPGA with static configuration

```
 1: while ($sl$ = ListScheduling($S$)) $> D$ do
 2:     detect the tasks on the critical path of $S$
 3:     $best$ = FindBestECU($S$)
 4:     if $best \neq$ null then
 5:         for each $ed$ on $best$ do
 6:             MoveToFPGA($ed$)
 7:         end for
 8:     else
 9:         save the critical E/D tasks on FPGA into $edList$
10:         if $edList$ is empty then
11:             return $S$ cannot satisfy the deadline
12:         end if
13:         sort $edList$ in descending order of PCP length
14:         for each $ed$ in $edList$ do
15:             move $ed$ back to the original ECU
16:             if (ListScheduling($S$) $\leq sl$) then
17:                 break
18:             else
19:                 MoveToFPGA($ed$) in $SKIP$ mode
20:                 if ListScheduling($S$) $< sl$ then
21:                     break
22:                 else
23:                     move $ed$ back to the previous FPGA
24:                     release the FPGA if one was added in line 19
25:                 end if
26:             end if
27:         end for
28:     end if
29: end while
```

previously, and release the FPGA unit if a new one was added into the system (line 23-24). The algorithm continues until the system can be scheduled within the global deadline, and returns the solution. Or, it terminates when $edList$ is empty, which means that the system cannot satisfy the deadline constraint while achieving the designated confidentiality protection.

We have two settings that help avoid deadlocks. Each E/D process has a history list keeping track of the FPGAs that it has been implemented on. If it needs to be moved to FPGA in $SKIP$ mode, it avoids being moved to an FPGA unit that is in its history list. And we ignore the E/D tasks that have been moved too many times, and keep them in the ECU or FPGA where it appears the most times. The length of the history list and the value of maximal moves can be tuned for each problem size to reach the best performance.

B. Partial Dynamic Reconfiguration

PDR enabled FPGAs allow reconfiguring a portion of the circuit, while the other parts remain functional. In this case, different types of cryptographic operations, i.e. encryption and decryption processes, can share the same FPGA unit, if there is enough time for reconfiguring in between. The PDR capability increases the flexibility of resource sharing, thereby reducing the hardware overhead.

1) CLP formulation:

The CLP formulation for the PDR enabled FPGAs is to some extent similar with the previous case. In this section, we only focus on the constraints that differ, namely, the resource sharing and reconfiguration constraints. The E/D processes can be implemented on the same FPGA unit. Hereby, the unit needs to be reconfigured if a process is of different kind with the previous one. We define a new kind of tasks, called the PDR tasks, to capture the reconfiguration procedures of the FPGAs. The WCET of the PDR tasks can be 0 or ρ. A PDR task is an empty task if its WCET is 0, or a valid task otherwise. ρ is a constant value given by the designer specifying the partial dynamic reconfiguration time of his FPGA implementation.

- Partial dynamic reconfiguration:
  The FPGA units must be reconfigured on the fly to support resource sharing between encryption and decryption operations. These constraints can be formulated as follows.

$$\forall p_i \in P \text{ and } \forall fp_j \in \{\text{all potential FPGA units on } p_i\},$$
$$\forall ed_s \in \{\text{all E/D tasks on } p_i\},$$
$$ed_t \text{ is the previous task of } ed_s \text{ and } hw_{ed_t}^{fp_j} = 1$$
$$WCET(r_{ed_s}^{fp_j}) = \begin{cases} 0, \text{ if } \begin{cases} hw_{ed_s}^{fp_j} = 0 \\ hw_{ed_s}^{fp_j} = 1 \text{ and } E(ed_s) = E(ed_t) \end{cases} \\ \rho, \text{ if } hw_{ed_s}^{fp_j} = 1 \text{ and } E(ed_s) \neq E(ed_t) \end{cases} \quad (7)$$

  where, $r_{ed_s}^{fp_j}$ is the PDR task for ed_s on potential FPGA unit fp_j, and $E(ed_s) = \begin{cases} 0, \text{ if } ed_s \text{ is an encryption task} \\ 1, \text{ if } ed_s \text{ is a decryption task.} \end{cases}$
- PDR dependencies:
  The executions of E/D tasks and PDR tasks on the same FPGA unit cannot overlap with each other. An E/D task can only start its execution after the module has been successfully reconfigured, which is already captured by dependency constraints.

Optimization objective: The objective is to minimize the total number of FPGA units that must be added into the system as the previous static configuration approach.

2) Heuristic:

The structure of our proposed heuristic for PDR enabled FPGAs is similar to that of Alg. 1. We also try to guide the resource distribution by analyzing the E/D tasks on the critical path. But when looking for the ECU $best$ (line 3), the algorithm counts the number of critical E/D processes of both encryption and decryption, since the E/D tasks can share the same FPGA unit by dynamically reconfiguration. If an E/D task ed needs to be moved to FPGA (the MoveToFPGA(ed) function), the algorithm only checks whether ed and each FPGA unit belong to the same computation node. In addition, we insert a set of new tasks capturing the reconfiguration operations when we schedule the application. In order to avoid deadlocks, the ideas of maximal moves and history lists are reserved from the static configuration approach.

VII. Experimental Results

We have performed experiments on six sets of applications having 10, 15, 20, 40, 80 and 120 processes (excluding E/D tasks) that are mapped to 2, 3, 4, 7, 10 and 15 processors respectively. All experiments were performed on a Linux machine having a four-core Intel Xeon CPU with 2.66GHz frequency and 8GB RAM. In our experiments, we set ($w_{I(ed_i)}$, $r_{I(ed_i)}$) in software and FPGA as (4, 10) and (2, 4) respectively. The messages are assumed to have the same confidentiality requirement of 10 E/D rounds. The CLP experiments were implemented in the *ECLiPSe* constraint programming system [18], and were carried out with a timeout setup of 1800 seconds.

A. FPGA with Static Configuration

Due to the fact that the CLP formulation does not scale even for middle-sized systems, we also compare our heuristic with a greedy straight-forward approach. This approach repeatedly assigns a new FPGA unit to take over the E/D tasks from the ECU that currently has the most number of encryption or decryption tasks. It terminates when the system can be scheduled within the deadline, that is, a solution is found. Otherwise, it terminates when there is no software implemented E/D process left, which means that the system cannot be protected with the designated requirements, while satisfying the time constraints, using this greedy approach.

For each set of applications, we have performed experiments on 20 individual applications that have the same number of non-E/D processes and similar number of bus communications. The average execution times (AETs) of CLP optimization, our heuristic and the greedy approach are presented in Fig. 5, in which the y-axis is in logarithmic scale. For the AET calculation of CLP, we only considered those experiments which terminated before the timeout (for example, in the case of application size 20, only 5 out of 20 CLP runs finished before the timeout). Starting from size 40, no CLP experiment terminated before the timeout.

When comparing the results obtained by two approaches, we calculate the average additional hardware expenditure (AAHE) of one approach over the other. This is formulated as

$$AAHE = \frac{1}{n}\left(\sum_{\forall i \in n} \frac{H_i' - H_i}{H_i}\right) * 100\% \quad (8)$$

where, H_i and H_i' are the obtained results with the two approaches for the same application, and n is the number of valid experiments, i.e. both approaches found solutions. Fig. 6

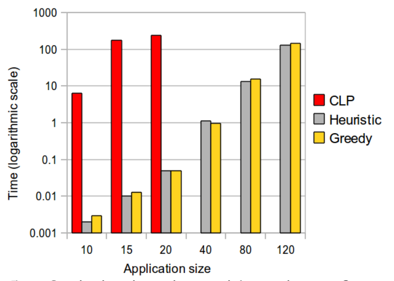

Fig. 5. Optimization time with static configuration

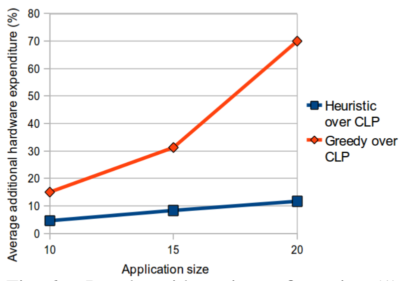

Fig. 6. Results with static configuration (1)

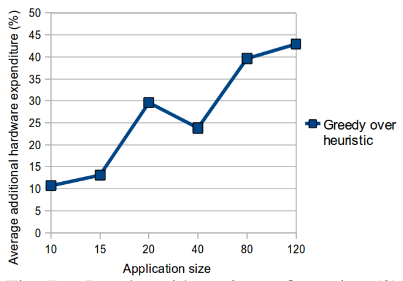

Fig. 7. Results with static configuration (2)

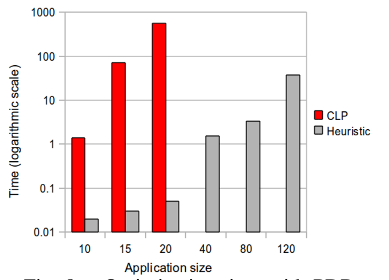

Fig. 8. Optimization time with PDR

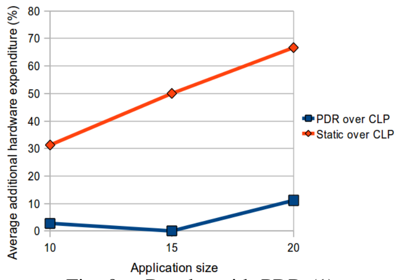

Fig. 9. Results with PDR (1)

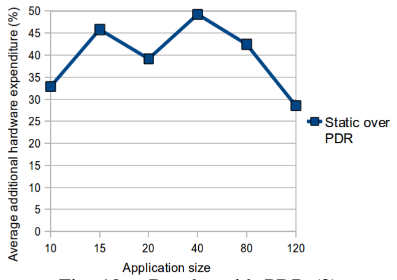

Fig. 10. Results with PDR (2)

presents the AAHE of our proposed heuristic over CLP and of the greedy approach over CLP. Fig. 7 depicts the AAHE of the greedy approach over our proposed heuristic for all application sizes. As can be observed, our heuristic approach provides close results to the optimum. For example, on the largest application size, 20, where CLP did not fail in all experiments, our heuristic only requires about 11% extra FPGA units compared to the optimal solutions. In the meantime, our heuristic approach saves significant amount of FPGA units compared with the greedy straight-forward method.

B. *Partial Dynamic Reconfiguration*

We have conducted the same set of experiments as those in static configuration considering FPGAs with PDR capability. The AET comparison of the CLP and heuristic approach can be found in Fig. 8. Starting from application size 40, no execution time is presented for the CLP formulation, since there is no experiment finished within the timeout.

We use the same analytical method, (Eq. 8) as before, to compare the quality of results. In Fig. 9, we compare the optimal solutions obtained by CLP with the results returned by our heuristic approaches for both FPGA techniques. The experiments that CLP failed to find the optimum within the timeout setup are ignored. The hardware saving using PDR enabled FPGA over static configured FPGA for all application sizes is illustrated in Fig. 10. It can be observed that our heuristic performs well comparing with the optimal results obtained from CLP, and achieves large FPGA units saving compared with static configured FPGAs.

VIII. Conclusion

In this paper, we have presented our optimization techniques for protecting the confidentiality requirements of internal communication using iterated block ciphers. We investigated the problem in two different FPGA techniques which are FPGA with static configuration and PDR capability. We proposed heuristic approaches for solving the problems. The time efficiency and result quality of our heuristics were demonstrated from extensive experiments. This work can be employed in the design of any distributed real-time embedded systems where the internal communication security is required.

References

[1] “CAN in Automation (CiA),” http://www.can-cia.org/.
[2] “FlexRay,” http://www.flexray.com/.
[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, “Experimental Security Analysis of a Modern Automobile,” in *The IEEE Symposium on Security and Privacy*, May 2010.
[4] T. Zellar and N. Mayersohn, “Can a virus hitch a ride in your car?” *New York Times*, March 13, 2005.
[5] M. Wolf, A. Weimerskirch, and C. Paar, “Secure In-Vehicle Communication,” in *Embedded Security in Cars*, 2006, pp. 95–109.
[6] T. Hoppe, S. Kiltz, and J. Dittmann, “Security threats to automotive CAN networks–practical examples and selected short-term countermeasures,” *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11 – 25, 2011, special Issue on Safecomp 2008.
[7] M. Raya and J.-P. Hubaux, “Securing vehicular ad hoc networks,” *Journal of Computer Security*, vol. 15, no. 1, pp. 39–68, 2007.
[8] B. Parno and A. Perrig, “Challenges in Securing Vehicular Networks,” in *Proceedings of Workshop on Hot Topics in Networks (HotNets-IV)*, Nov. 2005.
[9] P. Golle, D. Greene, and J. Staddon, “Detecting and correcting malicious data in VANETs,” in *VANET ’04: Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*. New York, NY, USA: ACM, 2004, pp. 29–37.
[10] K. Jiang, P. Eles, and Z. Peng, “Optimization of message encryption for distributed embedded systems with real-time constraints,” in *14th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, ser. DDECS ’11, april 2011, pp. 243–248.
[11] P. Schaumont and I. Verbauwhede, “Domain-specific codesign for embedded security,” *Computer*, vol. 36, no. 4, pp. 68 – 74, april 2003.
[12] A. Lifa, P. Eles, Z. Peng, and V. Izosimov, “Hardware/software optimization of error detection implementation for real-time embedded systems,” in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES/ISSS ’10. ACM, 2010, pp. 41–50.
[13] A. Elbirt, W. Yip, B. Chetwynd, and C. Paar, “An Fpga-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, no. 4, pp. 545 –557, aug. 2001.
[14] A. Dandalis, V. K. Prasanna, and J. D. P. Rolim, “A comparative study of performance of aes final candidates using fpgas,” in *Proc. Cryptographic Hardware and Embedded Systems Workshop*, ser. CHES 2000. Springer-Verlag, 2000, pp. 125–140.
[15] *The design of Rijndael: AES–the advanced encryption standard*. Springer, 2002.
[16] “Xilinx,” http://www.xilinx.com/.
[17] P. Eles, Z. Peng, P. Pop, and A. Doboli, “Scheduling with Bus Access Optimization for Distributed Embedded Systems,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, no. 5, pp. 472–491, 2000.
[18] *Constraint Logic Programming using ECLiPSe*. Cambridge University Press, 2006.