Security-Driven Task Scheduling under Performance Constraints for MPSoCs with Untrusted 3PIP Cores

Nan Wang, *Member, IEEE,* Song Chen, *Member, IEEE,* Hongqin Zhu, *Member, IEEE,* and Yu Zhu, *Member, IEEE,*

***Abstract*—The high penetration of third-party intellectual property (3PIP) brings a high risk to MPSoCs, and a set of security-driven constraints are imposed on task scheduling to protect MPSoCs against malicious modifications. Due to the significant performance and area overheads incurred, designers often treat these security-driven constraints as loose constraints during the design process, and achieve the design goals within a predefined permissible security risk. In this study, a security-driven task scheduling method is proposed to achieve a high security level under performance constraints, and the number of cores required is also optimized. First, the schedule length is iteratively optimized by assigning sets of data-dependent tasks in the performance constraint-violated paths to the same core, with a maximum weight independent set-based method. Second, the number of cores required is analyzed during vendor assignment according to the distributions of tasks and optimized by maximizing the core sharing of tasks. Finally, tasks are scheduled to time periods using the force-directed scheduling method. Experimental results demonstrate the effectiveness of the proposed method in reducing the number of cores while maintaining system security under performance constraints.**

***Index Terms*—MPSoC, hardware trojan, task scheduling, security, third-party IP core.**

I. Introduction

The increased design productivity requirements for heterogeneous multiprocessor System-on-Chip (MPSoC) require the industry to procure and use the latest commercial-off-the-shelf (COTS) electronic components to track the most cutting edge technology while reducing manufacturing costs. This has given rise to the trend of outsourcing the design and fabrication of 3PIP components, which may not be trustworthy, and the hardware trojans in these 3PIP components present high risks of malicious inclusions and data leakage in products [1]. This raises security concerns [2] because a small hardware modification by an adversary in the 3PIP cores can compromise the whole chip [3]. If such chips run time-critical applications (e.g., in autonomous vehicles), the hardware trojan attack may lead to catastrophic or life-threatening consequences [4]. Similarly, if these chips are used in information critical systems (e.g., banking), the confidentiality and integrity of the user's data can be compromised [5].

Emerging security problems bring an urgent need to detect possible hardware Trojan attacks or mitigate their effects. Methods for detecting hardware Trojans can primarily be classified into the following groups: physical inspection [6], functional testing [7], [8], built-in tests [9], [10], and side-channel analyses [11]–[13]. However, it is impossible to detect advanced hardware trojans, such as A2, due to its trojan insertion stage and software triggered mechanism [14]. To safeguard against potentially undetected trojans, runtime validation approaches provide a last line of defense against trojan attacks and often attempt to contain the effect of an activated trojan [2].

Runtime monitoring techniques always insert hardware sensors in circuits to check abnormal runtime behaviors by monitoring side channel signals or circuit operations [15], [16]. The main challenge in runtime monitoring is that many extra circuit overheads must be added to ensure the monitoring effectiveness and accuracy even though many studies have been proposed to reduce the overheads [3], [17]. However, hardware trojans may still escape these runtime monitoring techniques because too few gates are used, and hardware trojans will not cause any thermal or power fluctuations.

Design-for-trust techniques also provide comprehensive protections to circuits and verify the correctness of system functionality at runtime. Incorporating security constraints in the MPSoC design process is one of the most popular design-for-trust techniques, which can mitigate the effects of the hardware trojans and enable trustworthy computations using untrusted 3PIP cores [18]–[24]. This is achieved by duplicating tasks and mapping them on 3PIP cores of different vendors to detect trojans that alter task outputs or mute potential trojan effects by preventing collusion between malicious 3PIP cores from the same vendor. Designing MPSoCs with these security constraints brings significant design overheads, and researchers have developed a number of solutions and created trusted designs with minimum resource overhead, performance degradation and energy consumption [25]–[27]. Some researchers have started to consider security constraints as loose constraints to satisfy the design goals while maximizing system security [28]–[31].

Existing studies focused on area optimization primarily address the problem in the task scheduling stage [25] because the number of cores required can only be evaluated after vendor assignment. However, vendor assignment is a prior stage of task scheduling, whose results strongly affect area optimization results of task scheduling, and ignoring the area optimization during vendor assignments limits the area optimization results. Also, researchers have treated each communication equally when optimizing the system

This work was supported by the National Key R&D Program of China under Grant 2022YFD2000400.

Nan Wang, Hongqin Zhu and Yu Zhu are with the School of Information Science and Engineering, East China University of Science and Technology, Shanghai, 200237, China.

Song Chen is with the School of Information Science and Technology, University of Science and Technology of China, Hefei, 230026, China.

performance [29]–[31], but removing security constraints from different communications yields different security losses, and communications with larger security importance should have higher protection priorities [28].

In this study, we focus on the design of MPSoCs though security-driven task scheduling, and set the desired performance as constraints. The goal of this study is to find a schedule with a high security level and a small number of cores required. A three-step design method that consists of task clustering, vendor assignment and task scheduling is proposed to enable MPSoC designers to achieve the desired performance by setting it as a constraint, and obtain a high-security design with a small number of cores. To satisfy performance constraints, a maximum weight independent set-based method is proposed to minimize the induced system security risk by iteratively assigning edges to intra-core communications. Furthermore, the numbers of cores are optimized in both the vendor assignment and task scheduling stages, by iteratively assigning tasks that share the most common cores to the same vendor and scheduling these tasks evenly in each time period. Experimental results demonstrate the high quality of the task scheduling results in reducing both system security risks and the number of cores under performance constraints.

The earlier conference version of this paper appeared at [31]. Compared with [31], this paper has a number of new contributions.

1) This study treats communications with different security importance, and provides a schedule with a minimized system security risk.
2) Rather than assigning one task to an IP vendor each time, this vendor assignment method groups all tasks into a number of *vc* (vendor constraint) clusters and assigns an entire cluster to an IP vendor at a time. Also, this vendor assignment method evaluates the number of cores saved when clustering tasks rather than estimating the number of cores required, which speeds up the processing time and provides better results.
3) This study considers core speed variation in the design process. All tasks are first assumed to be performed with the slowest speed, and the vendor assignment will be adjusted after the exact core speeds are determined to further reduce the system security risk.

The remainder of this paper is organized as follows. Section II describes the related literature, and Section III demonstrates the motivations and describes the optimization problem. Section IV presents the details of the proposed task scheduling method. Section V illustrates the experimental results, and Section VI provides the conclusions.

II. Related Work

In general, the IPs procured from third-party vendors are usually not 100% trustworthy. There may be a rogue insider in a 3PIP house who may insert trojan logic in 3PIPs coming out of the IP house, and the outsourced design and test services, as well as electronic design automation software tools supplied by different vendors, also make ICs vulnerable to malicious implants (see Fig. 1) [2].

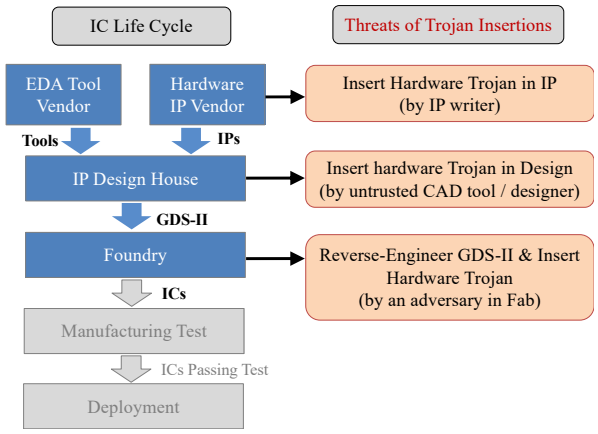


Fig. 1. Hardware trojans inserted at different stages of IC life cycle [2].

A. Security Countermeasures

Numerous and various functional and parametric tests are required to verify whether a 3PIP contains hardware trojans. However, testing a black-box component is difficult and time-consuming, and it is impractical to perform such an exhaustive test for a large and complex design. Therefore, a number of countermeasures have been developed against hardware trojans at the design stage [38]. Hardware security primitives use random number generators (TRNG) [39] or physical unclonable functions (PUF) [40] to provide built-in self-authentication against various threats and vulnerabilities arising at different phases of the IC life cycle [41], [42]. System and architectural protection techniques prevent information leakage through shared resources [43] and build trusted execution environments [44]. Side-channel protection techniques introduce noise or randomization in the software implementation to eliminate side-channel leakage [45], [46]. IP protection techniques use hardware watermarking or steganography to protect an IP against threats [47], [48]. Machine learning-assisted designs provide defenses against security threats or enhance system robustness [49], [50].

Although hardware trojan detection methods are implemented in different design stages, finding all hardware trojans cannot be guaranteed even with the most cutting-edge technologies. However, many applications, such as banking and military systems, have high security requirements [51]. Therefore, trojan-tolerant design methodologies are another way to protect designs from trojan attacks [52].

B. Design-for-Trust

The design-for-trust technique uses strategies at design time to help detect hardware trojans or mute the attack effects at runtime [18], [19]. Many studies have attempted to detect malicious outputs by duplicating 3PIPs and comparing their results and to avoid collusion between parent and child tasks from the same vendor. Incorporating the above design constraints (i.e., security constraints) in the MPSoC design process has attracted the attention of researchers.

Reecee *et al.* [20] identified hardware trojans through comparisons of two similar untrusted designs by testing

TABLE I
COMPARISON WITH RELATED WORKS

Technique	Detect Trojans	Prevent collusion malicious 3PIPs	Recovery	Optimize area	Optimize delay	Optimize power or energy
[18]	Yes	Yes	No	No	No	Yes
[22]	Yes	No	Yes	No	No	No
[25]	Yes	Yes	No	Yes	Loose delay constraints	Loose power constraints
[26]	Yes	Yes	No	No	No	Yes
[27]	Yes	Yes	No	No	Loose delay constraints	Yes
[28]	Yes	Yes	Yes	Loose area constraints	Loose delay constraints	No
[29]	Yes	In most cases	No	No	Tight delay constraints	No
This work	Yes	In most cases	No	Yes	Tight delay constraints	No

functional differences for all possible input combinations. Beaumont *et al.* [21] developed an online trojan detection architecture that implements fragmentation, replication, and voting. Cui *et al.* [22] implemented both trojan detection and fast recovery at runtime for mission-critical applications, using recomputation with IP cores from different vendors. Shatta *et al.* [23] presented methodologies that detect the errors triggered by hardware trojans in 3PIPs using voters, and recover the system by replacing the error.

Security constraints, including duplication and 3PIP vendor diversity, have been recently proposed for hardware trojan protection, and high-level synthesis is the ideal level for incorporating security constraints [24]. However, fulfilling the security constraints in task scheduling on heterogeneous multicore platforms always results significant overheads in system performance, chip area and power, and researchers have started to reduce these overheads of MPSoCs built from 3PIP cores. Rajmohan *et al.* [25] proposed a PSO-based hybrid evolutionary algorithm, and Sengupta *et al.* [26] proposed a bacterial foraging optimization-based design space exploration method to achieve a task schedule with higher security and less hardware overhead. Sun *et al.* [27] minimized the energy consumption while simultaneously protecting the MPSoC against the effects of hardware trojans with security constraints. Cui *et al.* [28] solved the online hardware trojan detection and recovery problem with graph-theory models that minimize the implementation cost of the design budget and area overhead. Liu *et al.* [29] proposed a set of task scheduling methods to reduce the increments of performance and hardware due to security constraints. Wang *et al.* [30], [31] optimized the design budget and system performance with a minimized number of unprotected communications.

The comparison between the proposed technique and other techniques that implement security constraints is provided in Table I. Security constraints cause significant power, delay, and area overheads [18], [22], and many techniques were developed to optimize these overheads, but the optimization space is limited [25]–[28]. Recent researchers treat security constraints as loose constraints (some constraints that can be violated) to achieve tight design targets, but they forget to minimize the induced security risks [29]–[31]. In this study, the proposed method provides solutions with smaller circuit areas under tight performance constraints. Moreover, the proposed approach also optimizes the system security risk, which is ignored by most existing approaches.

III. PRELIMINARIES AND PROBLEM DESCRIPTION

In this section, we present preliminaries and describe the problem considered in this study.

A. Threat Model

Hardware trojan attacks are intended to affect normal circuit operation, potentially with catastrophic consequences in critical applications in the domains of communications, banking, space and military [32]. They can also aim to leak secret information from inside a chip through covert channels or affect the reliability of an IC through undesired process changes that cause device ware-out and long-term reliability issues [33]. In addition, they can be used to assist software attacks by providing hardware back-doors, and make the system operate incorrectly, such as by modifying the scheduling results of a real-time system [34]–[37]. From the perspective of the activation methods, hardware trojans can be classified as either *always-on* or *conditionally triggered*. An always-on trojan is inserted in rarely accessed places and its footprint is kept small. Conditionally triggered trojans hibernate initially, and are activated either by the trojan implanter or by on-chip triggers [29].

In this study, we adopt the same threat model in [27], [29], which primarily focuses on detecting (or mitigating) malicious modifications. The trojan may cause the task running on the malicious 3PIP to either produce incorrect output or generate additional output to trigger trojans in another 3PIP core from the same vendor. As a result, the following two cases can occur at runtime: 1) due to the insertion of the malicious logic into a 3PIP core, the outputs of the infected cores will be altered at some undetectable points; 2) trojans that are distributed on multiple cores to reduce the chance of being detected can also form secret communication paths, and a malicious logic in one core can trigger the trojans in another core using a secret communication channel.

B. Security Constraints

The security constraints improve the design reliability using untrusted 3PIPs [24]–[31], and the effectiveness of the security constraints in detecting the deliberate faults caused by trojans and isolating the triggered trojans are explained in [18]. The two types of security constraints that are commonly used are described as follows.

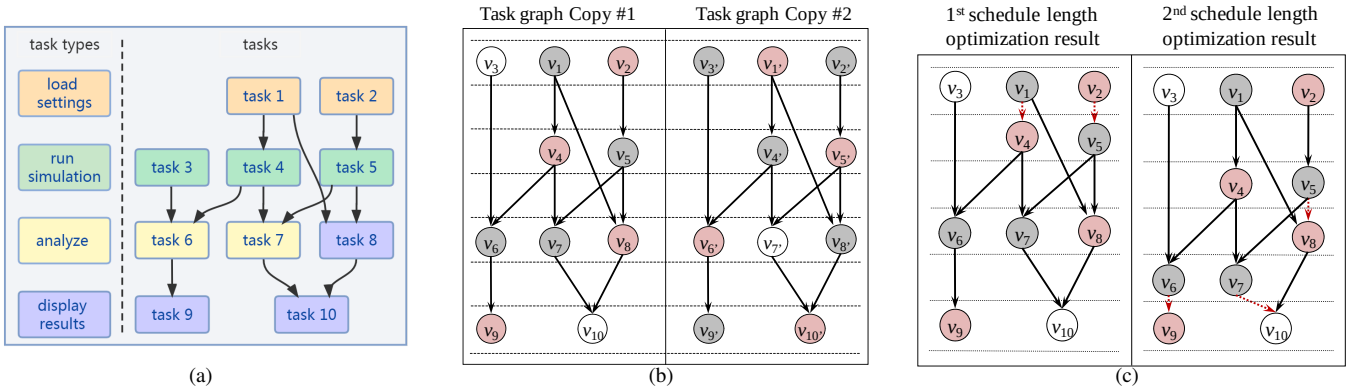


Fig. 2. Example of task graph and its schedules. (a) Task graph. (b) ASAP schedule with security constraints. (c) Schedule length optimization results.

1) Duplication-With-Diversity: Each task is executed in duplicate on the cores from different vendors, and the outputs of these cores are compared by a trusted component (not designed by the third party) to ensure the trustworthiness of the comparison step [53].

Duplication-with-diversity is set based on the fact that the probability of trojans implanted by different attackers having the same trigger input is quite low, and it is virtually impossible that two cores from different IP vendors will output the same tampered results after the same trigger input [54].

2) Isolation-With-Diversity: Data-dependent tasks are executed on the cores fabricated from different IP vendors to isolate the triggered hardware trojans.

To mute Trojan footprints, attackers always distribute Trojans in multiple IP cores and construct secret communications between IP cores to leak information or to trigger the hibernating trojans. These secret communications between IP cores from the same vendor cannot be acquired by other vendors [18]. Although redundant execution approaches, including voting architecture [21], dual/triple modular redundancy [53], and duplication-with-diversity, can detect hardware trojans by comparing the outputs of cores from different vendors with the same input, they cannot cut off secret communications between multiple IP cores. Therefore, this isolation-with-diversity is also introduced to isolate the triggered hardware trojans from the rest of the system.

C. Motivations

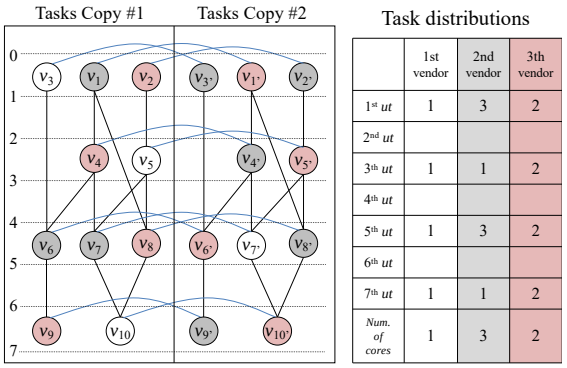
Fulfilling security constraints incurs significant overheads of performance and area, and Fig. 2 shows an example, where 10 tasks are sorted into 4 different types. All intra-core communication delays are ignored, and the computational times of all tasks are assumed to be 1 unit of time (*ut*), which also equals their inter-core communication delays. Fig. 2(a) and Fig. 2(b) show the task graph and its as-soon-as-possible (ASAP) task schedule with security constraints, where tasks colored white, gray, and pink are assigned to the 1st, 2nd, and 3rd vendors, respectively. The duplicated task of v_i is denoted as $v_{i'}$. Satisfying all security constraints makes the schedule length 7 *ut*, and 6 cores from 3 IP vendors are needed.

In this study, we target embedded platforms which execute application-specific tasks and have high security requirements. Such platforms cannot handle tasks that occur suddenly at runtime, and they are widely-used in automotive, safety-critical systems, etc. In such systems, designers always have prior knowledge of the application and its runtime constraints, which requires designers to perform security-driven customizations to meet performance and area requirements [27].

1) System Risks in Performance Optimization: To reduce performance overhead, researchers have imposed loose security constraints during task scheduling and explored the possibility of assigning data-dependent tasks into a single core to hide the inter-core communication delay [29]–[31] within a predefined maximum permissible security. In this study, clustering the connected data-dependent tasks into one core is denoted as **edge contraction**, and the edge that represents an inter-core communication is a **contracted edge**.

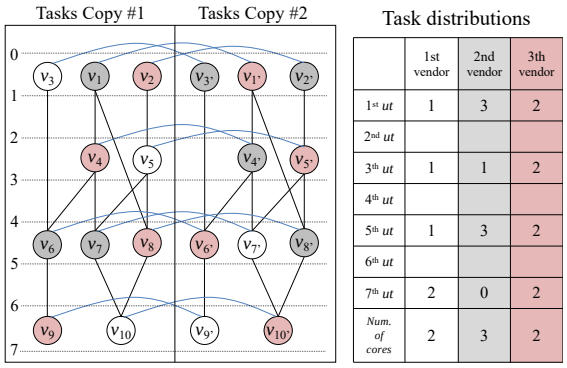
Edge contraction brings risks to the system, and traditional methods that optimize system performance either ignore the consequence security loss [29] or treat every communication with the same security importance [30], [31]. However, different edges face different security risks [25], [28], and the example in Fig. 2(c) shows the reason, where the target is to reduce the schedule length in Fig. 2(b) by 1 *ut*. Traditional methods [30], [31] contract the fewest edges, which are $e_{1,4}$ and $e_{2,5}$ (see the 1st schedule length optimization result in Fig. 2(c)), but attackers may likely modify the system settings to crash the whole system or output their desired results, meaning that edges $e_{1,4}$ and $e_{2,5}$ require high-priority protection. Instead, tasks 8-10 demonstrate the results, and the probabilities of attacking these tasks and related communications as well as the consequence system damages are much lower. Therefore, contracting $e_{5,8}$, $e_{6,9}$, and $e_{7,10}$ causes less security risk (see the 2nd schedule length optimization result in Fig. 2(c)), even though this contracts more edges.

2) Reducing Cores in Vender Assignment: Traditional methods always start to optimize the number of cores after the vendor assignment stage when the number of cores required can be evaluated [27]–[30]. However, the vendor assignment results also determine the number of cores required, and Fig. 3 gives an example. The performance constraint is assumed to be 7 *ut*, and all security constraints are satisfied, which are represented by the blue (*duplication-with-diversity*) and



	1st vendor	2nd vendor	3th vendor
1st ut	1	3	2
2nd ut			
3th ut	1	1	2
4th ut			
5th ut	1	3	2
6th ut			
7th ut	1	1	2
Num. of cores	1	3	2

(a)



	1st vendor	2nd vendor	3th vendor
1st ut	1	3	2
2nd ut			
3th ut	1	1	2
4th ut			
5th ut	1	3	2
6th ut			
7th ut	2	0	2
Num. of cores	2	3	2

(b)

Fig. 3. Example of vendor assignments. (a) Vendor assignment and its ASAP schedule, which requires 6 cores. (b) Vendor assignment and its ASAP schedule, which requires 7 cores.

black (*isolation-with-diversity*) lines between tasks. Fig. 3(a) and Fig. 3(b) give two different vendor assignments and their ASAP schedules. With the vendor assignment given in Fig. 3(a), the scheduling result requires 6 cores, but 7 cores are required with the vendor assignment shown in Fig. 3(b).

D. Problem Description

Clustering data-dependent tasks to reduce schedule length violates *isolation-with-diversity*, leaving the corresponding intra-core communications unprotected. Although incorporating security constraints in the design process cannot guarantee the detection of all hardware trojan attacks, the risks caused by hardware trojans can be significantly reduced. In this study, the security risk of an intra-core communication is regarded as the reduced risk after applying security constraints to this communication, and we assume that different communications face different security risks. The security risk of each communication is determined by the MPSoC designers, which can be evaluated by the method given in [25], and communications facing larger security risks should have higher protection priorities. The problem considered in this study can be described as follows.

***Problem** 1:* The inputs of this problem are the task graph TG, vendor and performance constraints, core speeds of vendors, and security risk of each communication. The target is to find a schedule with the lowest system security risk, and the number of cores required is also optimized.

The system security risk $risk_s$ is regarded as the accumulated security risks of all unprotected communications, which can be calculated as follows:

$$risk_s = \sum_{e \in E_c} risk(e) \tag{1}$$

where E_c is the set of all all unprotected communications, and $risk(e)$ is the security risk that e faces. The notations used in this work are shown in Table II.

TABLE II
DESCRIPTIONS OF NOTATIONS

Notation	Description
v_i	The i-th task, and its duplicated task is $v_{i'}$.
c	The cluster, which consists of a set of tasks.
e_{ij}	The communication from task v_i to task v_j.
$risk(e)$	The security risk hidden in the communication e.
$risk_s$	The total system security risk.
$dly(e)$	The inter-core communication delay of e.
$w(e)$	The evaluated weight of contracting e.
$prob(v_i, t_j)$	The probability that task v_i is executed in time t_j.
$DG(c, t_j)$	The number of tasks in cluster c that are executed in time period t_j.
$DG_{max}(c)$	The maximum of all $DG(c, t_j)$, which estimates the number of cores required for cluster c.
$Merge(c_i, c_j)$	The number of cores reduced after merging clusters c_i and c_j.

IV. Security-Driven Task Scheduling with Performance Constraints

In this section, a three-step task scheduling method is proposed, and both the system security risks and the number of cores required are optimized under performance constraints. The three steps of the proposed method are performance-constrained task clustering, vendor assignment with core minimization, and task scheduling.

A. Performance-Constrained Task Clustering

In this stage, we first apply security constraints to all tasks and communications, and then iteratively assign data-dependent tasks into the same core to meet the performance constraints with an optimized system security risk. Typically, the cores produced by different vendors have different speeds, and the exact speed of each core is not yet determined; thus, we assume that tasks are performed with the slowest speed when optimizing the schedule length. In addition, we discuss only the method of contracting edges in TG, and schedule length optimization of the duplicated task graph TG' can be performed in the same manner.

Source and sink nodes s and t are added to TG, and directed edges that point from s to 0-indegree nodes and from 0-outdegree nodes to t are also added. An example of the task graph from Fig. 2(a) with s and t added is given in Fig. 4(a). The **timing violated graph** ($TVG = (V_T, E_T)$) is then constructed by all paths from s to t whose lengths exceed the performance constraints, and it is an induced subgraph of TG. V_T consists of s, t and all tasks with negative slacks, and $E_T = \{(v_i, v_j) \in E, v_i \in V_T \text{ and } v_j \in V_T\}$. Let $slack(v)$ be the

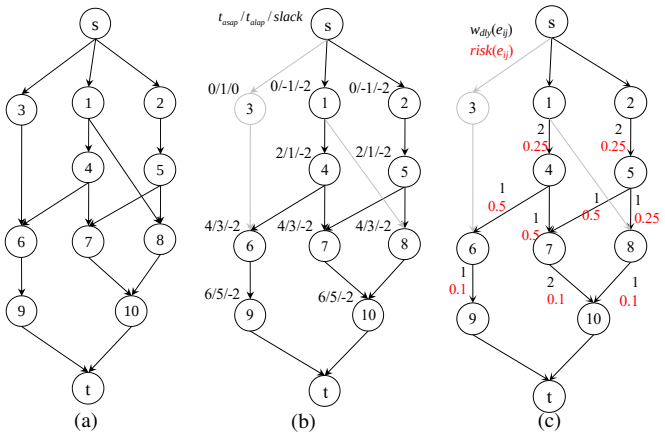


Fig. 4. Example of evaluating the timing violated graph. (a) Task graph with s and t. (b) TVG with a timing constraint of 5 ut. (c) The evaluation of $w_{dly}(e)$.

slack time of v under the performance constraint, which is calculated as follows:

$$slack(v) = t_{alap}(v) - t_{asap}(v) - exec(v) \tag{2}$$

where $exec(v)$ is the execution time of task v, and $t_{asap}(v)$ and $t_{alap}(v)$ are the ASAP and as-late-as-possible (ALAP) schedules, respectively. Fig. 4(b) shows an example of TVG, where the performance constraint is 5 ut and the delay is 1 ut for each edge.

For application-specific IPs, data-dependent tasks might have to be executed by different IP cores, resulting in the corresponding edge being unable to be contracted. Therefore, we only focus on the edges that can be contracted to reduce the schedule length. In the following discussion, we assume that all edges can be contracted for simplicity.

Let $dly(e_{ij})$ be the inter-core communication delay of e_{ij}, and all intra-core communication delays are ignored in this study. Contracting an edge (e_{ij}) reduces the lengths of all paths that pass through e_{ij} by $dly(e_{ij})$. Let $w_{dly}(e_{ij})$ be the sum of the reduced schedule lengths of all paths (from s to t) in TVG after contracting e_{ij}, and it is calculated by the following equation:

$$w_{dly}(e_{ij}) = path_{tvg}(e_{ij}) * dly(e_{ij}) \tag{3}$$

where $path_{tvg}(e_{ij})$ is the number of paths in TVG that pass through e_{ij}.

The target in this stage is to contract the edges with larger $w_{dly}(e_{ij})$, and the increased security risk $risk(e_{ij})$ is smaller. Therefore, the total weight that evaluates an edge e_{ij} contraction in TVG, which is denoted as $w(e_{ij})$, can be calculated as follows:

$$w(e_{ij}) = \frac{w_{dly}(e_{ij})}{risk(e_{ij})} \tag{4}$$

where $risk(e_{ij})$ is the security risk if e_{ij}. Equ 4 shows that we hope to choose the edge with larger $w_{dly}(e_{ij})$, while the increased security risk is smaller, in the task clustering s Fig. 4(c) illustrates the $w_{dly}(e_{ij})$ and $risk(e_{ij})$ in TVG, which are indicated next to the edges.

However, not all edges can be contracted with respect to multicore parallel execution. Let $in\_edge(v)$ be the set of edges that end with v, and let $out\_edge(v)$ be the set of edges that start from v. Edges in TG that belong to the same $in\_edge(v)$ or $out\_edge(v)$ are called **brother edges**. If an edge is contracted during performance optimization, all its brother edges can no longer be contracted. The reason is that contracting brother edges means the tasks that once could be executed parallel in different cores now must be executed sequentially in the same core, and this may result in an increased schedule length. For example, contracting brother edges $e_{4,6}$ and $e_{4,7}$ in Fig. 4(b) makes v_6 and v_7 need to be conducted sequentially in the same core, but they can be computed once concurrently in different cores.

In addition, two edges belonging to the same path in the TVG should not be contracted simultaneously, and this avoids the over-optimization of the path length, which causes additional system security risk. The following example explains the reason. The edges $e_{1,4}$ and $e_{4,7}$ belong to the same path in TVG (refer to Fig. 4(b)), and suppose that contracting either $e_{1,4}$ or $e_{4,7}$ will make the path length smaller than the performance constraint. However, if $e_{1,4}$ and $e_{4,7}$ are both contracted, the path length is over-optimized, which causes an additional system risk increment.

Then, a weighted **edge contraction conflict graph** ($ECCG = (V_E, E_E)$) is constructed to represent whether every pair of edges in TVG can be contracted simultaneously. Each vertex in V_E represents an edge in TVG that can be contracted, and the weight of a vertex in V_E equals the weight of the corresponding edge in TVG. Two vertices in V_E are connected when their corresponding edges cannot be contracted simultaneously, under one of the following two situations:

1) Two edges are brother edges (with respect to the multicore parallel execution);
2) These two edges belong to the same path in TVG (to prevent the over-optimization of path length).

The maximum weight independent set (MWIS) of the weighted $ECCG$ is calculated by the method proposed in [56], and the target is to a set of edges with maximum weight that can be contracted simultaneously. Algorithm 1 shows details of the performance-constrained task clustering algorithm with the goal of minimizing the system security risk. In the first step (*Lines 2-5*), TVG is constructed from TG, and the weights of all edges in TVG are evaluated. In the second step (*Lines 6-10*), the weighted $ECCG$ is built, and its MWIS is calculated. In the third step (*Lines 11-13*), the MWIS-selected edges in TG are contracted. These three steps are iteratively repeated until the performance constraint is satisfied.

With the security risks of communications given in Fig. 4(c), an example of performance-constrained task clustering is shown in Fig. 5, where we are about to optimize the schedule length by 2 ut. The TVG consists of the nodes and edges with black color, and dashed lines are the contracted edges. $ECCG$ is given beneath the corresponding TVG, and the weight of contracting an edge is marked next to the node in $ECCG$. TVG and its corresponding $ECCG$ are constructed (see Fig. 5(a)), and its MWIS is $\{e_{5,8}, e_{6,9}, e_{7,10}\}$ which is contracted in the first iteration. Then, both TVG and $ECCG$ are updated as shown in Fig. 5(b), where $e_{5,7}$ and $e_{8,10}$ are not in $ECCG$ because

Algorithm 1 Task clustering with performance constraint, *task_cluster*(TG, pc).

Input: task graph, TG;
performance constraint, pc.
Output: performance-constrained clustering result, TC.

while $TG.schedule\_length > pc$ **do**
  Construct TVG from TG.
  for each e in TVG **do**
    Calculate $w(e)$;
  end for
  Construct $ECCG$ from TVG;
  for Each node e in $ECCG$ **do**
    $ECCG.node\_weight(e) = w(e)$;
  end for
  Calculate $MWIS$ in $ECCG$;
  for each node e in $MWIS$ **do**
    Contract the corresponding edge e in TG;
  end for
end while

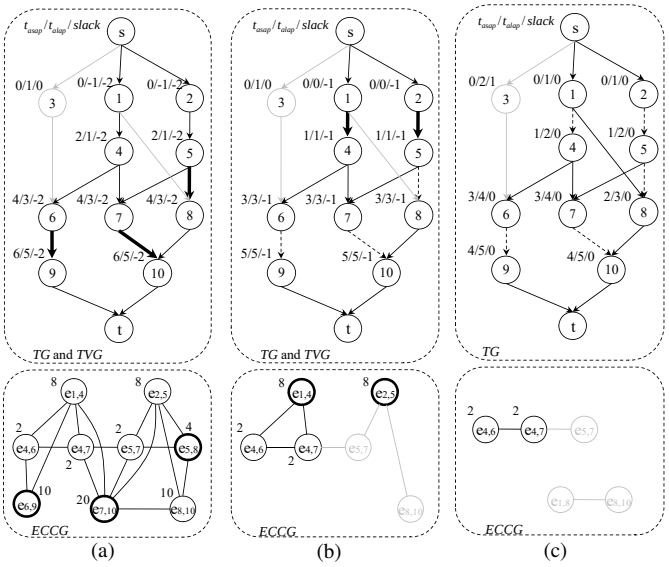


Fig. 5. Example of performance-constrained task clustering procedure. (a) TVG and its corresponding $ECCG$ before task clustering. (b) TVG and its corresponding $ECCG$ after 1st iteration of task clustering. (c) TVG and corresponding VCG after 2nd iteration of task clustering.

their brother edges $e_{5,8}$ and $e_{7,10}$ are already contracted. The MWIS of the current $ECCG$ is $\{e_{1,4}, e_{2,5}\}$, and after contracting these edges, Fig. 5(c) yields the final clustering results, with the performance constraint satisfied.

B. Vendor Assignment with Core Minimization

The principle of vendor assignment is to iteratively cluster tasks into a number of v_c (vendor constraint) clusters, and assign each cluster with an IP vendor according to its core speed. Different from task clustering in the performance-constrained task clustering stage that violates isolation-with-diversity by clustering date-dependent tasks, clustering (also named as **cluster merging**) in vendor assignment follows all security constraints.

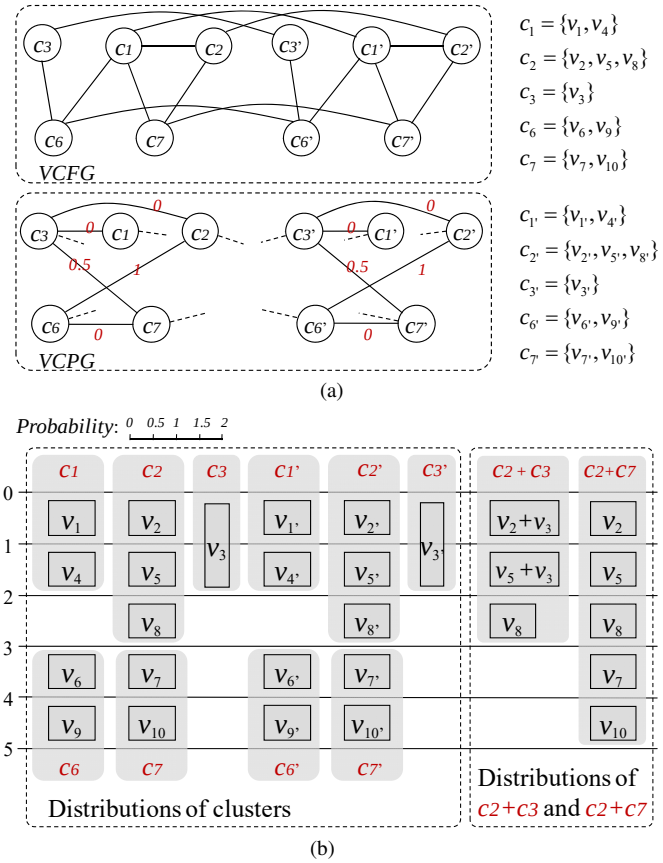


Fig. 6. Example of evaluating vendor compatible graph. (a) $VCFG$ and $VCPG$ derived from task clustering results. (b) Distributions of clusters.

The **vendor conflict graph** ($VCFG = (V_c, E_{cf})$) is constructed from the performance-constrained clustering results, and it represents whether two clusters must be assigned to different vendors. V_c is the set of all clusters from TG and TG', and a cluster is determined by the following two situations: 1) a task that is not connected by any contracted edge is regarded as a cluster; and 2) tasks that are connected to each other by contracted edges are in the same cluster, and the index of this cluster is decided by the minimum index of the tasks in this cluster. E_{cf} is the edge set in $VCFG$, and if two tasks must be assigned to different IP vendors due to security constraints, the two clusters that contain these two tasks will be connected in $VCFG$.

The **vendor compatible graph** ($VCPG = (V_c, E_{cp})$) is the complement graph of $VCFG$, and an edge in E_{cp} indicates that the connected clusters can be assigned to the same vendor. Fig. 6(a) gives examples of $VCFG$ and $VCPG$, which are constructed from the performance-constrained clustering result shown in Fig. 5(c). Because the edges in $VCPG$ are too numerous to demonstrate, we use dashed lines to represent the remaining edges that are connected to this cluster.

Then, the accumulated probability of task concurrency is calculated, which is denoted as *distribution graph* (DG). We assume that the probabilities of a task on all its possible scheduling results are the same [55], and the probability that v_i is executed in time t_j is denoted as $prob(v_i, t_j)$. The summation of the probabilities of all tasks in a cluster c for the time period

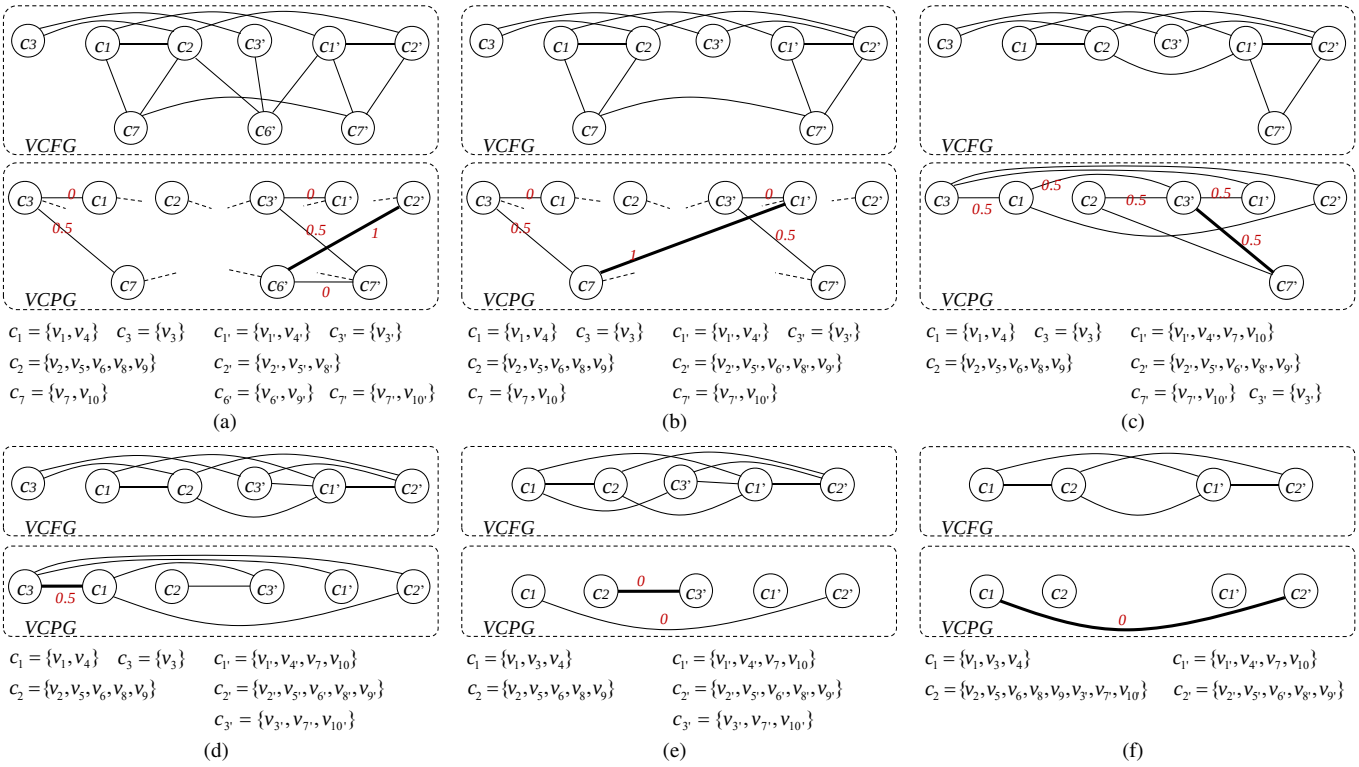


Fig. 7. Example of cluster merging procedure. (a) *VCFG* and *VCPG* in 2nd iteration of cluster merging. (b) *VCFG* and *VCPG* in 3th iteration of cluster merging. (c) *VCFG* and *VCPG* in 4th iteration of cluster merging. (d) *VCFG* and *VCPG* in 5th iteration of cluster merging. (e) *VCFG* and *VCPG* in 6th iteration of cluster merging. (f) *VCFG* and *VCPG* in 7th iteration of cluster merging.

t_j is denoted as $DG(c, t_j)$ and calculated as follows:

$$DG(c, t_j) = \sum_{v_i \in c} prob(v_i, t_j) \tag{5}$$

The maximum of all $DG(c, t_j)$, $\forall t_j \in [1, p_c]$ is denoted as $DG_{max}(c)$, which is used to estimate the number of cores required for all tasks in cluster c. Fig. 6(b) presents the distribution graphs of all clusters, where the width of a task means the probability that this task will be conducted at the corresponding time period.

The number of cores required may be reduced by merging two clusters c_i and c_j, which is denoted as $Merge(c_i, c_j)$, and it can be calculated as follows:

$$Merge(c_i, c_j) = DG_{max}(c_i) + DG_{max}(c_j) - DG_{max}(c_i + c_j) \tag{6}$$

A larger $Merge(c_i, c_j)$ indicates a higher probability that tasks in c_i and c_j can share the same cores, and therefore, assigning these tasks to the same IP vendor reduces the number of cores. Examples of calculating $Merge(c_2, c_3)$ and $Merge(c_2, c_7)$ are shown in Fig. 6(b). $Merge(c_2, c_3) = 1 + 0.5 - 1.5 = 0$, which means that core reduction cannot be achieved by merging c_2 and c_3. $Merge(c_2, c_7) = 1 + 1 - 1 = 1$, indicating that merging c_2 and c_7 can reduce one IP core.

$Merge(c_i, c_j)$ is then set as the weight of edge (c_i, c_j) in *VCPG*. The edge with maximum weight is chosen, and the connected clusters are merged into one; this procedure continues until the number of clusters equals the number of vendors available. Because *VCPG* with $O(n)$ nodes has nearly $O(n^2)$ edges, the maximum weight independent set of *VCPG* is not introduced to determine the clusters to be merged due to its large time complexity.

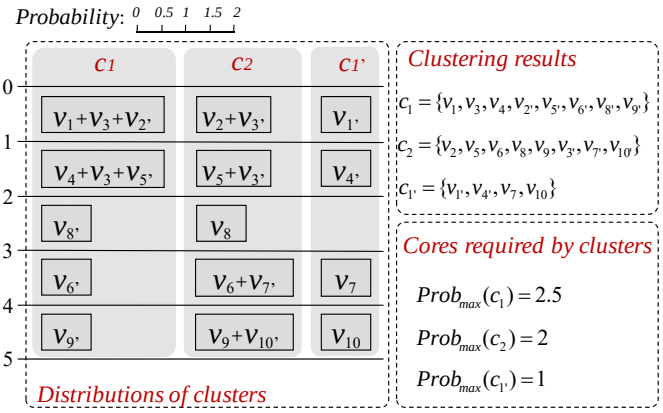


Fig. 8. Cluster merging results under vendor constraint.

An example of the cluster merging procedure is presented in Fig. 7, where the initial *VCFG* and *VCPG* are shown in Fig. 6(a), and the vendor constraint is 3. The maximum weight of all edges in *VCPG* is 1, and c_2 and c_6 in Fig. 6(a) are merged into one cluster, named c_2. All edges that once connected to c_2 and c_6 in *VCFG* now connect to c_2 in the updated *VCFG*, and the weights of edges that connect to c_2 in *VCPG* are also updated. Fig. 7(a) shows *VCFG* and *VCPG* after the 1st iteration of merging clusters. Then, c_2' and c_6' are merged in the 2nd iteration because the weight of their connecting edge is 1. Fig. 7(b) shows the updated *VCFG* and *VCPG*. With the

Algorithm 2 Vendor-assignment with core minimization, $vendor\_assign(TC, TC', vc, pc)$.

Input: performance-constrained clustering results of task graph and duplicated task graph, TC, TC'.
performance and vendor constraints, pc, vc.
Output: vendor assignment, VA.

1: **for** each cluster c **do**
2: calculate $DG_{max}(c)$.
3: **end for**
4: Construct $VCFG$ and $VCPG$;
5: Calculate the weights of edges in $VCPG$
6: **while** $VCPG.node\_num > vc$ **do**
7: Choose the edge $e_{max} = (c_i, c_j)$ with the maximum weight in $VCPG$, and merge c_i and c_j into one cluster.
8: Update $VCFG$ and weighted $VCPG$,
9: **end while**
10: **while** Not all clusters are assigned with IP vendors **do**
11: assign c_i to vendor $vendor_j$, where c_i is an unassigned cluster with the most timing critical tasks and $vendor_j$ is the available vendor with the fastest core speed.
12: **end while**
13: Update the distribution graphs of tasks with assigned core speeds;
14: E_{cv} is the set consists of all intra-core communications;
15: **while** $E_{cv} \neq \emptyset$ **do**
16: Find $e \in E_{cv}$ with the largest $risk(e)$;
17: **if** Assigning e with security constraints does not violate performance constraints **then**
18: Assign either $source(e)$ or $target(e)$ with another IP vendor;
19: Update the distribution graphs of tasks;
20: **end if**
21: Remove e from E_{cv};
22: **end while**

corresponding $VCFGs$ and $VCPGs$ shown in Fig. 7(b)-Fig. 7(f), the pairs of clusters (c'_1, c_7), (c'_3, c'_7), (c_1, c_3), (c_2, c'_3) and (c_1, c'_2) are iteratively merged. This procedure terminates when the number of clusters equals the vendor constraint, and Fig. 8 gives the final results, where the total estimated number of cores is 5.5.

Then, the number of timing-critical tasks in each cluster is countered, and the clusters containing more timing-critical tasks are assigned to the vendor with faster core speeds. Some timing-critical paths may be over-optimized because all tasks are treated with the lowest core speed in the performance-constrained task clustering stage, and we need to adjust the vendor assignment to meet more security constraints. The slacks of tasks are updated with the assigned core speeds, and every intra-core communication is checked in descending order of $risk(e)$ to determine whether this communication can be reassigned with security constraints. An intra-core communication (e) can be reassigned to inter-core communication to satisfy security constraints only when all tasks in the paths that pass through e have slack times no smaller than $dly(e)$, and one of its connected tasks will be assigned to the IP vendor with the least core increment.

Algorithm 2 describes the proposed vendor assignment algorithm. First (*Lines 1-5*), the number of cores required by each cluster is estimated by $DG_{max}(c)$, and $VCFG$ and weighted $VCPG$ are constructed. Second (*Lines 6-9*), the maximum weight edge in $VCPG$ is chosen, and the connected clusters are merged into one cluster. Both $VCFG$ and weighted $VCPG$ are then updated, and this procedure continues until the number of clusters equals the number of IP vendors available. Third (*Lines 10-13*), the cluster containing more timing-critical tasks is assigned to the IP vendor with faster core speed. Finally (*Lines 14-22*), the unprotected edges are checked in descending order of security risk, and the vendor assignments are adjusted under performance constraints to further reduce the system security risk.

C. Procedure of the Proposed Task Scheduling Method

With all security constraints satisfied, the number of IP vendors is always equal to the number of nodes in the maximum clique (denoted as *maximum clique size*) of $VCFG$. However, performance-constrained task clustering and vendor assignment may potentially increase the number of vendors required, and we must check every contracted edge if the resulting maximum clique size exceeds the vendor constraint. Computing the maximum clique size of a graph is NP-complete, and an efficient heuristic approach [29] is introduced. Each time after determining a contracted edge, the impact on the maximum clique size of the corresponding $VCFG$ is evaluated, and the edge is not contracted if the vendor constraint is violated. Instead, the algorithm chooses the second-best solutions.

Algorithm 3 Security-aware task scheduling with performance constraints, $task\_schedule(TG, pc, vc)$.

Input: task graph, TG
performance and vendor constraints, pc, vc.
Output: scheduling results, TS.

1: $TC = task\_cluster(TG, pc)$;
2: $TC' = task\_cluster(TG', pc)$, where TG' is the duplicate of TG;
3: $VA = vendor\_assign(TC, TC', vc, pc)$;
4: **for** each vendor $vendor_i$ **do**
5: V_{vendor_i} is the set of all tasks assigned to $vendor_i$;
6: $FDS(V_{vendor_i}, pc)$;
7: **end for**

After vendor assignment, tasks with the same IP vendor are scheduled together using the force-directed scheduling (FDS) method [55]. FDS schedules tasks evenly across each time period, requiring only a small number of cores. Algorithm 3 gives the whole procedure of the proposed task scheduling algorithm. Tasks in the task graph and duplicated task graph are clustered under the performance constraint p_c (*Lines 1-2*) and then assigned to IP vendors with a minimized number of cores required (*Line 3*). Finally, tasks in each IP vendor are scheduled concurrently by the FDS (*Lines 4-7*). In this study, we considered system risk optimization with two types of security constraints, and the proposed method can also be applied when more types of security constraints are introduced.

Our proposed methods can also be easily adopted in other actual scenarios. 1) In the first scenario, the number of vendors available might be less than the maximum clique size of the corresponding $VCFG$. In this situation, a vendor-constrained task clustering method [30] can be introduced before vendor assignment, so that the vendor constraint can be satisfied with a minimized number of contracted edges. In the second scenario, application-specific IPs might not be able to support every task. For each type of IP, we first build its $VCFG$ and $VCPG$ according to the clustering results of tasks that belong to this type, and then perform the vendor assignment along with task scheduling individually.

D. Time Complexity Analysis

The time complexity of the proposed method is analyzed as follows, and the input task graph has n nodes and m edges.

In each iteration of the performance-constrained task clustering stage, constructing $ECCG$ from TVG requires $O(m^2)$, and finding the MWIS in $ECCG$ also requires $O(m^2)$ [56]. Only a constant number of iterations are conducted before reaching the performance constraint, and finding all contracted edges to meet the performance constraint requires $O(m^2)$. In addition, each time before contracting an edge, updating $VCFG$ and evaluating its impact on the maximum clique size requires $O(n^2)$, and only a limited number of edges are contracted, making its computational cost remains at $O(n^2)$. The total time complexity of performance-constrained task clustering is $O(m^2)$ (because $O(n) \leq O(m)$).

In the vendor assignment and task scheduling stage, constructing $VCFG$ and $VCPG$ requires $O(n^2)$. In each iteration of merging clusters, $O(m)$ is required to estimate the maximum clique size, and $O(n)$ is required to update both $VCFG$ and $VCPG$. Vender assignment requires $O(n)$ iterations of merging clusters, and its time complexity is $O(mn)$. Performing the force-directed scheduling method to schedule all tasks requires $O(n^2)$, and the total time complexity of the vendor assignment and task scheduling stage is $O(mn)$.

The sum of $O(m^2)$ and $O(mn)$ is $O(m^2)$, which is the total time complexity of the proposed method.

V. Experimental Results

A. Experimental Setups

All experiments were implemented in C on a Linux Workstation with an E5 2.6-GHz CPU and 32-GB RAM. We tested eight benchmarks from two sources[1]: task graphs modeled from real application programs, including robot control (robot), sparse matrix solver (sparse), and SPEC fpppp (fpppp); and randomly generated task graphs (rnc500, rnc1000, rnc2000, rnc3000 and rnc5000). To simplify the experiments, all intra-core communication delays were ignored, and we set the step of core speed differences equal to 5% of the fastest core speed.

Table III lists the details of these benchmarks. Columns n and m give the numbers of tasks and communications in each task graph, respectively. Column Para. shows the parallelism of each task graph, which is the ratio of the total processing time of all tasks to the ASAP schedule length (without communication delays). Column ACC shows the average computational cost of each task. Considering that the maximum clique sizes of most task graphs modeled from real application programs are no larger than 4 [29], the maximum clique sizes (see column *maxClique*) of all randomly generated TGs in Table III are 3 or 4, and the IP vendor constraint is set to be the maximum clique size of the benchmark.

TABLE III
Details of Benchmarks

task graph	n	m	Para.	ACC (ut)	*maxClique*
robot	88	131	4.4	28.2	3
sparse	96	67	16.0	20.2	3
fpppp	334	1145	6.7	21.3	3
rnc500	500	1910	27.7	10.6	3
rnc1000	1000	3005	60.2	7.8	3
rnc2000	2000	3930	151.9	10.6	3
rnc3000	3000	39034	34.2	13.0	4
rnc5000	5000	55432	90.8	11.0	4

Our proposed method is then compared with two other methods to demonstrate its effectiveness. The first approach is the "*cluster-based approach*" (**C-B** for short) [29]. This approach optimizes the schedule length by placing critical tasks on a single core, and then colors the performance-driven schedule to fulfill security constraints. The second approach is the "*min-cut-based approach*" (**MC-B** for short) [30]. This approach boosts performance by iteratively contracting the edges selected by the max-flow min-cut algorithm, then assigns tasks to the IP vendor with the traditional graph coloring method, and finally schedules tasks with the force-directed scheduling-based method.

B. Performance-Constrained Task Scheduling Results

The performance-constrained task scheduling results are shown in Table V. The communication-to-computation ratio (CCR) is the ratio of the inter-core communication delay to the computational cost of the task, and two $CCRs$ (0.5 and 1.0) are tested. The performance constraint is set to pc=0.8SL, where SL is the ASAP schedule length with all security constraints satisfied. The security risk variation among communications is also counted in this set of experiments, and we first assume that the task outputs may contain more confidential information and that the system damage caused by triggered hardware trojans becomes more serious when the application proceeds. Therefore, we set the security risk of e_{ij} in TG or TG' as follows:

$$risk(e_{ij}) = \frac{2 * dist(e_{ij}) * pte}{SL * m} \tag{7}$$

where $dist(e_{ij})$ is the distance from s to v_i. pte is the *possible trojan entries* [25], and we set pte equal to 3 for all benchmarks.

Table IV compares the task scheduling results of these methods on three real application graphs. The results show that our proposed method obtained the lowest $risk_s$ for all

[1]https://www.kasahara.cs.waseda.ac.jp/schedule/index.html.

TABLE IV
PERFORMANCE-CONSTRAINED TASK SCHEDULING RESULTS ON REAL APPLICATION GRAPHS.

task graph	CCR	SL (ut)	pc (ut)	$risk_s$ (10^{-3})			Num. of cores			average runtime (s)		
				C-B [29]	MC-B [30]	Our	C-B [29]	MC-B [30]	Our	C-B [29]	MC-B [30]	Our
robot	0.5	839	671	353.68	206.57	155.35	14	12	11	3.9	17.5	18.7
	1.0	1114	892	274.91	185.62	147.83	14	12	10	3.9	17.6	18.9
sparse	0.5	179	143	146.75	93.48	76.07	21	18	16	5.3	41.2	33.8
	1.0	236	189	157.48	98.47	83.59	19	18	15	5.5	42.3	34.5
fpppp	0.5	1590	1272	15.23	6.37	4.52	13	12	10	7.5	50.2	47.6
	1.0	2119	1695	11.49	6.85	4.97	12	11	10	8.3	53.5	49.8
avg.	0.5			171.89	102.14	78.65						
	1.0			147.96	96.98	78.80						

TABLE V
PERFORMANCE-CONSTRAINED TASK SCHEDULING RESULTS ON RANDOMLY GENERATED TASK GRAPHS.

task graph	CCR	SL (ut)	pc (ut)	$risk_s$ (10^{-3})			Num. of cores			average runtime (s)		
				C-B [29]	MC-B [30]	Our	C-B [29]	MC-B [30]	Our	C-B [29]	MC-B [30]	Our
rnc500	0.5	280	224	25.13	15.70	12.46	68	65	58	18.9	113.6	95.4
	1.0	373	300	20.41	16.35	12.95	67	63	56	20.3	121.3	98.7
rnc1000	0.5	190	152	47.92	33.94	26.71	95	88	78	38.8	259.3	183.5
	1.0	254	203	48.73	31.95	23.69	87	81	74	42.1	273.3	189.7
rnc2000	0.5	199	159	20.61	16.53	12.37	217	184	167	57.5	715.7	553.6
	1.0	268	214	21.47	14.51	10.62	206	180	164	59.1	748.2	572.9
rnc3000	0.5	1336	1069	56.41	43.27	35.58	73	67	62	182.5	3582.6	3014.9
	1.0	1779	1423	53.49	37.74	31.18	68	62	56	192.2	3715.3	3253.4
rnc5000	0.5	850	680	48.13	43.17	34.52	142	132	122	384.9	9004.5	6764.2
	1.0	1146	917	54.84	47.25	36.55	137	125	115	395.4	9528.3	6816.1
avg.	0.5			39.64	30.52	24.33						
	1.0			39.79	29.56	23.01						

benchmarks. MC-B optimized the system risk by reducing the number of unprotected communications but ignored the security risk variation of communications. MC-B might also choose brother edges to contract, resulting in a $risk_s$ larger than that of our proposed method. Furthermore, MC-B forgot to optimize the number of cores in the vendor assignment stage, and it required more cores than our method. Regarding method C-B, it ignored both the system risk and the number of cores required during task scheduling, and both the $risk_s$ and number of cores are the largest among the compared methods.

The comparison results of five randomly generated task graphs are shown in Table V. The number of edges in these task graphs is much larger than the edges of *robot*, *sparse*, and *fpppp*, and therefore, the risk of each communication is relatively much smaller (refer to Equ. (7)). The comparison results show that our proposed method also obtained the best results in reducing both the $risk_s$ and the number of cores.

In terms of algorithm runtime, all of the algorithms can produce solutions within 1 min for the benchmarks modeled from real applications (see Table IV). For benchmarks that contain many nodes and edges, such as *rnc5000* which has 5000 nodes and 55432 edges (see Table V), our proposed method can output a solution with in approximately 10 minutes, which indicates that our proposed method is applicable for most benchmarks in real practice.

C. Comparison of Security Risk Optimization Results

Then, the effectiveness of our method in reducing the system security risk is tested. CCR is set to CCR=1.0 for all benchmarks, and the performance constraint is set to $pc = 0.9 * SL$. Fig. 9 shows the comparative results of security risks among C-B, MC-B, and our methods, with the following three sets of risk configurations:

1) **Randomly Set**: the risk of each communication is randomly set with the value among [0, $\frac{2*pte}{m}$];
2) **Equally Set**: the risks of all communications are the same, with the value of $\frac{pte}{m}$;
3) **Linearly Set**: The communication closer to the source has higher security risk, and the risk of e_{ij} is set as $risk(e_{ij}) = \frac{2*(SL-dist(e_{ij}))*pte}{SL*m}$;

The pte is set to 3 in this set of experiments. Because both C-B and MC-B ignore the security risk variation among communications, for each benchmark, they produced the same task scheduling result for different risk configurations. However, different risk configurations assign communication with different risks, and therefore, the system security risks of the same task schedule are not the same with different risk configurations. Our proposed method seeks the task scheduling result with the lowest system security risk, and produces different task scheduling results for different risk configurations.

Figs. 9(a), 9(b), and 9(c) show the system security risks with the risk of each communication *randomly set*, *equally set*, and *linearly set*, respectively. For each tested benchmark, our proposed method outperforms both C-B and MC-B with different risk configurations, and detailed comparisons are given as follows:

1) When the risks are *randomly set*, the averaged system security risks of C-B and MC-B are 36.72*10^{-3}, and

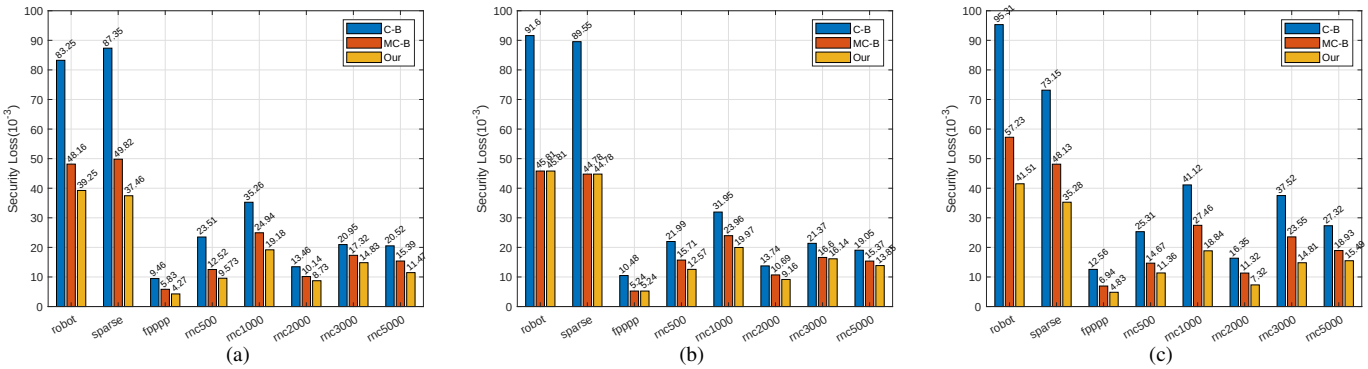


Fig. 9. Comparison of system security risks. (a) System security risk evaluations with risks *randomly set*. (b) System security risk evaluations with risks *equally set*. (c) System security risk evaluations with risks *linearly set*.

TABLE VI
COMPARISONS OF CORES REQUIRED.

task graph	*SL* (*ut*)	Loose vendor constraints				Tight vendor constraints						
		vc	Num. of cores			*vc*	Num. of cores			Proportion of unprotected edges		
			C-B [29]	MC-B [30]	Our		C-B [29]	MC-B [30]	Our	C-B [29]	MC-B [30]	Our
robot	1114	3	10	9	8	2	10	8	7	2.29%	1.53%	1.53%
sparse	236	3	14	12	12	2	14	11	10	8.96%	5.97%	5.97%
fpppp	2119	3	10	9	8	2	9	7	6	1.57%	0.96%	0.96%
rnc500	373	3	47	45	39	2	42	38	35	4.50%	3.72%	3.72%
rnc1000	254	3	68	63	56	2	58	51	47	1.73%	1.30%	1.30%
rnc2000	268	3	156	148	130	2	139	127	116	0.74%	0.46%	0.46%
rnc3000	1779	4	56	51	48	2	52	48	45	58.52%	53.15%	53.15%
rnc5000	1146	4	106	98	88	2	97	91	83	70.24%	67.89%	67.89%

$23.02*10^{-3}$, respectively, and our proposed method obtains the lowest security risk, which is only $18.11*10^{-3}$.

2) If the risks are *equally set*, our method and MC-B obtain nearly equivalent results because MC-B treats each communication equally when optimizing system performance. The average system security risks of C-B, MC-B and our methods are $37.47*10^{-3}$, $22.27*10^{-3}$, and $20.94*10^{-3}$, respectively.
3) If the risks are *linearly set*, our proposed method still demonstrate its advantage when reducing system risks: its average system risk is $18.68*10^{-3}$, while the average system risks of C-B and MC-B are $41.08*10^{-3}$ and $26.03*10^{-3}$, respectively.
4) For some benchmarks (*robot* and *sparse*), their system security risks are relatively high because the risks of communications are also determined by the number of edges in task graphs. These task graphs have much fewer edges, making the risks of communications in *robot* and *sparse* much larger than that in the other task graphs.

D. Comparison of Cores Required

To demonstrate the effectiveness of our method in reducing the number of cores required, the performance constraint is set to *SL* so that the performance-constrained task clustering stage can be skipped. Table VI shows the numbers of cores required, where two sets of vendor constraints are tested. The loose vendor constraints are first set for all benchmarks, and the vendor constraint is set to be the maximum clique size of the corresponding *TG*. C-B ignores core optimization, and MC-B only minimizes the number of cores when scheduling tasks. In our proposed method, reducing the number of cores is considered in both vendor assignment and task scheduling, which enlarges the optimization space for saving cores. The results indicate that our proposed method needs the fewest numbers of cores in all benchmarks, and MC-B and our proposed method reduce the numbers of cores by 8.44% and 17.11%, respectively, compared to the C-B method.

Then, the tight vendor constraints are set because there might not be sufficient vendors for some specific IPs, and the vendor constraints of all benchmarks are set to 2. To meet the tight vendor constraints, the vendor-constrained task clustering method proposed in [30] is used to remove some security constraints from communications and allow the adjacent tasks to be executed on the cores from the same IP vendor. Therefore, more communications become unprotected, and the proportions of the number of unprotected edges to the number of all edges are given in the last column of Table VI. Because MC-B and our method use the same vendor-constrained task clustering, the proportions of their unprotected edges are the same. C-B only ignored reducing the number of unprotected edges to meet vendor constraints, and its numbers of unprotected edges are larger. For the benchmarks *rnc3000* and *rnc5000*, whose task graphs contain many 4-cliques, many edges become unprotected to satisfy the vendor constraint. The number of cores required shows that our proposed method also obtained the fewest cores among these three methods, and the average numbers of cores

required by MC-B and our proposed method are 13.47% and 21.49% less than that of C-B, respectively.

VI. Conclusions

In this study, a security-driven task scheduling method is proposed to reduce the performance and area overheads of implementing security constraints in the design process, and the desired performance is set as a constraint. Communications are treated with different security importance due to the security risk variation, and a maximum weight independent set-based task clustering method is proposed to reduce the schedule length while maintaining a high security level. In addition, the numbers of cores required are optimized in both vendor assignment and task scheduling stages by assigning tasks that can share most cores to the same vendor and scheduling them evenly in each time period, which enlarges the optimization space for reducing cores. Experimental results demonstrate that our proposed method obtains the highest system security and the fewest cores among all compared methods.

References

[1] X. Wang and R. Karri, "NumChecker: detecting kernel control-flow modifying rootkits by using hardware performance counters," *Proc. Design Automation Conference*, pp. 1-7, May 2013.

[2] S. Bhunia, M.S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229-1247, Aug. 2014.

[3] M. Hussain, A. Malekpour, H. Guo, and S. Parameswaran, "EETD: an energy efficient design for runtime hardware Trojan detection in untrusted network-on-chip," *Proc. IEEE Computer Society Annual Symposium on VLSI*, pp. 345-350, 2018.

[4] A. Malekpour, R. Ragel, A. Ignjatovic, and S. Parameswaran, "DosGuard: protecting pipelined MPSoCs against hardware trojan based DoS attacks," *Proc. International Conference on Applications-specific Systems, Architectures and Processors*, pp. 45-52, 2017

[5] F. Kounelis, N. Sklavos, and P. Kitsos, "Run-time effect by inserting hardware trojans in combinational circuits," *Euromicro Conference on Digital System Design*, pp. 287-290, 2017.

[6] S. Swapp, *Scanning Electron Microscopy (SEM)*, University of Wyoming.

[7] M. Banga and M.S. Hsiao, "A novel sustained vector technique for the detection of hardware trojans," *Proc. International Conference of VLSI Design*, pp. 327-332, 2009.

[8] B. Bilgic and S. Ozev, "Guaranteed activation of capacitive Trojan triggers during post production test via supply pulsing," *Proc. Design, Automation & Test in Europe Conference*, pp. 993-998, 2022.

[9] K. Xiao and M. Tehranipoor, "BISA: Built-in self-authentication for preventing hardware Trojan insertion," *Proc. International Symposium on Hardware-Oriented Security and Trust*, pp. 45-50, 2013.

[10] D. Deng, Y. Wang, and Y. Guo, "Novel design strategy toward A2 Trojan detection based on built-in acceleration structure," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4496-4509, Feb. 2020.

[11] Y. Huang, S. Bhunia, and P. Mishra, "Scalable test generation for Trojan detection using side channel analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2746-2760, Nov. 2018.

[12] L. Nguyen, C. Cheng, M. Prvulovic, and A. Zajić, "Creating a backscattering side channel to enable detection of dormant hardware Trojans," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 7, pp. 1561-1574, Apr. 2019.

[13] S. Yang, T. Hoque, P. Chakraborty, and S. Bhunia, "Golden-free hardware Trojan detection using self-referencing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 3, pp. 325-338, Mar. 2022.

[14] Y. Hou, H. He, K. Shamsi, Y. Jin, D. Wu, H. Wu, "R2D2: runtime reassurance and detection of A2 Trojan," *Proc. International Symposium on Hardware-Oriented Security and Trust*, pp. 195-200, 2018.

[15] J. He, X. Guo, H. Ma, Y. Liu, Y. Zhao, and Y. Jin, "Runtime trust evaluation and hardware Trojan detection using on-chip EM sensors," *Proc. Design Automation Conference*, pp. 1-6, Jun. 2020.

[16] Y. Hou, H. He, K. Shamsi, Y. Jin, D. Wu and H. Wu, "On-chip analog Trojan detection framework for microprocessor trustworthiness," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1820-1830, Oct. 2019.

[17] J. Zhu *et al.*, "Jintide: utilizing low-cost reconfigurable external monitors to substantially enhance hardware security of large-scale CPU clusters," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 8, pp. 2585-2601, Aug. 2021.

[18] J. Rajendran, O. Sinanoglu, and R. Karri, "Building trustworthy systems using untrusted components: a high-level synthesis approach," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 9, pp. 2946-2959, Apr. 2016.

[19] S. Sethumadhavan, A. Waksman, M. Suozzo, Y. Huang, and J. Eum, "Trustworthy hardware from untrusted components," *Communications of the ACM*, vol. 58, no. 9, pp. 60-71, Sep. 2015.

[20] T. Reece and W. H. Robinson, "Detection of hardware Trojan in third-party intellectual property using untrusted modules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, pp. 357-366, Jul. 2015.

[21] M. Beaumont, B. Hopkins, and T. Newby, "SAFER PATH: security architecture using fragmented execution and replication for protection against Trojaned hardware," *Proc. Design, Automation & Test in Europe Conference*, pp. 1000-1005, Mar. 2012.

[22] X. Cui et al., "High-level synthesis for run-time hardware Trojan detection and recovery," *Proc. Design Automation Conference*, pp. 1-6, Jun. 2014.

[23] M. Shatta, I. adly, H. Amer, G. Alkady, R. Daoud, S. Hamed, and S. Hatem, "FPGA-based architectures to recover from hardware Trojan horses, single event upsets and hard failures," *Proc. International Conference on Microelectronics*, pp. 1-4, 2020.

[24] J. Rajendren, H. Zhang, O. Sinanoglu, and R. Karri, "High-level Synthesis for Security and Trust," *Proc. International On-Line Testing Symposium*, pp. 232-233, 2013.

[25] S. Rajmohan, N. Ramasubramanian, and N. Naganathan, "Hybrid evolutionary design space exploration algorithm with defence against third party IP vulnerabilities," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2602-2614, May 2022.

[26] A. Sengupta and S. Bhadauria, "Untrusted third party digital IP cores: power-delay trade-off driven exploration of hardware Trojan secured datapath during high level synthesis," *Proc. Great Lakes Symposium on VLSI*, pp. 167-172, May 2015.

[27] Y. Sun, G. Jiang, S.-K. Lam, and F. Ning, "Designing energy-efficient MPSoC with untrustworthy 3PIP cores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 51-63, Jan. 2020.

[28] X. Cui, X. Zhang, H. Yan, L. Zhang, K. Cheng, Y. Wu, and K. Wu, "Toward building and optimizing trustworthy systems using untrusted components: a graph-theoretic perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1386-1399, Oct. 2020.

[29] C. Liu, J. Rajendran, C. Yang, and R. Karri, "Shielding heterogeneous MPSoCs from untrustworthy 3PIPs through security-driven task scheduling," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 4, pp. 461-472, Aug. 2014.

[30] N. Wang, S. Chen, J. Ni, X. Ling, and Y. Zhu, "Security-aware task scheduling using untrusted components in high-level synthesis," *IEEE Access*, vol. 6, pp. 15663-15678, Jan. 2018.

[31] N. Wang, M. Yao, D. Jiang, S. Chen, and Y. Zhu, "Security-driven task scheduling for multiprocessor system-on-chips with performance constraints," *Proc. IEEE Computer Society Annual Symposium on VLSI*, pp. 545-550, 2018.

[32] S. Bhunia, M. Abramovici, D. Agrawal, P. Bradley, M.S. Hsiao, J. Plusquellic, M. Tehranipoor, "Protection against hardware trojan attacks: towards a comprehensive solution," *IEEE Design & Test*, vol. 30, no. 3, pp. 6-17, Jun. 2013.

[33] R.S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware trojan: threats and emerging solutions," *Proc. IEEE International High Level Design Validation and Test Workshop*, pp. 166-171, 2009.

[34] S. Moulik, R. Devaraj, A. Sarkar, and A. Shaw, "A deadline-partition oritented heterogeneous multi-core schedule for periodic tasks," *Proc. International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 204-210, 2017.

[35] S. Moulik, Z. Das, and G. Saikia, "CEAT: A cluster based energy aware scheduler for real-time heterogeneous systems," *Proc. International Conference on Systems, Man, and Cybernetics*, pp. 1815-1821, 2017.
[36] S. Moulik, "RESET: a real-time scheduler for energy and temperature aware heterogeneous multi-core systems," *Integration, the VLSI Journal*, vol. 77, pp. 59-69, 2021.
[37] S. Moulik, Z. Das, R. Devaraj, and S. Chakraborty, "SEAMERS: A semi-partitioned energy-aware scheduler for heterogeneous multicore real-time systems," *Journal of Systems Architecture*, vol. 114, pp. 101953-101953, 2021.
[38] W. Hu, C.-H. Chang, A. Sengupta, S. Bhunia, R. Kastner, H. Li, "An overview of hardware security and trust: threats, countermeasures, and design tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 6, pp. 1010-1038, Jun. 2021.
[39] Y. Cao, C.Q. Liu, and C.-H. Chang, "A low power diode-clamped inverter-based strong physical unclonable function for robust and lightweight authentication," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 11, pp. 3864-3873, Nov. 2018.
[40] F. Pareschi, G. Setti, and R. Rovatti, "Implementation and testing of high-speed CMOS true random number generators based on chaotic systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 10, pp. 3124-3137, Dec. 2010.
[41] Q. Shi, M.M. Tehranipoor, and D. Forte, "Obfuscated built-in self-authentication with secure and efficient wire-lifting," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 1981-1994, Nov. 2019.
[42] D. Meng, R. Hou, G. Shi, B. Tu, A. Yu, Z. Zhu, X. Jia, Y. Wen, and Y. Yang, "Built-in security computer: deploying security-first architecture using active security processor," *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1571-1583, Nov. 2020.
[43] A. Ardeshiricham, Y. Takashima, S. Gao, and R. Kastner, "VeriSketch: Synthesizing secure hardware designs with timing sensitive information flow properties," *Proc. ACM SIGSAC Conference on Computer and Communications Security*, pp. 1623-1638, 2019.
[44] T. Alves and D. Felton, TrustZone: Integrated hardware and soft ware security-enabling trusted computing in embedded systems, ARM, Cambridge, U.K., White Paper, 2014.
[45] H. Kim, S. Hong, B. Preneel, and I. Verbauwhede, "STBC: Side channel attack tolerant balanced circuit with reduced propagation delay, *Proc. IEEE Computer Society Annual Symposium on VLSI*, pp. 74-79, 2017.
[46] S. Patranabis, A. Chakraborty, D. Mukhopadhyay, and P.P. Chakrabarti, "Fault space transformation: a generic approach to counter differential fault analysis and differential fault intensity analysis on AES-like block ciphers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 5, pp. 1092-1102, May 2017.
[47] A. Sengupta, D. Roy, and S.P. Mohanty, "Triple-phase watermarking for reusable IP core protection during architecture synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 4, pp. 1092-1102, Apr. 2018.
[48] A. Sengupta and M.Rathor, "IP core steganography for protecting DSP kernels used in CE systems," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 4, pp. 506-515, Nov. 2019.
[49] K. Hasegawa, M. Yanagisawa, N. Togawa, "Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifer," *Proc. IEEE Symposium on Circuits and Systems*, pp. 74-79, 2017.
[50] W. Shan, S. Zhang, J. Xu, M. Lu, L. Shi, and J. Yang, "Machine learning assisted side-channel-attack countermeasure and its application on a 28-nm AES circuit," IEEE Journal of Solid-State Circuits, vol. 55, no. 33, pp. 794-804, Mar. 2020.
[51] X. Zhang and M. Tehranipoor, "Case study: detecting hardware trojans in third-party deigital IP cores," *International Symposium on Hardware-Oritented Security and Trust*, pp. 67-70, 2011.
[52] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: lessons learned after one decade of research," *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 1, pp. 6-29, May 2016.
[53] D. Gizopoulos *et al.*, "Architectures for online error detection and recovery in multicore processors," *Proc. Design, Automation and Test in Europe Conference*, pp. 533-538, 2011.
[54] N. Veeranna and B.C. Schafer, "Hardware Trojan detection in behavioral intellectual properties (IP's) using property checking techniques," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 576-585, Oct. 2017.
[55] P.G. Paulin and J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661-679, Jun. 1989.
[56] L. Chang, W. Li, and W. Zhang, "Computing a near-maximum independent set in linear time by reducing-peeling," *Proc. ACM International Conference on Management of Data*, pp. 1181-1196, 2017.

PLACE PHOTO HERE

Nan Wang received a B.E. degree in computer science from Nanjing University, Nanjing, China, in 2009, and M.S and Ph.D. degrees from the Graduate School of IPS, Waseda University, Japan, in 2011, and 2014, respectively. He is currently an associate professor in School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China. His current research interests include VLSI design automation, low power design techniques, network-on-chip and reconfigurable architectures. Dr. Wang is a member of IEEE and IEICE.

PLACE PHOTO HERE

Song Chen received a B.S. degree in computer science from Xi'an Jiaotong University, Xi'an, China, in 2000, and M.S. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China, in 2003 and 2005, respectively. From August 2005 to March 2009, he served as a research associate at the Graduate School of IPS, Waseda University, Japan, and from April 2009 to August 2012, he served the same university as an assistant professor. He is currently an associate professor at the Dept. of Electronic Sci. and Tech., University of Science and Technology of China (USTC). His current research interests include several aspects of VLSI physical design automation, on-chip communication system, and computer-aided design for emerging technologies. Dr. Chen is a member of IEEE and IEICE.

PLACE PHOTO HERE

Hongqing Zhu received the ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 2000. From 2003 to 2005, she was a Post-Doctoral Fellow with the Department of Biology and Medical Engineering, Southeast University, Nanjing, China. She is currently a Professor at the East China University of Science and Technology, Shanghai. Her current research interests include deep learning, pattern recognition, and information security. She is a member of IEEE and IEICE.

PLACE PHOTO HERE

Yu Zhu received the B.S. and Ph.D. degrees in electronics and communication engineering from Nanjing University of Science and Technology, Nanjing, China, in 1995 and 1999 respectively. She is currently a professor of electronics and communication engineering in East China University of Science and Technology, Shanghai, P.R. China. In 2005, she was a research scholar in UIUC. Her current research interests include computer design automation, pattern recognition and machine learning.

AUTHORS' RESPONSE

We greatly appreciate the Editor's and the reviewers' insightful and scrupulous reviews of our paper. The comments provided have contributed substantially to the improvement of our manuscript. In what follows, we present the detailed explanations of how the manuscript has been revised to respond to the comments of the associate editor and the reviewers. In the previous pages, the sentences colored blue, red, and cyan are the modified parts.

REVIEWER 1

There is no comment.

REVIEWER 2

Comment 1

I believe the paper has left some major points unaddressed, and the most important one is the threat model.

The followings are missing questions related to this point: What is the addressed threat model? How does a Trojan (defined at the gate/RTL level) is supposed to tamper tasks running on the cores without having the cores failed? What is the Trojans purpose in this paper? How the Trojan is activated? If one or more of the vendors are Trojan infected how can compromise the whole MPSoC/cluster?

Responses

Q1. What is the addressed threat model?

A1. In this revised manuscript, we have included the threat model that we focused on in this work. This threat model is presented in Section III-A *Threat Model* (in the lower-right of page 3 in cyan color), which is also presented as follows.

In this study, we adopt the same threat model in [27], [29], which primarily focuses on detecting (or mitigating) malicious modifications. The trojan may cause the task running on the malicious 3PIP to either produce incorrect output or generate additional output to trigger trojans in another 3PIP core from the same vendor. As a result, the following two cases can occur at runtime: 1) due to the insertion of the malicious logic into a 3PIP core, the outputs of the infected cores will be altered at some undetectable points; 2) trojans that are distributed on multiple cores to reduce the chance of being detected can also form secret communication paths, and a malicious logic in one core can trigger the trojans in another core using a secret communication channel.

Q2. How does a Trojan (defined at the gate/RTL level) is supposed to tamper tasks running on the cores without having the cores failed?

A2. In this revised manuscript, the details of trojan attacks are described as follows.

Hardware trojan attacks are intended to affect normal circuit operation, potentially with catastrophic consequences in critical applications in the domains of communications, banking, space and military [32]. They can also aim at leaking secret information from inside a chip through covert channels or affect the reliability of an IC through undesired process changes that cause device ware-out and long-term reliability issues [33]. In addition, they can be used to assist software attacks by providing hardware back-door, and make the system operate in an incorrect way, such as modifying the scheduling results of a real-time system [34]–[37].

The above explanations are also presented in the mid-right of page 3, magenta colored.

Q3. What is the Trojans purpose in this paper?

A3. The purpose of the trojan that we have focused on in this paper can be summarized as follows.

The trojan may cause the task running on the malicious 3PIP to either produce incorrect output or generate additional output to trigger trojans in another 3PIP core from the same vendor.

The above description of trojans can also be found in the lower-right of page 3, cyan colored.

Q4. How the Trojan is activated?

A4. Thanks for your comments. We realized that we forgot to include a discussion on how the trojan is activated in the previous manuscript. In this revised version, we have added the following discussion to clarify how the trojans are activated.

From the perspective of the activation methods, hardware trojans can be classified as either *always-on* or *conditionally triggered*. An always-on trojan is inserted in rarely accessed places and its footprint is kept small. Conditionally triggered trojans hibernate initially, and are activated either by the trojan implanter or by on-chip triggers [29].

The above discussion can also be found in the mid-right of page 3, blue colored.

Q5. If one or more of the vendors are Trojan infected how can compromise the whole MPSoC/cluster?

A5. This depends on what type of the trojan is, and one of the conclusions in [3] is "a small hardware modification by an adversary in the 3PIP cores can compromise the whole chip".

In this work, we focused on the types of trojans that may cause the following two situations:

1) Due to the insertion of the malicious logic into a 3PIP core, the outputs of the infected cores will be altered at some undetectable points, 2) Trojans distributed on multiple cores (in order to reduce the chance of being detected) can also form malicious communication paths where a malicious logic in one core triggers the trojans in another core using a secret communication channel.

The above descriptions can also be found in the lower-right of page 3, cyan colored.

Comment 2

There should be a clear discussion at the end of the introduction to clarify new contributions of the submitted paper with respect to the conference version. In the cover letter, the authors mentioned in a number of experiments are added.... The reader needs to clearly know what these new experiments are and why they are informative.

Responses

Q1. There should be a clear discussion at the end of the introduction to clarify new contributions of the submitted paper with respect to the conference version.

A1. Thanks for your valuable suggestion. In this revised manuscript, we have highlighted the new contributions of the work compared to our earlier conference version. This part is presented at the end of Section *Introduction*, and is indicated in red in the mid-left of page 2. The new contributions are also listed below.

The earlier conference version of this paper appeared at [31]. Compared with [31], this paper has a number of new contributions.

1) This work treats communications with different security importance, and provides a schedule with a minimized system security risk.
2) Rather than assigning one task to an IP vendor each time, this vendor assignment method groups all tasks into a number of *vc* (vendor constraint) clusters and assigns an entire cluster to an IP vendor at a time. Furthermore, this vendor assignment method evaluates the number of cores saved when clustering tasks rather than estimating the number of cores required, which speeds up the process time and provides better results.
3) This work considers core speed variation in the design process. All tasks are first assumed to be performed with the slowest speed, and the vendor assignment will be adjusted after the exact core speeds are determined to further reduce the system security risk.

Q2. In the cover letter, the authors mentioned in a number of experiments are added.... The reader needs to clearly know what these new experiments are and why they are informative.

A2. In this revised manuscript, the Section *Experimental Results* has been reorganized. First, we present an overall evaluation of our method's effectiveness in reducing system risks and the number of cores. Next, we compare the system risk optimization results under different risk configurations, and our method is shown to achieve the highest level of system security with all risk configurations. Finally, we present the results of saving the number of cores with different vendor constraints, which demonstrate that our method requires the least number of cores under both loose and tight vendor constraints.

The above explanation is also presented in the cover letter.

Comment 3

The authors have used a clustering method. However, it is not clear how much effective it is. For example, some IPs might be application-specific IPs and might not be able to support every task assigned to them. At least, clustering tasks to different vendors IPs seems not an effective decision.

Responses

Thank you for your advice. In this work, we focused on the application-specific tasks and our method is applicable to such tasks as well. We apologize for forgetting to provide the task model and explain how our method is suitable for application-specific tasks, and we include these descriptions in our revised manuscript. The detailed task model is presented in the lower-left and upper-right of page 4 in red color, and the follow paragraph also gives the task model.

In this work, we target embedded platforms which executes application-specific tasks and have high security requirements. Such platforms cannot handle tasks that occur suddenly at runtime, and they are widely-used in automotive, safety-critical systems, etc. In such systems, designers always have the prior knowledge of the application and its runtime constraints, which requires designers to perform security-driven customizations to meet performance and area requirements [27].

To deal with the application-specific tasks in task clustering stage, our proposed method only clusters tasks that can be executed using the same type of IP cores. For data-dependent tasks that require different types of IP cores, they cannot be clustered, and communications between them are always inter-core communications. This explanation is presented in the mid-left of page 6 in red color, which is also given as follows.

For the application-specific IPs, data-dependent tasks might have to be executed by different IP cores, resulting that the corresponding edge cannot be contracted. Therefore, we only focus the edges that can be contracted to reduce the schedule length. In the following discussion, we assume that all edges can be contracted for simplicity.

Comment 4

In Formula (1), the alpha parameter is not clearly defined. Is that a normalized parameter and how it can be obtained? What is the minimum/maximum of alpha?

Responses

In the previous manuscript, The following objective function was introduced.

The objective function of the above problem can be formulated as follows:

$$min: \ \alpha * risk_s + core \tag{8}$$

where $risk_s$ is the system security risk due to the unprotected communications, core is the number of cores required by the schedule, and α is a parameter large enough to keep the minimization of system security risk as the first priority.

The objective function was introduced to demonstrate that this work jointly optimizes both system security risk and the number of cores. However, we realized that these two

optimization targets do not have a trade-off relationship. Therefore, in this revised manuscript, we have removed this objective function.

Comment 5

The graph theory terms used in the paper need to be discussed based on their application in this paper.

Responses

Thank you very much for your suggestions. In this revised manuscript, we have included the discussions of graph theory terms, which are presented as follows.

(1) The TVG is presented in cyan color and located in the lower-right of page 5.

The **timing violated graph** ($TVG = (V_T, E_T)$) is then constructed by all paths from s to t whose lengths exceed the performance constraints, and it is an induced subgraph of TG.

(2) The $ECCG$ is presented in cyan color and located in the mid-right of page 6.

Then, a weighted **edge contraction conflict graph** ($ECCG = (V_E, E_E)$) is constructed to represent whether every pair of edges in TVG can be contracted simultaneously.

(3) MWIS of weighted $ECCG$, it is in cyan color in the lower-right of page 6.

The maximum weight independent set (MWIS) of the weighted $ECCG$ is calculated by the method proposed in [56], and the target is to a set of edges with maximum weight that can be contracted simultaneously.

(4) The $VCFG$ is presented in cyan color and located in the lower-left of page 7.

The **vendor conflict graph** ($VCFG = (V_c, E_{cf})$) is constructed from the performance-constrained clustering results, and it represents whether two clusters must be assigned to different vendors.

(5) The $VCPG$ is presented in cyan color and located in the upper-left of page 8.

The **vendor compatible graph** ($VCPG = (V_c, E_{cp})$) is the complement graph of the $VCFG$, and an edge in E_{cp} indicates that the connected clusters can be assigned to the same vendor.

Comment 6

Results shown in figure 10 do not show a breaking achievement for the proposed method, maybe, the method shows a better performance in a different benchmark. We still see that the system has a relatively high risk in many benchmarks. In several benchmarks, the numbers are very close except for robot and sparse benchmark. Benchmarks can be categorized by their application for easier comparison.

Responses

In this revised manuscript, the Section *Experimental Results* has been re-organized. We first demonstrate the effectiveness of our method in reducing system risks and the number of cores. Then, we test the system risk optimization results under different risk configurations. Finally, we show the results of saving the number of cores with both loose and tight vendor constraints. The benchmarks are categorized into two types, which are tasks graphs modelled from actual applications (the results are presented in Table IV), and random task graphs (the results are given in Table V). Furthermore, we also discussed the reasons why some task graphs still have a relatively high risk. The discussion is presented in the mid-left of page 11 in red color, which is also given as follows.

Table V illustrates the comparison results of five random generated task graphs. The number of edges in these task graphs are much larger then the edges of robot, sparse, and fpppp, and therefore the risk of each communication is relatively much smaller (refer to Equ. 7). The comparison results indicate that our proposed method also obtained the best results in reducing both the $risk_s$ and the number of cores.

Comment 7

This paper's English is poor and needs a proofread. I saw several grammatical errors here and there. For example, the very first sentence of the abstract says: "... and a set of security-driven constraints is (−− > are) imposed..."

Responses

Thanks for your valuable comments. We have carefully checked every sentences and improved the writing. Such as the first sentence of the abstract, the corrected version is "a set of security-driven constraints are imposed on task scheduling to protect MPSoCs against malicious modifications."

We hope this revised manuscript meets your requirements.

Reviewer 3

Comment 1

There is a lack of clarity in the motivation of the work, especially when the problem is getting introduced in the first section. Although, I understood the problem but it will help the readers if the motivation is rewritten. It should be concise and clear.

Responses

Thank you very much for your valuable suggestions. In this revised manuscript, we have made two modifications to better demonstrate our motivation.

The first modification is given in Section I *Introduction*, which is colored blue in the lower-right of page 1. This part of motivation is also given as follows.

The above mentioned works focused on area optimization mainly address the problem in task scheduling stage [25], and it is because that the number of cores required can

only be evaluated after vendor assignment. However, vendor assignment is a prior stage of task scheduling, whose results significantly affect the area optimization results in task scheduling, and ignoring the area optimization during vendor assignments limits the area optimization results. Furthermore, these researchers treated each communications equally when optimizing the system performance [29]–[31], but removing security constraints from different communications causes different security loss and communications with larger security importance should have higher protection priorities [28].

The second modification is given in Section III-C *Motivations*. This section has been rewritten to briefly explain our motivation, and it is presented in the lower-left of page 4.

Comment 2

Although the paper proposes a security-aware task schedule, however scheduling aspects on heterogeneous multicore platforms have not been discussed at all. I understand that the work is security aware, but a discussion on recent task schedulers for heterogeneous multicore platforms is required.

Responses

Thanks for your suggestions. In the Section II-B *Related Work* of this revised manuscript, we have discussed a number of studies that incorporate security constraints in task scheduling, and all these studies are for heterogeneous multicore platforms. The discussions of these studies are given as follows, which are also presented in the mid-left of page 3 in red color.

However, fulfilling the security constraints in task scheduling always incurs significant overheads of system performance, chip area and power of heterogeneous multicore platforms, and researchers have started to reduce these overheads of MPSoCs built from 3PIP cores through task scheduling. Rajmohan *et al.* [25] proposed a PSO-based hybrid evolutionary algorithm, and Sengupta *et al.* [26] proposed a bacterial foraging optimization-based design space exploration method to achieve the task schedule with higher security and less hardware overheads. Sun *et al.* [27] minimized the energy consumption while simultaneously protecting the MPSoC against the effects of hardware trojans with security constraints. Cui *et al.* [28] solved the online hardware trojan detection and recovery problem with graph-theoretic models that minimize the implementation cost of the design budget and area overhead. Liu *et al.* [29] proposed a set of task scheduling methods to reduce the increments of performance and hardware due to the security constraints. Wang *et al.* [30], [31] optimized the design budget and system performance with a minimized number of intra-core communications which are not protected by security constraints.

Comment 3

A symbol table is required which the readers can consult throughout the paper.

Responses

Thanks for your suggestion. We have added a table named "Descriptions of Notations" in Section III, which is given in the mid-right of page 5, and this table is quoted in the mid-right of page 5 in red color.

Comment 4

In the objective function (Equation 1), explain about alpha more. It is not clear.

Responses

In the previous manuscript, The following objective function was introduced.

The objective function of the above problem can be formulated as follows:

$$min: \ \alpha * risk_s + core \tag{9}$$

where $risk_s$ is the system security risk due to the unprotected communications, core is the number of cores required by the schedule, and α is a parameter large enough to keep the minimization of system security risk as the first priority.

The objective function was introduced to demonstrate that this work jointly optimizes both system security risk and the number of cores. However, we realized that these two optimization targets do not have a trade-off relationship. Therefore, in this revised manuscript, we have removed this objective function.

Comment 5

There are a few typos. For example, u.t.. on page 6, right column. Correct them.

Responses

Thanks for your valuable comments. We have carefully checked every sentences and improved the writing. We hope this revised manuscript meets your requirements.

Comment 6

The algorithms with whom the proposed work has been compared, need to be explained in a better way.

Responses

In this revised manuscript, we have followed your suggestion and provided a more detailed explanation of the algorithms that were compared. This explanation is presented in the mid-right of page 10 in red color, and it is also given as follows.

Our proposed method is then compared with two other methods to demonstrate the effectiveness. The first approach is "*Cluster-based approach*" (**C-B** for short) [29]. This approach optimizes the schedule length by placing critical

tasks on a single core, and then colors the performance-driven schedule to fulfill security constraints. The second approach is "*Min-cut-based approach*" (**MC-B** for short) [30]. This approach boosts performance by iteratively contracting the edges selected by the max-flow min-cut algorithm, then assigns tasks to IP vendor with traditional graph coloring method, and finally schedules tasks with the force-directed scheduling-based method.

Comment 7

In Section - Performance-Constrained Task Scheduling Results, why is the proposed work is faring better results? Analyze in a better way. It is true for other results as well.

Responses

In this revised manuscript, the Section *Experimental Results* has been reorganized. First, we present an overall evaluation of our method's effectiveness in reducing system risks and the number of cores (please refer to Section V-B). Next, we compare the system risk optimization results under different risk configurations, and our method is shown to achieve the highest level of system security with all risk configurations (please refer to Section V-C). Finally, we present the results of saving the number of cores with different vendor constraints, which demonstrate that our method requires the least number of cores under both loose and tight vendor constraints (please refer to Section V-D).

The experimental results are compared, and the reasons behind the results are also discussed. We hope this version of the *Experimental Results* section meets your requirements.

Reviewer 4

Comment 1

One of the problems is that the paper is a marginal paper compared to related work. Adding number of cores as a new objective is the main contribution which is not enough for being original for publication. Originality is limited.

Responses

Thanks for your comments. However, we believe that our work has made contributions that are worth publishing in IEEE TVLSI, and these contributions can be summarized as follows.

1) There are several related works that optimize system performance by clustering tasks, and some of them have observed that attacks on different communications can cause different system loss [25], [28]. However, little attention has been given to incorporating these different system loss (which we refer to as security risk) in the design process. This is the first work that optimizes system while accounting for security risk variations. Our work enables designers to provide a security-driven task schedule, where communications that have the potential to cause larger system loss are given higher priority for protection.
2) Incorporating security constraints into task scheduling incurs significant overheads of area, and reducing the number of cores is one of the effective methods to reduce the chip area. Different from existing works that attempt to reduce the number of cores after the vendor assignment stage, this work aims to reduce the number of cores starting from the vendor assignment stage, which expands the optimization space of reducing cores. Moreover, our proposed vendor assignment method evaluates the number of cores saved when clustering tasks, rather than estimating the number of cores required. This speeds up the processing time and provides better results.
3) This work can also handle the following scenarios. 1) The core speed variation, which occurs because cores from different IP vendors might have different speed. 2) Tight vendor constraints, which can occur when there are not enough IP vendors to purchase all required IPs. Moreover, purchasing the same function IP from many IP vendors can increase design costs and may exceed the design budget. 3) Application-specific IPs, which may not support every task. To address this, we build the $VCFG$ and $VCPG$ for each type of IP, and conduct the vendor assignment along with task scheduling individually.

We have included the aforementioned reasons in this revised manuscript. The first two reasons are presented in the mid-left of page 2 (highlighted in red), while the third reason is provided in the lower-right of page 9 (highlighted in red).

Comment 2

Reasoning behind the work and motivations are not strong and not acceptable sometimes. For example, authors said in Introduction that "... the growing number of mission-critical applications (e.g., finance and military) that use MPSoCs means that security is the highest priority issue, whereas the increasing integration of third-party Intellectual Property (3PIP) and the outsourcing of fabrication indicate that most MPSoCs are not 100% trustworthy" which brings some key questions: 1) Security is the highest priority in MPSoC-based mission-critical apps. Are you sure? How do you prove? Any references/evidences are required. 2) Although outsourcing of fabrication indicate MPSoCs are not 100% trustworthy, why should mission-critical applications use them (because we know they are not trustworthy. They need to use from trusty vendors)?

Responses

Q1. Security is the highest priority in MPSoC-based mission-critical apps. Are you sure? How do you prove? Any references/evidences are required.

A1. Thanks for your advice. In our previous manuscript, we used the phrase "security is the highest priority", which is not appropriate. However, many applications, such as banking and military systems, have high security requirements [51]. This explanation is also given in the lower-right of page 2 in blue color.

Q2. Although outsourcing of fabrication indicate MPSoCs are not 100% trustworthy, why should mission-critical applications use them (because we know they are not trustworthy. They need to use from trusty vendors)?

A2. The mission-critical applications still use the MPSoCs that are designed from 3PIPs. Similar comments can be found in [27], [51]. The reasons are summarized as follows:

(1) In practice, very seldom IPs are developed by the SoC integrator, and most of them are currently being designed offshore by 3PIP vendors [51].

(2) Integrating 3PIP cores allows designers to quickly respond to the increasing demands in energy consumption, functionality as well as programmability without sacrificing design productivity [27].

To better describe the situations that 3PIP cores are used in many applications that have high security requirements, the follow discussion is also presented in the lower-left of page 1 in red color.

This raises security concerns [2] since a small hardware modification by an adversary in the 3PIP cores can compromise the whole chip [3]. If such chips run time-critical applications (e.g. in autonomous vehicles), the hardware trojan attack may lead to catastrophic or life-threatening consequences [4]. Similarly, if these chips are used in information critical systems (eg. banking), the confidentiality and integrity of the user's data can be compromised [5].

Comment 3

This work is under concept of hardware trojans that any security issues should be detected by the literature in hardware trojan detection methods. Detection at high level of abstraction (scheduling) is not smart enough and has false positive or false negative ratios. Moreover, diversity over cores and others, are conventional methods for reliability purposes. Consequently, core-level reliability models (that are also popular in mission-critical applications) can reduce sensitivity to hardware trojans.

Responses

Q1. This work is under concept of hardware trojans that any security issues should be detected by the literature in hardware trojan detection methods.

A1. Actually, detecting all hardware trojans cannot be promised by these security constraints. What we suppose is that the communication protected by security constraints causes less security loss if attacked by the hardware trojans, and the unprotected communication faces larger security risks. Therefore, the target of this work is to reduce these risk to reach the performance constraints.

To clearly describe this, we have also included the following description is this revised manuscript, which is presented in the lower-left of page 5 in cyan color.

Although incorporating security constraints in the design process cannot guarantee the detection of all hardware trojan attacks, the risks caused by hardware trojans can be significantly reduced. In this paper, the security risk of a intra-core communication is regarded as the reduced risk after applying security constraints to this communication, and we assume that different communications face different security risks.

Q2. Detection at high level of abstraction (scheduling) is not smart enough and has false positive or false negative ratios. Moreover, diversity over cores and others, are conventional methods for reliability purposes. Consequently, core-level reliability models (that are also popular in mission-critical applications) can reduce sensitivity to hardware trojans.

A2. Thank you for your comments. We think that incorporating security constraints in high-level synthesis is also an effective way to protect the MPSoCs, and there exists a number of recent studies that focus on hardware trojan protection through task scheduling in context of security constraints [18], [22], [24]–[31].

Furthermore, the paper [18] gives a conclusion "a high-level synthesis is the ideal level for incorporating security constraints ", and this paper also gives the reasons, which are listed as follows.

1) The 3PIPs are typically designed and delivered as RTL. Furthermore, the targeted trojans enter the trustworthy SoC through these potentially untrusted 3PIPs.
2) security constraints are independent of the gate-level libraries and the implementation technology are structured as interactions between the 3PIPs.
3) Several high-level synthesis tools are available, thereby enabling incorporation of these constraints.

In this revised manuscript, the following description is also included to explain the reason why we incorporating security constraints at high-level of abstraction. This description can also be found in the mid-left of page 3 in blue color.

Security constraints for duplication, 3PIP vendor diversity have been recently proposed for hardware trojan protection, and high-level synthesis is the ideal level for incorporating security constraints [24].

Comment 4

The level of abstraction of the proposed solution i.e., scheduling method, is far away of scope of this journal (Trans. On VLSI).

Responses

Thanks for your comments, but the following three reasons make us believe that our work, which focuses on the task scheduling in context of security constraints, belongs to the scope of this journal (IEEE TVLSI).

(1) IEEE TVLSI has published a number of papers that incorporated security constraints in high-level synthesis, such as:

–J. Rajendran, O. Sinanoglu, and R. Karri, "Building trustworthy systems using untrusted components: a high-level synthesis approach," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 9, pp. 2946-2959, Apr. 2016.

(2) IEEE TVLSI has published a number of papers that focused on task scheduling, such as:

– J. Li, Y. Liu, H. Li, Z. Yuan, C. Fu, J. Yue, X. Feng, C.J. Xue, J. Hu, and H. Yang, "PATH: performance-aware task scheduling for energy-harvesting nonvolatile processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 9, pp. 1671-1684, Sep. 2018.

– Y. Wang, J. Liu, and J. Hu, "Communication-aware task scheduling for energy-harvesting nonvolatile processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 8, pp. 1796-1806, Aug. 2020.

(3) One of the Topics listed in the "**aims & scope**" of IEEE TVLSI is "**Cost, Performance Tradeoffs of VLSI/ULSI Systems**", and scheduling is one of the methods to realize the cost-performance tradeoffs. Therefore, we think that our work is within the scope of IEEE TVLSI.

Comment 5

Verification of the method is weak. How do authors ensure the method is effective against hardware trojans? What type of trojans is the main concern? How much effective is the method?

Responses

Q1. How do authors ensure the method is effective against hardware trojans?

A1. The effectiveness of security constraints in protecting MPSoC was demonstrated by [25], and there is several studies that incorporate security constraints in task scheduling [25]–[31]. These security constraints were proposed to enable MPSoCs to achieve two goals: (1) detect trojans that maliciously alter task outputs, and (2) mute trojan effects or prevent collusion between 3PIP cores from the same vendor. [29].

Additionally, these security constraints cannot guarantee that the system is 100% secure. However, communication without security constraints faces a greater security risk. Our objective is to minimize the total increased security risks of all unprotected communications, and the effectiveness of minimizing the increased security risk is demonstrated by the results in Tables IV and V (in page 11) and Fig. 9 (in page 12).

Q2. What type of trojans is the main concern?

A2. In this revised manuscript, the threat model is also presented, which can be found in Section III-A, mid right of page 3. This section includes the type of trojan and its purpose we focused on in this paper, which can be summarized as follows (also highlighted in cyan color in the mid-right of page 3).

In this work, we adopt the same threat model in [27], [29], which mainly focuses on detecting (or muting) malicious modifications. The trojan may cause the task running on the malicious 3PIP to either produce incorrect output or generate additional output to trigger trojans in another 3PIP core from the same vendor. As a result, the following two cases can occur at runtime: 1) Due to the insertion of the malicious logic into a 3PIP core, the outputs of the infected cores will be altered at some undetectable points, 2) Trojans distributed on multiple cores (in order to reduce the chance of being detected) can also form malicious communication paths where a malicious logic in one core triggers the trojans in another core using a secret communication channel.

Q3. How much effective is the method?

A3. To better illustrate the effectiveness of our proposed method, the Section *Experimental Results* has been reorganized in this revised manuscript. First, we present an overall evaluation of our method's effectiveness in reducing system risks and the number of cores (please refer to Section V-B). Next, we compare the system risk optimization results under different risk configurations, and our method is shown to achieve the highest level of system security with all risk configurations (please refer to Section V-C). Finally, we present the results of saving the number of cores with different vendor constraints, which demonstrate that our method requires the least number of cores under both loose and tight vendor constraints (please refer to Section V-D).

Comment 6

How is security risk evaluated (Fig. 4(C) red values)?

Responses

The security risk of each communication is the input of our problem, and our proposed method enables designers to handle communications with different security risks. We have stated this in Section III-D *Problem Description*, and it is in blue color in the lower-left and upper-right of page 5, which is also given as follows.

The inputs of this problem are task graph TG, vendor and performance constraints, core speeds of vendors, and **security risk of each communication**. The target is to find a schedule with the least system security risk, and the number of cores required is also optimized.

Traditional methods ignore variations in security risks and treat all communications equally. However, attackers typically select their targets of interest to attack, resulting in varying levels of security risks for different communications in most situations. As a result, MPSoC designers can prioritize communications with relatively higher security risks if they want to provide them with higher priority of protection.

Comment 7

Problem definition which is done in Section III is clear, however, the method of optimization is to solve dual objective problem with only one single objective optimization. As authors may know, this approach finds only one of possible solutions not all the cases. Moreover, this is the first time that I see performance parameter does not have more interested compared to security. Also, why are not there any defined constraints?

Responses

Q1. Problem definition which is done in Section III is clear, however, the method of optimization is to solve dual objective problem with only one single objective optimization

A1. Thank you very much for your suggestion, and we apologize for the confusion caused by the objective function in the previous manuscript. In the previous manuscript, The following objective function was introduced.

The objective function of the above problem can be formulated as follows:

$$min: \ \alpha * risk_s + core \tag{10}$$

where $risk_s$ is the system security risk due to the unprotected communications, core is the number of cores required by the schedule, and α is a parameter large enough to keep the minimization of system security risk as the first priority.

The objective function was introduced to demonstrate that our work optimizes both system security risk and the number of cores. However, we have realized that these two optimization targets do not have a trade-off relationship. Therefore, we have removed this objective function in this revised manuscript. Instead, we use the problem description (highlighted in blue in the lower-left and upper-right of page 5) to show the targets of this work, which is given as follows.

The inputs of this problem are task graph TG, vendor and performance constraints, core speeds of vendors, and security risk of each communication. The target is to find a schedule with the least system security risk, and the number of cores required is also optimized.

Q2. This is the first time that I see performance parameter does not have more interested compared to security

A2. We have set performance as a constraint in this work, and this indicates that performance is also a key consideration. Additional, one of the objectives in this work is to sacrifice the least amount of security to achieve the desired system performance, which also indicates the importance of performance.

Q3. Why are not there any defined constraints?

A3. Vendor constraints are also taken into account in this work. This is because we may not always have sufficient IP vendors to purchase IPs, and purchasing IP of the same function from multiple vendors can significantly increase the design cost. The vendor constraints are described in the problem description below, which is also highlighted in blue in the lower-left and upper-right of page 5.

The inputs of this problem are task graph TG, **vendor and performance constraints**, core speeds of vendors, and security risk of each communication. The target is to find a schedule with the least system security risk, and the number of cores required is also optimized.

Comment 8

Equation (3) means that how much in total e_{ij} will decrease time, however only critical path (longest path) is more important.

Responses

Equation (3) is given as follows.

$$w_{dly}(e_{ij}) = path_{tvg}(e_{ij}) * dly(e_{ij}) \tag{11}$$

where $path_{tvg}(e_{ij})$ is the number of paths in TVG that pass through e_{ij}.

The edge e_{ij} we discussed in equation (3) is the edge in timing violated graph (TVG), which consists of all tasks with negative slack time. The edges in TVG are in the paths whose delay exceeds the performance constraints. Optimizing the critical path (longest path) length can reduce the current schedule length, but if we try to reduce the schedule length to meet the performance constraints, all the paths in TVG have to be optimized sooner or later. Therefore, all the edges in TVG are treated equally when optimizing the performance constraints.

Comment 9

Equation (4) which divides delay by security does not have reasons and is strange (why are not differentiation or any other functions used? For example, root function with one as radicand and the other as index?).

Responses

Equation (4) is presented as follows.

The total weight that evaluates an edge e_{ij} contraction in TVG is denoted as $w(e_{ij})$, which is calculated as follows:

$$w(e_{ij}) = \frac{w_{dly}(e_{ij})}{risk(e_{ij})} \tag{12}$$

where $risk(e_{ij})$ denotes the security risk if e_{ij}. Fig. 4(c) illustrates the $w_{dly}(e_{ij})$ and $risk(e_{ij})$ in TVG, which are indicated next to the edges.

The purpose of Equation (4) is to calculate the weight of an edge, which evaluates the effects of contracting this edge. The effects of an edge contraction can be summarized into the following two categories:

1) The lengths of paths passing though this edge are reduced, which is evaluated by $w_{dly}(e_{ij})$;

2) The system security risk increases because the security constraint (isolation-with-diversity) is removed, which is denoted as $risk(e_{ij})$.

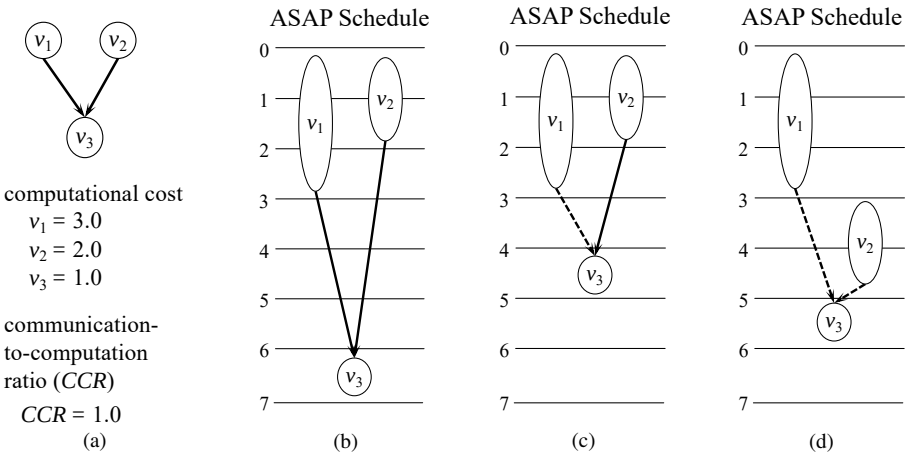


Fig. 10. Example of brother edge contraction. (a) Example of task graph. (a) ASAP schedule of the task graph. (c) ASAP schedule if $e_{1,3}$ is contracted. (d) ASAP schedule if $e_{1,3}$ and $e_{2,3}$ are both contracted.

In the performance optimization stage, we hope to choose the edge with larger $w_{dly}(e_{ij})$, while the increased security risk is smaller. Therefore, Equation (4) is introduced to evaluate the weight of contracting an edge.

In addition, we also added the discussion before introducing Equation (4) (please see the lower-left of page 6 in blue color), and it is given as follows.

Our target is to contract the edges with larger $w_{dly}(e_{ij})$, and the increased security risk $risk(e_{ij})$ is smaller. Therefore, the total weight that evaluates an edge e_{ij} contraction in TVG, which is denoted as $w(e_{ij})$, can be calculated as follows:

Comment 10

The constraint said as "If an edge is contracted during performance optimization, all its brother edges can no longer be contracted" is also hold for contacting a candidate edge. This constraint is not normal. Moreover, the reason said for this constraint did not consider that communication could be reduced for brother edge.

Responses

Thanks for your advice. We apologize for any confusion caused by our previous manuscript. The reason of introducing the constraint "If an edge is contracted during performance optimization, all its brother edges can no longer be contracted" can be explained as follows.

Suppose e_{ij} is an brother edge of a contracted edge e_{kj}, and this e_{ij} is in the critical path. Contracting e_{ij} reduces the communication delay in the critical path by $dly(e_{ij})$, but tasks v_k and v_i may both in the current critical path, resulting that the schedule length reduction SL_{re} to be.

$$SL_{re} = dly(e_{ij}) - dly(v_k) \tag{13}$$

The above equation means that SL_{re} may be negative, indicating the schedule length might be increased after contracting a brother edge. The above explanation is also demonstrated by the following example:

Fig. 10(a) gives a task graph, and the communication-to-computational ratio is set to be 1. Fig. 10(b) shows its ASAP schedule, whose schedule length is 7 *ut*. The critical path is $e_{1,3}$, and the ASAP schedule length can be reduced to 5 *ut* by contracting $e_{1,3}$ (see Fig. 10(c)). Then, the critical path becomes $e_{2,3}$, and $e_{2,3}$ is also a brother edge of contracted edge $e_{1,3}$. Contracting $e_{2,3}$ means that all these three tasks are executed sequentially in the same IP core, and the ASAP schedule increases to 6 *ut* (see Fig. 10(c)).

Because we want to avoid such situation, and therefore, the constraint "If an edge is contracted during performance optimization, all its brother edges can no longer be contracted" is introduced in our work. The above example is not introduced in the manuscript due the space limitation, and we use the following example (simpler version) to explain the reason. This explanation is presented in the upper-right of page 6 in red, which is also given as follows.

If an edge is contracted during performance optimization, all its brother edges can no longer be contracted. The reason is that contracting brother edges means the tasks that once could be executed parallel in different cores now must be executed sequentially in the same core, and this may result in an increased schedule length. For example, contracting the brother edges $e_{4,6}$ and $e_{4,7}$ in Fig. 4(c) makes v_6 and v_7 must be conducted sequentially in the same core, but they once can be computed at the same time in different cores.

Comment 11

Constraints listed as (1) and (2) in lines 42 and 44, respectively, do not make sense. It suppresses optimization solutions radically.

Responses

The constraints used in the previous manuscript when constructing $ECCG$ is listed as follows.

Two vertices in V_E are connected when their corresponding edges cannot be contracted simultaneously, under one of the following three situations:

1) *These two edges are brother edges;*
2) *These two edges belong to the same path in TVG.*

The purpose of introducing these two constraints is to minimize the induced security risk due to edge contraction, which can be explained as follows.

(1) The reason of introducing the first constraint: The detailed explanation can be found in the *response* to your *Comment 10* (in previous page).

(2) The reason of introducing the second constraint: We aim to avoid over-optimizing the path length, as doing so would result in additional security risk to the system. The following example explains the reason. Suppose that edges e_1 and e_2 belong to the same path in the TVG, and contracting either e_1 or e_2 would make the path length smaller than the performance constraint. However, if both e_1 and e_2 are contracted, the path length is over-optimized, and this causes additional system risk increment. The second constraint is used to avoid the above situation.

To better describe our purposes of introducing these constraints, we also added some breif descriptions in this revised paper (in the mid-right of page 6 in red colored), which are also given as follows.

Two vertices in V_E are connected when their corresponding edges cannot be contracted simultaneously, under one of the following two situations:

1) these two edges are brother edges (with respect to the multicore parallel execution);
2) These two edges belong to the same path in TVG (to prevent the over optimization of path length).

Moreover, explanations of introducing these two constraints are also included in this revised manuscript (in the upper-right of page 6 in blue color), which are presented as follows.

However, not all edges can be contracted with respect to the multicore parallel execution. Let $in\_edge(v)$ be the set of edges that end with v, and let $out\_edge(v)$ be the set of edges that start from v. Edges in TG that belong to the same $in\_edge(v)$ or $out\_edge(v)$ are called **brother edges**. If an edge is contracted during performance optimization, all its brother edges can no longer be contracted. The reason is that contracting brother edges means the tasks that once could be executed parallel in different cores now must be executed sequentially in the same core, and this may result in an increased schedule length. For example, contracting the brother edges $e_{4,6}$ and $e_{4,7}$ in Fig. 4(c) makes v_6 and v_7 must be conducted sequentially in the same core, but they once can be computed at the same time in different cores.

In addition, two edges belong to the same path in TVG should not be contracted simultaneously, and this avoids the over optimization of path length, which causes additional system security risk. The following example explains the reason. Suppose that edges e_1 and e_2 belong to the same path in TVG, and contracting either e_1 or e_2 will make the path length smaller than the performance constraint. However, if e_1 and e_2 are both contracted, the path length is over optimized, and this causes additional system risk increment.

Comment 12

All instances of trojan are written with capital <T>.

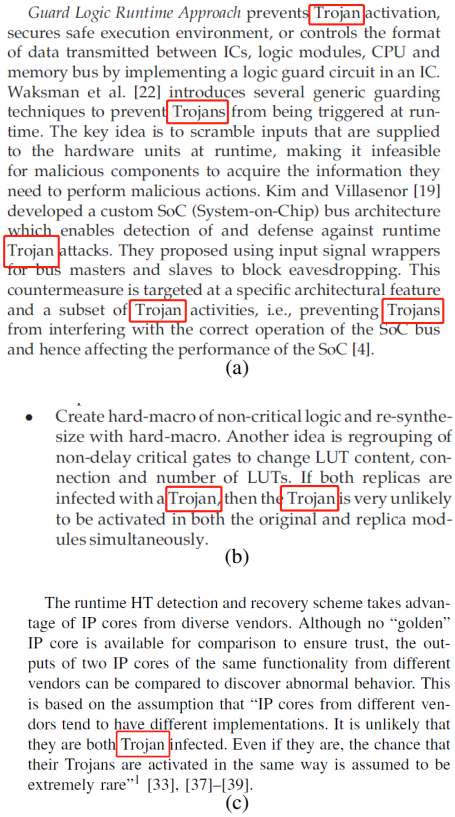
Responses

Thank you for your suggestion. We have revised the manuscript accordingly, and corrected this writing in the revised manuscript.

The reason that we wrote "Trojan" in the previous manuscript is that:

We noticed that some researchers have consistently used "Trojan" throughout their papers, such as "Run-time monitoring and validation using reverse function (RMVRF) for hardware trojans detection, IEEE Transactions on Dependable and Secure Computing, 2021", "Applying chaos theory for runtime hardware trojan monitoring and detection, IEEE Transactions on Dependable and Secure Computing, 2020", "Toward building and optimizing trustworthy systems using untrusted components: a graph-theoretic perspective, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020". Hence, in our previous manuscript, we wrote "Trojan" with capital <T>.

Fig. 11 presents examples of "Trojan" writings found in the papers mentioned above.



Guard Logic Runtime Approach prevents Trojan activation, secures safe execution environment, or controls the format of data transmitted between ICs, logic modules, CPU and memory bus by implementing a logic guard circuit in an IC. Waksman et al. [22] introduces several generic guarding techniques to prevent Trojans from being triggered at runtime. The key idea is to scramble inputs that are supplied to the hardware units at runtime, making it infeasible for malicious components to acquire the information they need to perform malicious actions. Kim and Villasenor [19] developed a custom SoC (System-on-Chip) bus architecture which enables detection of and defense against runtime Trojan attacks. They proposed using input signal wrappers for bus masters and slaves to block eavesdropping. This countermeasure is targeted at a specific architectural feature and a subset of Trojan activities, i.e., preventing Trojans from interfering with the correct operation of the SoC bus and hence affecting the performance of the SoC [4].

(a)

- Create hard-macro of non-critical logic and re-synthesize with hard-macro. Another idea is regrouping of non-delay critical gates to change LUT content, connection and number of LUTs. If both replicas are infected with a Trojan, then the Trojan is very unlikely to be activated in both the original and replica modules simultaneously.

(b)

The runtime HT detection and recovery scheme takes advantage of IP cores from diverse vendors. Although no "golden" IP core is available for comparison to ensure trust, the outputs of two IP cores of the same functionality from different vendors can be compared to discover abnormal behavior. This is based on the assumption that "IP cores from different vendors tend to have different implementations. It is unlikely that they are both Trojan infected. Even if they are, the chance that their Trojans are activated in the same way is assumed to be extremely rare"[1] [33], [37]–[39].

(c)

Fig. 11. Examples of "Trojan" writings. (a) "Trojan" writings in "Run-time monitoring and validation using reverse function (RMVRF) for hardware trojans detection". (b) "Trojan" writings in "Applying chaos theory for runtime hardware trojan monitoring and detection". (c) "Trojan" writings in "Toward building and optimizing trustworthy systems using untrusted components: a graph-theoretic perspective".

Comment 13

In the following descriptions, the duplicated task of v_i is denoted as v_i' is meaningless.

Responses

Thanks for your suggestion, and we have removed the sentence in this revised manuscript.

Comment 14

Fulfills security constraints at the finest granularity, but this incurs significant overheads of system performance and area What does the sentence mean?

Responses

"Fulfills security constraints at the finest granularity" is a conclusion drawn from the reference "Shielding Heterogeneous MPSoCs From Untrustworthy 3PIPs Through Security Driven Task Scheduling, 2017" [29]. We apologize for the confusion caused by our improper writings, as the readers might not have the necessary context of that paper.

Therefore, This sentence has been modified as follows. "Fulfilling security constraints incurs significant overheads of performance and area". This modified sentence is also presented in cyan color in the mid-left of page 4.

Reviewer 5

Comment 1

The biggest concern is that some of the key experimental configurations are overly simplified, which make the contribution claim of the manuscript weaker. For example, vendor count is a new input that is added to the scheduling process, compared to the conference version. However, in experiments all vendor counts are just set to be the same as the maximum clique size of the benchmark. This setting simplifies the problem as the maximum clique size should just be a loose upper bound of vendor requirement. I will expect the scheduling scheme to deal with more restricted vendor constraint, which is also more realistic: 6 or 7 vendors as shown in Fig. 9 is just impossible.

Responses

Thank you for your valuable suggestion. In the Section V-D *Comparison of Cores Required* (refer to page 12), we present the effectiveness of our method in reducing the number of cores under both loose and tight vendor constraints. A "loss" vendor constraint means that the vendor constraint is set to the maximum clique size of the task graph. When the "tight" vendor constraint is set, it is set to 2 for all benchmarks. The description of these two types of vendor constraints is also provided in the mid-right of page 12 in blue color.

The results presented in Table VI demonstrated that our proposed method outperforms C-B [29] and MC-B [30] under both loose and tight vendor constraints.

Comment 2

Another example of the overly simplified setting is the assignment of security risk. The variance of security risk is a main differential point compared to the conference paper. However, this manuscript just assumes the security risk of a communication is linearly proportional to the distance from source. This claim appears to be quite baseless. Is there a reference to support such an assumption? If not, would it be better to test different ways of assigning security risks to edges, for example the edged closer to the source have higher security risk, or randomly assign security risk to all edges. Then we will see how well the proposed scheme works under different settings.

Responses

In the Section V-C *Comparison of Security Risk Optimization Results* (refer to page 11) of this revised manuscript, we present a comparison of security risks with three sets of risk configurations. These three risk configurations are given as follows.

1) **Randomly Set**: the risk of each communication is randomly set with the value among $[0, \frac{2*pte}{m}]$;
2) **Equally Set**: the risks of all communications are the same, with the value of $\frac{pte}{m}$;
3) **Linearly Set**: The communication closer to the source have higher security risk, and the risk of e_{ij} is set as $risk(e_{ij}) = \frac{2*(SL-dist(e_{ij}))*pte}{SL*m}$;

The experimental results given in Fig. 9, which is discussed as follow (also can be found in the mid-left of page 12 in red color), indicate that our proposed method is effective under different risk configurations.

1) When the risks are *randomly set*, the averaged system security risk of C-B and MC-B are $36.72*10^{-3}$, and $23.02*10^{-3}$, respectively, and our proposed method obtained the lowest security risk, which is only $18.11*10^{-3}$.
2) If the risks are *equally set*, our method and MC-B obtains almost equivalent results, and it is because that MC-B treats each communication equally when optimizing the system performance. The averaged system security risks of C-B, MC-B and our methods are $37.47*10^{-3}$, $22.27*10^{-3}$, and $20.94*10^{-3}$, respectively.
3) If the risks are *linearly set*, our proposed method still show its advantage in reducing the system risks, and its average system risk is $18.68*10^{-3}$, while the averaged system risk of C-B and MC-B are $41.08*10^{-3}$ and $26.03*10^{-3}$, respectively.
4) For some benchmarks (robot, sparse), their system security risks are relatively high, and this is because the risks of communications are also determined by the number of edges in task graphs. These task graphs have

much less edges, making the risks of communications in *robot* and *sparse* much larger than that in the other task graphs.

Comment 3

Section V.C and V.D appear to be somewhat redundant compared to Section V.B. Security risk, core counts, and CPU runtime are already compared in Table II and Fig. 9. Why not just discuss based on the existing data instead of padding data with very similar experiment configurations?

Responses

Thank you for your valuable suggestion. In this revised manuscript, the Section *Experimental Results* has been reorganized. First, we present an overall evaluation of our method's effectiveness in reducing system risks and the number of cores. Next, we compare the system risk optimization results under different risk configurations, and our method is shown to achieve the highest level of system security with all risk configurations. Finally, we present the results of saving the number of cores with different vendor constraints, which demonstrate that our method requires the least number of cores under both loose and tight vendor constraints.

Comment 4

Scheme PSO-B is mentioned as one of the previous works for comparison, but it is only compared in Table IV. I will suggest removing it instead of performing incomplete comparison here.

Responses

Thank you for your suggestion. We have removed the PSO-B method from this revised manuscript.

Comment 5

Equation (1) shows the security risk and core count are linearly combined as a single objective function. However, there is no discussion of what value is selected as alpha, and Table II only compares the reduction of core count instead of the optimization objective.

Responses

In the previous manuscript, The following objective function is introduced.

The objective function of the above problem can be formulated as follows:

$$min: \ \alpha * risk_s + core \tag{14}$$

where $risk_s$ is the system security risk due to the unprotected communications, core is the number of cores required by the schedule, and α is a parameter large enough to keep the minimization of system security risk as the first priority.

The objective function was introduced to demonstrate that this work jointly optimizes both system security risk and the number of cores. However, we realized that these two optimization targets do not have a trade-off relationship. Therefore, we have removed this objective function in this revised manuscript.

Reviewer 6

Comment 1

The authors must provide more references, as they miss out several important ones in Section I and II (like authentication techniques, etc.).

Responses

Thank you for the your advice. The main types of hardware trojan countermeasures (including authentication techniques) are also discussed in Section II-A *Hardware Trojans and Countermeasures*, located in the mid-right of page 2 (in red color). These discussions are also presented as follows.

Numerous and various functional and parametric tests are required to verify whether a 3PIP contains hardware trojans. However, testing a black-box component is very difficult and time-consuming, and it is impractical to do such an exhaustive test for a large and complex design. Therefore, a number of countermeasures have been developed against hardware Trojans at design stage [38]. Hardware security primitives use random number generator (TRNG) [39] or physical unclonable function (PUF) [40] to provide built-in self authentication against various threats and vulnerabilities arising at different phases of IC life cycle [41], [42]. System and architectural protection techniques prevent information leakage through shared resource [43] and build trusted execution environment [44]. Side-channel protection techniques introduces noise or randomization in the software implementation to eliminating side-channel leakage [45], [46]. IP protection techniques use hardware watermarking or steganography to protect an IP against threats [47], [48]. Machine learning-assisted designs provide defenses against hardware security threats or enhance the systems robustness [49], [50].

Comment 2

In Section II, the authors need to include another sub-section related to hardware trojan attacks on real time systems, which includes hardware trojan attacks on real time task schedules, etc and their related security strategies. This will provide a completeness to the literature survey.

Responses

In this revised manuscript, we have also included a discussion of hardware trojan attacks in Section III-A *Threat Model*, which is located in the mid-right of page 3, magenta

colored. This threat model includes an explanation of how hardware trojans can attack real-time systems, and the references you suggested are also cited in this revised manuscript. The description of this threat model is provided as follows.

Hardware trojan attacks are intended to affect normal circuit operation, potentially with catastrophic consequences in critical applications in the domains of communications, banking, space and military [32]. They can also aim at leaking secret information from inside a chip through covert channels or affect the reliability of an IC through undesired process changes that cause device ware-out and long-term reliability issues [33]. In addition, they can be used to assist software attacks by providing hardware back-door, and make the system operate in an incorrect way, such as modifying the scheduling results of a real-time system [34]–[37].

Comment 3

The authors need to present a table on how their methodology surpasses the others in Section II.

Responses

In this revised manuscript, we have included a table (Table I on page 3) in Section II *Related Work* that presents a comparison between our proposed technique and other design-for-trust techniques that implement security constraints. The comparison is discussed in the lower-left of page 3 (blue colored), and it is also presented as follows.

The comparison between the proposed technique and other techniques that implement security constraints is summarized in Table I. The security constraints cause significant overheads of power, delay, and area [18], [22], and a number of techniques were developed to optimized these overheads, but the optimization space turns to be very limited [25]–[28]. Recent researchers treat security constraints as loose constraints (some constraints that can be violated) to achieve tight design targets, but they forget to minimize the induced security risks [29]–[31]. In this work, our proposed method can provide solutions with smaller circuit area under tight delay constraints. Moreover, our approach optimized system security risk, which is always ignored by the conventional approaches.

Comment 4

The authors claim to work with tasks. Hence, give a proper task description. They must explicitly state their task model, which is not present in the current work. Tasks may be of several types, periodic, non-periodic...

Responses

In this work, we target embedded platforms which executes application-specific tasks. We have added this explanation in the revised manuscript (lower-left and upper-right of page 4 in red color), which is also given as follows.

In this work, we target embedded platforms which executes application-specific tasks and have high security requirements. Such platforms cannot handle tasks that occur suddenly at runtime, and they are widely-used in automotive, safety-critical systems, etc. In such systems, designers always have the prior knowledge of the application and its runtime constraints, which requires designers to perform security-driven customizations to meet performance and area requirements [27].

Comment 5

The authors need to present the difference from their conference version in the main text of the paper (In Introduction section), and cite their previous work.

Responses

Thank you for your valuable suggestion. In this revised manuscript, we have included a discussion of the differences from our conference version in Section I *Introduction* (in the mid-left of page 2 in red color), and we have cited the previous work. These discussions are also presented as follows.

The earlier conference version of this paper appeared at [31]. Compared with [31], this paper has a number of new contributions.

1) This work treats communications with different security importance, and provides a schedule with a minimized system security risk.
2) Rather than assigning one task to an IP vendor each time, this vendor assignment method groups all tasks into a number of *vc* (vendor constraint) clusters and assigns an entire cluster to an IP vendor at a time. Furthermore, this vendor assignment method evaluates the number of cores saved when clustering tasks rather than estimating the number of cores required, which speeds up the processing time and provides better results.
3) This work considers core speed variation in the design process. All tasks are first assumed to be performed with the slowest speed, and the vendor assignment will be adjusted after the exact core speeds are determined to further reduce the system security risk.

Comment 6

The authors need to state the complexities of their proposed algorithms.

Responses

In this revised manuscript, we have included the detailed time complexities in Section IV *Time Complexity Analysis*. This analysis is also presented as follows.

The time complexity of our proposed method is analyzed as follows, and the input task graph has n nodes and m edges.

In each iteration of the performance-constrained task clustering stage, constructing *ECCG* from *TVG* requires

$O(m^2)$, and finding the MWIS in $ECCG$ also requires $O(m^2)$ [56]. Only a constant number of iterations are conducted before reaching the performance constraint, and finding all contracted edges to meet the performance constraint totally needs $O(m^2)$. In addition, each time before contracting an edge, updating $VCFG$ and evaluating the impact on the maximum clique size of $VCFG$ need $O(n^2)$, and only a limited number of edges are contracted, making its computational cost $O(n^2)$. The total time complexity of performance-constrained task clustering is $O(m^2)$ (due to $O(n) \leq O(m)$).

In the vendor assignment and task scheduling stage, constructing $VCFG$ and $VCPG$ requires $O(n^2)$. In each iteration of merging clusters, $O(m)$ is required to estimate the maximum clique size, and $O(n)$ is needed to update both $VCFG$ and $VCPG$. Vender assignment requires $O(n)$ iterations of merging clusters, and its time complexity is $O(mn)$. Performing the force-directed scheduling method to schedule all tasks needs $O(n^2)$, and the total time complexity of the vendor assignment and task scheduling stage is $O(mn)$.

The sum of $O(m^2)$ and $O(mn)$ is $O(m^2)$, and it is the total time complexity of our proposed method.

Comment 7

Can this mechanism handle tasks which occurs suddenly at runtime? If not, the authors must state this.

Responses

Q1. Can this mechanism handle tasks which occurs suddenly at runtime?

A1. This mechanism only handle application-specific tasks, and therefore, cannot handle tasks that occur suddenly at runtime.

Q2. If not, the authors must state this.

A2. We have stated that our mechanism cannot handle tasks that occur suddenly at runtime. This statement is presented in upper-right of page 4 (in cyan), and it is also presented as follows.

Such platforms cannot handle tasks that occur suddenly at runtime, and they are widely-used in auto-motive, safety-critical systems, etc.

Comment 8

Duplication with diversity mechanism can guarantee full proof results and can handle any fault related issues at runtime. The authors must also state whether their proposed mechanism can mitigate errors and bypass faults, if any occur at runtime. If so, how?

Responses

Q1. Duplication with diversity mechanism can guarantee full proof results and can handle any fault related issues at runtime.

A1. The duplication with diversity mechanism cannot handle fault-related issues at runtime, and this mechanism can only be used to detect the faults caused by hardware trojans.

Q2. The authors must also state whether their proposed mechanism can mitigate errors and bypass faults, if any occur at runtime. If so, how?

A2. In the 'Recovery' column of table I (page 3), we have stated that our proposed method has "No" ability to recover from the errors.
