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

Nan Wang, *Member, IEEE,* Lijun Lu, Songping Liu, Hongqing Zhu, *Member, IEEE,* and Yu Zhu, *Member, IEEE,*

Abstract—The high penetration of third-party intellectual property in MPSoCs gives rise to security concerns, and a set of security-driven constraints is imposed into task scheduling step of the design process to protect MPSoCs against hardware Trojan attacks. Due to the significant performance and area overheads incurred, designers start to selectively apply security-driven constraints to achieve the design targets, but they often ignore that parts of a design may be more vulnerable to hardware Trojan attacks. In this study, the differences in vulnerability to hardware Trojan attacks are also considered in the MPSoC design process, and a security-driven task scheduling method is proposed to minimize both the design vulnerability and chip area under deadline constraints. First, the schedule length is iteratively optimized by a maximum weight independent set-based method that minimizes the vulnerability increment. Second, tasks are assigned to IP vendors with a minimized number of cores required 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 deadline constraints.

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

1 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 electronic components to track the most cutting edge technology while reducing manufacturing costs [1]. This has given rise to the trend of outsourcing the design and fabrication of third-party intellectual property (3PIP) components, which may not be trustworthy, and the hardware Trojans (HTs) in these 3PIP components present high risks of malicious inclusions and data leakage in products [2]. This raises security concerns [3] because a small hardware modification by an adversary in the 3PIP cores can compromise the whole chip [4]. If such chips run safety-critical applications (e.g., autonomous vehicles), the HT attacks may lead to catastrophic or life-threatening consequences [5]. 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 [6].

Emerging security problems bring an urgent need to detect possible HT attacks or mitigate their effects. Methods for detecting HTs can primarily be classified into the following groups: physical inspection [7], functional testing [8], built-in tests [9], and side-channel analyses [10]. However, it is impossible to detect advanced HTs, such as A2, due to its insertion stage and software triggered mechanism [11].

Modular redundancy is an HT-tolerance technique that provides comprehensive protections to circuits and verify the correctness of system functionality at runtime [12]–[15]. 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 HTs and enable trustworthy computations using untrusted 3PIP cores [16]–[18]. This is achieved by duplicating tasks and mapping them on 3PIP cores from different vendors to detect HTs that alter task outputs or mute potential HT effects by preventing collusion between malicious 3PIP cores from the same vendor. But these security constraints in the design stage incur significant overheads (e.g., approximately 200% area and 50% performance overheads [19]). As each task needs to be conducted duplicately to ensure the correctness of the outputs, and this brings significant redundant computation cost; all data-dependent tasks must be computed by the cores from different vendors to establish trustworthy communications, and all these communications are inter-core communications with long delays. Therefore, researchers have developed a number of solutions and created trusted designs with minimum resource overheads, performance degradation and energy consumption [19]–[21].

Some researchers have also started to consider security constraints as loose constraints (security constraints are not applied to all tasks and communications) to satisfy the design targets [22]–[24]. However, their studies ignore that parts of a design are much more vulnerable to HT attacks [25], and removing security constraints from the parts of a circuit that are more susceptible to Trojan insertion may yield significant security losses [19]. Furthermore, these studies only optimize the system performance in the context of security constraints, and might incur a significant area overhead, which is also one of the critical issues towards trusted design. Therefore, performance and security along with chip area should be jointly considered for MPSoC design, especially for heterogeneous MPSoCs built from untrusted 3PIPs.

This study extends our prior work [24] which considered only performance constraints in security-aware task scheduling, by co-optimizing design vulnerability against HTs and the number of required cores. A three-step design method that consists of task clustering, vendor assignment and task scheduling is proposed to enable MPSoC designers to achieve a high-security design with the desired performance and optimized chip area. The contributions of the paper are summarized as follows:

This work was supported by the National Key R&D Program of China under Grant 2022YFD2000400.
Nan Wang, Lijun Lu, Songping Liu, Hongqing Zhu and Yu Zhu are with the School of Information Science and Engineering, East China University of Science and Technology, Shanghai, 200237, China.

1) This study treats the communications between tasks with different vulnerabilities against HT attacks, and a maximum weight independent set-based method is proposed to minimize the design vulnerability under the deadline constraint, by iteratively selecting a set of maximum weighted inter-core communications and assigning them to intra-core communications with much smaller delays.
2) 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. Furthermore, the proposed 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 and provides better results.
3) This study considers core speed variation in the task scheduling process. All tasks are first assumed to be performed with the slowest speed, and after the exact core speeds are determined, the vendor assignment will be adjusted to assign the unprotected communications with security constraints in descending order of vulnerability, which further reduces the total vulnerability of the design.

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.

2 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. The outsourced design and test services, as well as electronic design automation software tools supplied by different vendors, also make circuits vulnerable to malicious implants.

2.1 Security Countermeasures

Numerous and various functional and parametric tests are required to verify whether a 3PIP contains HTs. 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 HTs at the design stage [26]. Hardware security primitives provide built-in self-authentication against various threats and vulnerabilities arising at different phases [27]. System and architectural protection techniques prevent information leakage through hardware isolation and build trusted execution environments [28]. Side-channel protection techniques introduce noise or randomization in the software implementation to eliminate side-channel leakage [29]. IP protection techniques use hardware watermarking or steganography to protect an IP against threats [30]. Machine learning-assisted designs provide defenses against security threats or enhance robustness [31].

Although HT detection methods are implemented in different design stages, finding all HTs cannot be guaranteed even with the most cutting-edge technologies. However, many applications, such as banking and military systems, have high security requirements [32]. Therefore, Trojan-tolerant design methodologies are another way to protect designs from HT attacks [1].

2.2 Modular Redundancy for Trojan-tolerant Design

The modular redundancy is an HT-tolerance strategy used in hardware design to mitigate the effects of hardware Trojan attacks during runtime [12]. The modular redundancy instantiates copies of the design along with voter or comparator logic, and in the case of a different output, the voting or comparator circuit determines the correct output. Researchers have implement modular redundant designs in different structures with different constraints. Reece *et al.* [13] identified HTs through comparisons of two similar untrusted designs by testing functional differences for all possible input combinations. Beaumont *et al.* [14] developed an online HT detection architecture that implements fragmentation, replication and voting. Shatta *et al.* [15] presented methodologies that detect the errors caused by HTs in 3PIPs using voters, and recover the system by replacing the errors. Unfortunately, modular redundancy suffers from performance degradation and the overhead of extra logic, which results in increase in power, area and energy [12].

2.3 Security-Driven Task schedules for MPSoC

With the modular redundant techniques, many studies have attempted to detect malicious outputs by duplicating tasks and to avoid HT collusion between IP cores from the same vendor. Incorporating the above design constraints (i.e., security constraints) in the tasks scheduling process for MPSoC that uses untrusted 3PIPs has attracted the attention of researchers. Rajendran *et al.* [16] identified design constraints for HT detection to achieving detection, collusion prevention, and isolating the HT-infected 3PIP, and incorporated them during high-level synthesis. Cui *et al.* [17] implemented both HT detection and error recovery at runtime for mission-critical applications, using recomputation with IP cores from different vendors. Rajmohan *et al.* [18] proposed an HT detection and activation prevention mechanism for high level synthesis, using a particle swarm optimization-based design space exploration method.

However, fulfilling the security constraints in task scheduling may result significant overheads in system performance, chip area, and power consumption. As every task is computed duplicately, and all communications become inter-core communications with long delays. Therefore, researchers have started to reduce these overheads along with optimizing the system security. Cui *et al.* [19] solved the online HT detection and recovery problem with graph-theory models that minimize the implementation cost of the design budget and area overhead. Sengupta *et al.* [20] proposed a bacterial foraging optimization-based design space exploration method to find a task schedule with higher security and less hardware overhead. Sun *et al.* [21] minimized the energy consumption while simultaneously protecting the MPSoC against the effects of HTs with security constraints. Liu *et al.* [22] proposed a set of task scheduling methods to reduce the increments of performance and hardware due to security constraints. Wang *et al.* [23], [24] optimized the design budget and system performance with a minimized number of unprotected communications.

To further optimize the design targets, some researchers also treat security constraints as loose constraints (constraints are not applied to some tasks or communications) during task scheduling, but they forget to minimize the induced design's security losses [22]–[24]. In the chip design, HT implanters intend to attack the parts of circuits with higher vulnerabilities to create larger damages to the systems or leakage the confidential information, and the vulnerabilities of tasks or communications can differ by 10^3 times in the same benchmark [25]. This indicates that the design's performance and area can be further reduced with a small penalty of vulnerability increment by removing some "proper" security constraints from tasks and communications.

3 Preliminaries and Problem Description

Designers may need to use untrusted 3PIP cores to build trustworthy system, where the application is partitioned into a series of tasks and these tasks are scheduled to time periods and bound to IP cores. Task scheduling mechanisms are designed to provide high security and low cost of the system, and the task scheduling problem we considered is presented in this section.

3.1 Threat Model

HT attacks are intended to affect normal circuit operation, potentially with catastrophic consequences in critical applications in the domains of banking, space and military [33]. They can also aim to leak secret information from inside a chip through secret channels or affect the reliability of a circuit through undesired process changes [34]. From the perspective of the activation methods, HTs can be classified as either *always-on* or *conditionally triggered*. An always-on Trojan may be 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 [22].

In this study, we adopt the same threat model in [21], [22], which primarily focuses on detecting or mitigating malicious modifications. The HT may cause the task running on the malicious 3PIP to either produce incorrect output or collude with Trojans in another 3PIP core from the same vendor. As a result, the following two cases can occur at runtime: 1) *Malfunction:* due to the insertion of the malicious logic into a 3PIP core, the outputs of the infected cores will be altered at some unexpected points; 2) *Trojan collusion and Trojan triggering between Cores:* Trojans that are distributed on multiple cores to reduce the chance of being detected, and some malicious communication paths can also be established between cores by writing illegal values to certain secret memory space. Therefore, with these secret communication paths, a malicious logic in one core can trigger the Trojans in another core, and the active HTs in different cores can collude to cause catastrophic consequences to the systems.

In this study, we target embedded platforms which execute application-specific tasks and have high security requirements, and such platforms are widely-used in auto-motive, safety-critical systems, etc [21]. The SoC used in these systems are vulnerable to various HT attacks when the untrusted 3PIPs get integrated into this SoC. Because the HTs in 3PIPs could be passed down the design cycle to post-silicon and all the fabricated chips contain such HTs [35]. In such security-critical systems, designers always have prior knowledge of the application and its runtime constraints, and they can perform security-aware design to meet performance requirements and reduce the chip area. In addition, designers also have the ability to purchase 3PIPs from different vendors and implement design techniques to improve security.

HDL codes	Statement weight [value range]	
	X	***Z***
1. PORT(CLK: IN BIT; 2. ***X*** : IN INTEGER RANGE 15 DOWNTO 0; 3. ***Z*** : OUT INTEGER RANGE 15 DOWNTO 0); 4. PROC1: PROCESS(CLK)	/	/
5. BEGIN	{1[0,15]}	{1[0,15]}
6. FOR ***X*** IN 0 TO 10 LOOP	{1[0,15]}	{1[0,15]}
7. IF (***X*** < 3) THEN	{0.6875[0,10]}	{1[0,15]}
8. ***Z*** <= ***X***;	{0.1875[0,2]}	{0.1875[0,2]}
9. ELSEIF (***X*** > 5) THEN	{0.6875[0,10]}	{0.1875[0,2]}
10. ***Z*** <= 14 - ***X***;	{0.3125[6,10]}	{0.3125[4,8]}
11. END IF;	{0.6875[0,10]}	{0.6875[0,2] U [4,8]}
12. END LOOP;	{1[0,15]}	{0.6875[0,2] U [4,8]}
13. END PROCESS PROC1;	{1[0,15]}	{0.6875[0,2] U [4,8]}

Fig. 1. Example of vulnerability analysis.

3.2 Security Constraints

Runtime validation approaches provide a last line of defense against potentially undetected HTs [3], and integrating security constraints in the task scheduling process enables the runtime validation using untrusted 3PIPs. Two types of security constraints, which are also introduced in [18]–[24], are used in task scheduling for runtime detection and mitigation of HT attacks: 1) *duplication-with-diversity* constraints are used to detect Trojans that tamper with program outputs; 2) *isolation-with-diversity* constraints are employed to prevent HT collusion between 3PIP cores such as leaking information via secret communication paths. The effectiveness of the security constraints in detecting the deliberate faults caused by HTs and isolating the active HTs are explained in [16].

3.2.1 Duplication-With-Diversity

To detect HTs that alter the task outputs at unexpected time, 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). This type of security constraints ensures the trustworthiness of task outputs [36].

Duplication-with-diversity is set based on the fact that the probability of Trojans implanted by different attackers having the same trigger 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 [37]. Therefore, the cores will not produce the same incorrect output under the same input if the malicious HT is activated, and the presence of the implanted HT is detected when there is a mismatch in the outputs.

3.2.2 Isolation-With-Diversity

To hide Trojan footprints, attackers may distribute Trojans in multiple IP cores and construct secret communication paths between IP cores to leak information or to trigger the hibernating Trojans. These secret communication paths between IP cores from the same vendor cannot be acquired by other vendors [16]. Although redundant execution approaches, including voting architecture [14], dual/triple modular redundancy [36], and duplication-with-diversity, can detect HTs by comparing the outputs of cores from different vendors with the same input, they cannot cut off secret communications between multiple IP cores.

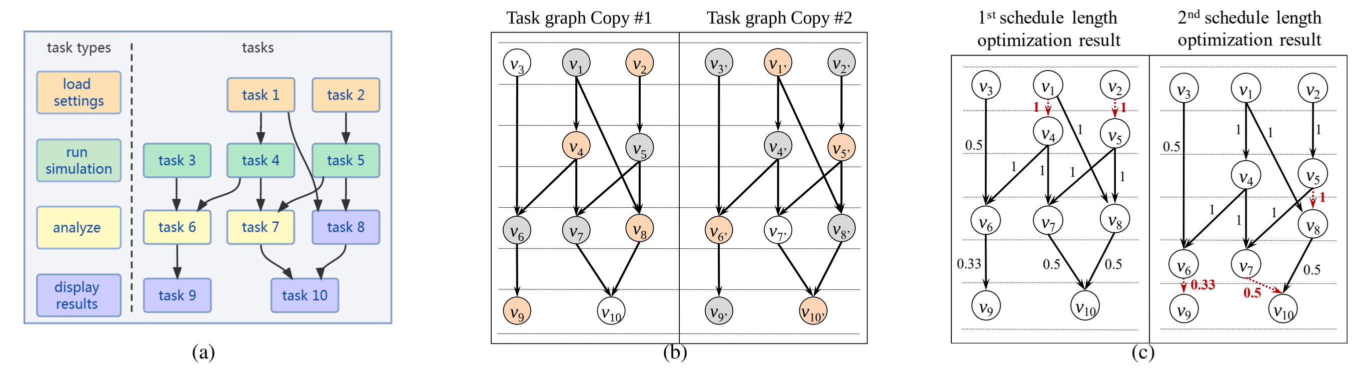


Fig. 2. Example of a task graph and its schedules. (a) Task graph. (b) ASAP schedule with security constraints. (c) Schedule length optimization results.

To mute undesired and potentially malicious communication paths and at the same time isolate an active Trojan from the rest of the system, data-dependent tasks must be computed by the cores fabricated from different IP vendors. This type of security constraint ensures that all the valid communications are between 3PIPs from different vendors.

3.3 Vulnerability Analysis

Analyzing a circuit's vulnerability against HT attacks is a key step toward trusted design, because sections of a circuit with low controllability and observability are considered potential areas for HT insertions [38], [39]. Adopted from the work presented in [25], the vulnerability analysis involves statement analysis, observability analysis and detectability analysis.

The *statement analysis* first measures the statement execution conditions of signals. Let $T_W(sig, l)$ be the *statement weight* of signal *sig* in line *l*, which is defined as $\frac{U-L+1}{U_O-L_O+1}$, where L and U are the lower and upper limits of the value range, respectively, and where U_O and L_O are the declared upper and lower limits of the controlling signals, respectively. Fig.1 shows the statement weights of signals X and Z with the sample codes given on the left column. For example, the range of X in line 8 is from 0 to 10, which means that $L = 0$, $U = 10$, $L_O = 0$ and $U_O = 15$. Thus, the statement weight of X in line 8 is $T_W(X, 8) = \frac{10-0+1}{15-0+1} = 0.6875$.

The *observability analysis* evaluates the *observability* of signals through the circuit's primary output, and it is computed by summing the statement weights of the signal that influences the target signal. The observability from signal *sig* to its target signal sig_t is denoted as $T_O(sig, sig_t)$, and the method of evaluating $T_O(sig, sig_t)$ is demonstrated via an example of calculating $T_O(X, Z)$. The X and Z signals both appear in lines 8 and 10, meaning that $T_O(X, Z) = 0.1875 + 0.3125 = 0.5$, where 0.1875 and 0.3125 are the statement weights of X in lines 8 and 10, respectively.

The *detectability analysis* defines the detectability of a signal based on its statement weight and observability. The detectability of signal *sig* in line *l* is defined as $T_D(sig, l) = T_W(sig, l) \times T_O(sig, sig_t)$, and an example of computing $T_D(X, 8)$ is given as follows. With $T_W(X, 8) = 0.1875$ and $T_O(X, Z) = 0.5$, the detectability is calculated as $T_D(X, 8) = 0.1875 \times 0.5 = 0.09375$. Similarly, the detectability of X in line 10 is $T_D(X, 10) = 0.3125 \times 0.5 = 0.15625$. A lower T_D indicates a higher vulnerability to Trojan insertion, and the signal X at line 8 has a higher vulnerability to Trojan attacks in this example. Therefore, the vulnerability of a signal (also referred to as *communication*) between tasks is set as $1/T_D$ in this study.

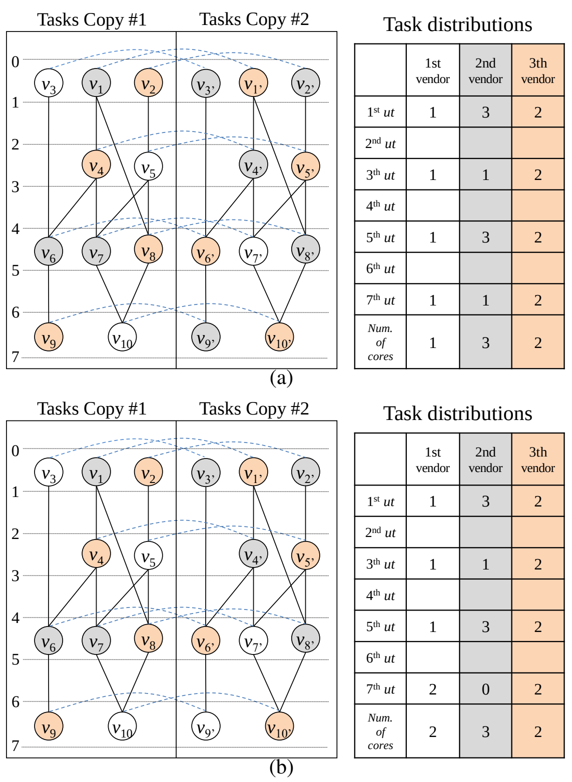
3.4 Motivations

Fulfilling security constraints may incur 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 required.

3.4.1 Vulnerability Increment in Performance Optimization

Security constraints in design process may cause significant performance overheads (e.g., approximately 50% performance overheads [19]), because all communications between tasks are inter-core communications. However, many security-critical applications also have high performance requirements, such as autonomous vehicles. To reduce the performance overheads, researchers have selectively removed isolation-with-diversity constraints from some communications, and these unprotected communications can be assigned as intra-core communications with much smaller delays [22]–[24]. But this increases the systems' vulnerabilities because these unprotected communications are vulnerable to HT attacks. In this study, we refer to the practice of clustering the data-dependent tasks into one core as **edge contraction**, and these tasks are connected by the **contracted edge**, which can be set as intra-core communication. All the inter-core communications are assigned with isolation-with-diversity constraints to maximize the security, while the intra-core communications cannot be protected by these constraints.

Traditional methods for system performance optimization either ignore the consequent security loss [22] or treat every communication with the same security importance [23], [24]. In fact, different communications have varying vulnerabilities to HT attacks, and these vulnerabilities can differ by up to 10^3 times within the same benchmark [25]. Therefore, optimizing the performance with the awareness of vulnerability variations helps to design the system with maximized security. The reasons are shown in Fig. 2(c), where the vulnerabilities of communications


Tasks Copy #1
Tasks Copy #2
Task distributions
0
1
2
3
4
5
6
7
v_3 v_1 v_2 $v_{3'}$ $v_{1'}$ $v_{2'}$
v_4 v_5 $v_{4'}$ $v_{5'}$
v_6 v_7 v_8 $v_{6'}$ $v_{7'}$ $v_{8'}$
v_9 v_{10} $v_{9'}$ $v_{10'}$
(a)
(b)

(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*	1	1	2
Num. of cores	1	3	2

(b)

	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

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.

are indicated next to the edges, and the performance optimization target is to reduce the schedule length in Fig. 2(b) by 1 *ut*. Traditional methods [23], [24] contract the fewest edges, which are $e_{1,4}$ and $e_{2,5}$ (see the 1st schedule length optimization result in Fig. 2(c)). However, $e_{5,8}$, $e_{6,9}$, and $e_{7,10}$ might be less vulnerable to HT attacks if compared to $e_{1,4}$ and $e_{2,5}$, and contracting these edges causes less vulnerability (see the 2nd schedule length optimization result in Fig. 2(c)), even though this contracts more edges.

3.4.2 Core Optimization in Vendor Assignment

In the vendor assignment stage, all tasks are partitioned into many groups such that all tasks in a group are executed by the cores from the same vendor. Traditional methods start to optimize the number of cores after the vendor assignment stage when the number of cores required can be evaluated [19], [21]–[23]. However, the vendor assignment results also determine the number of cores required, and the example in Fig. 3 explains the reason. In this example, the deadline constraint is assumed to be 7 *ut*, and all security constraints are satisfied, which are represented by the blue (*duplication-with-diversity*) and 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).

3.5 Problem Description

Although incorporating security constraints in the design process cannot guarantee full protection from all HT attacks, the vulnerability against HT attacks can be significantly reduced. Clustering data-dependent tasks to reduce the schedule length leaves the corresponding intra-core communications unprotected and makes these communications more vulnerable to HT attacks. In this study, the vulnerability of a communication is regarded as the reduced vulnerability after applying security constraints to this communication, and vulnerability analysis [25] can be performed before our method is used to first determine the vulnerabilities of communications. Let the application task graph be $TG = (V, E)$, where V and E are the sets of tasks and communications, respectively; the problem of this work can be described as follows.

Problem 1. The inputs of this problem are the application task graph TG, vendor constraints vc, tight deadline of this application t^d, core speeds of vendors, and vulnerability of each communication. The objective is to find a schedule with the lowest design vulnerability against HT attacks, and the number of cores required is also optimized.

The design vulnerability vul_s is regarded as the accumulated vulnerabilities of all unprotected communications, which can be calculated as follows:

$$vul_s = \sum_{e \in E_c} vul(e) \tag{1}$$

where E_c is the set of all unprotected communications, and $vul(e)$ is the vulnerability of e. The following constraints must also be satisfied.

(1) For any task, its finish time must not be earlier than its start time plus the execution time, which is:

$$t_i^f \geq t_i^s + exec(v_i), \quad \forall v_i \in V \tag{2}$$

where t_i^s, t_i^f and $exec(v_i)$ are the start time and finish time and execution time of v_i, respectively.

(2) For any task, its finish time must not exceed the tight deadline of the application, which is:

$$t_i^f \leq t^d, \quad \forall v_i \in V \tag{3}$$

(3) For any communication $e_{ij} = (v_i, v_j)$, the start time of v_j should not be earlier than the finish time of v_i plus the communication delay of e_{ij}, which is:

$$t_j^s \geq t_i^f + dly(e_{ij}), \quad \forall e_{ij} \in E \tag{4}$$

where $dly(e_{ij})$ is the communication delay of e_{ij}.

4 Security-Driven Task Scheduling Methods

In this section, a three-step task scheduling method is proposed, and both the design vulnerability and the number of cores are optimized under deadline constraints. The three steps of the proposed method are deadline-constrained task clustering, vendor assignment with core usage minimization, and task scheduling. Table 1 gives the descriptions of the notations and acronyms used in this paper.

4.1 Deadline-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 tight deadline constraints with the vulnerability of the circuit design optimized. 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

TABLE 1
Variables and acronyms used in this paper.

Notation	Description
Variables	
$vul(e)$	Vulnerability of edge e.
$dly(e)$	Communication delay of edge e; $dly_{inter}(e)$ and $dly_{intra}(e)$ are the inter-core and intra-core communication delays of e, respectively.
$dly_{rd}(e)$	Reduced communication delay of e after contracting e.
$w_{dly}(e)$	Total reduced delay after contracting edge e.
$w(e)$	Weight of e that evaluates both delay reduction and vulnerability increment.
$prob(v_i, T_j)$	The probability that v_i is executed in time T_j.
$DG(c, T_j)$	The probabilities of all tasks in cluster c in time T_j.
$DG_{max}(c)$	The maximum of all $DG(c, T_j)$, and it estimates the required number of cores for all tasks in cluster c.
$Merge(c_i, c_j)$	Number of cores reduced by merging clusters c_i and c_j.
Acronyms	
TG/TG'	Task graph and its duplicate;
TVG	Timing violated graph, consisting of all paths that violate the timing constraints;
$ECCG$	Edge contraction conflict graph, representing every pair of edges in TVG can be contracted simultaneously.
$MWIS$	Maximum weight independent set;
$VCFG$	Vendor conflict graph, representing whether two clusters must be assigned to different vendors;
$VCPG$	Vendor compatible graph, representing whether two clusters can be assigned to the same vendor;

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). Let $slack(v)$ be the slack time of v under the deadline constraint, which is calculated as follows:

$$slack(v) = T_{alap}(v) - T_{asap}(v) - exec(v) \tag{5}$$

where $T_{asap}(v)$ and $T_{alap}(v)$ are the ASAP and as-late-as-possible (ALAP) schedules, respectively.

The **timing violated graph** ($TVG = (V_T, E_T)$) is then constructed by all tasks with negative slacks, and it is an induced subgraph of TG. V_T consists of s, t and all tasks with negative slacks, and $E_T = \{(v_i, v_j) \in E, v_i \in V_T \text{ and } v_j \in V_T\}$. Fig. 4(b) shows an example of TVG, where the deadline constraint is 5 ut and the delay is 1 ut for each edge.

Edge contraction is then performed to optimize the schedule length, but some data-dependent tasks have to be executed by different types of IP cores, making the corresponding edge unable to be contracted. For the edge e_{ij} that can be contracted, $dly_{inter}(e_{ij})$ and $dly_{intra}(e_{ij})$ are its inter-core communication delay and intra-core communication delay, respectively. After contracting e_{ij}, the reduced communication delay of e_{ij} is $dly_{rd}(e_{ij})$, which equals $dly_{inter}(e_{ij}) - dly_{intra}(e_{ij})$, and the lengths of all paths that pass through e_{ij} are also reduced by $dly_{rd}(e_{ij})$. Let the sum of the reduced schedule lengths of all paths (from s to t) in TVG be $w_{dly}(e_{ij})$, and it is calculated as follows:

$$w_{dly}(e_{ij}) = \begin{cases} path_{tvg}(e_{ij}) * dly_{rd}(e_{ij}), & if \;\; v_i.type == v_j.type; \\ -1, & \text{otherwise}; \end{cases}$$

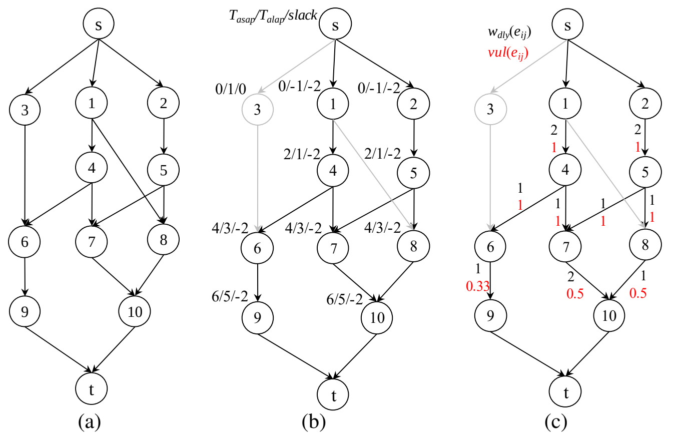


Fig. 4. Example of evaluating the timing violated graph. (a) Task graph with s and t. (b) TVG with a timing constraint of 5 ut. (c) The evaluation of $w_{dly}(e)$.

where $path_{tvg}(e_{ij})$ is the number of paths in TVG that pass through e_{ij}. If task v_i and v_j are of the same type, the weight of e_{ij} is $path_{tvg}(e_{ij}) * dly_{rd}(e_{ij})$; otherwise, $w_{dly}(e_{ij})$ is set to -1, meaning that such edge will not be selected for edge contraction.

Fig. 4(c) illustrates the $w_{dly}(e_{ij})$ and $vul(e_{ij})$ of all edges in TVG, which are indicated next to the edges, and all tasks are assumed to be the same type. The target in the schedule length optimization stage is to contract the edges with larger schedule length reduction $w_{dly}(e_{ij})$ and smaller vulnerability increment $vul(e_{ij})$. Therefore, the total weight that evaluates an edge e_{ij} contraction, denoted as $w(e_{ij})$, can be calculated as follows:

$$w(e_{ij}) = \frac{w_{dly}(e_{ij})}{vul(e_{ij})} \tag{6}$$

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 vulnerability against HT attacks. Suppose that contracting either $e_{1,4}$ or $e_{4,7}$ in Fig. 4(b) will make the path length smaller than the deadline constraint, and contracting $e_{1,4}$ and $e_{4,7}$ at the same time causes additional vulnerability.

Then, **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:

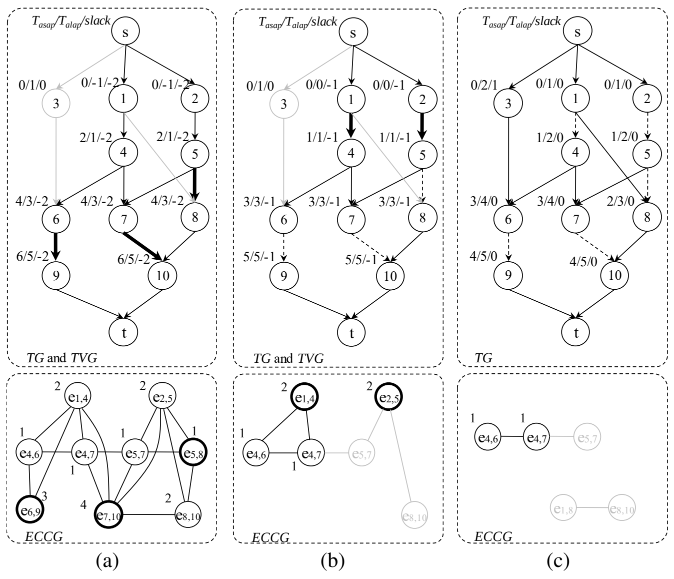


Fig. 5. Example of deadline-constrained task clustering procedure. (a) *TVG* and its corresponding *ECCG* before task clustering. (b) *TVG* and its corresponding *ECCG* after 1st iteration of task clustering. (c) *TVG* and corresponding *VCG* after 2nd iteration of task clustering.

Algorithm 1 Deadline-constrained task clustering.

Input: application task graph, TG;
deadline of the application, t^d.
Output: deadline-constrained clustering results.

```
    /* Step 1: Construct TVG and calculate edge weights. */
 1: while $TG.schedule\_length > t^d$ do
 2:     Construct $TVG$ from $TG$.
 3:     for each $e$ in $TVG$ do
 4:         Calculate $w(e)$;
 5:     end for
    /* Step 2: Construct ECCG and find the MWIS. */
 6:     Construct $ECCG$ from $TVG$;
 7:     for Each node $e$ in $ECCG$ do
 8:         $ECCG.node\_weight(e) = w(e)$;
 9:     end for
10:     Calculate $MWIS$ in $ECCG$;
    /* Step 3: Contract edges based on the MWIS. */
11:     for each node $e$ in $MWIS$ do
12:         Contract the corresponding edge $e$ in $TG$;
13:     end for
14: end while
```

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 the path length).

The maximum weight independent set (MWIS) of *ECCG* is calculated by the method proposed in [40], and the target is to find a set of edges with maximum weight that can be contracted simultaneously. Algorithm 1 shows the deadline-constrained task clustering algorithm with the goal of minimizing the design vulnerability. 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 steps are iteratively repeated until the deadline constraint is met.

With the vulnerabilities of communications given in Fig. 4(c), an example of deadline-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 *ECCG* are first 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 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 deadline constraint satisfied.

4.2 Vendor Assignment with Core Usage Minimization

For each type of 3PIP cores, the principle of vendor assignment is to iteratively cluster tasks into a number of v_c^t (vendor constraint for the IP cores with type t) clusters, and assign each cluster with an IP vendor according to its core speed. Different from task clustering in the deadline-constrained task clustering stage that violates isolation-with-diversity, clustering (also named as **cluster merging**) in vendor assignment follows security constraints.

The **vendor conflict graph** of type t ($VCFG^t = (V_c^t, E_{cf}^t)$) is constructed from the deadline-constrained clustering results, and it represents whether two clusters must be assigned to different vendors. V_c^t is the set of all clusters from TG and TG' with type t. 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}^t is the edge set in $VCFG^t$, and if two tasks are connected by the inter-core communication under the protection of security constraints, the two clusters that contain these two tasks will be connected in $VCFG^t$. The **vendor compatible graph** ($VCPG^t = (V_c^t, E_{cp}^t)$) is the complement graph of $VCFG^t$, and an edge in E_{cp}^t indicates that the connected clusters can be assigned to the same vendor.

For different types of tasks, their corresponding vendor assignments are performed independently, and we assume that all tasks are with the same type in the following discussion for simplicity. The examples of vendor conflict graph and vendor compatible graph, denoted as *VCFG* and *VCPG*, are presented in Fig. 6(a), and they are constructed from the deadline-constrained clustering results shown in Fig. 5(c). Because the edges in *VCPG* are too many to demonstrate, we use dashed lines to represent the remaining edges that are connected to this cluster.

The main challenge in optimizing the number of cores in the vendor assignment stage is that the vendors of tasks have not yet been determined, and therefore, the accurate number of cores from each vendor can hardly be evaluated. Inspired by the probabilistic approach in [41], we also assume that the probabilities of a task on all its possible scheduling results are the same, and employ a probability-based method to analyze the number of cores required. Let $prob(v_i, T_j)$ be the probability that v_i is executed in time T_j, and the accumulated probability of task concurrency is calculated and denoted as the *distribution graph* (DG). The summation of the probabilities of all tasks in a cluster c for the time period T_j is

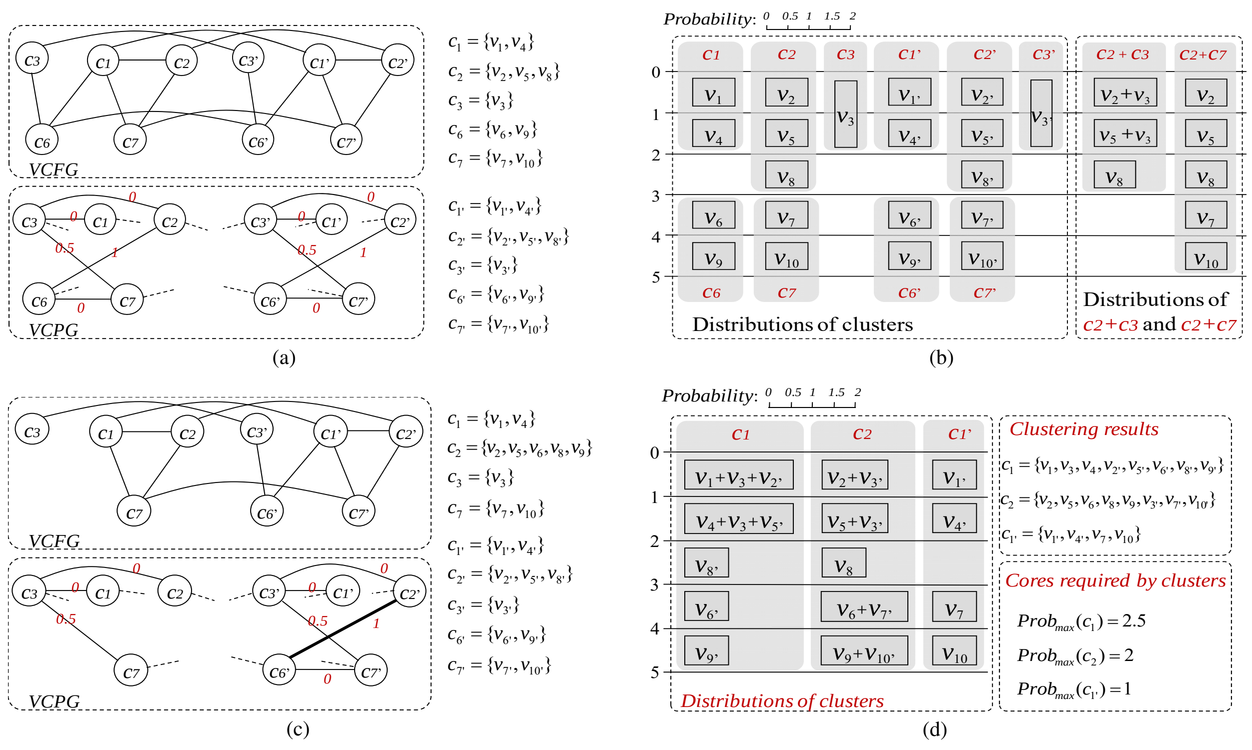

VCFG
VCPG
$c_1 = \{v_1, v_4\}$
$c_2 = \{v_2, v_5, v_8\}$
$c_3 = \{v_3\}$
$c_6 = \{v_6, v_9\}$
$c_7 = \{v_7, v_{10}\}$
Probability:
Distributions of clusters
Distributions of $c2+c3$ and $c2+c7$
Clustering results
$c_1 = \{v_1, v_3, v_4, v_{2'}, v_{5'}, v_{6'}, v_{8'}, v_{9'}\}$
$c_2 = \{v_2, v_5, v_6, v_8, v_9, v_{3'}, v_{7'}, v_{10'}\}$
$c_{1'} = \{v_{1'}, v_{4'}, v_7, v_{10}\}$
Cores required by clusters
$Prob_{max}(c_1) = 2.5$
$Prob_{max}(c_2) = 2$
$Prob_{max}(c_{1'}) = 1$
(a)
(b)
(c)
(d)

Fig. 6. Example of vendor assignment. (a) *VCFG* and *VCPG* derived from task clustering results. (b) Distributions of clusters. (c) *VCFG* and *VCPG* in 2nd iteration of cluster merging. (d) Cluster merging results under vendor constraint.

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

$$DG(c, T_j) = \sum_{v_i \in c} prob(v_i, T_j) \tag{7}$$

The maximum of all $DG(c, T_j)$, $\forall T_j \in [1, t_d]$ is denoted as $DG_{max}(c)$, which estimates the required number of cores 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 computed 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:

$$Merge(c_i, c_j) = DG_{max}(c_i) + DG_{max}(c_j) - DG_{max}(c_i + c_j) \tag{8}$$

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 may 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 used to determine the clusters to be merged due to its large time complexity.

Fig. 6 shows an example of cluster merging procedure, 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 are also updated. Fig. 6(c) shows *VCFG* and *VCPG* after the 1st iteration of cluster merging. This procedure terminates when the number of clusters equals the vendor constraint, and Fig. 6(d) gives the final cluster merging results, where the total estimated number of cores is 5.5.

Then, the number of timing-critical tasks in each cluster is counted, 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 deadline-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 vulnerability $vul(e)$ 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 $dly_{rd}(e)$, 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 which consists of vendor assignment and vendor adjustment stages. In the first step (*Lines 2-3*), the potential core usage reduction obtained from clustering tasks is evaluated, and the corresponding weighted *VCPG* is constructed. In the second step (*Lines 4-7*), for each type t, tasks are merged into vc clusters with consideration of core usage minimization. In the third step (*Lines 8-10*), each cluster is assigned to IP vendors according to the core speed. In the fourth step (*Lines 12-21*), unprotected edges

Algorithm 2 Vendor assignment with core usage minimization.

```
Input: deadline-constrained clustering results.
       deadline and vendor constraints, $t^d$, $vc$.
Output: vendor assignment.
1:  for Each type of cores, whose vendor constraint is $vc^t$ do
    /* Step 1: Weight VCPG edges by the potential core reduction.*/
2:      Calculate $DG_{max}(c)$ for each cluster $c$ with type $t$.
3:      Construct $VCFG^t$ and weighted $VCPG^t$;
    /* Step 2: Cluster merging for core usage minimization. */
4:      while $VCPG^t.node\_num > vc^t$ do
5:          Find the edge $e_{max} = (c_i, c_j)$ with the maximum weight
            in $VCPG^t$, and merge $c_i$ and $c_j$ into one cluster.
6:          Update $VCFG^t$ and weighted $VCPG^t$,
7:      end while
    /* Step 3: Assign clusters to vendors. */
8:      while Not all clusters are assigned with IP vendors do
9:          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.
10:     end while
11: end for
    /* Step 4: Adjust vendor assignment considering core speed. */
12: Update $DGs$ with the determined task execution times;
13: $E_{cv}$ is the set consists of all intra-core communications;
14: while $E_{cv} \neq \emptyset$ do
15:     Find $e \in E_{cv}$ with the largest $vul(e)$;
16:     if Setting $e$ as inter-core communication still meets $t^d$ then
17:         Assign either $source(e)$ or $target(e)$ with another vendor;
18:         Update $DGs$;
19:     end if
20:     Remove $e$ from $E_{cv}$;
21: end while
```

are examined in descending order of vulnerability, and vendor assignments are adjusted under deadline constraints to further reduce the overall design vulnerability, accounting for variations in core speed across vendors after initial assignment.

4.3 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, deadline-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 [22] 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.

Fig. 7 gives the whole procedure of our proposed security-driven task scheduling method. First, task clustering method (refer to Algorithm 1) iteratively contracts edges to meet the deadlines with a minimized vulnerability increment; Then, vendor assignment method (refer to Algorithm 2) assigns tasks to IP vendors with core usage minimization, and the vendor assignment is further adjusted considering core speed variation. Finally, tasks from the same IP vendor are scheduled together using the force-directed scheduling method [41], because it schedules tasks evenly across each time period, requiring only a small number of cores.

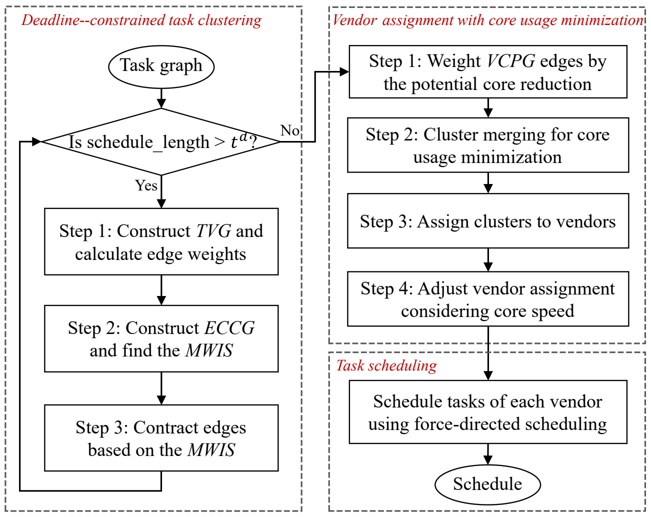

Deadline--constrained task clustering
Task graph
Is schedule_length > t^d?
No
Yes
Step 1: Construct TVG and calculate edge weights
Step 2: Construct $ECCG$ and find the $MWIS$
Step 3: Contract edges based on the $MWIS$
Vendor assignment with core usage minimization
Step 1: Weight $VCPG$ edges by the potential core reduction
Step 2: Cluster merging for core usage minimization
Step 3: Assign clusters to vendors
Step 4: Adjust vendor assignment considering core speed
Task scheduling
Schedule tasks of each vendor using force-directed scheduling
Schedule

Fig. 7. The overall flowchart of the proposed security-driven task scheduling.

Our proposed methods can also be easily adopted in the following scenario, where 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 [23] is conducted before vendor assignment, so that the vendor constraint can be satisfied with a minimized number of contracted edges.

4.4 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 deadline-constrained task clustering stage, constructing $ECCG$ from TVG requires $O(m^2)$, and finding the MWIS in $ECCG$ also requires $O(m^2)$ [40]. Only a constant number of iterations are conducted before reaching the deadline constraint, and finding all contracted edges 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 deadline-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 $VCPG$. Vendor 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.

5 Experimental Results

5.1 Experimental Setup

All the experiments were implemented in C on a Linux Workstation with an E5 2.6-GHz CPU and 32-GB of RAM.

TABLE 2
Deadline-Constrained Task Scheduling Results.

task graph	nodes	CCR	SL (ut)	t^d (ut)	vul_s GT-B	MIP-B	MC-B	PSO-B	Our	Number of cores GT-B	MIP-B	MC-B	PSO-B	Our
robot	88	0.5	839	671	58.7	54.7	42.6	44.2	31.4	12	11	11	11	11
		1.0	1114	892	61.5	58.2	44.3	41.9	30.3	12	10	11	10	10
sparse	96	0.5	179	143	44.8	41.7	42.5	35.8	28.4	18	16	17	16	16
		1.0	236	189	49.5	43.8	43.8	37.2	29.3	18	16	16	16	15
fpppp	336	0.5	1590	1272	34.6	34.6	3.27	3.05	2.37	12	10	11	10	10
		1.0	2119	1695	38.5	36.8	35.2	31.4	25.1	11	10	10	10	10
rnc500	500	0.5	280	224	77.8	70.5	68.2	62.5	44.6	65	60	62	58	58
		1.0	373	300	83.5	75.9	71.9	64.8	49.5	63	58	60	58	56
rnc1000	1000	0.5	190	152	105.4	94.2	92.5	88.3	74.1	88	82	84	79	78
		1.0	254	203	99.5	96.8	94.3	95.8	78.9	81	77	77	76	74
rnc2000	2000	0.5	199	159	74.3	55.3	61.8	44.7	33.7	184	172	175	170	167
		1.0	268	214	6.81	52.9	54.7	41.2	36.2	180	168	170	168	164
rnc3000	3000	0.5	1336	1069	132.7	115.2	100.4	85.3	72.8	67	64	64	64	62
		1.0	1779	1423	115.4	104.9	103.9	80.5	67.6	62	56	58	56	56
rnc5000	5000	0.5	850	680	146.7	135.4	126.9	122.1	103.7	132	124	128	125	122
		1.0	1146	917	152.5	139.1	124.3	127.9	112.9	125	118	122	118	115
avg.		0.5			84.4	75.2	71.0	64.2	51.6	72.3	67.4	69.0	66.6	65.5
		1.0			83.6	76.1	71.6	65.1	53.7	69.0	64.1	65.5	65.3	62.5

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). The numbers of nodes in the real application benchmarks range from 88 to 334, and the randomly generated task graphs are much larger, with the numbers of nodes ranging from 500 to 5000. Considering that the maximum clique sizes of most task graphs modeled from real application programs are no larger than 4 [22], the maximum clique sizes of the randomly generated task graphs are 3 or 4. The vulnerabilities of communications are analyzed via the method proposed in [25], which consists of statement analysis, observability analysis and detectability analysis. Because the benchmarks do not provide the HDL source codes for statement analysis, we assume that the statement weights of all signals are one in these experiments. To simplify the experiments, all intra-core communication delays were ignored, and we set the step of the core speed differences equal to 5% of the fastest core speed.

Our proposed method was then compared with four other methods to demonstrate its effectiveness. The first method is the "*graph theoretic-based approach*" (**GT-B** for short) [19], which uses graph-theoretic techniques to detect the security problem with security constraints, and the recovery phase is ignored in our experiments because it incurs significant area and delay overheads to the design. The second method is the "*mixed integer programming-based approach*" (**MIP-B** for short) [21], which jointly considers the energy consumption and security constraints in the MPSoC design, using a mixed integer programming model with the objective of minimizing the energy consumption. The third method is the "*min-cut-based approach*" (**MC-B** for short) [23], which boosts performance by iteratively contracting the edges selected by the max-flow min-cut algorithm, and schedules tasks with the force-directed scheduling-based method. The fourth method is the "*particle swarm optimization (PSO)-based approach*" (**PSO-B** for short) [18], which uses a PSO-based method to explore the design space, and find a resource-optimized solution with the security constraints.

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

5.2 Security-Driven Task Scheduling Results

The security-driven task scheduling results are shown in Table 2. Column $nodes$ give the number of tasks in each task graph. 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 deadline is set to $t^d = 0.8SL$, where SL is the ASAP schedule length with all security constraints satisfied. The IP vendor constraint is set to the maximum clique size of the benchmark.

The results in Table 2 show that our proposed method obtains the lowest vul_s for all benchmarks. When the CCR is set to 0.5, our method reduces the vulnerabilities by 32.8, 23.6, 19.4 and 12.6, respectively, compared to those of GT-B, MIP-B, MC-B and PSO-B. When the CCR becomes 1.0, the vulnerabilities saved by our method are 29.9, 22.4, 17.9 and 11.4, respectively, compared to those of GT-B, MIP-B, MC-B and PSO-B. The reasons that our proposed method outperforms the other methods are as follows. GT-B developed a graph-based method to assign tasks to cores following vendor diversity, but all communications are treated equally during the scheduling. MIP-B established a set of formulations representing the data dependencies between tasks to minimize the runtime energy, and the vulnerabilities are not its first optimization target. MC-B optimized the design vulnerability by reducing the number of unprotected communications, but ignored the vulnerability variation of communications, and it might also choose brother edges to contract, resulting in a larger vul_s. PSO-B explores the design space to find the near-optimal solution, and the quality of its output also depends on the iteration of the algorithm.

Furthermore, our proposed method obtains the fewest cores among these methods. The goal of GT-B is to detect HT attacks, and optimizing the number of cores is not considered. MIP-B minimizes the runtime energy of MPSoC, and reducing the number of cores is not the key design target. MC-B does not optimize the number of cores in the vendor assignment stage because the vendors have not yet been determined, and estimating the number of cores during vendor assignment might not be accurate. PSO-B outputs schedules with low vul_s and small numbers of cores, but its CPU runtimes are much larger. Unlike MC-B, our method uses a probability-based method to evaluate the number of reduced cores during cluster merging, which reduces the computational cost and compensates for the errors caused by the probability-based method.

The CPU runtimes of these methods are compared in Table 3. For the benchmarks modeled from real applications, all of the methods can produce solutions within several minutes. The time complexity of our method is $O(m^2)$, meaning that the computational time is primarily decided by the scale of m (the number of edges). In addition, the actual CPU runtime is also affected by the number of iterations executed in Algorithms 1 and 2 to satisfy the deadline and vendor constraints. For the benchmarks that contain many nodes and edges, such as *rnc5000* which has 5000 nodes and 55432 edges, our proposed method can output a solution within approximately 10 minutes. This finding indicates that our proposed method is applicable for most benchmarks in real practice.

5.3 Design Vulnerability *vs.* Schedule Length

Then, the effectiveness of our method in optimizing the design vulnerability is tested, and the CCR is set to 1.0 for all the benchmarks. Three deadline constraints are tested in the experiments, which are $0.95SL$, $0.9SL$ and $0.85SL$, and the corresponding design vulnerabilities are presented in the Figs. 8(a), 8(b), and 8(c), respectively.

For each benchmark, our proposed method outperforms GT-B, MIP-B, MC-B and PSO-B with different deadlines. The main reason is that our proposed method minimizes the vulnerabilities in both the schedule length optimization and vendor assignment stages. When the deadline is set to $0.95SL$, the average vulnerability of our proposed method is 5.5, and the vulnerabilities of GT-B, MIP-B, MC-B and PSO-B are 9.4, 9.1, 8.4 and 7.2, respectively. Our method also obtains better results when the deadlines are set to $0.9SL$ and $0.85SL$, and the corresponding vulnerabilities are 15.7 and 32.7, respectively. Furthermore, our method shows more advantages than the other methods with shorter deadlines. The vulnerabilities saved by our method are 3.9, 3.6, 2.9 and 1.7 with the deadline $t^d = 0.95SL$, compared to MC-B, GT-B and PSO-B, respectively; when the deadline becomes $0.85SL$, the vulnerabilities saved by our method are 13.8, 13.2, 9.6 and 6.8, compared to MC-B, GT-B and PSO-B, respectively.

Furthermore, we evaluate the robustness of the designs under HT attacks to demonstrate the necessity of optimizing the

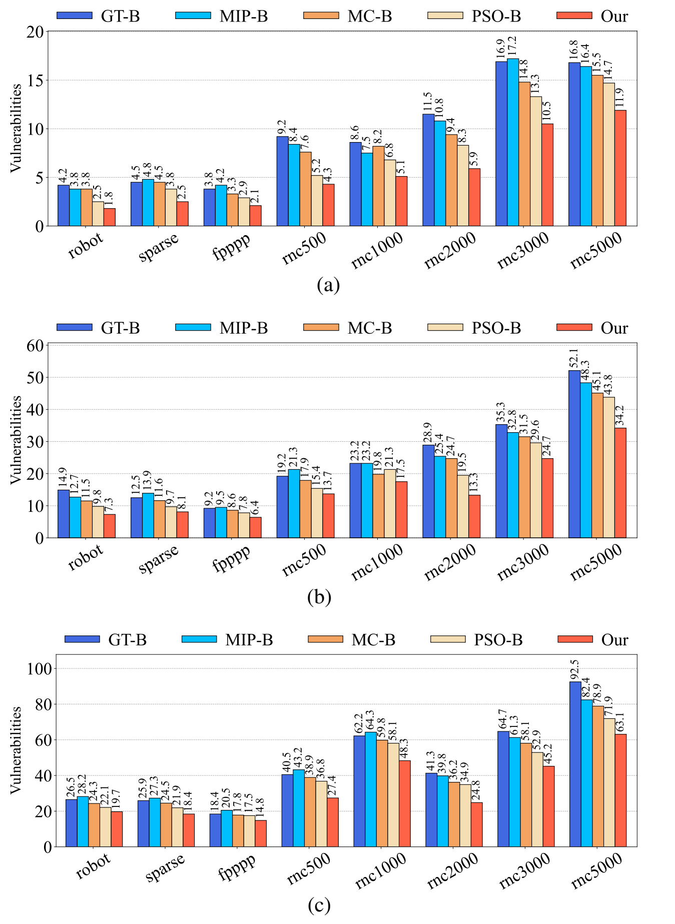


Fig. 8. Comparison of design vulnerabilities under different deadlines. (a) Design vulnerability with t^d=0.95SL. (b) Design vulnerability with t^d=0.9SL. (c) Design vulnerability with t^d=0.85SL.

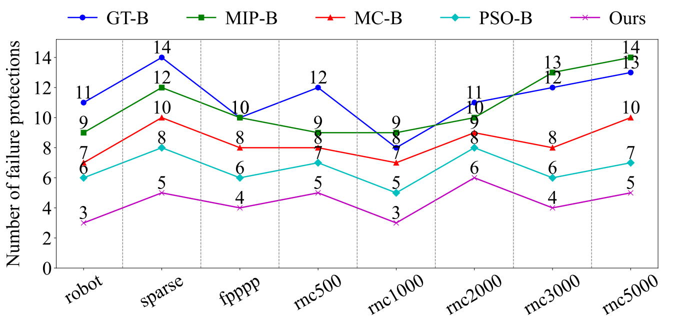


Fig. 9. Number of failure protections under 1000 attacks.

TABLE 3
Comparisons of CPU Runtime.

task graph	nodes	edges	average CPU runtime (s)				
			GT-B	MIP-B	MC-B	PSO-B	Our
robot	88	131	15.4	13.5	17.5	221.6	18.7
sparse	96	67	27.3	21.8	41.2	285.6	33.8
fpppp	334	1145	41.9	232.9	50.2	369.2	47.6
rnc500	500	1910	72.8	926.3	113.6	823.5	95.4
rnc1000	1000	3005	176.2	2842.7	259.3	3271.3	183.5
rnc2000	2000	3930	351.9	3959.8	715.7	8661.3	553.6
rnc3000	3000	39034	1227.4	8127.3	3582.6	33425.6	3014.9
rnc5000	5000	55432	3626.1	15412.6	9004.5	91423.9	6764.2

vulnerabilities in task scheduling. For each benchmark, we assume that the HTs only choose one communication to attack, and the attack possibility for each communication depends on the vulnerability of the communication. The possibility to attack the communication e is set to $vul(e)/Total\_vul$, where $Total\_vul$ is the accumulated vulnerabilities of all edges in the benchmark. If the attacked communication is protected with security constraints, we consider it as a successful protection; otherwise, it is a failure protection. The deadline is set to $0.85SL$, and 1000 such attacks are conducted on each benchmark. The numbers of failure protections are presented in Fig. 9, which shows that our method results in fewer failure protections than the other methods do. The average number of failure protections of our method is only 4.38, whereas the numbers of failure protections of GT-B, MIP-B, MC-B and PSO-B are 11.38, 10.75, 8.38 and 6.63, respectively.

5.4 Comparisons of Cores Required

Finally, the numbers of cores needed by the different methods are compared. The deadline of the application is set to *SL* so that the deadline-constrained task clustering stage can be skipped, and this eliminates the impacts of deadline-constrained task clustering results on the core usages. Tables 4 and 5 show the numbers of cores needed, where both loose and tight vendor constraints tested.

TABLE 4
Numbers of Cores Required with the Loose Vendor Constraint.

task graph	*SL* (*ut*)	*CLQ*	Loose vendor constraints (*vc* = *CLQ*)				
			GT-B	MIP-B	MC-B	PSO-B	Our
robot	1114	3	9	9	9	8	8
sparse	236	3	12	12	12	12	12
fpppp	2119	3	9	8	9	8	8
rnc500	373	3	45	42	45	40	38
rnc1000	254	3	65	61	63	58	56
rnc2000	268	3	145	138	148	132	130
rnc3000	1779	4	53	50	51	48	48
rnc5000	1146	4	96	93	95	90	88
avg.			54.3	51.6	54.0	49.5	48.5

The loose vendor constraints are first set for all benchmarks, where the vendor constraint is set to be the maximum clique size of the corresponding TG. Table 4 shows the results, and the column CLQ gives the maximum clique size of each TG. The results indicate that our proposed method needs the fewest number of cores among these methods, and the average numbers of cores required by GT-B, MIP-B, MC-B, PSO-B and our method are 54.3, 51.6, 54.0, 49.5 and 48.5, respectively. GT-B ignores core optimization, MIP-B treats the chip area as the secondary optimization target, MC-B minimizes the number of cores only in the task scheduling stage, and PSO-B explores the design space with a limited number of iterations. In our method, reducing the number of cores is considered in both vendor assignment and task scheduling, which enlarges the optimization space for saving the number of cores.

TABLE 5
Numbers of Cores Required with the Tight Vendor Constraint.

task graph	*SL* (*ut*)	*CLQ*	Tight vendor constraints (*vc* = 2)				
			GT-B	MIP-B	MC-B	PSO-B	Our
robot	1114	3	8	7	8	7	7
sparse	236	3	11	10	11	10	10
fpppp	2119	3	7	7	7	6	6
rnc500	373	3	38	38	37	36	35
rnc1000	254	3	51	48	50	48	46
rnc2000	268	3	127	122	124	118	116
rnc3000	1779	4	50	46	48	45	45
rnc5000	1146	4	91	87	88	84	82
avg.			47.9	45.6	46.6	44.3	43.4

The tight vendor constraints are also tested 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 [23] 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. This introduces additional vulnerabilities to the designs, although fewer cores are needed under tight vendor constraints. Our method also obtains the fewest cores among these compared methods, and the average numbers of cores needed by TG-B, MIP-B, MC-B, PSO-B and our method are 47.9, 45.6, 46.6, 44.3 and 43.4, respectively.

6 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. The communications between data-dependent tasks are treated with different vulnerabilities against HT attacks, and a maximum weight independent set-based task clustering method is proposed to meet tight deadline while maintaining a high security level. In addition, the numbers of cores required are optimized in both the 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] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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

[6] 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.

[7] M.T. Rahman, Q. Shi, S. Tajik, H. Shen, D.L. Woodard, M. Tehranipoor and N. Asadizanjani, "Physical inspection & attacks: new frontier in hardware security," *Proc. International Verification and Security Workshop*, pp. 93-102, 2018.

[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] 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.

[10] 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.

[11] 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.

[12] B.J. Mohd, S. Abed, T. Hayajneh, and M.H. Alshayeji, "Run-time monitoring and validation using reverse funciton (RMVRF) for hardware Trojans detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2689-2704, Nov. 2021.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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, Oct. 2020.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] H. Salmani and M. Tehranipoor, "Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level," *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pp. 190-195, 2013.

[26] 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.

[27] 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.

[28] N. Hu, M. Ye, and S. Wei, "Surviving information leakage hardware Trojan attacks using hardware isolation," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 2, pp. 253-261, Apr. 2019.

[29] 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.

[30] 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.

[31] S. Yu, C. Gu, W. Liu, and M. O'Neill "Deep learning-based hardware Trojan detection with block-based netlist information extraction," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 4, pp. 1837-1853, Oct. 2022.

[32] 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.

[33] 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.

[34] R. S. Chakraborty, S. Pagliarini, J. Mathew, S. R. Rajendran, and M. N. Devi, "A flexible online checking technique to enhance hardware Trojan horse detectability by reliability analysis," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 2, pp. 260-270, Apr. 2017.

[35] N. Pundir, S. Aftabjahani, R. Cammarota, M.Tehranipoor, and F. Farahmandi, "Analyzing security vulnerabilities induced by high-level synthesis," *ACM Journal of Emerging Technologies in Computing Systems*, vol. 18, no. 3, pp. 47-68, 2022.

[36] 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.

[37] 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.

[38] J. Cruz, P. Slpsk, P. Gaikwad, and S. Bhunia, "TVF: a metric for quantifying vulnerability against hardware Trojan attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 7, pp. 969-979, Jul. 2023.

[39] Y. Dou, C. Gu, C. Wang, W. Liu, and F. Lombardi, "Security and approximation: vulnerabilities in approximation-aware testing," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 1, pp. 265-271, Jan. 2023.

[40] 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.

[41] 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.

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

Lijun Lu received the B.E. degree in information engineering from East China University of Science and Technology, Shanghai, China, in 2024. He is currently working toward the M.S. degree in electronic information from East China University of Science and Technology, Shanghai, China. His current research interests include hardware Trojan detection and hardware security.

PLACE PHOTO HERE

Songping Liu received the B.E. degree in information engineering from East China University of Science and Technology, Shanghai, China, in 2021. He is currently working toward the M.S. degree in electronic information from East China University of Science and Technology, Shanghai, China. His current research interests include hardware Trojan detection and hardware security.

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, purple and cyan are the modified parts.

Reviewer 1

Comment 1

The authors may consider adding a table summarizing all mathematical symbols to improve the readability of the paper.

Responses

We appreciate the reviewer's valuable suggestion. To improve the readability of the paper, we have added a table summarizing key variables and acronyms at the beginning of Section 4 as Table 1.

Reviewer 2

Comment 1

My only comment is that I would clearly state that this work extends some previous work from the same authors. This was already done in the Motivation section, but in third-person. I would clearly state this before the contribution in the Introduction, so that it is apparent the work extends a conference one, which is my understanding.

Responses

We appreciate the reviewer's valuable suggestion. We agree that the connection to our prior work should be stated more clearly. Accordingly, we have added a statement in the Introduction to clarify that this manuscript is an extended version of our previous conference paper.

This modification can be found on the right-hand column of page 1. The added sentence, highlighted in cyan in the revised manuscript, is as follows:

This study extends our prior work [24] which considered only performance constraints in security-aware task scheduling, by co-optimizing design vulnerability against HTs and the number of required cores.

Reviewer 3

Comment 1

One issue in the previous round of review is the difficulty in keeping in mind all the acronyms. Although the authors removed some unnecessary acronyms, the reviewer still thinks that it is essential to provide list of acronyms/abbreviations for ease of understanding.

Responses

We thank the reviewer for this constructive feedback. We acknowledge that keeping track of acronyms can be challenging. To resolve this, we have created a dedicated table that lists key variables and acronyms at the beginning of Section 4 as Table 1.

Comment 2

In literature review, the authors may need to include a discussion and analysis on scheduling approaches specifically for MPSoCs incorporating untrusted 3PIP cores.

Responses

We thank the reviewer for this valuable suggestion. To better contextualize our work, we have significantly restructured and expanded the Section 2 to include a focused discussion on scheduling approaches for MPSoCs with untrusted 3PIP cores.

Specifically, we have added two new subsections to create a more logical flow:

- A new Section 2.2 Modular Redundancy for Trojan-tolerant Design, has been added (highlighted in purple). This section introduces the fundamental hardware-level strategies (e.g., replication, comparison, and voting) that form the basis for security-driven scheduling, providing essential background.
- A new Section 2.3 Security-Driven Task schedules for MPSoC, directly addresses the reviewer's comment. This section is dedicated to analyzing prior art in our specific domain. It reviews works that:
  - Integrate security constraints (like duplication and vendor diversity) directly into the task scheduling flow for systems using untrusted IPs.
  - Propose optimization techniques to mitigate the significant performance, area, and energy overheads caused by these security-driven scheduling approaches.

Comment 3

Referring to Eq. (3), does the performance constraint in this work literally represent the deadline for task finish time? If so, why not directly using deadline constraints?

Responses

We thank the reviewer for this insightful comment. The reviewer is correct; in our work, the term ”performance constraint” was used to represent a deadline for task finish time.

To improve clarity and adopt more standard terminology, we have revised the manuscript to use “deadline constraint” consistently throughout the paper. This change has been applied globally. Key examples include:

- The paper’s title and abstract, which now refer to “Deadline Constraints”.
- The problem formulation in Section 3.5, where the input is now described as a “tight deadline of this application t^d”.
- All related algorithm and section titles have also been updated, such as the title of Section 4.1, “Deadline-Constrained Task Clustering”.

Comment 4

The term “core minimization” is to some extent awkward. Apparently, it means to achieve the lowest number of required cores. It is more appropriate to use “core usage minimization”.

Responses

We thank the reviewer for this excellent suggestion to improve the clarity of our terminology. We agree that ”core usage minimization” is a more appropriate term. We have revised the manuscript accordingly, replacing all instances of “core minimization” with “core usage minimization”.

Key instances of this revision include:

- The title of Section 4.2, which now reads:
  “Vendor Assignment with Core Usage Minimization”
- The overview of the methodology in the first paragraph of Section 4, which now states:
  “...deadline-constrained task clustering, vendor assignment with core usage minimization, and task scheduling.”
- The caption for Algorithm 2, which has been updated to:
  “Vendor assignment with core usage minimization.”

Comment 5

This paper heavily relies on pseudocodes to explain the key steps of the methodology, The authors can include flowcharts to help clarify the algorithmic flow.

Responses

We appreciate this constructive feedback. To make our methodology easier to follow, we have made several improvements by adding both a high-level flowchart and more detailed in-text descriptions.

1) To provide a clearer, high-level overview, we have added a new flowchart as Figure 7. To guide the reader to this figure, we also inserted a new descriptive sentence at the beginning of Section 4.3, highlighted in blue:
   “Fig. 7 gives the whole procedure of our proposed security-driven task scheduling method. First, task clustering method (refer to Algorithm 1) iteratively contracts edges to meet the deadlines with a minimized vulnerability increment; Then, vendor assignment method (refer to Algorithm 2) assigns tasks to IP vendors with core usage minimization, and the vendor assignment is further adjusted considering core speed variation. Finally, tasks from the same IP vendor are scheduled together using the force-directed scheduling method [41], because it schedules tasks evenly across each time period, requiring only a small number of cores.”
2) To enhance the clarity of the pseudocode itself, we have added descriptive step-by-step comments directly into Algorithms 1 and 2. For example, in Algorithm 1, we added:
   / Step 1: Construct TVG and calculate edge weights. */*
   ...
   / Step 2: Construct ECCG and find the MWIS. */*
   Furthermore, we added summary paragraphs in the main text to walk through the logic of each algorithm. For instance, the following blue-highlighted paragraph was added in Section 4.2 to explain the flow of Algorithm 2:
   “Algorithm 2 describes the proposed vendor assignment algorithm which consists of vendor assignment and vendor adjustment stages. In the first step... In the second step...”

Comment 6

The authors claim a low complexity of $O(m^2)$, in which m is the number of nodes. How comes that the scheduling approach takes exceptionally long execution times (regarding the CPU runtime statistics in Table 2)?

Responses

We thank the reviewer for this important question regarding the relationship between the theoretical complexity and the practical runtime. We apologize if our complexity description caused any confusion. The complexity of our method is $O(m^2)$, where m is the number of edges.The execution times reported in Table 3, especially for the largest benchmarks, are attributable to two main factors, which we have now clarified in the text:

The time complexity of our method is $O(m^2)$, meaning that the computational time is primarily decided by the scale of m (the number of edges). In addition, the actual CPU runtime is also affected by the number of iterations executed in Algorithms 1 and 2 to satisfy the deadline and vendor constraints.

The exceptionally long execution times for the largest benchmarks (e.g., rnc5000 with 5000 nodes and 55,432 edges) reported in Table 3 are primarily attributable to the following reasons:

1) **Benchmark Scale:** The benchmark *rnc5000* is exceptionally large (5,000 tasks, 55,432 edges). While the $O(m^2)$ complexity correctly predicts that processing time will increase quadratically with the number of edges, the absolute time for a graph of this size is

naturally substantial. The reported time of approximately 10 minutes is, in our view, still practically acceptable for a design-time scheduling algorithm applied to a complex MPSoC design, which is typically an offline process.

2) **Iterative Nature of the Algorithm:** Our method is iterative. Algorithms 1 and 2 repeat until deadline and vendor constraints are met. While the number of iterations is typically a small constant, this constant can be meaningful for large graphs with tight constraints. Each iteration involves the costly $O(m^2)$ step of building and solving the MWIS on the ECCG. For a large graph requiring several iterations, the total time is effectively a multiple of $O(m^2)$.