VAST: Validation of VP-based Heterogeneous Systems against Availability Security Properties using Static Information Flow Tracking

Ece Nur Demirhan Coşkun* Muhammad Hassan*ⓤ Mehran Goliⓤ Rolf Drechsler*ⓤ
*Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
ⓤInstitute of Computer Science, University of Bremen, 28359 Bremen, Germany
{ece.coskun, muhammad.hassan}@dfki.de {drechsle, mehran}@uni-bremen.de

***Abstract*—Ubiquitousness of modern feature-rich heterogeneous systems has significantly increased their security requirements. One weak point of entry might spread catastrophically over large areas, blocking the accessibility of different *Intellectual Properties* (IPs), and thereby disabling the system's functionality. Hence, it becomes vital to consider the trust and security implications during the design phase of these heterogeneous systems and identify possible security breaches due to the system design itself. Recently, various security validation methods have been successfully employed very early in the design phase at the system level using *Virtual Prototypes* (VPs). These methods have facilitated the investigation of digital systems with a focus on data leakage and untrusted access. However, modern systems are heterogeneous with heavy reliance on sensor inputs. Hence, similar security validation methods should also be considered from the *analog/mixed-signal* (AMS) perspective using SystemC AMS, to ensure availability security properties.**

In this paper, we propose VAST, a novel validation tool for VP-based heterogeneous systems against availability security properties. VAST employs static *Information Flow Tracking* (IFT) at the system-level to ensure the availability, i.e. timely accessibility, of IPs. In this regard, VAST analyzes analog-to-digital, digital-to-analog, as well as digital-to-digital behaviors of the underlying heterogeneous system. We demonstrate the applicability and scalability of the proposed tool on two real-world VPs with different sizes of complexity, a car anti-trap window system, and a thermal house system.

I. Introduction

Heterogeneous embedded systems are ubiquitous in *Internet of Things* (IOT) devices. Combined with *Software* (SW), digital *Hardware* (HW) employing microcontrollers and microprocessors, and *Analog/Mixed-Signal* (AMS) *Intellectual Property* (IP), these heterogeneous systems provide feature-rich functionality. Simultaneously, IOT devices are storing an increasing amount of private and sensitive data in addition to carrying out security-critical tasks. As a result, the security requirements of such devices have increased significantly in the last decade. Since IOT devices are heterogeneous, security validation for such systems should not only focus on SW or digital HW, but also on AMS IPs (e.g., physical interfaces, sensors, and actuators) [1]–[3] in a comprehensive and inclusive manner, i.e., a holistic view rather than focusing on individual IPs.

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project AUTOASSERT under contract no. 16ME0117

Traditionally, security validation was always considered an afterthought, however, recent security breaches, such as a backdoor via JTAG [4], emphasize the importance of considering security implications from the start. Also, since a *System-on-Chip* (SoC) cannot be patched once it has been manufactured, fixing hardware security vulnerabilities, after deployment becomes quite costly. As a result, it is vital for the industry to integrate security considerations early into the SoC design process [5], [6].

Therefore, *Completeness-Driven Development* (CDD) [7] is often used in which the design process is divided into different abstraction levels, consequently, shifting focus towards verification. The high-level idea is to use *Virtual Prototypes* (VPs) at the abstraction of *Electronic System Level* (ESL) as the starting point for early design and verification process, and it progresses to the next abstraction level only after achieving completeness, i.e verifying the complete behavior of the design at each level of abstraction. With this aim in mind, *Virtual Prototypes* (VPs), which are abstract SW models of the HW implemented in SystemC [8] with its *Transaction Level Modeling* (TLM) [9] and AMS extensions [10], are currently heavily used as a golden reference for early SW and HW development [11]–[14].

Recent advances in security validation methods in the digital domain have been successfully employed very early in the design phase at the system-level using VPs [5], [15]–[18]. These methods leverage *Information Flow Tracking* (IFT) as the underlying technology and mainly focus on digital HW from different aspects, e.g., confidentiality, integrity, and timing channel. By using IFT, the flow of information across a system can be understood. IFT techniques target security flaws in circuit design, verification, testing, manufacturing, and deployment. They can identify malicious circuit modifications, timing side channels, access control violations, unintended design faults, and other insecure hardware behaviors. IFT tags data objects to represent security classes, which have varying meanings based on the type of *security property* (SP) being analyzed. It observes the status of tags to verify information flow properties and updates tags when the data is computed. With the help of information tracking, it is possible to enforce rules for secure information flow, including confidentiality, integrity, isolation, constant time, and availability.

Availability describes the requirement for data, services, or

TABLE I
A COMPARISON OF VAST WITH STATE-OF-THE-ART IFT TECHNIQUES AT THE SYSTEM LEVEL

Source	Method	Target	Security Property
[5]	Static IFT	Data leakage, Untrusted access	Confidentiality, Integrity
[16]	Static IFT	Timing Flow	Constant Time
[18]	Dynamic IFT	Run-Time SW Vulnerabilities	Confidentiality
[21]	Dynamic IFT	Data leakage,Untrusted access	Confidentiality, Integrity
[22]	Dynamic IFT	Data leakage, DIFT for accelerators	Confidentiality, Integrity
[23]	Symbolic Execution	Verify firmware security properties	Confidentiality, Integrity
VAST	Static IFT	Availability of IPs for AMS Systems	Availability

other assets to be readily available and usable when requested by authorized entities [19]. One weak point of entry into IOT devices might spread over large areas if the availability of security-critical signals is dependent on the values of exposed signals. These compromised signals can cause the system dysfunctional and possibly block the accessibility of IPs, if the availability of critical signals is not guaranteed. Accordingly, there have been numerous attacks, such as *Denial of Service* (DoS) and flooding attacks, which endanger availability and cause hardware and software failure [19]. Thereby, to assure the availability of information in the IOT context, proper detection and protection algorithms must be established.

Although some initial works towards security validation methods and tools at system-level VPs using IFT have been done, as discussed in [20], there is a lack of information flow research in three directions:

1) Developing IFT techniques for heterogeneous embedded systems, in particular SystemC AMS VPs.
2) Specifying security properties for availability problems.
3) Verifying them at the system level.

In this paper, we introduce *VAST*: a novel VP-based IFT tool against availability security properties for heterogeneous systems. *VAST* employs a method that combines multiple passes of static analysis and operates directly on the SystemC/AMS VP models. At the heart of *VAST* is a scalable static IFT analysis, which not only checks information flow through digital-to-digital interactions, but also analyzes analog-to-digital, and digital-to-analog interactions. *VAST* leverages the flexible LLVM/Clang compiler infrastructure to gather useful information from the VP, e.g., data flow, call-graph, *Data Dependency Graph* (DDG), and *Control Flow Graph* (CFG). As a result, *VAST* can capture static paths based on availability security properties. These captured paths are reported for further analysis. *VAST* provides a sound analysis, i.e., it over approximates but never misses a violating path if it exists. We demonstrate the applicability and scalability of VAST on two real-world VPs, a car anti-trap window system and a thermal house system.

II. RELATED WORK

One of the powerful security validation methodologies, IFT, is particularly useful in detecting unintentional design flaws, malicious circuit modifications, timing side channels, access control violations, and other insecure hardware behaviors [20]. Depending on the assumptions regarding modifiability, IFT can have different variations. Several IFT methods and tools have been successfully introduced as seen in Table I. The appropriate technique among these depends on the requirements and needs of the target system. For instance, if we cannot change or examine the source code, we are confined to binary or dynamic analysis [18], [21], [22], which is performed on an executing program on real or virtual hardware and can be done without access to the source code. On the other hand, if we have access to the target system's source code, we can undertake static analysis [5], [16], which can be performed without executing the program and requires access to the source code and/or object code. Symbolic execution [23] is another reasonable and convincing solution for SystemC AMS VP models, but it is a challenging approach that still needs further research.

Regarding the application of IFT to AMS systems, there has been a first attempt at the transistor level in [24]. The authors demonstrated that the method can detect sensitive data leakage from the analog to the digital domain and vice versa. Their work expands on the previous PCHIP-based IFT techniques [25] and establishes information flow policies for numerous analog components, including MOSFETs, bipolar transistors, capacitors, inductors, resistors, and diodes. They employ an automated framework called VeriCoq-IFT to convert the netlist of the analog/RF circuitry into a Verilog representation. Another similar work [26] detects charge-domain Trojans at the circuit level. They employ a technique that puts forth an abstracted model of contaminated information, flowing to user-controllable or accessible flip-flops from charge-domain leakage structures.

Although, as seen in Table I, information flow properties such as confidentiality, integrity, isolation, timing channels, and Hardware Trojans have been specified in various powerful hardware IFT tools and methods, in this paper, we have focused on the validation of heterogeneous systems using SystemC AMS VPs against availability SPs.

III. VAST OVERVIEW

This section provides a high-level overview of the architecture of VAST. First, we describe the threat model under consideration. Then, we discuss a motivating example to highlight the security problem VAST solves. At the end, we provide an overview of the complete workflow.

A. Threat Model

Threat modeling is a structured method of identifying and prioritizing potential threats to a system. One security hole is all it takes for an attacker to take control of an entire system. As a result, it's crucial to follow a set of methodologies when it comes to threat modeling so that all known/unknown threats and vulnerabilities can be handled. Considering a heterogeneous system, we are particularly interested in protecting

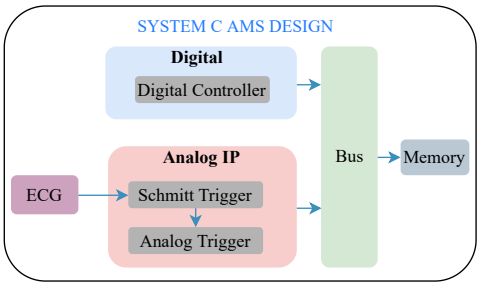

Fig. 1. The SystemC AMS design of the motivating example

assets that rely on AMS interactions, e.g., sensors. These interactions happen at analog-to-digital as well as digital-to-analog interfaces in conjunction with transporting the sensitive data via buses between the different IP components. We consider a threat model based on the *Confidentiality, Integrity, Availability, Authentication* (CIAA) principles [20], in particular **availability**. Availability problems arise when an IP uses some shared resources to the point that they are unavailable to other IPs.

B. Motivating Example

We provide here an example that is used to demonstrate the fundamental aspects of our tool throughout this paper. The motivating example represents a simplified AMS design consisting of an *Electrocardiography* (ECG) sensor, an *Analog IP* with *Schmitt Trigger* and *Analog Trigger* circuit, *Digital Controller*, *Bus*, and *Memory* as shown in Fig. 1. The ECG sensor is connected to the *Schmitt Trigger* circuit, which uses hysteresis behavior as described in [27] to eliminate minor fluctuations in the signal. Then, it is passed to the *Analog Trigger* circuit, which determines whether the signal should be stored in the *Memory*.

In principle, this system is designed to detect a person's level of activity using the ECG sensor. When ECG signals exceed a predefined threshold, the *Analog IP* must detect irregularity and send a request via *Bus* to *Memory* to save this information. The *Digital Controller* performs read/write operations in the *Memory* as a normal routine. The intuition here is that the *Digital Controller* and *Analog IP* should have equal opportunity to access the *Memory*, i.e., no resource should be allowed to block access to *Memory*. This is ensured by access policies commonly implemented in components with routing functions, in our case the *Bus*.

Now consider a scenario where a designer implements the access control policy in *Bus* as shown in Fig. 2. The access policy uses a priority encoder to give the highest priority to *Analog IP*[1]. The benefit of such a policy is to detect abnormal levels of heartbeats for precautionary or emergency purposes. In *Line 4* (Fig. 2), the first priority is determined: If the output signal of *Analog IP* has a voltage of 1.75 V (*cap == 1.75*), the signal *grant_analog* is set and data is transferred from *Analog IP* to *Memory*. The second priority is shown in *Line 8*.

[1]It is a common practice to use such policies in interrupt controllers.

```
1  ...
2  grant_analog = 0;
3  grant_digital = 0;
4  if (cap == 1.75) { // 1.75 Volts
5    grant_analog = 1;
6    write_mem.write(1);
7  }
8  else if (request_digital == "1") {
9    grant_digital = 1;
10   write_mem.write(1);
11 }
12 else
13   write_mem.write(0);
14 if (grant_analog == "1")
15   data_bus_out.write(data_analog_in);
16 else if (grant_digital == "1")
17   data_bus_out.write(data_digital_in);
18 ...
```

Fig. 2. Code excerpt from Bus IP implementing priority encoded access policy

When *request_digital* signal for the *Digital Controller* is high, *grant_digital* is set to allow access to *Memory*.

Because of the higher priority for *Analog IP*, the *Digital Controller* and the *Bus* have an indirect information flow. An adversary can exploit the *Analog IP* to block the *Digital Controller*, rendering the *Memory* inaccessible. Such indirect information flow across another IP is difficult to detect, particularly without an automated analysis, such as used by VAST.

C. VAST Overall Workflow

Fig. 3 depicts the overall workflow of VAST to perform IFT analysis for heterogeneous systems. As VAST leverages scalable static analysis, it only needs to be executed once to validate the security properties. Essentially, VAST reads the security properties and identifies the critical signals and their security attributes. It leverages data flow analysis to perform static taint analysis, and finally IFT to validate the availability security properties. VAST performs the IFT in two phases: 1) Information extraction, 2) Information flow analysis.

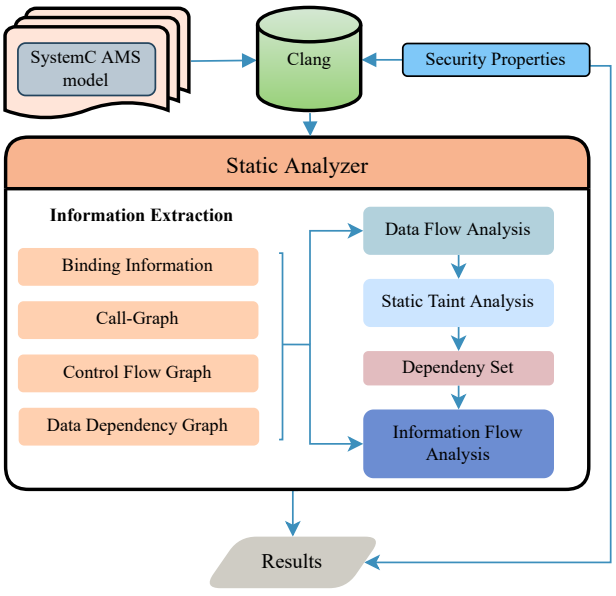

Fig. 3. VAST Tool Overview

During the information extraction phase, VAST obtains IP connectivity information (binding information) to identify how data flows through the heterogeneous VP. It is followed by the construction of call-graphs which are used in IFT at the end. Afterwards, the CFG is obtained for each IP to get more information about access control policies in particular and the control flow of the VP in general. At the end, the DDG is constructed to extract the data path including all modules' signals, ports, and global and local variables.

During phase two, information flow analysis is performed. It starts with over-approximated data flow analysis to identify the set of all possible data flow values computed at different points in a system. Afterwards, static taint analysis between critical signals (as defined in security properties) is performed and the dependency sets based on them are obtained. In the end, all the information is interleaved to identify if there exists any direct or indirect information flow between critical signals which can potentially make an IP unavailable for use. Essentially, the result shows which availability security properties have been satisfied and which are not satisfied at the end. A property is satisfied if both of the two following conditions are satisfied:

1) There does not exist direct information flow between critical signals.
2) There does not exist indirect information flow between critical signals.

In the following section, we detail the essential blocks of VAST as well as demonstrate them using the motivating example.

IV. Static Information Flow Analysis and Illustration

A. Information Extraction

Information flows can be captured by detecting updates caused by conditional statements (i.e. if-else) induced by critical signals and sensitive data. This can be handled by detecting and tracking interactions of all the variables until output. In order to detect these interactions, the identification of internal variables is necessary that have the possibility of accepting new values, as well as the ones that remain at the current value. As a result, the first step is to extract information from the given VP.

1) Security Property Specification: As mentioned earlier, we focus on the availability SP. It is defined such that various IPs are required to be available in a timely manner [28]. Each SP has inputs with the *High Security* (HS) tag and outputs that must be *Always Available* (AA) when needed. Thus, we define SPs as follows:

$$SP = \{(SI, SO) | SI \leftarrow \{.. = HS\}, SO \leftarrow \{.. = AA\}\} \quad (1)$$

For example, the SP of the motivating example can be defined as follows:

$$SP = (\{\texttt{st\_in} = HS\}, \{\texttt{grant\_digital} = AA\}) \quad (2)$$

The SP in Eq. (2) ensures that the signal `grant_digital` sent by the *Digital Controller* to the *Bus* module must not be dependent on the primary input `st_in` of the *Schmitt Trigger* module (Fig. 4).

```
1  ...
2  z_hysteresis = ltfz(num, den, s, dz_hysteresis);
3  st_module_out.write(z_hysteresis+ vertical_shift);
4  if (z_hysteresis >= 0) {
5    if (st_in + a - horizontal_shift >= 0)
6      dz_hysteresis = alpha * (b - z_hysteresis);
7    else if (st_in + a - horizontal_shift < 0)
8      dz_hysteresis = alpha * (-b - z_hysteresis);
9  }...
```

Fig. 4. Code excerpt from Schmitt Trigger implementing hysteresis behavior

2) Binding Information: Next, we extract the *Binding Information* (BI). If BI is not available, module connectivity cannot be determined statically before execution. This assists in determining how data flows through the system and also in the construction of call-graphs.

3) Call-Graph: VAST constructs the call-graph once at the beginning. Furthermore, the behavior of commonly occurring system function calls is already described within the analysis, e.g., *memcpy()*. The call-graph is used to coordinate the analysis so that the information is propagated to the correct function inside the AMS VP. As a result, it makes use of the BI to appropriately identify the function calls.

4) Control Flow Graph: VAST leverages the CFG and the *Abstract Syntax Tree* (AST) of the VP in order to know how different statements (data and control flow) of a given design are related to each other. For example, a part of the CFG of *Schmitt Trigger* is shown in Fig. 5a, according to its code excerpt (Fig. 4). In Fig. 5a, the nodes with claret red borders refer to the conditional statements *(L4, L5, L7)*. Similarly, a part of the CFG of *Bus* is shown in Fig. 5b, where conditional statement nodes *(L4, L8, L14, L16)* are circled with dark green.

5) Data Dependency Graph: The DDG explains how the design's variables (such as all modules' signals, ports, and

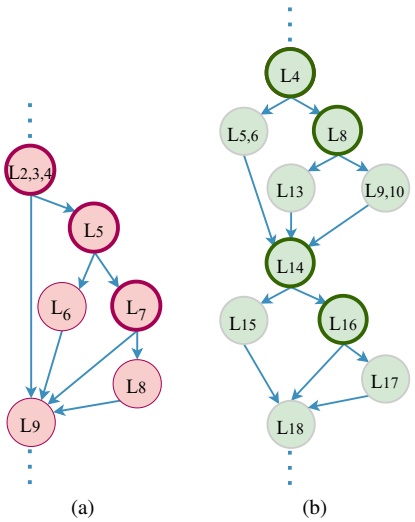

Fig. 5. A part of the CFG of **(a)** the Schmitt Trigger design and **(b)** the Bus design

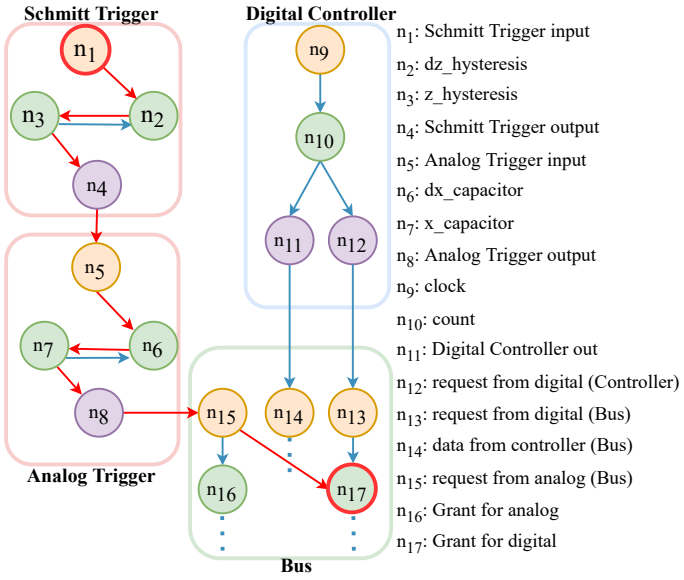

Fig. 6. A part of extracted DDG of the model

variables) relate to one another [5]. The DDG can be formally defined as follows:

Definition 1: A Data Dependency Graph (DDG) is a structure (N, E, Z), where N is a set of nodes, E is a set of edges, and $Z \subseteq N$ is a set of output variables. The edge from node A to node B shows that B is dependent to A.

Based on the definition, a part of the DDG of the SystemC AMS model is shown in Fig. 6. Orange-colored nodes represent input ports of modules, green-colored nodes are internal signals, and purple-colored nodes are output ports of modules. Please note that binding information is abstracted away.

B. Static Analysis

1) Data Flow Analysis: A data-flow analysis algorithm takes as input the VP to compute test objectives (i.e., definition-use (*def-use*) pairs). A *reaching uses* procedure - an instance of data flow analysis techniques - is used to identify these test objectives for a VP, which actually answers such a question: for each variable defined, which uses can potentially use the values? Our data flow analysis is inspired by [5] but we do not use the associations as defined. Rather, we only define *def-use* pairs according to their classification to help in our analysis. VAST defines a *def-use* pair as an ordered triple (x, d, u) such that d is a statement where variable x is defined and u is a statement where x is used. Furthermore, there is a static path from d to u in the program without a re-definition of x in-between. The analysis identifies all possible *def-use* pairs of a VP by performing intra-function analysis. Inter-function analysis is deliberately not performed as it will be compensated for during the taint analysis.

2) Static Taint Analysis: Static taint analysis is used to generate the *Dependency Set*. It starts with a tainted source and adds variables based on dependence data from the CFG, such as *def-use* pairs and *use-to-dependence* pairs. Similar to *def-use* pairs, *use-to-dependence* pairs represent dependence for

Algorithm 1: Information Flow Analyzer

Require: Security property SP, DDG, CFG
Ensure: Information Flows Violation IFV, $IFV_{suspicious}$
1: Secure Inputs $SI \leftarrow$ Extract from (SP)
2: Secure Outputs $SO \leftarrow$ Extract from (SP)
3: Sensitive List Secure Inputs $SL_{SI} \leftarrow SI$
4: Sensitive List Secure Outputs $SL_{SO} \leftarrow \emptyset$
5: **for all** secure input $S_i \in SI$ **do**
6: $SL_{SI} \leftarrow$ ForwardTraverse (DDG, Si, SO)
7: **for all** secure output $S_o \in SO$ **do**
8: $SL_{SO} \leftarrow$ BackwardTraverse (DDG, Si, SO)
9: $n_{\text{conds}} \leftarrow \{n \,|(n \in CFG)\, [n.type() = cond]\}$
10: **for all** $n_{\text{cond}} \in n_{\text{conds}}$ **do**
11: $n_{ctrl} \leftarrow$ Extract list of controllers from n_{cond}
12: **if** $n_{ctrl} \cap SL_{SI} \neq \emptyset$ **then**
13: $n_{children} \leftarrow$ Extract child elements of n_{cond}
14: **for all** child $c \in n_{children}$ **do**
15: **if** $(c.type() \neq cond)$ **and** (vars in $c \in SO$) **then**
16: $IFV \leftarrow (n_{\text{cond}}, c)$
17: **else if** $(c.type() \neq cond)$ **and** (vars in c $\in SL_{SO}$) **then**
18: $IFV_{suspicious} \leftarrow (n_{\text{cond}}, c)$
19: **else**
20: **for all** child $c \in n_{children}$ **do**
21: **if** $(c.type() \neq cond)$ **and** (vars in $c \in SO$) **then**
22: $FC_{flag} \leftarrow 1$
23: **if** $(IFV = \emptyset)$ **and** $(FCflag = 0)$ **then**
24: $IFV \leftarrow IFV_{suspicios}$
25: **return** IFV , $IFV_{suspicios}$

variables in the conditional statement of CFG blocks, where definitions in the possible successors to the conditional statement are stored as dependent to the variables in the conditional statement. In Fig. 6, we aim to show clearly the critical path from the *Schmitt Trigger* input to the *grant_digital* with red arrows, which consist of nodes in the *Dependency Set*.

3) Information Flow Analysis: Algorithm 1 is proposed to detect all potential information flows in a given AMS VP. The motivating example is used to illustrate each part of the algorithm. The algorithm takes as inputs DDG, CFG, and a set of SPs, and returns a list of nodes in the CFG.

To determine whether a variable is affected by secure inputs (HS tag), a DDG uses forward tracing from the corresponding secure input node to an output node (AA tag). The HS tag is assigned to all nodes in this trace that are related to the secure input and are added to the sensitive list of secure inputs SL_{SI} (*Lines 5–6*). Furthermore, because the output variables may receive their final values via the intermediate variables, a backward tracing on the DDG is also performed to extract the variables of assignment statements that are implicitly or explicitly related to the outputs with the AA tag. These nodes (along with the corresponding design variables) are added to the sensitive list of secure outputs SL_{SO} (*Lines 7–8*).

With regards to the motivating example, the list of secure inputs is $SL_{SI} = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_{15}\}$ and the secure outputs is $SL_{SO} = \{n_{17}, n_{15}, n_8, n_7, n_6, n_5, n_4, n_3, n_2\}$.

The CFG of VP is then analyzed (*Lines 9-22*) to find all sensitive control signals (which are in SL_{SI}) that influence the occurrence of updates on variables with AA tags (which are in SO and SL_{SO}). Each CFG condition node type (e.g., if-else) is visited, and its control variables n_{ctrl} are retrieved (*Lines 9–11*). If the intersection of the condition node's extracted control

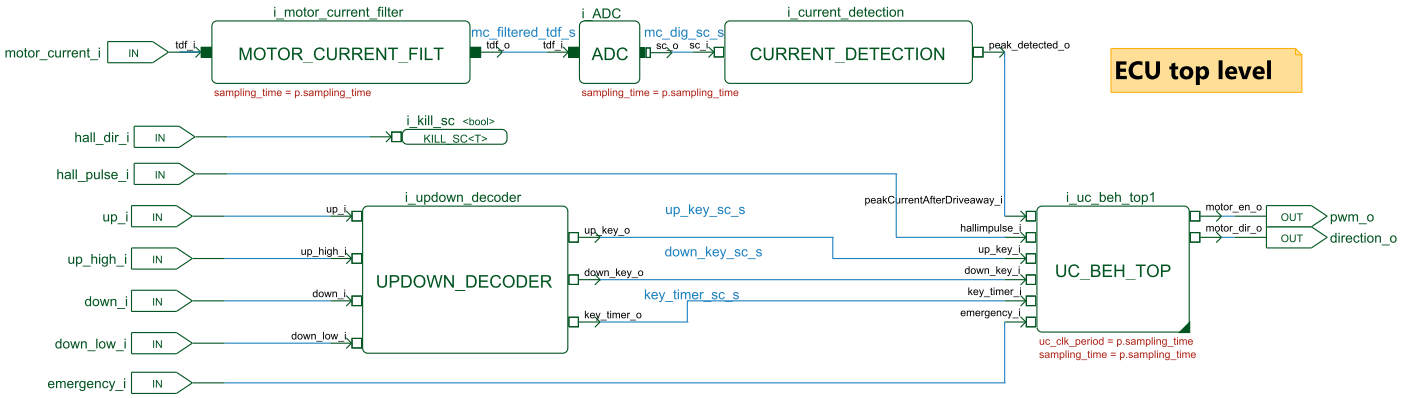

Fig. 7. The block diagram of the Electrical Control Unit subsystem of the Car Anti-Trap Window System

variables n_{ctrl} and the sensitive list of secure inputs SL_{SI} is not empty, additional analysis is performed on the condition node's child nodes (which are not condition node types) $n_{children}$ (*Lines 12–18*). The purpose of this analysis is to identify assignment statements whose left-hand side variables are in the secure outputs list SO (in the case of explicit flow) or the sensitive list of secure outputs SL_{SO} (case of implicit flow). A flow occurs in the design if there is at least one child node that matches the first criteria (*Line 15*). As a result, the nodes in the CFG that include the condition and assignment (as well as the matching LoC in the design source code) are stored in IFV and reported to designers. If, on the other hand, the second criterion (*Line 17*) occurs, there may be a flow in the design. To determine whether the suspicious situations are actual leakage flows, we must identify a circumstance in which its control signals are not in SL_{SI} and it fully controls the updates on variables in SO (*Lines 19–22*). Fully controlling control signal means that variables in SO only receive a new value if and when the controllers do. The suspicious case becomes an actual leakage flow scenario if there isn't such a condition in the design and no direct leakage flow is present; in this case, it is kept in IFV (*Lines 23–24*).

In the motivating example, we found an explicit flow by using Algorithm 1. As seen in the source code of the *Bus* design, given in Fig. 2, *Line 4* (L4) is a conditional type statement. Its only control variable in the n_{ctrl} list, "`cap`", is in the list SL_{SI}. We, therefore, look at its children, which are L5, L6, L8, L9, L10, and L13, as seen in Fig. 5b. Since the variable "`grant_digital`" (in L9) is in the list SO, there exist an explicit flow. The runtime of this analysis took 6.58 s.

V. Experimental Results

In this section, we demonstrate how various security properties with respect to availability are specified and verified using VAST, based on the concept of IFT. We consider two complex real-world heterogeneous systems from our industrial partner implemented as SystemC AMS designs: 1) car anti-trap window system (26604 LoC), 2) thermal house system (7381 LoC). All the experiments were carried out on a PC equipped with 24 GB RAM and an Intel Core i7-8565U CPU running at 1.8 GHz.

```
1   ...
2   clamping_protection_current_state =
        iop->clamping_protection_next_state;
3   switch (clamping_protection_current_state) {
4    case clamping_protection__STOP_state: {
5      ...
6     if (transition_fired == 0) {
7      if ((downKey_i && !upKey_i)) {
8       iop->clamping_protection_next_state =
            clamping_protection__Move_state;
9   ...
10   case clamping_protection__BACK_state: {
11     ...
12    if (transition_fired != 0) {
13     switch (transition_fired) {
14       ...
15      case 198: {
16        obstacle_detected_o = false;
17  ...
```

Fig. 8. Code excerpt from the Car Anti-Tap Window System

A. Car Anti-Trap Window System

The Anti-Trap Window System dramatically increases car safety by reversing the window when sensing a human presence to prevent individuals from getting trapped, hurt, or killed. In this experiment, we concentrate on an AMS system that regulates the window's movement (up and down) while preserving the safety of the passengers. The system includes the *Electrical Control Unit* (ECU) given in Fig. 7 and a complete window environment, which includes the motor, mechanical elements such as the window, and control buttons. The ECU model consists of a motor current filter to reduce noise from current measurements, an ADC for motor current conversion, a current detector for over-current detection, a decoder for the raw signals from the control buttons (*UPDOWN_DECODER*), and a microcontroller (*UC_BEH_TOP*). In the microcontroller of this modern window system, an algorithm that is implemented as a state machine, manages the movement of the window. As the window moves, the current flowing through the lifter motor is continually monitored. When an obstacle such as a passenger's finger is detected, the current flow changes, notifying the controller to halt and prevent hazards. An excerpt from this algorithm is given in Fig. 8.

Now consider a scenario where a security breach results in an attacker being able to take control of the system. Regarding

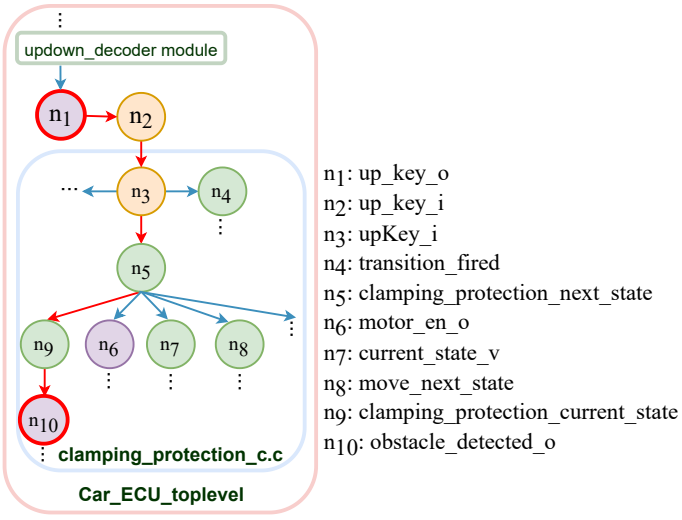

Fig. 9. A part of extracted DDG of the Car Anti-Trap Window System

availability, the detection of the obstacle must be independent from unrelated signals such as the up/down control buttons that change the movement of the car window. The signals that are used for the detection of the obstacle, such as the *obstacle_detected_o* signal from Line 16, Fig. 8, must be available in the ECU regardless of the values of any unrelated signal, such as the *up_key_o* signal sent by the *UPDOWN_DECODER* module. Otherwise, an attacker can alter the *up_key_o* signal in a way that the controller does not stop the window movement, endangering the passenger. To assess the security of the design against this attacker, we have defined five availability SPs. They check whether the flow of the *obstacle_detected_o* signal to the ECU is dependent on the other signals. Our tool finds that one of these SPs is unsatisfied, which is given in Eq. 3.

$$SP = (\{up\_key\_o = HS\}, \{obstacle\_detected\_o = AA\}) \tag{3}$$

This means that the signal *up_key_o*, which is supposed to be isolated from *obstacle_detected_o*, may affect the detection of the obstacle. The DDG created by our static analyzer had 996 nodes, and 482 nodes were determined to be dependent to *up_key_o*. An excerpt of this DDG is shown in Fig. 9, where the critical path from *up_key_o* to *obstacle_detected_o* is highlighted with red arrows. The connection between n_1 and n_2 is seen in Fig. 7, whereas the portion of the DDG from n_3 until n_{10} is seen in Fig. 8. The variable *clamping_protection_current_state*, which VAST found to be dependent to *up_key_o*, was a controlling variable of a switching block that affects *obstacle_detected_o*, and caused the SP to fail. The run time for this example was 167.2 s.

B. Thermal House System

The Thermal House System given in Fig. 10 computes the energy loss over different surfaces (window, wall, roof, etc.), considering the ambient temperature over time of a typical day and night cycle. The inside temperature is monitored by a thermostat controlling the heater. Essentially, the inside temperature of the house is compared with the set temperature and their difference is sent to the *Heat Control Mechanism* (HCM). The HCM controls the heater's activation, by checking whether the difference is positive or negative. To assess the security of the design, we have defined three availability SPs. For example, one of the SPs is given in Eq. 4.

$$SP = (\{ambient\_temp\_in = HS\}, \{heat\_enable = AA\}) \tag{4}$$

They check whether the flow of the ambient temperature signal to the house is dependent on the *SIs*. The SPs ensure that there are no dependencies that affect the ambient temperature signal. The created DDG had 550 nodes, whereas the dependency sets of the SPs had between 100 to 300 nodes. No variable that was dependent on *SIs*, was a controlling variable to the ambient temperature signal, therefore all SPs were satisfied. The static analyzer took 58.3 s to finish.

VI. Conclusion

In this paper, we presented VAST: a novel VP-based IFT tool against availability security properties for heterogeneous systems. At the heart of VAST is a scalable static information flow analysis that operates directly on the SystemC AMS VP models. The analysis performs data flow analysis and static taint analysis to identify static paths that violates specified availability properties. These potentially vulnerable paths are

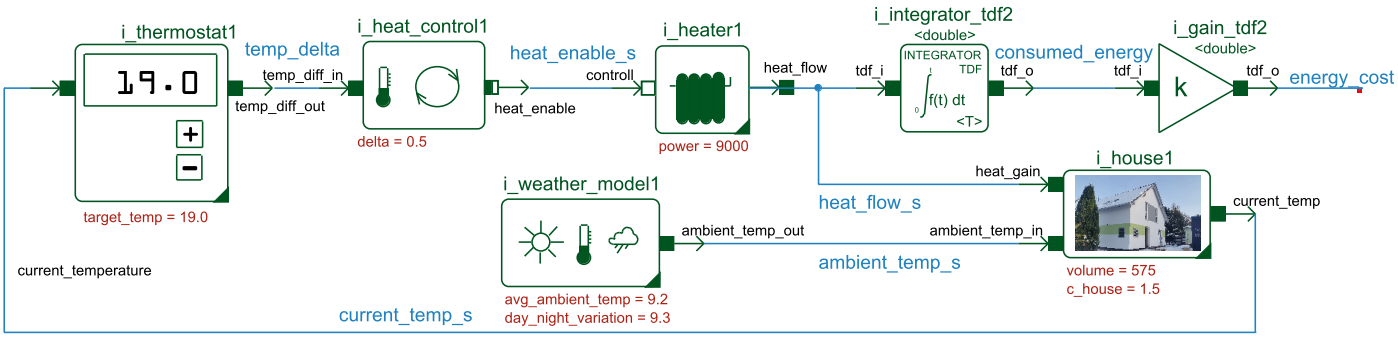

Fig. 10. The block diagram the Thermal House System

reported back to the user for further inspection. We have demonstrated the effectiveness of VAST on two real-world systems.

VAST utilizes an overapproximation technique, which guarantees the correctness of all solutions found, while also potentially providing additional solutions for further analysis. Future work in this area could include investigating correctness of these security properties at lower levels of abstraction. These studies would help to further our understanding of the transferability of verified security properties. Overall, this research represents a significant step forward in security validation.

REFERENCES

[1] I. Polian, "Security aspects of analog and mixed-signal circuits," in *2016 IEEE 21st International Mixed-Signal Testing Workshop (IMSTW)*, Jul. 2016, pp. 1–6.

[2] A. Antonopoulos, C. Kapatsori, and Y. Makris, "Trusted Analog/Mixed-Signal/RF ICs: A Survey and a Perspective," *IEEE Design & Test*, vol. 34, no. 6, pp. 63–76, Dec. 2017.

[3] M. Elshamy, "Design for security in mixed analog-digital integrated circuits," Ph.D. dissertation, Sorbonne université, 2021.

[4] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems*, ser. CHES'12. Berlin, Heidelberg: Springer-Verlag, Sep. 2012, pp. 23–40.

[5] M. Hassan, V. Herdt, H. M. Le, D. Große, and R. Drechsler, "Early SoC security validation by VP-based static information flow analysis," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 400–407.

[6] M. Hassan, D. Große, and R. Drechsler, *Enhanced Virtual Prototyping for Heterogeneous Systems*. Springer, 2023.

[7] R. Drechsler, M. Diepenbeck, D. Große, U. Kühne, H. M. Le, J. Seiter, M. Soeken, and R. Wille, "Completeness-Driven Development," in *Graph Transformations*, ser. Lecture Notes in Computer Science, H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, Eds. Berlin, Heidelberg: Springer, 2012, pp. 38–50.

[8] "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, Jan. 2012.

[9] F. Ghenassia, Ed., *Transaction Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Boston, MA: Springer US, 2005.

[10] M. Barnasconi, C. Grimm, M. Damm, K. Einwich, M. Louërat, T. Maehne, F. Pecheux, and A. Vachoux, "Systemc ams extensions user's guide," *Accellera Systems Initiative*, 2010.

[11] M. Lora, S. Vinco, E. Fraccaroli, D. Quaglia, and F. Fummi, "Analog models manipulation for effective integration in smart system virtual platforms," vol. 37, no. 2, pp. 378–391, 2018.

[12] M. Barnasconi and S. Adhikari, "ESL Design in SystemC AMS: Introducing a top-down design methodology for mixed-signal systems: Invited," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 1–5.

[13] F. Pêcheux, C. Grimm, T. Maehne, M. Barnasconi, and K. Einwich, "SystemC AMS based frameworks for virtual prototyping of heterogeneous systems," 2018, pp. 1–4.

[14] M. Hassan, D. Große, T. Vörtler, K. Einwich, and R. Drechsler, "Functional coverage-driven characterization of RF amplifiers," 2019, pp. 1–8.

[15] M. Goli, M. Hassan, D. Große, and R. Drechsler, "Security validation of VP-based SoCs using dynamic information flow tracking," *it - Information Technology*, vol. 61, no. 1, pp. 45–58, Feb. 2019.

[16] M. Goli and R. Drechsler, "ATLaS: Automatic Detection of Timing-based Information Leakage Flows for SystemC HLS Designs," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2021, pp. 67–72.

[17] M. Goli and R. Drechsler, "Early SoCs Information Flow Policies Validation using SystemC-based Virtual Prototypes at the ESL," *ACM Transactions on Embedded Computing Systems*, Jun. 2022.

[18] P. Pieper, V. Herdt, D. Große, and R. Drechsler, "Dynamic Information Flow Tracking for Embedded Binaries using SystemC-based Virtual Prototypes," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, Jul. 2020, pp. 1–6.

[19] K. Abdul Sattar and A. Al-Omary, "A survey: Security issues in IoT environment and IoT architecture," in *3rd Smart Cities Symposium (SCS 2020)*, vol. 2020, Sep. 2020, pp. 96–102.

[20] W. Hu, A. Ardeshiricham, and R. Kastner, "Hardware Information Flow Tracking," *ACM Computing Surveys*, vol. 54, no. 4, pp. 83:1–83:39, May 2021.

[21] R. Kastner, J. Oberg, W. Huy, and A. Irturk, "Enforcing information flow guarantees in reconfigurable systems with mix-trusted IP," in *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. The Steering Committee of The World Congress in Computer Science, Computer ..., 2011, p. 1.

[22] L. Piccolboni, G. Di Guglielmo, and L. P. Carloni, "PAGURUS: Low-Overhead Dynamic Information Flow Tracking on Loosely Coupled Accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2685–2696, Nov. 2018.

[23] P. Subramanyan, S. Malik, H. Khattri, A. Maiti, and J. Fung, "Verifying information flow properties of firmware using symbolic execution," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2016, pp. 337–342.

[24] M.-M. Bidmeshki, A. Antonopoulos, and Y. Makris, "Information flow tracking in analog/mixed-signal designs through proof-carrying hardware IP," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, Mar. 2017, pp. 1703–1708.

[25] M. M. Bidmeshki, A. Antonopoulos, and Y. Makris, "Proof-Carrying Hardware-Based Information Flow Tracking in Analog/Mixed-Signal Designs," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 415–427, Jun. 2021.

[26] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog Malicious Hardware," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 18–37.

[27] N. I. Ponce de León Puig, L. Acho, and J. Rodellar, "A Hysteresis Dynamic Mathematical Model Approach to Parametric Estimation System," *Mathematical Problems in Engineering*, vol. 2021, p. 6628380, Feb. 2021.

[28] E. Jonsson, "Towards an integrated conceptual model of security and dependability," in *First International Conference on Availability, Reliability and Security (ARES'06)*, Apr. 2006, pp. 8 pp.–653.