AFRL-RI-RS-TR-2024-036

PASHI: PROGRAMMABLE ASSURE AND SECURE HARDWARE INTERFACE

ARIZONA STATE UNIVERSITY

APRIL 2024

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2024-036 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
TODD N. CUSHMAN
Work Unit Manager

/ S /
JAMES S. PERRETTA
Deputy Chief
Information Warfare Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
APRIL 2024	FINAL TECHNICAL REPORT	JUNE 2023	DECEMBER 2023

4. TITLE AND SUBTITLE

PASHI: PROGRAMMABLE ASSURE AND SECURE HARDWARE INTERFACE

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
N/A	FA8750-23-1-0500	62788F
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
N/A	N/A	R3H4

6. AUTHOR(S)

Michel A. Kinsy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Arizona State University
699 Mill Avenue, Suite 395
Tempe AZ 85281

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Air Force Research Laboratory/RIGA 525 Brooks Road Rome NY 13441-4505	AFRL/RI	AFRL-RI-RS-TR-2024-036

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The core innovations in the research project are a formal method for designing device-oblivious secure interface architecture with programmable access control and authentication parameters, and an automated hardware logic mapping.

15. SUBJECT TERMS

FPGA, Hardware Formal Verification Methods, Hardware security semantic models, VHDL, Verilog, Cyber Risk Assessment

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	18

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
TODD N. CUSHMAN	N/A

PREVIOUS EDITION IS OBSOLETE.

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 SUMMARY

In this project, as part of the TRENCH Program's discovery efforts, the PI design, developed, and demonstrated a secure computing framework where Commercial-Off-The-Shelf (COTS) devices with different trust levels, i.e., secure and non-secure, can be safely integrated onto the same system.

The core innovations in the research project are a formal method for designing device-oblivious secure interface architecture with programmable access control and authentication parameters, and an automated hardware logic mapping.

The secure hardware attachment that enables untrusted/legacy embedded COTS devices to securely interoperate within distributed connected heterogeneous environments is called PASHI (Programmable Assure and Secure Hardware Interface). PASHI provides a set of formal verification methods for (1) defining a secure hardware interface and the associated digital logic, (2) providing a user tool for specifying the composition and enforcement of user-defined secure rules at a device's interface, and (3) generating a set verifiable authentication attribute for the hardware logic to securely interact with the rest of the connected distributed computing environment.

PASHI assists system engineers in designing their system through an iterative process consisting of the steps (1) specification analysis, (2) hardware model generation (3) hardware model and logic checking, and (4) FPGA instantiations.

The PI evaluated and validated PASHI and its security hardening capabilities through a concrete implementation instance that uses an off-the-shelf low-cost camera system and an FPGA board.

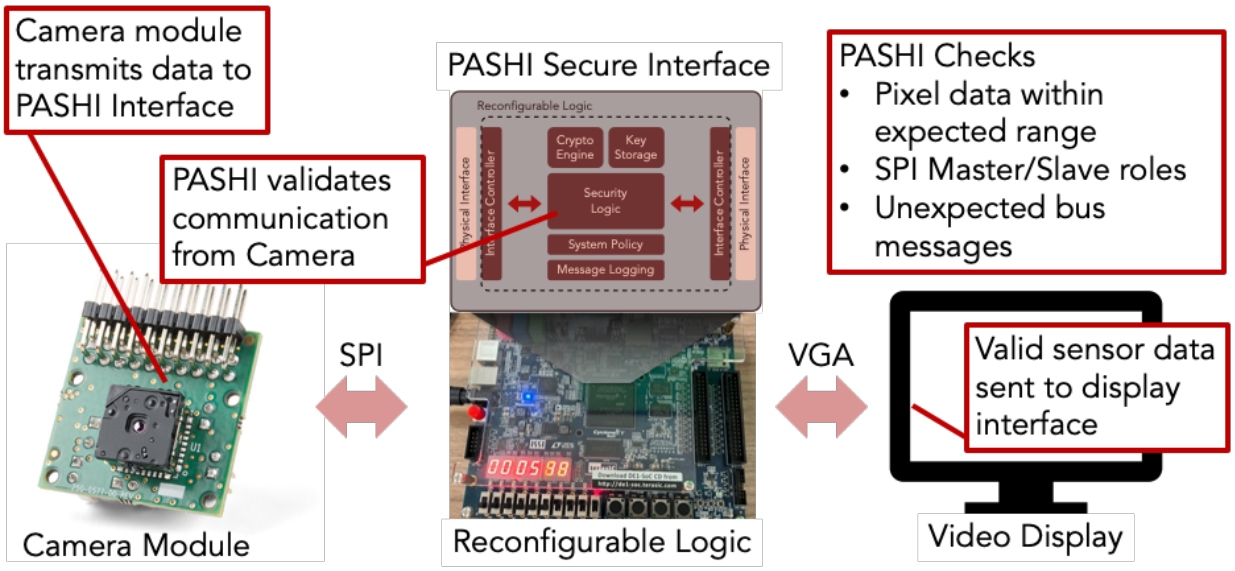

Figure 1: Programmable Assure and Secure Hardware Interface (PASHI) System Demonstration

2.0 INTRODUCTION

With the new distributed computing systems, like IoT-based systems, the provenance of hardware components and software modules is harder to establish. These systems are complex heterogeneous compute environments with legacy software and hardware, third party devices, ad-hoc or changing connected networks, etc. with very murky security models.

To address this challenge, as part of the TRENCH discovery effort, the PI proposed, designed, implemented, and validate a new hardware-enforced interface-based protection mechanism where different trust level devices can be integrated onto the same connected distributed system called ***PASHI (Programmable Assure and Secure Hardware Interface)***.

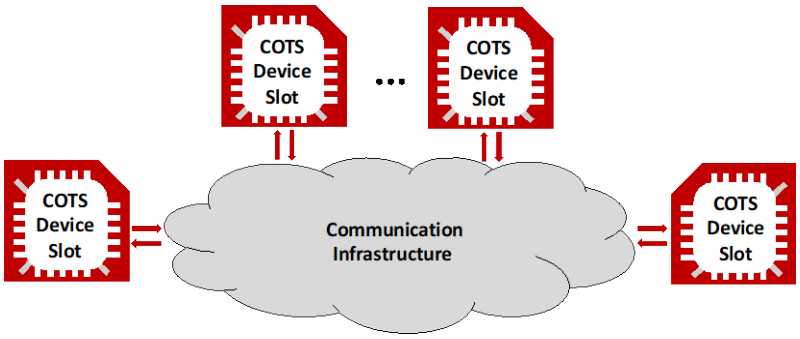

Figure 2: Current situation where different devices with difference provenances and varying levels of trust compute and share resources unchecked.

Figure 2 shows the conventional unsecure physically distributed connected device network. These commercial off-the-shelf (COTS) devices, e.g., camera, may come from different vendors and executing varying levels of trust firmware code while processing data and sharing resources with the rest of the network – including network bandwidth. This is a very fertile attack ground and represents the Achilles' heel of these heterogeneous architectures of distributed connected devices.

The central idea demonstrated in this project is a generalized security computing paradigm for constructing secure heterogeneous distributed connected devices, by developing a programmable, in-hardware, monitor and security rule enforcement at the interface of COTS devices. Figure 3 shows an overall architecture of this new secure distributed system.

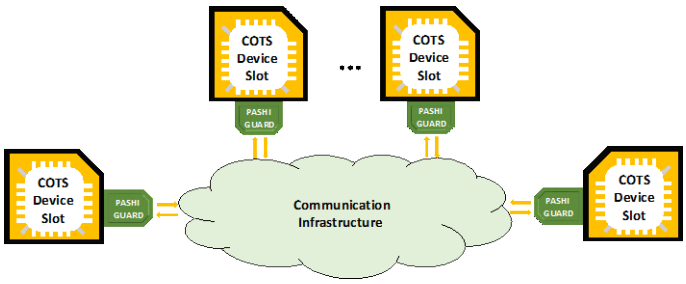

Figure 3: PASHI-Based Secure Distributed Connected Devices Architecture.

The intellectual merit of the PASHI (Programmable Assure and Secure Hardware Interface) approach set of principles for specifying user-defined security policies and designing the associated formally proved, composable, and programmable hardware guards. These guards use hardware as root-of-the-trust techniques, process and verify incoming and outgoing data communications against user-defined security rules, and can perform authentication, access privilege identification, and contain malicious behaviors of rogue devices. Figure 4 illustrates the guard's role in the system architecture.

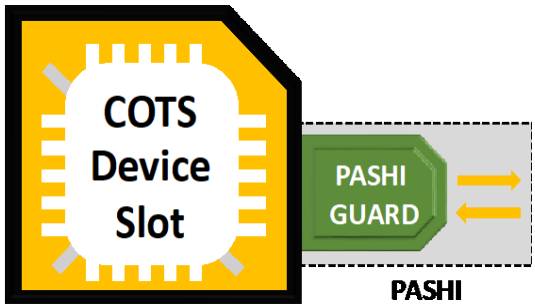

Figure 4: Interface-based, programmable, in-hardware, user-defined security guard.

PASHI provides a set of formal verification methods for (1) defining a secure hardware interface and the associated digital logic, (2) providing a user tool for specifying the composition and enforcement of user-defined secure rules at a device's interface, and (3) generating a set verifiable authentication attribute for the hardware logic to securely interact with the rest of the connected distributed computing environment. PASHI aims to assist system engineers in designing their system through an iterative process consisting of the steps (1) specification analysis, (2) hardware model generation (3) hardware model and logic checking, and (4) FPGA instantiations.

The programmability of the PASHI guard allows for the implementation of simple access and authorization rule checking logic at the interface to more complex packet inspection and encryption acceleration modules. Figure 5 shows a such complex deployment instance.

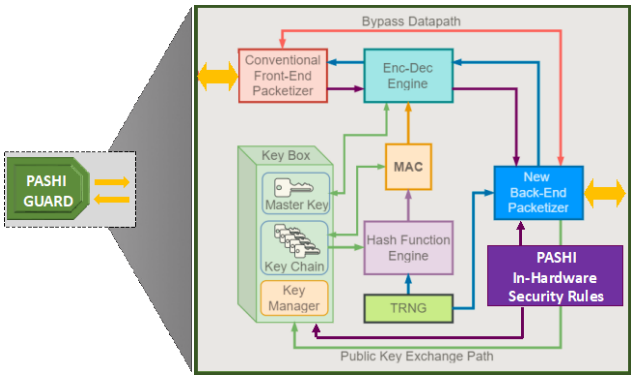

Figure 5: Complex hardware logic implementation in the guard.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

PASHI (Programmable Assure and Secure Hardware Interface) provides a systematic approach for specifying user-defined security policies and designing the associated formally proved, composable, and programmable hardware guards for these devices. Figure 6 depicts the system hardening flow using PASHI.

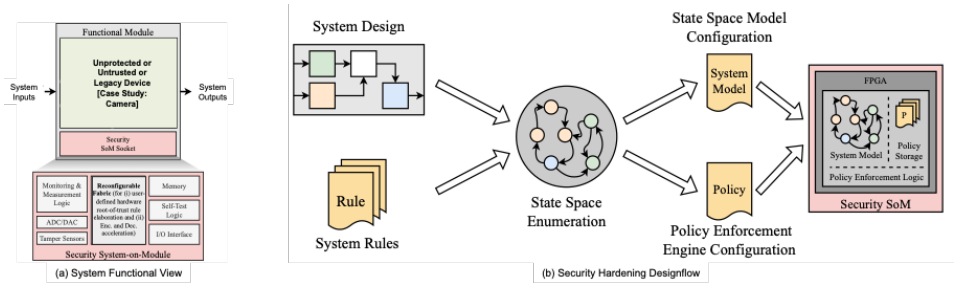

Figure 6: PASHI-Enabled System Hardening Designflow.

3.1 Pashi Security Rules Specification

To assess the security of a system, a list of security requirements is needed. Generally, it is common for the Subject Matter Expert (SME) to provide the design engineers with insight into the core concerns in a given domain. The SME possesses the knowledge related to the attack and threat models relevant to a system when deployed in an environment. This information is needed to both engineer a more secure but also formally verify and validate the current implementation of the security posture of the system.

For this aspect of the design, in the PASHI framework, we use Controlled Natural Language (CL) which closely resemble a natural language but with a stripped-down grammar. We are exploring the use of Semantic Application Design Language (SADL). Semantic Application Design Language is a Domain Specific Language (DSL) developed by General Electric (GE) [1]. A key feature of SADL is the familiar sentence-like structure which is used to define the semantic model. In PASHI, SADL acts as a bridge between

natural language requirements specification and formal logic. SMEs describe security requirements by first defining entities relevant to the system being designed. Complex concepts are then built on top of the primitive entities to create security requirements.

3.2 State Space Enumeration Through Formal Medule Simulation

PASHI uses a hybrid model checking and simulation approach to facilitate agile assessment of hardware interface security specifications. We model the simulated activity using the notion of a transition system. Transition systems are formally defined as a six-tuple $\langle S, ACT, \rightarrow, I, AP, L, \rangle$ where:

- S is the set of system states;
- Act is the set of actions or transition triggers;
- $\rightarrow$ is the transition relation $SxAxS$;
- AP is the set of atomic propositions (conditions);
- L is the labeling function.

Overall simulation can be viewed as three transition systems: the system under design, monitors, and test cases: (i) **System Under Design (SUD**): in this transition system, the labeling function takes security requirements as arguments. Therefore, in each state zero or more requirements may be satisfied. In a more abstract sense, the system transitions between *valid* and *invalid* states. Invalid states are those states in which there is at least one requirement that is not satisfiable; valid states are the complement of invalid states; (ii) **Monitors:** monitors utilize a given model of a system to check for security vulnerabilities. Conceptually, a monitor has two significant states: vulnerability detected, and vulnerability not detected; and (iii) **Test Case:** encapsulates a threat stimulus which begins in a dormant state and eventually becomes active by affecting the system. Removal of the stimulus marks the end of a test case.

3.3 Model Configuration Through Discrete-Time Markov Chains Simulations

Because the simulation transition system should attempt to emulate potential threats which could occur in a given deployment environment, randomness is needed within the model. Cyber-threats are not predictable in nature and therefore it is intractable to predict when an adversarial entity will target a system [2]. Conversely, it is difficult to predict how the monitors perform in response to a randomly occurring cyber-effect. Therefore, the simulation is *augmented* by treating the simulation as a stochastic process. Specifically, the simulation transition system is transformed into a *Discrete-time Markov Chain* (DTMC).

A labeled DTMC is a four tuple $\langle S, S_0, P, L \rangle$:

- S is a finite set of states;
- $S_0 \subset S$ is the initial set of states;
- $P : S \; x \; S \rightarrow [0, 1]$ is the transition probability matrix;
- $L : S \rightarrow 2^{AP}$ is the labeling function which assigns to each state atomic propositions which are currently true.

In a Markov Chain transition system, the probability transition matrix assigns to each possible transition a real number value indicating the chance of a transition occurring in

the future. A process described with probability distribution assigned to state transitions is said to satisfy the *Markov Property*. Stochastic processes which satisfy this property are "memoryless"; the knowledge of previous states is not needed but rather only knowing the present state is sufficient in predicting the future. Conceptually, we apply the Markov Property as a tool for measuring security.

3.4 Model Configuration Refinement & Optimization

Generating the hardware model for full formal verification requires to collect snapshots of the system state. Formally, we define a snapshot, S_t, to be a vector of system variables $\langle v_1^t, v_2^t, v_3^t, \dots, v_n^t \rangle$ captured at some time t. The sampling frequency or snapshot-latency can be configured to allow for the trading of space for model fidelity. After simulation, a vector of snapshots or simulation trace is generated. Similar to snapshots, a simulation trace is defined a n dimensional vector of snapshots $\langle \mathrm{S_t}, \mathrm{S_{t+1}}, \mathrm{S_{t+2}}, \dots, \mathrm{S_{t+n}} \rangle$ captured at each time-step.

Traces captured during simulation describe how frequent states were visited during execution. Using this knowledge, we generate a model in the form of a probability distribution for each state in the simulation transition system. The probability distribution function is defined as follows:

$$P(s, s') = \frac{visit(s, s')}{\Sigma_{s' \in Post(s)} visits(s, s')}$$

where:

- s is the current state;
- s' is a successor state;
- *visit* is the frequency of visits from state s to successor state s';
- *Post* is the set of all valid successor states s' given an origin state s.

The simulation transition system is *augmented* with the probability distribution. Model generation concludes with defining the model using a model description language. For this work, we chose to use PRISM [3] modeling language for later use of the PRISM model checking. In this language, models are grouped into *modules* which are logical containers for *variables* and *commands*. Transitions from one state to another are dictated by the defined commands.

$$[\,]\ gaurd \rightarrow prob_1 : update_1 + \cdots + prob_n : update_n$$

Commands describe one or more updates which modify variables within a module. Updates are *guarded* by only taking place if the guard condition is true. It is required that each update probability will sum to the value of one for each command to be deemed valid. It is also important to note that in a normal DTMC each transition counts as a single time-step; all time-steps are given a unit value.

3.5 Model Configuration Translation Into Hardware Logic

The project will produce a model checking and verification framework with FPGA validation implementation. Conducting model checking requires the translation of security specifications defined in SADL into Probabilistic Computational Tree Logic or PCTL. PCTL is commonly known as a superset of Computation Tree Logic (CTL) with a notable addition of real-valued probability added to propositions. Grammar of PCTL is roughly partitioned into two groups: *states* and *paths*.

- **State:** $\Phi ::= true \mid proposition \mid \neg\Phi \mid \Phi \wedge \Phi \mid P_{\sim p\,[\phi]}$;
- **Path:** $X\,\Phi \mid \Phi \cup^{\leq k} \Phi$.

State formula, Φ, consists of one or more propositions or conditions. A state is said to satisfy a state formula if the transition probability is within the range of $\sim (p)$. Formally, we can say that a state, s satisfies a formula using $s \vDash \Phi$. Path syntax describes propositions on the level of paths, which are a finite or infinite sequence of states derived from an execution of a DTMC. Path formula ϕ can be composed of state formula through the use the until operator $\cup$ which can be bounded by some $k \in \mathbb{N} \cup \{\infty\}$. It is also possible to construct path formula by using the logical operators $\Box$ (globally) and $\diamond$ (eventually). A path which satisfies proposition $P_{\sim p}[\Box\Phi]$consists of only states in which this proposition is true. Path formula $P_{\sim p}[\diamond\,\Phi]$ can be interpreted as a relaxation; it is satisfied if the proposition *eventually* evaluates to true within one of the states in the path. As an example, the security requirement specified in SADL will be translated into the following PCTL formula:

$$P_{\leq 0.5}[false \cup^{\leq 10} detected]$$

Providing this formula as input into a probabilistic model checker, such as PRISM, will return a qualitative answer in the form of true or false.

3.6 Pashi Interface Guard Architecture

The PASHI hardware guard is responsible for monitoring, filtering, and analyzing data traffic coming from the COTS or legacy device into data packets formatted according to the security rule specifications that can safely be routed inside the network, and for reconstructing packets into data traffic at the opposite side when exiting the network.

Based on the security rules, the guard can completely transform the traffic from and to the legacy/COTS device, and render it unrecognizable via obfuscation, communication protocol changes, and secure through adaptive encryption mechanisms.

The guard has two datapaths: one encrypted and one bypass. The bypass path allows for the disabling or power-gating of the encryption function per device or session or through event specific event detection. The master encryption keys are localized, device specific through PUF-based (Physical Unclonable Function) approaches, and made anti-tamper.

The PASHI hardware guard (i) provides trusted interface to system, (ii) supports formal verification for security validation, (iii) guarantees system policies, (iv) enforces master-slave bus roles, (v) provide encryption transparently to COTS/legacy devices, and (vi) enables configuration validation with hardware Root-of-Trust.

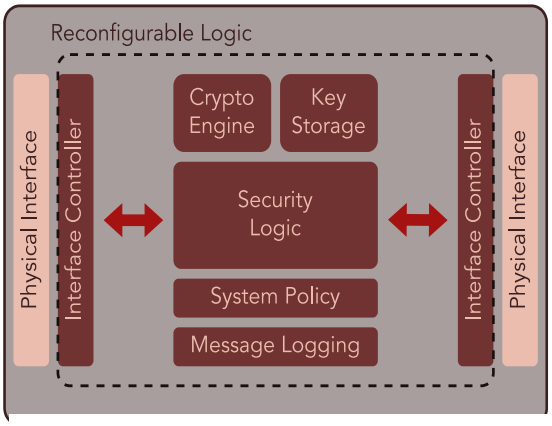

Figure 7: PASHI Guard Architecture

4.0 RESULTS AND DISCUSSION

To demonstrate the effectiveness and practicality of the PASHI, we implemented a prototype system using a camera and a field programmable gate array (FPGA). We developed parameterized register-transfer level (RTL) components (hardware reconfiguration library) for the PASHI Guard for user-defined security rules, hardware-based runtime analysis on synthetically generated data for design evaluation and validation. We created a functional FPGA system prototype with real-time security monitoring and rule executions with low system and latency overhead.

4.1 Pashi Fpag Project Setup & Synthesis

The PASHI project demonstration folder (https://github.com/stamcenter/PASHI) includes all of the Verilog and Quartus project files necessary for FPGA synthesis on the TERASIC DE2-115 development board. To re-synthesize the PASHI demo, make sure the following files are included in the Quartus project. The DE2_115_GOLDEN_TOP.qpf Quartus project file has been setup with the settings and files necessary for the PASHI demo. A pre-synthesized bitstream is provided in fpga/quartus/DE2_115_GOLDEN_TOP.sof

Table 1: Demonstration Deliverable GitHub Structure

File Name	Description
Quartus/DE2_115_GOLDEN_TOP.qpf	Quartus project file
Quartus/DE2_115_GOLDEN_TOP.qsf	Quartus settings derived from the DE2-115 golden template settings
Quartus/DE2_115_GOLDEN_TOP.sdc	Synopsys Design Constraints for clocks and I/O on the DE2-115 development board.
Src/top.v	Top level Verilog module for the PASHI Demo
Src/vospi.v	Flir Lepton VoSPI interface controller.
Src/vga_ctrl.v	VGA display driver controller
Src/dual_port_BRAM.v	Dual port memory for VGA frame buffer. One port for VGA read, one port for VoSPI write.
Src/security_monitor.v	Monitoring logic to observe VoSPI signals and enable/disable communication based on user-programed security policies.
Quartus/Pll_100.qip	Quartus PLL IP module for 100MHz clock to drive VoSPI controller.
Quartus/Pll_25_125.qip	Quartus PLL IP module for 25.125MHz clock to drive VGA module.
Quartus/DE2_115_GOLDEN_TOP.sof	Pre-Synthesized PASHI demonstration bitstream for the DE2-115 FPGA.

4.2 Pashi Demonstration Camera Setup

The PASHI demonstration connects the FLIR (Forward Looking Infrared) Lepton camera to the DE2-115 FPGA through the FPGA's GPIO (General Purpose Input/Output) header. The figure below shows the pin connections between the FPGA and FLIR Camera. Note the notch in the FPGA GPIO connector to determine polarization/orientation. Power is supplied to the FLIR Lepton camera through the 3.3V and GND pins on the FPGA GPIO header. Connect the 3.3V and GND pins to the dedicated power pins on the back of the FLIR camera instead of through the 20-pin header on the bottom. Additional documentation from FLIR describing the camera and the VoSPI interface is included in the resources/ directory.

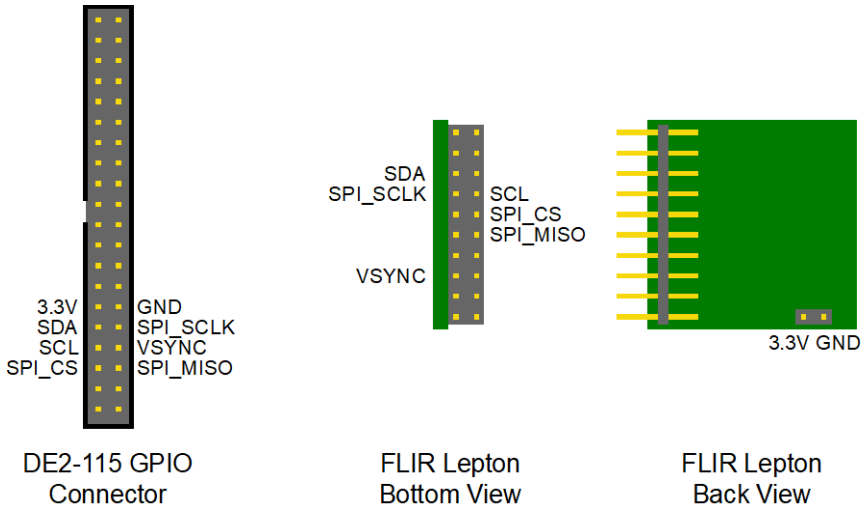

Figure 8: Camera Application Setup Components

4.3 Pashi Full System Demonstration

Figure 9 shows connections necessary for the PASHI demonstration (left) and the complete PASHI demonstration setup (right). In addition to the camera hookup described above, be sure to connect the FPGA power and VGA (Video Graphics Array). Note that the USB-blaster USB cable connection necessary to configure the FPGA is not shown here. DE2-115 Switch 0 enables or disables the FLIR Lepton camera display.

Setting the switch to the On position will begin capture of 80x60 8-bit images from the Flir lepton camera at 27 frames per second (FPS). Switching off the switch will freeze the last captured image on the VGA display.

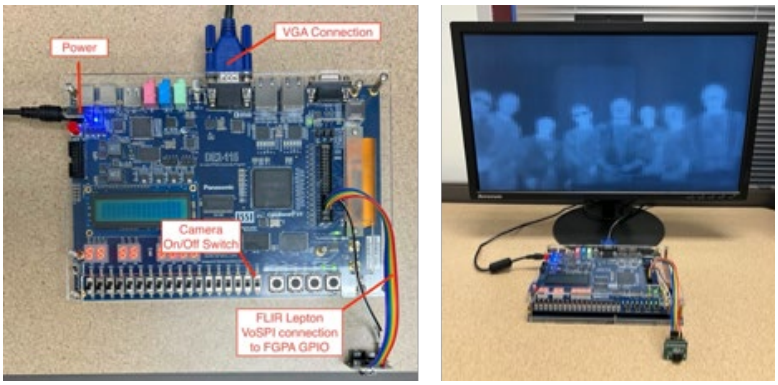

Figure 9: FPGA Connections Setup

Figure 10 shows two 80x60 8-bit color photos captured from the FLIR Lepton camera while the security monitor observed the SPI (Serial Peripheral Interface) traffic and enforced user-defined security policies. The photo on the left is a group photo of eight members of the STAM Center. The photo on the right is portrait of a single person with glasses and another person in the distant background on the left.

Figure 10: Experimental Results

5.0 CONCLUSIONS

Under this project, the PI was able to demonstrate the feasibility to significantly improve the security of COTS and legacy devices integration in critical distributed systems. The PI successfully formulate a formal specification, design, verification, and validation algorithms for hardening COTS and legacy devices. The PI produced a prototype system using a camera as the COTS Device and an FPGA for the PASHI.

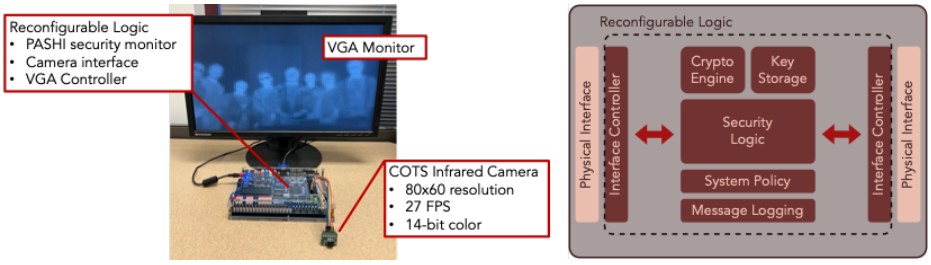

Figure 11: Working Deliverable of PASHI Project

It should be noted that although the PASHI-based design methodology shows a lot promise, this effort is merely a discovery effort over a short period. Many aspects of work remained unexplored, including (a) a design automation platform to aid in the process, (b) scalability and flexibility study across multiple devices, and (c) comprehensive latency and throughput evaluations.

6.0 REFERENCES

[1] A. Crapo and A. Moitra, "Toward a Unified English-Like Representation of Semantic Models, Data, and Graph Patterns for Subject Matter Experts," International Journal of Semantic Computing, vol. 07, no. 03, pp. 215–236, 2013.

[2] K. Jabbour and J. Poisson, "Cyber Risk Assessment in Distributed Information Systems," The Cyber Defense Review, no. i, pp. 91–112, 2016.

[3] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in Proc. 23rd International Conference on Computer Aided Verification (CAV'11), ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–59

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

DOD	Department of Defense
TRENCH	Tech Reprogrammable Hardware to Enhance Heterogeneous Connected Networks
PASHI	Programmable Assure and Secure Hardware Interface
FPGA	Field Programmable Gate Array
RTL	Register Transfer Level
GPIO	General Purpose Input/Output
FLIR	Forward Looking Infrared
SPI	Serial Peripheral Interface
VGA	Video Graphics Array
ASU	Arizona State University
STAM	Secure, Trusted, and Assured Microelectronics