Architectural Enhancements to Increase Trust in Cyber-Physical Systems Containing Untrusted Software and Hardware

Mohammed M. Farag

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Cameron D. Patterson, Chair
Thomas L. Martin
Henning S. Mortveit
Binoy Ravindran
Sedki M. Riad
Mohamed R. Rizk

September 17, 2012
Blacksburg, Virginia

Keywords: Cyber-Physical Systems, Trusted Computing,
Cognitive Radio, Embedded Systems Security, Reconfigurable Hardware,
Hardware Trojans, Process Control Systems

Architectural Enhancements to Increase Trust in Cyber-Physical Systems Containing Untrusted Software and Hardware

Mohammed M. Farag

Abstract

Embedded electronics are widely employed in cyber-physical systems (CPSes), which tightly integrate and coordinate computational and physical elements. CPSes are extensively deployed in security-critical applications and nationwide infrastructure. Perimeter security approaches to preventing malware infiltration of CPSes are challenged by the complexity of modern embedded systems incorporating numerous heterogeneous and updatable components. Global supply chains and third-party hardware components, tools, and software limit the reach of design verification techniques and introduce security concerns about deliberate Trojan inclusions. As a consequence, skilled attacks against CPSes have demonstrated that these systems can be surreptitiously compromised. Existing run-time security approaches are not adequate to counter such threats because of either the impact on performance and cost, lack of scalability and generality, trust needed in global third parties, or significant changes required to the design flow.

We present a protection scheme called Run-time Enhancement of Trusted Computing (RETC) to enhance trust in CPSes containing untrusted software and hardware. RETC is complementary to design-time verification approaches and serves as a last line of defense against the rising number of inexorable threats against CPSes. We target systems built using reconfigurable hardware to meet the flexibility and high-performance requirements of modern security protections. Security policies are derived from the system physical characteristics and component operational specifications and translated into synthesizable hardware integrated into specific interfaces on a per-module or per-function basis. The policy-based approach addresses many security challenges by decoupling policies from system-specific implementations and optimizations, and minimizes changes required to the design flow. Interface guards enable in-line monitoring and enforcement of critical system computations at run-time. Trust is only required in a small set of simple, self-contained, and verifiable guard components. Hardware trust anchors simultaneously addresses the performance, flexibility, developer productivity, and security requirements of contemporary CPSes.

We apply RETC to several CPSes having common security challenges including: secure reconfiguration control in reconfigurable cognitive radio platforms, tolerating hardware Trojan threats in third-party IP cores, and preserving stability in process control systems. High-level architectures demonstrated with prototypes are presented for the selected applications. Implementation results illustrate the RETC efficiency in terms of the performance and overheads of the hardware trust anchors. Testbenches associated with the addressed threat models are generated and experimentally validated on reconfigurable platform to establish the protection scheme efficacy in thwarting the selected threats. This new approach significantly enhances trust in CPSes containing untrusted components without sacrificing cost and performance.

Dedication

I dedicate this dissertation to my family, particularly …

to my late father
— your aspirations have inspired my life,
I wish you were among us to see your dreams realized;

to my beloved mother
— your encouragement was my main motivation,
and your prayers for me were what sustained me thus far;

to my precious wife
— without your understanding, support, and patience,
I would not be able to continue and succeed;

to my wonderful children
— your innocence was my fuel and your smile was my remedy,
I will do my best to grant you a better future.

Acknowledgements

First and foremost, praises and thanks to Allah, the Almighty, for His immense blessings and help granted to me throughout my life, and for providing me this opportunity and granting me the capability to proceed successfully. This thesis appears in its current form due to the assistance and guidance of several people. I would therefore like to offer my sincere thanks to all of them.

I would like to express my deepest respect and most sincere gratitude to my advisor, Dr. Cameron D. Patterson, for his encouragement, guidance, and support and for providing the opportunity to work on novel and challenging research problems. He did not save an effort to improve my research and technical writing skills. His constructive criticism and invaluable suggestions from the initial conception to the end of this work have helped my research and enabled me to develop a better understanding of the subject. Especially, I would like to thank him for the great effort and considerable time he invested in the dissertation review. I am greatly indebted to his assistance and understanding in matters of non-academic concern which have helped me endure some difficult times during my study period. He is a dedicated person and real advisor, and I am extremely privileged to have been a student under his supervision.

I extend my gratitude as well to the remaining committee members for serving on my committee. Especially, I would like to thank Dr. Sedki M. Riad for his extraordinary effort to found the VT-MENA program and struggle to establish a research foundation in Egypt. Also, I would like to thank Dr. Henning S. Mortveit for his advice about the dissertation topic and interest in this research. I thank Dr. Thomas L. Martin and Dr. Binoy Ravindran for willingly accepting to be on my committee. I would like to sincerely thank Dr. Mohammed R. Rizk, my master's degree advisor, for supporting me during my undergraduate and graduate levels and serving on my committee.

I want to thank my colleague Lee W. Lerner for the helpful collaboration and fruitful discussions we held throughout our research partnership, and for his contributions to this dissertation.

A very special appreciation is due to Dr. Ioannis Besieris, an emeritus professor in VT, for his hospitality, concern, and unreserved support he gave not only for me but also to my colleagues. He is a real scholar who taught us many valuable lessons on both the academic and personal levels.

From the bottom of my heart, I would like to thank my family for their endless support, love, prayers, and advice throughout my life. I cannot find words to express my gratitude to my parents who sacrificed a lot in their lives to bring me to my current position. Certainly, I would not have reached this stage without their love and sacrifices. I would like to thank my sister for her support; she always had faith in all my endeavours.

I wish to express my heartfelt thanks and deepest gratitude to my wife, my life-partner, not only for her constant encouragement but also for her patience and understanding throughout my career. Without her unreserved support, completion of this work would not have been possible. I am also greatly indebted to our children for granting me warmth and happiness throughout.

Last but not least, for those who are not mentioned specifically, thank you.

Contents

List of Figures

List of Tables

List of Abbreviations

AES Advanced Encryption Standard

ASIC Application-Specific Integrated Circuit

AUSTIN Assuring Software Radios have Trusted Interactions

BDD Binary Decision Diagram

BMM Block RAM Memory Map

CBC Cipher Block Chaining

CC Clock Correction

CHARE Configurable Hardware-Assisted Rule Enforcement

CLB Configurable Logic Block

CoRaL Cognitive Radio Language

COTS Commercial of-the-shelf

CPLD Complex Programmable Logic Device

CPS Cyber-Physical System

CR Cognitive Radio

CTL Computation Tree Logic

DEFENSE Design-for-Enabling-Security

DLL Delay-Locked Loop

DMS Dynamic Module Server

DoS Denial-of-Service

DPR Dynamic Partial Reconfiguration

DREC Design Rule Enforcement Controller

DSA Dynamic Spectrum Access

DSP Digital Signal Processor

ECB Electronic CodeBook

EDA Electronic Design Automation

EDK Embedded Development Kit

FCC Federal Communications Commission

FPGA Field-Programmable Gate Array

FPL Field Programmable Logic and Applications

FSM Finite-State Machine

FTP File Transfer Protocol

GPIO General-purpose Input and Output

HDL Hardware Description Language

HLH Hardware limits raw hardware resource access

HLS Hardware limits access to software and data

HMAC Hash-based Message Authentication Code

HOST Hardware-Oriented Security and Trust

HTH Hardware Trojan Horse

I/O Input and Output

IC Integrated Circuit

ICAP Internal Configuration Access Port

IOB Input and Output Block

IP Intellectual Property

ISE Integrated Synthesis Environment

ISQ Idle Sequence

IT Information Technology

LFSR Linear Feedback Shift Register

LQG Linear Quadratic Gaussian

LTL Linear Time Logic

LUT Look-Up Table

LWIP Light Wight Internet Protocol

MGT Multi-Gigabit Transceiver

MIMO Multiple-Input and Multiple-Output

MMU Memory Management Unit

NGSCB Next Generation Secure Computing Base

NITRD US Federal Network and Information Technology Research and Development Program

NoC Network on Chip

NSF National Science Foundation

OS Operating System

PAU Plug-in Assist Unit

PCM Pulse Code Modulation

PCS Physical Coding Sublayer

PLB Processor Local Bus

PLC Programmable Logic Controller

PLL Phase-Locked Loop

PMA Physical Medium Attachment

PMD Physical Medium Dependent

PRM Partially Reconfigurable Module

PRR Partially Reconfigurable Region

PSL Property Specification Language

PUF Physically Unclonable Function

PWM Pulse Width Modulation

RAM Random Access Memory

RETC Run-time Enhancement of Trusted Computing

RMS Root Mean Square

ROBDD Reduced Order Binary Decision Diagram

RTL Register-Transfer Level

RTM Root of Trust for Management

SDK Software Development Kit

SDR Software-defined Radio

SecurIT Security of Internet of Things

SeReCon Secure Reconfiguration Controller

SHA Secure Hash Algorithm

SLH Software limits raw hardware resource access

SLS Software limits access to software and data

SMT Satisfiability Modulo Theories

SMV Symbolic Model Verifier

SoC System-on-Chip

SRAM Static Random-Access Memory

SRC Secure Reconfiguration Controller

SSPL Standard Security Policy Language

STH Software Trojan Horse

TCG Trusted Computing Group

TFTP Trivial File Transfer Protocol

TPM Trusted Platform Module

TSS Trusted Software Stack

TTS Tailored Trustworthy Spaces

TXT Trusted Execution Technology

USRP Universal Software Radio Platform

XG neXt Generation

XPS Xilinx Platform Studio

Chapter 1

Introduction

The rapid evolution of electronic design technologies and automation tools have led the way to the information technology (IT) revolution in the last two decades. Extremely fast personal computers and portable cell phones, numerous software applications and tools, the high-speed worldwide Internet, and computer networks have changed the way we live. The impact of this revolution obviously affects every single aspect of our lives. The vast majority of computing devices, however, are much less visible and embedded in most contemporary electronic devices in common use today. Embedded systems are special-purpose computing platforms designed for specific control functions under a set of constraints. Applications of embedded systems range from portable devices like PDAs and cellular phones, to real-time controls equipped in industrial systems and nuclear power plants. Many embedded systems are deployed in physical systems. However, there is still a computation gap between the discrete-time cyber world, where computations and information exchange take place, and the physical world of continuous dynamics in which we live [20].

The term cyber-physical system (CPS) was introduced by the National Science Foundation (NSF) to mean the integration of computation and a physical process [97]. Typically, a CPS is composed of physical process monitored and controlled by a cyber system, which is a networked system of distributed sensing, communication, and computational devices [170]. In a CPS, heterogeneous embedded systems monitor and control a physical process, usually via feedback loops where the

physical process affects computation and vice versa. Emerging CPSes are characterized by large-scale system sizes, heterogeneity of resources, uncertain system dynamics, and extensive physical interactions. Unlike traditional embedded systems aiming to optimize computation in highly constrained environments using limited resources, the emphasis of CPS research tends to be more on tight coupling between the computing and physical parts. Embedded systems can be considered CPSes when they sense or affect dynamic physical environments.

CPSes are the next computing revolution and, recently, the NSF has identified them as a key area of research. CPS applications include high-confidence medical systems, assisted living, traffic control and safety, advanced automotive systems, process control, energy conservation, environmental control, avionics, instrumentation, critical infrastructure control, defense systems, distributed robotics, industrial control systems, smart structures, cellular communication, and autonomic computing systems [96]. CPS design is an interdisciplinary research area of computer architecture, software, network, physical systems, control, and other engineering disciplines. This emerging area enables new opportunities and poses additional research challenges including: CPS decomposition and interoperability; robustness, safety and security; control of hybrid systems; real-time embedded system abstractions; architecture development; and model-based development [137, 151, 154]. Figure 1.1 illustrates CPS domains, design challenges, and potential applications.

Although CPSes have the potential to dramatically enhance our life and dwarf the 20^{th} century IT revolution, cyber threats targeting such systems can cause tremendous loss of lives and property. Traditional embedded systems require more security assurances than general-purpose computing platforms. In the transition to CPSes, security requirements should be increased to counter the rising number of threats aiming to damage physical systems through cyber attacks. Without enhanced security, they will not be deployed in applications such as health care, critical infrastructure, and other safety-critical systems. Nonetheless, interacting with the physical world opens the door for new generations of threats specifically targeting CPSes. Unfortunately, cyber security practices are commonly reactive, increasingly cumbersome, and struggle to mitigate rapidly evolving threats.

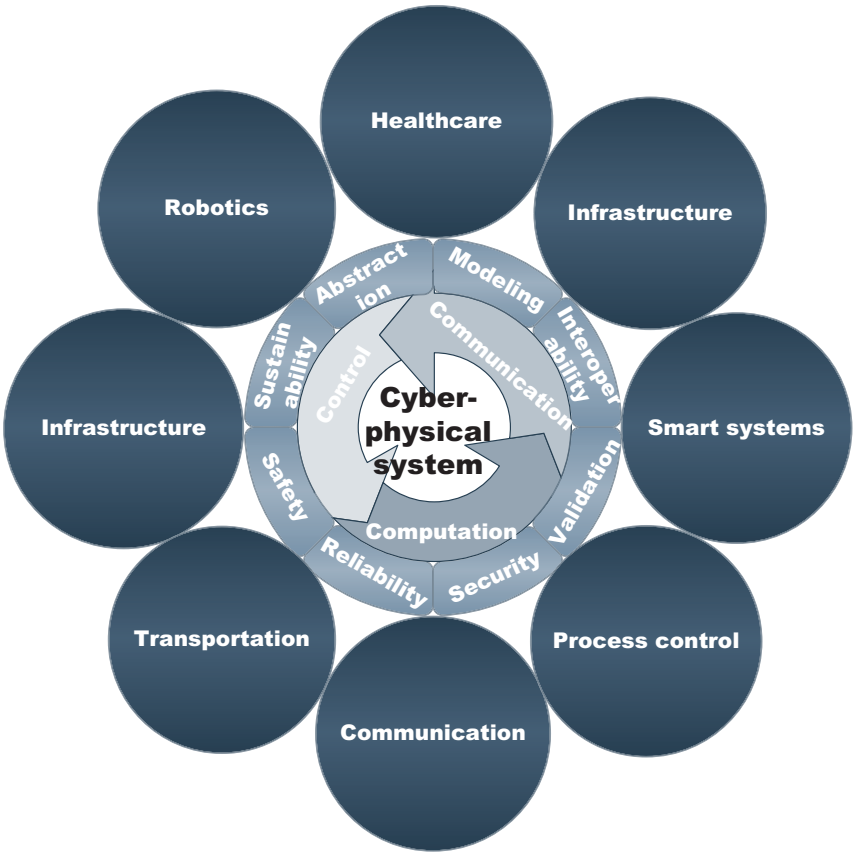


Figure 1.1: CPS domains, design challenges, and applications.

Typically, a CPS is configured as a closed-loop feedback control system. Figure 1.2 illustrates a CPS basic architecture, where sensors, actuators, and controllers can be a network of elements. Embedded controllers are computational systems incorporating a set of components such as processors, memory, storage, and input and output (I/O) devices. This infrastructure is abstracted to layers in order to simplify the development flow. System abstraction layers include hardware, operating system (OS), software, and data. Modern components are characterized by the complexity of functions and interactions, with data traversing through different abstraction layers.

To increase development productivity and reduce design costs, embedded controllers are often assembled from commercial of-the-shelf (COTS) components and third-party intellectual property (IP) modules. Even domestic development of modern systems is often assisted by third-party

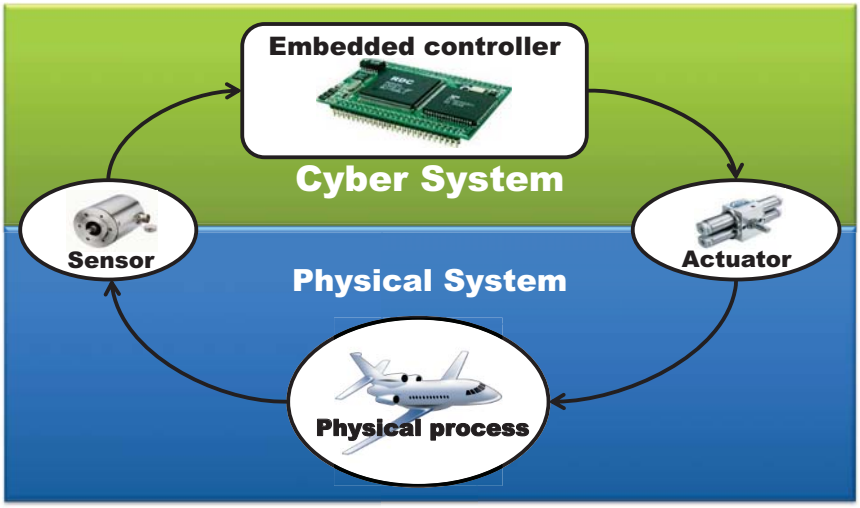


Figure 1.2: CPS basic architecture.

software tools. Open-source development is the major supplier of embedded software. IP cores are widely employed in both application-specific integrated circuit (ASIC) and field-programmable gate array (FPGA) designs and commonly distributed as netlist files. For example, IP cores may be delivered as synthesizable hardware description language (HDL) source that is accessible by the designer or as an encrypted netlist that can only be treated as a black box module.

Threats to CPSes can originate from numerous sources, including hostile governments, terrorist groups, disgruntled employees, malicious intruders, and untrusted insiders. Imported components and integrated circuits (ICs) are vulnerable to tampering and change by untrusted parties throughout the design and fabrication process. Commonly, such components cannot be trusted because of either lack of verification or the potential of Trojan inclusions. Trojans can be introduced to the global supply chain as either hardware or software modifications to embedded components. Programmable systems have added complexity as software and IP modules can also interface with hardware configuration controls. Since these controllers are often programmable, many Trojans need only to be implemented in software. Hardware trust may be even misplaced in reconfigurable platforms such as FPGAs.

Ensuring trust in third-party IP modules and COTS components individually is extremely challeng-

ing because there is neither an accompanying specification nor a golden reference design [167]. This problem is amplified and may result in system security violations when a controller is composed of numerous modules interfacing in a poorly trusted or understood manner. Even components developed by the system designer or other trusted parties cannot be verified using formal methods because of the associated difficulties. Despite the lack of trust in such components, feasible alternatives may not exist for the timely development of modern, complex systems. Therefore, using untrusted components in embedded controllers is inevitable. Enabling trusted computing requires threat-tolerant security solutions instead of solely relying on threat avoidance techniques.

Generally speaking, security solutions can be classified as either design-time (compile-time) or run-time [166]. Design-time techniques seek to verify that a computing system is flaw-free pre-deployment. Such techniques are extremely expensive in terms of both time and money, and can be only afforded for a limited set of applications. Verification techniques analyze systems against assigned specifications, which may not capture all vulnerabilities. An example of a design-time method is formal verification. Run-time techniques employ run-time trusted components to provide assurances about certain aspects of system behavior. The design flow can be changed or modified to add the trusted components at design-time. Encryption and authentication modules are typical examples of run-time trust anchors. They help to ensure information integrity and provide trusted compartments for secure applications. Nevertheless, such techniques do not ensure trust in all system components, and the system may remain vulnerable to unanticipated cyber threats.

The security research and design community is largely reactionary due to the imbalance of an attacker versus defender initiative. Consequently, systems are typically evaluated for trust and security or protections are created as an afterthought to the initial design process. This is especially true for legacy embedded systems, such as those used in the CPSes. The constant race to patch system vulnerabilities against newly discovered exploits indicates the need for proactive protections to be built-in during the design process. While such protections may be difficult to provide for highly complicated, general-purpose desktop and server computing platforms, it is feasible to create them for CPSes and embedded systems intended for specific applications. This is of critical importance as new threats are now targeting embedded controllers in CPSes deployed in national

infrastructure. Since it is reasonable to assume these threats are built into independently developed modules, and even trusted designers themselves will likely always be able to build systems they cannot entirely verify, protections should be built-in as well. New solutions are needed to tailor embedded systems and CPSes with control assurances that provide system-level trust throughout the system's lifetime.

1.1 Run-time Enhancement of Trusted Computing in CPSes

The US Federal Network and Information Technology Research and Development Program (NITRD) has been charged with leading the national cyber security initiatives. NITRD identified three initial R&D themes to exemplify and motivate future cyber security research activities: tailored trustworthy spaces (TTS), moving targets, and economic incentives [4]. For the TTS theme, meeting all security requirements derived from different contexts and purposes is infeasible and the alternative is establishing trustworthy sub-spaces defined in terms of specific uses or types of interactions, each with its own set of tailored security policies, services, and mechanisms. The moving target theme addresses resilience through agility by making the security environment more dynamic and therefore harder to predict. Finally, the economic incentives theme involves finding ways to encourage good security practices and motivating users to adopt cyber security defenses, recognizing that convenience has caused consumers to ignore security pop-ups [153]. These initial cyber security R&D themes pose many challenges for the research community to significantly enhance the trustworthiness of cyberspace [111].

The TTS theme provides flexible, adaptive, distributed trust environments that can support functional and policy requirements arising from a wide spectrum of activities to thwart evolving threats. User context is recognized and the TTS evolves with the context. The user chooses to accept the protections and risks of a tailored space. Attributes of the space must be expressed in an understandable way to support informed choice and must be readily customized, negotiated and adapted. The primary goal of the TTS theme is to identify and develop a common framework supporting

various trustworthy space policies and services for different types of actions. These policies and services provide visibility into the rules and attributes of the space to inform trust decisions, create a context-specific set of trust services, and a means for negotiating the boundaries and rules of the space [71]. This framework offers assurances about accurate articulation of user requirements in the TTS policy, true separation of included spaces, and build-up and tear-down of each space is clean and trustworthy. The scientific challenges of TTS are providing separation, isolation, policy articulation, negotiation, and assurances necessary to support specific cyber security sub-spaces [2].

Basic security objectives of CPSes as well as embedded systems include confidentiality, integrity, availability, and authenticity [30]. Future CPS generations must address several requirements of trusted computing, and require secure communication and data storage to achieve these goals. Security can be ensured only if these requirements are built into, rather than onto, system architectures and implementations. Trusted computing is an emerging technology aiming to implant trust in computer systems as well as hardware computing platforms. It has been developed and promoted by the Trusted Computing Group (TCG) with the goal of enforcing consistent behavior of the computing platform using security building-blocks rather than complete systems [68]. The TCG idea is to embed trust into a computing platform by providing it with a certain hardware module responsible for the platform security.

Lack of trust in the underlying system components is a common threat to CPSes as well as embedded systems. Run-time enforcement and verification of security policies is an emerging research topic gaining a growing interest of the security research community [23, 34, 99, 101–103]. In this dissertation, we advance a run-time protection scheme adopting the TTS theme to enable trusted computing in CPSes containing untrusted software and hardware. Particularly, this protection scheme targets CPSes built using reconfigurable hardware. The protection scheme is called Run-time Enhancement of Trusted Computing in CPSes (**RETC-CPS**).

Figure 1.3 illustrates a simplified block diagram of the RETC-CPS protection scheme. To protect against various cyber threats, a secure cyber-barrier is placed around the system control path. A

security policy captures permissible I/O behavior, and is enforced at run-time to simultaneously address design-for-security and -trust. Monitoring I/O behavior is particularly relevant for CPS applications. System behavior and security checks are synthesized at design-time from the application operational and security specifications. Guard components surround the system control logic, but do not affect the implementation of the control logic itself. Using such an approach, a tailored trustworthy control flow acts as a last line of defense for the target application. However, this fundamentally new approach is not domain-specific and provides a proactive approach to sustain system-level security with reliable control. Existing verification techniques are complemented but not exclusively relied upon to ensure functional system trust and security compliance.

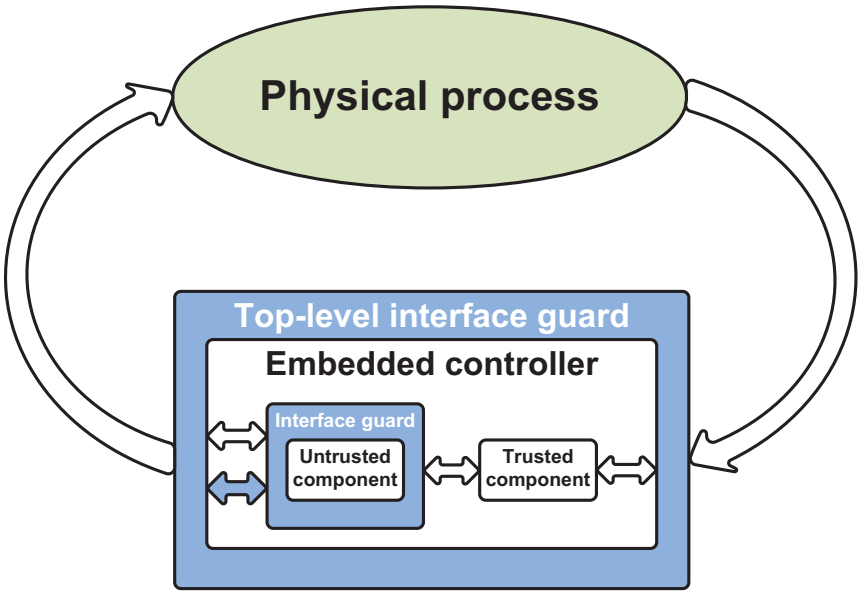


Figure 1.3: RETC-CPS block diagram.

Run-time policy-monitoring and -enforcing guards are tightly integrated into embedded controller interfaces. Abstract and system-level policy rules may only need to be evaluated on the top-level interfaces, such as statistical analysis of the I/O activity. This is an ideal scenario because designers can more easily reason about system security without the burden of developing specialized guards on a per-module basis. However, suspect or complex IP may still require intra-datapath interface guards enabling run-time monitoring and enforcement of operational and security specifications as

well as statistics gathering to evaluate performance. Protected modules and systems are treated as black boxes, and the security policy needs to be enforced at interfaces which can reveal the component internal transactions as well as external interactions. The guard components are provided with inclusive control and overriding capabilities over the protected system allowing them to respond appropriately to non-compliant behaviors according to the underlying policy measures. For example, detection of anomalous behavior can initiate a set of countermeasures such as forcing a safe operational mode, disabling the system, or switching to a backup system.

Interacting with the physical world is a featured property of CPSes distinguishing them from general-purpose computing platforms. This unique characteristic enables framing high-level policies and trustworthy spaces describing secure and trusted operation of the system. This application-specific security policy is tailored to a certain class of physical systems. However, the overall protection scheme is still applied for a wide range of CPSes. Further, circuit-dependent checks and user-specified rules such as restricted access resources and disallowed simultaneous control signals can be incorporated into the security policy addressing traditional IT threats. Furthermore, temporal logic propositions and assertion checkers employed in compile-time verification methods such as model checking can be included in the security policy enabling run-time verification of the system. Moreover, intrusion and anomaly detection algorithms can be adopted in the security policy and enforced at the top-level interfaces to detect specific cyber threats. The security policy is developed and integrated into the system by a trusted party aware of the system vulnerabilities, potential threats, and security requirements.

The policy-based approach to enhance trusted computing in CPSes addresses many security challenges by decoupling policies from system-specific implementations and optimizations. For example, this can enable maintaining system security without sacrificing other important metrics such as power and speed. Such an approach does not impose significant changes to the existing design flow, and does not need golden designs as security references. Further, RETC-CPS enables extensibility of the security policy over time to address zero-day attacks and threats. It enables updating guard components without requiring changes to the system implementation. Furthermore, the proposed protection scheme can combat both external and internal threats targeting CPSes because it

responds to the effect of an attack instead of attempting to prevent its cause. Therefore, RETC-CPS can be considered a threat-tolerant protection scheme, whereas compile-time security techniques seek threat avoidance. Moreover, the RETC-CPS protection scheme can be generally applied to enhance trust in various CPSes while having acceptable development and performance costs.

Hardware mechanisms have not been fully utilized or exploited in security solutions. Recently, hardware-enabled security has gained increasing interest from the research community because of the potential gains. Software-based protections can easily be undermined by compromising lower layers of the system software or attacking the underlying hardware. To counter such threats, the software has to be provided with hardware trust anchors supporting secure bootload and continuous monitoring of critical software while minimizing the trust chain. Hardware is the lowest abstraction layer of the computing platform and enforcing trust at this level enables detection of cyber threats targeting various layers of the system, including the hardware layer itself. Moreover, hardware designs are verifiable because of their limited state-space and reduced design complexity compared to software designs.

Hardware-based security solutions may fail because of their impact on performance, power consumption, cost, and usability. Such solutions are dealing with security as an ancillary rather than the main part of a system, and most these solutions target general-purpose platforms. Therefore, development of hardware-based architectures addressing these security concerns is a very challenging task. RETC-CPS tolerates implementing guard components at different abstraction layers, which facilitates the development of hardware-based security architectures. In this work, we employ reconfigurable hardware to implement the interface guard components of the RETC-CPS protection scheme. This simultaneously addresses the performance, flexibility, power, developer productivity, and security requirements of modern CPSes.

Reconfigurable hardware is a computing architecture characterized by both the flexibility of software and the high-performance of ASICs. A bitstream representing a digital design is used to configure fine-grained customizable logic and programmable interconnect to implement the design functionalities. The principal difference when compared to software architectures adopting

hardwired processors is the ability to make substantial changes to the datapath. On the other hand, the main difference with custom static hardware such as ASICs is the possibility to adapt the platform functionality in the field by loading new configuration bitstreams. Modern reconfigurable platforms such as FPGAs have dynamic partial reconfiguration capabilities allowing run-time adaptation of a part of the FPGA while the remaining part is still operating. The flexibility and high-performance characteristics of reconfigurable hardware fulfill the requirements of modern, data-intensive, computing applications. Reconfigurable hardware serves the purpose of RETC-CPS in terms of flexibility, locality, verifiability, resistance to software threats, and performance.

1.2 Research Challenges and Example Applications

The RETC-CPS protection scheme is applied to several problems associated with common security research challenges, and is supported by prototype developments illustrating scheme efficacy. The first problem is secure reconfiguration control of a dynamic reconfigurable system containing untrusted components. Challenges associated with this problem are validating various update requests issued by untrusted components and authenticating update contents delivered via open communication channels. The second problem is tolerating hardware Trojan horses (HTHs) in third-party IP cores used in CPS hardware platforms. Challenges associated with this problem are detecting anomalous behavior of untrusted components treated as black boxes and enforcing appropriate countermeasures in response. The third problem is mitigating cyber threats to process controllers containing untrusted components. Challenges associated with this problem are detecting erroneous behavior introduced by potential cyber threats and preempting their effects to preserve the controlled physical process stability. These security challenges commonly arise in computing platforms containing untrusted software and hardware components.

A major challenge associated with any security solution is evaluating it not only in terms of performance and overhead, but also in terms of trust and security [10]. Our security architectures are evaluated by presenting specific threat models, developing testbenches generating these models,

and validating the RETC capability to mitigate theses threats. The presented threat models are associated with either state-of-the-art cyber threats or novel zero-day attacks. We present our development of new HTHs supporting covert communication in high-speed serial interface adapters widely deployed in modern networks. The RETC-CPS protection scheme is also evaluated in terms of performance degradation, induced overheads, verifiability, automation applicability, trust locality, and attainable trust.

1.2.1 Secure Reconfiguration Control

Technological advancement, from high density ICs to communication protocols, has resulted in a push to develop a new type of software-defined radio (SDR), known as cognitive radio (CR) [7, 100], which can adapt its configuration based on perceived changes or opportunities in the radio environment. These radios support dynamic physical layer adaptation by scanning the spectrum, identifying spectrum opportunities from a wide range of operating frequencies, and adapting modulation waveforms and transmission power to take advantage of opportunities. These modern software radios are being developed with powerful protocol processors capable of supporting advanced networking, ranging from cross-layer protocols that tightly tune the operation of the MAC, network and transport layer to a new generation of distributed signal coding methods. Since CR devices are being built using programmable hardware which has computational capabilities rivaling closed commodity ASICs, they will be highly programmable with open interfaces and specifications, thus supporting open-source development by the broader community [141].

CR programmability introduces security concerns since all layers of the protocol stack can be modified, including hardware-implemented layers. Controls must be imposed on the allowed changes, and hardware should retain some oversight rather than rely solely on software correctness and integrity. Physical attacks on the mobile platform must also be considered. Trust inheritance becomes complicated in configurable platforms when software updates the underlying hardware structure. “In hardware we trust” is no longer axiomatic since the hardware can be modified to violate specific policies. Current practice places dynamic hardware configuration under the control

of application-level software. Partial hardware modules are application-specific and potentially untrusted. Software modification of hardware structure is analogous to self-surgery, and independent hardware should provide oversight rather than rely solely on the correctness and integrity of application software and circuits.

We develop a configurable hardware-assisted rule enforcement for CR (CHARE-CR) framework to automatically integrate PHY- and MAC-layer policy enforcement into the reconfigurable hardware blocks of a CR platform. A remote trusted server generates hardware plug-in by wrapping the waveform baseband modules in a controller block. Consequently, all update requests from CR software pass through the controller that serves as an abstraction layer for baseband datapath modules. An input to the controller synthesis algorithm is the subset of policy database rules affecting the PHY and MAC layers. CHARE-CR simultaneously addresses the performance, power, developer productivity, and security requirements of high-throughput, reconfigurable platforms.

The generated controller ensures that the requested operations are consistent with CR security policies, and returns an error status otherwise. Hardware detection of a policy violation indicates a possible corruption of the CR software. The controller-based abstraction simplifies the CR software, enhance portability, and increase radio efficiency and performance by managing the data path with optimized hardware. When the trusted authority changes the capabilities of the CR, a new configurable hardware plug-in is quickly generated using the synthesis software. Plug-in configuration data are securely transferred and stored so that the decrypted versions are physically inaccessible to CR software. CHARE-CR addresses a typical security threat associated with most modern CPSes containing untrusted components responsible for system updating control. This reconfiguration control approach can be generally applied to control updating of different CPS.

1.2.2 Tolerating HTHs in Third-party IP Cores

HTHs are malicious inclusions or alterations to ICs added deliberately by an entity involved in the IC development and production flow with the purpose of breaking one of the system's security

objectives. HTHs are a direct example of threats raised by untrusted components deployed in CPSes. Trojan effects on a host system range from subtle disturbances to catastrophic system failures. These effects include subtle changes to the chip functionality, performance downgrades, leaking information, and denial-of-service (DoS). Many taxonomies and classifications have been proposed to develop a better understanding of HTHs and create effective defenses [84, 85, 166]. HTHs are classified according to the insertion phase, abstraction level, activation mechanisms, effects or payload, and insertion location. These classifications are very useful in assessing Trojan detection, mitigation, thwarting, and protection techniques [140].

Development of HTH detection and protection methods is an emerging research topic gaining increasing interest in the security research community. Design-time detection approaches include physical analysis, logic-testing, Trojan activation, and side-channel analysis [120]. Contemporary ICs are characterized by high transistor density making it impossible to find subtle alterations using chip physical analysis in which suspicious ICs are compared to unaltered reference ICs. Moreover, such an approach to detect HTHs is destructive and the alternative is using extremely expensive, non-destructive imaging techniques. HTHs can add malicious functionality to a chip such as data leakage without disturbing the chip's normal operation. The Trojan's activity is usually triggered by rarely occurring conditions to make it difficult to detect the Trojan using activation and test vector generation approaches especially with the huge state spaces of modern ICs. Furthermore, conventional test and validation techniques cannot reliably detect HTHs because these methods focus on identifying undesired functional behavior rather than additional functionality added by a HTH. Other detection methods rely on the potential effects induced by HTHs to side-channel parameters such as current consumption, power trace, and path delay. However, the effectiveness of side-channel analysis is limited by process-variation effects and measurement noise which induce effects similar to small HTHs implanted in high-density ICs.

Trust verification in third-party IP cores is further complicated by the need for golden reference models which are very difficult to obtain in the global supply chain. Third-party IP cores are treated as black boxes where one can only trust functional specifications. A possible approach for trust verification is applying direct functional tests. However, such tests might not be able to

detect cleverly developed Trojans activated under rarely occurring conditions because of the huge state space of modern IP cores. Formal verification is another design-time technique that can be used to detect HTHs in IP cores pre-deployment. A high-level reference model is developed from the IP functional specifications and acts as a golden reference. However, formal verification techniques still experience many difficulties because of the huge state spaces of modern designs. As design-time HTH detection solutions might not provide comprehensive Trojan coverage, run-time monitoring can be employed to improve the provided assurances. Run-time approaches for Trojan detection rely on monitoring execution of critical computations to identify specific malicious or anomalous behaviors triggered during long in-field operation times. Potential countermeasures include disabling the chip upon malicious activity detection or bypassing the compromised modules and using backup components to allow reliable operation.

We apply the RETC-CPS protection scheme to detect and tolerate HTHs in third-party IP cores widely employed in reconfigurable hardware designs. Guard components are developed at design-time and attached to the IP module interfaces. A security policy is formulated from the IP functional specifications and translated into synthesizable hardware guards and security primitives responsible for monitoring and enforcement functionalities. Deviation from the nominal behavior triggers specific countermeasures mitigating HTH effects. Guard components enforcing functional specifications cannot detect non-functional security violations such as information leakage via side-channels, for example. However, the security policy can include custom rules to address non-functional specifications.

To evaluate our approach, we advance a lightweight HTH supporting two-way covert communication in serial high-speed point-to-point physical links by exploiting vulnerabilities in the underlying media layer protocols. Covert data channels are established by inserting HTHs exploiting unenforced, loose specifications of IP cores implementing media layer functionalities in a manner that maintains system operability. Specifically, we explore insertions in a PCS/PMA 10GBASE-X IP core implementing the physical layer functionalities of the 10GbE protocol specified in the IEEE 802.3-2008 standard [3], and in an Aurora IP core implementing the link-layer functionalities in a high-speed serial protocol. Aurora is an open-source implementation of a link-layer protocol

developed by Xilinx to support serial links between chips employing multi-gigabit transceivers (MGTs), with the protocol specifications adopted from the IEEE 802.3 standard [173].

1.2.3 Protecting Process Control Systems Against Cyber Attacks

Process control systems monitor and control physical processes using closed-loop feedback. Sensory information gathered by physical sensors are fed into an embedded system controlling electromechanical actuators to preserve process stability. Process controllers are usually developed using untrusted components and third-party IP cores and consequently are vulnerable to cyber threats raised by lack of trust in the internal components. Process control systems are widely used in infrastructure and safety-critical applications where successful cyber attacks can result in catastrophic disasters. Because of their connection to national security, process control systems are exposed to a growing number of attacks.

Preventing malware infiltration has not succeeded because of the complexity of modern networked control systems having zero-day exploits. Trojans may also arise from the global supply chain and use of third party IP. This leads to a demonstrated possibility of controllers being surreptitiously compromised. Erroneous controller behavior must be detected before it critically affects the physical process. Existing approaches to run-time bug and fault detection include monitoring the process state arising from past controller actions, or comparing present outputs from independent controllers. The large number of successful attacks targeting process control systems indicates the need for proactive security defenses. The most well-known incident exemplifying cyber attacks against process control systems is the Stuxnet attack targeting an Iranian nuclear power plant.

We provide an exploratory application of the RETC-CPS protection scheme to process control systems. Our approach does not distinguish between bugs, faults, and malware, and the run-time system includes a second instance of the active controller connected to a model of the plant giving a short-term projection of future controller actions and process state. The model's state is periodically synchronized with the plant's state to prevent divergence. Guard components enforcing

permissible operation rules are integrated to the predictive system and process controller interfaces. Erroneous controller behavior is detected before it affects the physical process, allowing preemptive alarms or actions. Trust is required in only a small set of simple, self-contained, and verifiable guard components, and the inserted guards are synthesized from the process models and specifications. An aircraft pitch control system illustrates these ideas. Complementary to conventional security and trust schemes, our's serves as a last line of defense against cyber threats to process controllers.

1.3 Thesis Organization

The essence of this work is as follows: Design-time security solutions are necessary but not sufficient to completely protect security-critical CPSes containing untrusted components. Complementary trustworthy, extensible, threat-tolerant, efficient, and application-specific run-time protections should be adopted to serve as a last line of defense against cyber threats evading detection by design-time solutions. The remaining chapters are organized as follows:

Chapter 2,"Background"

The selected applications and previous security work related to the research presented herein are discussed in this chapter.. Characteristics, vulnerabilities, and CPS threats are surveyed. An introduction to configurable hardware and existing security approaches to control reconfiguration in dynamic reconfigurable platforms is presented. This chapter contains an overview of run-time trusted computing in reconfigurable hardware, with a special treatment of HTHs as emerging threats to computing platforms, and distinguishing our protections from other works.

Chapter 3, "Concepts and Overview"

The requirements, assumptions, limitations, and expectations of the proposed protection scheme are discussed in this chapter. RETC-CPS guard development follows a systematic, straightforward approach which enables design automation of the RETC security anchors. Although we will not cover design automation of RETC guards, we present a complete design flow of the RETC-CPS

components developed in reconfigurable hardware. This chapter also introduces the RETC-CPS high-level architecture and basic building blocks.

Chapter 4, "Application to Cognitive Radio Platforms"

In this chapter, the basic concepts and potentials of CR are described followed by discussing security challenges preventing wide deployment of CR technology in modern cellular systems. RETC-CPS is used to enforce secure reconfiguration in a CR platform. The reconfiguration control architecture provided in this chapter is general enough to apply to different CPSes other than CR. The CHARE-CR implementation is presented and evaluated in terms of the achieved security, induced performance degradation, and overheads. This implementation is based on commercially available Virtex-5 devices, and uses the standard Xilinx design flow with some extensions to automate integrating trust anchors. This work was also presented in the Field Programmable Logic and Applications conference (FPL) 2011 as a full paper [57].

Chapter 5, "Application to Untrusted Hardware Blocks"

An application of RETC-CPS to detect HTHs in untrusted IP cores is presented in this chapter. A high-speed serial link adapter used in network backbones is advanced as an example of a complex, untrusted module widely deployed in cyber networks. We discuss low-level guards to enforce IP functional specifications. Particularly, we develop interface guards for IP cores with high-speed interactions or complex behaviors. To evaluate the effectiveness of these guards, we develop novel side-channel attacks exploiting loose specifications in the link-layer protocols of ubiquitous MGTs. The RETC protection scheme is evaluated in terms of security, overheads, and performance. This work was presented in Hardware-Oriented Security and Trust conference (HOST) 2012 as a full paper [58].

Chapter 6, "Exploratory Application to Process Control Systems"

An exploratory application of RETC-CPS to process control systems is provided in this chapter. An aircraft pitch controller is selected as an example of a security-critical, feedback process control system. A system-level security policy defining the permissible system behavior is formulated from equations and models governing the system physics. RETC-CPS is applied to build a system-

level interface guard enforcing the system security policy. We exploit the unique properties of process control systems in a novel way to predict and preempt erroneous controller behavior resulting from either controller faults or cyber threats. This application illustrates how to apply RETC to CPSes developed using model-based design flows. Work in this chapter was presented as a full paper in the Security of Internet of Things conference (SecurIT) 2012 [98], and ideas and results of this chapter formed the cornerstone of a recent NSF award.

Chapter 7, "Conclusions"

Conclusions are drawn about the overall RETC-CPS protection scheme, CHARE architecture, runtime guard components as trust anchors, and RETC defenses for process control system. Future directions for automation, verification, and potential applications are discussed.

1.4 Thesis Contributions

The unique contributions of this work are the following:

- **Introducing a threat-tolerant security scheme**

  The idea of fault-tolerant systems is not new, and has been widely deployed in reliable systems. Surprisingly, this age-old idea has not been previously employed to build trusted systems containing untrusted components. RETC-CPS provides a threat-tolerant scheme addressing security concerns of CPSes containing untrusted components. The protection scheme described in this dissertation is general enough to be applied to a wide spectrum of CPSes and embedded systems, and it can be easily integrated with existing design flows, especially model-based design techniques. The protection scheme is evaluated in terms of functional correctness and the security component performance and overheads.

- **Localization of trust**

  Existing efforts to build trustworthy systems struggle to assure trust in all system components and interactions pre-deployment. Such approaches are extremely expensive in terms

of cost and time because of the modern system complexity. Moreover, they often cannot guarantee that systems are vulnerability-free. The alternative is localizing trust in a small set of system components and entrusting these components to enhance trusted computing in all system components and interactions. In the RETC protection scheme, trust is required in only a small set of simple, localized, and self-contained guard components, which can be easily verified using traditional formal verification methods, especially when built in hardware. Such an approach can significantly enhance trusted computing in CPSes containing untrusted components. Localization of trust is assessed in terms of the trust anchor complexity compared to the whole design complexity, where the hardware complexity is measured in terms of the resource usage. Trust localization reduces the verification effort.

- **Secure reconfiguration control**

  Existing approaches to secure reconfiguration in dynamic reconfigurable hardware provide solutions to ensure partial bitstream integrity. However, these efforts do not address validating reconfiguration requests issued by untrusted components. CHARE-CR addresses secure reconfiguration control in terms of both validating update requests and authenticating update contents. It also provides the required assurances about the security of the CR platform to spectrum regulators and stakeholders. Supporting prototypes are developed for the RETC trust anchors responsible for secure reconfiguration control. The prototypes are evaluated in terms of automation feasibility, functional correctness, and localization of trust. The CHARE architecture presented in this dissertation is general enough to apply to other reconfigurable applications needing secure reconfiguration control.

- **Addressing new vulnerabilities and measures**

  We investigate the development of HTHs supporting remote interaction over wired computer networks. Software-based covert communication in computer networks is an active research area investigating how Trojans can exploit vulnerabilities in the protocol stack. Development of HTHs exploiting network protocol vulnerabilities is an interesting new area to consider. In this work, we introduce novel HTHs supporting covert communication over a network

by exploiting newly discovered vulnerabilities in high-speed interface adapter IP cores. We provide measures enabled by RETC-CPS to counter such threats, and use these novel attacks to evaluate the protection scheme. The protections are evaluated in terms of functional correctness, localization of trust, and the response time compared to software defenses.

- **Predicting and preempting erroneous controller behaviors**

  Existing run-time security solutions to process control systems rely on observing either the embedded controller or the physical process and detecting violations affecting them. However, these solutions are reactive, and can only detect erroneous controller behavior after its occurrence which might allow a physical processes to become unstable before corrective action can be taken. We investigate the development of run-time protections leverage RETC-CPS to enable predicting and preempting erroneous controller behaviors before they affect the physical process stability. The prediction architecture is evaluated in terms of functional correctness, localization of trust, and response time.

- **Enabling trust extensibility in reconfigurable architectures**

  Comprehensive security is impossible, and every system has its vulnerabilities that can be later discovered and exploited. Successful zero-day attacks against various CPSes indicate that security policies and architectures need to be extended to address new vulnerabilities. Trust extensibility is commonly employed in software solutions via security updates including definitions of new threats and fixes for system vulnerabilities. However, trust extensibility in hardware architectures is very challenging because of the coupling between the security components and the system implementation. Localization of trust, separation between system implementations and trust anchors, and secure control of system reconfiguration are the key enablers of trust extensibility in hardware architectures. RETC-CPS enables trust extensibility in hardware architectures by addressing these key enablers without significantly altering the conventional design flow.

Chapter 2

Background

The problem of building trusted CPSes containing untrusted components was introduced in the previous chapter. The background and work related to this problem are presented in this chapter. An overview of CPS characteristics, vulnerabilities, and potential threats, and illustrates CPS parts addressed by the RETC-CPS protection scheme is provided in Section 2.1. In Section 2.2, an overview of reconfigurable hardware technology is given including a summary of associated threats and existing measures, with a special treatment of partial reconfigurable systems. Section 2.3 surveys trusted computing approaches in reconfigurable hardware, and compares them to the RETC-CPS approach. This chapter is summarized in Section 2.4.

2.1 CPS Overview

In order to understand the security requirements of CPSes, we need to describe their characteristics and distinguish between CPSes, traditional embedded systems, and general-purpose computing platforms. Irrespective of the domain, CPS characteristics include: intensive interaction with physical systems; heterogeneous resources and diverse capabilities; and increased network interactions. Interacting with the physical world governs the CPS behavior, commonly configured as

closed-loop feedback control systems where a small change in the physical system behavior induces an equivalent change in the cyber system behavior and vice versa. Interacting with physical systems having continuous temporal dynamics requires real-time computing performance on the cyber side. Further, the physical world is not entirely predictable, and therefore, a CPS must be robust to unexpected conditions and subsystem failures.

Interacting with the physical world is the fundamental difference between CPSes and general-purpose computing platforms, where only users and IT environments are the contributors to system changes. From the security point of view, as the interaction between the cyber and physical systems increases, the physical world becomes more susceptible to security vulnerabilities arising in the cyber system. Further, interacting with physical systems will introduce a new generation of cyber threats specifically targeting CPSes besides the traditional IT threats. These security concerns necessitate augmenting trust in CPSes and assuring their safe and trusted operation. Interacting with the physical world requires creating a TTS and deriving high-level security policies from the physical system characteristics, and protecting CPSes from the conventional IT threats.

A CPS typically incorporates various heterogeneous components, which can be treated as subsystems, including simple sensors with limited capabilities, wired and wireless communication and networking devices, and embedded computing systems. Component diversity and large system scales distinguish CPSes from traditional embedded systems, which can be viewed as subsystems of larger CPSes. Despite the component diversity in CPSes, all components are orchestrated together to accomplish a unified objective serving the physical systems. On the other hand, general-purpose computing platforms are not only characterized by the diversity of components and interactions, but also featured by the variety of goals and objectives. This diversity of resources, interactions, and objectives in general purpose platforms makes it is very difficult, if not impossible, to build trustworthy personal computers.

A CPS is a system of systems with a tight coupling between computing and physical components. Coupling between different subsystems, which can be physically separated and distributed over long distances, is achieved through network interactions in various network domains. This differs

from traditional embedded systems commonly deployed as standalone system-on-chip (SoC) or system-on-board platforms. The CPS networking capabilities shifts malware spreading via computer networks from the cyber world to the physical world. The Stuxnet attack is an example of cyber threats targeting CPSes and propagating over computer networks. Table 2.1 summarizes differences between general-purpose computers, embedded systems, and CPSes.

Table 2.1: A comparison between general purpose computers, embedded systems, and CPSes.

	Personal computers	**Embedded systems**	**CPSes**
Interacting with the physical world	No	Yes	Yes
Diversity of components	Yes	No	Yes
Diversity of objectives	Yes	No	No
Networking capability	Yes	Limited	Yes

2.1.1 CPS Security Vulnerabilities

In the past few years, computer security has gained significant attention from the research community. Various security protocols and standards such as IPSec, SSL, WEP, and WLTS have been developed to secure communication. While such protocols and standards address security considerations from a functional perspective, several factors shift these concerns to implementation and architectural perspectives. For example, although a functional security protocol can theoretically protect the privacy and confidentiality of data, it cannot protect the underlying implementation against particular threats exploiting certain architectural flaws. Among these factors are increasing number of successful attacks such as software, hardware, and side-channel attacks against cyber systems. Power and computation constraints associated with the limited processing capabilities of embedded platforms result in undesirable tradeoffs between security and other substantial metrics such as cost, overhead, and performance [90].

Attacks against a CPS exploit vulnerabilities of a particular system implementation to achieve their goals, rather than attempting to break cryptographic algorithms and protocols. A security vulnerability is a flaw, unintentionally developed or deliberately included in a system, which could later be exploited to cause a loss of system confidentiality, integrity, or availability. A general work flow can be used to illustrate various CPS security vulnerabilities. This work flow is based on the general closed-loop feedback control flow and can be categorized into four main steps [170]:

1. **Monitoring and sensing:** Monitoring of the physical process is a fundamental CPS function where a set of sensors are used to gather continuous information about the physical system and send it to the computational elements and controllers of the cyber system. A sensor is an electromechanical transducer which measures certain attributes of the physical system such as speed, temperature, and pressure and converts it to an electrical signal. Closed-loop control systems mainly rely on sensory information to compute the feedback decisions. Sensor faults are common to all physical systems [145], and many fault-tolerant techniques and algorithms have been developed to cope with them [152]. Nevertheless, cyber attacks can target sensors to compromise sensory information. Clever attacks can evade detection using current fault-tolerant approaches. Hence, these new vulnerabilities need to be identified and countered using more innovative, application-specific defenses incorporating system physics. Vulnerabilities related to sensing are not the focus of this dissertation.

2. **Communication and networking:** A typical CPS is composed of several sensors and embedded controllers interacting with each other. Networks are often used to exchange real-time data between sensors, computing subsystems, and actuators. Communication between different CPS elements is vulnerable to various computer network threats such as eavesdropping, DoS, data modification, man-in-the-middle, and many other attacks. Existing practices and solutions addressing the security of communication channels in traditional computer networks can be utilized to protect CPSes against these threats [159]. Encryption, authentication, and authorization constitute the essence of information and communication security. Numerous research efforts have already addressed secure communication in differ-

ent network types and topologies. In this work, state-of-the-art standards and practices are employed to secure communication between CPS elements.

3. **Processing and computing:** In this step, a number of embedded controllers process sensing data and compute feedback decisions. An embedded controller is a computational platform incorporating a mixture of software-based and hardware-based processing devices, storage elements, I/O peripherals, and communication devices interacting together. Processing devices include simple micro-controllers, single and multi-core processors, digital signal processor (DSPs), ASICs, and FPGAs. CPSes are not only exposed to all known vulnerabilities of embedded systems and personal computers, but also are susceptible to zero-day attacks and cyber threats specifically targeting CPSes such as the Stuxnet worm, which infected a large number of personal computers hosting the Windows OS, but only affected those controlling the targeted nuclear power plant.

   Most computational platform vulnerabilities result from lack of trust in the underlying system components and their interactions. The resulting threats are magnified by interacting with the physical world where successful attacks can endanger human lives. Lack of trust in the incorporated components is caused by several factors, including verification difficulties and using imported COTS components and third-party IP cores. Unfortunately, these factors are promoted throughout modern electronic design flows because they significantly reduce design costs and development time. In this dissertation, we provide a run-time protection scheme tolerating inevitable vulnerabilities in embedded controllers incorporating untrusted software and hardware components.

4. **Actuation:** In this step, feedback decisions issued by the computing elements are executed using electromechanical transducers called actuators. Most vulnerabilities associated with this step are related to the physical part of the system. However, this does not imply that the actuators are not exposed to cyber threats exploiting actuators vulnerabilities. Several fault-tolerant techniques have been developed to handle both sensors and actuator faults in control systems [117, 143]. Threats associated with actuation are not the focus of this dissertation.

2.1.2 Vulnerabilities Associated with CPSes Containing Untrusted Components

The reasons why untrusted components are used in CPSes, and associated potential vulnerabilities are discussed in this subsection. Established design-for-security approaches seek to avoid threats by developing trusted application-specific solutions according to rigorous security mechanisms such as physical separation, information flow analysis, and formal verification methods. However, these techniques are applied less frequently in the modern design flow for several reasons, including high development costs and significant verification time [104]. Recently, the security community has started realizing that threat avoidance techniques are insufficient and impractical to build trustworthy complex systems. Such techniques can only reduce the number of system vulnerabilities rather than eliminate all of them. A similar lesson was learned in the reliability community's adoption of fault tolerance as a complement to fault prevention.

Most software and hardware components employed in modern embedded systems are imported from various sources, and cannot be treated as either certified or trusted. This introduces additional concerns about deliberate malicious inclusions or alterations to the component, besides inadvertent development flaws. Third-party IP cores and COTS components can contain logic bombs, worms, viruses, Trojans, and trap doors allowing execution of illegitimate actions violating system operating and security policies with malicious objectives. Such deliberate vulnerabilities can be included to the component at any stage of its production flow, including development, fabrication and manufacturing of hardware components, or delivering and updating of software programs. Generally, such inclusions or modifications are very small and extremely hard to detect in contemporary hardware components comprising millions of transistors or software programs containing millions of lines of code [155].

Security threats associated with imported components can arise in different phases of the system development and operation flow, including component design, procurement, integration, employment, maintenance, and updating [104]. For example, delivering a hardware IP core through an insecure channel makes it susceptible to tampering and manipulation by malicious parties. Further-

more, homogeneity and transparency promoted by the mass production of COTS components incur additional serious concerns about the security of CPSes constructed using these components [11]. Homogeneity means the deployment of a component in several identical systems, and even adopting several instances of the component in the same system. Transparency indicates the availability of implementation details to all parties, including malicious ones.

2.1.3 Attackers Classification

Basic security objectives of CPSes as well as embedded systems are confidentiality, integrity, availability, and authenticity [30], which are also the main goals of information security in general. Confidentiality refers to preventing information disclosure to unauthorized parties eavesdropping on communication channels. Integrity refers to trust in data or resources, and lack of it can lead to deception with compromised sensing data, for example. Availability is the capability of a system or component to be accessed and respond on demand, and lack of it results in DoS. Availability is of special importance to CPSes, since DoS can eventually lead to serious damage to the controlled physical systems. Authenticity is the system's ability to ensure that data and transactions are genuine. In CPSes, authenticity aims to ensure the correctness of monitoring data sent by sensors, and feedback information issued by computing elements and received by actuators [170]. Cyber threats usually target one or more of these objectives.

Cyber attacks against CPSes can be launched by different entities with a range of skills and resources. Modern attackers are characterized by increased skills, innovative thinking, and availability of resources. CPS attackers can be categorized into four categories, based on their motivations and objectives: cyber criminals and skilled hackers; disgruntled employees; terrorists and organized criminal groups; and nation-states [32]. The first category of attackers seeks for fame or money by attacking different computer systems, and their attacks may cause serious side effects to CPSes. Most of these non-targeted threats can be thwarted using traditional IT security solutions, and attacks launched by this category are of limited effect.

A significant number of successful attacks targeting different CPSes are launched by cyber criminals and skilled attackers. For example, a classic computer-based targeted attack has been launched against the Maroochy Shire Council's control system in Queensland, Australia in 2000 [157]. A CIA report reveals that hackers penetrated power systems in several countries and caused power outages in several cities [126]. The U.S. Federal Aviation Administration has been hacked several times [94]. Even medical devices implanted into humans have been hacked [95].

Currently disgruntled employees are the main source of attacks against industrial control systems [32]. They make use of their knowledge and access to system internals to inflict damages to the controlled physical system. Targeted attacks launched by system operators are still of limited effect because acting alone limits the consequences of such threats, despite internal access to the system. Several examples and incidents of attacks launched by this category are cited by [31].

Recently, cyber attacks have become the preferred choice for terrorists, organized criminal groups, and even nation-states because they are cheaper, less risky, not constrained by distance, and easier to replicate and coordinate. This category of attacker is more dangerous than the previous two categories because more powerful cyber attacks can be launched against critical infrastructure and safety-critical CPSes to inflict economic and strategic damage without resorting to armed intervention. Such attacks are extremely serious because of the resources, organization, and motivation of such groups. Innovative and non-traditional protections should be developed to protect CPSes against organized cyber attacks to avoid the potential destruction.

Many cases have been reported about organized cyber attacks against critical infrastructure, transportation systems, process control systems, and other CPSes [60, 136, 142]. However, no other attack has demonstrated CPS vulnerabilities and the inadequacy of existing security techniques as much as Stuxnet. It is described as the real start of cyber warfare, with speculation that Israel and the United States may have been involved [37]. The Stuxnet attack has announced that there is a certain group with the skills, resources, and motivation to launch sophisticated attacks against CPSes.

The Stuxnet worm infects Windows computers, spreads via networks and removable storage de-

vices, and exploits four zero-day attacks to accomplish its mission [36]. Antivirus software missed the attack because programmable logic controller (PLC) rootkits hide Stuxnet modifications to the system, and two stolen certificates validate new drivers. The goal of Stuxnet was to sabotage a specific physical system by reprogramming embedded controllers to operate outside their nominal bounds by intercepting routines that read, write and locate PLC commands and data. The compromised PLCs caused periodic variations in the uranium-enrichment centrifuge rotor speed, destroying them over time. Many security companies stated that Stuxnet was the most sophisticated attack they have ever analyzed [31], and it was estimated to have infected 50,000–100,000 computers. The primary target of the Stuxnet is believed to be the Bushehr nuclear plant in Iran, and it likely caused a 15% drop in the production of highly enriched uranium [35], and damage to more than 1000 centrifuges in a few months compared to the normal rate of 800 per year [62].

2.1.4 Cyber Attacks Types and Objectives

Figure 2.1 illustrates the general CPS architecture and potential threats. Attacks exemplified by A1, A2, and A3 are direct attacks against the physical system, sensors, and actuators, respectively. Such attacks cannot be mitigated by traditional cyber security measures, and significant physical measures must be adopted to deter and prevent them. Nevertheless, A2 and A3 can take the form of cyber attacks against sensors and actuators [122]. The goal of such attacks is compromising sensor and actuator data to conduct deception and DoS attacks. Therefore, rigorous security measures must be deployed to protect these components against such threats.

The A4, A5, and A6 attacks against a CPS have one of the following objectives: erroneous system computations; DoS; eavesdropping on the system; leaking secret information by compromising security keys; or some combination. A4 refers to all possible forms of cyber attacks against computer networks, where the communication channel is exposed to man-in-the-middle threats. Thwarting such attacks requires adopting adequate mechanisms for communication security such as encryption and authentication, and eliminating plausible covert communication channels. A covert channel can be defined as “an enforced, illicit signaling channel that allows a user to sur-

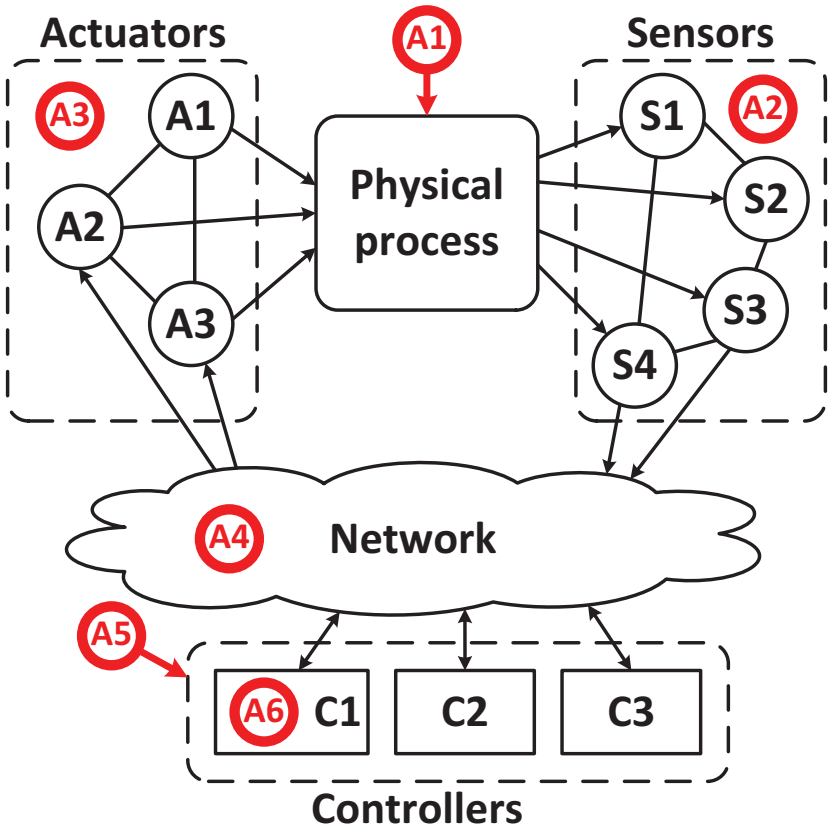


Figure 2.1: CPS Attacks.

reptitiously contravene the security policy and unobservability requirements of the system" [168]. Oftentimes, covert channels are eliminated by identifying them and preventing their creation at the communication endpoints, which will be discussed in Chapter 5 and illustrated by a detailed case study. Nonetheless, network security is not the main focus of this work, and we rely on the standard practices and techniques to secure communication in CPSes.

A5 and A6 demonstrate both external and internal or insider attacks against computing devices or controllers. External attacks or intrusions can be launched by a malicious entity, such as system operators, having physical access to the controller which has some advantages over network attacks such as the ability to acquire side-channel information. Embedded controller peripherals are compromised to gain access to the controller and exploit its vulnerabilities. For example, a sys-

tem operator having certain access rights can deceive an industrial control system by deliberately entering misleading operational instructions. Many external attacks against embedded controllers can be launched without human intervention to circumvent the controller's legitimate operation or leak classified information. Most external threats targeting a CPS exploit existing vulnerabilities of the system and the inadequacy of perimeter security defenses.

A computation platform is composed of several layers of abstraction, including application software, middleware, OS and device drivers, and the hardware or physical layer. The hardware abstraction layer may incorporate processors, memory, and I/O peripherals where each one of these components is associated with a hierarchical computation model and a specific implementation. Although this view of a computation platform reduces the complexity of the engineering design problem, it increases security concerns about potential vulnerabilities associated with each layer of abstraction.

External threats targeting CPS computation elements and controllers include software and hardware attack. Previously, cyber attacks have targeted the software stack of computing platforms by exploiting various software vulnerabilities such as buffer overflow and format string to execute the attacker's malicious code [135]. Recently, hardware attacks have increased because of their ability to evade detection by the top layers security defenses, and the power gained by controlling the most privileged layer in the computational platform.

A serious threat to CPS embedded controllers results from the incorporation of untrusted elements and components. One example of internal threats is a Trojan horse — a malicious inclusion or alteration to the system to perform certain actions and functionalities not captured by the design specifications. Trojans execute a predefined malicious operation, and are deliberately developed and included into a system by an entity participating in the system production flow. Thus, Trojans are serious threats to the CPS embedded controllers as they aim to disrupt the physical system. On the other hand, other system vulnerabilities can be inadvertently created by developers which can be later discovered and exploited.

A Trojan rely on system internals to conduct its functionality without the need for external acti-

vating triggers or stimuli. Often times, Trojans can evade detection by most perimeter security measures such as firewalls and intrusion-detection techniques which are primarily developed to counter attacks from external sources. Trojan effects on the host platform can be partitioned into: modify the functionality of the target device; modify device specifications and parameters to reduce reliability; leak information; and DoS [166]. Trojans can be classified based on their layer of inclusion into HTHs embedded in the hardware components, and software Trojans associated with any layer of the software stack. Recently, HTHs have gained an increasing interest from the security research community because of their potential to conduct more powerful attacks without being detected using software security solutions.

2.2 An Overview of Reconfigurable Hardware

Reconfigurable hardware has gained a growing interest in the last decade primarily due to three factors. First, IC density has increased significantly due to the successive reductions in transistor feature size. This growth has occurred not only in ASICs but also in reconfigurable hardware, including configurable logic components as well as routing resources [171]. Second, the cost of ASICs has considerably increased as the feature size has shrunk. This growth in development expenses has led the way to the wide deployment of reconfigurable hardware for lower volume ICs because of lower non-recurring costs. Further, reconfigurable hardware provides an ideal platform for prototype development in terms of cost, time, and the associated development and test efforts. Third, most contemporary high-performance applications requires increasing flexibility unavailable in ASICs. The required flexibility is expected to increase for next generation systems and applications. Software platforms can provide the flexibility but at the expense of reduced performance and increased power consumption. These factors have combined to increase the reliance on reconfigurable hardware in modern embedded systems and CPSes.

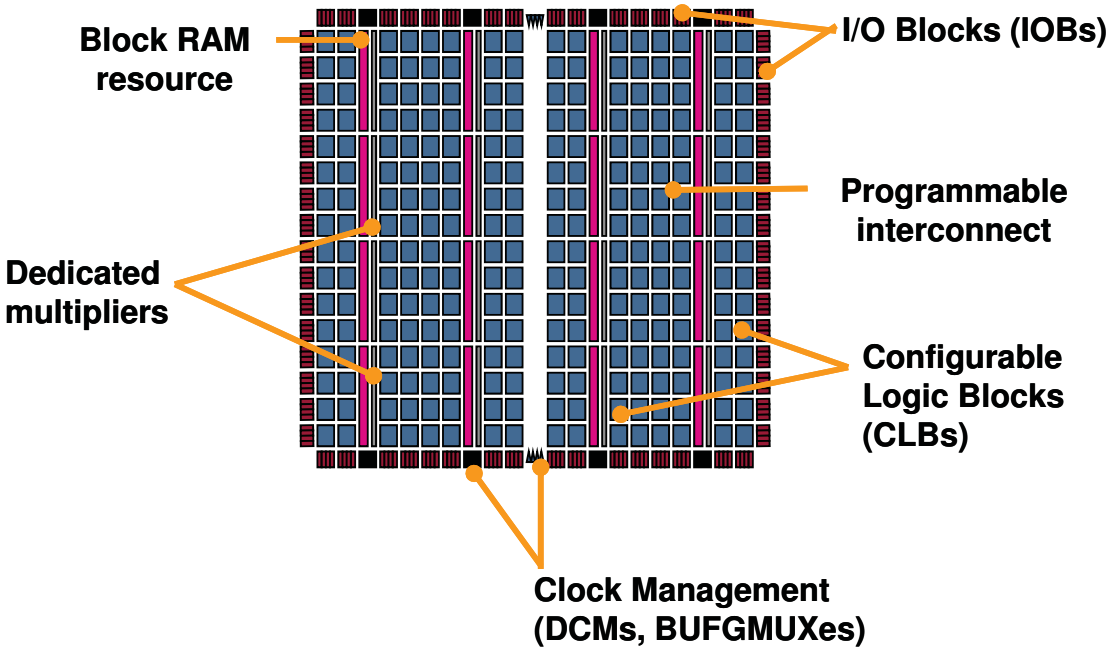
Configurable hardware platforms are programmable logic devices originally manufactured with no particular logic function. There are many types of configurable platforms such as complex

programmable logic devices (CPLDs) and FPGAs, the latter being the dominant platform for reconfigurable hardware design. FPGAs provide an array of logic structures that can be configured in the field to perform specific logic functions dictated by the developer. Most FPGAs are static random-access memory (SRAM)-based, though there are other programmable technologies such as flash and anti-fuse. SRAM-based devices do not alter the physical structure of the device and can be reconfigured by changing the SRAM memory contents.

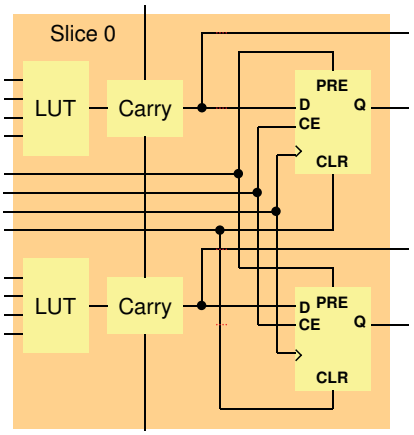
Figure 2.2 illustrates the basic architecture and building blocks of most contemporary FPGAs. A typical FPGA is composed of an array of configurable logic blocks (CLBs), several I/O blocks (IOBs), programmable interconnect, clock management resources, and block random-access memory modules (Block RAMs) as shown in Figure 2.2(a). The CLB incorporates several look-up tables (LUTs) which can be programmed to implement combinational logic, and D flip-flops supporting sequential and basic memory elements as illustrated in Figure 2.2(b). The IOB contains a set of logic cells supporting various I/O standards that permit interfacing to diverse systems and components as demonstrated in Figure 2.2(c). Programmable interconnect is used to customize connections between different CLBs and other components within the FPGA according to the required logic function and routing details. Clock management resources such as phase-locked loops (PLLs) and delay-locked loops (DLLs) are used to generate and manage clock signal delays according to the design requirements. Block RAMs depicted in Figure 2.2(d) are high-speed on-chip dual-port memory modules which can be utilized in applications requiring small yet fast memory. Many FPGAs also incorporate powerful computational elements such as DSPs and embedded hardwired microprocessors.

2.2.1 Dynamic Partial Reconfiguration

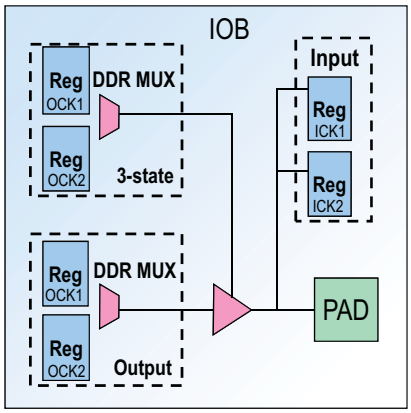
The density of reconfigurable hardware platforms such as FPGAs has grown exponentially over the last decade enabling the development of complex systems previously implemented using either ASICs or embedded processors. While FPGAs have always provided designers with the reconfiguration flexibility accommodating design revisions, this full reconfiguration capability may not



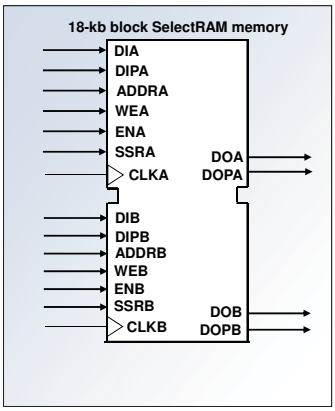
(a) Basic FPGA architecture.



(b) Configurable logic block.



(c) I/O block.



(d) Block RAM.

Figure 2.2: FPGA architecture and components.

be adequate to satisfy all system constraints. Dynamic partial reconfiguration (DPR) dramatically extends this inherent flexibility by allowing specific regions of the FPGA to be reconfigured while the rest of the device continues its operation. DPR enables new types of FPGA designs that provide efficiencies unattained by conventional design techniques, and addresses three objectives: reducing cost; allowing run-time adaptation; and reducing power consumption. Partial reconfiguration is

suitable for designs with many subsystems that do not operate simultaneously such as CR and SDR, and for applications requiring resource sharing and online modification such as high-performance computing and real-time systems.

In SRAM-based FPGAs, all user-programmable features are controlled by a volatile memory that is configured on power-up. The configuration memory is programmed using a bitstream containing all data for programmable components and interconnections. In Xilinx FPGAs, for example, devices can be programmed remotely via external ports such as SelectMAP or self-programmed via the internal configuration access port (ICAP). In partial reconfiguration, a configuration port loads partial bitstreams to the configuration memory to modify certain regions while the rest of the FPGA remains fully active [54]. The ICAP is typically controlled by an embedded processor with access to storage for partial bitstreams. Figure 2.3 depicts two methods of delivering partial bitstreams via external and internal configuration ports.

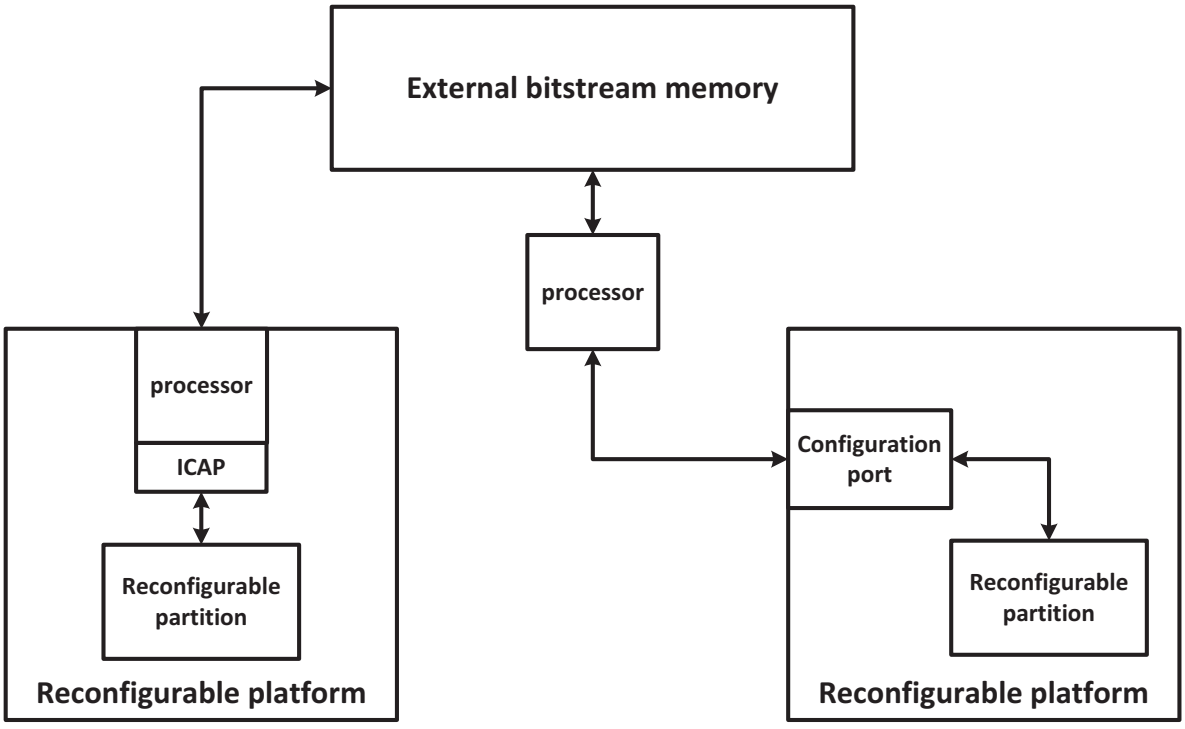


Figure 2.3: Methods of delivering a partial bitstream

2.2.2 Reconfigurable Hardware Threats and Measures

Partial reconfiguration can dramatically enhance FPGA functionality because of the flexibility provided. However, it also introduces new security concerns in terms of validating reconfiguration bitstreams and authorizing reconfiguration requests issued by untrusted components. Such vulnerabilities are especially important in CPSes because of their interaction with the physical world. These threats obviously appear in reconfigurable applications such as CR where compromised update requests issued by untrusted components can directly affect spectrum integrity. Consequently, secure reconfiguration control is a critical aspect of CR technology that can be widely applied for other CPS applications employing dynamic reconfigurable hardware. In this section we discuss security threats targeting reconfigurable hardware and discuss existing solutions and their impact on the secure reconfiguration control problem.

Significant research effort has been devoted to assure integrity of IP cores and manage digital rights in reconfigurable platforms [19]. IP cores can be delivered as HDL files, synthesized netlists, or configuration bitstreams. Each IP type is vulnerable to a set of security threats [169]. Threats to HDL and netlist IP include stealing and tampering by inserting malicious code into the design. Accredited certificates or golden reference models are needed to ensure design integrity. Often times, integrity assurances are very difficult to acquire for third-party IP cores.

Methods thwarting HDL IP threats include watermarking [129] and fingerprinting [92,93] to identify the design owner and even the design version. However, all these methods are passive implying they can provide ownership proof, but would not be enough to deter piracy. Active hardware metering can thwart cloning of netlist IP cores by including physically unclonable functions (PUFs) generating a device-specific signature to attach the design to a specific platform [41,91]. PUFs can be also used to generate cryptography secret keys specific to an FPGA device.

Threats to bitstream IP cores include cloning, tampering, and reverse engineering. Solutions thwarting HDL-level and synthesized netlist IP can be applied to protect bitstream IP as well. Moreover, there are techniques specifically developed to address bitstream IP threats. Most fo-

cus on functional security practices such as cryptography and authentication [50, 51, 87]. Modern FPGAs is equipped with dedicated cryptographic and authentication modules supporting several operating modes and key lengths to guarantee secure delivery of bitstream files to the intended device. Security solutions addressing IP security in reconfigurable hardware mainly focus on protecting IP against stealing and tampering, yet do not address either deliberate functional inclusions and alterations or secure control of dynamic partial reconfiguration. Furthermore, such solutions do not consider potential implementation vulnerabilities and run-time threats.

Zeineddini and Gaj developed a software tool and IP core library addressing security requirements such as encryption and authentication of partial bitstreams [176]. Current design tools can automatically generate encrypted and authenticated partial bitstreams. Modern FPGAs such as Xilinx's Virtex-6 and Virtex-7 families inherently support these functionalities by incorporating hardwired decryption and authentication modules capable of treating full as well as partial bitstreams. Such an approach can help to securely deliver partial bitstreams to the intended recipient. Nonetheless, it does not address either partial bitstreams with self-contained Trojans or validating update requests in reconfigurable applications such as CR.

Kepa *et al*. presented a secure reconfiguration controller (SeReCon) algorithm with an embedded IP core to provide protection for self-reconfigurable systems [89]. The SeReCon algorithm includes a remote trusted authority performing IP encryption and authentication, and generates a metadata description of modules as shown in Figure 2.4. The SeReCon IP module serves the role of a TPM-like platform root of trust, and includes public and private key decryption modules performing cryptography and authentication functionalities. An IP analysis module determines the structure of dynamic modules via bitstream reverse engineering, and compares the result to the associated metadata. The SeReCon module controls and limits access to the ICAP. Unfortunately, this approach requires undesirable modifications to the FPGA fabric to generate device-specific signatures and to store sensitive key materials. Bitstream formats are proprietary and family-specific requiring a different implementation of the SeReCon module for each FPGA family. In addition, this approach does not address validating update requests issued by untrusted components.

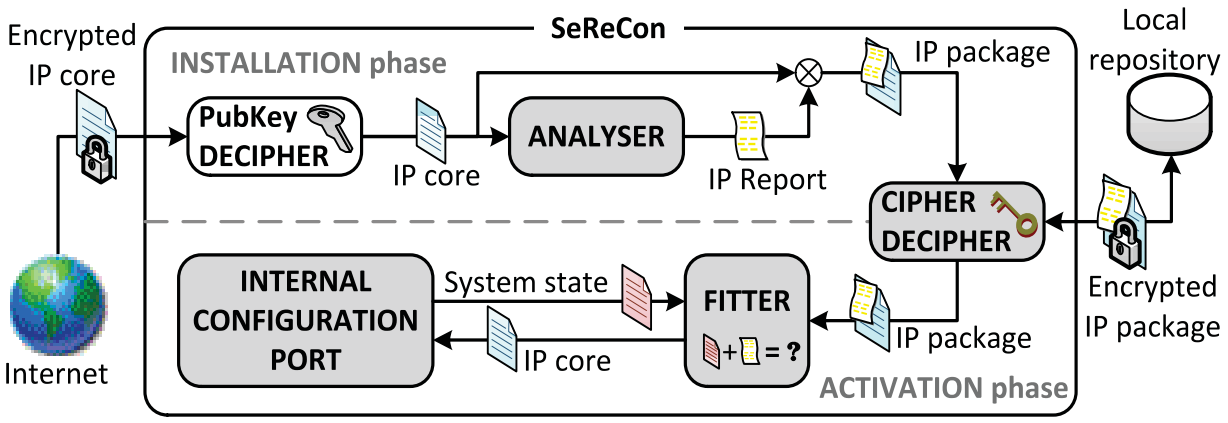


Figure 2.4: Reconfiguration algorithm implemented within the SeReCon IP [88].

2.3 A Survey of Trusted Computing in Reconfigurable Hardware

Trust is an increasingly difficult characteristic to sustain for computing systems in an era of global supply. Counterfeit electronic hardware devices are epidemic in consumer, industry, and government supply chains. Foreign outsourcing of semiconductor fabrication creates a channel for targeted attacks on a nation or consumer base. Systems are assembled from numerous software and hardware source modules, often developed by third parties. Moreover, the software tools used to design and implement these modules are themselves vulnerable to errors and insider threats. Increasingly exposed hardware configurations can be modified to violate system security policies or expected behaviors. The resulting state of system security is that once unquestionably trusted components must now be protected from both the outside world and internal threats. A system in total can be considered untrusted and insecure until novel techniques are adopted to secure the underlying components [167].

In this section we present the progression of trusted computing techniques, and survey existing run-time approaches for trusted computing in reconfigurable hardware. As illustrated by Figure 2.5, trusted computing applications used in general purpose (and later embedded) systems have had the

following progression:

1. *SLS: Software limits access to software and data.*
   Software is structured in layers, and application or user requests requiring services from a more privileged layer are vetted by software-implemented processes. The goal is to separate layers with robust APIs that cannot be circumvented. Java virtual machines and packet filters are an example of software stratification. Attacks generally have executable code masquerade as data, and use vulnerabilities such as buffer overflows to execute the data.

2. *SLH: Software limits raw hardware resource access.*
   Innermost software layers have exclusive responsibility for allocating and managing hardware resources such as CPUs, memory, and peripherals. Examples of layers interacting directly with hardware are virtual machines and operating systems. Access to hardware does not necessarily imply access to (possibly encrypted) software and data. Direct programmatic access to hardware is different from physical access to hardware, which introduces the possibility of side-channel and probing attacks.

3. *HLS: Hardware limits access to software and data.*
   Static hardware units, possibly controlled by the innermost software layer, assist in the separation of software processes and layers. Examples are memory management units (MMUs), protected execution and launch portions of Intel's trusted execution technology (TXT) [79], and the use of eTokens. Although hardware provides enforcement, trust in supervisory software may still be needed.

4. *HLH: Hardware limits raw hardware resource access.*
   Static hardware controllers are exclusively responsible for managing hardware-implemented processes or channels, and can deny requests from software at any layer. Examples are the protected input and graphics parts of Intel's TXT, Intel's Virtualization Technology [80], and hardware firewalls.

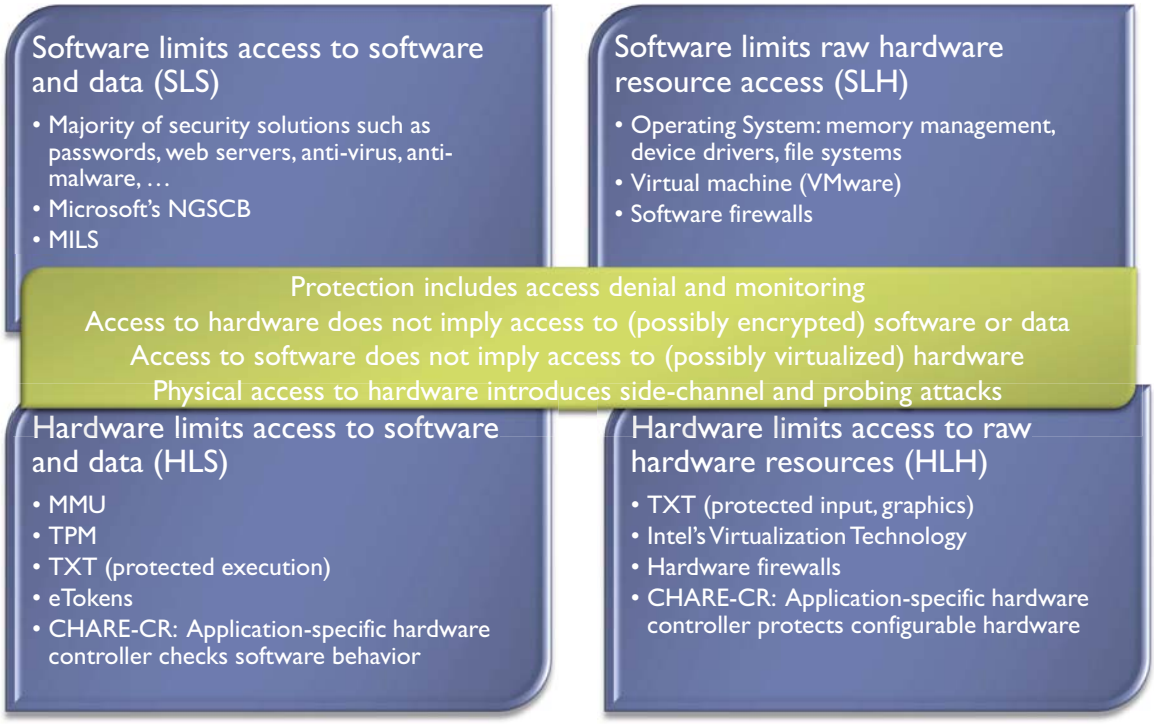


Figure 2.5: Trusted computing approaches.

Current support for secure computing uses a fixed hardware root of trust to enforce resource separation between applications. The TCG developed a trusted platform specification to provide consistent behavior for a certain purpose using software and hardware enforcement [68]. Root of trust for management (RTM), trusted platform module (TPM), and trusted software stack (TSS) components provide three basic features: protected capabilities, attestation, and integrity measurement and reporting. Microsoft's next generation secure computing base (NGSCB) is a software architecture exploiting the security provided by the TPM. The NGSCB consists of two kernels: an untrusted mode kernel, and a trusted NEXUS mode kernel that provides a secure environment for trusted code [133]. Intel's TXT uses processor enhancements, a TPM, operating system extensions such as the NGSCB, and enabled applications to protect sensitive information from software-based attacks [79]. ARM's TrustZone processor extensions support normal and secure environments, with a monitor mode providing robust context switches [15].

In all these commercial examples, the sole focus is software separation, the execution model considers neither hardware adaptability nor hardware threats, and there is excessive reliance on software correctness and integrity [148]. Furthermore, several TPM evaluation efforts demonstrate non-compliance with the TCG specifications for several TPM implementations [144]. Even the TCG specifications are criticized because of the inadequacy of the incorporated cryptographic algorithms, and the lack of the algorithm agility [67].

Reconfigurable hardware has been used to implement a TPM [55]. Modifications to existing FPGA architecture are required, including updates to the advanced encryption standard (AES) core bitstream decryptor and adding an on-chip non-volatile memory. An FPGA has been augmented with a trust block consisting of a TPM, a secure ROM storing FPGA configuration data, and switch logic used to configure the FPGA solely from secure ROM during system boot [64]. Unfortunately, these approaches still suffer from the TPM problems and limitations. These efforts do not target platforms where the hardware structure potentially changes during operation. On the other hand, many security solutions address trusted computing in embedded systems independent of the TCG specifications [16, 130, 138, 155]. Most of these solutions aim to secure program execution in embedded processors using hardware trust anchors. However, they do not address trusted computing in reconfigurable hardware.

Strict separation and isolation are the first step towards ensuring trusted module interaction in reconfigurable hardware. One method is to define minimum areas of separation to ensure independent modules on a reconfigurable system cannot interact even in the presence of a fault or radiation-induced upset. Module fences or guard bands can be created using resources that are off-limits to the design during placement and routing. Fault monitoring circuits may even be carefully added to the guard band regions without compromising the isolation between independent working regions. While this method suits systems implementing modular redundancy, a higher degree of module interaction is required to build most systems. McLean and Moore showed that fences and red/black analysis can be applied to certify a cryptographic solution on a single reconfigurable chip [113]. Bus macros with direct routing are carefully inserted to allow communication between isolated regions. However, the red / black analysis is not well suited for analyzing control flows in

systems with mixed means of reconfiguration.

Reconfigurable systems often use third party IP cores. Although ideally these cores would be verified by a trusted party, cost and source code requirements can make such a development model impractical. Reliance on off-the-shelf IP modules provided by multiple vendors with different levels of trust introduces serious security concerns [75]. Huffmire *et al*. developed a system of moats and drawbridges to provide module isolation and communication control for multiple interacting cores [76]. Moats are similar in concept to guard bands or fences in that they ensure modules do not share resources that enable communication between them. Drawbridges, which can be opened or closed, are then used to limit a module's ability to send and receive information on an interface, such as a connection to a shared bus. Drawbridges may also help to prevent modules from propagating the effects of undesired behaviors to one another. The moat and drawbridges model provides spatial module isolation and statically verifiable communication flow [74], but does not address modules with self-contained Trojan horses. Moats and drawbridges are limited in their ability to enforce broader system security policies.

Kastner *et al*. proposed run-time enforcement of information flow guarantees in reconfigurable systems with mix-trusted IP [86]. This bottom-up approach aims to build a secure computing base by verifying that interconnected IP adheres to information flow policies. Gate-level information flow tracking can be used to analyze a netlist through the addition of logic that enables each individual bit to be followed through the system. This technique may be used to look for security violations such as the information flow between untrusted and trusted modules, or timing-enabled side-channels. While this technique could be used to develop run-time information flow tracking, it does not address non-information flow attacks, such as that targeting configuration control. This bottom-up approach of building secure systems has significant overheads exceeding 100% to the gate count since typical implementations require the replication of logic gates.

Proof-carrying hardware is a new approach addressing trust in third-party IP cores. The concept originates from the proof-carrying code software solution [121] which provides a new way of assuring safety of untrusted software programs. A formal, automatically verifiable proof of safety

incorporating a set of predefined rules is generated by the code producer, combined with the code, and checked by the end user prior to the program execution. Only if the check is passed, the end user can assure the safety of the untrusted program. A number of problems later have arisen such as how to verify the code producer [13].

The concept of proof-carrying software was expanded to the hardware trust domain to address the problem of using untrusted IP cores. In the proof-carrying hardware approach, the IP core producer generates a configuration bitstream including a hardware design and a formal proof of safety. Drzevitzky and Platzner presented a proof-of-concept tool flow employing FPGA CAD tools, a satisfiability (SAT) solver, and a SAT trace checker to generate proof-carrying hardware [53]. The producer generates a combinational equivalence proof incorporating checking traces for all logic functions implemented in a bitstream IP core, and distributes this correctness proof along with the bitstream. The consumer regenerates the traces for each logic function from the design netlist, and checks it against the corresponding proof traces generated by the producer.

In the proof-carrying hardware approach, the burden of demonstrating security falls on the producer. This approach can theoretically address the security aspects of dynamic reconfiguration control and self-contained Trojans since each module carries its correctness proof [52]. Unfortunately, equivalence checking limits the expressiveness of such an approach because of the need to specify the exact Boolean functionality of the hardware design [82]. To overcome this limitation, other solutions have been proposed to provide proofs about HDL IP files rather than bitstream IP cores [108]. Such solutions require substantial changes to the design flow, and trust in the IP vendor which is not feasible for the vast majority of third-party IP cores.

Recent progress has been made to embed security-enhancing, application-specific, run-time protection circuits within a system to establish tailored trustworthy computing bases. Abramovici and Bradley proposed an application-dependent security infrastructure monitoring datapath signals for illegal behaviors. This approach adds reconfigurable Design-for-Enabling-Security (DEFENSE) logic to the functional design to implement run-time security monitors as shown by 2.6 [6]. The signals are selected by a designer directly in the RTL and grouped to create multiplexed probe

networks sourcing information to security monitors. The hardware-based monitors are configurable finite-state machines (FSMs) that check the current set of signals for behavioral properties specified by the designer. Probe and monitor configurations are controlled by a security control processor, which may also initiate countermeasures implemented by controlling specified datapath signals. When a security violation is detected, the software control processor may override signals or take actions to isolate a core. However, the authors acknowledge that broader countermeasures are required to create a system-level protection scheme. In general, it is an intractable problem to create real-time countermeasures tailored specifically to every possible attack on every system interface. Therefore, the security designer must also be given the flexibility to create abstract countermeasures or real-time enforcement practices that align with system specifications.

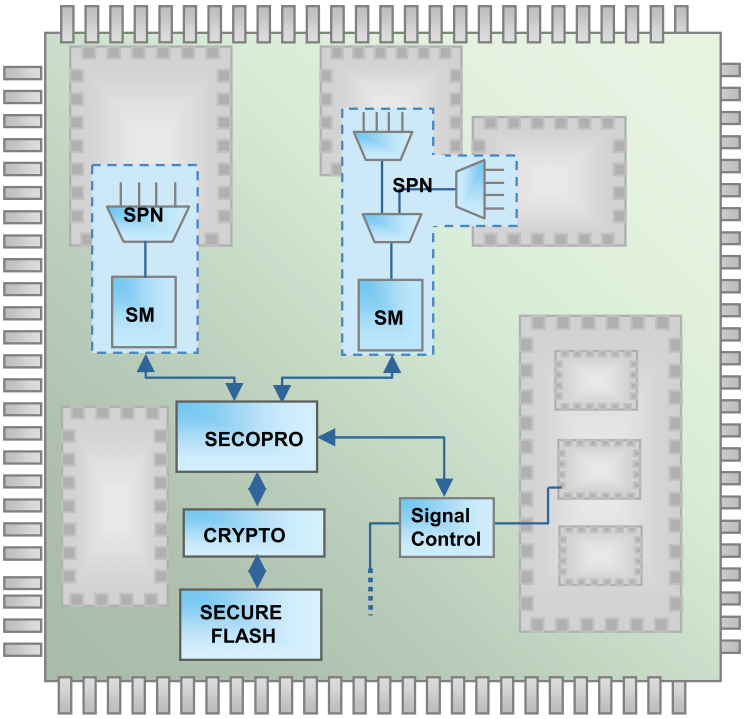


Figure 2.6: Reconfigurable platform with DEFENSE logic [6].

For security-critical applications, it is necessary to develop hardware defense measures that overcome software's verification challenges and response latency. This necessity becomes more obvious as systems are increasingly networked. Recent security practices recommend using hardware-enabled architectures to address the security requirements in modern embedded systems. As dis-

cussed in [6], checkers should be invisible to embedded software, although the proposed protection scheme uses embedded software to control the hardware configurations of the distributed defense infrastructure. The proposed system of countermeasures is also software-initiated, which limits hardware performance and is often inappropriate for attacks executing at hardware speeds. This overall approach is unnecessarily complex if trust and security are more tightly integrated with suspicious modules.

Huffmire *et al*. make use of reconfigurable hardware to implement policy-driven memory protection mechanisms [76, 78]. This work develops an access policy language that precisely describes the fine-grained memory separation of modules on an FPGA. A policy compiler converts the specified memory access policies into enforcement hardware modules. This work was further developed into a method of generating hardware-based security checkers to detect processor malicious inclusions at run-time [26]. Security-relevant invariants of a processor's architectural specification are described on corresponding circuits of the processor's design using formal assertions. Security checkers are then automatically generated as synthesizable hardware to verify linear temporal logic properties of expected behaviors.

The Huffmire *et al*. system of security checkers is further limited when used to create a general run-time protection scheme. This monitoring-oriented approach does not include integration of countermeasures or real-time enforcement within a system. Another drawback of this approach to property specification is that it requires low-level knowledge of the designed circuits. This limits the ability for more abstract, specification-level security assertions to be made without incurring additional designer translations. The use of domain-specific languages also has drawbacks of unfamiliarity to designers, and limited ability to concisely capture complex monitoring and enforcement circuit behaviors. It is unclear how this solution provides security assurances for reconfiguration control. The authors indicate that a limitation of this approach is its susceptibility to attacks that are effective before they are detected and an opportunity for countermeasures is provided.

2.4 Summary

In summary, internal threats and many external threats to CPS embedded controllers are enabled by vulnerabilities associated with most computing platforms, and insufficient security defenses. Security vulnerabilities can be deliberately included in the platform by a malicious party as Trojan horses, or unintentionally created by the system developers and exploited later. Such vulnerabilities are leveraged by lack of trust in the underlying system components and interactions between them. Exploiting system vulnerabilities leads to a set of unintended behaviors violating design specifications and the system security policy. Recently, hardware-based attacks and HTHs have started to gain interest by the security research community. Several solutions have been introduced attempting to detect various vulnerabilities pre-deployment. However, no single solution can discover all potential vulnerabilities in a computing system, and even collectively these techniques cannot guarantee that the system is free of vulnerabilities [166].

State-of-the-art run-time solutions to enhance security in CPSes and embedded systems built using reconfigurable hardware has been presented in this chapter. Configurable platforms are programmable devices characterized by both software flexibility and ASIC performance and they are widely deployed in CPSes. Most security research in reconfigurable hardware is to assure IP integrity and manage digital rights against tampering, stealing, cloning, and reverse engineering. Established solutions include watermarking, fingerprinting, encryption, and authentication of IP bitstreams. These solutions do not address either deliberate functional inclusions and alterations to IP, secure reconfiguration control, potential implementation vulnerabilities, and run-time threats. In other words, such protections do not enhance security in reconfigurable platforms containing untrusted third-party IP cores from the user's and application's perspectives although they address security concerns from the IP vendor's perspective.

We also discussed state-of-the-art security solutions to enhance trusted computing in reconfigurable platforms and embedded systems in general. The TCG has developed a set of specifications, protocols, and modules to enhance trusted computing in general-purpose and embedded platforms. Most TCG-compliant solutions aim to secure program execution in software architectures using

hardware trust anchors yet do not either address reconfigurable platform security or trusted computing in CPSes containing untrusted software and hardware, where trust is needed in all system functionalities. Some security approaches seek to enforce communication flow guarantees either by strict separation and isolation between system components, or by adopting a bottom-up approach verifying that interconnect IP adheres to information flow policies. Such approaches do not address reconfigurable platform security or trusted computing in systems containing untrusted components with self-contained Trojans, and induces significant overheads.

Other security approaches employ formal methods either pre-deployment or at run-time to verify system adherence to a set of security specifications. Although such approaches can address untrusted component threats, substantial changes are needed to the design flow and trust is required in the IP vendor which is not feasible for the vast majority of third-party IP cores. Moreover such approaches suffer from formal verification difficulties and do not address secure reconfiguration control. The most relevant work to RETC-CPS is the DEFENSE logic added to reconfigurable platforms to monitor and enforce trusted computing. The DEFENSE protection scheme is limited in terms of enforcing system-level policies and relies on software for enforcement which is often inappropriate for attacks executing at hardware speeds. Table 2.2 provides a comparison of security approaches presented in this chapter in terms of the security objectives, required architectural and design flow changes, and limitations. Table 2.3 summarizes limitations of existing run-time security approaches and illustrates how RETC-CPS addresses these limitations.

Table 2.2: A comparison between different run-time security approaches.

	Objectives	**Required changes**	**Limitations**
SeReCon	Secure reconfiguration control	Platform	Does not authenticate reconfiguration requests
TPM-based solutions	Enhancing trusted computing	Architectural	Mainly targets software-based general purpose architectures
Moats & drawbridges	Separating components and regulating communication	Architectural and floorplanning	Does not address Trojan threats
Information flow guarantees	Enforcing information flow policies	Architectural and design flow	Requires substantial flow changes and induces significant overheads
Proof-carrying hardware	Pre-deployment formal verification	Design flow	Requires trust in the IP developer
DFENESE protections	Enhancing trusted computing	Architectural	Employs software for enforcement
Security checkers system	Formal verification at runtime	Architectural and design flow	Mainly targets processor architectures

Table 2.3: Existing run-time security approach limitations and RETC enhancements

Existing protection limitations	**RETC enhancements to address run-time approach limitations**
Impact on performance and cost	Using hardware trust anchors, and localizing trust in a small set of components to reduce the verification effort
Lack of scalability and generality	Addressing trust at different levels of hierarchy, and adopting an application-specific approach
Requiring trust in global third parties	Treating untrusted modules as black boxes, and framing security policies from physical characteristics and functional specifications
Significant changes to the design flow	Separating the system and security design flows, and adopting a top-down design methodology

Chapter 3

Concepts and Overview

The main philosophy of RETC-CPS is that augmenting a computing platform with simple, verified, and distributed trust anchors can enhance system security in the context of a specific application. Trust anchors are the means to enforce application-oriented security policies at the system top-level and untrusted component interfaces. Integrating trust anchors into untrusted component interfaces enables inspecting I/O transactions, inferring component internal states, detecting component violations, and enforcing trusted computing policies. The top-level interface guard enforces a system-level security policy and validates system internal and external interactions. The concept of compositional security adopted by RETC-CPS has been previously employed in design-time verification methods such as model checking to reduce the verification complexity [114]. A theory of compositional security has been outlined in [44]. Compositional security provides the RETC-CPS security scheme with the scalability required by modern complex systems.

In RETC-CPS, reconfigurable hardware is used to develop embedded controllers as well as interface guards. There are numerous examples of FPGA deployment in CPSes including medical, industrial control systems, automotive, avionics, and aerospace applications. Reconfigurable hardware platforms can be customized to implement different computational architectures as needed by various applications. Typical reconfigurable designs adhere to the standard layered computation model. Each layer is implemented using a number of interacting hierarchical components to

accomplish a set of functionalities.

Most reconfigurable systems are built based on software architectures to realize the layered computation model. A typical software architecture consists of a microprocessor interacting with various on-chip and off-chip peripherals and storage devices. Usually, software processors are assisted by on-chip hardware components to accelerate computation according to the performance requirements and constraints. Another typical architecture adopted in CPSes such as SDR and CR applications is a supervisory processor managing and controlling a hardware datapath, yet not as a computational element. Various hardware components and processors constitute the physical layer of a computing platform which can be entrusted by adopting the RETC-CPS protection scheme.

The choice of the computation layer in which trusted computing should be enforced is a critical decision for any protection scheme. Recently, software-based security approaches have been dominant in both research and practice. However, software security mechanisms are no longer adequate to counter the growing number of hardware-based attacks and meet the increasing performance requirements. The RETC-CPS approach targets the hardware layer to add guard components to enhance trusted computing in the system under protection. Hardware guards enable formal verification, wide threat detection scope, robustness against software attacks, and controllability over all computation layers. They also address the performance requirements of modern computing systems. However, in some applications, other defenses may be still required to enable trusted computing at higher abstraction layers as well.

Interface guards are integrated into untrusted module and top-level system interfaces. Each hardware module might be composed of smaller building sub-modules organized in a hierarchical structure. Interface guards can be integrated into hardware components at different levels of hierarchy as long as the component interfaces are precisely defined. Monitoring hardware interfaces alleviates the need to comprehensive details about the component internals. Moreover, most third party developers reveal detailed information about the external operation and functional specifications of their IP yet do not provide precise details about the internal structure. This facilitates deriving security policies to be enforced at hardware interfaces.

To apply the RETC protection scheme to CPSes incorporating extensive network access, distributed subsystems can be protected individually while considering typical network security practices. The system security policy is decomposed into sub-policies enforced by distributed, trusted components. Security policies are formulated for individual embedded controllers based on their role in the whole system. A top-level network protection mechanism can be applied in conjunction with subsystem guards to enforce overall system security. Such an approach to protect CPS networks follows our main philosophy of enforcing trusted computing in embedded controllers using low-level interface guard components and a top-level supervisory guard. In this work, we will confine ourselves to address the main research topic of this dissertation which is run-time enforcement of trusted computing in individual embedded controllers deployed in a CPS.

The remainder of this chapter is organized as follows: In Section 3.1, design assumptions and target application requirements and outcomes gained by adopting the RETC-CPS security scheme are discussed. The system high-level architecture and RETC-CPS trust anchors are described in Section 3.2. RETC-CPS design flow in reconfigurable hardware is presented in Section 3.3. This chapter is summarized in Section 3.4.

3.1 Design Requirements

In order to apply the RETC protection scheme, the target system, application or block must have the following characteristics:

- Security objectives and, consequently, specifications can be precisely identified and explicitly formulated for the target application. This assumption clarifies why the RETC protection scheme targets CPSes. A major CPS feature is that all cyber system resources serve the physical system which enables clearly stated security objectives and policy specifications from the system physical properties. This assumption also illustrates why such an approach cannot be applied to enhance trusted computing in general purpose computing systems characterized by the diversity of both goals and resources. This diversity complicates framing enforceable

policies addressing all system security requirements. Nevertheless, such a policy-based approach only addresses security requirements identified by particular policy specifications.

- Systems targeted by the RETC protection scheme are built with both trusted and untrusted components. Trust is measured in terms of the provided assurances against the security requirements dictated by the application context. For example, formally verified modules can be considered trusted as long as the verification properties address the application security requirements. Trust certificates issued by trusted third parties are another means of qualifying a component as trusted. Extensively tested components in the same application class may be considered trusted. Other custom assurances can be considered as well if they address the application security needs. A component may lack such assurances but still can be treated as trusted in the application context because it is unable to contravene the system security requirements. On the other hand, critical components without sufficient assurances are treated as untrusted. The protection scheme must incorporate a top-level enforcement guard accounting for unanticipated security violations not captured by lower-level guards or arising from untrusted components.

- Untrusted modules are treated as black boxes without associated formal models or golden references. This is a very realistic assumption in contemporary security practices dictated by globalization which plays a major role in silicon technology supply. In a global supply chain, assuring component genuineness, provenance, and trustworthiness is extremely difficult, if not impossible. Furthermore, most third party developers prefer to not reveal the design models and details. Even if such models are available, there may not be trust in the model being consistent with the implementation. However, development constraints such as productivity, cost, and competition dictate the deployment of such untrusted components. In this scenario, system designers have to use the products offered by anonymous sources, without any sort of associated assurances, to build security-critical systems. Therefore, the RETC-CPS protection scheme provides an ideal approach to keep the balance between security and other design concerns. It enables the deployment of untrusted modules without

accompanying golden references, and compensates for that by adding trusted components enhancing system security.

- The system under protection is composed of a mixture of software- and hardware-based components and, consequently, vulnerable to both software- and hardware-based threats. In the layered computation model, trusted computing needs to be enforced at different layers of abstraction. Hardware trust anchors can detect a wide range of threats targeting different software and hardware abstraction layers. However, not all software attacks are detectable using hardware defenses. For instance, some software layers employ certain mechanisms such as data encryption and error correction algorithms preventing data inspection at the hardware layer. Nonetheless, in this work, we only illustrate the enforcement of security specifications at the hardware layer to thwart a certain class of software- and hardware-based threats. Cross-layer approaches to enhance trusted computing in CPSes can be investigated in future work.

- The platform is susceptible to both internal threats such as Trojan horses and external threats such as malware and viruses. This assumption acknowledges that lack of trust in system components can lead to various sorts of threats. The proposed protection scheme is not limited to a specific threat. A security violation causes anomalous, non-compliant, erroneous, or extraneous behavior that can be captured by the system specifications. Threats are found by observing their effects rather than detecting their sources. This addresses different threats regardless of their origin.

- Protected systems contain updatable software programs and possibly reconfigurable hardware blocks. This assumption admits the fact that complex modern systems require system updates and reconfigurations to meet adaptive computation requirements. System updates may contain Trojans and development bugs, and are susceptible to tampering and change pre-deployment. Consequently, the adopted security defenses should enable trust extensibility to address modifiable systems and components. Not only system updates are considered untrusted, but also update requests can be issued by untrusted software and hardware compo-

nents. Therefore, such update requests must be scrutinized against well-defined, application-specific, and rigorously enforced policies.

- We do not assume that policy enforcement will always result in typical system behavior, only that it mitigates the attack consequences. For example, assume that a certain attack can manipulate the output data of a cryptographic module and this attack can be detected somehow by the security components. A plausible countermeasure is bypassing the compromised module and switching to a redundant concurrent backup. Thus, functional duplication might be needed to enforce typical system behavior. However, not only functional duplication is needed, but also trust in the redundant function which contradicts the goal of using untrusted components without accompanying models. Therefore, RETC-CPS can only enhance trust in a computing platform yet cannot guarantee its typical operation.

- We do not assume that the RETC-CPS protection scheme is comprehensive, instead, view it as a complementary approach that can serve as a last line of defense against the rising number of inexorable threats targeting security-critical CPSes. RETC-CPS relies on detecting security violations captured by security policies tailored to a specific application and enforced by hardware guards monitoring and overriding untrusted component interfaces. In certain cases where rule enforcement is infeasible, countermeasures may simply be threat reporting. Such an approach is limited to counting specific threats addressed by the security policy rather than a holistic security approach which may be infeasible. For instance, it does not address physical attacks such as probing or side-channel attacks such as power analysis attacks unless the policy captures such threats.

3.2 RETC-CPS High-level Architecture

In this section we present the RETC-CPS architecture to enhance trusted computing in CPSes containing untrusted components and built using reconfigurable hardware. Figure 3.1 illustrates the RETC-CPS high-level architecture. The added trust anchors are: a top-level interface guard,

untrusted component interface guards, and a configuration firewall. The protected system communicates with a trusted remote dynamic module server (DMS) responsible for generating and delivering partial reconfiguration bitstreams to the system.

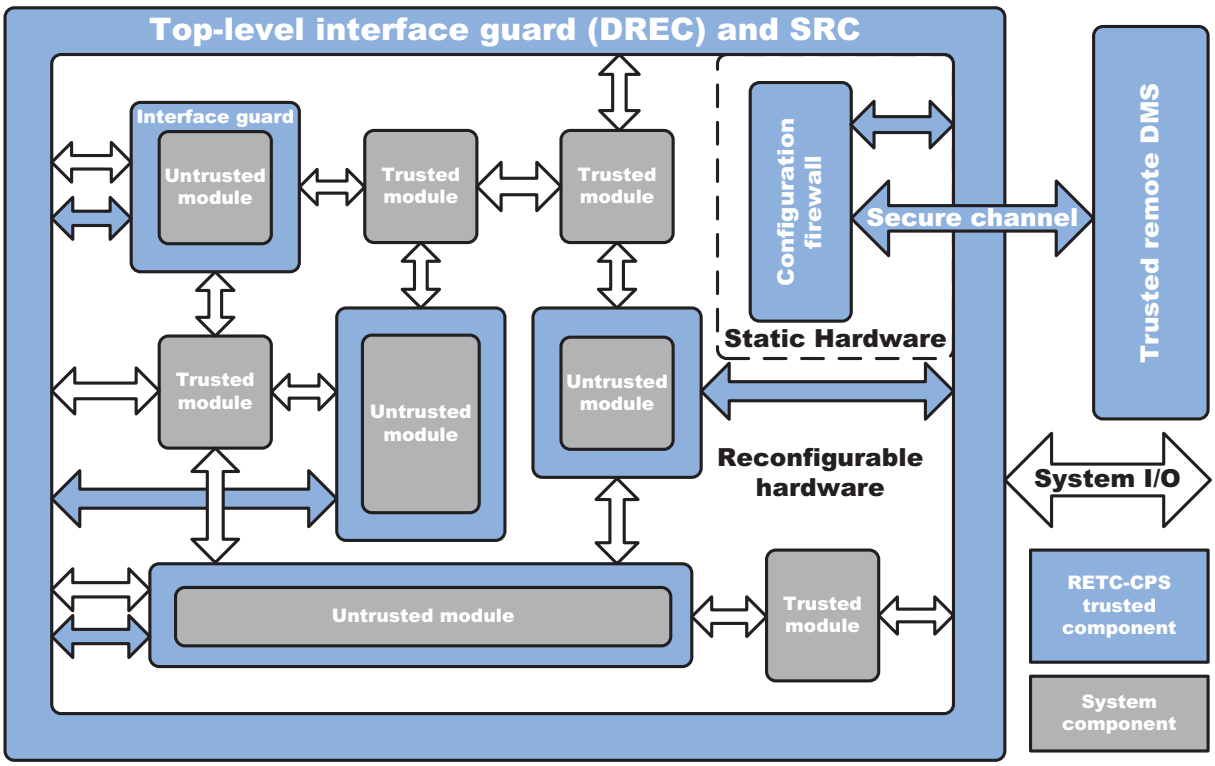


Figure 3.1: RETC-CPS High-level Architecture.

3.2.1 Interface Guards

An interface guard is a trusted hardware wrapper limiting access to an untrusted component, monitoring its I/O flow, and enforcing the permissible behavior rules. Trust in guard components is gained through trusted development and integration flow, formal verification at design-time, secure delivering and reconfiguration. Trusted development eliminates deliberate vulnerabilities while formal verification reduces unintentional design bugs. Limiting access to a component is achieved through physically separating the component resources, and enforcing the security policy at the I/O interfaces. In reconfigurable hardware, physical separation is implemented by surrounding the

component of interest by a hardware fabric isolation band in which reconfiguration and routing are forbidden. The physical separation procedure is conducted at design-time in the placement and routing phase. At run-time, configuration bitstreams can be inspected to check that isolation bands have not been exploited to disguise malicious logic such as HTHs.

Figure 3.2 shows how an interface guard wraps untrusted component interfaces. An interface guard runs concurrently with the protected component as it is implemented using separate resources, yet shares timing resources such as clock signals. The guard external interface is identical to the component interface, whereas the internal interface has reversed I/O directions such that the component inputs are the guard outputs and vice versa. System interfaces are connected to the component via the interface guard acting as a component firewall. In this way, the guard can monitor the component I/O and control signals and enforce policy rules, while the system can only interact with the guard rather than the untrusted component. This arrangement adds resource overheads in the interface guard and isolation logic, and induces timing latency in the I/O and control signals, yet does not cause a performance degradation as long as the guard timing characteristics accommodate the component operating frequency. The challenge in developing interface guards is reducing the induced overheads and latency while maintaining the security requirements.

An interface guard is a hardware component with input ports including platform inputs and component outputs, and output ports including platform outputs and component inputs. The guard monitors the platform inputs and component outputs, and possibly overrides these signals according to the policy rules. If no violation is detected, the guard inputs are simply connected to the corresponding outputs. If a violation is detected, guard actions include overriding the interface signals with safe values determined by the security policy and/or reporting the violation to the top-level guard. Such an approach monitors the component I/O interfaces to detect both external and internal violations of security policies. At the component level of hierarchy, the security policy is circuit-dependent which can consolidate I/O data flow analysis, control signal inspection, and internal state inference rules. Known examples of security policies that can be enforced at I/O interfaces are access policies and information flow guarantees. Usually, framing security policies capturing every possible threat is an intractable problem, and the alternative is generalizing and

abstracting security specifications.

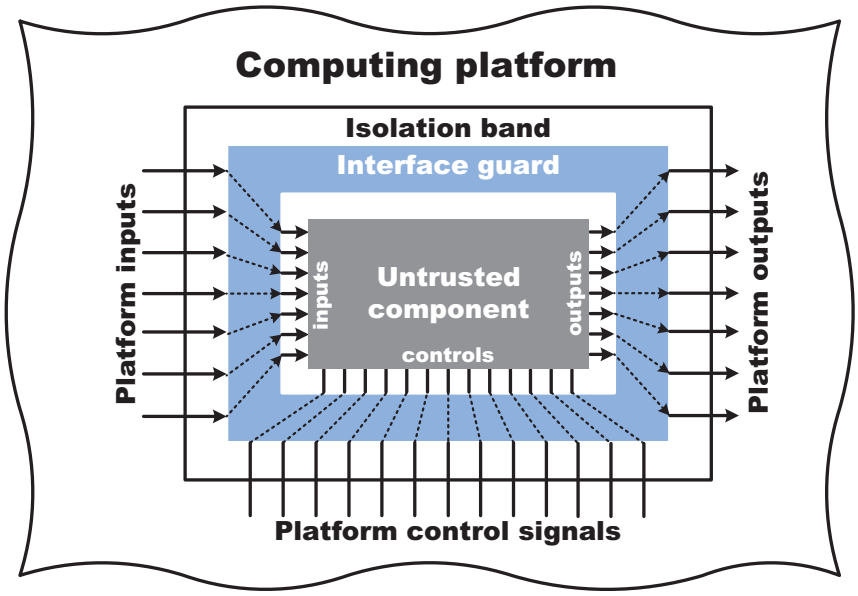


Figure 3.2: Untrusted component interface guard.

The top-level interface guard is a hardware component integrated into the system interface. In the RETC-CPS architecture, this top-level guard is called design rule enforcement controller (DREC). The DREC acts as a firewall monitoring the system I/O buses, and enforces a high-level security policy. The main distinction between the DREC and untrusted component interface guards is that the system-level security policy enforced by the DREC is application-oriented rather than circuit-dependent policies enforced by low-level guards. In CPSes, the security policy is derived from the characteristics and models of the controlled physical system. The DREC does not need an isolation band because it wraps the system top-level. Main functions of the DREC are monitoring the system I/O interfaces, interacting with low-level interface guards, inspecting suspect internal signals, and enforcing the overall system security policy.

The DREC is equipped with a secure reconfiguration controller (SRC) responsible for receiving run-time partial reconfiguration requests and evaluating them against a predefined reconfiguration policy. If a reconfiguration request is approved, the SRC initiates the reconfiguration process; otherwise an error status is returned which can include the denial justification. In CPSes, typically

a software-based processor is responsible for issuing reconfiguration requests based on perceived variations of physical process dynamics, computation requirements, and system operating conditions. A reconfiguration request can be a single module swap, multiple module change, or entire datapath replacement. As illustrated by Figure 3.1, system components are built using reconfigurable logic enabling partial run-time reconfiguration at different levels of the datapath hierarchy. The DREC issues reconfiguration commands to the SRC to swap compromised modules in response to security violations either detected by the DREC or raised by low-level interface guards. Replacement of compromised modules can serve as a countermeasure for a certain class of violations which cannot be countered with conventional measures. The SRC also can receive reconfiguration requests from the external trusted DMS to update trust anchors according to a trust extensibility framework.

Figure 3.3 illustrates the DREC reconfiguration request flowchart. The SRC receives reconfiguration requests from a software-based processor, usually untrusted and vulnerable to software attacks, and evaluates their conformance to the security specifications. If the request is not approved, the SRC denies the request and sends an error status to the requester. Approved requests are analyzed to check if they are parameter updates, single module swap, or a whole plug-in change. Parameter updates are very common in reconfigurable hardware designs because they enable a module's function to change without the need to completely replace the module. Parameter updates are performed by the DREC by controlling low-level components parameters and attributes, while approved reconfiguration requests are escalated to the configuration firewall.

3.2.2 Configuration Firewall

The configuration firewall is the entity responsible for receiving reconfiguration commands from the SRC and taking actions to fulfill them. The configuration firewall can be developed in either software or hardware depending on the reconfigurable platform's available resources. In the software design, all firewall functionalities can be implemented in a single processor using established, trusted software programs and libraries. This approach saves hardware resources needed to build

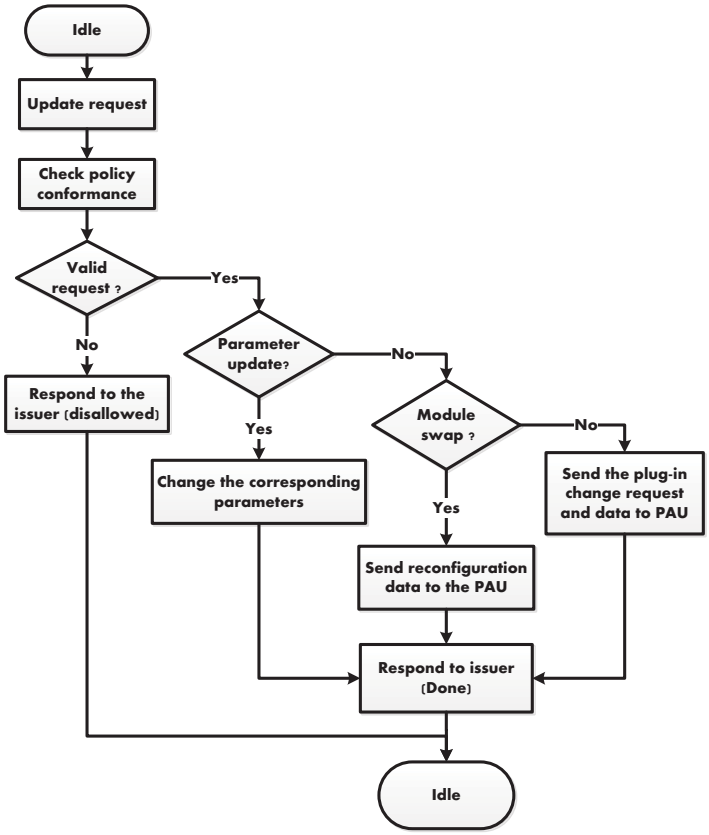


Figure 3.3: SRC reconfiguration flow.

complex modules such as Ethernet protocols. However, hardware implementation of the firewall is preferred to avoid vulnerabilities associated with software, yet it adds a significant overhead. Hardware-software co-design of the configuration firewall is a good solution to gain advantages of both. Security-critical components such as encryption and authentication modules are built in hardware, whereas high overhead components such as Ethernet protocols are developed in software. Regardless of the adopted implementation architecture, the configuration firewall is built in reconfigurable hardware as a static partition which is never changed during platform operation.

Figure 3.4 depicts the basic architecture of the mixed configuration firewall. The main components are:

- External storage: partially reconfigurable modules (PRMs) are stored on an external non-

volatile memory such as a flash memory or a hard disk drive due to the on-chip storage constraints of FPGA platforms. As off-chip memories are vulnerable to tampering at several points such as the data bus, PRMs are securely stored using standard encryption and authentication techniques. The PRM memory is divided into a number of partitions based on the reconfigurable platform floorplanning and the number of PRM alternatives included in a dynamic subset. This partitioning helps to expedite the search and fetch of a requested PRM. However, the PRM memory cannot simultaneously retain all PRM alternatives because of the embedded storage limits. This justifies the need to a remote DMS generating and delivering PRMs not available on the platform local PRM storage.

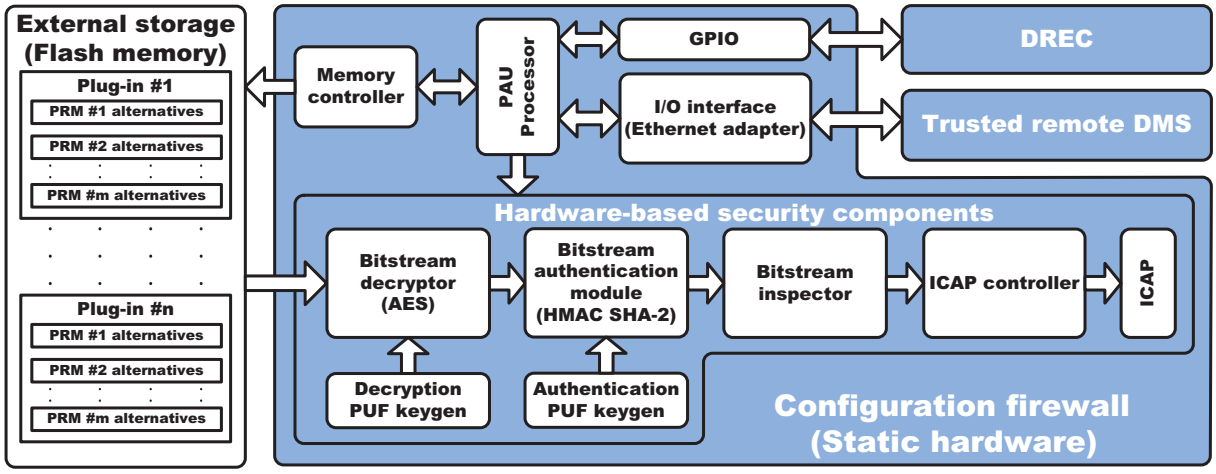


Figure 3.4: Configuration firewall architecture.

- Plug-in Assist Unit (PAU): The PAU is a software-based processor receiving reconfiguration commands from the DREC, initiating and controlling the reconfiguration flow, and communicating the reconfiguration status back to the DREC. The PAU controls an I/O interface such as an Ethernet adapter, a memory controller, and the hardware-based security modules. An Ethernet protocol stack such as lightweight internet protocol (LWIP) accommodating the embedded platform resource constraints is developed in software to support file transfer protocols (FTPs) between the DMS and the platform. The PAU interacts with the DREC via a general-purpose I/O (GPIO) peripheral. A handshaking protocol between the DREC

and PAU is developed based on sender initiated protocols and interrupt service routines. Hardware security components employed in the configuration firewall are synchronized and controlled by the PAU.

- Hardware security components: Authentication, decryption, and bitstream inspection are the main security components employed in the PAU. These components are managed and coordinated with the assistance of the PAU, yet decrypted bitstreams are not revealed to the processor. The AES algorithm is used to decrypt the PRM bitstreams loaded from the local PRM memory. The Hash-based Message Authentication Code (HMAC) is used in conjunction with the Secure Hash Algorithm (SHA-2) to authenticate the decrypted PRM bitstreams. PUF-based structures are used to generate unique cryptography and authentication keys which are known to the DMS. A bitstream inspector is used to check a meta-data file sent along with a PRM bitstream. This file defines the PRM location and interfaces to avoid the need to apply bitstream reverse engineering techniques to extract module placement information. The ICAP is controlled using a hardware module supporting various port widths. Full implementation details and results of the configuration firewall are provided in Chapter 4.

Figure 3.5 illustrates the flowchart of reconfiguration commands to the SRC. The DREC sends PRM reconfiguration commands to the PAU including the module tag and location data. The PAU searches for the PRM bitstream on the local memory storage, and if the bitstream is not found, the PAU downloads the bitstream file from the remote DMS and stores the PRM file to the appropriate location on the PRM flash memory. The PAU initiates and controls the memory reading operation via the memory controller. Retrieved bitstreams are decrypted and passed to the authentication module to be validated, and if a bitstream is not approved, an error status is sent to the DREC. Then, the bitstream inspector checks PRM locality and interfaces, and an error status is sent to the DREC if the inspection fails. If the bitstream passed all checks, the ICAP controller writes the bitstream to the reconfiguration memory while the PAU sends a completion status to the DREC.

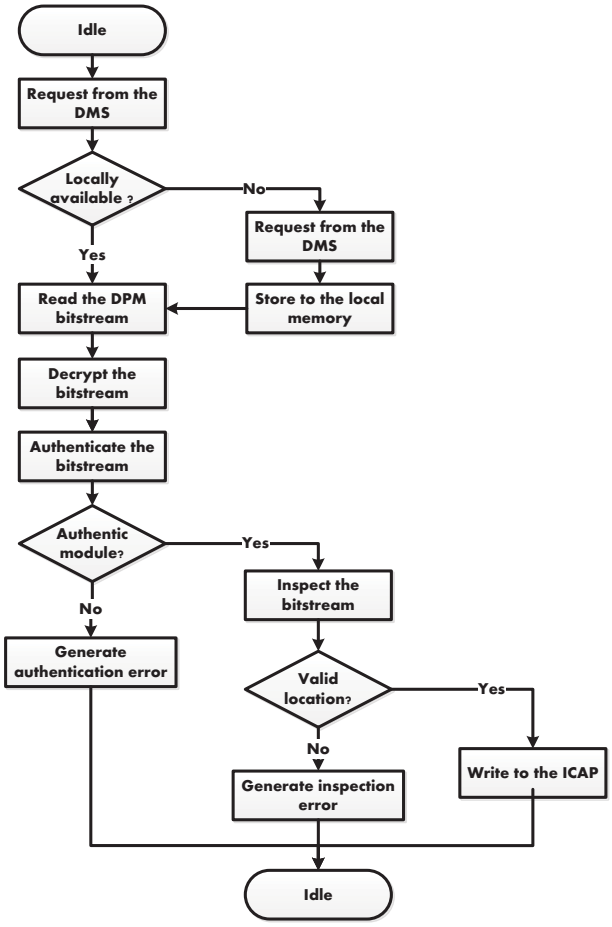


Figure 3.5: PAU reconfiguration flow.

3.2.3 Trusted Remote Dynamic Module Server

The trusted remote DMS is a powerful workstation responsible for generating PRMs on the fly and delivering them to the served platforms via a dedicated network. The DMS is equipped with the RETC-CPS development tools and FPGA design software enabling expedited generation of the PRMs. The DMS generates PRMs and transfers them to the reconfigurable platform in response to update requests. The DMS also can send PRMs incorporating updated trust anchors to extend trust in the computing platform if security violations are reported. In both cases, the PRM bitstream includes a RETC interface guard wrapping the requested module. These two parts can-

not be separated and transferred individually because of the security requirements as well as the dynamic reconfiguration flow limitations. In case of a whole system reconfiguration request, the DMS generates a hardware plug-in incorporating the new datapath and RETC-CPS guards. The DMS extracts placement data from PRMs and generates meta-data description files in a predefined format which is sent along with the module bitstream. Other functionalities performed by the DMS are encryption and authentication of PRM files.

In bitstream generation, the DMS considers on-chip storage limitations in the FPGA platform. PRM bitstream files are partitioned into equal size, small blocks to enable buffering bitstream blocks on the on-chip memory during the decryption and authentication operations. Encryption and authentication procedures are consequently executed on a block basis. The secure version of a PRM bitstream is constructed by concatenating encrypted blocks and authentication message digests in a proprietary sequence to increase the difficulty of potential bitstream IP attacks. The transferred PRM file consists of the meta-data file concatenated with the secure PRM bitstream. In the reconfigurable platform, bitstream blocks are loaded in tandem, decrypted, authenticated, inspected, and temporarily stored in the on-chip memory sourcing the ICAP.

3.3 Design Flow

Advances in semiconductor technology and successive reductions in transistor dimensions have increased the complexity and density of modern systems. Manual design techniques cannot support development of high-density circuits. Electronic design automation (EDA) enables development of such complex systems. Current directions in EDA dramatically improve productivity by integrating methods and tools for digital system design, verification, and implementation. State-of-the-art EDA technologies include methods capturing abstract system-level design specifications, synthesis and implementation tools addressing the requirements of high-speed and low power designs, and validation tools enabling high-speed simulation and formal verification methods.

Advancements in EDA technologies create new opportunities and challenges requiring effective

design methodologies. Top-down design addresses productivity, cost, design complexity, and time-to-market constraints in modern digital IC and SoC design. System-level models and verification environments accelerate digital system development by avoiding the need to deal with low-level circuit details required in bottom-up design methodologies. Automated tools synthesize high-level system descriptions into optimized logic- or transistor-level circuits. The following subsections describe the RETC-CPS design flow targeting reconfigurable hardware platforms.

3.3.1 Reconfigurable Hardware Design Flow

The typical reconfigurable hardware design flow used in FPGA development is illustrated in Figure 3.6. The main stages of this flow are as follows:

- Design specification: This stage involves analysis of the design functional and architectural requirements, system decomposition, workload partitioning, and creation of verification environments. Inter- and intra-system interfaces, timing characteristics, and design constraints are precisely defined in this step. Design tools, target platforms and architectures, and self-developed and imported components are determined according to the development budgets, goals, and time-to-market constraints. In good design practices, the output of this stage is a document describing the overall design plan, specific procedures and goals, and timelines to achieve these objectives.

- Design entry: In this stage, the design is captured and managed using software tools such as Xilinx ISE and Altera Quartus. Digital designs can be captured in HDLs such as VHDL or Verilog, schematic representations, or a combination of both. HDLs capture the design at multiple abstraction levels including behavioral, data flow, RTL, and structural descriptions. Schematic-based entry provides more visibility into the design, while HDLs offer a level of abstraction isolating the designer from low-level details. Modern HDLs such as Bluespec [124] and SystemVerilog [164] can provide software-like levels of abstraction and clean semantics. All design entry methods enable design reuse and productivity by instanti-

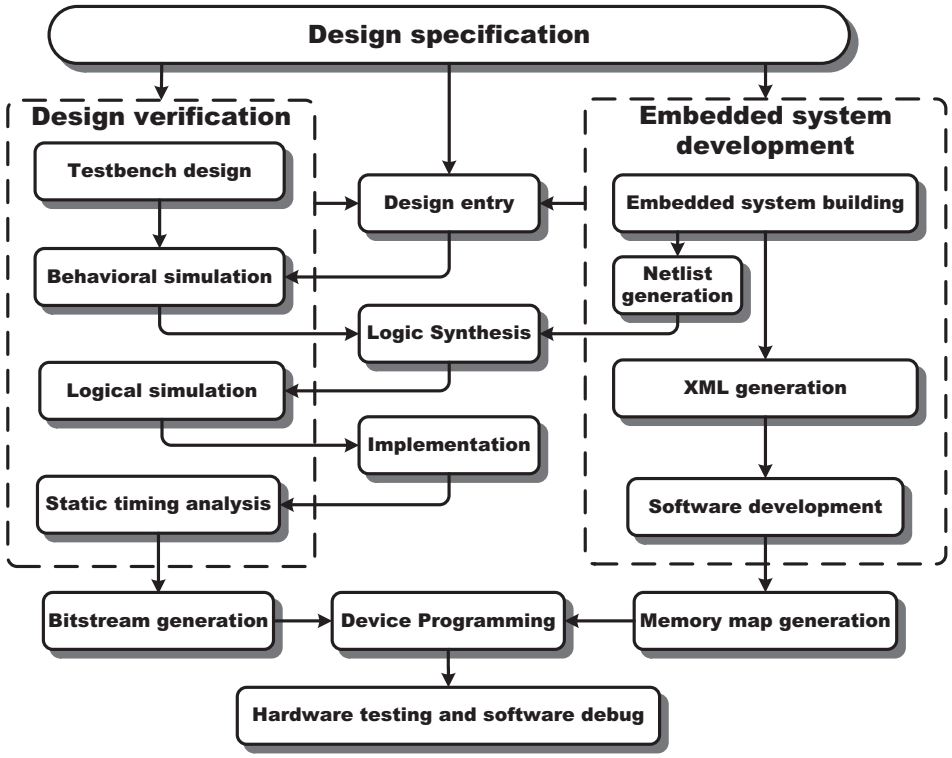


Figure 3.6: FPGA design flow.

ating already developed components and third-party IP cores.

- Logic synthesis: This stage includes translation of HDL and RTL descriptions into a netlist which is a circuit description format enumerating the design logic gates and interconnections. Synthesis checks the HDL code syntax, analyzes the design hierarchy, generates design netlists of instantiated components, and optimizes the design architecture according to the design specifications and requirements. Logic synthesis optimization objectives include minimizing occupied area, maximizing design speed, and reducing power consumption. The design netlist provides rudimentary timing information denoting logic gate delays. There are numerous synthesis tools available from different FPGA and EDA vendors.

- Implementation: This stage includes three phases: translate, map, and place and route. In the translate phase, multiple design netlists and constraints such as port assignments and timing requirements are merged into a single netlist. An automated Map tool groups the

logic blocks defined in the design netlist into corresponding FPGA physical resources. Place and route software selects the positions of resources needed on the chip according to design and placement constraints, connects them, and extracts precise timing data.

- Bitstream generation and device programming: In this step, the design bitstream is generated from the physical implementation files. The generated bitstream can be protected using specific encryption and authentication techniques. This bitstream is used to program the FPGA device using the chosen programming port.

- Design verification: Each stage of the design flow is followed by a verification procedure to assure compliance with the design specifications. Test environments and behavioral models are captured using traditional design entry methods. Timing simulation is the main verification tool in reconfigurable hardware design flows. Modern tools and flows enable assertion-based verification and temporal logic checks used in formal verification methods. Such flows are assisted by advanced HDLs such as Bluespec [123] and SystemVerilog [25], and state-of-the-art simulation and synthesis tools.

  In simulation, signals and variables are observed, procedures and functions are traced, and breakpoints are set. Behavioral simulation follows the design entry phase, and validates the functional correctness of the design as captured by HDLs or schematics. Timing delays do not appear in this functional simulation. Gate delays are known after synthesis, and routing delays are known after placement and routing. Negative verification results after any design phase require changing the design to resolve the raised problems and meet the design requirements.

- Embedded system development: Commonly, modern SoC designs rely on software architectures providing both flexibility and design productivity. Modern FPGAs are equipped with hardwired processors or are capable of hosting processor IP cores supporting software. Standard processor architectures such as PowerPC and ARM are available in FPGA platforms to enable standard embedded development. Vendor-specific design tools such as Xilinx Platform Studio (XPS) are used to define the embedded processor architecture, memory and

storage devices, I/O interfaces, and incorporated peripherals based on the architectural requirements and available platform resources. The output of this step is a netlist file defining logic resource requirements and an XML file describing the processor architecture, memory ranges, I/O interfaces and peripherals. Embedded software development is conducted using FPGA vendor-specific tools such as Xilinx SDK, and the output of this stage is a BRAM memory map (BMM) file mapping software executable binaries to the platform BRAMs. The generated bitstream and BMM files are used to program the device which is followed by the hardware testing and software debugging phases.

Run-time partial reconfiguration introduces additional steps to the conventional FPGA design flow. Figure 3.7 depicts the DPR design flow [109]. In the planning and synthesis phase, DPR opportunities are identified by recognizing mutually exclusive functionalities, and the design is partitioned into static and dynamic subsets. This part of DPR design closely maps to the traditional FPGA flow. The output of this stage is a set of HDL files describing the design top-level interfaces, static logic, and PRMs.

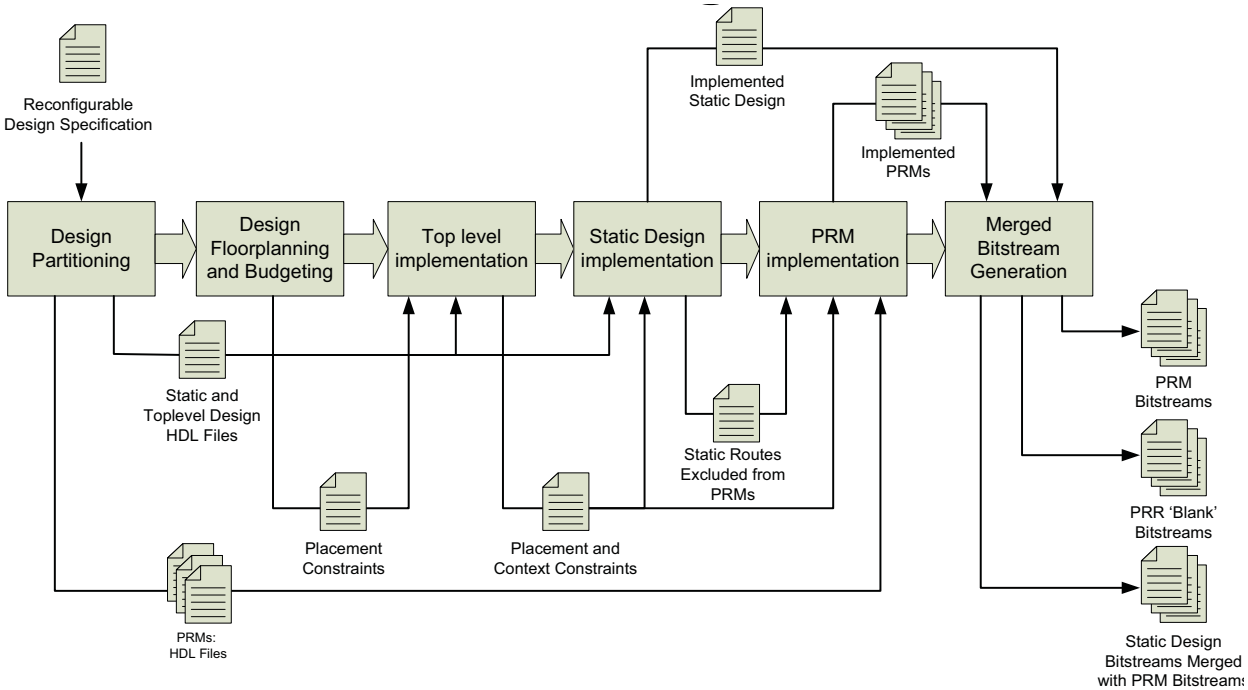


Figure 3.7: Dynamic partial reconfiguration design flow [109].

The next phase is design floorplanning and budgeting. Location and size of partially reconfigurable regions (PRRs) are allocated based on PRM needed resources and interfaces. A PRR is a physical area on the FPGA device reserved for implementing PRMs grouped in one dynamic subset. Figure 3.8 exemplifies a simplified floorplan of a dynamic reconfigurable design. Intra-chip interfaces between PRRs and the static partition must be precisely defined and unchanged throughout the DPR design flow. Design floorplanning can be done manually or automatically using specific tools such as Xilinx PlanAhead which can be employed between the synthesis and implementation stages. Partition pins are automatically inserted between PRRs and static logic to provide static routing connections minimizing any timing closure issues related to the PRR interfaces. The current implementation of partition pins employs proxy logic, which is a LUT inserted between input and output paths of the PRR.

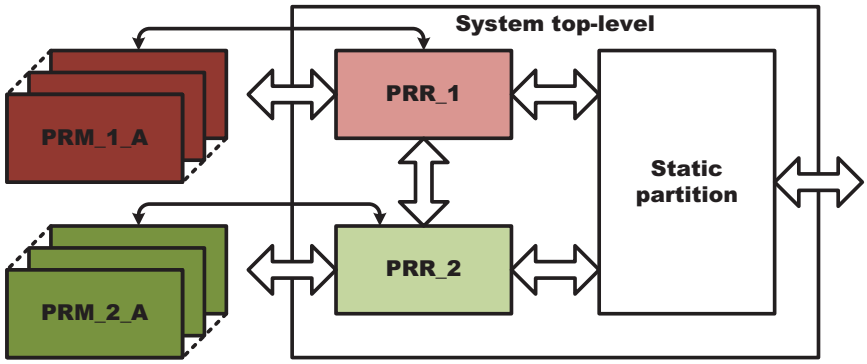


Figure 3.8: Typical DPR floorplan

Subsequent phases are top-level and static logic implementation. Top-level and static design netlists are implemented using conventional tools while considering the DPR placement constraints. Static modules are prohibited from using logic resources in PRRs, but are allowed to use routing resources in these regions. This is followed by implementing PRMs according to their placement and route constraints while excluding routing resources used by the static design. Each PRM is separately implemented from other PRMs in the same dynamic subset. In the final phase, a number of complete designs, depending on PRM combinations, are built from the static design integrated with all possible combinations of PRMs. For example, if the number of dynamic subsets

is two, and the number of PRMs per set is three, then the number of complete designs is nine. The output of this phase is a group of bitstream files representing the full design and individual PRMs.

3.3.2 RETC-CPS Design Flow

Lack of automation is a major bottleneck confronting many security solutions in both the software and hardware domains. For example, most formal verification methods and especially theorem proving techniques suffer from automation difficulties because of the need for human interaction and the huge state space of modern systems. Some hardware-based security solutions such as proof-carrying hardware [53] and gate-level information flow enforcement [86] adopt a bottom-up design methodology. Such an approach complicates secure system development by introducing a huge amount of low-level details to the automation process. The RETC-CPS design flow addresses these challenges by decoupling security development from the system design flow.

RETC-CPS adopts a top-down design methodology which can be easily integrated into reconfigurable hardware design flow to build the trust anchors. The RETC-CPS flow is independent of and parallel to the conventional FPGA design flow as illustrated by Figure 3.9. The FPGA design flow is kept unchanged because of the policy-based security approach adopted by RETC-CPS. Guard components are developed by a trusted party aware of the system security requirements, top-level interfaces, and untrusted components. This party can be the system developer if the condition of security awareness is satisfied. In this section we present the extra design phases added by the RETC-CPS flow.

Security Policy Specification, Description, and Translation

Policies are the means to dynamically specify security specifications in computational platforms without changing system-specific implementations. By changing policies, trust anchors can be continually updated enabling trust extensibility. A security policy is a high-level specification of the security properties that a trusted system should possess. These specifications must be explicitly

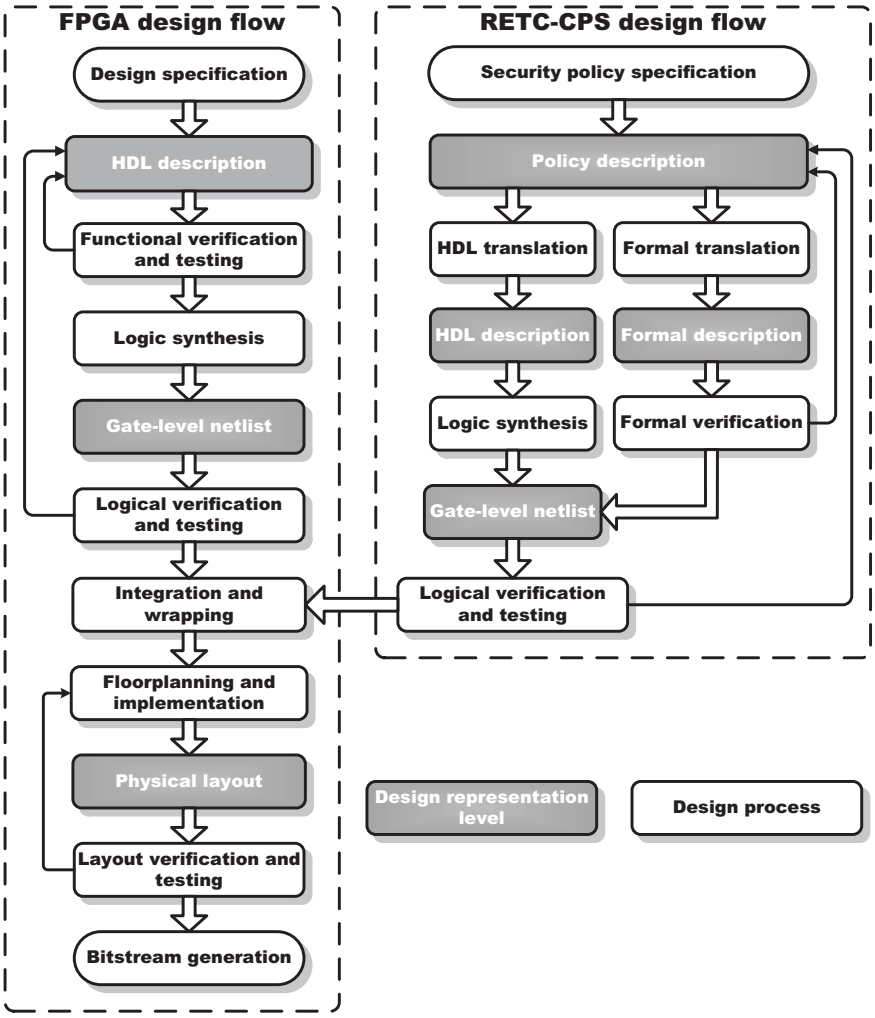


Figure 3.9: RETC-CPS design flow.

stated at the start of system development [12]. Unlike design specification which should capture all system functional and architectural details, policy specification only describes a set of rules and boundaries defining the system secure operation. This distinction is extremely important because it sets an upper limit on the complexity and overhead induced by RETC-CPS security components. In complex systems, the security policy can be decomposed into smaller sub-policies facilitating the allocation of efficient security mechanisms to enforce the overall policy [147]. However, a top-level policy is essential to any security scheme because sub-policies may not capture component interactions.

In the policy specification phase, a high-level security policy is described and documented by a trusted party aware of the security requirements and threats. In CPSes, security policies can be described in several ways with a major objective of protecting physical systems against various cyber threats. Top-level security policies can be assisted by physical models and equations governing the controlled physical systems. Model-assisted security solutions can accurately detect anomalous behaviors and accordingly enforce corrective actions based on the models. This approach is quite similar to the security through diversity approach deployed in the software security domain [105]. Such an approach can be easily integrated into model-based design flows to enhance development productivity of RETC-CPS security components. However, model-based security schemes may induce significant resource and computation overheads because of the associated redundancies.

On the other hand, security policies can be established by defining permissible and impermissible behaviors of the protected system in the form of rules and assertions. In CPSes, policy rules must be capable of describing both application-dependent as well as component-based security rules. To enable run-time enforcement of security policies, rules should define violation detection criteria along with the corresponding countermeasures. This policy-based approach requires an appropriate representation and management of security policies. An application-specific policy language is the best means to describe security policies. Many research efforts have been devoted to developing policy languages such as Ponder [43], Rei [73], and the property specification language (PSL) [1]. Some have proposed development of security-specific policy languages such as the standard security policy language (SSPL) [9]. Regardless of specific details, security policy languages must possess certain characteristics including declaration, formality, expressiveness, scalability, and extensibility.

The next phase is policy translation into HDL descriptions and formal models using language-specific compilers. A detailed example of translating a memory access policy language into Verilog HDL is presented in [77]. The policy description language must be formal to enable formal verification of security policies. Formal languages are capable of expressing stateless as well as stateful policies incorporating temporal logic propositions. Security policies described by formal languages can be translated into formal models verifiable using an appropriate model checking

tool. Temporal assertions generated by formal policy languages can be translated into advanced HDLs such as SystemVerilog. Such HDL descriptions can be synthesized using advanced tools enabling synthesis of temporal propositions, and implemented as run-time hardware checkers [26]. However, development of new security policy languages and compilers is not the focus of this work.

Formal Verification

Verification is the major bottleneck in modern design flows, where up to 80% of the overall design costs are due to verification. Simulation can confirm design correctness under some conditions but cannot affirm that the design is free of errors under all operating conditions. The compelling need for error-free designs in security-critical applications has increased interest in formal verification methods [49]. Formal verification is the use of mathematical techniques to ensure that a design conforms to some precisely expressed notion of functional correctness. Formal verification is a substantial requirement in digital system design, not only to enable trusted system development but also to provide a proof of trust to regulators and stakeholders. Although formal verification research has achieved significant improvements in the recent decade, simulation remains the main validation tool of complex digital design, especially at the higher levels of design abstractions.

Our vision for RETC-CPS is to integrate simple, trusted, and verifiable components to top-level system and untrusted component interfaces. These guards monitor and possibly override the underlying interfaces to regulate their operation according to an explicitly defined policy addressing the system security requirements tailored to a certain class of applications. Verification of third-party IP cores requires the availability of associated formal models which can be only obtained from the IP developer. This leads to the dilemma of trust in the third party. Unlike conventional security approaches, RETC-CPS does not require formal verification of untrusted components because of reliance on verifying in-house guard components. Furthermore, the RETC-CPS approach alleviates the need to fully verify complex systems and components which can be intractable in modern designs. Formal verification is only required for guard components to prove their correct-

ness against a set of application-driven security properties. The policy-based approach keeps guard components as simple as possible to ensure verification tractability.

The two main elements of the formal verification problem are the design model and specifications. There are various approaches to formally verify a digital design, with theorem proving (logical inference) and model checking being the most deployed verification techniques. In theorem proving, both the design and specifications are formally described by applying rules of inference to the specifications in order to derive new properties of interest. Theorem proving techniques are often too expensive because of automation difficulties and the need for human interaction [72].

Model checking was developed by Clarke and Emmerson in 1981 [39] and, currently, it is the most widespread technique for digital design verification. In model checking, the verified design is abstracted as an FSM and the specifications are written in temporal propositional logic, either as computation tree logic (CTL) or linear time logic (LTL). Model checking is a fully automated exhaustive search in the state space of the FSM to verify that the specifications are satisfied by the FSM model [83]. The model checking algorithm is developed based on the mathematical fixed point theorem of complete lattices [165]. Most automatic model checkers are capable of providing a counterexample in case of verification failure.

The main concern with model checking is a possible state explosion problem caused by the enormous state space of current digital designs [22]. The state explosion problem constitutes the main bottleneck confronting model checking of software designs due to the enormous state space. On the other hand, digital hardware designs are characterized by a finite state space facilitating model checking [149]. However, high-density ICs and complex hardware components and IP cores may still suffer from the space explosion problem. RETC-CPS addresses this problem by reducing the need to formally verify whole systems or components. The provided alternative only requires formal verification of RETC guard components having reduced state spaces.

Many enhancements have been added to the original model checking algorithm to deal with the state explosion problem. The introduction of symbolic model checking by McMillan in 1992 is generally considered a breakthrough, enabling verification of systems clearly out-of-reach of any

explicit-state model checker [115]. Symbolic model checking applies Clarke's model checking algorithm for a symbolic version of the verified model. The FSM is represented with boolean encoding and is manipulated using reduced order binary decision diagrams (ROBDDs). Symbolic model verifier (SMV) is the software tool developed by McMillan to enable automatic verification. Verified designs are modeled in a custom description language while the verification properties are expressed as temporal logic propositions.

Considerable research effort has been devoted to enable formal verification of both software and hardware systems, especially in safety-critical applications. Most work seeks to provide a set of formally verified primitives forming a basis for system design. However, some efforts attempt to integrate formal verification with the hardware design flow in a simplified way facilitating verification practises for the majority of digital system developers. Arvind *et al*. highlighted research challenges that reduce the gap between formal methodologies and engineering practices [17]. Formal methods must be invoked through high-level design languages and must present a semantic model that makes sense to the designer. Examples incorporating assertions in SystemVerilog language expose the need for several verification techniques even in the same application. Integration of model checking techniques with the IBM hardware development flow has been recently described by Schlipf *et al*. [146] and Abarbanel-Vinov *et al*. [5]. They concluded that model checking is a powerful extension of formal verification complementing simulation and emulation techniques. Development of a memory bus adapter at IBM illustrates that 24% of design defects have been detected with model checking, while 40% of these bugs have not been caught by simulation.

Integration and Wrapping

Subsequent steps include the conventional phases of the FPGA design flow. Guard component logic synthesis translates the guard's HDL description and formal assertions into logical netlists incorporating hardware checkers. The next phase is logical verification and testing of the interface guard logical operation. Failure of the formal verification or the logical verification phases requires manipulating the security policy to fix verification errors. Outputs of this phase are netlist files of

the system top-level guard and the untrusted component guards.

The next phase is integrating guard components with the target system. In this step, guard component netlists are instantiated in the design along with the system components. Guard modules are inserted into system top-level and untrusted component interfaces. The next step is floorplanning and implementation of the full design incorporating the guard components. Physically separated partitions are allocated to different system components according to the component's needed resources and interfaces. Physical separation helps to regulate data flow between internal system components and minimizes physical shared resources, such as LUTs and BRAMs. The design is then implemented using conventional FPGA design tools and verified using simulation to assure the satisfaction of timing constraints. The bitstream file is generated and protected using standard authentication and encryption methods to be securely delivered to the target FPGA platform.

3.4 Summary

In this chapter we discussed the RETC-CPS design concerns including assumptions and outcomes, high-level architecture, and development flow. RETC-CPS is a complementary protection scheme serving as a last line of defense against various cyber threats. This scheme targets systems and applications where security policies can be clearly formulated. The system under protection contains untrusted components and IP cores developed by third parties and treated as black boxes. Targeted systems incorporate untrusted software- and hardware-based components, and attacks against them can originate from different sources including the components themselves. Such systems accommodate software updates as well as hardware reconfigurations enabling potential exploits. Trust needs to be enforced at different abstraction layers using both software and hardware trust anchors. In this dissertation, we only study development of hardware interface guards to address specific threat models targeting both software and hardware layers.

RETC-CPS does not change the conventional design flow, yet it adds a number of phases relying on existing tools in reconfigurable hardware design and formal verification methods to develop

the system trust anchors. An application-specific formal policy language describes a system-level security policy to be enforced by hardware guards. Established physical and mathematical models of the target system can constitute the basis of the security policy. Language-specific compilers translate the captured policy specifications into HDL descriptions and formal models in preparation for circuit implementation and verification. Guard components are synthesized and verified using existing reconfigurable hardware design and model checking tools. RETC trust anchors are integrated with the system netlists developed using the conventional design flow.

Trust anchors added by the RETC-CPS protection scheme include the DREC, SRC, configuration firewall, and low-level interface guards. The DREC is a hardware component wrapping the target system and enforcing a system-level, application-oriented security policy at the top-level system interface. The SRC is a hardware module responsible for receiving update or reconfiguration requests from specific system components and validating these requests according to a predefined reconfiguration policy. The configuration firewall is a static hardware module which receives update commands from the SRC, and securely manages transfer, authentication, and reconfiguration of the requested PRM bitstreams. Low-level interface guards are hardware trust anchors wrapping untrusted components and enforcing low-level, circuit-dependent security policies at the component interfaces. The RETC protection scheme is developed in reconfigurable hardware, yet the presented concepts are generally applicable to software-based systems as well as ASICs and SoCs. Hardware-based trust anchors simultaneously address system security and performance requirements.

In order to illustrate the RETC-CPS features, we apply RETC to different design examples in the next three chapters. These applications are carefully selected to demonstrate applicability to a wide range of CPSes, and the efficacy in addressing various threats raised by incorporating untrusted components in CPS embedded controllers. In this dissertation, the main focus is presenting high-level architectures and prototype development for the RETC-CPS protections and evaluating these defenses against specific software- and hardware-based threat models. The selected applications demonstrate the RETC-CPS main functionalities presented in this chapter. Design automation is planned to be a part of our future work.

A CR example illustrating the high-level architectures and prototype developments of the DREC, SRC, and PAU is provided in Chapter 4. The main objective of this chapter is demonstrating how RETC-CPS can secure dynamic reconfiguration in a CR platform containing untrusted software and hardware components. The major threats are compromised update requests issued by untrusted software modules responsible for device reconfiguration, and open communication channels used to deliver PRM bitstreams. CR security threats can directly affect spectrum integrity and, consequently, primary users via illegal channel allocation. This example also demonstrates the language-based design approach to develop RETC-CPS guards.

An example of developing low-level interface guards protecting untrusted components in embedded controllers is presented in Chapter 5. The main objective of this chapter is demonstrating how to apply RETC-CPS low-level protections to secure third-party IP cores extensively employed in reconfigurable designs. A high-speed I/O serial interface adapter IP widely deployed in modern CPS controllers illustrates RETC-CPS low-level protections. Third-party IP cores are exposed to malicious insertions and alterations embodied by HTHs. The main challenge facing detection of HTHs in third-party IP cores is difficulty of acquiring golden references and trusted models. This example also demonstrates how to secure network interactions against unconventional threats raised by embedded HTHs.

An example of developing a system-level interface guard in process control systems is given in Chapter 6. The main objective of this chapter is to demonstrate how RETC-CPS can be applied to secure process controllers and preserve the stability of the controlled physical processes. The threat model is erroneous controller behavior associated with untrusted components employed in PCS embedded controllers. An aircraft pitch controller is selected as an example of a security-critical control system. System models and control principles are employed to build the DREC for the chosen controller. The DREC exploits timing characteristics of process control systems to predict and preempt erroneous behavior resulting from either faults or cyber threats. This application also demonstrates how to apply the RETC protection scheme to CPSes developed using model-based design flows.

Chapter 4

Application to Cognitive Radio Platforms

CR is an emerging technology that aims to improve radio spectrum utilization in wireless communication applications. The main idea is that a secondary user can dynamically utilize vacant radio channels allocated to primary licensed users according to a spectrum access policy. This concept of opportunistic access of spectrum holes can significantly increase spectrum utilization efficiency. A CR device is a smart SDR capable of changing its architecture to exploit different spectrum opportunities. The main functionalities of a CR device are spectrum sensing, opportunity reasoning, access policy enforcement, architectural adaptation, and spectrum access. Reconfigurable hardware is a potential platform offering both flexibility and performance required by CR. Commonly, CR devices are built with untrusted software and hardware components using model-based design tools. Therefore, CR technology provides a typical example of a CPS containing untrusted components, built using reconfigurable hardware, and including frequent system updates.

The RETC-CPS protection scheme secures reconfiguration control in CPS applications including frequent updates with potential exploits. Two main challenges are associated with secure reconfiguration control in a CPS containing untrusted components. The first challenge is validating update requests issued by an untrusted component, which is usually a software processor. Commonly, software verification is extremely difficult and software attacks are more prevalent. Therefore, reliance on software correctness to validate update requests threatens the platform security. The

second challenge is authenticating update contents and ensuring secure update generation, transfer, and application.

As described in Chapter 3, RETC-CPS provides a secure, rigorous, and straightforward update mechanism addressing the foregoing challenges. In this chapter, we introduce the application of the RETC-CPS protection scheme to CR. The main threat model introduced by this example is compromising update requests issued by untrusted software. We do not discuss potential software exploits that can result in compromised update requests, yet we acknowledge plausible consequences due to lack of trust in software. The main components responsible for secure reconfiguration control in the RETC-CPS architecture are the DREC, SRC, and configuration firewall. CHARE-CR is the name of the framework integrating the DREC, SRC, and configuration firewall implemented in reconfigurable hardware to enforce spectrum access policies in a CR platform. CHARE is an essential part of the RETC-CPS overall security architecture.

The results of this research have been applied to the development of a trusted platform for cognitive radios, as part of the NSF-funded AUSTIN (Assuring Software Radios have Trusted Interactions) project with the collaboration between researchers from the Wireless Information Network Laboratory (WINLAB) at Rutgers University, Wireless @ Virginia Tech (VT), and the University of Massachusetts (Amherst). AUSTIN consists of efforts covering theoretical work on regulating CR interactions, the design of architectures for securing networks of CRs, on-board hardware and software security methods, and algorithms and protocols to enact regulation at the local, network or external level. The proposed work provides a trusted platform capable of understanding and enforcing policies and regulations associated with CR interactions. Designing security into (rather than onto) the reconfigurable embedded platform along with hardware policy enforcement make this project considerably different from previous work in the area, and serves as a primary focus for the proposed research.

In this chapter we present the high-level architecture, implementation details, and evaluation of the CHARE-CR framework. The remaining of this chapter is organized as follows: An overview of CR potentials and security challenges is provided in Section 4.1. In Section 4.2, we present

an overview of the CR policy engine, highlight previous approaches to build the policy engine, and introduce our approach to develop the CHARE-CR policy engine. Description and translation of spectrum access policies into hardware assertion checkers are presented in Section 4.3. The CHARE-CR high-level architecture in reconfigurable hardware is presented Section 4.4. Implementation details and results of the SRC and the configuration firewall are given in Section 4.5. This chapter is summarized in Section 4.6.

4.1 An Overview of CR Potentials and Security Challenges

Radio spectrum is a vital yet limited resource in wireless communication. In current wireless allocation charters, governmental agencies assign radio spectrum to license holders on a long-term basis for large geographical regions. Recently, because of the increase in spectrum demand, this technique has led to scarcity in particular spectrum bands [8]. Open spectrum bands (e.g., the 900MHz, 2.4GHz and 5GHz ISM/U-NII bands) have become overcrowded with various wireless applications. In contrast, a large portion of the assigned spectrum is used sporadically, leading to underutilization of a significant amount of the spectrum. Figure 4.1 illustrates the frequency allocation chart in the United States as reported by the National Telecommunications and Information Administration.

Dynamic radio access is an alternative to inefficient static spectrum allocation. In dynamic spectrum access (DSA), licensed radio bands are opened to unlicensed operations on a non-interfering basis to incumbent users. To achieve interference-free coexistence, secondary users identify fallow licensed spectrum and opportunistically utilize these white spaces as shown in Figure 4.2. The main components of DSA are spectrum sensing, spectrum decision, spectrum sharing, and spectrum mobility. CR is the key enabling technology of DSA [8], and IEEE 802.22 is the first worldwide wireless standard that is designed to achieve opportunistic spectrum sharing based on CR technology [160].

A CR device is a smart SDR, a radio that can be configured by software to dynamically access the

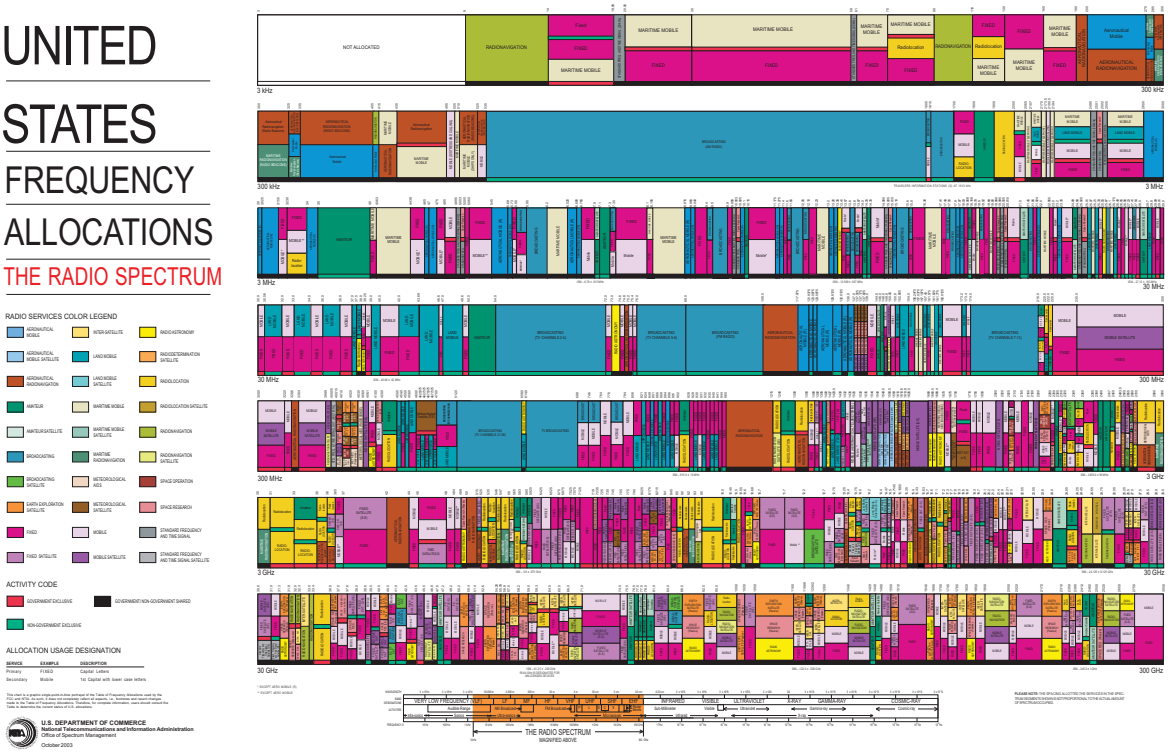


Figure 4.1: Frequency allocation in the United States.

spectrum and adapts its configuration based on perceived changes in its environment. The main characteristics of CR are cognitive capability and reconfigurability, and the basic functionalities of a CR system include spectrum sensing, cognitive medium access control and cognitive networking [100]. A CR dynamically adapts the physical layer by scanning spectrum and changing modulation waveforms and transmission power to utilize spectrum opportunities [139]. These new SDRs are developed with powerful processors supporting advanced networking functionalities. CRs are built on programmable platforms supporting open-source development [107].

Traditional radios are developed with fixed communication parameters to use predetermined channels licensed to particular radio applications. Enforcing proper transmission behavior in such legacy radios (e.g., cellular phones) is relatively straightforward since the spectrum access policies are an inseparable part of the radios firmware and platform. In contrast, CR is a highly programmable technology that can opportunistically use fallow radio channels. As shown in Fig-

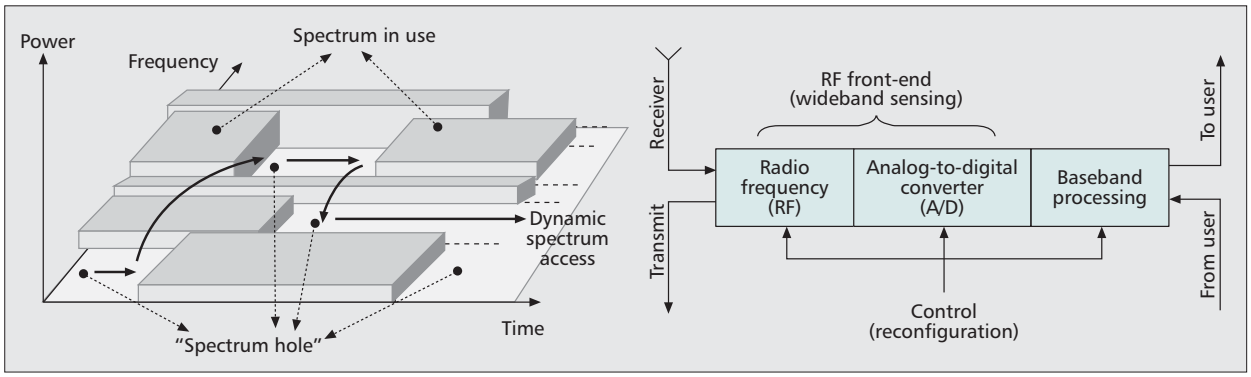


Figure 4.2: Dynamic spectrum access and CR transceiver architecture [8].

ure 4.3, traditional computing platforms have a fixed trust hierarchy consisting of static hardware, operating system, middleware, and application software. However, trust inheritance becomes complicated in configurable platforms when software updates the underlying hardware structure. "In hardware we trust" is no longer axiomatic since the hardware can be modified to violate specific policies. CR programmability can be abused by malicious entities participating in radio software development to violate or bypass proper spectrum access policies.

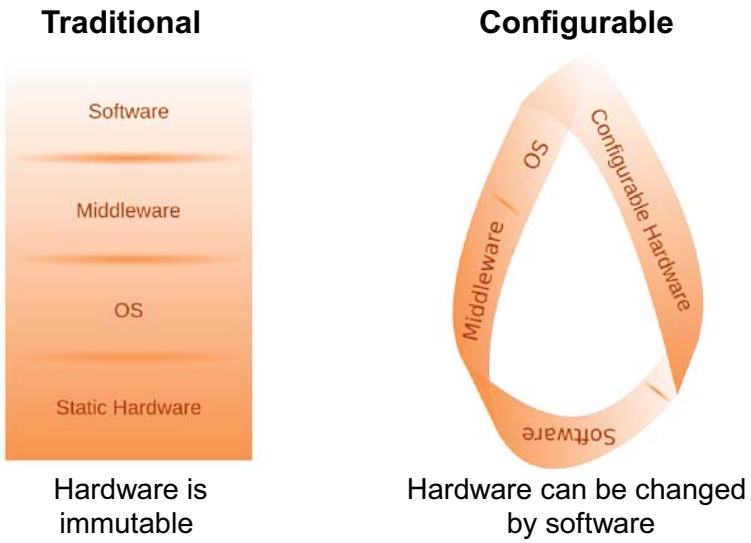


Figure 4.3: Static versus reconfigurable platforms.

CR security threats can directly affect spectrum integrity and, consequently, primary users via illegal channel allocation. Plausible spatial and temporal long-term effects may occur because the

cognitive nature of CR [29,38]. Programmability of reconfigurable hardware-based CR introduces additional security concerns since all layers of the protocol stack can be modified, including hardware implemented-layers. In traditional computing platforms, the software has a static hardware root of trust. In a reconfigurable hardware platform, trust inheritance is complicated by the ability of software to modify the hardware structure. An analogy is trying to build a secure (software) fence on a shifting foundation of (hardware) sand. Thus, it is essential that security is designed into, rather than onto, the next generation of SDR platforms. Controls must be imposed on the allowed hardware changes, and hardware should retain some oversight rather than rely solely on software correctness and integrity.

In order to fully reap the potential benefits of dynamic spectrum access, radios need to be able to cope with evolving spectrum access policies and constantly changing application requirements. Policy-based cognitive radios address these challenges by decoupling policies from device-specific implementations and optimizations. These radios can invoke situation-appropriate adaptive actions based on policy specifications and the current spectrum environment [63]. Figure 4.4 illustrates the policy-based CR architecture proposed by DARPA's neXt Generation (XG) communication program [134]. Basic components include a policy manager for managing local policies and communicating with a remote server, a policy conformance reasoner for reasoning over policies and device-provided evidence, a system strategy reasoner for adjusting and selecting the device's operational mode, and a policy enforcer for governing the radio by permitting only allowed transmission requests.

The policy reasoner is a component that evaluates compliance of transmission requests to DSA policies by making logical inferences from a set of policies. It should be able to formally prove or disprove a hypothesis that a transmission request is policy compliant, and capable of inferring additional knowledge such as identifying transmission opportunities for unbound transmission requests. The first generation of policy reasoners are designed as stateless systems to facilitate component verification and certification. The policy enforcer examines transmission control commands received from the system strategy reasoner to ensure its conformance to the DSA policies. The policy enforcer may maintain a record of previous transmission decision to accelerate future

computations.

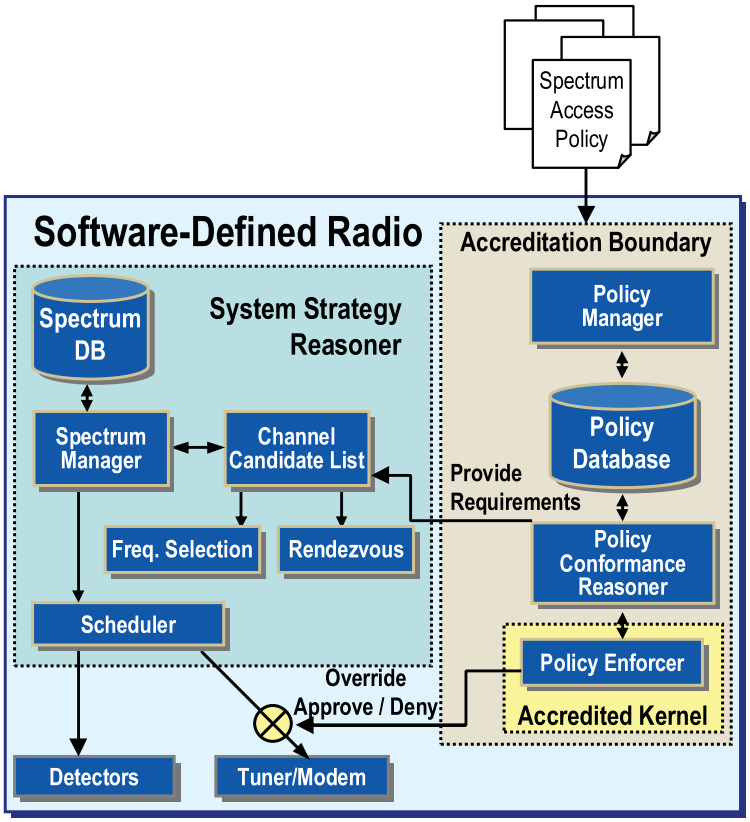


Figure 4.4: Policy-based CR architecture [134].

There is a dramatic increase in standards development due to the progress of CR technology and rapid changes in spectrum allocation charters dictated by the Federal Communications Commission (FCC). With the introduction of DSA techniques, the need for new radio spectrum policies necessitated the development of modern standards to manage spectrum access rights. The IEEE P1900 committee was established in 2005 to develop standards dealing with new technologies and techniques of the next-generation radio and advanced spectrum management [66]. The IEEE P1900.5 working group focuses on policy language and policy architecture for managing CR for DSA applications.

Policy languages must serve all the participants of the spectrum utilization including regulators, operators, stakeholders, manufacturers, and consumers. A standardized policy language adds value

in situations where flexibility is needed such as frequent changes of radio access policies [118]. DARPA's XG communication program developed a cognitive policy radio language (CoRaL), based on a subset of first-order logic, supporting description of various DSA policies and tolerating policy extensibility [46]. CoRaL satisfies the requirements of having expressive constructs for numerical constraints, supporting efficient reasoning, and being verifiable [56].

The state of the art in CR as an emerging technology is an extensive growth in standard development supported by a far less implementation effort. Most CR implementations are software-based employing the open-source GNU radio software development toolkit that provides signal processing blocks to implement software radios, and the universal software radio platform (USRP) for RF transmission. CR implementations on reconfigurable hardware platforms are much less common. Delorme *et al*. presented an FPGA partial reconfiguration design approach for CR based on network-on-chip (NoC) architecture [45]. Lotze *et al*. advanced a model-based design approach to CR design [106]. To the best of our knowledge, CHARE-CR is the first showing how reconfigurable hardware enables creation of a trusted CR platform containing untrusted components. [57].

4.2 CR Policy Engine Overview

As shown by Figure 4.5, the main components of a policy-based CR device are a system strategy reasoner, a policy reasoner and enforcer. The system strategy reasoner gathers sensory information, formulates transmission strategies based on these information, and interacts with the policy reasoner to determine policy-conformant opportunities. Design of the system strategy reasoner is not the main concern of this work. The policy reasoner is responsible for validating transmission requests according to the DSA policies and the underlying radio capabilities. Different policies can include conflicts between permissive and prohibitive rules. Transmission requests may include missing or incomplete information required for reasoning. The policy reasoner should consider and resolve conflicts between different DSA policies and should be capable of dealing with underspecified transmission requests. Resolving DSA policy conflicts necessitates prioritizing prohibitive

rules over permissive rules to avoid potential spectrum interference. A secondary functionality of the policy reasoner is to provide optimal transmission constraints to the strategy reasoner addressing denied and underspecified requests.

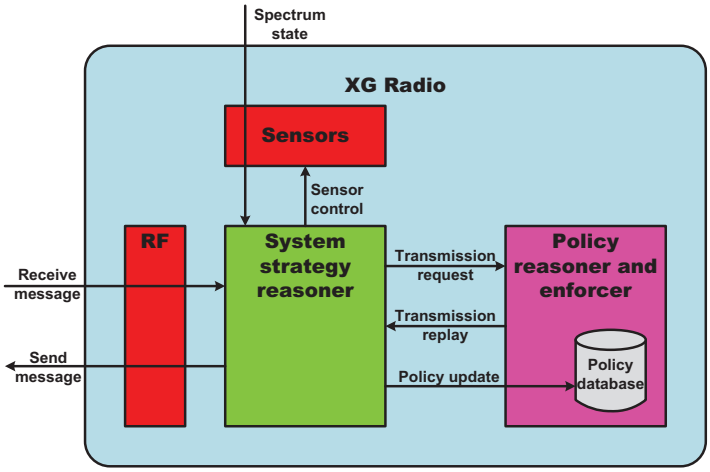


Figure 4.5: XG CR architecture [172].

Significant research efforts have been expended in the investigation of software development of the DSA policy reasoner. The core reasoning problem is inferring transmission legitimacy from a set of policies, evidences, and facts about the radio which is quite similar to classical theorem proving problems. The transmission permit can be obtained by an incremental proof including several rounds of interaction between the policy reasoner and the strategy reasoner.

Wilkins *et al*. present a software implementation of the policy engine based on CoRaL [172]. This implementation uses a custom-made proof system supporting a combination of functional, equation, and logic programming language and automated theorem proving. In this system, some techniques are employed to reason about transmission requests. Forward reasoning enables quick filtering of inapplicable policies. Partial evaluation techniques evaluate policy terms. Backward chaining supports user-defined predicates and hierarchies. Conditional equation rewriting enables reasoning with user-defined functions. Constraint propagation and simplification techniques support built-in predicates and functions, and enables detecting inconsistencies. Unfortunately, this implementation does not return missing parameter values for underspecified requests. Rather, it

returns an arbitrarily complicated first-order formula representing all possible constraints enabling transmission.

Bahrak *et al*. developed a policy reasoner for processing spectrum access policies represented by binary decision diagrams (BDDs) [21]. This work does not provide an expressive language such as CoRaL to specify DSA policies, yet the system reads certain types of policies written in XML and converts them to Boolean functions represented as BDDs. This is done by assigning a distinct Boolean variable to each atomic expression appears in the policy. A number of graph algorithms are used to translate policies into BDDs, merge policies into a single meta-policy, and reason about transmission request conformance to the DSA meta-policy. Transmission requests are interpreted as Boolean variables by assigning attribute values such as power and frequency to the appropriate atomic expressions in the meta-policy and evaluating the corresponding Boolean values. The policy reasoner is capable of computing opportunity constraints for underspecified and denied transmission requests. In DSA, transmission parameters do not have the the same willingness to change. For example, if a request is denied and the constraints to allow transmission are to reduce the power level or change the transmission time, then the power parameter is more willing to change than the time parameter. This approach supports assigning weights to transmission parameters representing their willing to change. Three graph-based algorithms are proposed to compute opportunity constraints. The developed policy reasoner is evaluated in terms of the decision computation time versus the number of policies and the number of transmission parameters, and the computation time is in the range of 1-20 msec. Such an approach lacks an expressive policy representation framework and semantics for the numeric constraints pervading DSA policies which are represented in Boolean forms enabling BDD manipulations.

Arkoudas *et al*. present a software development of a CR policy reasoner based on Aethna, an interactive proof system for polymorphic multi-sorted first-order logic with equality, algebraic data types, and subsorting [14, 158]. DSA policies are represented and manipulated in the Aethna proof framework based on first-order logic with arithmetic and algebraic data types. Reasoning about transmission request validity is formulated as a satisfiability modulo theories (SMT) problem. Commercial SMT solvers are used to evaluate transmission queries and provide opportunity

constraints for underspecified and denied transmission requests. Optimal reasoning about transmission requests and opportunity constraints is achieved by modeling the problem as an SMT instance of weighted MAX-SAT. The developed policy reasoner is evaluated in terms of the decision computation time versus the number of policies and transmission parameters, and the computation time is in the range of 50-400 msec. This approach provides both expressiveness and performance required by CR technology. Unfortunately, such policy reasoner relies on the Aethna proof framework and cutting-edge SMT solvers requiring extensive computation resources which exceeds the capabilities of embedded CR platforms. This approach suits remote reasoning about DSA queries using powerful servers running the required software tools rather than run-time, infield reasoning required by CR devices.

4.2.1 CHARE-CR Policy Engine Security Enhancement

Unlike previous CR architectures where the policy reasoner and enforcer are developed in software, the CHARE-CR policy engine is composed of a software-based system strategy reasoner and policy reasoner and a hardware-based policy enforcer. Functionalities assigned to the software-based components include discovering spectrum opportunities and reasoning about the legitimacy of these opportunities in the light of predefined DSA policies. Reasoning is the ability to make inferences, and automated reasoning is concerned with developing computational systems that automate this process. Automated reasoning techniques extensively use complex algorithms and data structures suiting software implementations. Often times, automatic reasoners adopt third-party compilers, theorem proving tools, and SAT solvers to accelerate both development and computation. Such critical components are vulnerable to security threats which can result in compromised decisions. The main threat associated with the policy reasoner is allowing transmission requests not conforming to DSA policies and denying requests conforming to DSA policies. Therefore, decisions issued by the policy reasoner software should be inspected by a trusted, policy-aware entity to prove their correctness.

In CHARE-CR, the hardware-based policy enforcer is responsible for inspecting transmission re-

quests and reasoner decisions and validate their conformance to DSA policies. Unlike the policy reasoner, the policy enforcer does not need reasoning capabilities to validate reasoner decisions or evaluate transmission constraints for underspecified requests. Inputs to the policy enforcer can be either a complete request-decision pair or an underspecified request-transmission constraints pair which indicates that all parameter values needed to compute a transmission decision will be available to the enforcer. The policy enforcer simply assigns request parameter values and transmission constraints to the corresponding policy rules, evaluates request legitimacy, and validates reasoner decisions. The hardware-based policy enforcer adds another layer of security to the CR policy engine by validating reasoner decisions in trusted hardware.

The policy enforcer can be implemented in hardware as assertion checkers encoding DSA policy rules. In CHARE-CR, the SRC is the entity playing the role of the policy enforcer. Figure 4.6 illustrates the CHARE-CR policy engine architecture. Inputs to the SRC include transmission request parameters, sensory evidences, radio parameters, and decisions issued by the policy reasoner. Transmission requests address changes to some physical quantities such as power and frequency. In reconfigurable platforms , a secondary functionality of the SRC is mapping physical parameters of approved transmission requests to the corresponding platform attribute changes and/or dynamic reconfiguration commands. The SRC communicates with the configuration firewall to initiate reconfiguration for allowed requests and sends the status back to the system strategy reasoner.

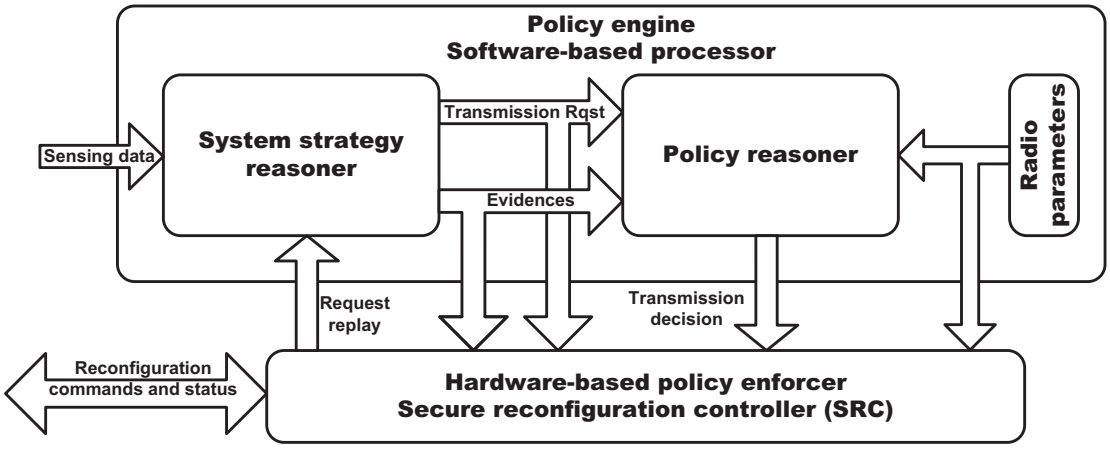


Figure 4.6: CHARE-CR policy engine architecture.

4.3 Dynamic Spectrum Access Policy Description and Translation

DSA policies regulate how a CR device is allowed to behave. Different policies are applicable for different geographical regions and various spectrum bands. Even in the same region or band, different policies may be imposed by different regulators and spectrum allocators. Therefore, a CR device should be capable of loading new policies at run-time. Advantages of using policy-based approach to enable DSA include:

- Policy-based DSA enables expedited adaptation of radio behavior according to situation and environment changes.
- Policies can be easily manipulated by regulators and stakeholders to fit their objectives.
- A policy-based approach decouples policy definition, loading, and enforcement from device-specific implementations, allowing independent evolution of policies and devices, and reducing certification efforts.
- DSA policies can be extended to accommodate adaptable operating conditions and support new policy parameters.

DSA policies are declarative statements dictating permissible and impermissible behaviors of CR devices on a non-interference basis. The main concern of regulators is enforcement of admissible transmission behavior regardless of specific implementation details. On the other hand, radio engineers are interested in exploiting various transmission opportunities by enhancing the system strategy and policy reasoner capabilities to discover, exploit, and validate various opportunities. The policy-based DSA key enabler is a declarative policy specification language with simple and unambiguous semantics serving the purposes of both spectrum regulators and radio engineers. In this section we provide a brief overview of the CoRal policy language developed by DARPA's XG communication program, and present an example of DSA policies described by CoRaL and

enforced by CHARE-CR. There is no standard policy language for DSA, yet CoRaL is the state-of-the-art language for describing DSA policies. The overview and examples presented in this section are mainly acquired from the XG policy language description document [47] and the XG policy engine article [46].

CoRaL is a typed fragment of first-order logic with equality, enabling representation of various domain concepts such as: frequency, power, location, powermask, and signal. A policy is composed of several permissive and restrictive rules expressed using `allow` and `disallow` predicates. Policy rules are logical axioms expressing conditions for which these predicates hold, and can involve declared parameters representing the radio capabilities and sensory information. Conditions can use predicates expressing mode of operation, location, etc. to enable dynamic policy adjustment to the current situation. CoRal enables expressing numerical constraints and powermasks, which are extensively employed in DSA policies, using built-in predicates. For example, a policy can allow transmission in the frequency range between 5 MHz and 5.5 MHz if the sensed channel power is less than -100 dBm. Restrictive policy rules take precedence over permissive rules in case of policy conflicts. Some simple example DSA policy rules encodable by CoRaL are as follows [172]:

- **Frequency band**: Allow transmission between 5180 and 5250 MHz.
- **Time**: Allow transmission between 06:00 and 13:00 local time.
- **Location**: Allow transmission if the radio is at most 30 miles away from the geographic coordinates (39 10’ 30”N, 75 01’ 42”).
- **Node identity**: Allow transmission if the radio belong to Red Cross.
- **Sensed data**: Allow transmission if radio’s peak sensed power is at most -80 dBm and the maximum transmit power is 10 mW.

CoRaL uses ontologies to represent hierarchies of types and related functions or predicates. Ontologies only define concepts, whereas policies must define rules and may also define concepts.

DARPA's XG program defines ontologies for basic types (such as frequency, power, and bandwidth), radio capabilities, evidence, signals, time, powermasks, and transmission and request parameters as summarized in Figure 4.7.. Ontologies and domain concepts are defined using type and subtype declarations. The request-parameter ontology defines three transmission request variables: *req_radio* describes requesting radio characteristics; *req_transmission* represents transmission request parameters; and *req_evidence* provides sensory information evidences.

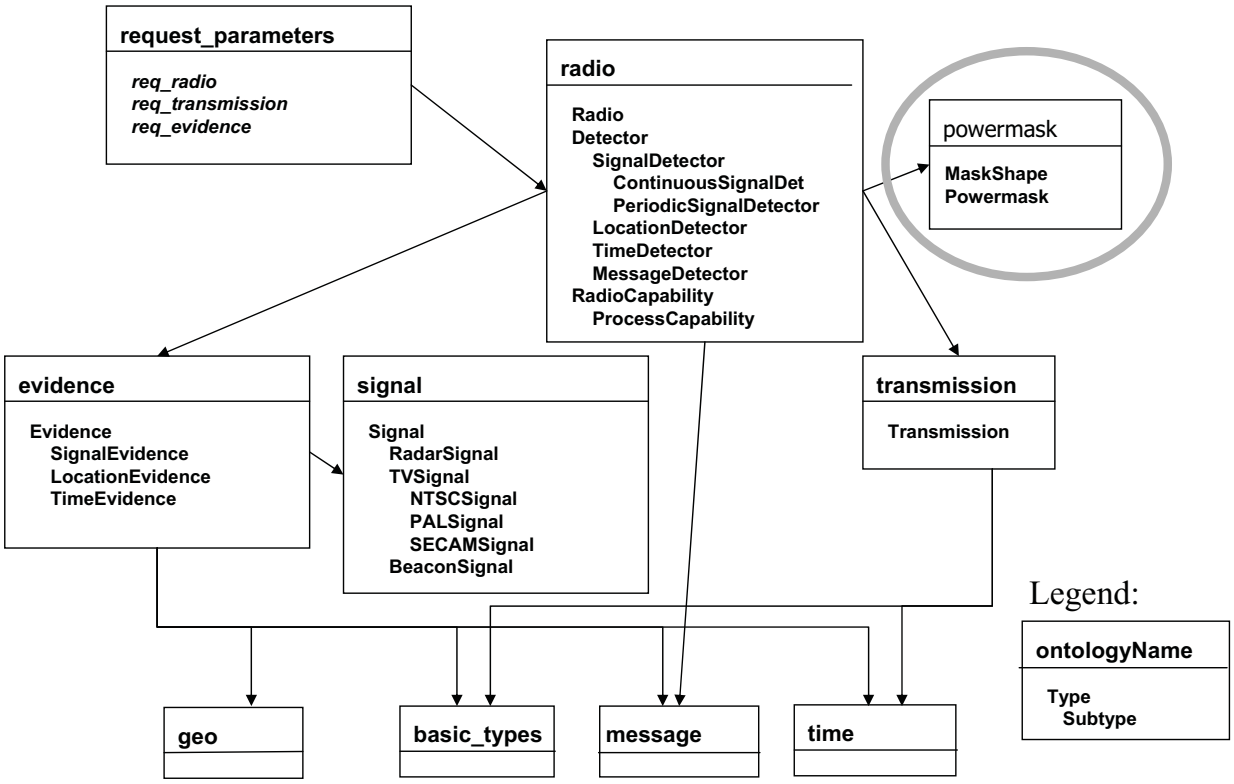


Figure 4.7: Basic XG ontologies [46].

4.3.1 Translating CoRaL into HDL

CoRaL is a formal language backed by a first-order logical system enabling expressing DSA policies and reasoning about transmission requests in an efficient manner. A logical system is a formal system with a form of semantics which assigns truth values to sentences of the formal language. Logical systems include propositional logic, first-order logic, temporal logic, and higher-order

logic. In propositional logic, propositions are statements formulating certain facts about a system of interest and can be only evaluated as true or false. For example, "measured power is less than -100 dBm" is a proposition which can be true or false. Propositional logic lacks the expressiveness required to describe DSA policies because this logic does not support quantification over variables.

First-order or predicate logic is a rich, expressive, and formal language that can be used to describe stateless DSA policies. In first-order logic, each sentence, or statement, is broken down into a subject and a predicate [131]. The predicate modifies or defines the properties of the subject and a predicate can only refer to a single subject. Complete sentences are logically combined and manipulated according to the same rules as those used in propositional logic. The main difference between propositional and predicate logic is that the latter enables quantification over individual variables. In other words, propositional logic is a subset of first-order logic. The alphabet of the first-order logic language includes:

- Constants such as $(a,\ b,\ c,\ \dots)$ denote fixed individual objects.
- Variables such as $(x,\ y,\ z,\ \dots)$ denote variable individual objects.
- Functions such $(f,\ g,\ h,\ \dots)$ denote operations that may be performed on a sequence of individual objects to yield another object.
- Predicates such as $(P,\ Q,\ R,\ \dots)$ denote properties or relations that hold for a sequence of individual objects.

Every function and predicate has an arity indicating the number of associated arguments. The formal language of predicate logic consists of two parts: terms are intended to denote objects in the world, and propositions are intended to express facts about these objects. Both terms and propositions are defined recursively:

- Constants and variables are terms.
- If f is an n-ary function, and $t_1,\ t_2,\ \dots, t_n$ are terms, then $f(t_1, t_2, \dots, t_n)$ is a term too.

- If P is an n-ary predicate, and $t_1,\ t_2,\ \ldots, t_n$ are terms, then the expression $P(t_1, t_2, \ldots, t_n)$ is an (atomic) proposition.
- If ϕ is a proposition, then $\neg\phi$ is a proposition too.
- Nothing else is a term.
- If ϕ, ψ are propositions, then $\phi \vee \psi, \phi \wedge \psi, \phi \rightarrow \psi, \phi \leftrightarrow \psi$ are propositions too.
- If x is a variable, and ϕ is a proposition, then $\forall x\phi$ and $\exists x\phi$ are propositions too.
- Nothing else is a proposition.

The symbols $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ are logical connectives denoting negation, disjunction, conjunction, implication, and equivalence, respectively. The symbols $\forall$ and $\exists$ are quantifiers, called the universal and the existential quantifiers which denote for all and there exists, respectively. Figure 4.8 illustrates syntactic constructs applicable to rule declarations in CoRaL [47].

CoRaL is a declarative language used to describe DSA policies without specifying how to enforce these policies. In the XG project, the policy reasoner uses theorem proving tools to reason about policies described in CoRaL. Automated reasoning is a static technique adopting loop invariant inference algorithms to prove certain facts about first-order logic predicates such as the request eligibility and transmission constraints. Static proving methods do not require translating declarative policies into imperative or executable policies. In CHARE-CR, development of the policy enforcer requires translating declarative policies described in first-order logic into an imperative, synthesizable HDL describing the SRC. Such an approach shifts the problem from the static reasoning domain to the dynamic verification domain. The main challenge is that CoRaL is a language of invariants and assertions designed for simplicity of semantics and tractability of proofs, and not for run-time checking.

In order to implement the hardware SRC, we present some rules to translate CoRaL constructs shown in Figure 4.8 into an HDL. CoRaL constructs include standard and equational rules applied to propositional and first-order formulas. Propositional formulas can include predicates, binary

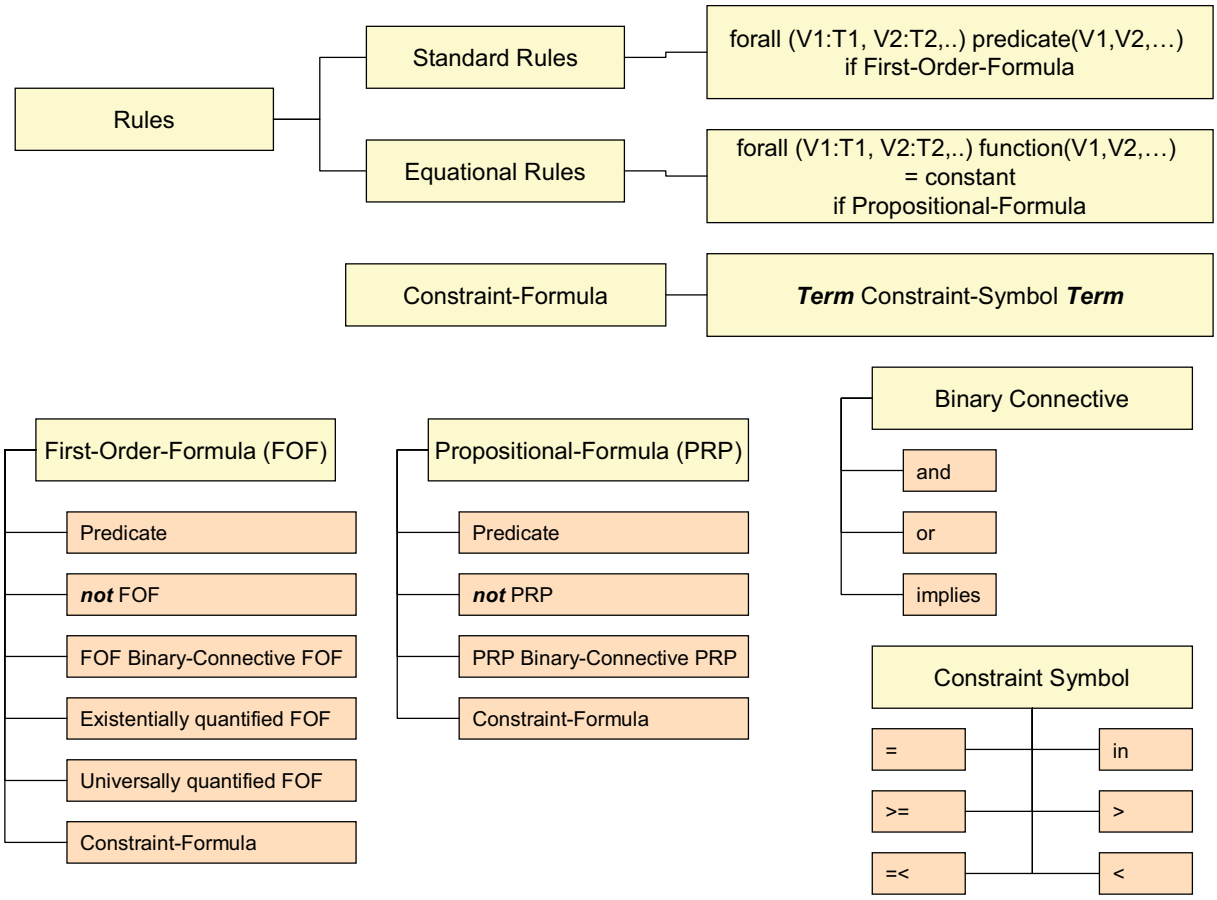


Figure 4.8: Overview of syntactic constructs in CoRaL

connectives, and constraint formulas. In an HDL, predicates are expressed as boolean variables with true or false values. Binary connectives including $\neg$, $\wedge$, and $\vee$ are equivalent to the negation, conjunction, and disjunction logical operators. Implication and equivalence connectives can be expressed using HDL conditional statements and atomic guards. Constraint formulas are directly translated into HDL relational and equality operators. Mathematical functions and numerical constraints extensively used in DSA policies can be efficiently handled using HDL types, constructs, and mathematical operators. In this way, propositional rules and formulas can be easily translated into HDL assertions.

The main difference between propositional and first-order logic is quantifiers which are the main source of the expressiveness power of first-order logic. Quantifiers are essential to state that a group

of objects satisfies the desired properties. Unlike imperative constructs such as loops, quantifiers feature a number of well-understood mathematical properties facilitating automated reasoning using static theorem proving tools. On the other hand, quantifiers are one of the main difficulties confronting run-time checking of policies expressed in first-order logic. In CoRaL, quantification is performed over finite and infinite variable types. Variables can be classified as temporal or non-temporal variables. Temporal variables are of infinite type while non-temporal variables can be of infinite types such as integers or of finite types such as user-defined sets. CoRaL only allows a very specific form of infinite quantified formulas to ensure proper processing between the radio and the policy reasoner.

The main challenge associated with translating first-order logic policies into synthesizable logic is realizing quantification over unbounded variables. In the dynamic domain, quantification over unbounded variables cannot be realized using finite resources. In practice, however, most unbounded variables can be limited to a finite range in the application context. Therefore, we will adopt a context-specific approach to reduce the range of quantified, unbounded variables introduced by the DSA policy. This is quite similar to run-time checking for program verification presented in [175].

In CHARE-CR, a formula quantified over a temporal variable is translated into a hardware checker continuously asserting corresponding predicates and computing associated functions. In other words, quantification over time is translated into continuous run-time monitoring. A formula quantified over a non-temporal variable is translated into multiple instances of the formula realization developed for different values of the quantified variable. For non-temporal variables of infinite type, quantification is bounded by identifying the application context that enforces bounds on the quantified variables. For example, if the quantified variable is a mobile base station in the U.S., this variable can be bounded for a CR device located in a certain geographical area. This approach can be generalized to bound variable range for different CR device situations and contexts. The CHARE-CR framework can exploit dynamic reconfiguration of the CR device to update the SRC according to the device situation and the application context.

In this work, we do not develop algorithms to automatically translate from CoRaL to HDL, but

rather just establish the translation feasibility. Table 4.1 presents some examples of CoRaL rules and their SystemVerilog HDL equivalents. SystemVerilog supports enumerated types, permits a `for` loop index to be declared in the for statement, , adds the `++` operator, and has a `foreach` statement to iterate over an array. Presented examples cover all CoRaL syntactic constructs illustrated by Figure 4.8. Registers and variables representing request parameters and predicates presented in Figure 4.7 are declared once for all examples. The modulus operator is used to map integer types to a subrange, prevent overflow, and automatically wrap from the last value to the first. Processes set a register to a default value, and override that assignment only when the condition or its negation is met. This synthesis-friendly scheme works nicely for both "if exists()" and "if forall()" quantifiers, and allows both loop forms to exit early using a `break` statement.. Note that because synthesis must unroll `for` loops, the `for` loops essentially have implicit `generate` statements.

Table 4.1: CoRaL rule examples and their equivalent in SystemVerilog HDL.

Rule type	**CoRaL description and SystemVerilog HDL translation**

CoRaL types definition and predicate declaration	`deftype Frequency = Float;`
	`deftype Power = Float;`
	`deftype Speed = Float;`
	`deftype DetectionParameters =`
	`{minDuration:TimeDuration,`
	`dwellTime:TimeDuration,`
	`minRate:SensingRate,`
	`allFrequencies:{Frequency}};`
	`const device_ok : Pred;`
	`const sensing_ok : Pred;`
	`const usage_ok : Pred;`
	`const frequency_ok : Pred;`
	`const channel_ok_slave_rd: Pred;`
	`const database_test_ok : Pred(Int);`
	`const in_restricted_area : Pred(Int);`
	`const dbtest_ok : Pred(Int);`
	`const emission_test_ok : Pred(Int);`
	`const db_and_em_test_ok : Pred(Int);`
	`const max_detected : Pred(Signal, Power, TimeDuration,`
	`DetectionParameters);`
	`const tests_ok : Pred;`
	`const maxSpeed : Speed;`

SV HDL type definition and register declaration

```
typedef enum {yes=1'b1, no=1'b0} Pred;
typedef real Frequency;
typedef real Speed;
typedef real Power;
typedef real Rate;
typedef Signal;
Pred req_ready; //This signal indicates the existence of a
    DSA request
//////////////// Predicate registers ////////////////
Pred device_ok;
Pred sensing_ok;
Pred usage_ok;
Pred frequency_ok;
Pred allow;
Pred channel_ok_slave_rd;
Pred max_detected;
// Array size is assigned to an arbitrary large value for
   infinite variable types
Pred in_restricted_area[0:1023];
Pred database_test_ok[0:1023];
Pred dbtest_ok[0:1023];
Pred emission_test_ok[0:1023];
Pred db_and_em_test_ok[0:1023];
//////////////// Request parameters ////////////////
Power MaxRadarPower[0:127];
Power MaxThreshold[0:127];
Signal radarSignal[0:127];
Signal positiveGPS;
struct {time: minDuration = 60;
        time: dwellTime;
        Rate: minRate = 5;
        Frequency: allFrequencies;
    } DetectionParameters;
time smallAge;
time Age;
Speed maxSpeed;
```

Standard rule with predicate

```
allow if device_ok;
```

```
always @ (posedge clk iff req_ready) begin
  allow <= no;
  if (device_ok) begin
    allow <= yes;
  end
end
```

Standard rule with negated formula	`(forall M:int) database_test_ok(M) if not in_restricted_area(M);`
	
	`always @ (posedge clk iff req_ready) begin`
	`  foreach(database_test_ok[i])  //The for loop and foreach statement can be used to realize quantification over a variable`
	`  database_test_ok[i] <= no;`
	`  if (! in_restricted_area[i]) begin`
	`  database_test_ok[i] <= yes;`
	`  end`
	`end`

Standard rule with binary connective	`allow if device_ok and sensing_ok and usage_ok;`
	
	`always @ (posedge clk iff req_ready) begin`
	`  allow <= no;`
	`  if (device_ok && sensing_ok && usage_ok) begin`
	`  allow <= yes;`
	`  end`
	`end`

Standard rule with constraint formula	`frequency_ok if carrierFrequency in {470..512};`
	
	`always @ (posedge clk iff req_ready) begin`
	`  frequency_ok <= no;`
	`  if ((carrierFrequency > 470.00) && (carrierFrequency < 512.00)) begin`
	`  frequency_ok <= yes;`
	`  end`
	`end`

Standard rule with universally quantified formula (infinite variable type)	`(forall M:int) db_and_em_test_ok(M) if dbtest_ok(M) and emission_test_ok(M);`
	
	`always @ (clk iff request_ready) begin`
	`  for (i = M1; i < M2; i++)  //Infinite variables are bounded based on the CR device context`
	`  db_and_em_test_ok[i] <= no;`
	`  if (dbtest_ok[i] && emission_test_ok[i]) begin`
	`  db_and_em_test_ok[i] <= yes;`
	`  end`
	`end`

<table>
<tr>
<td>Standard rule with universally quantified formula (finite variable type)</td>
<td>

```
emission_test_ok if
    (forall (maxSignal:Signal,maxPower:Power))
    max_detected(radarSignal,MaxRadarPower,someAge
                ,someSensingParams);
```

```
always @ (clk iff request_ready) begin
  emission_test_ok <= yes;
  foreach(maxSignal[i]) begin
    if (!(max_detected(radarSignal[i], MaxRadarPower[i],
       someAge, someSensingParams)) begin
      emission_test_ok <= no;
      break;
    end
  end
end
```

</td>
</tr>
<tr>
<td>Standard rule with existentially quantified formula (finite variable type)</td>
<td>

```
channel_ok_slave_rd if
    (exists MaxRadarPower:Power, MaxThreshold:Power)
    max_detected(radarSignal, MaxRadarPower,
    Time_duration , {minDuration=60, minRate=5,
    allFrequencies={5250..5350,5470..5725}}) and
    (MaxRadarPower >= 200 and MaxThreshold =< -64)
```

```
always @ (clk iff req_ready) begin
  channel_ok_slave_rd <= no;
  foreach(MaxRadarPower[i]) begin
    if ((max_detected(radarSignal[i],MaxRadarPower[i],
      Time_duration,DetectionParameters) &&
      (MaxThreshold[i] <= -64) &&
      (MaxRadarPower[i] >= 200)) begin
    //max_detected is a predefined function returning
       predicate
      channel_ok_slave_rd <= yes;
      break;
    end
  end
end
```

</td>
</tr>
</table>

<table>
<tr>
<td>Standard rule with existentially quantified formula over time</td>
<td>

```
tests_ok if
   (exists Age:TimePoint)
   received(positiveGPS,Age) and
   db_and_em_test_ok(modificationFactor(maxSpeed,Age));
```

```
always @ (clk) begin
//quantification over time is translated into continuous
    monitoring which requires direct, continuous access to
     the checked parameters and sensors rather than just
    reading some registers representing instantaneous
    values
//In this example, received_positiveGPS and maxSpeed are
    the signals needing direct access
  Age <= (Age + clk_period) % positiveGPS_expiration_time;
  if (received_positiveGPS && db_and_em_test_ok(
      modificationFactor(maxSpeed,Age)) begin
    Age <= 0;
    tests_ok <= yes;
  end
  else if (Age == 0)
    tests_ok <= no;
end
```

</td>
</tr>
<tr>
<td>Equational rule w/ constraint formula</td>
<td>

```
(forall S:Speed, A:TimePoint)
    modificationFactor(S,A) = S*A+10 if A>=smallAge ;
(forall S:Speed,A:TimePoint)
    modificationFactor(S,A) = 0 if A< smallAge ;
```

```
//All equational rules can be translated into function
    assignments guarded by a condition which can be a
    predicate, propositional formula, or constraint
    formula
function int modificationFactor(Speed S, time A, time
    smallAge) begin
  if (A < smallAge)
    return 0;
  else
    return S*A+10;
end
```

</td>
</tr>
</table>

4.4 Configurable Hardware-assisted Rule Enforcement Architecture

The programmability and performance of reconfigurable hardware suits data-intensive embedded computing applications. Current practice places dynamic hardware configuration under the control of application-level software, and even proposed OS-managed reconfiguration remains an SLH solution. Dynamic hardware blocks are application-tailored and potentially untrusted. In such platforms, software control of hardware configuration introduces new security concerns compared to static hardware systems. Software modification of hardware structure is analogous to self-surgery, and independent hardware should provide oversight rather than solely rely on the correctness and integrity of application software and circuits.

System trust can be enhanced by enforcing application-specific access control policies using either software or hardware. Hardware, which has greater tamper resistance and is better suited to formal analysis than software, provides policy oversight in the CHARE platform. We insert a hardware-implemented, application-specific controller and monitor on the boundary between static hardware (which hosts software) and dynamic application hardware. This HLH approach retains most of the flexibility of application software directly controlling dynamic hardware, while enhancing trustworthiness, performance, and hardware abstraction.

Separation is a fundamental tool in secure system design and should be used in reconfigurable platforms with hardware and software interactions. As shown in Figure 4.9, the CHARE architecture has four major components: an embedded application processor, a reconfigurable controller-wrapped datapath, dedicated hardware to securely configure the datapath, and secure access to a shared configuration server. Xilinxs Embedded Development Kit (EDK) connects one of the embedded PowerPC 440 processors to peripherals over a Processor Local Bus (PLB). The processor runs real-time Linux for data-intensive applications implemented with both software and custom hardware. The system strategy reasoner and the policy reasoner are implemented in this processor. A GPIO control interface stores datapath update request parameters while buffers transfer data

between software and the reconfigurable hardware.

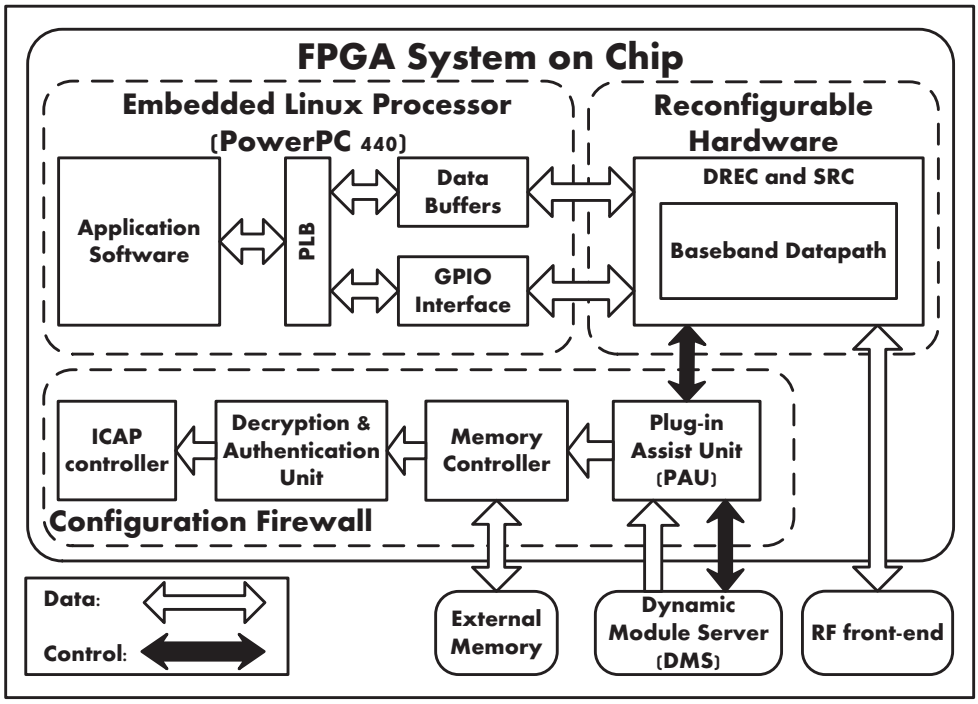


Figure 4.9: CHARE prototype platform.

The reconfigurable hardware block consists of a DREC-wrapped datapath. Parameterized IP cores are connected to implement streaming algorithms in domains such as DSP, communication, and video. The SRC is a hardware-based controller responsible for checking that software-issued datapath update requests conform to policy rules embedded in the DREC. In CR platforms, the SRC is developed by translating a set of active DSA policies into hardware assertion checkers representing the policy enforcer module. Software-visible datapath update registers contained in the GPIO bus interface are not directly connected to the datapath, and may not even have a one-to-one correspondence with actual datapath parameters. Invalid update requests return an error, while requests conforming to policy rules can result in a parameter update, individual module swaps, or a complete datapath plug-in replacement.

The DREC serves as a datapath hardware abstraction layer. This has two advantages: software interaction with the hardware is simplified to enhance portability, and the datapaths detailed im-

plementation is not revealed to software. For example, software is oblivious to the possible use of reconfiguration for swapping cores in response to datapath update requests. The hardware model presented to software restricts the set of control capabilities to that of a virtual ASIC implementing only the configuration options currently authorized. An additional advantage of custom hardware plug-ins is the provision of software-independent cryptographic services. For example, data may be automatically encrypted or decrypted by the plug-in using a key embedded in the datapath controller. Configurable hardware is generally more efficient than software for cryptographic algorithms [132], yet can be changed as readily as software implementations. Session keys may be used as a means of imposing expirations on particular capabilities.

The DREC logic is never updated independently of the datapath, and any policy updates necessitate a complete plug-in replacement. Plug-ins may include monitors to check the operation of individual cores that may be untrusted or subject to single-event upsets. Detection of anomalous behavior signals the DREC to reload the plug-in if an upset occurred or a Trojan horse was enabled. New plug-ins are securely (and perhaps wirelessly) transferred from a remote, shared and trusted DMS. Server-class hardware suits the time- and memory-intensive EDA tools required to generate new FPGA configurations; these tools exceed the resources available in embedded platforms. The DMS runs the PATIS tools to accelerate hardware plug-in implementation through the automatic parallel application of standard implementation tools [33].

The CHARE platform has a configuration firewall containing a dedicated PAU processor for secure communication with the DMS. Similar to the protocols and safeguards used in cryptographic co-processors and TPMs, the configuration firewall performs critical functions in isolated hardware and does not share processor, logic, memory or routing resources with other CHARE subsystems. A private key embedded within the PAU provides a public key-based authentication protocol with the DMS. Decrypted partial FPGA configurations for hardware plug-ins are not revealed outside the configuration firewall or even to PAU software. External flash memory stores encrypted partial bitstreams received from the DMS, with just-in-time decryption of bitstreams transferred to the FPGAs ICAP. For the sake of both speed and security, a hardware-implemented flash memory controller controls the ICAP.

4.5 Implementation Details and Results

CHARE-CR is implemented on the ML510 evaluation platform offering a versatile Virtex-5 FXT platform for rapid prototyping and system verification. In addition to the more than 130,000 logic cells, over 10,700 Kb of block RAM, dual IBM PowerPC 440 processors, and RocketIO transceivers available in the FPGA, the ML510 provides an on-board Ethernet MAC PHY, DDR2 memory, multiple PCI bus slots, and standard front panel interface ports within an ATX form factor motherboard. The ML510 platform is based on the Xilinx Virtex-5 XC5VFX130T FPGA. A custom board interfaces two RocketIO gigabit serial transceivers on the ML510 to two USRP2 modules, for optional multiple-input and multiple-output (MIMO) CR clients. The USRP2 serves as the RF interface for the CR client. Figure 4.10 demonstrates the prototype resources used in this work.

A CHARE-CR prototype is developed with the aid of the XPS, Integrated Synthesis Environment (ISE), and Software Development Kit (SDK) tools enabling hardware-software co-design of CHARE and CR components. The XPS tool helps the designer to easily build, connect and configure embedded processor-based systems. The CHARE-CR prototype block diagram is shown in Figure 4.11 with two strictly separate PowerPC 440 processors. One of the processors is operated by the Linux OS and hosts the CR software including the system strategy and policy reasoner functionalities while the second PowerPC implements the PAU module of the configuration firewall. Both processors run at 400 MHz clock frequency and connected to memory devices and I/O peripherals via completely separate PLBs running at at a 100 MHz clock frequency.

This architecture realizes the separation of resources security principle by separating the application processor from the reconfiguration processor and resources. The Linux PowerPC processor is equipped with 1 GB of DDR2 RAM memory and a floating point unit facilitating memory intensive mathematical operations extensively deployed in DSP and communication applications. The SRC is developed as a custom peripheral IP attached to the Linux PowerPC PLB in a slave configuration and provided with a peripheral interrupt service. Reconfiguration requests are directly written by the CR processor to the SRC peripheral address range and stored in dedicated,

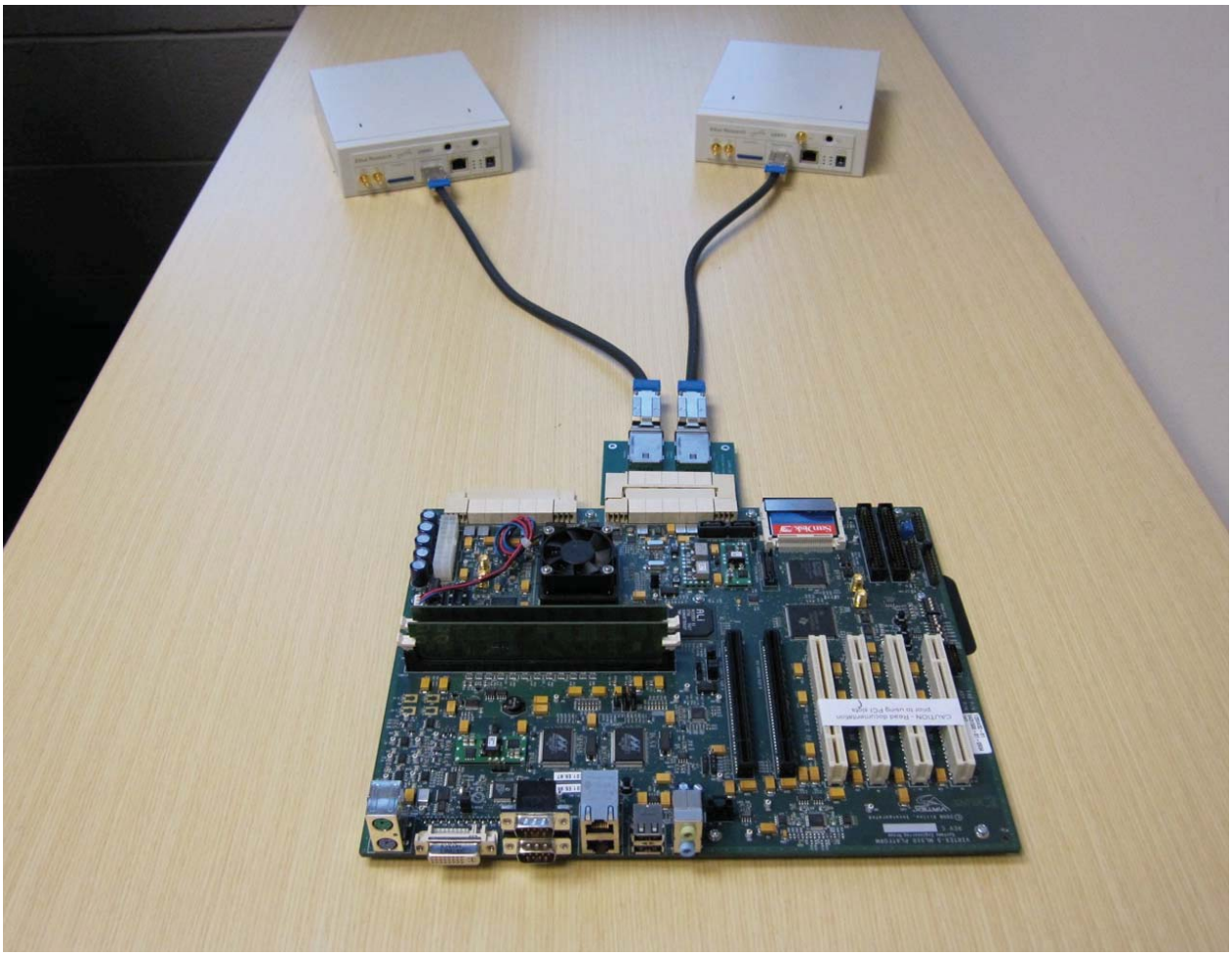


Figure 4.10: Prototype resources.

software controlled registers. Reconfiguration decisions are communicated to the CR processor through the assertion of predefined interrupt signals initiating service routines to read SRC dedicated software registers. Figure 4.12 shows the separate regions allocated to these components on the Xilinx Virtex-5 FX130T FPGA used in the initial prototype, with controlled communication between regions. There are few static routes crossing the dynamic region since most of the I/O signals connected to the two processors reside in the leftmost I/O banks. Roughly 70% of the chip area, including all 320 DSP slices and the majority of the Block RAM, is allocated to the reconfigurable plug-in region.

The configuration firewall PowerPC processor is interfaced to a set of peripherals enabling various

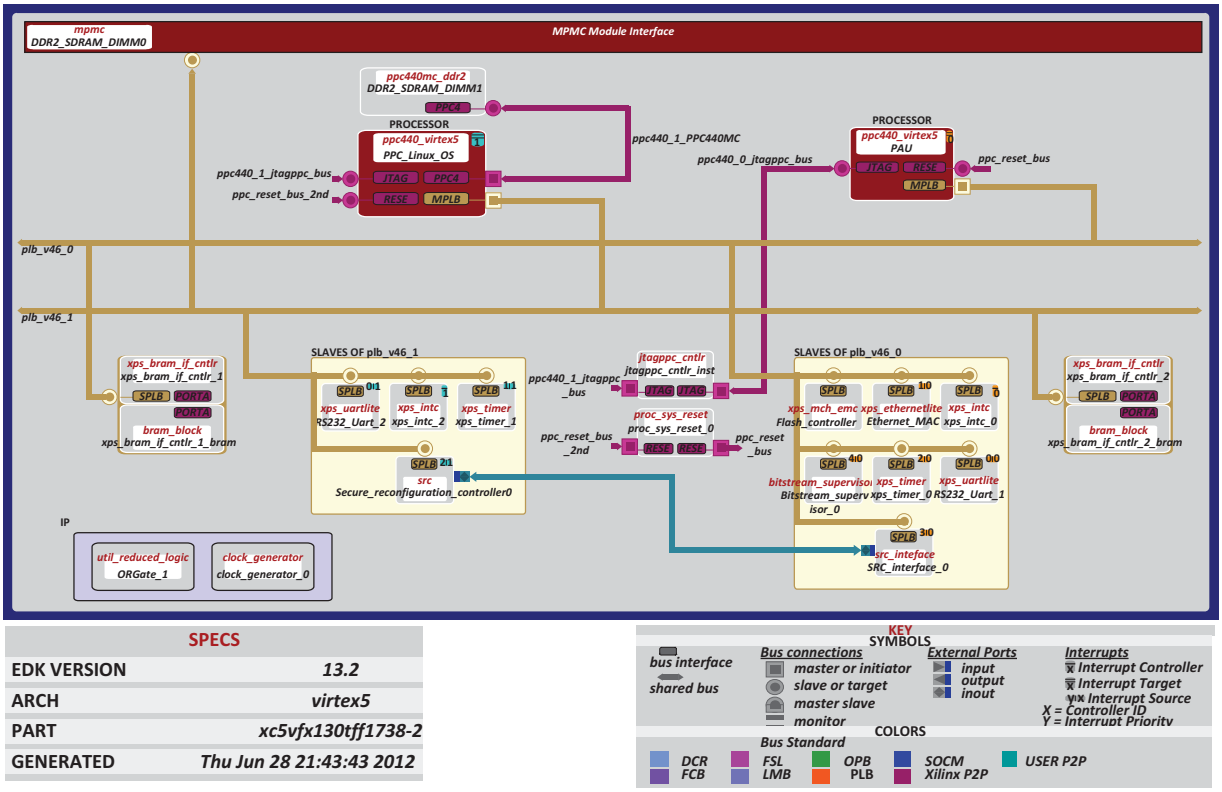


Figure 4.11: CHARE-CR prototype block diagram.

functionalities described in Section 3.2. The AES decryption, HMAC authentication, and ICAP controller modules are trusted hardware components combined in a single peripheral called the bitstream supervisor. An SRC interfacing peripheral connects the SRC to the configuration firewall through a custom I/O bus. The SRC interfacing peripheral can be developed as a partially reconfigured block within the bitstream supervisor peripheral. Custom registers are developed in the SRC and SRC interfacing modules to communicate configuration commands and status between the two peripherals without resorting to add a shared communication facility between the separate processors. The bitstream supervisor is connected to the PAU PLB in a slave configuration and equipped with a peripheral interrupt service. Other peripherals connected to the PAU processor include an Ethernet media access controller and a flash controller providing network communication and bitstream storage functionalities. The PAU processor fetches approved reconfiguration bitstreams from the flash storage or downloads them from a remote DMS and writes them to a

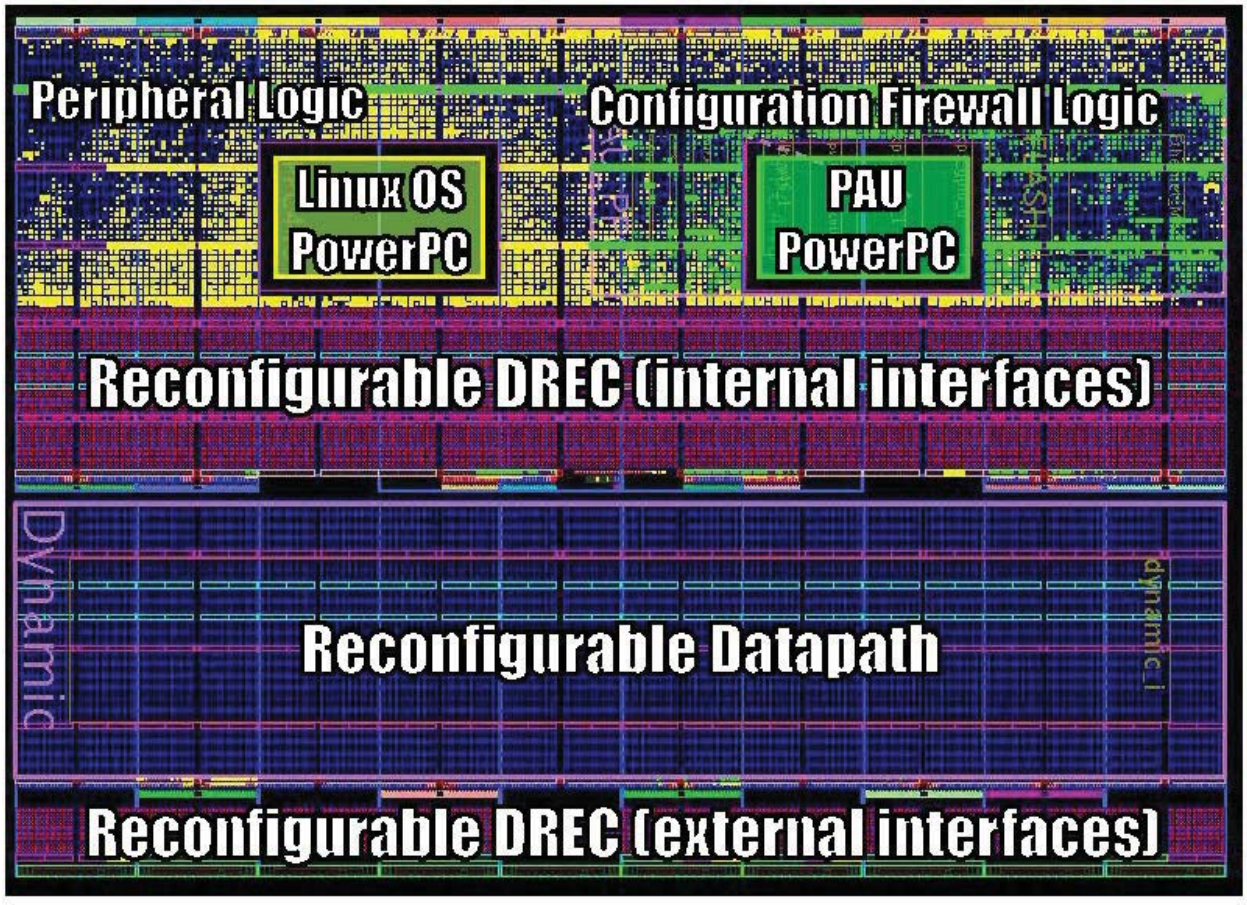


Figure 4.12: CHARE floorplan on a Xilinx Virtex-5 FX130T FPGA.

dedicated FIFO buffer within the bitstream supervisor peripheral. A predefined interrupt service communicates the reconfiguration status back to the PAU processor which, in turn, writes them to the appropriate registers in the SRC interfacing component. Implementation details and results of the SRC and bitstream supervisor peripherals are presented in the following sections.

4.5.1 Secure Reconfiguration Controller

The SRC is a hardware peripheral interfaced to an application processor running the Linux OS. Figure 4.13 illustrates the block diagram of the SRC peripheral module. The SRC peripheral performs three main roles: receiving update requests and reasoning decisions from the application proces-

sor, checking the request-response pair validity and translating valid request physical parameters to the corresponding parameter changes and/or reconfiguration commands, and interacting with the configuration firewall to send reconfiguration commands and receive reconfiguration status. Two sets of software registers called the request parameter and SRC reply registers communicate DSA requests and SRC responses between the peripheral and the application processor with the aid of an interrupt source controller module. Two sets of registers called the reconfiguration parameter and reconfiguration status registers communicate SRC reconfiguration commands and configuration firewall responses between the SRC peripheral and the bitstream supervisor peripherals. The SRC module contains two blocks: the policy enforcer module implementing functionality described in Section 4.2.1 and a mapping module translating approved requests into parameter changes or reconfiguration requests. An FSM controller manages the operation of the SRC peripheral and coordinates between different components as demonstrated by Figure 4.13.

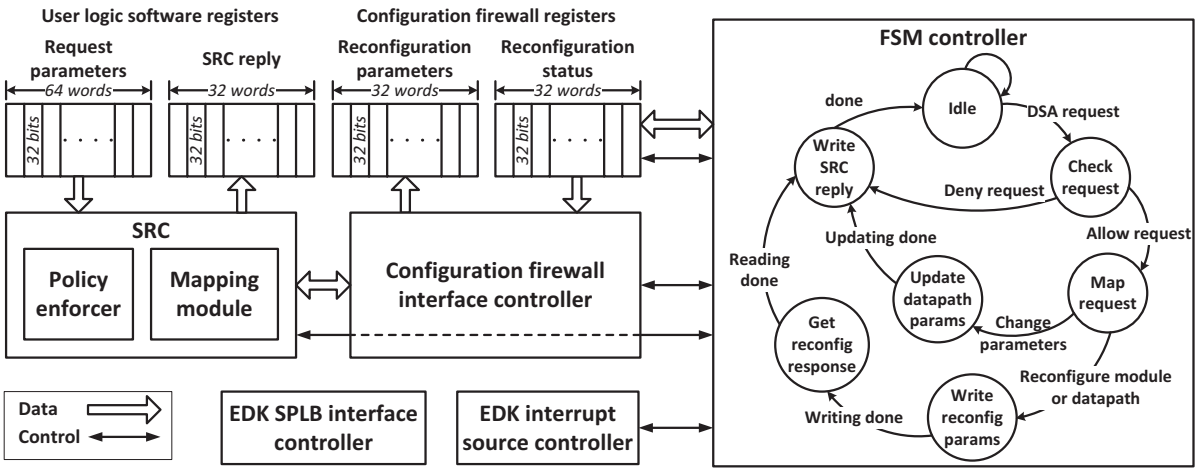


Figure 4.13: SRC peripheral block diagram.

In order to illustrate the SRC implementation details, we provide an example of a DSA policy as described in CoRaL and the corresponding hardware implementation of the SRC module enforcing this policy. This example is acquired from CoRaL description article [47], and applies to CR devices operating in the Radar "S" band according to a regulating DSA policy. A CR device senses channel power within the intended operating frequency, and can access the channel as needed based on the DSA policy. Transmit power is regulated as a function of the received power. The CoRaL

description of the Radar band access policy is shown in Figure 4.14 [47]. Specific rules for CR radios operating in this band are:

1. Transmission is allowed within 3100 to 3300 MHz frequency band.
2. The radio's sensing threshold must be less than -110 dBm in a 100 KHz channel.
3. The device should be equipped with an opportunity identification mechanism with the following specifications:
   (a) The device should sense received power within the intended frequency channel of no more than 100 KHz width. There must be a look-through interval with a minimum of 3 seconds, and a dwell time of 500 μs. Look-through is the period between successive sensing operations, and the dwell time is the time over which energy is accumulated and integrated by the sensor.
   (b) The maximum authorized transmit power is a function of the sensed channel power:
      i. The maximum transmit power is 50 mW if the peak sensed power is less than -100 dBm.
      ii. The maximum transmit power is 10 mW if the peak sensed power is between -80 and -100 dBm.
      iii. Transmission is prohibited if the peak sensed power is greater than -80 dBm.
   (c) To authorize transmission or power level increasing, the sensed power should not exceed the prescribed bound for at least 5 minutes.
4. The device should be equipped with a mechanism of interference-limited use of the spectrum opportunity having the following properties:
   (a) Power outside the radio channel should be less than 1% of the transmit power.
   (b) The maximum continuous on time is 1 second and the minimum off time is 100 ms.
   (c) The device response time to sensor changes should be less than 1 second.

```
policy RadarBandSensingBased is

    include XGTypes.xg;

    /* Predicate declarations */

    const device_ok : Pred;
    const sensing_ok : Pred;
    const usage_ok : Pred;
    const frequency_ok : Pred;
    const sensing_threshold_ok : Pred;
    const channel_width_ok : Pred;
    const look_through_interval_ok : Pred;
    const dwell_time_ok : Pred;
    const peak_received_power_ok : Pred;
    const signal_age_ok : Pred;
    const outside_band_leakage_ok : Pred;
    const max_ontime_ok : Pred;
    const min_offtime_ok : Pred;
    const channel_closing_time : Pred;

    /* Policy Rules */
    allow if device_ok and sensing_ok and usage_ok;

    device_ok if
        frequency_ok and sensing_threshold_ok
        and channel_width_ok;

    sensing_ok if
        look_through_interval_ok and dwell_time_ok and
        peak_receieved_power_ok and signal_age_ok;

    usage_ok if
        outside_band_leakage_ok and max_ontime_ok
        and min_offtime_ok and channel_closing_time_ok;

    frequency_ok if carrier_frequency in {3100..3300};
    sensing_threshold_ok if sensing_threshold =< -110;
    channel_width_ok if channel_width =< 100;

    look_through_interval_ok if look_through_interval =< 3;
    dwell_time_ok if dwell_time =< 500;

    peak_received_power_ok if
        (peak_received_power < -100 implies max_tx_power=50)
        and (peak_received_power >= -100 and peak_recieved_power < -80
        implies max_tx_power=10)

    disallow if peak_received_power >= -80;

    signal_age_ok if signal_age >= 5;

    outside_band_leakage_ok if outside_band_leakage < 0.1;

    max_ontime_ok if max_ontime =< 1;
    min_offtime_ok if min_offtime >= 100;
    channel_closing_time_ok if channel_closing_time =< 1;

end
```

Figure 4.14: CoRaL Description of Radar “S” band access policy

The policy enforcer for the Radar "S" band policy mainly performs device-, sensing-, and usage-oriented checks, where the transmission is allowed if the three checks are OK. Transmission request parameters are the carrier frequency and maximum transmitted power. Sensing parameters are the peak received power, signal age, look through interval, and dwell time. Device parameters are the CR sensing threshold and channel width. Usage parameters are the max on time, minimum off time, channel closing time, and outside band leakage. Each of these parameters is treated as an input to the policy reasoner which evaluates transmission requests based on these inputs and the embedded policy rules.

Figure 4.15 illustrates snapshots from the SystemVerilog code describing the SRC enforcing the Radar policy. Inputs to the SRC module are the `request_ready` and `reasoner_decision` signals, and the outputs include the `transmission_decision` and `reasoner_check` signals. Upon a transmission request, the SRC reads the software parameters registers written by the application processor, performs DSA rule checks, compares the enforcer's decision with the reasoner's decision, and initiates the reconfiguration operation for allowed requests. DSA policy checks are translated from the CoRaL policy description into SystemVerilog assertions according to translation rules presented in Section 4.3. Figure 4.15 shows selected snapshots from the HDL description of some DSA policy rules presented in Figure 4.14. Resources needed by the SRC module include 15 32-bit registers, 15 flip-flops, and 10 comparators. Resource overhead introduced by the SRC module is negligible when compared to the total resources offered by modern FPGAs.

In order to evaluate the SRC security, we developed a testbench that generates different values for input parameters and signals, monitors internal signals and predicates, and displays output signals and states. Different parameter value sets representing four transmission requests are generated of which two conform to the policy and two do not conform. For non-conforming requests, only a single parameter is generated to deviate from the sanctioned range to demonstrate the SRC's ability to detect minor attacks. Figure 4.16 illustrates simulation results for the generated stimulus. In this testbench, a signal representing the reasoner's decision is generated to indicate two compromised decisions and two valid decisions. Compromised decisions are detected by comparing the

```
module SRC(input clk, rst, req_ready,
  //the policy reasoner decision
    reasoner_decision,
  //transmission request parameters
    [31:0] carrier_frequency,
    [31:0] max_tx_power,
  //sensing parameters
    [31:0] peak_received_power,
    [31:0] signal_age,
    [31:0] look_through_interval,
    [31:0] dwell_time,
  //device parameters
    [31:0] sensing_threshold,
    [31:0] channel_width,
  //usage parameters
    [31:0] max_ontime,
    [31:0] min_offtime,
    [31:0] channel_closing_time,
    [31:0] outside_band_leakage,
    output reg allow,reasoner_check);

//Some predicate declaration
typedef enum {yes, no} Pred;
Pred device_ok;
Pred sensing_ok;
Pred usage_ok;
Pred frequency_ok;
Pred peak_received_power_ok;

//Some policy rules translated into
   SystemVerilog processes
always @ (posedge clk) begin
  if (req_ready) begin
    if ((device_ok) && (sensing_ok)
       && (usage_ok)) begin
      allow <= 1'b1;
    end
    else begin
      allow <= 1'b0;
    end
  end
  else begin
    allow <= 1'b0;
  end
end

always @(posedge clk iff req_ready)
   begin
  if (allow == reasoner_decision)
     begin
    reasoner_check <= 1'b1;
  end
  else begin
    reasoner_check <= 1'b0;
  end
end

always @ (posedge clk iff req_ready)
   begin
  device_ok <= no;
  if ((frequency_ok) && (
     sensing_threshold_ok) && (
     channel_width_ok)) begin
    device_ok <= yes;
  end
end

always @ (posedge clk iff req_ready)
   begin
  frequency_ok <= no;
  if ((carrier_frequency >3100)&&(
     carrier_frequency <3300)) begin
    frequency_ok <= yes;
  end
end

always @ (posedge clk iff req_ready)
   begin
  peak_received_power_ok = no;
  if (((peak_received_power < -100)
     && (max_tx_power == 50)) || ((
     peak_received_power >= -100) &&
     (peak_received_power < -80) && (
     max_tx_power == 10))) begin
    peak_received_power_ok <= yes;
  end
  else if (peak_received_power >=
     -80) begin
    peak_received_power_ok <= no;
  end
end

endmodule
```

Figure 4.15: Snapshots from the SRC SystemVerilog code for the Radar "S" band DSA policy

reasoner's decision to the enforcer's decision. The `reasoner_check` output signal is asserted to logic "1" if both decisions are identical. Computation time of the reasoner's decision represented by the `allow` output is two clock cycles as depicted by the third and forth requests. In a more exhaustive testbench, we generate different combinations of device parameters for which more than one are out of authorized ranges to test the SRC's ability to detect different attacks. Simulation results indicate that the SRC is able to detect attacks targeting different parameters in several combinations.

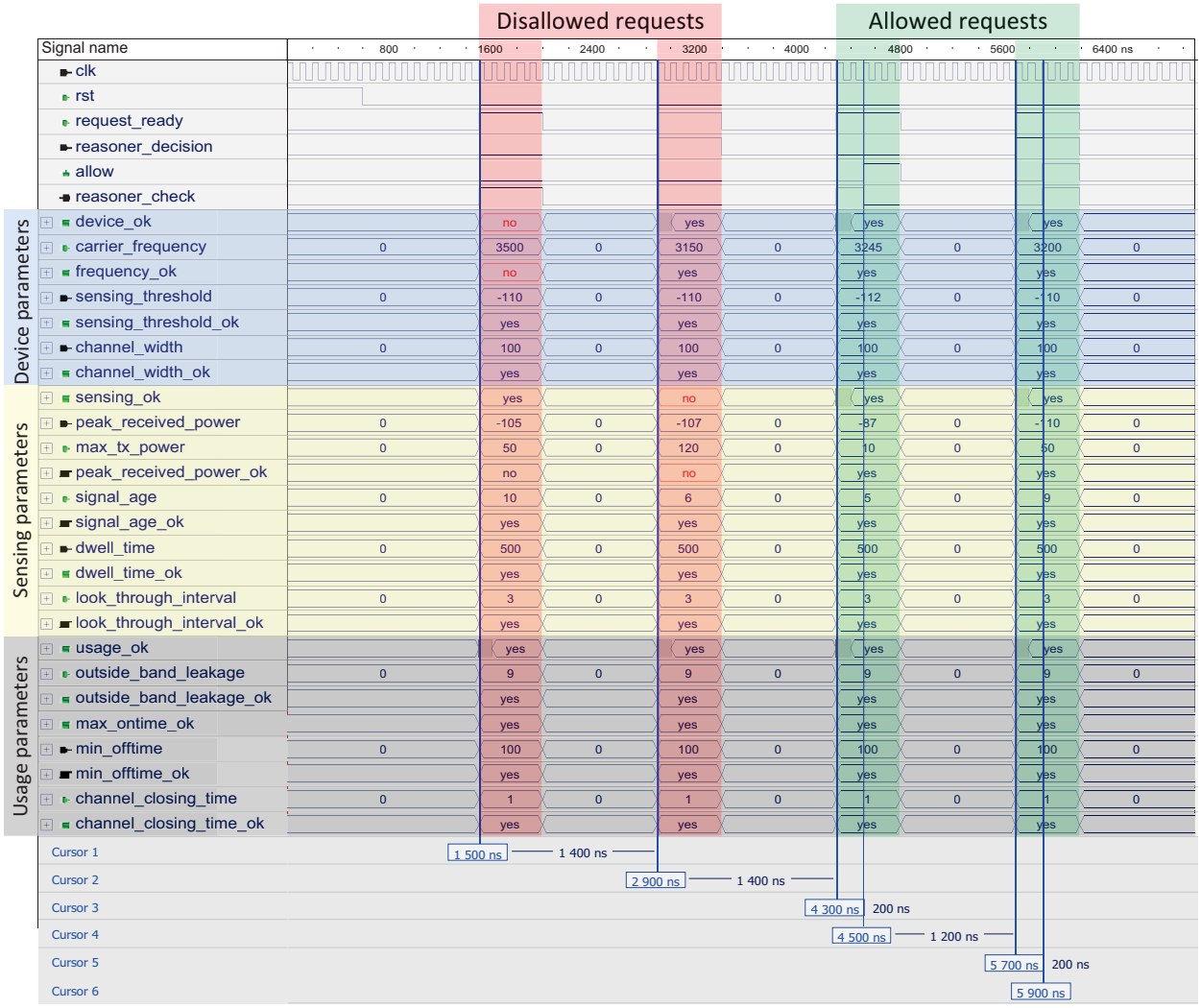


Figure 4.16: Simulation output for the SRC module enforcing the Radar "S" band DSA policy.

4.5.2 Configuration Firewall

The configuration firewall is composed of software- and hardware-based components interacting under the control of the PAU processor to actualize SRC reconfiguration commands and ensure bitstream integrity. The PAU processor is connected to a flash memory, a remote DMS, the SRC via several interface peripherals and controllers operated by a simple, standalone, low-level software layer executing custom software programs, APIs, and drivers written in the C programming language. The standalone software layer provides access to basic processor features such as caches, interrupts and exceptions as well as basic features of the hosted environment, such as standard input and output, profiling, abort and exit. The standalone software layer is an OS alternative to simplify the PAU software verification and limit the memory requirements. The PAU processor has 512 KB of dedicated BRAM memory accommodating the standalone software execution requirements.

Major software programs included in the standalone layer are flash memory read and write APIs, the LwIP protocol stack, the trivial file transfer protocol (TFTP), and custom-developed read, write, and control APIs and interrupt service routines for the bitstream supervisor. The LwIP is a widely used, open source TCP/IP stack designed for embedded systems with the goal of reducing resource usage. The TFTP is a simple, unsecured file transfer protocol implemented using a very small amount of memory to enable bitstream transfers from the DMS to the configuration firewall. PRM bitstreams are generated, authenticated, and encrypted in the remote DMS on a block-by-block basis, and a unique ID is assigned to each PRM. The block size can be parametrized according to a desirable resource-speed trade-off considering the buffering requirements, authentication standard, and the decryption and authentication module architecture. Requested bitstreams are downloaded from the DMS and stored in the flash memory. The PAU processor does not have access to the decryption keys generated by the hardware PUFs. Bitstream supervisor APIs initiate and control writing bitstreams to the dedicated FIFO buffer and reading reconfiguration commands and status from the corresponding registers.

Figure 4.17 illustrates the bitstream supervisor peripheral block diagram with a simplified version of the module FSM controller. The AES decipher, HMAC module, PUFs, comparator, ICAP, and

FSM controllers are user-developed hardware blocks while the FIFO buffers, SPLB interface, and interrupt source controllers are Xilinx IP cores automatically generated by the XPS tool. All components are developed and optimized for the 32-bit bus architecture of the PAU processor. Most FIFO buffers and registers are synchronous dual port BRAMs enabling control signal exchange and data flow from the processor memory to the FPGA configuration memory. FIFOs also enable data flow through several computation modules running at different data rates.

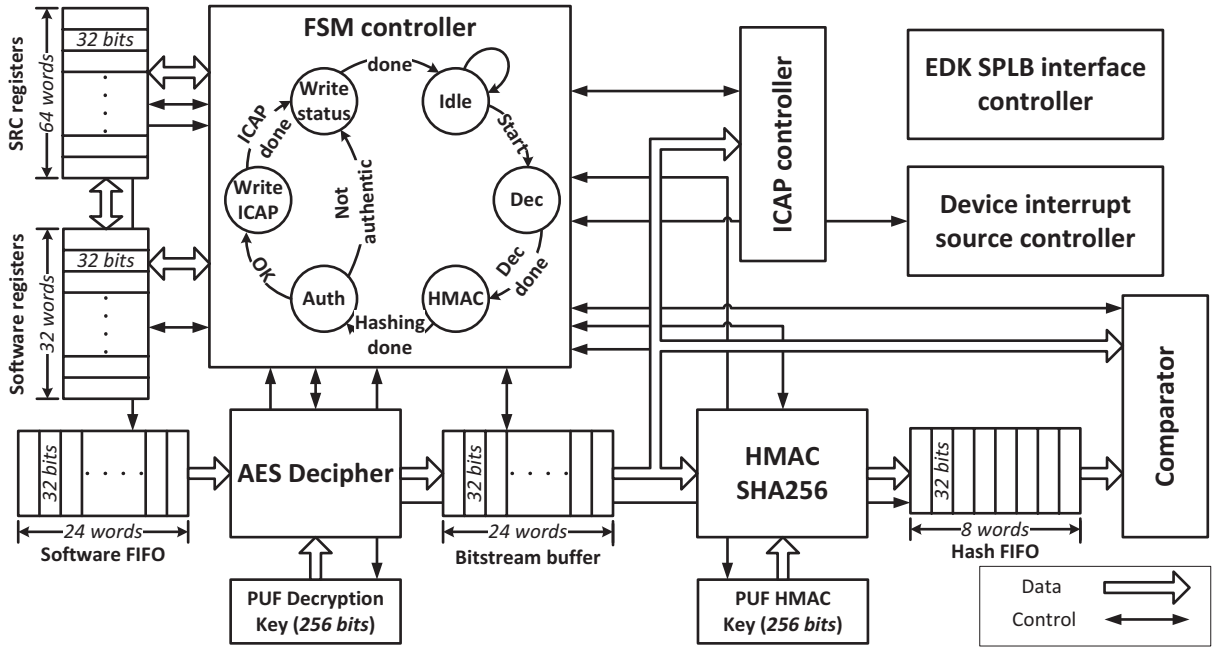


Figure 4.17: Bitstream supervisor peripheral block diagram.

SRC registers enables exchanging configuration commands and status between the bitstream supervisor and SRC peripherals. The bitstream supervisor has write access to a set of the SRC registers which are read by the SRC and the SRC has write access to a set of the SRC registers which are read by the bitstream supervisor. Software registers enable exchanging configuration commands and status between the bitstream supervisor and the PAU processor. A software FIFO stores PRM bitstream blocks written by the PAU processor in preparation to the decryption process. The FIFO depth depends on the bitstream block size parameter which is assigned to 24 words in our design to satisfy resource-speed constraints. A bitstream buffer stores decrypted blocks to be authenticated

with the HMAC module. A Hash FIFO stores hash values computed by the HMAC in order to be validated with the block hash.

The FSM controller initiates and manages data flow between different computing modules and storage elements based on a set of control words in the SRC and software registers and control signals from various system components. The SRC writes reconfiguration commands and PRM IDs to a set of software registers continuously monitored by the FSM controller during the idle state. The controller, in turn, interrupts the PAU processor to read the PRM IDs. The PAU fetches the PRM from the local storage and writes bitstream blocks to the software FIFO. The FIFO full signal starts the AES decryption operation whereas the empty signal terminates the block decryption. Unlike the software FIFO, the bitstream buffer is a content addressable memory enabling data buffering rather than data flow.

The FSM controller initiates the HMAC module operation after receiving the decryption completion signal. The SHA256-based HMAC computes the block hash values from the first 16 words of the bitstream buffer — the size of a single SHA256 block is 512 bits or 16 words — and writes the 8-word hash value to the hash FIFO. The hash FIFO full signal initiates the comparator operation to validate the block integrity by comparing the computed hash to the attached hash stored in the bitstream buffer. Authentic blocks are sourced to the ICAP controller while non-authentic blocks terminates the reconfiguration process. The FSM controller writes reconfiguration status to a software register and interrupts the PAU processor to run a service routine to read this status. Failing to authenticate any of the bitstream blocks makes the PAU write "reconfiguration failed" status to an SRC register, whereas writing all PRM blocks successfully to the ICAP causes the PAU to write "reconfiguration done" status to the SRC register.

AES Decryption Module

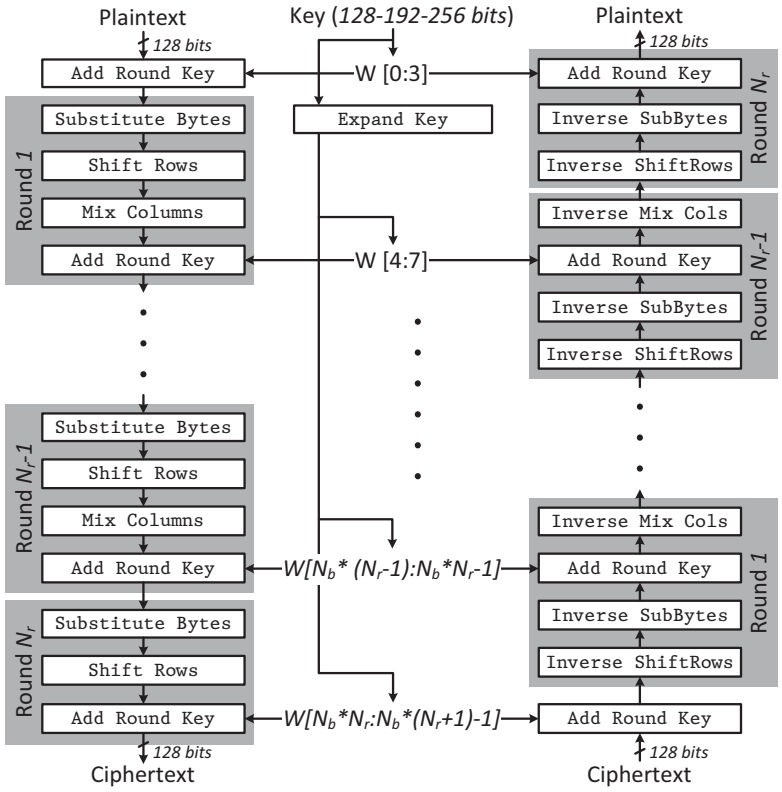
AES is a symmetric-key block cipher based on a substitution-permutation network in which the block length is 128 bits and the key length can be 128, 192, 256 bits [127]. AES operates on a 4x4 column-major matrix of bytes called the state, and most AES calculations are performed in a

special finite field. The key size determines the number of transformation rounds N_r that convert the plaintext into the ciphertext and vice versa. Each round consists of several processing steps including one that depends on the encryption key. All encryption transformations are reversible to enable decryption. Figure 4.18(a) illustrates the AES algorithm flow for arbitrary number of rounds N_r. A key expansion algorithm generates a number of sub-keys required by the AES cipher and decipher rounds. Each encryption round consists of four transformations operating on input states:

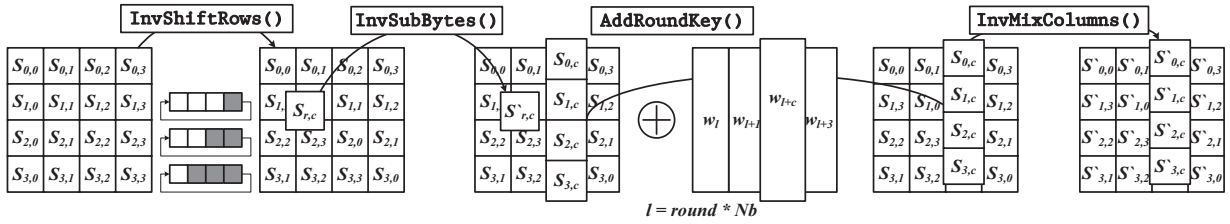
- `Substitute Bytes` transformation is a non-linear byte substitution that operates independently on each byte of the state using a substitution table (S-box).
- `Shift Rows` is a transposition step where each row of the state is cyclically shifted a certain number of steps.
- `Mix Columns` is a mixing operation operates on the state column-by-column performing modulo multiplication by a constant in Galois Field 2^8.
- `Add Round Key` is a simple bitwise XORing operation adding a state column to the corresponding sub-key word generated by the key expansion algorithm.

These transformations are reversed in the decipher round as depicted in Figure 4.18(b). The key expansion round adopts `Substitute Bytes` transformation, cyclic shift, and predefined round constants to generate 4 sub-key words from the previous round sub-keys as shown in Figure 4.18(c).

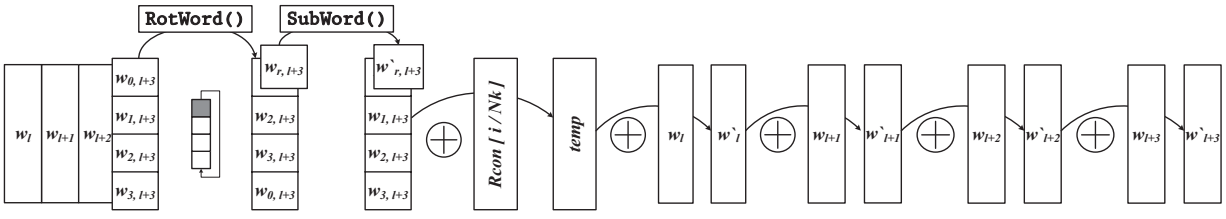
AES can operate in one of several modes for which some of them are non-feedback modes such as electronic codebook (ECB) while the others are feedback modes such as cipher block chaining (CBC). Non-feedback modes offer less security but can achieve more speed-up by processing multiple block simultaneously. In the CBC feedback modes, input to the cipher is constructed by feeding back the previous ciphertext to be XORed with the new plaintext. In our design, we implement the AES decipher in the CBC mode which can offer more security at the expense of the achievable speed-up. Many architectural and algorithmic optimizations investigating area-speed trade-offs have been proposed for AES implementations in FPGAs. Since AES rounds



(a) AES algorith flow



(b) Decipher round.



(c) Key expansion round.

Figure 4.18: AES algorithm flow, decipher round, and key expansion round.

are identical, architectural optimization depends on either resource reuse with feedback to save area or resource duplication with pipelining to increase speed. Algorithmic optimization exploits the ability to implement individual AES transformations and functions in several ways and the possibility of sharing resources between some functionalities. A detailed explanation of the AES and modes of operation and architectural and algorithmic optimization approaches is provided in [59].

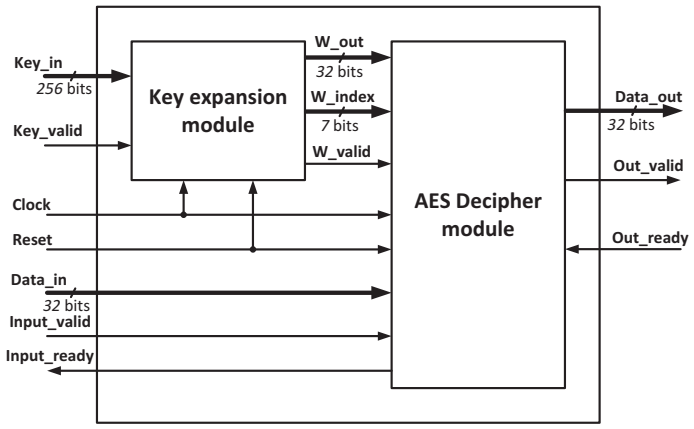
Our approach to implement the bitstream AES decipher relies on something arising in both the decipher and key expansion rounds. All round transformations can be performed on a word-by-word basis, where the word size is 32 bits, instead of a state-by-state basis as illustrated by Figure 4.18. This indicates that the basic building block of AES can be word transformations rather than state transformations to accommodate 32-bit processor architectures. Applying such an approach to implement a hardware AES cipher/decipher can result in either a very high-speed architecture by reducing the critical path of the pipeline building units or a very small-area architecture by limiting resource requirements to a small building block. Since the security is the main goal of this work, we decide to implement the AES in CBC feedback mode to gain more security. Speed optimizations will not result in significant improvements for feedback modes of operation. Moreover, the ICAP maximum operating frequency is 100 MHz which is considered the main bottleneck in the bitstream supervisor module. Therefore, we decide to implement the 256-bit key, area-optimized AES bitstream decipher and present the implementation results. In this work we do not discuss side-channel attacks on block ciphers and their potential countermeasures.

Figure 4.19(a) illustrates the AES decryption module block diagram. Module inputs include a 256-bit key and valid signal generated by the PUF, 32-bit data input; module outputs include 32-bit data out. Valid and ready signals perform simple I/O handshaking procedures. The decryption module comprises two building blocks: the key expansion and decipher modules. These modules are composed of a set of registers, computational elements, multiplexers and demultiplexers, indexers, and controllers as depicted in Figure 4.19. Without delving into specific implementation and operational details, the main points demonstrating how the resource usage is optimized with the presented architecture are:

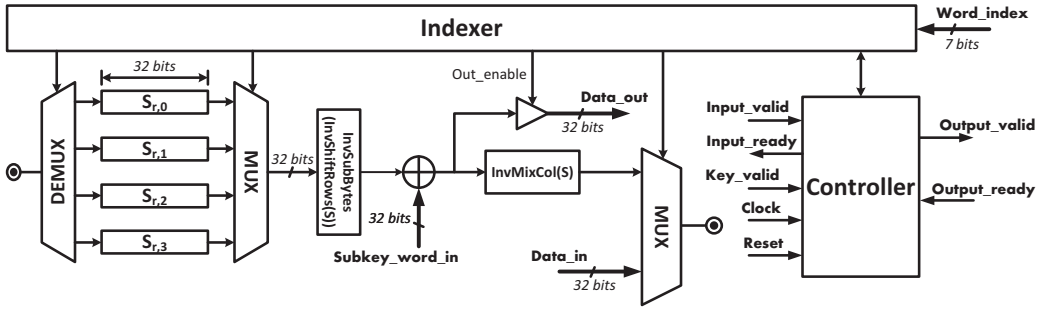
- Only a single round is implemented for both the key expansion and decipher modules.
- This single round is reused $4 * N_r$ times with feedback to compute the 128-bit plaintext from a given 128-bit ciphertext, where the round computation time is 4 clock cycles. For a 256-bit key, the decryption period of 128-bit ciphertext equals 56 clock cycles.
- Sourcing and sinking of I/O data are performed on a word-by-word basis.
- In the decipher block, buffering requirements correspond to the state size whereas computation resources are only developed for a single word as shown by Figue 4.19(b).
- In the key expander, buffering requirements correspond to the key size whereas computation resources are only developed for a single round as shown by Figure 4.19(c). The input key to this module is not the encryption key but the last eight sub-key words generated by the key expansion algorithm.
- Multiplexers select the appropriate word to be processed from various registers while demultiplexers select the appropriate register to store the fed-back computation result.
- Indexers incorporate multiple counters controlling multiplexer and demultiplexer selector and output enable signals according to the active round index.
- Controllers are mainly composed of FSMs managing I/O interfaces and coordinating the internal module operation in collaboration with the indexers.

HMAC-SHA-256 Authentication Module

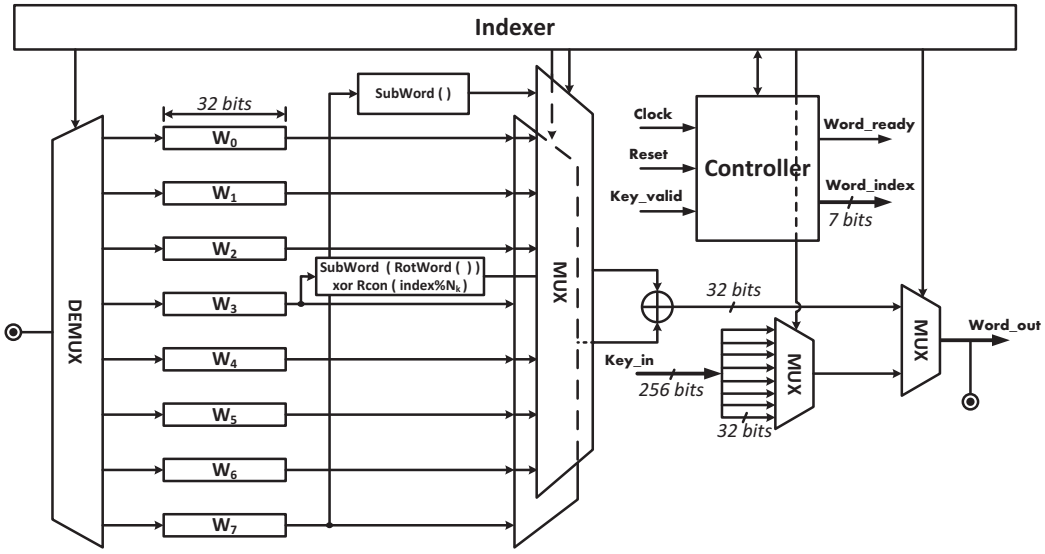
A cryptographic hash function is a function taking an arbitrary block of data called the message and returns a fixed-size bit string called the hash value or message digest, such that any change to the message will result in different hash value. SHA-256 is a hash function that takes an arbitrary message of a maximum length of $2^{64} - 1$ bits and produces a message digest of 256-bit



(a) AES decryption block diagram



(b) AES decipher module.



(c) Key expansion module.

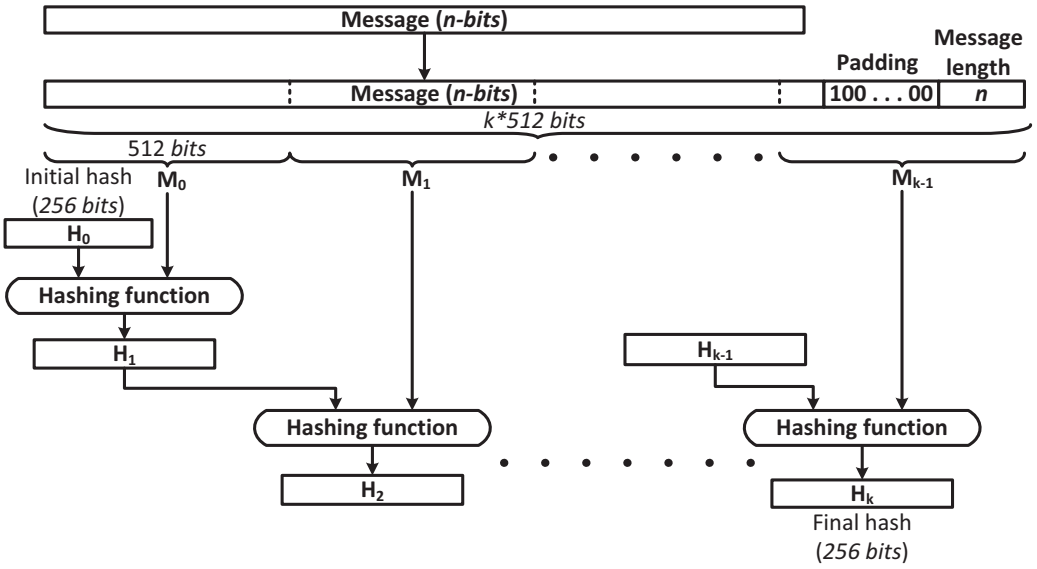
Figure 4.19: AES decryption block diagram, decipher module, and key expansion module.

length [128]. SHA-256 operations are performed on 32-bit words and include bitwise logical operations, bitwise rotation, and modulo addition. Figure 4.20 illustrates the SHA-256 algorithm flow and operations. Input message is padded and partitioned into 512-bit blocks for which a hashing function of 64 rounds is applied on a block-by-block basis. Message words are expanded to 64 words with a message schedule function as shown in Figure 4.20(b). Round function inputs are 8 words called the working variables, a message word, and a constant word K. The round function structure is depicted in Figure 4.20(c).

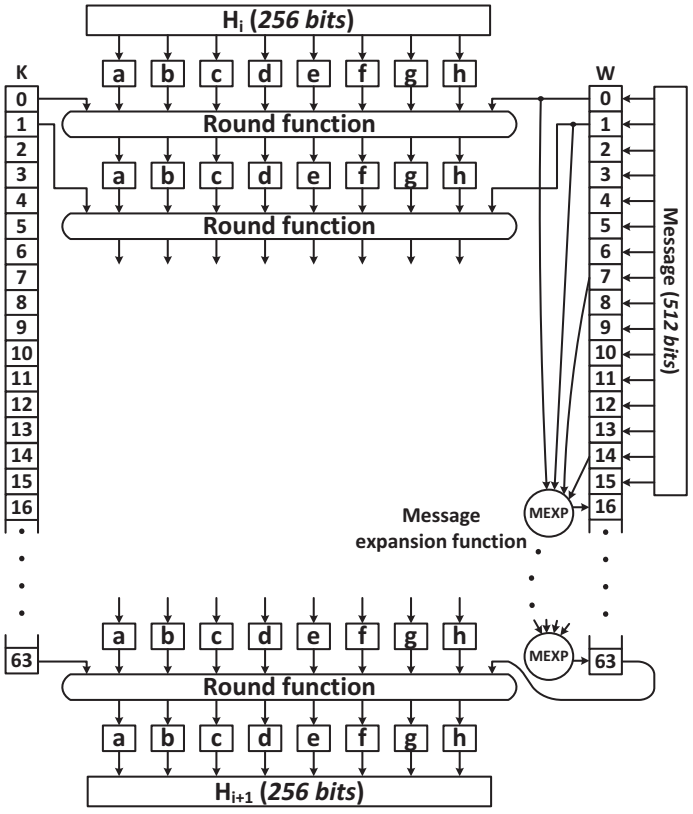
HMAC is an authentication algorithm computing a message authentication code using a cryptographic hash function in conjunction with a cryptographic key. HMAC can use any standard hash function and private key of an arbitrary length as shown in Figure 4.21. The key is padded or truncated to be of a standard size determined by the adopted hash function. The key is XORed with a predefined input pad and attached to the message to be hashed. The computed hash value is appended to the key XORed with an output pad, and this combination is hashed to produce a unique HMAC value for this particular pair of the key and message.

Many architectural optimizations investigating area-speed trade-offs have been presented for SHA-2 implementations in FPGAs [112, 162]. Loop unrolled and pipelined architectures can result in high-speed implementations while using the basic architecture with a single hashing round with feedback can result in reduced area implementations. In our development of the SHA-256 algorithm, resource limitations of the bitstream supervisor module are the main reason to investigate area-optimized architectures. Another factor that can help to reduce buffering requirements is limiting the message size which can be achieved by limiting the PRM bitstream block size to 512 bits, which is the basic size of the SHA-256 block.

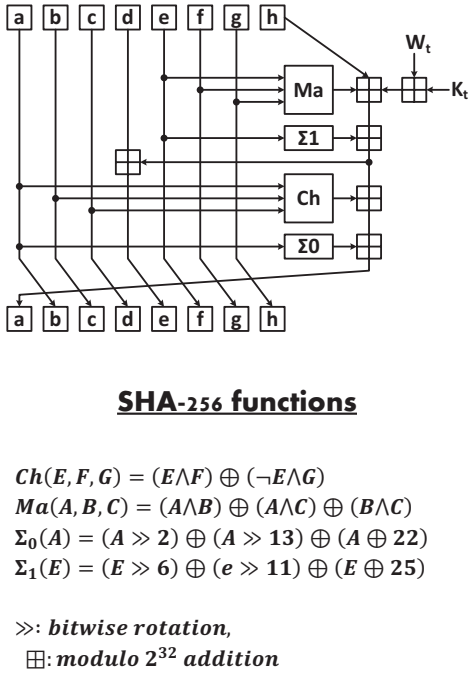
Figure 4.22(a) shows the block diagram of the area-optimized SHA-256 authentication module. The SHA-256 authentication module needs 64 clock cycles to hash a 512-bit block. Buffering requirements include 16x32-bit word registers storing input data and message schedule words, 8x32-bit registers buffering the working variables, and 64x32-bit ROM storing round constants. Computational resources are only required to implement the round function and the message ex-



(a) SHA-256 algorithm flow.



(b) SHA-256 hashing computation.



(c) SHA-256 round function.

Figure 4.20: SHA-256 algorithm description.

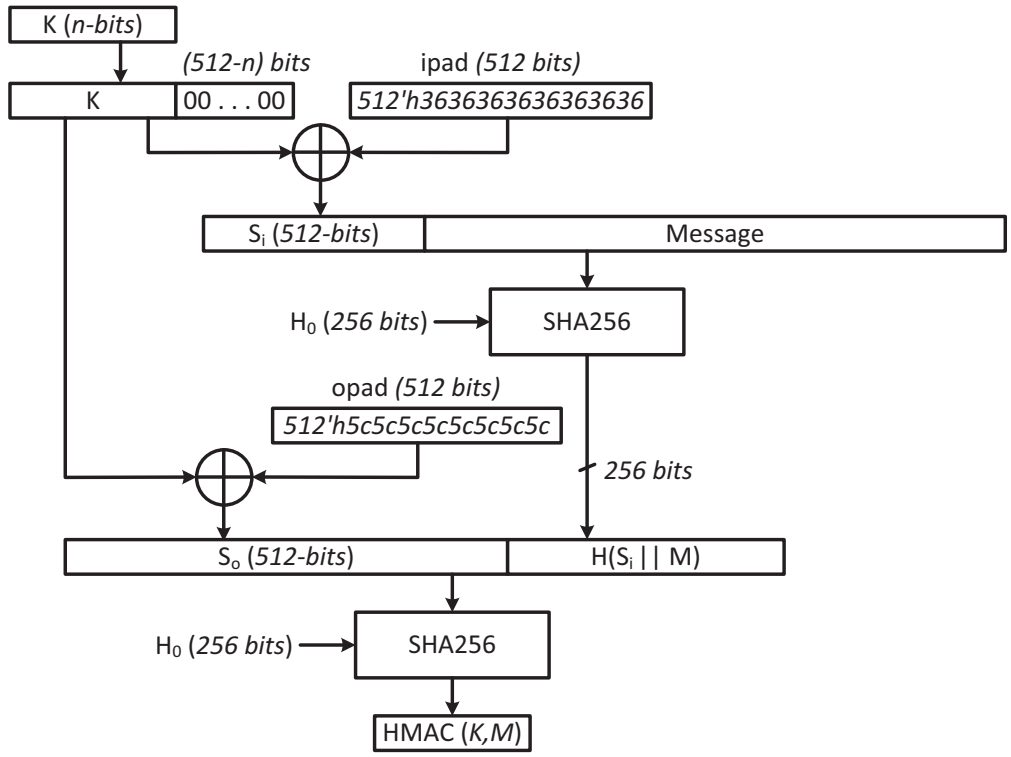
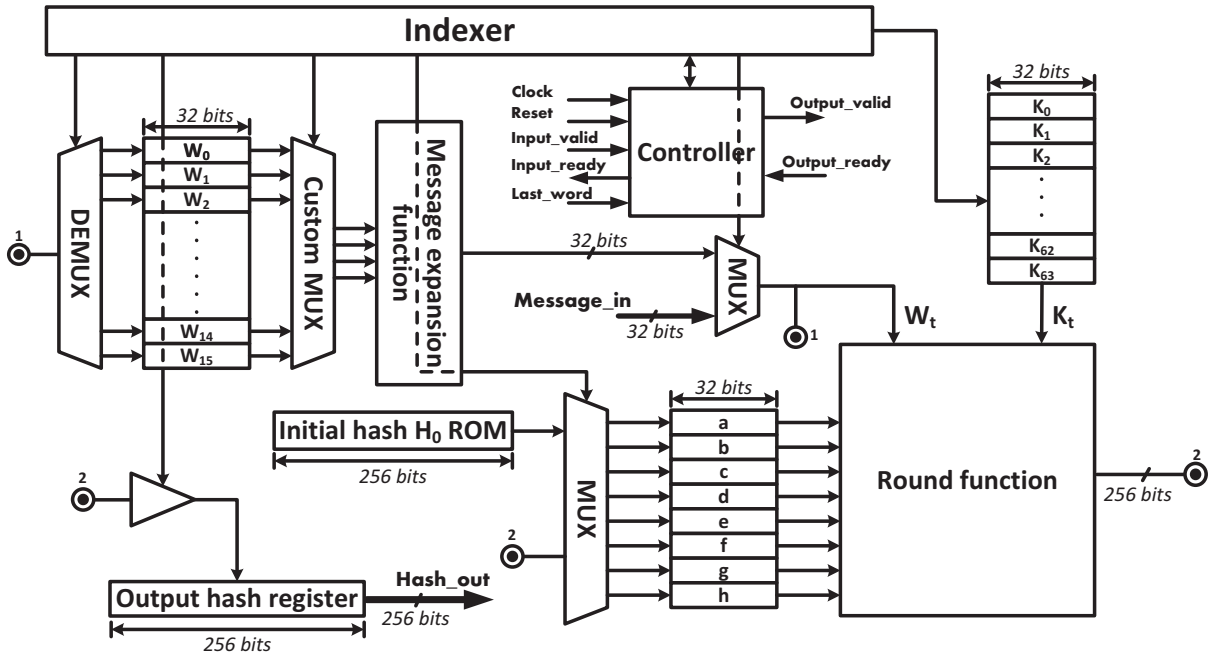


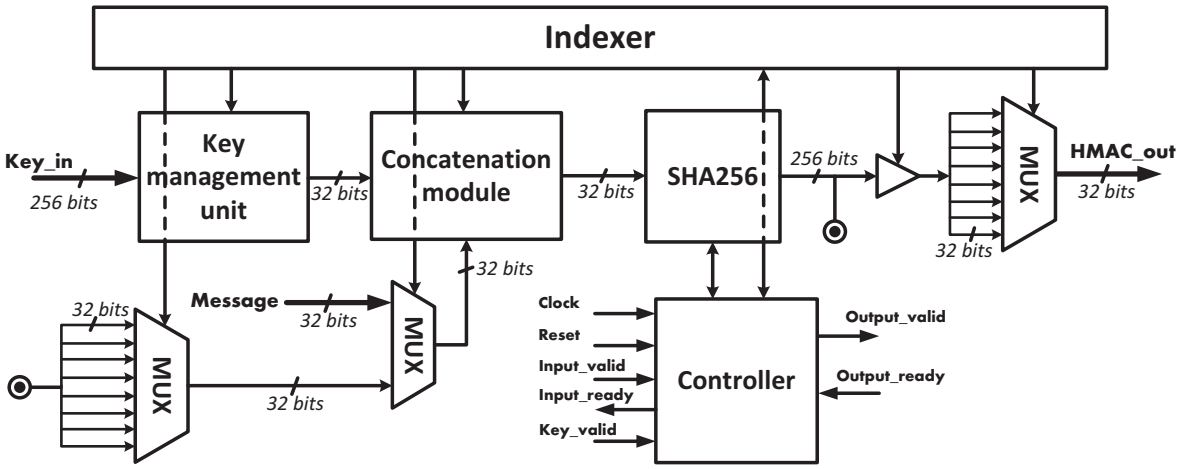
Figure 4.21: HMAC algorithm flow.

pansion schedule. The multiplexer selects the appropriate words to be processed from various registers while the demultiplexer selects the appropriate word register to store the computation result. The indexer controls multiplexer and demultiplexer selectors based on the active round index. The controller is an FSM managing I/O interface and coordinating the internal module operation in collaboration with the indexer. The HMAC module is implemented using a single instance of the SHA-256 module as depicted by Figure 4.22(b). The developed HMAC module requires 256 clock cycles to hash a 512-bit block.

Table 4.2 shows resource usage and maximum throughput achieved for the AES decipher and HMAC modules. It also illustrates total resource usage for the dual processors and the attached peripherals. Resource usage is depicted as the number of used LUTs, registers, and BRAMs and their ratio to the total resources available on the FPGA. Results demonstrate that the configuration firewall consumes less than 10% of the FPGA resources leaving more than 90% of the resources



(a) SHA-256 block diagram.



(b) HMAC block diagram.

Figure 4.22: HMAC and SHA-256 block diagrams.

to be utilized by the CR datapath. Total reconfiguration time equals the sum of the encryption, authentication, and writing to the ICAP time periods where the ICAP is a developed hardware controller with 32-bit data input and 100 MHz operating frequency.

Table 4.2: Resource usage and maximum throughput of the PAU processor and peripherals

	# of LUTs	**# of slice registers**	**# of BRAMs**	**Throughput**
AES	1314 (1.6%)	268 (0.0%)	0 (0.0%)	395 Kb/sec
HMAC	1464 (1.7%)	781 (0.95%)	0 (0.0%)	306 Kb/sec
PAU peripherals	7,341 (8.96%)	8,598 (10.5%)	241 (0.95%)	—
Total FPGA resources	81,920	81,920	25,280	—

4.6 Summary

In this chapter we provided an introduction to CR technology and an associated security challenge. In CR, all layers of the protocol stack are accessible for manipulation including the hardware layer. An untrusted software program controls platform reconfiguration which endangers radio spectrum integrity and primary users. We motivated the policy-based approach to protect CR devices against reconfiguration attacks, and presented existing techniques to enforce DSA policies in CR platforms 4.2. Such techniques solely rely on pure software architectures lacking basic security assurances. Most of the major advances in computer system security rely on hardware for enforcement. Data-intensive embedded platforms may require a hardware reorganization capability, introducing development complications and new types of security threats including zero-day attacks. We introduced the CHARE-CR framework to enhance the policy-based CR security in Section 4.4. CHARE-CR generates application-specific dynamic hardware plug-ins consisting of controller-wrapped datapaths. Software drivers are simplified by the controllers encapsulation and abstraction of the datapath structure. Hardware augments software monitoring and enforcement through the development of a controller incorporating datapath policy rules.

The CHARE framework provides secure reconfiguration control using the SRC and reconfiguration firewall. The main challenges associated with secure reconfiguration control in a CPS con-

taining untrusted components are validating update requests issued by untrusted components and authenticating update contents delivered via open communication channels. The SRC is responsible for validating update requests, and the reconfiguration firewall is responsible for authenticating update contents and securing the communication channel. We presented high-level architectures and prototype developments of the SRC and reconfiguration firewall in the context of the CR application in Section 4.5. The SRC is developed by translating DSA policies described in CoRaL, a domain-specific formal language based on first-order logic, into hardware synthesizable assertions. Although design automation is not the main focus of this work, we established automation feasibility by providing concrete examples showing the mapping from CoRaL to SystemVerilog HDL for all CoRaL syntactic constructs. As a result, CHARE simultaneously addresses the performance, developer productivity, and security requirements of high-throughput, reconfigurable platforms.

To illustrate how to map the conceptual CHARE-CR framework into a real reconfigurable platform, we presented the CHARE floorplan with an efficient placement of the RETC security anchors on a Xilinx Virtex-5 FPGA. We verified the CHARE-CR framework functional correctness by developing a testbench generating various update requests including allowed and compromised requests, and assessing the SRC decisions. The SRC is capable of validating transmission requests regardless of the software correctness. The SRC and reconfiguration firewall prototypes were evaluated in terms of overheads compared to the total resources of the FPGA evaluation platform. The SRC was developed for a CR device operating in the Radar "S" band enforced by a regulating DSA policy described in CoRaL. The hardware SRC overheads are negligible compared to the total platform resources and gained security. We utilized state-of-the-art cryptography and authentication standards and practices to build the reconfiguration firewall. Implementation results demonstrated that overheads do not exceed 10% of the total platform resources, leaving over 90% of the resources available for the CR datapath

Chapter 5

Application to Untrusted Hardware Blocks

Fabrication and assembly of contemporary electronics are generally outsourced to a global supply chain. Even domestic development of modern systems is often assisted by third-party IP modules and COTS components to increase productivity and reduce design costs. High volume chip manufacturing cannot be considered trusted since ICs are vulnerable to tamper and change by untrusted parties throughout the design and fabrication process. HTHs are malicious IC inclusions or alterations to perform certain actions and functionalities not captured by design specifications [166]. HTHs are emerging threats to embedded systems and CPSes with a potential to break all security objectives without being detected using traditional solutions. SoC ICs are especially vulnerable to HTHs as they are assembled from designer-generated register-transfer level (RTL) hardware descriptions, software, and third-party IP cores. All these sources can harbor undocumented, errant, and Trojan behaviors. Heterogeneous IP cores and components deployed in thousands of products are more dangerous, especially with plausible interactions between underlying HTHs. Actions of HTHs include function modification, specification modification, and information leakage.

Most research to develop trustworthy and security protections for SoCs has focused on developing pre-deployment evaluation techniques. Hardware fabrication might be inspected in a sample of devices through a variety of destructive and non-destructive techniques, such as sophisticated imaging of chip layers [26, 84]. The system design can be scrutinized through netlist analysis and

formal verification techniques, simulation, automatic test pattern generation, assertion verification, or model and information flow checking. Evaluation methods can also be applied on an implementation of a prototype or final system, such as functional testing or emulation with property checkers. Equivalence checking between various stages of the design implementation may even be used to ensure trust in the design tools themselves. Unfortunately, most trust evaluation methodologies require the existence of golden reference devices and netlists [167]. Each technique has its limitations and no single one is comprehensive enough to guarantee trust and security throughout a system's lifetime. For example, netlist analysis and emulation techniques are sometimes ineffective in finding Trojans and illegitimate behaviors that are difficult to activate and observe. Another unfortunate reality is that even if feasible evaluation methods are available, certifying an SoC as trusted pre-deployment is too expensive and time-consuming for many designers.

Many Trojan detection methodologies have been proposed to detect HTHs pre-deployment. The compile-time detection methods are categorized as either analysis of side-channels or Trojan activation methods [166]. Side-channel analysis techniques attempt to detect HTHs by measuring the induced changes of a side-channel signals, including timing and power. Trojan activation methods aim to increase the likelihood of activating HTHs. Unfortunately, detecting HTHs using compile-time approaches is extremely difficult for several reasons including:

- The large number of IP cores used in computing systems, in addition to the complexity of modern IP blocks.
- The small feature size of modern nanometer ICs complicates detecting such alterations using physical inspection and reverse engineering methods.
- HTHs are very small compared to the containing components.
- Usually, HTHs are kept dormant waiting for rarely-occurred triggering conditions, which complicates their detection using traditional simulation methods.
- Many detection methods need Trojan-free references which can be difficult to find.

- Side-channel variations induced by HTHs are comparable to those caused by process variations and measurement noise.

Design for hardware trust is the alternative approach to improve Trojan detection by modifying the design flow to add trust anchors. Run-time checkers are derived from assertions about a system's operational or security policies and embedded into modules to monitor and enforce properties in real-time. This protection scheme does not assume the existence of a golden reference or the ability to inspect the internals of third-party IP modules. It provides assurances for suspected behaviors that are difficult to activate or prove. Hardware-based checkers also offer the observability and performance necessary to deter high-speed attacks.

In RETC-CPS, hardware guard components are developed at design-time and integrated into the IP module interfaces to monitor and enforce permissible behavior policies at run-time. Unlike other design-time Trojan detection approaches, RETC-CPS treats untrusted components as black boxes without needing Trojan-free golden references. Regardless of the component complexity and Trojan features, guard components solely rely on monitoring component interfaces to detect anomalous behavior and enforce permissible operation. The generated run-time guard components are capable of detecting HTHs whenever they get activated during the component's life time. This approach requires that every impermissible behavior of the protected module, even those associated with Trojans supporting side-channels or covert communication, can be captured by the security policy. For example, a HTH leaking information from a chip via wireless channels, might consume large spikes in power during a relatively idle period when no communication is taking place. Hence, run-time monitoring of power traces can be used to detect such Trojan instances.

As described in Chapter 3, RETC-CPS provides low-level protections that can detect and tolerate threats raised by HTHs in untrusted components. In this chapter we apply the RETC-CPS protection scheme to third-party hardware IP cores as an example of untrusted components widely deployed in CPSes. The main components responsible for low-level protections in the RETC-CPS architecture are the untrusted component interface guards. IP functional specifications are the only thing that a designer can entrust to formulate the security policy because golden references

and trusted models for third-party IP cores are difficult to obtain. Therefore, the low-level security policies are circuit- and component-based rather than application-based policies enforced at system-level interfaces. Although functional specifications are necessary to enforce the required component behavior, they might not be adequate to enforce the typical behavior without extra functionalities. However, specific security policies and specialized primitives can be employed to enforce non-functional security specifications addressing extraneous functionalities and side-channel attacks, for example.

To illustrate our approach, we present a development of HTHs enabling covert communication in third-party IP cores of high-speed serial interface adapters commonly used in cyber networks. The presented HTHs exploit loose specifications of the underlying media link protocols characterizing the operation of the serial high-speed interface adapters. Usually, operating protocols and functional specifications retain some flexibility to support a variety of applications and operating conditions. However, this flexibility can serve as a double-edged sword that can be exploited by malicious entities to conduct Trojan functionalities without altering the component functionalities. The regular operation of the infected IP cores is not disturbed, yet the embedded HTHs perform extra, non-typical operations enabling covert communication in point-to-point physical links established between infected adapters.

Such HTHs are difficult to detect using conventional design-time HTH detection techniques because they get activated under rarely occurring triggering conditions and they induce marginal area and power consumption overheads. Therefore, the run-time monitoring and enforcement approach adopted by RETC-CPS is indispensable to detect such cleverly designed HTHs and counter their effects. Loose specifications of the underlying protocols are confined to the specific application requirements. Security policies and protection primitives are drawn from the tightened functional specifications, translated into hardware guard components, and integrated into the IP core interfaces. Countermeasures include overriding affected interfaces and sourcing typical alternative signals if possible, bypassing the infected component and activating a redundant backup if available, reporting the threat to the DREC, or swapping the compromised IP core with another one using dynamic reconfiguration.

The remaining of this chapter is organized as follows: In Section 5.1, we provide a brief introduction and background about the developed HTHs and their potentials in cyber networks. Also, we present a potential attack scenario illustrating how the developed HTHs can be exploited to leak confidential information about the containing systems. High-speed serial interface adapter IP cores hosting the developed HTHs are introduced in Section 5.1. RETC-CPS low-level interface guards are presented in Section 5.2. HTH and interface guard implementation details, results, and evaluation are given in Section 5.3. This chapter is summarized in Section 5.4.

5.1 HTH Example: Interacting with Hardware Trojans Over a Network

HTH insertion and detection methods are emerging research topics that can partially leverage software Trojan horse (STH) ideas and experiences. However, software platform homogeneity and programmability provide a higher degree of transitive (non-direct) observability and controllability compared to HTHs, making STHs the preferred choice for attacks. For SoC ICs, development of programmable and transitive HTHs would support more powerful attacks by enhancing controllability and observability. In a HTH context, transitivity is a module's covert ability to relay information of interest from a module's input interface, with or without processing, to a certain destination linked to the module's output interface. Transitivity would extend HTH communication beyond the local node's environment.

The enabling mechanism for HTH transitivity is a means of interaction. HTHs are stealthy by nature and covert communication is their usual means of interaction. A covert channel can be defined as "an enforced, illicit signaling channel that allows a user to surreptitiously contravene the security policy and unobservability requirements of the system" [168]. System vulnerabilities leading to covert channels are a result of design oversights, weakness inherent in the system design, or underspecification of underlying protocols. System vulnerabilities must first be identified in order to provide a defense against covert data channels. Vulnerabilities caused by design oversights

can be eliminated once they are discovered in the system design phase. On the other hand, it may not be possible to remove all potential weaknesses since doing so may lead to inefficient systems. Flexible protocol specifications enable application scalability even though loose specifications may lead to vulnerabilities that can be exploited to violate security specifications and establish covert channels.

We investigate development of HTHs supporting remote interaction over wired computer networks. Software-based covert communication in computer networks is an active research area investigating how STHs can exploit vulnerabilities in the protocol stack [119]. Development of HTHs exploiting network protocols vulnerabilities is an interesting new area to consider. However, it is also challenging due to the number and diversity of layers implemented with both hardware and software. Various layers of the protocol stack can be exploited to create covert data channels, but intended recipients may differ from one layer to another. To set up end-to-end (peer-to-peer) communication in different network topologies, a logical link between endpoints is established by employing a chain of physical links to a particular destination. The route to a particular destination is usually not fixed, and a logical link may be established with temporally varying physical links. Upper (host) layers of the protocol stack are typically connected to applications or users serving as the ultimate source and sink of commands and data.

On the other hand, covert data channels established with lower (media) layers of the protocol stack act as carriers across the endpoints of a physical link. Interactions between multiple HTHs along a logical link require connecting a chain of compromised physical links. Escalating covert information from a compromised lower layer to upper layers of the stack is another way of delivering covert data to intended recipients. Alteration between upper and lower layer covert channels might help to evade detection by specific defenses. However, it may not be necessary to create an end-to-end covert channel because the attacker may be a man-in-the-middle eavesdropping on the network.

In this chapter we advance a lightweight HTH supporting two-way covert communication in point-to-point physical links by exploiting vulnerabilities in the underlying media layer protocols. Covert

data channels are established by inserting HTHs exploiting unenforced, loose specifications of IP cores implementing media layer functionalities in a manner that maintains system operability. Specifically, we explore insertions in a PCS/PMA 10GBASE-X IP core implementing the physical layer functionalities of the 10GbE protocol specified in the IEEE 802.3-2008 standard [3], and in an Aurora IP core implementing the link-layer functionalities in a high-speed serial protocol. Aurora is an open source implementation of a link-layer protocol developed by Xilinx to support serial links between chips employing MGTs, with the protocol specifications adopted from the IEEE 802.3 standard [173].

5.1.1 Trojan-enabled Covert Communication Background

Previous research addressing covert communication using HTHs has emphasized the use of hidden modules and structures to leak sensitive information from a chip. Common underlying assumptions are that a HTH can be added to critical components on the chip such as cryptographic modules, an inability to detect the HTH with existing analysis techniques, and the attacker has local access to the compromised IC. HTHs of this type commonly employ electromagnetic radiation via hidden on-chip antennas, power, or temperature side-channels to encode and leak sensitive information off a chip. Such HTHs transmit covert data over a short range, limited by the hidden nature of the Trojan, such that an attacker residing close to the compromised IC can receive and decode leaked information. Karri *et al*. presented several examples of covert communication using power and temperature side-channels in their Trojan taxonomy [85]. Most of the existing covert communication techniques adopting HTHs and side-channels are localized in the sense that an attacker needs physical proximity or access to the compromised device to exploit information obtained by the embedded HTH.

Adding HTHs to input/output subsystems of a compromised IC provides another means to establish covert channels. Information is covertly transferred via existing interfaces and peripherals by hiding extra data in legitimate communication and interface protocols. For example, HTHs can support covert communication by changing signaling specifications of existing hardware in-

terfaces. Characteristics such as phase, rate, or sequencing can be modulated in unconventional way to encode covert data along with legitimate data. To evade detection by evaluations and typical run-time defenses, covert data transferred using a shared medium should mask itself in signaling or data that is generally ignored by the designer. Attributes characterizing legitimate communication should not be significantly affected by channel sharing. For example, the data rate of a legitimate interface should not be reduced by the insertion of covert information.

The Embedded Systems Challenge competition demonstrates leaking sensitive information from a BASYS FPGA board by changing the RS232 serial interface signaling specifications [24, 81]. Sun *et al*. developed a pin hijacking transceiver module exploiting idle or dead time in an I^2C interface to create two-way communication between an FPGA chip and a serial memory device [161]. They also described another covert channel encoding data in a DDR2 memory interface's phase delay attribute, which is normally used for calibration purposes. HTHs exploiting existing interfaces for covert communication are normally limited to inter-chip interactions, and we are not aware of HTHs supporting covert communication across computer networks.

Covert communication in computer networks uses protocols such as TCP/IP for information transfer instead of the payload data used in steganography. The vast amount of data, large number of existing protocols, and the ease of eavesdropping on computer networks provide many opportunities for high bandwidth covert communication. Zander *et al*. surveyed a number of software-based covert channels in network protocols, and possible countermeasures [174]. Covert channels may be classified as storage channels involving a shared medium accessed by both the transmitter and receiver, and timing channels involving modulation of certain characteristics.

Various protocols and layers have been exploited by STHs to create covert channels over networks. Examples of such exploits include unused header bits, header extensions and padding, TCP initialization sequences, IP identification and framing offset, checksum fields, and many other vulnerabilities. For example, a TCP initial sequence number is used to coordinate between transmitter and receiver, and may be optionally selected by the client according to loose rules. As reported in [174], a covert channel may encode information in this field while maintaining a uniform data

distribution. Previous research addressing covert channels in computer networks has emphasized software methods, with less attention to hardware exploits and countermeasures.

5.1.2 Example Attack Scenario

In this section we present an attack scenario to illustrate communicating useful information with HTHs. Our attack makes use of PUFs, which when challenged provide identifier responses unique to the devices on which they are implemented due to subtle variations in the fabrication process [156]. PUFs have been proposed for device and IP authentication across an untrusted supply chain [65]. It has also been suggested that the unique identifiers PUFs provide might be useful for device tracking [69]. Building on this idea, our attack utilizes PUFs to perform reconnaissance on target systems post-deployment.

Figure 5.1 illustrates conceptually how a PUF with additional Trojan circuitry inserted into a third-party IP module might be used to track systems. An attacker first distributes the Trojan IP as an encrypted netlist core to the target organization. After this point, the attacker may have no visibility into which systems use the IP. However, this attack does not require PUF signatures to be linked to devices beforehand as the attacker would still likely be able to generally target a specific project and may be able to infer system information from recovered PUF identifier information when the system comes online. A PUF is particularly useful for this attack because it enables a single Trojan IP core to be implemented on any number of devices while providing unique information for each. In Figure 5.1, devices A, B, and E make use of the Trojan IP while devices C and D do not. The attack is considered successful when the HTH communicates its PUF-created unique identifier, and perhaps an origination address copied from observed data packets, back to the attacker. The attacker must ultimately have some level of observability over a part of the cyber system in order to recover HTH communications.

We develop possible ways for the distributed Trojan IP to communicate tracking information to the attacker. Section 5.1.3 explores point-to-point covert data channels created over Ethernet links,

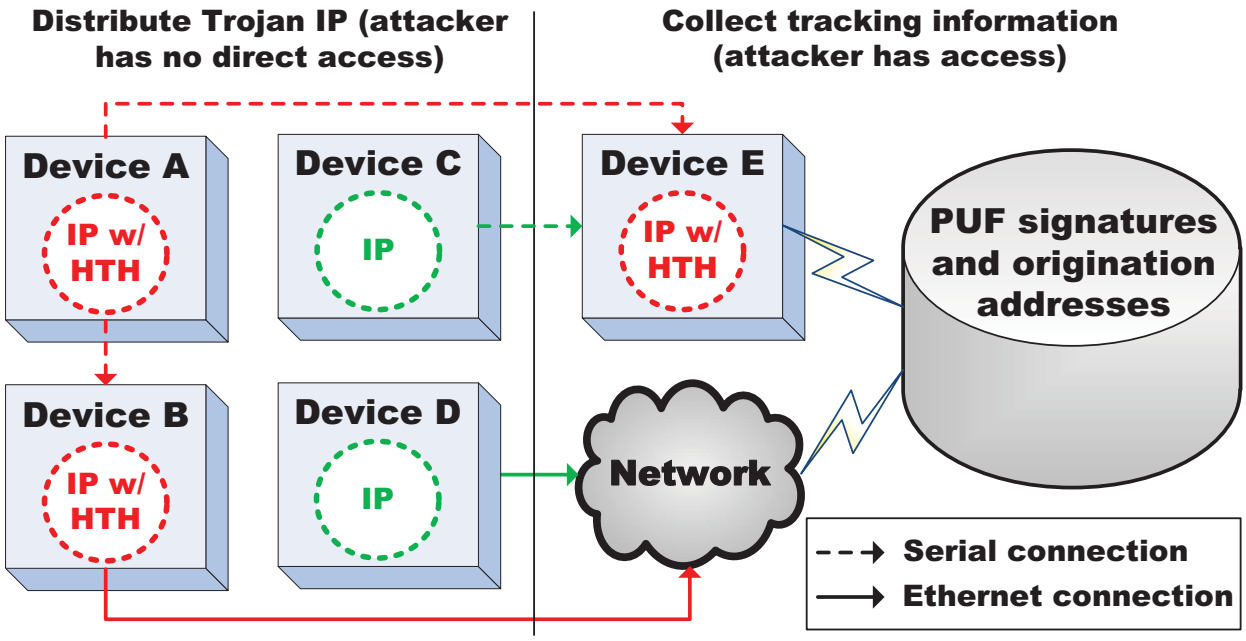


Figure 5.1: Trojan IP system tracking attack scenario.

illustrated in Figure 5.1 as the connection between device B and the computer network. This enables tracking information to be recovered by eavesdropping on the network with which the system communicates. More deeply embedded HTHs may require intra- or inter-device point-to-point links to propagate information out of a system. Section 5.1.4 therefore explores covert channels over serial links such as the connection between devices A and E. This also enables tracking information to be recovered by monitoring a device in a HTH chain. The attacker can leverage access to device E to recover tracking information sourced from device A.

5.1.3 HTH Interaction Across 10GbE Physical Links

We first consider how to propagate covert information across 10GbE links, enabling a PUF signature to be transmitted from a device to a network. 10GbE offers a more efficient and less expensive approach to moving data on backbone connections between network switches while also providing a consistent technology end-to-end. It uses the IEEE 802.3 MAC sublayer, connected through a 10 gigabit media independent interface (XGMII) to the 10GBASE physical layer entity specified

in IEEE 802.3 clause 48. The physical layer of 10GbE consists of the physical coding sublayer (PCS), the physical medium attachment (PMA), and the physical medium dependent (PMD) sublayers [3].

10GBASE-X is a serial interface IP block implementing the PCS and PMA functionalities between the XGMII MAC and the PHY layers. The 10GBASE-X PCS maps XGMII data and control characters to/from a stream of code groups according to an 8B/10B transmission code. The PCS is responsible for data encoding/decoding, lane synchronization and alignment, conversion of XGMII idle control characters to/from a randomized sequence of code groups, and PHY clock rate compensation achieved by embedding special non-data code groups in the idle stream. Clock recovery and serializing/deserializing data are performed in the PMA. The PMD layer consists of four lanes employing high speed serial transceivers running at 3.125 GHz. Figure 5.2 illustrates the block diagram of the PCS/PMA 10GBASE-X IP core.

The PCS transmit process continuously generates code groups based upon transmit data (TXD) and control (TXC) signals, while the PCS receive process continuously accepts code groups from the PMA service interface and generates receive data (RXD) and control (RXC) on the XGMII. All received idle code groups are replaced with idle characters before forwarding to the XGMII. The 8B/10B transmission code as well as the rules by which the PCS encode and decode code groups are specified in IEEE 802.3 clause 36. 10GBASE-X PCS ordered sets consists of combinations of special and data code groups of length four beginning in lane 0. The PCS defines special code groups for control purposes, and provides capabilities such as synchronization, deskew, and error detection.

Idle Sequence-based Covert Channels

In this section we sketch HTHs enabling covert communication in 10GbE physical links by changing signaling specifications of the underlying PCS layer. Idle ordered sets $\|\mathtt{I}\|$ are transmitted in full columns whenever the XGMII is idle. The idle sequence (ISQ) provides a continuous fill pattern to establish and maintain lane synchronization, perform lane-to-lane deskew, and achieve PHY

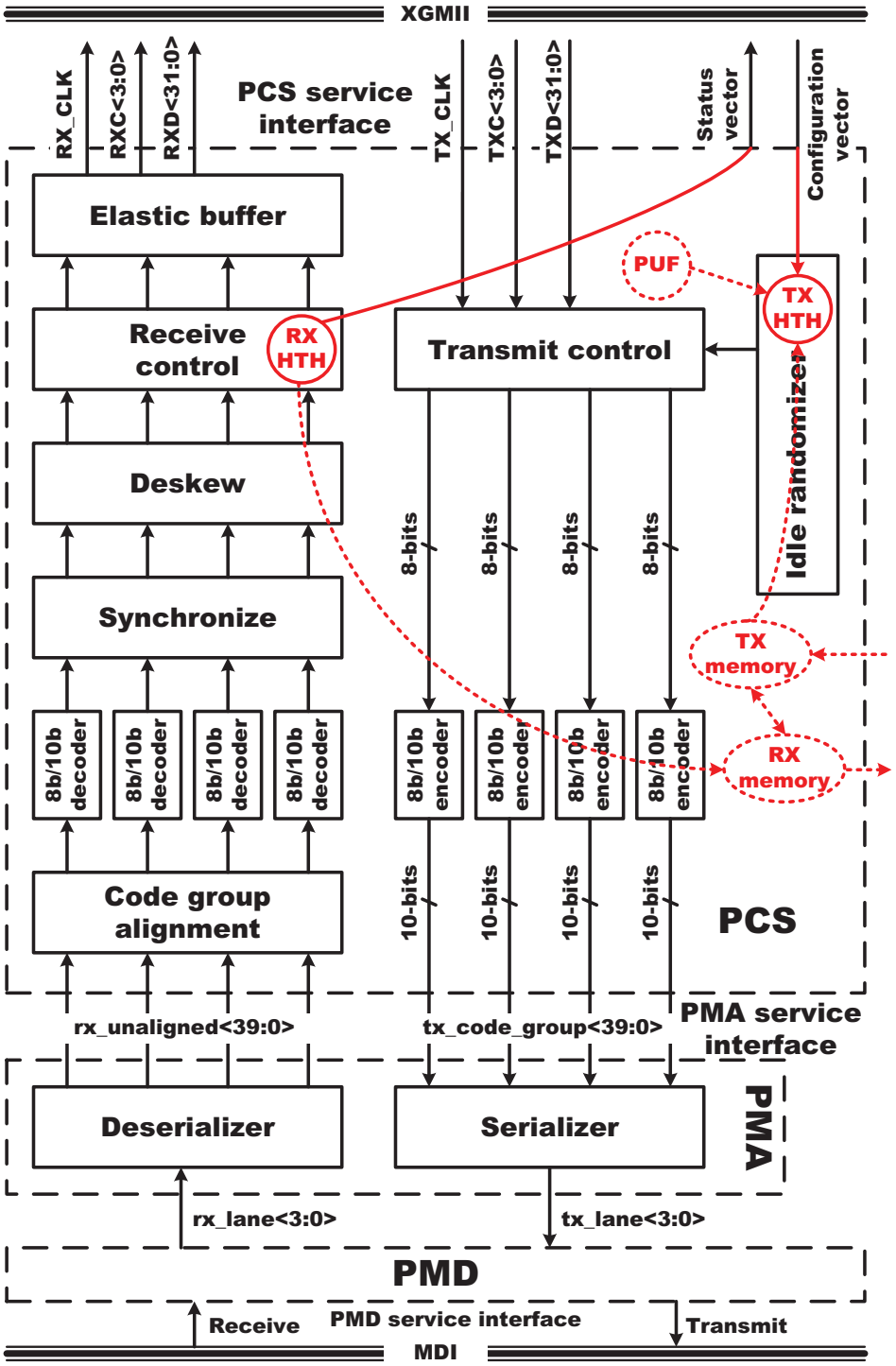


Figure 5.2: PCS/PMA 10GBASE-X IP core.

clock rate compensation. An ‖I‖ sequence consists of one or more consecutive sync_column ‖K‖, skip_column ‖R‖, or align_column ‖A‖ ordered sets. Some of the rules governing ‖I‖ ordered set sequencing are:

- Each ‖A‖ is sent after r non-‖A‖ columns where r is a randomly distributed number between 16 and 31.
- When not sending an ‖A‖, either ‖K‖ or ‖R‖ is sent with a random uniform distribution between the two.

Both ‖A‖ spacing as well as ‖K‖, ‖R‖, or ‖A‖ selection are based on a random integer r generated by a PCS idle randomizer employing a pseudo-random binary sequence generator. We exploit the ISQ ordered sets to encode covert data between the endpoints of the physical link by adding a HTH to the standard idle randomizer. To satisfy ISQ constraints, only ‖K‖ and ‖R‖ ordered sets are used to encode data as shown in Figure 5.3. Spacing between consecutive ‖A‖'s is fixed at 32 characters, the maximum value permitted. Other techniques such as Huffman encoding enable uniformly distributed covert data using the three ordered sets to increase entropy.

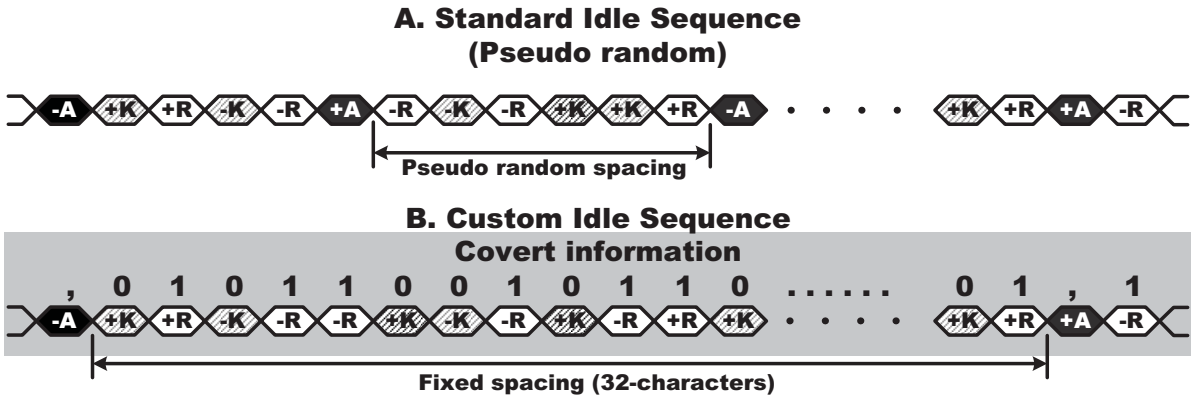


Figure 5.3: Standard and custom idle sequence timing diagram.

As shown in Figure 5.2, the TX HTH is a binary encoder instantiated inside the PCS idle randomizer, and the RX HTH is a binary decoder instantiated inside the receive control module. The TX and RX HTHs could be selectively enabled by a rarely occurring data sequence in the transmit and

receive sides. In our example scenario, a tracking PUF provides the TX HTH with the IP or device signature to be sent over the ISQ covert channel. Configuration and status vectors are possible interfaces for covert information flowing to and from the core. Portions of these interfaces may be used in the development and test phases while kept idle during normal operation. These interfaces could be employed by the TX and RX HTHs to forward covert data to upper layers of the protocol stack. Store and forward techniques using hidden memory modules might also also be used.

5.1.4 HTH Interaction in Multi-gigabit Transceivers

We now consider how to propagate covert information across high-speed serial links adopting MGTs to enable a PUF signature to be transmitted between devices. MGTs are the preferred means of transferring high-speed data between integrated circuits. The sender and receiver do not share a global or transmitted clock signal. Parallel data words are serialized by the MGT transmitter, and the receiver performs the reverse function. Low voltage current mode logic supports signaling rates up to 28 Gbps over a single differential pair of conductors. Compared to parallel data transfer, MGTs reduce pin count, electromagnetic interference, ground bounce due to simultaneous switching outputs, and power. MGTs are widely deployed in ASIC- and FPGA-based communication standards and applications such as PCIe, SATA, Fibre Channel, Infiniband, SAS, Serial RapidIO, and Gigabit Ethernet. Figure 5.4 illustrates the basic architecture of an MGT. The PCS is responsible for idle pattern generation, data encoding/decoding, and lane alignment. Channel transmission normally uses 8B/10B encoding, which defines 256 data characters and 12 control characters represented as 10-bit code groups. Clock differences between the sender and receiver can be tolerated without requiring flow control. The PMA is responsible for serializing and deserializing data [18].

An MGT integrates clock, data, and control in a single bitstream transferred over a point-to-point serial link. Data and control information are encoded in different characters groups and sequences, and a PLL is used to recover the clock. Channel control characters are added and interpreted by the link-layer protocol based on application requirements and physical channel state. Aurora 8B/10B

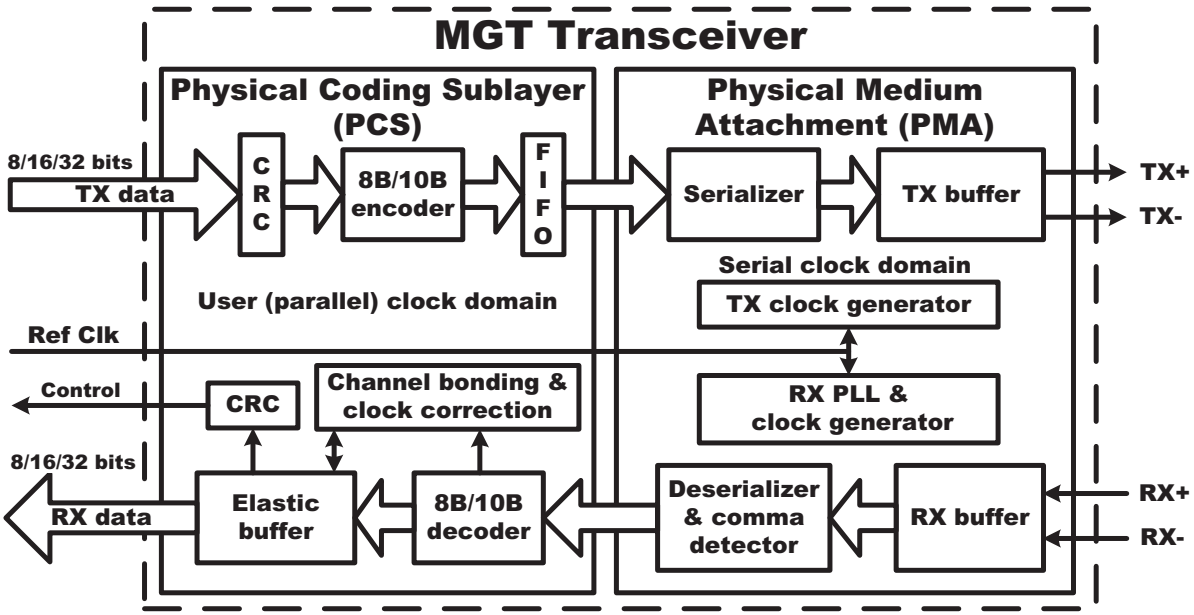


Figure 5.4: Multi-gigabit transceiver architecture.

is a simple example of a link-layer protocol defining the packet structure, communication channel initialization and validation, error handling, and clock compensation [173]. Figure 5.5 shows the top-level block diagram of the Aurora IP core, including a brief description of individual modules. Aurora shares characteristics and basic functionalities of other link-layer protocols such as PCIe and XAUI [125].

Though the design and use of MGTs has been extensively addressed, we are not aware of investigations regarding their use in covert communication. We present system modifications and internal IP alterations and additions enabling covert communication using MGTs, and demonstrate specific proof-of-concept implementations for each. Figure 5.6 depicts insertion points and top-level interfaces of the additions and modifications. The HTHs are evaluated in terms of performance measured by covert communication bit rate, cost estimated by resources usage, productivity assessed by implementation difficulty, and impact on the main communication channel.

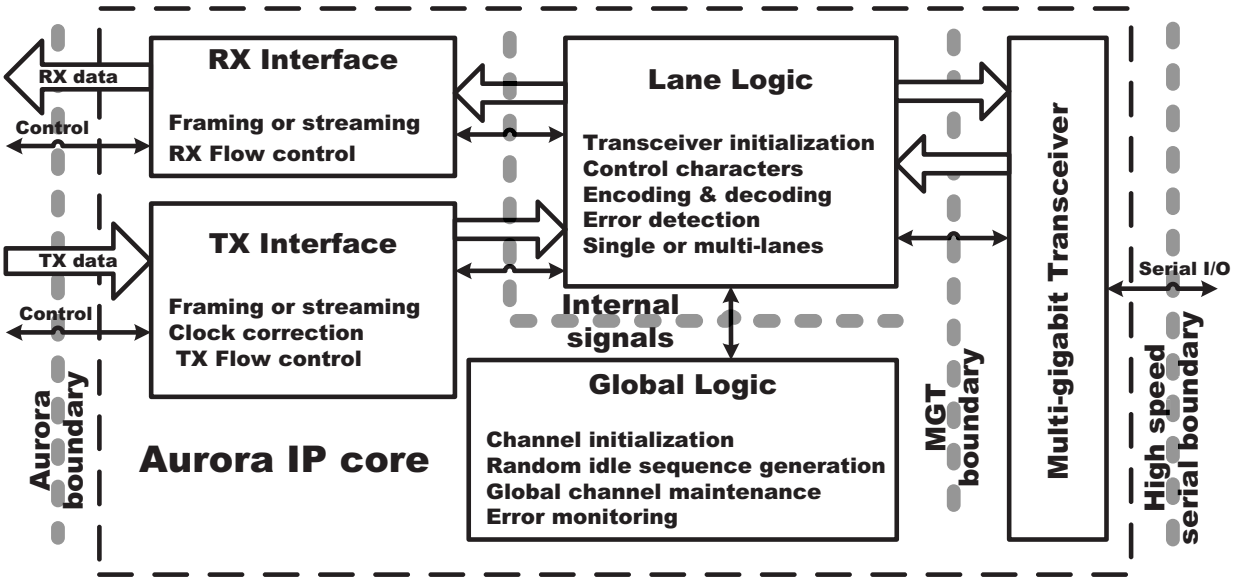


Figure 5.5: Aurora IP core structure.

Clock Correction-based Side-Channel

The tight jitter requirements on a transmission clock normally prevent an MGT from using the recovered clock as a reference clock. Each MGT system often has its reference oscillator generating its unique frequency. Small deviations between the transmitter and receiver clock frequencies result in either overflow or underflow of the receiver FIFO. Clock correction (CC) compensates advance/delay of the received clock due to the deviation between oscillators' frequencies. Most MGTs have a built-in CC option which involves a unique symbol or sequence of symbols not found elsewhere in the data stream.

On the transmitter side, the Aurora protocol implements CC by periodically inserting clock compensation sequences composed of multiple instances of the control character `/CC/` into idle patterns or user data. On the receiver side, if the FIFO is getting close to full, the PCS looks for the next CC sequence and drops it. Conversely, if the FIFO is getting close to empty, the PCS writes the next CC sequence twice into the FIFO [173]. The Aurora IP core is provided with an external CC module which generates a standard CC signal. Customizing the CC module should be

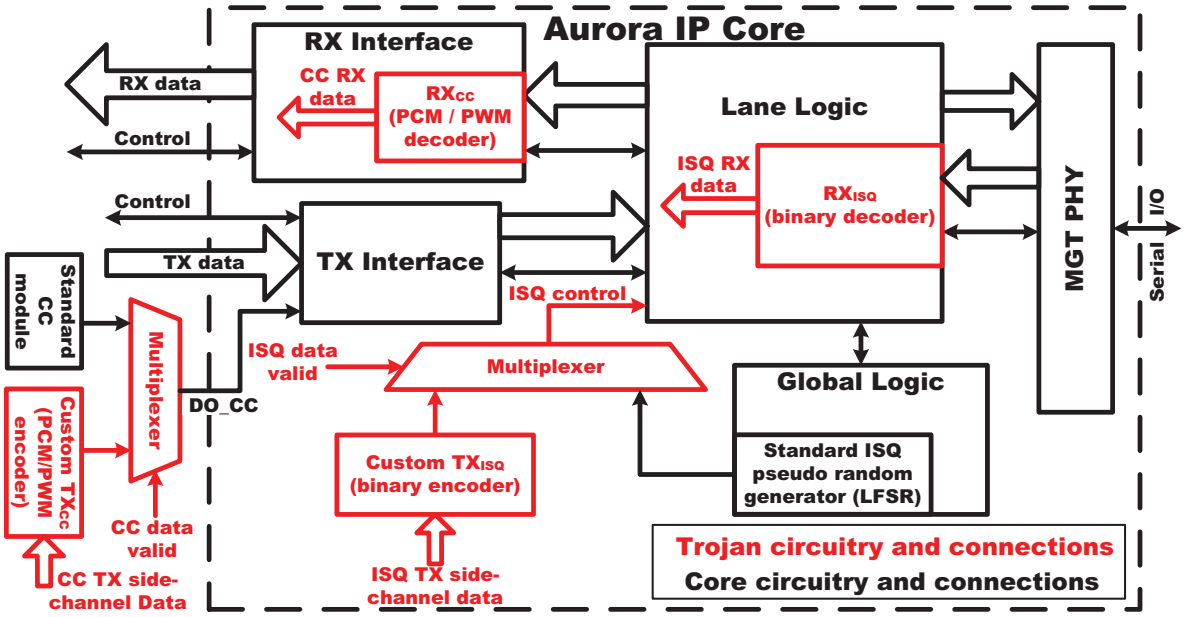


Figure 5.6: Modified Aurora IP core.

performed with careful analysis, testing, and consideration of the following guidelines:

- CC sequences should last at least two cycles to ensure they are recognized by the receiver ($T_{CC\_min} = 2\,T_{Ref\_clk}$), where the Ref_clk is the parallel data clock.
- Duration and period should be precisely assigned to correct for the maximum difference between frequencies of the used oscillators.
- The minimum separation time between consecutive CC sequences is eight clock cycles ($T_{Sep\_min} = 8\,T_{Ref\_clk}$).

CC can source covert information with CC variation instead of standard periodic initiation. The CC is performed by asserting a `DO_CC` control signal which stalls data and relays a sequence of repeated `/CC/` characters. The `DO_CC` is an input to the TX user interface of the Aurora IP shown in Figure 5.5. Different pulse modulation techniques can be used for covert data encoding. We employ pulse code modulation (PCM) and pulse width modulation (PWM) to encode data in the

CC, as illustrated in Fig. 5.7. In a PCM CC cycle, asserting `DO_CC` for a certain period denotes a logic 1 while releasing it denotes a logic 0. In a PWM CC cycle, n bits of data can be encoded in the `DO_CC` signal hold time (pulse width). The CC TX HTH is a custom encoder driving the `DO_CC` signal as shown by Figure 5.6. The CC TX HTH is a custom decoder instantiated inside the Aurora RX interface module to detect received CC sequences carrying covert information. In both encoding schemes, the CC is periodically asserted under the control of covert information. Transmitter circuits are selectively activated under a triggering condition of a certain data sequence.

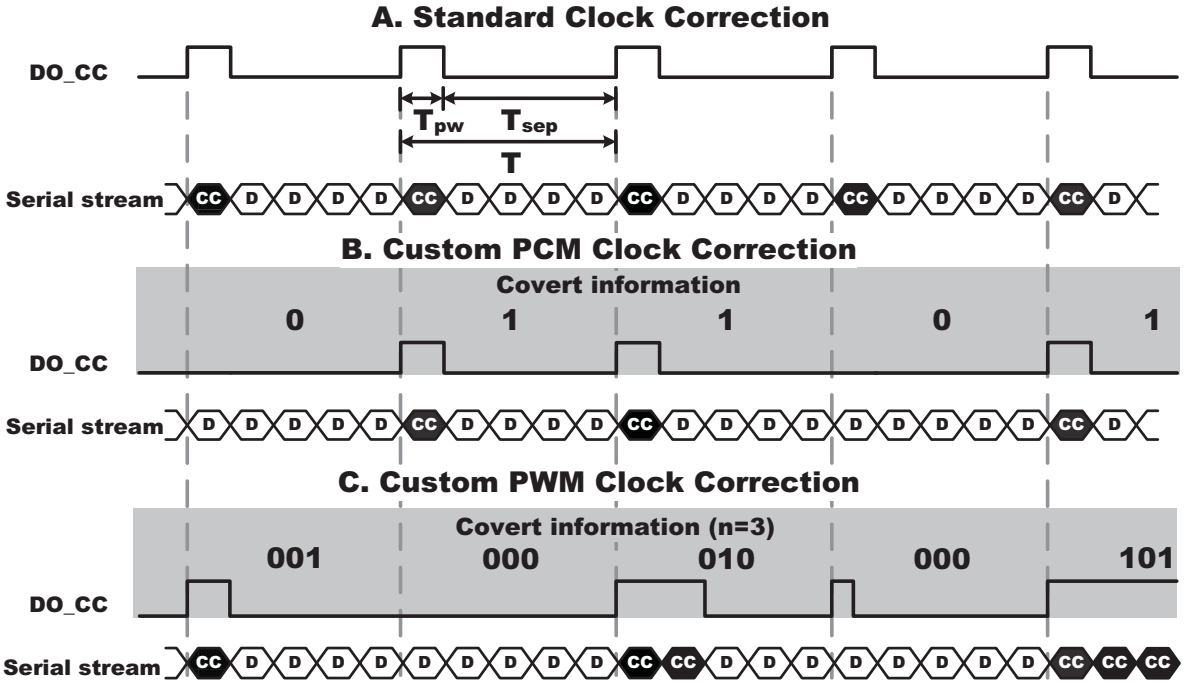


Figure 5.7: Standard, PCM, and PWM clock correction signals

The CC-based transmitter provides a general model for establishing covert communication using an MGT flow control mechanism. The transmitter is either a custom PCM or PWM module driving the `DO_CC` input to the Aurora IP, sharing communication channel with the main serial stream, and sending data by modulating the CC incident. We provide the transmitter with a a triggering input indicating the presence of covert information and a predefined initialization and termination sequences signaling the start and end of the transmission, respectively. The custom `DO_CC` output of the transmitter is multiplexed with a standard CC signal and the selector is a `data_valid`

signal indicating the existence of covert information. The receiver is either a custom PCM or PWM decoder module instantiated inside the Aurora IP core with internal access to the RX user interface internal signals shown ny Figure 5.6. The IP internal signals are used to detect received CC sequences carrying covert information.

The bitrate R of covert communication can be defined as the number of bits transferred via the covert channel per second. The number of bytes/lane does not contribute to the number of encoded bits/cycle since the same character denotes the CC for all bytes. The covert transmission stops the main data flow because of the channel sharing which reduces the data rate of the main communication. Rate reduction resulted by the channel utilization induced by the CC is a crucial factor in evaluating the HTHs. Channel utilization ratio U is function of the CC pulse width T_{PW}, period T, and occurrence probability P_{CC}.

$$R = \frac{\text{\# bits encoded/CC cycle}}{\text{CC period}} \tag{5.1}$$

$$U = \frac{T_{PW}}{T} P_{CC} \tag{5.2}$$

The PCM transmitter encodes a single bit per CC cycle, as exemplified by Figure 5.7, where the CC period is constant value. The probability of the CC occurrence equals the probability of logic "1" existence in the covert channel information which equals 50% in balanced data. The PCM transmitter maximum bitrate and channel utilization are evaluated by substituting the CC pulse width and separation time with the CC customization constraints.

$$R_{PCM} = \frac{1}{T_{PW} + T_{Sep}} \tag{5.3}$$

$$U_{PCM} = \frac{T_{PW}}{2(T_{PW} + T_{Sep})} \tag{5.4}$$

$$R_{PCM\_max} = \frac{1}{2T_{Ref\_clk} + 8T_{Ref\_clk}} = \frac{f_{Ref\_clk}}{10} \tag{5.5}$$

$$U_{PCM\_max} = \frac{2T_{Ref\_clk}}{2(2T_{Ref\_clk} + 8T_{Ref\_clk})} = 10\% \tag{5.6}$$

PWM transmitter can encode n bits of data in the DO_CC signal pulse width by asserting it for a variable number of clock cycles ranging from 2 to $(2^n + 1)$ and fixing the separation time which satisfies the CC constraint. To regulate the DO_CC signal, the period T is fixed to a predefined value as shown in Figure 5.7. In our design, T is selected to be m times the maximum pulse width where m is an integer number larger than one to satisfy the third constraint. The CC occurrence probability is 100% ($P_{CC} = 1$) since CC takes place regularly with a variable pulse width. Channel utilization U_{PWM} can be evaluated by assigning an average value to the CC pulse width. Assuming uniform distribution of covert information, T_{PW_avg} is the mean of the variable CC pulse width T_{PW}. The PWM transmitter maximum bitrate and channel utilization can be calculated by assigning the minimum value of m, which equals 2, in Eq (5.7) and Eq (5.8), respectively.

$$R_{PWM} = \frac{n}{T} = \frac{n}{mT_{PW\_max}} = \frac{n}{m(2^n + 1)} f_{Ref\_clk} \tag{5.7}$$

$$U_{PWM} = \frac{1/2((2^n + 1) + 2)T_{Ref\_clk}}{m(2^n + 1)T_{Ref\_clk}} = \frac{2^n + 3}{2m(2^n + 1)} \tag{5.8}$$

$$R_{PWM\_max} = \frac{n}{2(2^n + 1)} f_{Ref\_clk} \tag{5.9}$$

$$U_{PWM\_max} = \frac{2^n + 3}{4(2^n + 1)} \tag{5.10}$$

Idle Sequence-based Covert Channels

MGTs must incorporate a number of functions to permit high line rates. ISQs are ordered sets of control characters used to perform word boundary alignment and channel bonding during initialization. During operation, ISQs are inserted during wait states to keep the channel active. The Aurora protocol's ISQ uses `/A/`, `/K/`, and `/R/` control characters applied in a pseudo-random sequence subject to the Aurora constraints drawn from IEEE 802.3 [173]. We change the signaling specification of the Aurora protocol to encode covert information in ISQs in a manner similar to Section 5.1.3, and as follows:

- `/A/` spacing is randomized with a minimum of 16 code groups but no more than 32 code groups between any two `/A/` code groups.
- `/K/`'s and `/R/`'s are randomly placed between `/A/`'s as long as the 0/1 running disparity maintains DC balance.
- The minimum transmit pattern is one symbol pair. Any of the three characters can be sent, as long as the preceding two rules are obeyed. If the `/A/` is sent, the spacing rule must still be obeyed.
- For multi-lane channels, the same idle symbol-pair must be transmitted simultaneously on all lanes that require an idle at that time.

Wait states can encode covert information in the ISQ without affecting the main channel communication rate. To satisfy ISQ constraints, only `/K/` and `/R/` characters are used to encode data as shown in Figure 5.3. Spacing between consecutive `/A/`'s is fixed to 32 characters, the maximum value permitted. The ISQ TX HTH is a binary encoder instantiated inside the Aurora core and the ISQ RX HTH is a binary decoder instantiated inside the Aurora lane logic module. ISQ TX HTH encodes binary data in `/K/` and `/R/` characters between separating `/A/` characters as illustrated by Figure 5.6. The ISQ TX HTH bit rate depends on the number of bytes per lane, where every

byte corresponds to an idle character. The covert channel bandwidth B is a function of the achieved bit rate and the MGT wait state time percentage $I_{\%}$:

$$R_{ISQ} = 31/32 \cdot \text{Bytes/lane} \cdot f_{ref\_clk} \tag{5.11}$$

$$B_{ISQ} = \text{Bytes/lane} \cdot f_{ref\_clk} \cdot I_{\%} \tag{5.12}$$

5.2 Multi-gigabit Transceiver Interface Guards

Assuming that a designer needs to employ the Aurora IP core developed by a third-party in a security-critical application. Of course, the designer does not have any idea about the embedded Trojans and is not aware of the core's internal structure. Only the IP functional specifications can be obtained from the third-party developer. Hence, building run-time security defenses and protection primitives is necessary and such protections should mainly rely on the IP functional specifications. Specific protections can be developed to thwart cyber threats not captured by the IP functional specifications such as information leakage via side-channel communication.

Similar to the CHARE-CR framework, security policies can be drawn from the IP functional specifications and enforced at the module interfaces. Such an approach requires describing functional specifications in a formal language and translating these descriptions into hardware assertions. The functional specifications are a set of rules describing the component behavior in terms of time. Therefore, the language used to describe such properties must be supported by a temporal logic system. The IEEE PSL provides a formal notation for expressing temporal logic properties that can be automatically verified on electronic system models. Many ongoing research efforts attempt to develop automated tools enabling direct hardware synthesis from specifications [27]. In this work, we will not discuss the language-based approach to develop low-level guards because this approach has been described previously in Chapter 4.

In this chapter, our approach to develop low-level guards relies on building some security primitives directly synthesized from the IP functional specifications. Unlike language-based protections, security primitives are developed from informal specifications and might require formal verification pre-deployment. Moreover, a security primitive can be an imperative realization of a functional specification rather than a declarative description of what should be enforced as practised in language-based protections. Those primitives can build a security library which can be later used in future security projects. This idea is similar to building verifiable component libraries to reduce the formal verification effort. The security primitives are hardware components realizing the functional specifications of the IP core under protection. Such primitives can be employed in conjunction with language-based protections to build more robust low-level interface guards.

In this section we present our implementation of interface guards for low-level untrusted modules. In particular, we explore the effectiveness of using interface guards in detecting the MGT-based covert channel activities introduced in the last section, assuming lack of knowledge about the developed HTHs. In our specific examples, interface guard responsibilities include run-time detection of unsanctioned behaviors introduced by external and/or internal threats, and enforcement of countermeasures dictated by the policy. We provide several developments of the interface guards showing security alternatives available to the designer and analysing these choices in terms of the offered security and induced overheads. The diversity of security alternatives results from the variety of available detection methods and associated countermeasures.

5.2.1 Concepts

Policies enforced by interface guards comprise a number of guard-action rules with high-level coordination between them. Countermeasures enforced by an interface guard are specified as either passive or active measures. Passive measures do not affect the module's operation yet raise alarms and report anomalous events to the top-level DREC. Active measures involve enforcement of corrective actions such as overriding compromised interfaces, disabling infected modules, switching to redundant backups, or swapping compromised modules using partial reconfiguration. The en-

forced countermeasures are selected based on the policy specifications, threat severity level, and the provided assurances about detected violations.

Guard monitors are designated to either cover a wide range of threats or only catch specific threats. Therefore, violation detection methods can be classified into generic and specific methods. Both guard classes can be used to monitor the same interface at different levels of abstraction. In generic monitors, interface specifications are abstracted and generalized to extend the range of addressed threats including zero-day attacks. For example, in a serial high-speed interface, the effective data rate can be affected by several threats including the CC attack. Therefore, data rate change can indicate the incidence of an anomalous activity which can be raised by a cyber threat. However, because such an attribute can be affected by several stimuli including non-malicious operating conditions, the assurance level provided by such a generic monitor neither confirms the attack nor determines the threat origin if any. The advantage of using generic guards is that the number of required monitors, and consequently the induced overhead, can be reduced as several threats can be addressed by a single generic monitor. Because generic guards do not look for a predefined attack signature, such guards can detect zero-day threats lacking predefined signatures. Generic guards can raise proactive alarms about potential threats to the top-level DREC which decides the suitable response.

Specific monitors employ the component's functional specifications to detect malicious activities. A specific monitor assures that a certain functionality of the IP core under protection conforms to the corresponding functional specifications. Specific monitors must be capable of detecting both anomalous and extraneous behaviors to assure functional conformance. For example, the Aurora IP core performs multiple functionalities including physical and link layers interfacing, link initialization and error handling, data stripping, flow control, CC, channel synchronization, and ISQ generation. Each of these functionalities can be associated with different threats which requires allocating a specific monitor to detect functionality-oriented violations. This approach to detect specific violations requires functional duplication to create a monitor reference. Therefore, if we need to detect idle sequence-oriented violations, we should develop the original ISQ generator as a security reference and continuously ensure that the monitored conforms to the reference ISQ.

Some threats can be detected by adopting anomalous detection algorithms and intrusion detection methods without duplicate the complete functionality. Specific monitors increase the assurance level of detecting specific threats and identifying the threat origin. However, using specific monitors to detect all security violations can be intractable. Moreover, assigning monitors to particular functionalities limits their detection scope to specific and known threats. Interface guards can incorporate multiple monitors of different types to increase their detection scope. A specific guard can be initially integrated to a module interface or can be subsequently attached to the interface at run-time if an alarm is issued by a generic monitor.

Often times, the adopted monitor class dictates the type of countermeasures enforced by an interface guard. Generic monitors do not provide adequate assurances about a threat incidence and origin. Consequently, countermeasures associated with generic monitors are usually passive responses such as raising alarms about suspect modules and calling for specific monitors to increase certainty about a threat. On the other hand, specific monitors provide definite information about specific functionalities and, therefore, assurances offered by such monitors are adequate to identify a threat incidence and origin. Consequently, countermeasures associated with specific monitors can be active measures such as disabling compromised modules or enabling safe mode operation. In specific monitors developed using functional duplication, the reference functionality can be used to override compromised interfaces if possible during the attack period. Table 5.1 illustrates the main differences between generic and specific interface guards.

Table 5.1: A comparison between generic and specific interface guards

	Generic interface guards	**Specific interface guards**
Detection criteria	abstract attribute	reference functionality
Assurance level	low	high
Induced overhead	low	high
Detection scope	wide	narrow
Associated countermeasures	passive	active

IP core functional specifications usually describe the core's external operation yet do not provide

detailed information about the core's internal structure. Such specifications are associated with the core external interfaces which are clearly defined compared to internal nets and buses. This simplifies selecting and allocating the protection infrastructure monitoring probes. Therefore, the external interfaces of an untrusted component are the ideal place to add the RETC-CPS interface guards. However, not all security violations can be detected through interface monitoring because some threats can achieve their goals without affecting the component's external operation. An example of these threats are HTHs leaking information via a power side-channel. Threats which cannot be detected through interface monitoring and require deployment of specific protection techniques are outside the scope of this dissertation.

The insertion point of an interface guard directly affects the enforced policy rules, design complexity, guard detection scope, and resources requirements. In our specific example, potential interface guard insertion points are the Aurora IP boundary, MGT boundary, and high-speed serial boundary as shown in Figure 5.5. It is assumed that internal signals cannot be probed in black box modules supplied by third-party IP vendors. The serial boundary is the external transceiver pin interface. Adding a guard to this interface gives a high-level view of the overall system activity, extending guard detection scope but with considerable speed and complexity costs. We can integrate an interface guard at the Aurora IP boundary running at the parallel data rate to reduce the design complexity. However, interface guards inserted at certain points may not always be able to detect all the threats caused by internal IP alterations. For example, the ISQ HTH activity does not affect traffic on the the Aurora IP boundary and hence it cannot be detected using the Aurora interface guard. Therefore, several guards can be integrated into different interfaces to gain wider oversight of the system behavior.

In our implementation, interface guards are inserted at the Aurora IP boundary and the MGT hard core boundary to monitor all Aurora input and output signals. The MGT module is instantiated as a building block within the Aurora top-level module, and the MGT interfaces belong to the IP internal signals. However, the MGT interface is still accessible because the MGT is a physical block with a predefined interface to the FPGA programmable fabric. The MGT interface guard can be integrated into the IP netlist after the synthesis phase to enable probing the MGT interfaces.

Using such a configuration, enforced policies may apply statistical analysis of I/O activities.

5.2.2 Generic Interface Guards

In this section we present our development of the Aurora and MGT generic interface guards. Figure 5.8 illustrates the top-level architecture of the Aurora IP core with generic interfaces guards. The generic guards probe the Aurora IP and MGT interfaces but do not have the capability to override these interfaces. As Aurora is a serial high-speed interface, the effective data rate is a substantial assessment criteria of the IP operation. Therefore, we employ the data rate attribute as an indicator of anomalous behavior, and two rate meters acting as generic monitors are integrated into the Aurora and MGT I/O interfaces.

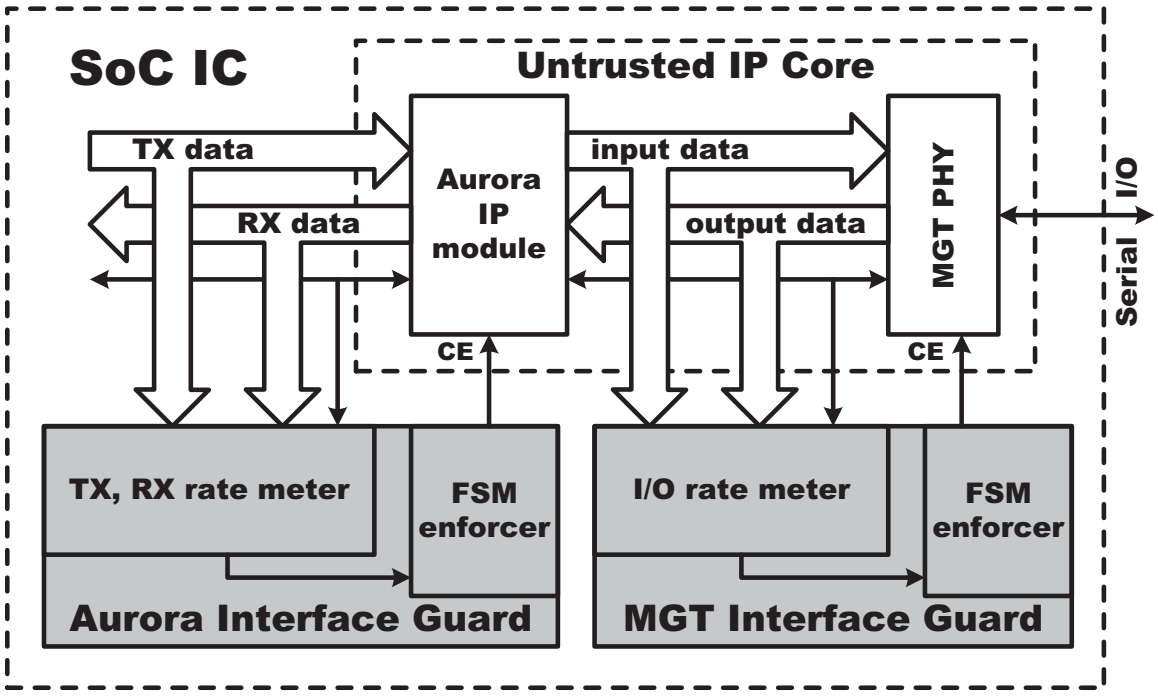


Figure 5.8: Aurora core architecture with generic interface guards.

The Aurora TX and RX rate meters measure the number of transmitted and received bits/sec during active transmit and receive periods. The MGT rate meters measures the number of transmitted and

received data bits/sec excluding initialization data, idle sequences, and flow control characters. The MGT generic guard filters out all control characters from the I/O parallel streams to enable the rate meter to measure the effective data rate. Countermeasures enforced by these two generic monitors include reporting potential threats to the DREC and disabling the compromised module if the threat is confirmed by the DREC.

We now discuss potential threats that can be detected by each interface guard. Nominal TX and RX effective data rates are predefined values calculated from the reference clock frequency and the actual line rate. Factors affecting the effective data rate include the communication channel bit error rate and potential cyber threats. Rate deviations from nominal bounds indicates either an increased bit error rate in the communication channel or a suspicious behavior which might be caused by several threats including CC-based covert communication. As both the TX and RX use the same channels, variations between the TX and RX effective data rates may affirm that the functionalities of the lower rate communication channel are affected by a cyber threat.

Threats associated with the previous cases might arise from compromising either the Aurora IP core or the MGT physical module, although the interface guards cannot exactly assert either the affected module or the compromised functionality. In this specific example, the MGT physical module is a fixed silicon block, and most probably that a rate deviation from nominal values or a rate variation between the TX and RX is more likely caused by a potential threat in the configurable logic used by the Aurora module. Another violation that can be detected by analysing the two data rates is rate variations between the Aurora and MGT inputs or between the two module outputs. The effective data rate of both modules in a certain direction must be equal and rate variations directly indicate a potential threat associated with the Aurora module. However, even in this case, the generic monitors cannot assert the compromised functionality is within the Aurora IP.

5.2.3 Specific Interface Guards

In this section we present our development of the Aurora and MGT specific interface guards. Figure 5.9 illustrates the top-level architecture of the Aurora IP core with specific interfaces guards. The developed specific guards follow the RETC-CPS architecture presented in Chapter 3 where the guard's inputs are the untrusted component's outputs and the guard's outputs are the component's inputs. This configuration enables the interface guard to override compromised interfaces if a violation is detected. As specific guards increase the assurance level about potential threats, affected modules can be immediately disabled or overridden if a violation is detected The Aurora- and MGT-specific guards can be inserted into an already running system using dynamic reconfiguration based on proactive alarms issued by in-field generic monitors attached to the protected components.

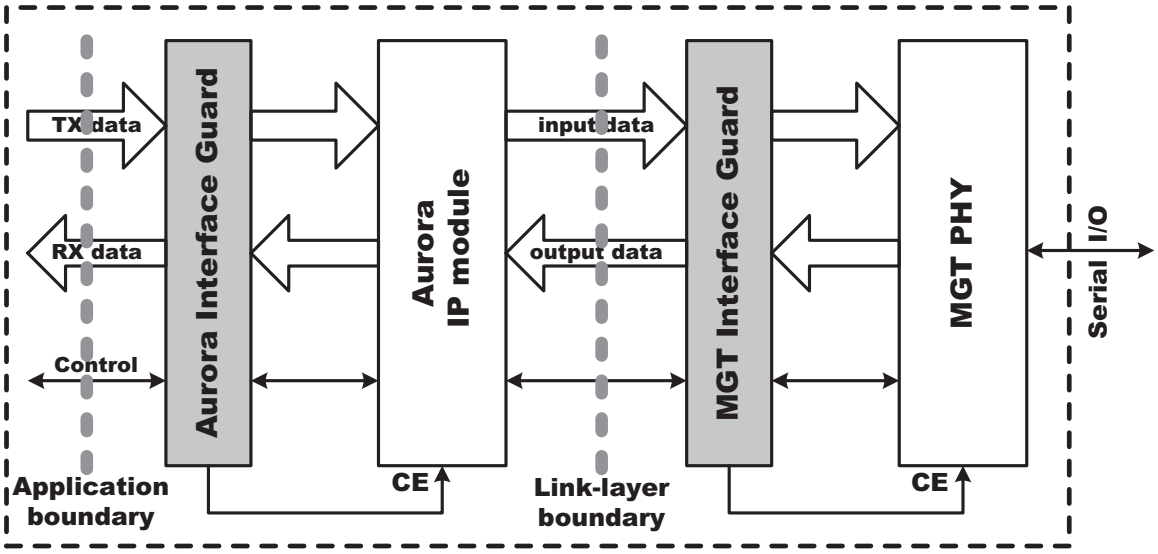


Figure 5.9: Aurora core architecture with specific interface guards.

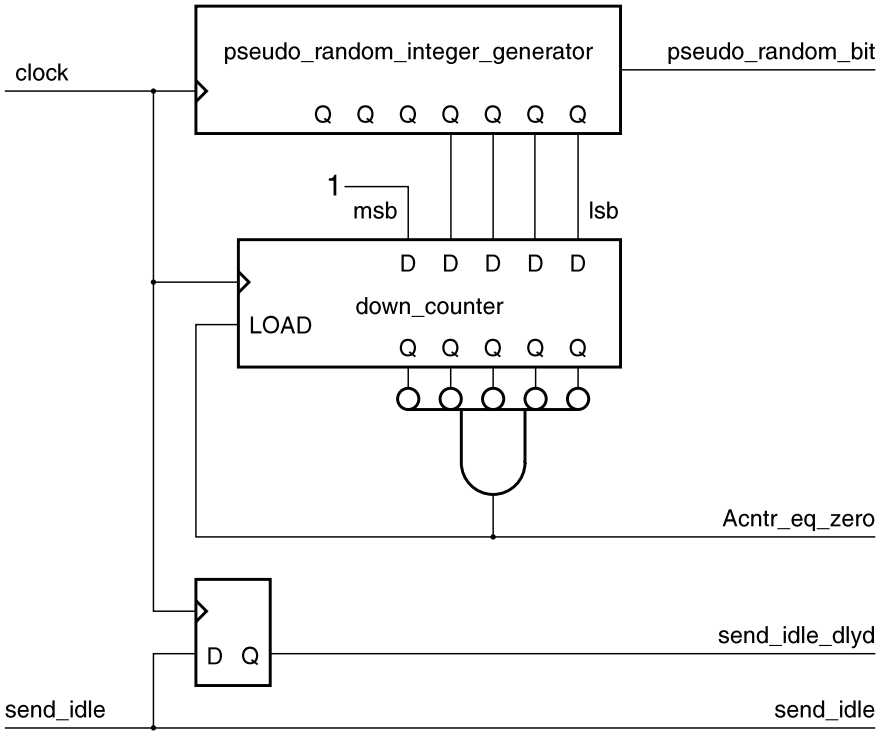
The main objective of the Aurora interface guard is to detect CC functional violations of the Aurora IP core. Although this guard is specifically developed to detect the CC HTHs presented in Section 5.1, the same concept can be generally applied to develop more specific guards for other IP functionalities. The standard CC module sources a fixed number of CC characters periodically

by asserting the `DO_CC` control signal to compensate for variations between the TX and RX clocks. The CC period and duration are predefined parameters depending on the frequency difference between the TX and RX clocks. Incorporating these specifications, the CC-specific interface guard is developed to monitor the `DO_CC` control signal at run-time and assert that the CC period and duration match the standard, predefined values. This interface guard can detect different CC HTHs which alter the standard functionality of the CC standard module generating the `DO_CC` signal on the TX side. Consequently, the Aurora-specific guard can override the compromised `DO_CC` signal with the standard signal defined by the specifications. However, because this interface guard does not have access to the link-layer boundary revealing the control characters received by the Aurora module, it cannot detect CC covert communication on the RX side.

The MGT-specific interface guard is placed at the link-layer boundary and is charged with three responsibilities:

- Monitoring the MGT input data and detecting ISQ violations caused by alterations to the Aurora IP module. The standard ISQ is a pseudo-random sequence generated from a linear feedback shift register (LFSR) module defined by the Aurora functional specifications. Figure 5.10 illustrates the LFSR module generating the Aurora standard ISQ as acquired from the functional specifications [173]. The ISQ LFSR generator is developed within the ISQ monitor as the ISQ functionality reference. The ISQ-specific monitor circuit decodes the parallel input stream looking for the ISQ control characters `/A/`, `/K/`, `/R/`. A symbol detection module aware of the Aurora control characters is developed to identify and track these characters. The monitor asserts at run-time that the ISQ generated by the Aurora IP matches the standard ISQ generated internally. Variations between the actual and standard ISQs indicates a problem with the Aurora ISQ functionality. The monitor immediately overrides the compromised ISQ with the standard ISQ in response to violation detection.

- Monitoring the MGT output data and detecting ISQ violations on the RX side. A second instance of the ISQ LFSR is instantiated in the ISQ monitor to set a reference for the RX stream. The ISQ specific monitor circuit decodes the parallel output stream looking for the




```
send_K = send_idle & (!send_idle_dlyd | send_idle_dlyd & !Acntr_eq_zero & pseudo_random_bit)
send_A = send_idle & send_idle_dlyd & Acntr_eq_zero
send_R = send_idle & send_idle_dlyd & !Acntr_eq_zero & !pseudo_random_bit
```

Figure 5.10: ISQ generator module

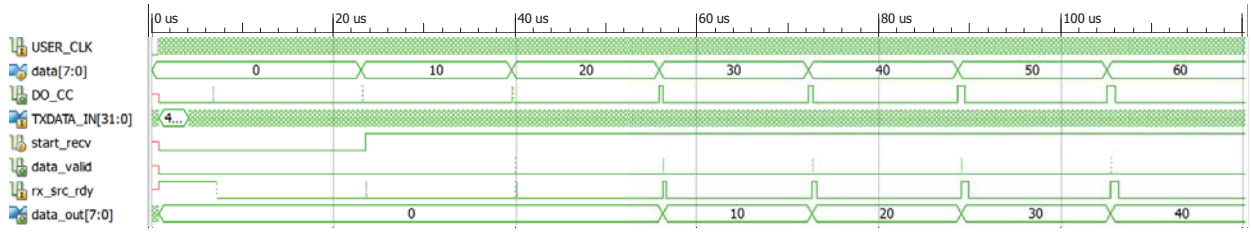
ISQ control characters `/A/`, `/K/`, and `/R/`. The monitor asserts at run-time that the ISQ received by the MGT physical module matches the standard ISQ generated internally. Variations between the actual and standard ISQs indicates that the RX stream is compromised. In this case, overriding a compromised RX stream does not affirm that covert information encoded in the stream have not been received elsewhere prior to detection. For example, the covert communication decoder module can be embedded in the MGT physical module. Therefore, the best countermeasure in this case is alerting the DREC and disabling the affected modules. The aforementioned cases demonstrate that detecting violations at their sources enables tolerating the associated threats which can be unaffordable if violations are detected elsewhere.

- Monitoring the MGT output data and detecting CC violations on the RX side. The ISQ-specific monitor circuit decodes the parallel output stream looking for the CC control character. Incorporating the CC functional specifications, this monitor is developed to decode the RX CC control character at run-time and assert that the CC period and duration match the standard, predefined values. This monitor can detect different CC variations on the RX side. Similar to the ISQ RX monitor, the best countermeasure in this case is alerting the DREC and disabling the affected modules.

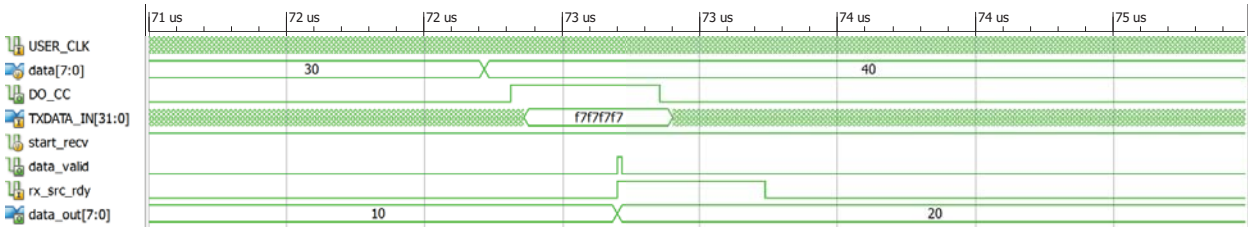
5.3 Results and Evaluation

To demonstrate the functionality of the developed HTHs and interface guards, we develop a testbench connecting two infected Aurora modules in a loopback configuration. Figure 5.11 illustrates selected waveforms from the developed testbench for the CC covert communication attacks and specific countermeasures. In the CC PWM attack, the `data` signal modulates the `DO_CC` duration to send a covert message to the RX node as shown by Figure 5.11(a). During the CC period, the control character `/CC/` is repeatedly sourced to the `TXDATA_IN` signal as shown by Figure 5.11(b). In the CC PCM attack, the `data` signal switches `DO_CC` between logic '1' and '0' to send a covert message to the RX node as shown by Figure 5.11(c) and Figure 5.11(d). On the RX side, the CC RX HTHs decode the covert messages by probing the `rx_src_rdy` signal to get `data_out`. The CC TX-specific interface guard detects the CC variations and overrides `DO_CC` with the standard CC signal generated internally by the guard as shown in Figure 5.11(e). The duration and period of the standard CC are fixed as shown by Figure 5.11(f).

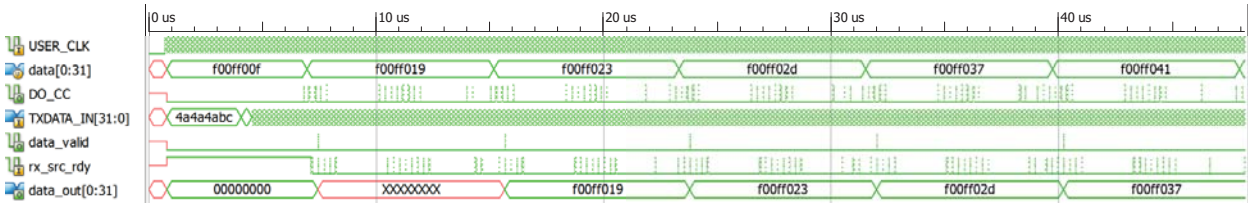
Figure 5.12 illustrates selected waveforms from a developed testbench for the ISQ covert communication attack and countermeasures. During the wait state indicated by `TX_VALID`, the `data` signal modulates the ISQ of `TXDATA` to send a covert message to the RX node as shown by Figure 5.12(a). On the RX side, the ISQ RX HTH decodes the covert message to get `data_out` during wait states indicated by the `data_valid` signal. The duration between consecutive `/A/`



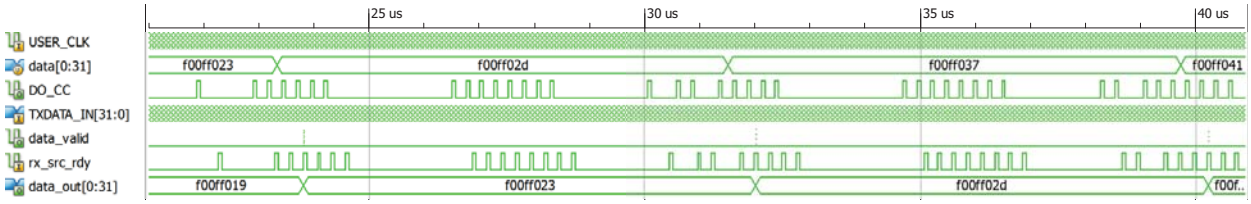
(a) CC PWM covert communication waveform.



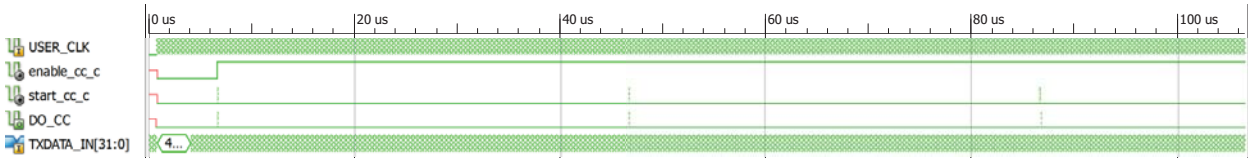
(b) Zoomed snapshot of the CC PWM covert communication waveform.



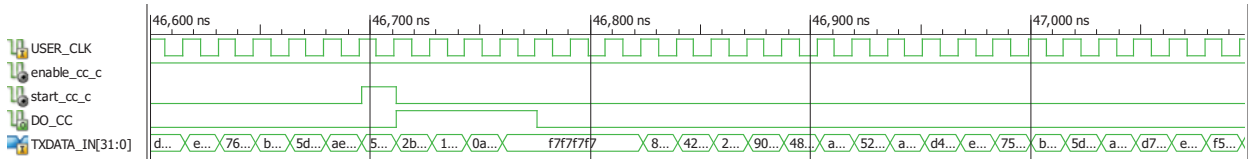
(c) CC PCM covert communication waveform.



(d) Zoomed snapshot of the CC PCM covert communication waveform.



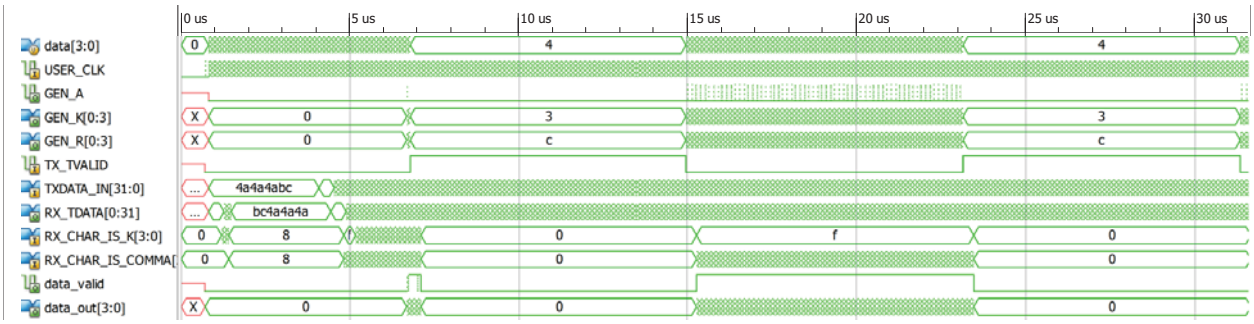
(e) Standard enforced CC waveform.



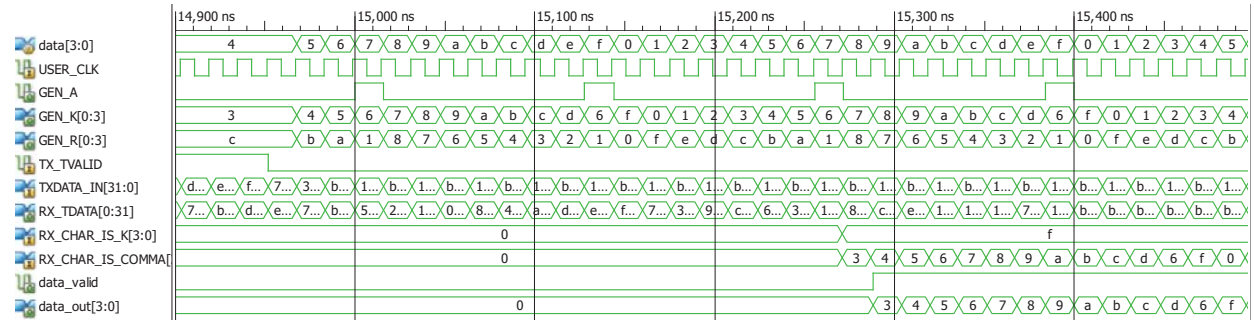
(f) Zoomed snapshot of the standard enforced CC waveform.

Figure 5.11: CC covert communication attacks and countermeasures waveform.

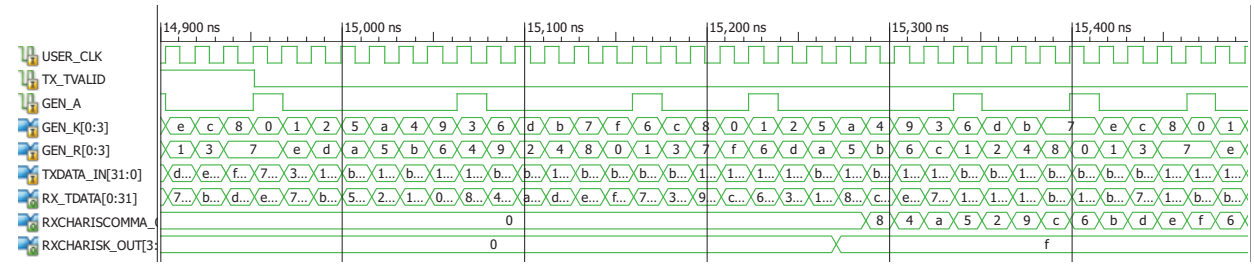
characters is fixed as shown by Figure 5.12(b). The ISQ TX-specific interface guard detects the ISQ variations and overrides the `TXDATA` signal with the standard ISQ generated internally by the guard in which the duration between consecutive `/A/` characters is random as shown in Figure 5.12(c).



(a) ISQ covert communication waveform.



(b) Zoomed snapshot of the ISQ covert communication waveform.



(c) Zoomed snapshot of the standard enforced ISQ waveform.

Figure 5.12: ISQ covert communication attack and countermeasure waveform.

The developed testbench is experimentally validated using the RocketIO transceivers on a Xilinx Virtex-5 FX130T FPGA. The Xilinx CoreGen tool provides HDL for the Aurora core. Two MGTs running at 2.5 Gbps line rate and 125 MHz reference clock frequency are connected on a custom MGT interface board attached to a Xilinx ML510 evaluation platform. The maximum bit rates of

the PCM and the PWM covert communication are 12.5 and 23.43 Mbps (for n=3), respectively, at 10% and 25% channel utilization. Moreover, the bit rate and channel utilization are parametric functions with tunable design parameters that can be adjusted to achieve significant data rates at marginal channel utilizations. Established pulse modulation and digital encoding practices simplify the design effort. In a test with a 125 MHz reference clock frequency, 4-byte lane width, and assuming 10% idle time, the ISQ-based covert channel communication bit rate is 600 Mbps while the bandwidth is 60 Mbps without decreasing the main channel bandwidth. Table 5.2 shows the incremental resource utilization of the HTHs in a 10GBASE-X IP core on a Xilinx Virtex-7 FPGA. Table 5.3 shows the absolute and relative resource utilization of TX and RX HTHs in an Aurora IP core on a Xilinx Virtex-5 FPGA. The overheads induced by the developed HTHs are marginal whereas the covert communication achieved data rates are significant.

Table 5.2: Xilinx Virtex-7 FPGA resource utilization for HTHs in a 10GBASE-X IP core (% of PCS/PMA core).

	# Regs	**# LUTs**
TX HTH	16 (0.7%)	13 (0.5%)
RX HTH	5 (0.23%)	5 (0.2%)
PCS/PMA IP	2146	2473

Table 5.3: XC5VFX130T FPGA resource utilization for HTHs in Multi-gigabit Transceivers (% of Aurora core).

	# Regs	**# LUTs**
PCM HTH (TX, RX)	24, 20 (1.4%, 1.2%)	27, 31 (1.4%, 1.6%)
PWM HTH (TX, RX)	22, 17 (1.3%, 1%)	34, 29 (1.8%, 1.5%)
ISQ HTH (TX, RX)	16, 5 (1%, 0.3%)	13, 5 (0.7%, 0.3%)
Aurora IP	1630	1870

Hardware guards are instantiated in the HDL top-level module to probe the Aurora core boundary and MGT interface. The guards comprise sequential and combinational monitors capturing multiple interface specifications, and are controlled by an FSM responsible for enforcement. Synthesis results and experimental tests of the developed guards on the XC5VFX130T FPGA show that the

maximum operating frequency of the developed guards is 432.8 MHz and the enforcement latency is 2.9 ns. Preventing attacks demands low latency countermeasures such as disabling the module. For example, defending against the ISQ-based covert channel attack requires detection of the threat and enforcement of its associated countermeasure in a single clock cycle. If the MGT reference clock frequency is 125 MHz, the maximum allowed latency to stop the attack is 8 ns which is achieved using the developed hardware guard. Table 5.4 shows the resources utilization of the deployed guards. Interface guards resource requirements are usually independent of the monitored module's implementation complexity.

Table 5.4: XC5VFX130T FPGA resource utilization for interface guards (% of Aurora core).

	# Regs	**# LUTs**
Aurora generic interface guard	57 (3.5%)	45 (2.4%)
MGT generic interface guard	69 (4.2%)	53 (2.8%)
Aurora-specific interface guard	50 (3.1%)	46 (2.5%)
MGT-specific interface guard	73 (4.5%)	61 (3.3%)
Aurora IP	1630	1870

Alternatives to the hardware-based interface guards are either software-implemented guards where both monitoring and enforcement are in software, or hybrid guards where both software and hardware are used. To assess these choices, assume a software-based interface guard utilizes a GPIO to probe a module interface and transfer data to a processor. Also assume that a dedicated embedded processor such as MicroBlaze is used to implement a single interface guard. For a MicroBlaze processor running at its maximum operating frequency of 125 MHz clock frequency in an XC5VFX130T FPGA, the measured latency to read a GPIO is 90 cycles (0.72 µs), and the write latency is 99 cycles (0.792 µs). This implies that the maximum operating frequency of modules protected by such guard cannot exceed 661.4 KHz. Software attack susceptibility and unacceptable latency make the use of software-based solutions impractical in the run-time protection of many hardware-implemented subsystems.

5.4 Summary

In this chapter, we presented an application of RETC-CPS to untrusted hardware IP cores deployed in various computing platforms. This application supports the RETC functionality of protecting low-level components against potential cyber threats. In particular, we applied RETC to untrusted hardware components vulnerable to HTHs. We provided a brief background about HTHs as an emerging threat to computing platforms, and some challenges confronting HTH detection methods. The run-time security approach to protect against HTH threats was motivated by enumerating the design-time detection method limitations.

We advanced novel implementations of HTHs enabling covert channel communication in 10GbE physical links and high-speed serial point-to-point physical links operated by the Aurora link-layer protocol IP. The developed HTHs exploit loose specifications of the underlying media link protocols to send and receive covert messages without violating the IP legal operation. The HTHs are activated under rarely occurring conditions and induce marginal area and power overheads to evade detection by design-time techniques. Significant covert communication data rates can be achieved with the developed HTHs, and multiple parameters are provided to change the data rate and channel utilization. The maximum bit rates of the PCM, PWM, and ISQ covert communication are 12.5, 23.43 Mbps, and 60 Mbps at 10%, 25%, and 0% channel utilization,respectively. The developed HTHs provides examples of threats raised by incorporating untrusted components in computing platforms.

We discussed how to use run-time protections to tolerate HTH threats in untrusted hardware components, and proposed two protection classes in Section 5.2. Generic guards are hardware security modules integrated into the module under protection interfaces to monitor abstract module attributes such as the data rate, detect and analyze anomalous variations to the monitored attributes, and raise threat alarms to the DREC. Assurances provided by the generic interface guards neither confirms the attack nor determines the threat origin and, consequently, countermeasures associated with generic guards are passive responses such as threat reporting. Specific guards are hardware components to enforce the IP functional specifications. We provided several developments of

both generic and specific guards thwarting HTHs, and assessed overheads compared to the total resources of the FPGA evaluation platform.

Functional simulations and timing diagrams of a testbench connecting two infected Aurora modules demonstrated the HTH effects and interface guard countermeasures. The developed testbench was experimentally validated using the RocketIO transceivers on a Xilinx Virtex-5 FX130T FPGA. The maximum operating frequency of the developed guards is 432.8 MHz and the enforcement latency is 2.9 ns. If the MGT reference clock frequency is 125 MHz, the maximum allowed latency to stop the attack is 8 ns which is achieved using the developed hardware guard. The measured latency of a guard processor running at its maximum operating frequency of 125 MHz is 1.5 µs. These results illustrate that the hardware guard is 500$\times$ faster than alternative software defenses employing a dedicated processor per guard. Run-time violations are rapidly detected and acted upon without incurring latencies by software or non-local communication. Overheads were marginal compared to the total resources consumed by the Aurora IP core. Trust is localized in Aurora guards consuming less than 10% of the total IP resources, reducing the verification effort by a factor of 10. As a result, the performance, developer productivity, and security requirements of data-intensive platforms are simultaneously addressed.

Chapter 6

Exploratory Application to Process Control Systems

Process control systems are a subset of CPSes adopting feedback control, where an embedded controller uses sensor measurements of a physical plant to compute feedback signals preserving system stability. They are widely used in infrastructure and safety-critical applications such as power grids, assembly lines, water systems, pipelines, industrial systems, and power plants [28, 40]. Process controllers are usually built using untrusted components and third-party IP cores. Recent attacks against process control systems such as Stuxnet, which is described as the real start of cyber warfare, have highlighted CPS vulnerabilities and the inadequacy of existing security solutions. This chapter illustrates how to apply the RETC architecture to protect model-based designs exemplified by an aircraft pitch control system. It also depicts how to formulate a system-level security policy using system physical models and specifications which are enforced at the controller's top-level interface.

In this chapter we apply the RETC-CPS protection scheme to build secure embedded controllers containing untrusted components. We exploit the model-based approach in a novel way to develop RETC guards in process control applications by predicting and preempting erroneous controller behavior at run-time. The process models normally developed when designing a process control

system capture the system-level behavior to be enforced by RETC. Prediction logic incorporates a second instance of the embedded controller connected to a physical plant model running faster than real time in order to predict the future state of the physical system. The model's state is periodically synchronized with the physical plant's state to prevent divergence. DREC interface guards check if future system states will violate system-level policies. If a violation is detected, enforcers switch from the faulty controller to a lower-performance and high-assurance backup controller until the system is stabilized. For process control networks, RETC may be collectively applied over the network of controllers, sensors, and supervisory software.

We generate RETC components to simultaneously address design-for-security, and -trust in process control systems. System-level reliability is also enhanced by incorporating specifications that should already be defined for high-reliability systems. RETC addresses several aspects of control security, including high-assurance module interactions and reconfiguration management. The DREC serves as the most privileged root-of-trust for control flow, and inserts a distributed set of policy-aware interface guards on module boundaries. Interface guards provide on-line monitoring and proactive enforcement of policy rules emanating from security, performance, or reliability specifications. The DREC may also contain specialized logic to increase system reliability, such as redundant controller modules or violation prediction logic. DREC reconfiguration allows policy changes, but DREC logic can only be updated by a trusted authority.

An exploratory application of RETC-CPS to process control systems is presented in this chapter. The remainder of this chapter is organized as follows: A survey of existing approaches to process control system reliability and security is presented in Section 6.1. A brief description of some basic process control concepts and introduces an aircraft pitch control system example is given in Section 6.2. Concepts, models, and simulation results for pitch control prediction and preemption are illustrated in Section 6.3. This chapter is summarized in Section 6.4.

6.1 Existing Approaches to Process Control System Security, Trust, and Reliability

Run-time fault detection techniques in process control typically observe either physical process measurements to new controller inputs or controller responses to new sensor measurements. Sha introduced a protection architecture based on monitoring physical process measurements to detect faults [150]. In this architecture, sensors measurements of the physical process are monitored by decision logic that determines if a process violation has occurred, as illustrated by Figure 6.1. If a violation is detected, the decision logic switches control to a high-assurance and presumably slower version of the controller until the system is stabilized. Unfortunately, system stability cannot always be recovered as the controller fault is not detected until after it has caused the physical process to deviate from allowed operational limits.

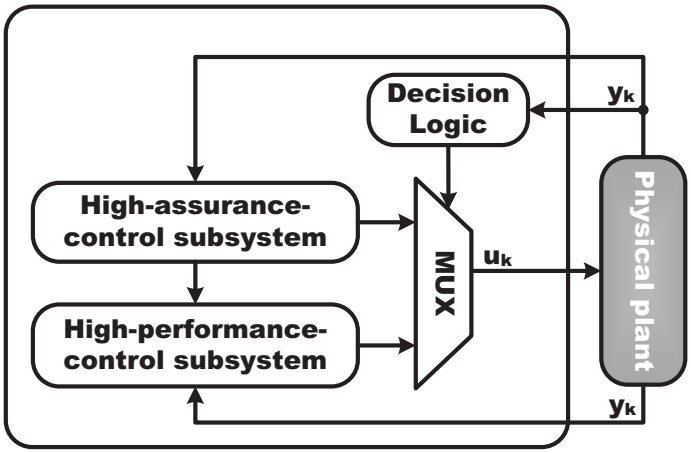


Figure 6.1: Plant fault detection [150].

Dai *et al*. advanced a fault detection architecture based on observing controller responses to new sensor inputs [42]. Physical process measurements are sent to both the regular high-performance version of the process controller and a trusted benchmark version of the controller algorithm. The responses of both controllers are used to generate a residual to determine if a controller fault has occurred, as shown in Figure 6.2. Unfortunately, the physical process is already affected by the

erroneous controller output by the time the controller fault is detected and corrective actions, such as switching over to a high-assurance version of the controller, can occur. This may result in the inability to return the system to a stable state before damage is incurred.

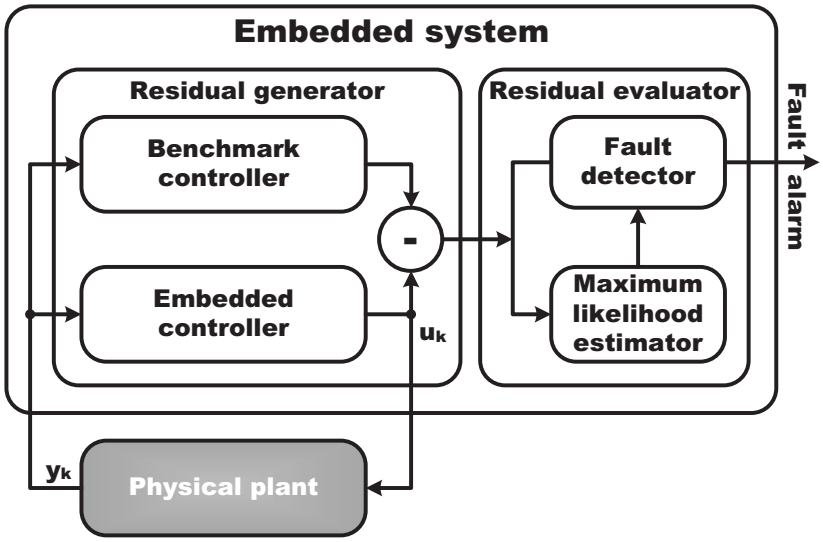


Figure 6.2: Controller fault detection [42].

Cárdenas *et al*. presented a physical model-based attack detection method complementing intrusion detection methods for networks and computer systems [31]. Threats addressed include embedded controller intrusion attacks arising from compromised measurements of plant sensors. Instead of using models of network traffic or software behavior, physical system models are used to develop a change detection-based intrusion detection algorithm. An embedded system implementation of a physical plant linear model runs concurrently with the plant, as illustrated in Figure 6.3. A controller response is sent to both the physical plant and the embedded model. In normal operating conditions, physical plant measurements are sent to the embedded controller to compute the feedback response. During intrusion detection, the embedded model output is sent to the embedded controller to filter out compromised measurements. This work develops a rigorous analysis and classification of intrusion attacks against process control systems and associated detection methods. However, it does not provide a means to detect and circumvent Trojan threats to controllers.

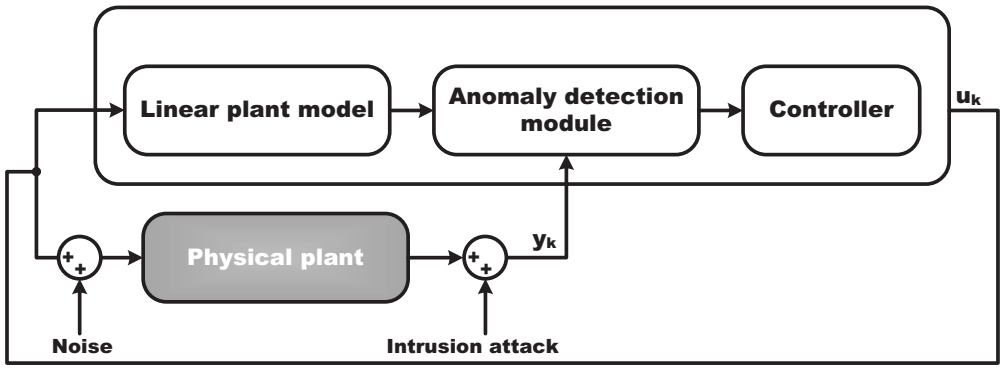


Figure 6.3: Controller intrusion attack detection [31].

Although many security solutions have been proposed for legacy embedded systems [30], these solutions are not optimized for process control applications. Design-time security techniques are very expensive and may not anticipate all system vulnerabilities. Such techniques often do not address vulnerabilities raised by software patches and updates, hardware reconfigurations, and zero-day exploits. This leads to a demonstrated possibility of controllers being surreptitiously compromised. The alternative approach is admitting the possibility of unanticipated threats and trying to cope with them using run-time security defenses. However, existing run-time protections are reactive, and can only detect erroneous controller behavior after its occurrence. Such detection methods may allow a physical processes to become unstable before corrective action can be taken. These techniques also tend to be threat-specific, leading to increased overheads for required defenses.

6.2 An Aircraft Pitch Model and Control

The motion of an aircraft is governed by a set of six nonlinear differential equations. These equations can be decoupled into longitudinal and lateral equations under certain assumptions [116]. The pitch angle is a third-order longitudinal problem and is controlled by adjusting the angle of the rear elevator. Figure 6.4 shows the basic coordinate axes and forces acting on an aircraft.

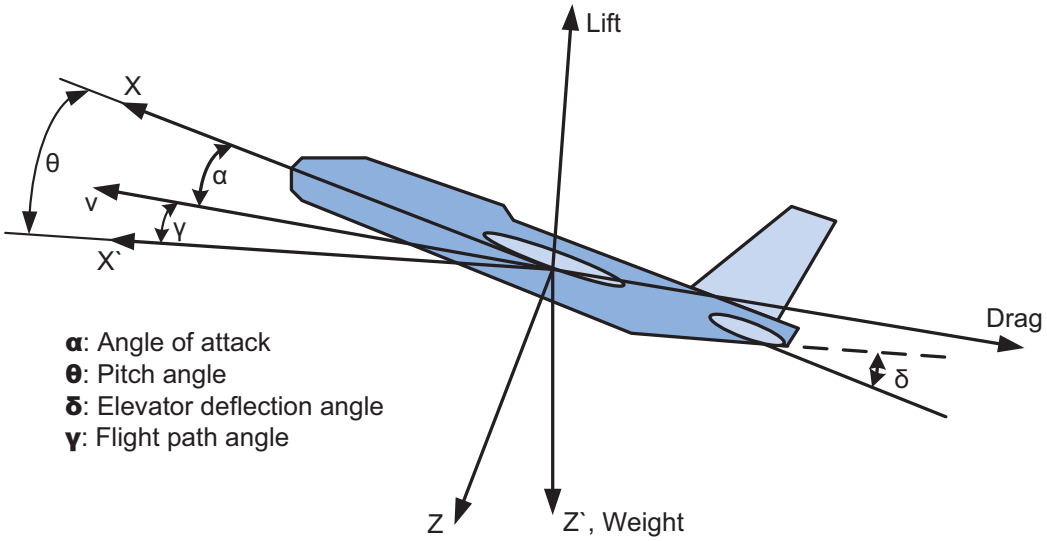


Figure 6.4: Coordinate axes and forces acting on an aircraft

As described in [116], the equations of motion of a Boeing commercial aircraft can be written as:

$$\dot{\alpha} = -0.313\alpha + 56.7q + 0.232\delta_e$$

$$\dot{q} = -0.0139\alpha - 0.426q + 0.0203\delta_e$$

$$\dot{\theta} = 56.7q$$

where α is the angle of attack; q is the pitch rate; θ is the pitch angle; and δ_e is the elevator deflection angle.

Using the differential equations controlling the plane motion, the state space representation of the pitch angle system is:

$$\begin{bmatrix} \dot{\alpha} \\ \dot{q} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -0.313 & 56.7 & 0 \\ -0.0139 & -0.426 & 0 \\ 0 & 56.7 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0.232 \\ 0.0203 \\ 0 \end{bmatrix} \begin{bmatrix} \delta_e \end{bmatrix}$$

$$y = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} \delta_e \end{bmatrix}$$

In the presence of noise, this equation can be written as:

$$\dot{x} = Ax + Bu + \omega_{proc}$$

$$y = Cx + Du + v_{sensor}$$

where x is a column matrix composed of α, q, and θ elements representing system's state; the input u is the elevator deflection angle δ_e; the output y is the pitch angle θ; and ω_{proc} and v_{sensor} are the process and measurement noise, respectively. The noise is assumed to be Gaussian distributed with zero-mean and constant variance.

The autopilot uses a feedback controller in a closed-loop configuration to stabilize the aircraft by adjusting its attitude angle. For this system, the input is the elevator deflection angle and the output is the pitch angle. Linear Quadratic Gaussian (LQG) control is one of the most commonly used optimal control techniques, and combines a Kalman filter (a linear-quadratic estimator) with a linear-quadratic regulator [163]. The Kalman filter optimally estimates the state of a linear system disturbed by noise. Figure 6.5 shows the structure of the LQG control technique.

Knowing the system state helps maintain synchronization between the physical system and our predictive subsystem. In the pitch control application we develop a discrete LQG controller, as is commonly done in modern control theory. Design constraints include limits on the maximum overshoot, rise time, settling time, and steady-state error. Figure 6.6 illustrates the step response of the pitch controller system for both the open-loop system and the closed-loop feedback control configurations.

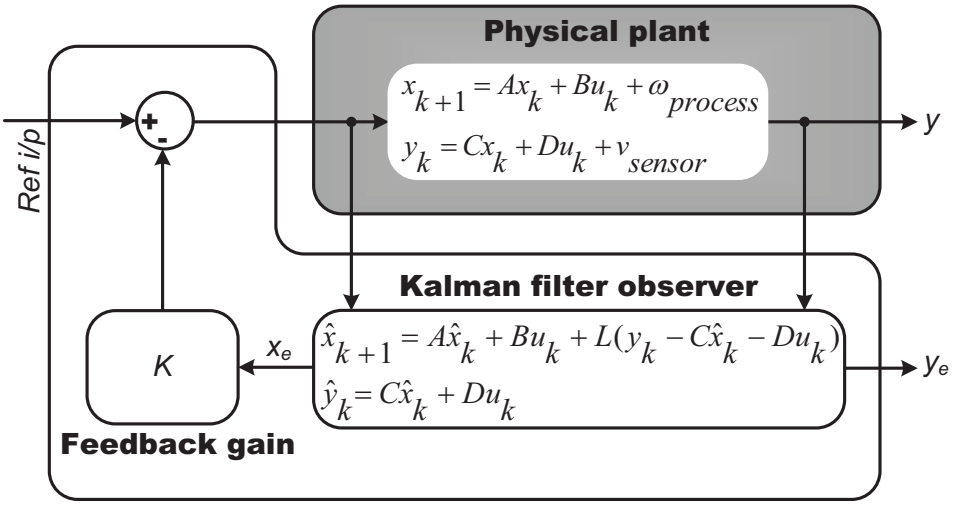


Figure 6.5: LQG control architecture.

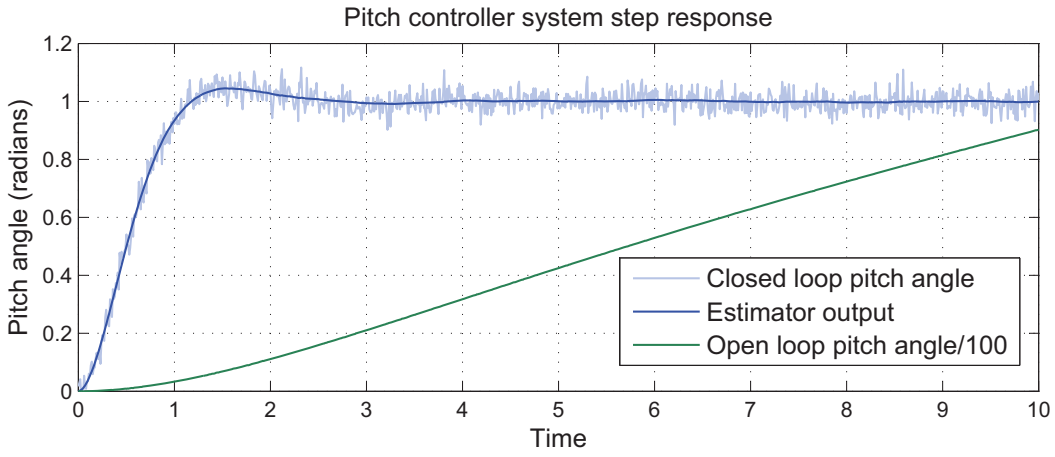


Figure 6.6: Open- and closed-loop step response of the pitch angle control system.

6.3 Controller Attack Prediction and Preemption

The goal of this example is to illustrate how to apply RETC-CPS in order to protect embedded controllers using the RETC security architecture. Trust is not required in the controller or its update and communication infrastructure, and applies only to a small set of simple, self-contained, and verifiable add-ons. The DREC ensures that security and reliability specifications are being observed, and essentially serves the role of an ideal control room operator. Interface guards enable

the DREC to directly monitor system operation and override the controller under protection. The RETC solution incorporates security enhancements to the system structure and automatic tool extensions to the existing design flow.

Our threat model does not distinguish between hardware faults, software bugs, and malware such as Trojans since the common denominator is non-compliant controller behavior. A cyber threat can hide itself using sophisticated means, but is less able to disguise its ultimate goal of disturbing system stability. Regardless of how the threat originates, the focus is anticipation and disabling of any negative consequences to the controlled process. Based on this philosophy, we present a novel method to predict and preempt erroneous behavior in physical process control. For the control domain, specifications for normal system behavior are already known, and accurate models for the controlled process usually exist. Our protection scheme is complementary to other approaches that try to validate the design or prevent malware infiltration, and serves as a last line of defense against various threats to embedded controllers.

6.3.1 Concepts

Physical systems and processes are characterized by quantitative temporal properties such as process response time, actuator delays, and sensor time constants. These physical latencies are not inherent in system models. The vast majority of physical processes can be described and modeled as linear time invariant systems with a high degree of accuracy under very realistic assumptions. Often, a plant model running on an embedded processor can be executed much faster than a real plant operating in a physical system. In a feedback control system, an embedded controller responds to variations in the state of a physical plant in order to maintain system stability. Execution speed of an embedded controller corresponds to the temporal characteristics of the associated physical process. The typical operating frequency of a digital embedded system controlling a physical process is proportional to the sampling rate of the physical process. Our approach to detect erroneous behavior of embedded controllers exploits potential speed differences between a physical plant and its model, which is analogous to the difference between running a physical system and

simulating it.

The main idea to predict faults is examining what the controller implementation will try to do in the future by embedding a second instance of the controller with an accelerated model of the plant. The model can be implemented in either reconfigurable hardware or software (perhaps on a separate processor) depending on the required speed-up. The second controller instance should be identical to the original controller and implemented on the same platform. To maintain convergence with the physical system, the model's state is periodically synchronized with the plant's state. The embedded controller instance should be subject to the same conditions as the active controller by synchronizing the model's input with the system reference input, and applying the same patches and updates to both instances. A redundant embedded subsystem incorporating these measures can accurately predict the behavior of an embedded controller for a certain period of future time.

6.3.2 Controller Organization

Basic concepts of feedback control and modern control systems are presented in [48, 61]. In order to enhance the security, trust, and reliability of an embedded controller, the existing system is augmented with modules synthesized from a process model and specifications. As shown in Figure 6.7, major components of the predictive and preemptive architecture are:

- A control system containing an active controller and a high-assurance backup controller, and a switching mechanism. This embedded system runs at the typical sampling rate of the physical plant.
- A predictive subsystem consisting of a plant model and a second instance of the embedded controller. This subsystem runs n times faster than the active control system. The model's state is periodically synchronized with the estimated state of the physical plant.
- A DREC wrapping the control system and predictive subsystem, containing interface guard monitors and enforcers, and synchronization and timing blocks.

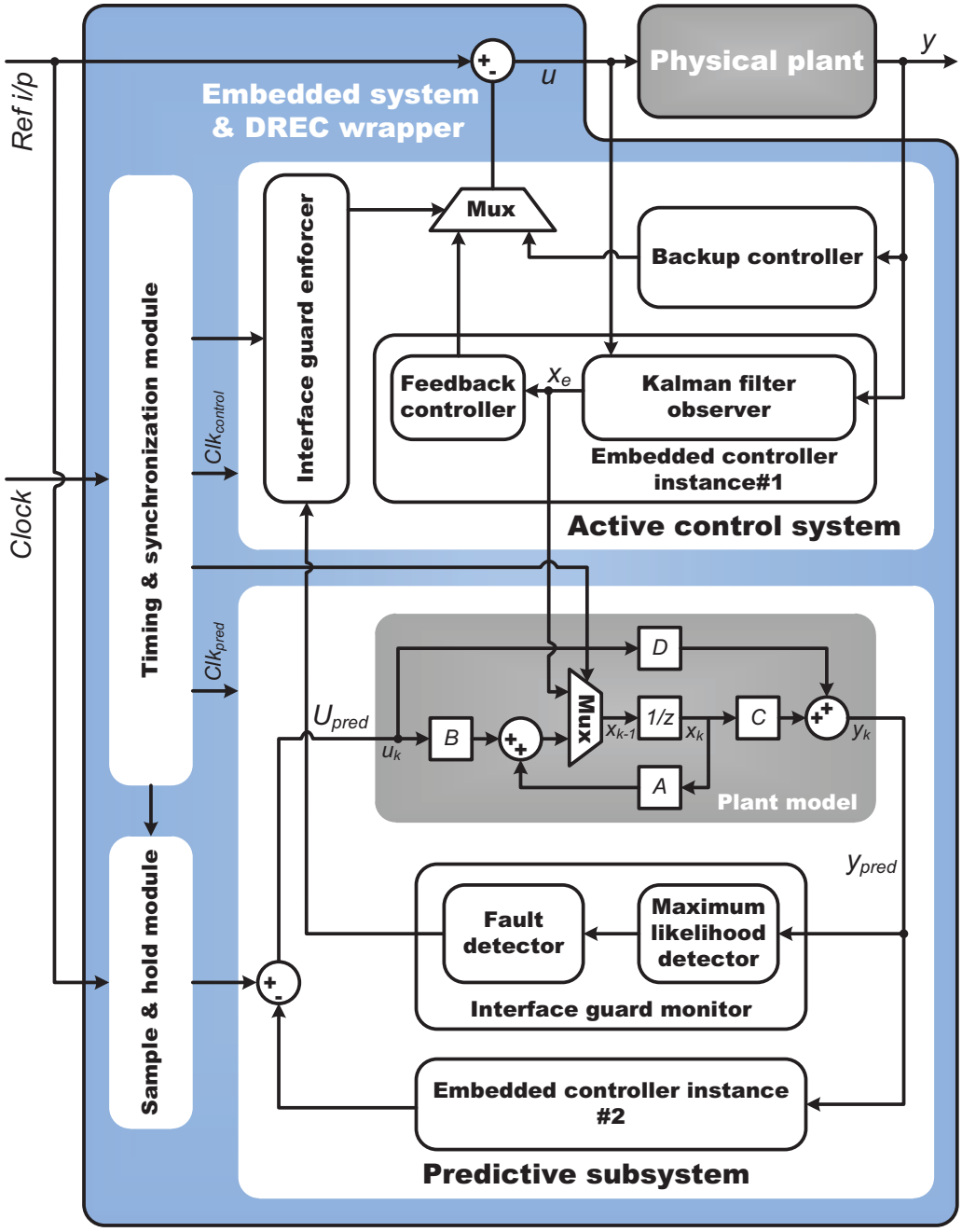


Figure 6.7: Predictive and preemptive DREC security architecture.

Interface guards can monitor either the physical plant or the embedded controller input/output activity to assure compliance with the desired behavior of the physical process or a benchmark controller. In our architecture, an interface guard monitor is integrated to the plant model in the

predictive subsystem, as shown by Figure 6.7. Detection of anomalous behavior in the predictive subsystem triggers the enforcer to switch the active controller of the physical plant to a high-assurance controller. Such an approach provides preemptive countermeasures in safety-critical process control systems. Recursion is possible with more than one backup controller and their predictive counterparts. The synchronization module is responsible for periodically updating the state of the model with the estimated state of the physical plant. A sample and hold module updates the predictive subsystem input with the physical plant reference input. The timing module is responsible for clock generation and time emulation for the predictive subsystem.

The monitoring guard module employs a maximum likelihood detector and a fault detector to predict faults before they really occur in the actual controller. Theoretically, if the predictive controller is secure and no threats are affecting it, the output of the controller will conform to the normal operating criteria described by the security policy and embedded in the fault detector module. Any threat affecting the predictive controller, which will show up later in the actual controller if not preempted, will result in a significant change in the predictive controller's output. In practice, however, many other factors other than security threats can cause such a deviation from the normal operating conditions such as miss-tuning of the controller parameters and more importantly the process random noise. The challenge is how to distinguish faults resulted by cyber attacks from the noise or faults caused by other reasons. Such a distinction requires an accurate description of the controller characteristics, operating conditions, and the noise statical distribution.

The fault detection module does not relay on a single sample to decide the controller's integrity. For process control systems, it has been shown that as the number of the observed samples increase, the statistical distribution of the process noise becomes Gaussian with constant mean and variance values [42]. Consequently, the statistical distribution of the controller's output follows the Gaussian noise distribution as it is the only random variable in the output equation. Deviation from the normal operating conditions caused by cyber threats and controller faults shifts the controller's output mean and variance computed over a significant number of samples to new values outside the range defined by the security policy. In this example, the fault detector is developed to detect the mean change through run-time monitoring. Other fault detection techniques attempt to detect

faults by inspecting the variance change, which can be more accurate yet expensive in terms of the computation and resource overheads. The maximum likelihood estimator computes the mean of the output moving samples and sends the results to the fault detector. Based on the updated mean, the detector tests the controller's output to validate if the likelihood ratio lies within the predefined threshold range captured by the security policy. Once the predictive controller's output is out of range, the detector raises an alarm to the interface guard enforcer to switch to the backup controller to preempt the fault before its occurrence in the actual controller.

6.3.3 Time Projection and Synchronization

Our approach advances new terminology such as: the time scaling factor n which indicates the predictive subsystem speed-up; the prediction window W_{pred} which denotes the foreseen time period; and synchronization time T_{sync} which determines the updating frequency of the model's state in the predictive subsystem. W_{pred} is function of n and T_{sync} as shown by equation (6.1). T_{sync} is application-dependent whereas n is both application- and platform-dependent. Assuming flexibility in assigning n and T_{sync}, increasing n improves W_{pred} at zero cost in terms of the updating frequency, while increasing T_{sync} augments W_{pred} on the expense of reducing the updating frequency. T_{sync} is often the more flexible and tunable parameter when significant changes to W_{pred} are needed. Multiple trade-offs must be evaluated when assigning values of n and T_{sync} where the physical process characteristics and the embedded platform features are the assignment criteria.

$$W_{pred} = n \cdot T_{sync} \tag{6.1}$$

Time scaling is accomplished by applying modifications to both system and input signals. System modifications vary for continuous and discrete time models of the physical process. For a continuous time model, time scaling is achieved by multiplying system state space matrices by the desired time scaling factor n. For discrete time systems such as those employed in embedded controllers, scaling down the sampling time of the physical process by a factor of n automatically

scales down the time of the system internal signals. Input signal time cannot be scaled down because this requires prior knowledge of signal contents. To tackle this problem, the model's input can be periodically synchronized with the reference input at the physical system sampling rate by assigning T_{sync} to be $T_{sampling}$ seconds. However, this approach limits the prediction window to $n \cdot T_{sampling}$ seconds.

Another approach can be adopted where the plant model and the physical system are synchronized whenever the reference input to the physical system is changed. Such an approach produces an adaptive prediction window, which may not be preferred for security reasons. In feedback control, the reference input to a physical process is often the desired stable output of the system which implies that reference input changes are limited in terms of both amplitude and frequency. This implies that a sample and hold technique can be used to periodically update the model's input with the reference input without the need for an adaptive synchronization method. We adopt this approach to establish a security scheme with a controllable synchronization time and a fixed prediction window.

6.3.4 Simulation Results and Evaluation

In order to illustrate and evaluate our approach, the aircraft autopilot pitch controller is used as a case study [116]. Flight control is a safety-critical application where controller faults can have catastrophic consequences. LQG control is a modern approach adopting time-domain analysis, state space representations, and state observers to enhance the control process. It concerns uncertain linear systems disturbed by additive white Gaussian noise and undergoing control subject to quadratic costs [163]. LQG controllers are widely deployed, and their structure helps to present our concepts and architecture effectively. Nevertheless, our approach is still applicable to other control techniques.

Matlab Simulink is used to model and evaluate the application and security enhancements. Commonly used model-based design flows permit software or hardware implementations to be automat-

ically synthesized from Matlab. In our experiments, the sensor sampling rate is 100 samples/sec, the time scaling factor n is 10, and two values of T_{sync} (1 and 10 seconds) are used for different prediction windows. Figure 6.8 illustrates the step response of a stable pitch control system versus the predictive subsystem, and synchronizes between both systems for the test case of T_{sync} = 1 second. A moving window allows periodic projection of the future system state from the updated current system state.

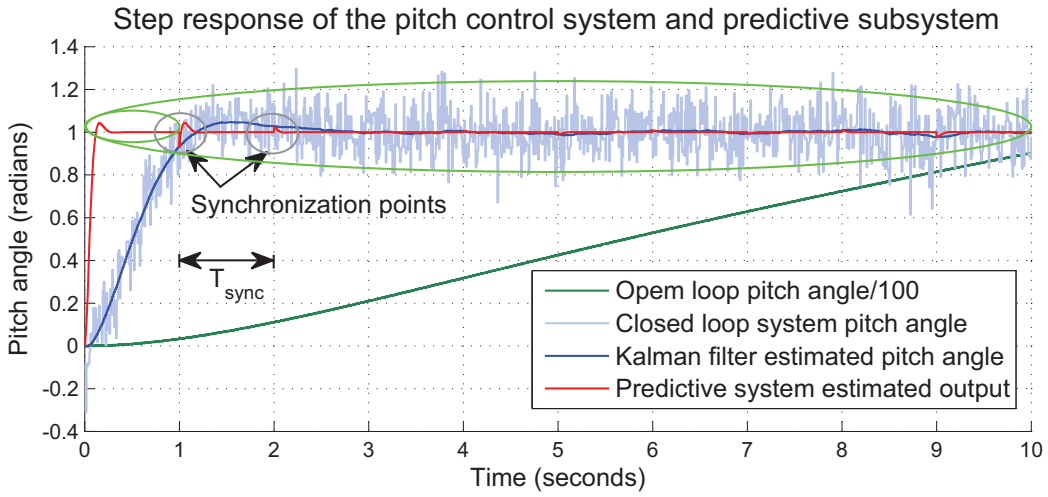


Figure 6.8: Step response of a stable pitch control system versus predictive subsystem.

Faults and cyber threats in embedded systems can be roughly classified into event-driven and time-driven faults. Accurate detection of event-driven faults depends on the proper and frequent updating of the model's state in the predictive subsystem. Figure 6.7 shows the switching technique created to update the model's state x_k with the estimated plant's state x_e generated by the Kalman filter state observer. The model's state updating frequency is a function of the desired prediction window and the model's input synchronization scheme. Predictive subsystem time must emulate the real time in order to successfully detect time-driven faults and properly operate time-driven modules and processes. Time emulation requires generating the predictive subsystem time in terms of n and T_{sync}, and relating it to the real time t. The predictive subsystem time is directly proportional to n, whereas T_{sync} formulates the reference time base which periodically resets the predictive time t_{pred} to the real time t as shown by equation (6.2). Figure 6.9 illustrates the predictive subsystem time for the selected case studies.

$$t_{pred} = T_{sync} \cdot \lfloor \frac{t}{T_{sync}} \rfloor + n \cdot \text{mod}(\frac{t}{T_{sync}}) \tag{6.2}$$

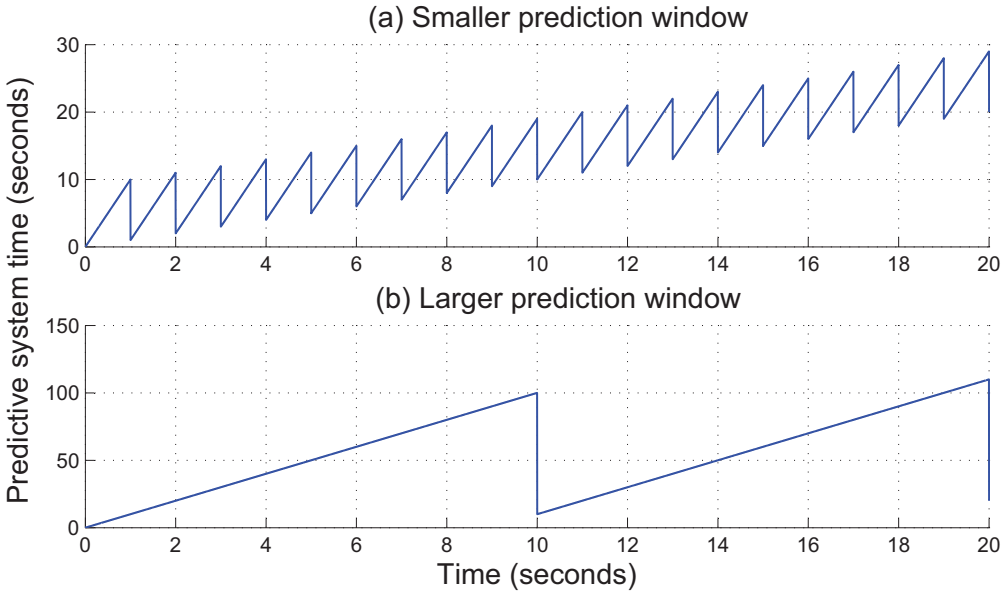


Figure 6.9: Relationship between real time and predictive subsystem time.

To illustrate the effectiveness of our approach in predicting and preempting erroneous behavior, we insert a time-driven fault in the embedded controller's model. The fault is injected by manipulating a single element of the gain matrix in the feedback controller model. This fault is kept dormant for a pre-assigned period of time as shown by Figure 6.10(a). Such a fault can be inserted during aircraft maintenance as a system patch or update to a rigorously verified embedded controller, and can force the plane out of its stable state in a very short time. Figure 6.10(b) and (d) illustrate the predictive subsystem output for different windows. Initial system stability is assumed. The smaller prediction window predicts the fault 10 seconds before its occurrence, and the larger prediction window foresees the fault about 50 seconds before its occurrence. Smaller windows are more accurate but see the fault later. As time advances, the predictive subsystem anticipates the fault as the sawtooth waveform with increasing peaks indicating fault advancement.

In the presence of noise, physical system faults can be detected with the aid of a Kalman filter innovation sequence characterized by a zero mean and a fixed covariance matrix in normal operation

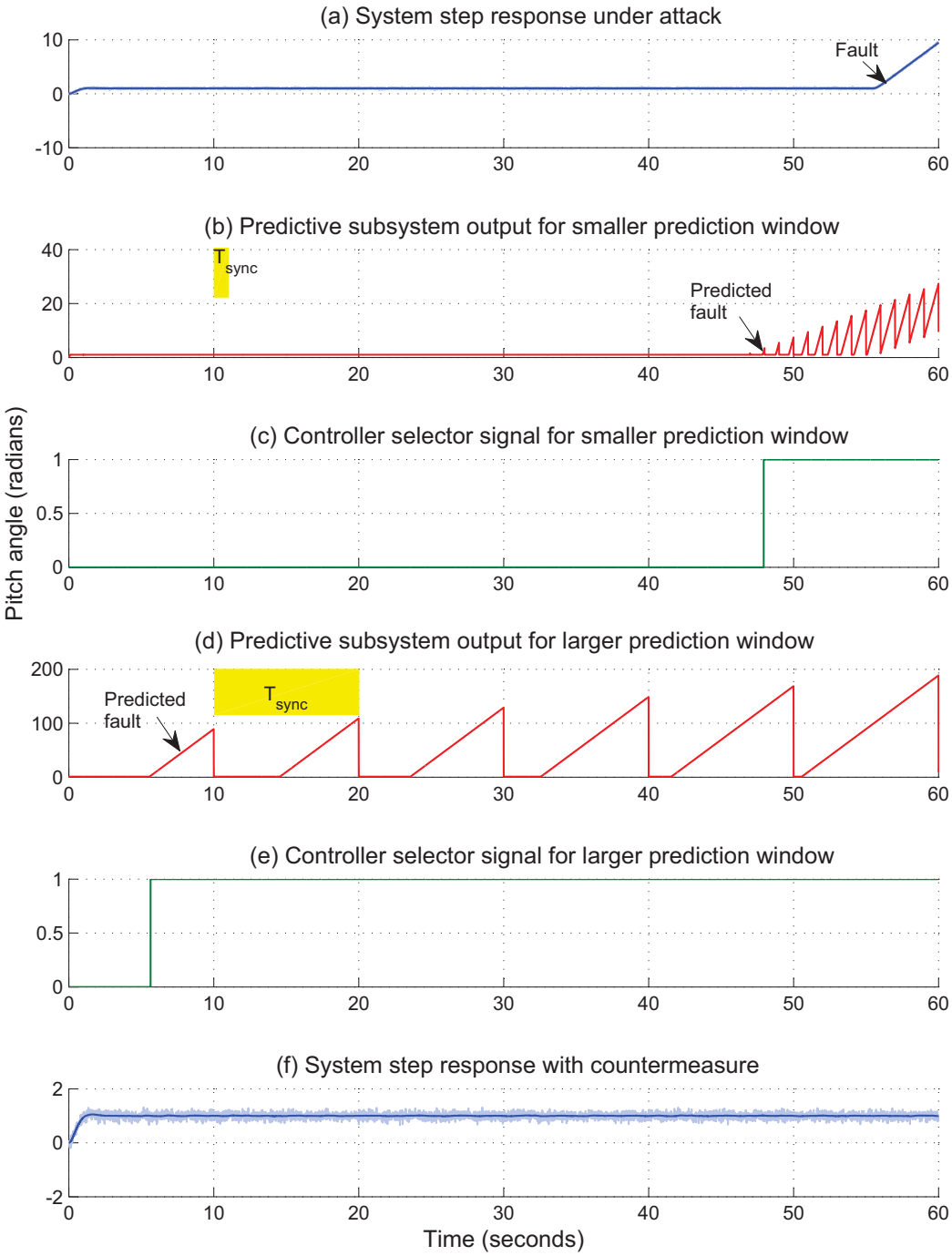


Figure 6.10: Step response of a pitch control system and the predictive subsystem under attack.

conditions [70]. Our approach to detect faults employs a maximum likelihood detector module to evaluate the root mean square (RMS) value of the model's output in the predictive subsystem.

Deviation of the output RMS value from nominal bounds indicates a fault sensed by the detector module shown in Figure 6.7. Fault detection in the predictive subsystem triggers the interface guard enforcer to switch from the active controller to the backup controller. The backup controller used in this evaluation is simply the developed fault-free LQG controller. Figure 6.10(c) and (e) show the multiplexer selection signal and the time of switching for the two selected prediction windows. For both case studies, the countermeasure has successfully prevented faults occurring as illustrated by Figure 6.10(f).

This proof-of-concept experiment was performed by modeling the process and our protection system in Simulink. Future experiments will examine a fully hardware-oriented root-of-trust. Simulink profiler results for the pitch controller model are given in Figure 6.11. One minute of real time is simulated in 22 sec of CPU time on a single core of a 2.80 GHz Intel Core i7 workstation with 24GB of memory and running the Linux 2.6.32 kernel. In practice, optimized C implementations would run more quickly than the Simulink models.

The predictive subsystem and active control system design complexity are almost the same, and the DREC contributes about 25% of the total complexity as measure in terms of total software methods. However, it is anticipated that the overhead of the root-of-trust will shrink significantly for a hardware-based implementation. Most of the CPU time is consumed by the predictive subsystem as it runs n times faster than the active controller. On a multicore platform, however, the predictive subsystem runs concurrently with the active controller.

6.4 Summary

In this chapter RETC-CPS was applied to build a DREC protecting a safety-critical process control system against cyber threats. This application illustrates the RETC-CPS mitigation of cyber attacks on CPSes containing untrusted components. The DREC is a system-level interface guard monitoring top-level system interfaces, and enforcing application-oriented security policies emanating from the system's physical characteristics. Background on the established security approaches to

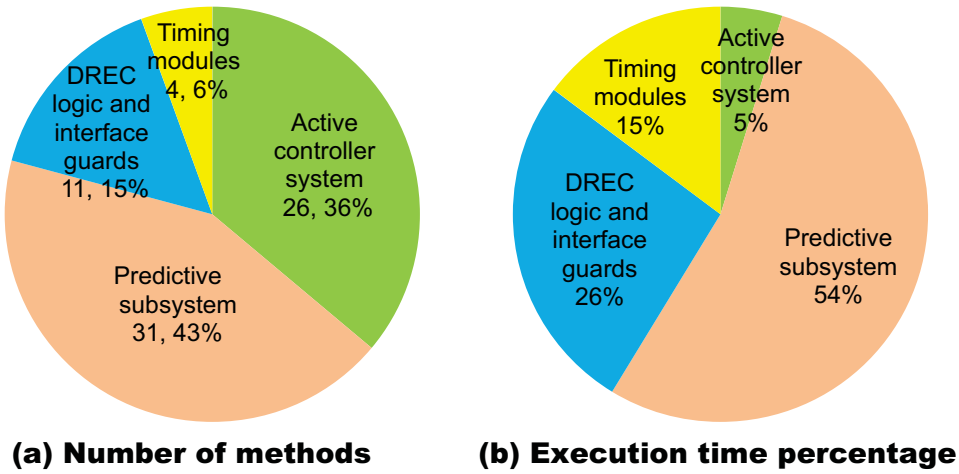


Figure 6.11: Pitch controller static/dynamic analysis

process control systems motivated the need for proactive run-time defenses preserving the physical process stability during potential cyber attacks. The pitch controller application introduced basic process control concepts.

Unlike previous RETC applications, unique process control system characteristics were exploited not only to protect the system against potential attacks, but also to predict and preempt these threats before their occurrence in real time. The DREC incorporates a second instance of the embedded controller connected to a physical plant model running faster than real time in order to predict the future state of the physical system. The model's state is periodically synchronized with the physical plant's state to prevent divergence. Erroneous controller behavior is detected before it affects the physical process, allowing preemptive alarms or actions. Simulation results and timing diagrams for the pitch controller illustrated that the DREC can predict erroneous behaviors and switch to a high-assurance backup controller to preserve process stability. Trust is localized in the DREC which contributes about 25% of the total complexity measured in terms of total software methods and, thus, the verification effort has been reduced by a factor of 4. This application also established the feasibility of integrating the RETC flow within the model-based design flow commonly used during process control systems development.

Chapter 7

Conclusions

As cyber technology becomes more pervasive in the physical world, the need to enhance CPS security increases. The reason underlying the work in this dissertation is that design-time security solutions are necessary but not sufficient to protect CPSes. Complementary run-time protections should be adopted to serve as a last line of defense against cyber threats evading detection by design-time solutions. Run-time protections must be trustworthy, threat-tolerant, extensible, efficient, and tailored to a certain application class. We advanced the RETC-CPS protection scheme to increase trust in CPSes containing untrusted software and hardware and built using reconfigurable hardware. This scheme targets systems and applications having clearly framed and enforceable security policies.

RETC enhances trust in platform reconfiguration, low-level components, and system-level interactions. RETC was illustrated with three applications having research challenges commonly arising in CPSes containing untrusted components. First, we provided an analysis of the CR security requirements, a high-level architecture of CHARE, a security framework realizing these requirements, and prototype development supporting the CHARE architecture. Second, RETC-CPS was applied to third-party hardware components with prototype development of novel HTHs and RETC low-level guards thwarting the associated threats. Third, RETC-CPS was applied to process control systems with a model-based design containing the RETC high-level security protections.

In this dissertation, the main focus has been to present high-level architectures, prototype developments, and threat models to establish the implementation feasibility, effectiveness, efficiency, and functional correctness of the RETC protection scheme. Although this work does not address RETC design automation, a detailed design flow established automation feasibility in Chapter 3. Applications are carefully selected to demonstrate the RETC main functionalities, applicability to a wide range of CPSes, and efficacy in tolerating various threats raised by incorporating untrusted components in CPSes. The RETC prototypes were developed in reconfigurable hardware, yet the concepts presented are generally applicable to software-based systems as well as ASICs and SoCs. Hardware trust anchors simultaneously address system security and performance requirements.

7.1 Review of Contributions

Introducing a threat-tolerant security scheme

Existing design-time security approaches seek to avoid threats by developing trusted application-specific solutions according to rigorous security practices and standards. However, such approaches fail to provide modern, complex systems with the desired comprehensive security. The RETC-CPS protection scheme does not rely on such techniques to verify that a system under protection is vulnerability-free, and augments the system with run-time interface guards monitoring and enforcing application-specific security policies. Run-time protections serve as a last line of defense against potential threats evading detection by design-time techniques. We presented RETC trust anchors for three different applications in Chapters 4, 5, 6. Applying RETC-CPS to different CPSes at different hierarchy levels demonstrates the protection scheme generality and scalability. Automation feasibility was established for the CHARE framework presented in Chapter 4. RETC functional correctness was established by validating the RETC prototypes against custom-developed testbenches addressing specific threat models commonly arising in CPSes. The developed testbenches were experimentally validated on a Xilinx Virtex-5 FX130T FPGA, and the testing results illustrated the RETC efficacy in thwarting the selected threats.

Localization of trust

Existing efforts to build trustworthy systems struggle to assure trust in all system components and interactions pre-deployment. Such solutions are extremely expensive and they often cannot guarantee that systems are vulnerability-free. The alternative is localizing trust in a small set of system modules and using these modules to enhance trusted computing in all system components and interactions. In the RETC protection scheme, trust is required in only a small set of simple, localized, verifiable, and self-contained guard components. We proposed a high-level architecture realizing these requirements, and presented the RETC-CPS requirements, assumptions, limitations, expectations, and design flow in Chapter 3. The prototypes were evaluated in terms of the performance and overheads indicating the achieved localization of trust and the verification effort reduction factor.

Trust anchors added by RETC-CPS include the SRC, configuration firewall, low-level interface guards, and DREC. The SRC is a hardware module responsible for receiving update or reconfiguration requests from untrusted system components and validating these requests according to a predefined reconfiguration policy. The configuration firewall is a static hardware module which receives update commands from the SRC, and securely manages transfer, authentication, and reconfiguration of the requested PRM bitstreams. A prototype development of the SRC and reconfiguration firewall for a CR was presented in Chapter 4. Implementation results demonstrated that overheads do not exceed 10% of the total platform resources, leaving over 90% of the resources available for the CR datapath. Low-level interface guards are hardware trust anchors wrapping untrusted components and enforcing low-level, circuit-dependent security policies at the component interfaces. A prototype development of RETC low-level interface guards protecting third-party IP cores widely employed in reconfigurable platforms was introduced in Chapter 5. Implementation results illustrated that the hardware guard is 500$\times$ faster than software defenses. Trust is localized in RETC guards consuming less than 10% of the total IP resources, reducing the verification effort by a factor of 10. The DREC is a hardware component wrapping the system under protection and enforcing a system-level, application-oriented security policy at the top-level system interface. A model-based design of the DREC in process control CPSes was advanced in Chapter 6. Trust was

localized in the DREC contributing about 25% of the total complexity measured in terms of total software methods and the verification effort was reduced by a factor of 4.

Secure reconfiguration control

Secure reconfiguration control was ensured by RETC-CPS, illustrated by a CR CPS demanding frequent platform updates in Chapter 4. The main threat models addressed by this application were compromised update requests issued by untrusted software and open communication channels used to deliver update contents. The CHARE-CR framework incorporates the SRC, reconfiguration firewall, and a trusted DMS to address reconfiguration threats. We presented the CHARE high-level architecture mapped reconfigurable platforms. The CHARE framework enhances trust in CR by adding a simple and verifiable hardware module responsible for platform adaptation without relying on software correctness or needing to software verification. Although the CHARE framework was originally developed for CR, concepts associated are generally applicable to other CPSes built using reconfigurable hardware.

CR DSA policies described in CoRaL, a domain-specific formal language based on first-order logic, were translated into hardware assertions to implement the SRC responsible for validating update requests issued by the policy engine software. Automation feasibility was established by providing concrete examples demonstrating the mapping from CoRaL to SystemVerilog HDL for all CoRaL syntactic constructs. We presented a prototype hardware development of the SRC for a CR operating in the Radar "S" band. State-of-the-art cryptographic and authentication standards were employed to implement a hardware/software reconfiguration firewall responsible for authenticating update contents delivered via open communication channels. Prototype reconfigurable implementations were provided for the SRC and reconfiguration firewall. A custom-developed testbench generating potential reconfiguration attacks was run on the Xilinx Virtex-5 FX130T FPGA to validate CHARE functional correctness. The SRC can detect compromised transmission requests regardless of software correctness.

Addressing new vulnerabilities and measures

MGTs are the preferred technology for high-speed communication in modern systems. We intro-

duced novel HTHs enabling covert communication in 10GbE physical links and high-speed serial point-to-point physical links by exploiting loose specifications in the underlying link-layer protocol 5. The maximum bit rates of the PCM, PWM, and ISQ covert communication are 12.5, 23.43, and 60 Mbps at 10%, 25%, and 0% channel utilization, respectively. System and module specifications can be necessarily loose to enhance portability or enable certain applications such as autonomic computing. As shown in this chapter, such ambiguities can be leveraged to violate system-level security policies while maintaining legal operation as defined in design specifications. Thus, verification of functional specifications alone is an inadequate method of evaluating and providing system security, and even specifications with rigorous provisions to enhance security may be insufficient to prevent new attacks. The attacks also illustrated security threats arising from the use of third-party IP to assemble computing platforms.

In response, we presented the RETC-CPS low-level protections employing both generic and specific interface guards to tolerate HTH threats in untrusted hardware components. RETC-CPS treats the module under protection as a black box without an associated golden reference. Security primitives were built to enforce abstract and specific IP functional specifications at the module's interfaces. Hardware was developed for both the HTHs and interface guards thwarting the associated threats. HTH and interface guard overheads were minor compared to the resources consumed by the IP blocks. A testbench demonstrated RETC interface guard countermeasures for HTH attempts at covert communication. The developed testbench was experimentally validated using the RocketIO transceivers on a Xilinx Virtex-5 FX130T FPGA. Run-time violations were rapidly detected and acted upon without latencies incurred by software or non-local communication.

Predicting and preempting erroneous controller behaviors

RETC-CPS application to process control systems was explored in Chapter 6. The DREC is a system-level guard monitoring top-level interfaces and enforcing application-oriented security policies capturing physical characteristics. The DREC high-level architecture was presented and illustrated by a model-based design for an aircraft pitch control application. The run-time system included a second instance of the active controller connected to a model of the plant giving a short-term projection of future controller actions and process state. The model's state was periodically

synchronized with the plant's state to prevent divergence. A simulation testbench generated time-triggered controller faults to validate the functional correctness of the DREC. Erroneous controller behavior was detected before it affects the physical process, allowing preemptive alarms or actions. The predictive subsystem and active control system design complexity are almost the same, and the CHARE trust anchor contributes about 25% of the total complexity as measure in terms of total software methods.

Enabling trust extensibility in reconfigurable architectures

The key enablers to trust extensibility in reconfigurable platforms are: decoupled system implementations and trust anchors, secure control of system reconfiguration, and separating the system and security design flows. The RETC-CPS security guards are localized and integrated to the protected system interfaces. RETC's policy-based approach allows evolution of trust components independent of particular system implementations. This was clearly depicted in the RETC-CPS high-level architecture presented in Chapter 3 and supported by the selected applications. Secure reconfiguration control enables trust in system security updates, as demonstrated by the CHARE framework presented in Chapter 4. RETC-CPS treats systems under protection as black boxes and relies on enforcing security policies independent of particular system implementations to isolate functionality and security This property was supported by the RETC-CPS design flow in reconfigurable hardware presented in Chapter 3, and illustrated by the selected applications.

7.2 Strengths and Limitations

We note the following strengths of the RETC-CPS protection scheme:

- It addresses several threats associated with untrusted components regardless of the threat origin because of the run-time monitoring approach adopted.
- Trust can be enhanced at the system-level or on a per-module basis or even on a per-function basis.

- It works across simulation and emulation and detects threats after deployment with the same trust anchors.
- System operation is not affected by the security components.
- Trust anchors are not fixed and can be added, removed, or swapped to enable new security mechanisms.
- RETC does not require either trust in the IP vendor or golden reference models, and treats the protected modules as black boxes
- RETC employs functional specifications revealed by the IP developers to enhance security at the IP interfaces.
- RETC does not require significant changes to the design flow.

Limitations include:

- RETC can only protect systems and components having clearly stated security policies.
- RETC induces area overheads and time delays which, in certain cases, can be significant.
- Countermeasures enforced by RETC can mitigate threat consequences but cannot guarantee normal system operation.
- Security policies may not contain all permitted and prohibited behaviors. For instance, if the security policy is derived from the IP functional specifications, threats that do not affect IP operation cannot be detected.
- Framing security policies for all system functionalities can be intractable.
- Not all cyber threats can be inferred and detected at component or system interfaces.
- Even hardware protections may not preclude zero-cycle attacks such as immediate DoS.
- The RETC protections can be bypassed by a malicious entity aware of the security scheme.

- Hardware protections do not enable detection of all violations targeting different software layers.

7.3 Future Work

The results of this research indicate that run-time security protections can enhance trust in CPSes, but as with any research, questions arise along the way that require further research. Future work directions are summarized in the following points:

- **Design automation**

  As mentioned in Chapter 3, lack of automation is a major bottleneck confronting many security solutions. Design automation was not the main focus of this dissertation, yet a detailed design flow was proposed in Chapter 3, and the automation feasibility was established for the CHARE framework presented in Chapter 4. We plan to develop tools to automatically generate the RETC trust anchors in both the model-based and language-based security design flows for the presented applications. This includes development of a trusted DMS responsible for automatic generation and delivery of the hardware plug-ins incorporating the RETC trust anchors and protected reconfigurable modules. In particular, we will focus on model-based design to generate solutions for problem domains that have a well-established mathematical basis, such as process control, signal processing, and communications. PSL will be investigated as a standard policy language to enable automatic design and verification of the RETC components in the policy-based design flow. Design experiences presented in [78] will be leveraged to automate the RETC-CPS component generation.

- **New threats arising from predictive architectures**

  The DREC architecture introduced in Chapter 6 enables predicting future controller behavior and preempting controller faults. The major advantage is gaining significant time that might be required to enforce countermeasures preserving physical process stability. Preliminary results demonstrate possible time gains in a safety-critical application. However,

we still need to investigate the protection scheme applicability to different process control systems by characterizing possible time scaling factors and prediction windows for various applications. Moreover, the trust anchors added to the process control system make it significantly more complex, and complexity is the enemy of security and trust. Subtle new attack mechanisms can be introduced to compromise the DREC protections such as a maliciously designed control algorithm exploiting the knowledge that it will be used for both prediction and production. We will investigate possible new threats arising from predictive architectures, and enhancements required to the DREC to address such threats.

- **Development of cross-layer defenses**

  Most of the major advances in computer system security rely on hardware-based security architectures for enforcement. The RETC-CPS hardware protection scheme simultaneously addresses the performance, developer productivity, and security requirements for the selected applications and threat models. However, as indicated in Chapter 3, hardware-based security cannot address all cyber threats associated with software or user layers. We plan to investigate cross-layer security approaches to enable trusted computing in CPSes for complex threat models. Specifically, we will focus on run-time, cross-layer security protections for PLCs to extend RETC-CPS applicability from configurable hardware platforms to software-based architectures extensively deployed in CPSes.

- **Establishing mathematical models, formal verification, and trust evaluation methods**

  The main focus of this work has been to present high-level architectures and prototypes for run-time protections, addressing specific threat models commonly arising in CPSes, and validating the protection scheme's ability to thwart these threats. This approach is common practice in the security research community to demonstrate the solution efficiency against different threats. However, the presented architectures need rigorous mathematical models that are formally verified. We plan to develop mathematical models for the RETC-CPS trust anchors and apply the model checking techniques introduced in Chapter 3. Trust inherited by adopting RETC-CPS needs to be evaluated for the whole design rather than solely relying on

trust anchor verification. We plan to evaluate trust of the presented applications using attack surface measurement techniques [110].

- **Run-time detection and mitigation of HTHs**

  As mentioned in Chapter 5, HTHs are emerging threats to embedded systems and CPSes with a potential to break security objectives without being detected using traditional solutions. In this work, we presented specific HTH threat models and countermeasures in the form of RETC-CPS low-level generic and specific interface guards. Future work will focus on using general guards to address the gap between loose design specifications and overall system security. The ability to automatically synthesize generic guards from succinct, high-level system operation and security specifications should provide an effective run-time protection scheme. The RETC low-level protections will be applied to a wider range of HTHs, especially those associated with software-like threat models. Moreover, we will investigate RETC applicability and required modifications to address side-channel enabled HTHs. Relying solely on specific guards is undesirable as they require low-level design expertise, and the number of specific guards needed may be excessive.

- **Application to networked CPSes**

  The emphasis of this dissertation has been on enhancing trust in individual embedded controllers deployed in a CPS assuming conventional network security practices. Novel HTHs enabling covert communication in cyber networks provided an example of unconventional threats resulting from lack of trust in individual network nodes. As a CPS is a system of systems featuring extensive network interactions and increasing number of nodes, we plan to extend the RETC-CPS protection scheme to enhance trust in cyber networks by developing a top-level network protection mechanism that can be applied in conjunction with individual controller defenses and RETC trust anchors to enforce overall system security. Formal methods will be applied to the extended protection scheme in order to investigate the state explosion problem on the CPS network scale. We plan to investigate design-for-verification methods to address potential state explosion by adding run-time monitors incor-

porating temporal logic assertions to dynamically verify specific security requirements in complex CPSes. The focus will be on extending the CHARE framework presented in Chapter 4 to enhance trust in CR networks against configuration attacks. A network model will be developed to predict the protection scheme efficiency in countering plausible spatial and temporal long-term effects raised by the cognitive nature of CR. This research may provide required assurances about the CR network and platform security to spectrum regulators and stakeholders, thus enabling wide deployment of this emerging technology.

Bibliography

[1] IEEE standard for property specification language PSL. *IEEE Std 1850-2005*, pages 1–143, 2005.

[2] A Roadmap for Cybersecurity Research, November 2009. http://www.cyber.st.dhs.gov/docs/DHS-Cybersecurity-Roadmap.pdf.

[3] IEEE standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements part 3: Carrier sense multiple access with collision detection CSMA/CD access method and physical layer specifications amendment 4: Media access control parameters, physical layers and management parameters for 40 Gb/s and 100 Gb/s operation. *IEEE Std 802.3ba-2010 (Amendment to IEEE Standard 802.3-2008)*, pages 1 –457, 22 2010.

[4] *NSF Workshop on the Future of Trustworthy Computing*, Waterview Conference Center, Arlington, VA, October 2010. http://tc2010.cse.psu.edu/index.html.

[5] Yael Abarbanel-Vinov, Neta Aizenbud-Reshef, Ilan Beer, Cindy Eisner, Daniel Geist, Tamir Heyman, Iris Reuveni, Eran Rippel, Irit Shitsevalov, Yaron Wolfsthal, and Tali Yatzkar-Haham. On the effective deployment of functional formal verification. *Formal Methods System Design*, 19(1):35–44, 2001.

[6] Miron Abramovici and Paul Bradley. Integrated circuit security: new threats and solutions. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence*

Research: Cyber Security and Information Intelligence Challenges and Strategies, CSIIRW '09, pages 55:1–55:3, New York, NY, USA, 2009. ACM.

[7] Ian F. Akyildiz, Won-Yeol Lee, Mehmet C. Vuran, and Shantidev Mohanty. NeXt generation dynamic spectrum access cognitive radio wireless networks: A survey. *Computer Networks*, 50(13):2127 – 2159, 2006.

[8] I.F. Akyildiz, Won-Yeol Lee, M.C. Vuran, and S. Mohanty. A survey on spectrum management in cognitive radio networks. *Communications Magazine, IEEE*, 46(4):40 –48, April 2008.

[9] M. Al-Morsy and H. Faheem. A new standard security policy language. *Potentials, IEEE*, 28(2):19 –26, march-april 2009.

[10] M. Anand, E. Cronin, M. Sherr, M. Blaze, Z. Ives, and I. Lee. Security challenges in next generation cyber physical systems. *Beyond SCADA: Networked Embedded Control for Cyber Physical Systems*, 2006.

[11] Robert H. Anderson and Richard Hundley. The implications of COTS vulnerabilities for the dod and critical U.S. infrastructures: What can/should the DoD do? Technical report, CA: RAND Corporation, Santa Monica, 1998. http://www.rand.org/pubs/papers/P8031.

[12] Ross Anderson, Frank Stajano, and Jong-Hyeon Lee. Security policies. Advances in Computers, pages 185 – 235. Elsevier, 2002.

[13] A.W. Appel. Foundational proof-carrying code. In *Logic in Computer Science, 2001. Proceedings. 16th Annual IEEE Symposium on*, pages 247 –256, 2001.

[14] K. Arkoudas, R. Chadha, and J. Chiang. An application of formal methods to cognitive radios. In *DIFTS'11: 1st INTERNATIONAL WORSKHOP ON DESIGN and IMPLEMENTATION OF FORMAL TOOLS AND SYSTEMS*, pages 3–12. Austin, TX, November 2011.

[15] ARM. ARM security technology: Building a secure system using TrustZone technology, July 1999.

[16] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha. Hardware-assisted run-time monitoring for secure program execution on embedded processors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(12):1295–1308, December 2006.

[17] Arvind, Nirav Dave, and Michael Katelman. Getting formal verification into design flow. In Jorge Cuellar, Tom Maibaum, and Kaisa Sere, editors, *FM 2008: Formal Methods*, volume 5014 of *Lecture Notes in Computer Science*, pages 12–32. Springer Berlin / Heidelberg, 2008.

[18] Abhijit Athavale and Carl Christensen. *High-Speed Serial I/O Made Simple*. Xilinx, 2005.

[19] B. Badrignans, J.L. Danger, G. Gogniat, V. Fischer, and L. Torres. *Security Trends for FPGAS: From Secured to Secure Reconfigurable Systems*. Springer Verlag, 2011.

[20] R. Baheti and H. Gill. Cyber-physical systems. *The Impact of Control Technology*, pages 161–166, 2011.

[21] B. Bahrak, A. Deshpande, M. Whitaker, and Jung-Min Park. Bresap: A policy reasoner for processing spectrum access policies represented by binary decision diagrams. In *New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on*, pages 1–12, April 2010.

[22] Christel Baier and Joost P. Katoen. *Principles of Model Checking*. The MIT Press, May 2008.

[23] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science*, pages 277–306. Springer Berlin / Heidelberg, 2004.

[24] Alex Baumgarten, Michael Steffen, Matthew Clausman, and Joseph Zambreno. A case study in hardware Trojan design and implementation. *International Journal of Information Security*, 10:1–14, 2011.

[25] J. Bergeron. *Verification methodology manual for SystemVerilog*. Springer-Verlag New York Inc, 2006.

[26] Michael Bilzor, Ted Huffmire, Cynthia Irvine, and Tim Levin. Security checkers: Detecting processor malicious inclusions at runtime. In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE InternationalSymposium on*, pages 34–39, June 2011.

[27] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Automatic hardware synthesis from specifications: A case study. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6, April 2007.

[28] M. Brundle and M. Naedele. Security for process control systems: An overview. *Security Privacy, IEEE*, 6(6):24–29, Nov–Dec 2008.

[29] J.L. Burbank. Security in cognitive radio networks: The required evolution in approaches to wireless network security. pages 1 –7, May 2008.

[30] A.A. Cárdenas, S. Amin, and S. Sastry. Secure control: Towards survivable cyber-physical systems. In *Distributed Computing Systems Workshops, 2008. ICDCS '08. 28th International Conference on*, pages 495–500, Jun 2008.

[31] Alvaro A. Cárdenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. Attacks against process control systems: risk assessment, detection, and response. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS'11, pages 355–366, 2011.

[32] Alvaro A. Cárdenas, Saurabh Amin, Bruno Sinopoli, Annarita Giani, Adrian Perrig, and Shankar Sastry. Challenges for securing cyber physical systems. 2010.

[33] A. Chandrasekharan, S. Rajagopalan, G. Subbarayan, T. Frangieh, Y. Iskander, S. Craven, and C. Patterson. Accelerating FPGA development through the automatic parallel application of standard implementation tools. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 53–60, December 2010.

[34] Feng Chen and Grigore Roşu. MOP: an efficient and generic runtime verification framework. *SIGPLAN Not.*, 42(10):569–588, October 2007.

[35] T.M. Chen. Stuxnet, the real start of cyber warfare? *Network, IEEE*, 24(6):2 –3, November–December 2010.

[36] T.M. Chen and S. Abu-Nimeh. Lessons from Stuxnet. *Computer*, 44(4):91 –93, April 2011.

[37] Steven Cherry. Sons of Stuxnet. *IEEE Spectrum*, December 2011. http://spectrum.ieee.org/podcast/telecom/security/sons-of-stuxnet.

[38] T.C. Clancy and N. Goergen. Security in cognitive radio networks: Threats and mitigation. pages 1 –8, May 2008.

[39] Edmund Clarke. The birth of model checking. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 1–26. Springer Berlin / Heidelberg, 2008.

[40] F. Cohen. Automated control system security. *Security Privacy, IEEE*, 8(5):62–63, Sep–Oct 2010.

[41] F. Dabiri and M. Potkonjak. Hardware aging-based software metering. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 460–465, april 2009.

[42] C. Dai, S.H. Yang, and Liansheng Tan. An approach for controller fault detection. In *Fifth World Conference on Intelligent Control and Automation (WCICA)*, volume 2, pages 1637–1641, Jun 2004.

[43] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The Ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, POLICY '01, pages 18–38, London, UK, 2001. Springer-Verlag.

[44] A. Datta, J. Franklin, D. Garg, Limin Jia, and D. Kaynar. On adversary models and compositional security. *Security Privacy, IEEE*, 9(3):26 –32, may-june 2011.

[45] J. Delorme, J. Martin, A. Nafkha, C. Moy, F. Clermidy, P. Leray, and J. Palicot. A FPGA partial reconfiguration design approach for cognitive radio based on NoC architecture. In *Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA 2008. 2008 Joint 6th International IEEE Northeast Workshop on*, pages 355 –358, June 2008.

[46] G. Denker, D. Elenius, R. Senanayake, M.-O. Stehr, and D. Wilkins. A policy engine for spectrum sharing. In *New Frontiers in Dynamic Spectrum Access Networks, 2007. DySPAN 2007. 2nd IEEE International Symposium on*, pages 55 –65, April 2007.

[47] G. Denker, D. Elenius, R. Senanayake, M.O. Stehr, C. Talcott, and D. Wilkins. Cognitive policy radio language (CoRaL) a language for spectrum policies, XG policy language. *Version 0.1, ICS-16763-TR-07-001, SRI*, 2007.

[48] Richard C. Dorf and Robert H. Bishop. *Modern Control Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 9th edition, 2000.

[49] R. Drechsler and G. Fey. Formal verification meets robustness checking: Techniques and challenges. In *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*, pages 4 –4, April 2010.

[50] Saar Drimer. Volatile FPGA design security – a survey, 2007.

[51] Saar Drimer. *Security for volatile FPGAs*. PhD thesis, November 2009.

[52] S. Drzevitzky. Proof-carrying hardware: Runtime formal verification for secure dynamic reconfiguration. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 255 –258, 31 2010-sept. 2 2010.

[53] S. Drzevitzky and M. Platzner. Achieving hardware security for reconfigurable systems on chip by a proof-carrying code approach. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*, pages 1 –8, June 2011.

[54] D. Dye. Partial reconfiguration of Virtex FPGAs in ISE 12. *Xilinx white paper WP374*, 2010.

[55] Thomas Eisenbarth, Tim Güneysu, Christof Paar, Ahmad-Reza Sadeghi, Dries Schellekens, and Marko Wolf. Reconfigurable trusted computing in hardware. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 15–20, New York, NY, USA, 2007. ACM.

[56] D. Elenius, G. Denker, M.-O. Stehr, R. Senanayake, C. Talcott, and D. Wilkins. CoRaL–policy language and reasoning techniques for spectrum policies. In *Policies for Distributed Systems and Networks, 2007. POLICY '07. Eighth IEEE International Workshop on*, pages 261 –265, June 2007.

[57] M.M. Farag, L.W. Lerner, and C.D. Patterson. Thwarting software attacks on data-intensive platforms with configurable hardware-assisted application rule enforcement. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 207–212, September 2011.

[58] M.M. Farag, L.W. Lerner, and C.D. Patterson. Interacting with hardware Trojans over a network. In *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*, pages 69–74, June 2012.

[59] Mohamemd M. Farag. Hardware implementation of advanced encryption standard on field programmable gate array. Master's thesis, Alexandria University, 2006.

[60] J.P. Farwell and R. Rohozinski. Stuxnet and the future of cyber war. *Survival*, 53(1):23–40, 2011.

[61] Gene F. Franklin, David J. Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 4th edition, 2001.

[62] K. Fysarakis, C. Manifavas, and K. Rantos. Embedded systems security. In *The 10th International Symposium on Ambient Intelligence and Embedded Systems*, AMIES 2011, 2011.

[63] A. Ginsberg, W.D. Home, and J.D. Poston. Community-based cognitive radio architecture: Policy-compliant innovation via the semantic web. In *New Frontiers in Dynamic Spectrum Access Networks, 2007. DySPAN 2007. 2nd IEEE International Symposium on*, pages 191 –201, april 2007.

[64] B. Glas, A. Klimm, O. Sander, K. Muller-Glaser, and J. Becker. A system architecture for reconfigurable trusted platforms. In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 541 –544, March 2008.

[65] Jonathan Graf and John Hallman. Trust in the FPGA supply chain using physically unclonable functions. In *The* 35^{th} *Annual Government Microcircuit Applications & Critical Technology Conference (GOMACTech)*, 2010.

[66] L.M. Grande. IEEE dynamic spectrum access policy standards work. In *Military Communications Conference, 2009. MILCOM 2009. IEEE*, pages 1 –4, October 2009.

[67] Johann Grossschadl, Tobias Vejda, and Dan Page. Reassessing the TCG specifications for trusted computing in mobile and embedded systems. In *Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, HST '08, pages 84–90, Washington, DC, USA, 2008. IEEE Computer Society.

[68] TRUSTED COMPUTING GROUP. Trusted computing group home page, 2009. http://www.trustedcomputinggroup.org.

[69] Jorge Guajardo, Sandeep Kumar, Geert-Jan Schrijen, and Pim Tuyls. FPGA intrinsic PUFs and their use for IP protection. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *LNCS*, pages 63–80. Springer, 2007.

[70] C.M. Hajiyev and F. Caliskan. Fault detection in flight control systems via innovation sequence of Kalman filter. In *Control '98. UKACC International Conference on*, pages 1528–1533 vol.2, September 1998.

[71] Melissa Hathaway. Cyberspace policy review and blueprint for research priorities. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '10, pages 11:1–11:1, New York, NY, USA, 2010. ACM.

[72] Constance Heitmeyer. Developing safety-critical systems: the role of formal methods and tools. In *Proceedings of the 10th Australian workshop on Safety critical systems and software - Volume 55*, SCS '05, pages 95–99, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[73] Lalana Kagal Hp and Lalana Kagal. A policy language for the me-centric project. Technical report, 2002.

[74] T. Huffmire, B. Brotherton, Gang Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine. Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems. In *Security and Privacy, IEEE Symposium on*, pages 281–295, May 2007.

[75] Ted Huffmire, Brett Brotherton, Nick Callegari, Jonathan Valamehr, Jeff White, Ryan Kastner, and Tim Sherwood. Designing secure systems on reconfigurable hardware. *ACM Trans. Des. Autom. Electron. Syst.*, 13:44:1–44:24, July 2008.

[76] Ted Huffmire, Timothy Levin, Thuy Nguyen, Cynthia Irvine, Brett Brotherton, Gang Wang, Timothy Sherwood, and Ryan Kastner. Security primitives for reconfigurable hardware-based systems. *ACM Trans. Reconfigurable Technol. Syst.*, 3:10:1–10:35, May 2010.

[77] Ted Huffmire, Shreyas Prasad, Tim Sherwood, and Ryan Kastner. Policy-driven memory protection for reconfigurable hardware. In *Proceedings of the 11th European conference on Research in Computer Security*, ESORICS'06, pages 461–478, Berlin, Heidelberg, 2006. Springer-Verlag.

[78] Ted Huffmire, Timothy Sherwood, Ryan Kastner, and Timothy Levin. Enforcing memory policy specifications in reconfigurable hardware. *Computers and Security*, 27(5-6):197–215, 2008.

[79] Intel Corp. Intel trusted execution technology overview.

[80] Intel Corp. Intel virtualization technology overview.

[81] Yier Jin, Nathan Kupp, and Yiorgos Makris. Experiences in hardware Trojan design and implementation. In *Proceedings of the 2009 IEEE International Workshop on Hardware-OrientedSecurity and Trust*, pages 50–57. IEEE Computer Society, 2009.

[82] Yier Jin, Eric Love, and Yiorgos Makris. Design for hardware trust. In Mohammad Tehranipoor and Cliff Wang, editors, *Introduction to Hardware Security and Trust*, pages 365–384. Springer New York, 2012.

[83] Edmund M. Clarke Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

[84] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor. Trustworthy hardware: Identifying and classifying hardware trojans. *Computer*, 43(10):39–46, October 2010.

[85] Ramesh Karri, Jeyavijayan Rajendran, and Kurt Rosenfeld. Trojan taxonomy. In *Introduction to Hardware Security and Trust*, pages 325–338. Springer New York, 2012.

[86] Ryan Kastner, Jason Oberg, Wei Huy, and Ali Irturk. Enforcing information flow guarantees in reconfigurable systems with mix-trusted IP. In *Engineering of Reconfigurable Systems and Algorithms (ERSA2011)*, July 2011.

[87] Tom Kean. Secure configuration of field programmable gate arrays. In *FPL '01: Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, pages 142–151, London, UK, 2001. Springer-Verlag.

[88] K. Kepa, F. Morgan, and K. Kosciuszkiewicz. Intellectual property protection in self-reconfigurable embedded systems. In *Consumer Electronics, 2009. ICCE '09. Digest of Technical Papers International Conference on*, pages 1–2, January 2009.

[89] K. Kepa, F. Morgan, K. Kosciuszkiewicz, and T. Surmacz. Serecon: a secure reconfiguration controller for self-reconfigurable systems. *International Journal of Critical Computer-Based Systems*, 1(1):86–103, 2010.

[90] P. Koopman. Embedded system security. *Computer*, 37(7):95–97, July 2004.

[91] Farinaz Koushanfar, Gang Qu, and Miodrag Potkonjak. Intellectual property metering. In *Proceedings of the 4th International Workshop on Information Hiding*, IHW '01, pages 81–95, London, UK, UK, 2001. Springer-Verlag.

[92] J. Lach, W.H. Mangione-Smith, and M. Potkonjak. Fingerprinting techniques for field-programmable gate array intellectual property protection. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(10):1253 –1261, October 2001.

[93] John Lach, William Mangione-Smith, and Miodrag Potkonjak. Fingerprinting digital circuits on programmable hardware. In *Information Hiding*, volume 1525 of *Lecture Notes in Computer Science*, pages 16–31. Springer Berlin / Heidelberg, 1998.

[94] Chris Lane. Systems software integrity assurance. In *Proceedings of the ACM SIGAda annual international conference onSIGAda*, SIGAda '10, pages 11–12, New York, NY, USA, 2010. ACM.

[95] Neal Leavitt. Researchers fight to keep implanted medical devices safe from hackers. *Computer*, 43(8):11–14, August 2010.

[96] E.A. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363 – 369, May 2008.

[97] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*. 2011. http://LeeSeshia.org.

[98] L.W. Lerner, M.M. Farag, and C.D. Patterson. Run-time prediction and preemption of configuration attacks on embedded process controllers. In *Security of Internet of Things (SecurIT), 2012 First International Conference on*, August 2012.

[99] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293 – 303, 2009.

[100] Ying-Chang Liang, Hsiao-Hwa Chen, J. Mitola, P. Mahonen, R. Kohno, J.H. Reed, and L. Milstein. Guest editorial - cognitive radio: Theory and application. *Selected Areas in Communications, IEEE Journal on*, 26(1):1–4, January 2008.

[101] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4:2–16, 2005.

[102] Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12(3):19:1–19:41, January 2009.

[103] Jay Ligatti and Srikar Reddy. A theory of runtime enforcement, with results. In *Computer Security ESORICS 2010*, volume 6345 of *Lecture Notes in Computer Science*, pages 87–100. 2010.

[104] U. Lindqvist and E. Jonsson. A map of security risks associated with using COTS. *Computer*, 31(6):60 –66, June 1998.

[105] Bev Littlewood and Lorenzo Strigini. Redundancy and diversity in security. In *Computer Security ESORICS 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 423–438. Springer Berlin / Heidelberg.

[106] J. Lotze, S.A. Fahmy, J. Noguera, and L.E. Doyle. A model-based approach to cognitive radio design. *Selected Areas in Communications, IEEE Journal on*, 29(2):455–468, February 2011.

[107] Jorg Lotze, Suhaib A. Fahmy, Juanjo Noguera, Linda Doyle, and Robert Esser. An FPGA-based cognitive radio framework. In *Signals and Systems Conference, 208. (ISSC 2008). IET Irish*, pages 138–143, June 2008.

[108] E. Love, Yier Jin, and Y. Makris. Enhancing security via provably trustworthy hardware intellectual property. In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pages 12 –17, June 2011.

[109] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford. Invited paper: Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs. In *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1–6, August 2006.

[110] P. Manadhata and J.M. Wing. An attack surface metric. Technical report, DTIC Document, 2005.

[111] Douglas Maughan. The need for a national cybersecurity research and development agenda. *Commun. ACM*, 53:29–31, February 2010.

[112] R.P. McEvoy, F.M. Crowe, C.C. Murphy, and W.P. Marnane. Optimisation of the SHA-2 family of hash functions on FPGAs. In *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, March 2006.

[113] Mark McLean and Jason Moore. Securing FPGAs for red/black systems: FPGA-based single chip cryptographic solution. *Military Embedded Systems Mag.*, 2007.

[114] K. L. McMillan. A methodology for hardware verification using compositional model checking. *Sci. Comput. Program.*, 37(1-3):279–309, May 2000.

[115] Kenneth Lauchlin McMillan. *Symbolic model checking: an approach to the state explosion problem*. PhD thesis, Pittsburgh, PA, USA, 1992. UMI Order No. GAX92-24209.

[116] W.C. Messner and D.M. Tilbury. *Control tutorials for MATLAB and Simulink: User's Guide*. Addison-Wesley, 1998.

[117] Prashant Mhaskar, Charles McFall, Adiwinata Gani, Panagiotis D. Christofides, and James F. Davis. Isolation and handling of actuator faults in nonlinear systems. *Automatica*, 44(1):53 – 62, 2008.

[118] J. Mitola. Cognitive radio policy languages. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–4, June 2009.

[119] Ira S. Moskowitz, Richard E. Newman, and Allen R. Crepeau, Daniel P. andMiller. Covert channels and anonymizing networks. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, WPES'03, pages 79–88. ACM, 2003.

[120] Seetharam Narasimhan and Swarup Bhunia. *Introduction to Hardware Security and Trust*, chapter Hardware Trojan Detection, pages 339–364. Springer New York, 2012.

[121] George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '97, pages 106–119, 1997.

[122] C. Neuman. Challenges in security for cyber-physical systems. In *DHS: S&T Workshop on Future Directions in Cyber-physical Systems Security*. Citeseer, 2009.

[123] R. Nikhil. Bluespec System Verilog: efficient, correct RTL from high level specifications. In *Formal Methods and Models for Co-Design, 2004. MEMOCODE '04. Proceedings. Second ACM and IEEE International Conference on*, pages 69 – 70, June 2004.

[124] Rishiyur S. Nikhil and Arvind. What is bluespec? *SIGDA Newsl.*, 39(1):1–1, January 2009.

[125] P. Noel, F. Zarkeshvari, and T. Kwasniewski. Recent advances in high-speed serial I/O trends, standards and techniques. In *Electrical and Computer Engineering, 2005. Canadian Conference on*, pages 1292–1295, May 2005.

[126] Kelly O'Connell. INTERNET LAW-CIA Report: cyber extortionists attacked foreign power grid, disrupting delivery. *Internet Business Law Services*, January 2008.

[127] National Institute of Standards and Technology. FIPS 197: Advanced Encryption Standard (AES). *Federal Information Processing Standards (FIPS) Publication*, 2001.

[128] National Institute of Standards and Technology. FIPS 180-3: Secure Hash Standard (SHS). *Federal Information Processing Standards (FIPS) Publication*, 2008.

[129] A.L. Oliveira. Techniques for the creation of digital watermarks in sequential circuit designs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(9):1101–1117, September 2001.

[130] S. Papa, W. Casper, and S. Nair. Placement of trust anchors in embedded computer systems. In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pages 111 –116, June 2011.

[131] D.L. Parnas. Predicate logic for software engineering. *Software Engineering, IEEE Transactions on*, 19(9):856–862, September 1993.

[132] C. Patterson. High performance DES encryption in Virtex FPGAs using JBits. In *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, pages 113 –121, 2000.

[133] Marcus Peinado, Yuqun Chen, Paul England, and John Manferdelli. NGSCB: A trusted open system. In *Information Security and Privacy*, volume 3108 of *Lecture Notes in Computer Science*, pages 86–97. 2004.

[134] F. Perich. Policy-based network management for next generation spectrum access control. In *New Frontiers in Dynamic Spectrum Access Networks, 2007. DySPAN 2007. 2nd IEEE International Symposium on*, pages 496 –506, april 2007.

[135] N. Potlapally. Hardware security in practice: Challenges and opportunities. In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE InternationalSymposium on*, pages 93 –98, june 2011.

[136] P. Quinn-Judge. Cracks in the system. *TIME Magazine (9th Jan 2002)*, 2002.

[137] R. Rajkumar, Insup Lee, Lui Sha, and J. Stankovic. Cyber-physical systems: The next computing revolution. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 731 –736, june 2010.

[138] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems*, pages 461–491, August 2004.

[139] D. Raychaudhuri, N.B. Mandayam, J.B. Evans, B.J. Ewy, S. Seshan, and P. Steenkiste. Cognet: an architectural foundation for experimental cognitive radio networks within the future internet. In *Proceedings of first ACM/IEEE international workshop on Mobility in the evolving internet architecture*, pages 11–16. ACM, 2006.

[140] T. Reece and W.H. Robinson. Hardware Trojans: The defense and attack of integrated circuits. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 293–296, October 2011.

[141] J.H. Reed. *Software radio: a modern approach to radio engineering*. Prentice Hall Professional, 2002.

[142] T. Reed. *At the abyss: an insider's history of the Cold War*. Presidio Press, 2005.

[143] F. Rouissi and G. Hoblos. Multi-criteria based approach for fault tolerant actuator selection. In *Control and Fault-Tolerant Systems (SysTol), 2010 Conference on*, pages 359 –364, October 2010.

[144] Ahmad-Reza Sadeghi, Marcel Selhorst, Christian Stüble, Christian Wachsmann, and Marcel Winandy. TCG inside?: a note on TPM specification compliance. In *Proceedings of the first ACM workshop on Scalable trusted computing*, STC '06, pages 47–56, New York, NY, USA, 2006. ACM.

[145] Ihab Samy, Ian Postlethwaite, and Da-Wei Gu. Survey and application of sensor fault detection and isolation schemes. *Control Engineering Practice*, 19(7):658 – 674, 2011.

[146] T. Schlipf, T. Buechner, R. Fritz, M. Helms, and J. Koehl. Formal verification made easy. *IBM J. Res. Dev.*, 41(4-5):567–576, 1997.

[147] Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, February 2000.

[148] Seth Schoen. Trusted computing: Promise and risk. *Electronic Frontier Foundation*, 16:26, October 2003.

[149] Carl Seger. An introduction to formal hardware verification. Technical report, Vancouver, BC, Canada, Canada, 1992.

[150] Lui Sha. Using simplicity to control complexity. *Software, IEEE*, 18(4):20–28, Jul–Aug 2001.

[151] Lui Sha, S. Gopalakrishnan, Xue Liu, and Qixin Wang. Cyber-physical systems: A new frontier. In *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC'08. IEEE International Conference on*, pages 1 –9, June 2008.

[152] Abhishek B. Sharma, Leana Golubchik, and Ramesh Govindan. Sensor faults: Detection methods and prevalence in real-world datasets. *ACM Trans. Sen. Netw.*, 6(3):23:1–23:39, June 2010.

[153] F.T. Sheldon and C. Vishik. Moving toward trustworthy systems: R&D essentials. *Computer*, 43(9):31–40, September 2010.

[154] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. A survey of cyber-physical systems. In *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, pages 1 –6, nov. 2011.

[155] R. Simha, B. Narahari, J. Zambreno, and A. Choudhary. Secure execution with components from untrusted foundries. 2006.

[156] Eric Simpson and Patrick Schaumont. Offline hardware/software authentication for reconfigurable platforms. In *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *LNCS*, pages 311–323. Springer, 2006.

[157] Jill Slay and Michael Miller. Lessons learned from the Maroochy water breach. In Eric Goetz and Sujeet Shenoi, editors, *Critical Infrastructure Protection*, volume 253 of *IFIP International Federation for Information Processing*, pages 73–82. Springer Boston, 2007.

[158] Dawn Xiaodong Song. Athena: a new efficient automatic checker for security protocol analysis. In *Computer Security Foundations Workshop, 1999. Proceedings of the 12th IEEE*, pages 192–202, 1999.

[159] W. Stallings. *Network Security Essentials: Applications and Standards*. Prentice Hall Press, Upper Saddle River, NJ, USA, 4th edition, 2010.

[160] C. Stevenson, G. Chouinard, Zhongding Lei, Wendong Hu, S. Shellhammer, and W. Caldwell. IEEE 802.22: The first cognitive radio wireless regional area network standard. *Communications Magazine, IEEE*, 47(1):130 –138, January 2009.

[161] Ji Sun, Ray Bittner, and Ken Eguro. FPGA side-channel receivers. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, pages 267–276, New York, NY, USA, 2011. ACM.

[162] Wanzhong Sun, Hongpeng Guo, Huilei He, and Zibin Dai. Design and optimized implementation of the SHA-2(256, 384, 512) hash algorithms. In *ASIC, 2007. ASICON '07. 7th International Conference on*, pages 858–861, October 2007.

[163] L.M. Surhone, M.T. Tennoe, and S.F. Henssonow. *Optimal Projection Equations*. VDM Verlag Dr. Mueller AG & Co. Kg, 2010.

[164] S. Sutherland, S. Davidmann, and P. Flake. *SystemVerilog for Design: A Guide to Using SystemVerilog for Hardware Design and Modeling*. Springer, 2006.

[165] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. In *Pacific J. Math. Volume 5, Number 2*, pages 285–309, 1955.

[166] M. Tehranipoor and F. Koushanfar. A survey of hardware Trojan taxonomy and detection. *Design Test of Computers, IEEE*, 27(1):10–25, January–February 2010.

[167] Mohammad Tehranipoor, Hassan Salmani, Xuehui Zhang, Michel Wang, Ramesh Karri, Jeyavijayan Rajendran, and Kurt Rosenfeld. Trustworthy hardware: Trojan detection and design-for-trust challenges. *Computer*, 44(7):66–74, July 2011.

[168] The Common Criteria Recognition Arrangement. Common criteria for information technology security evaluation. Technical report, Common Criiteria, September 2006.

[169] Steve Trimberger. Trusted design in FPGAs. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 5–8, New York, NY, USA, 2007. ACM.

[170] E.K. Wang, Yunming Ye, Xiaofei Xu, S.M. Yiu, L.C.K. Hui, and K.P. Chow. Security issues and challenges for cyber physical system. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 733–738, December 2010.

[171] Richard Wawrzyniak. Changing the systems landscape with low-cost FPGAs. *Xcell Journal*, Second Quarter 2005.

[172] D. Wilkins, G. Denker, M-O. Stehr, D. Elenius, R. Senanayake, and C. Talcott. Policy-based cognitive radios. *Wireless Communications, IEEE*, 14(4):41–46, August 2007.

[173] Xilinx. *Aurora 8B/10B Protocol Specification*, April 2010.

[174] S. Zander, G. Armitage, and P. Branch. A survey of covert channels and countermeasures in computer network protocols. *Communications Surveys Tutorials, IEEE*, 9(3):44–57, 2007.

[175] Karen Zee, Viktor Kuncak, Michael Taylor, and Martin Rinard. Runtime checking for program verification. In *Proceedings of the 7th international conference on Runtime verification*, pages 202–213, 2007.

[176] A.S. Zeineddini and K. Gaj. Secure partial reconfiguration of FPGAs. In *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on*, pages 155–162, December 2005.