Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Model-driven Security Engineering for FPGAs

Ing. Michael Vetter

disertační práce
k získání akademického titulu doktor
v oboru Informatika a výpočetní technika

Školitel: Doc. Ing. Vlastimil Vavřička, CSc.
Katedra: Katedra informatikya výpo četní techniky

Plzeň 2020

University of West Bohemia
Faculty of Applied Sciences

Model-driven Security Engineering for FPGAs

Ing. Michael Vetter

doctoral thesis
submitted in partial fulfillment of the requirements
for a degree of Doctor of Philosophy
in Computer Science and Engineering

Supervisor: Doc. Ing. Vlastimil Vavřička, CSc.
Faculty: Department of Computer Science and Engineering

Plzeň 2020

Prohlášení

Prohlašuji, že jsem disertační práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne ..... Podpis .....

Abstrakt

Tato práce obsahuje analýzu a adaptaci vhodných metod zabezpečení, pocházejících ze softwarové domény, do světa FPGA. Metoda formalizace bezpečnostní výzvy FPGA je prezentována jazykem FPGASECML, specifickým pro danou doménu, vhodným pro modelování hrozeb zaměřených na systém a pro formální definování bezpečnostní politiky. Vytvoření vhodných obranných mechanismů vyžaduje inteligenci o agentech ohrožení, zejména o jejich motivaci a schopnostech. Konstrukce založené na FPGA jsou, stejně jako jakýkoli jiný IT systém, vystaveny různým agentům hrozeb po celou dobu jejich životnosti, což naléhavě vyžaduje potřebu vhodné a přizpůsobitelné bezpečnostní strategie. Systematická analýza návrhu založená na konceptu STRIDE poskytuje cenné informace o hrozbách a požadovaných mechanismech protiopatření. Minimalizace povrchu útoku je jedním z nezbytných kroků k vytvoření odolného designu. Konvenční paradigmata řízení přístupu mohou modelovat pravidla řízení přístupu v návrzích FPGA. Výběr vhodného závisí na složitosti a bezpečnostních požadavcích návrhu. Formální popis architektury FPGA a bezpečnostní politiky podporuje přesnou definici aktiv a jejich možných, povolených a zakázaných interakcí. Odstraňuje nejednoznačnost z modelu hrozby a zároveň poskytuje plán implementace. Kontrola modelu může být použita k ověření, zda a do jaké míry, je návrh v souladu s uvedenou bezpečnostní politikou. Přenesení architektury do vhodného modelu a bezpečnostní politiky do ověřitelných logických vlastností může být, jak je uvedeno v této práci, automatizované, zjednodušující proces a zmírňující jeden zdroj chyb. Posílení učení může identifikovat potenciální slabiny a kroky, které může útočník podniknout, aby je využil. Některé metody zde uvedené mohou být použitelné také v jiných doménách.

Abstract

The thesis provides an analysis and adaptation of appropriate security methods from the software domain into the FPGA world and combines them with formal verification methods and machine learning techniques. The deployment of appropriate defense mechanisms requires intelligence about the threat agents, especially their motivation and capabilities. FPGA based designs are, like any other IT system, exposed to different threat agents throughout the systems lifetime, urging the need for a suitable and adaptable security strategy. The systematic analysis of the design, based on the STRIDE concept, provides valuable insight into the threats and the mandated counter mechanisms. Minimizing the attack surface is one essential step to create a resilient design. Conventional access control paradigms can model access control rules in FPGA designs and thereby restrict the exposure of sensitive elements to untrustworthy ones. A method to formalize the FPGA security challenge is presented. FPGASECML is a domain-specific language, suitable for dataflow-centric threat modeling as well as the formal definition of an enforceable security policy. The formal description of the FPGA architecture and the security policy promotes a precise definition of the assets and their possible, allowed, and prohibited interactions. This removes ambiguity from the threat model while providing a blueprint for the implementation. Model transformations allow the application of dedicated and proven tools to answer specific questions while minimizing the workload for the user. Model-checking can be applied to verify if, and to a certain degree when, a design complies with the stated security policy. Transferring the architecture into a suitable model and the security policy into verifiable logic properties can be, as demonstrated in the thesis, automated, simplifying the process and mitigating one source of error. Reinforcement learning, a machine learning method, can identify potential weaknesses and the steps an attacker may take to exploit them. The approach presented uses a Markov Decision Process in combination with a Q-learning algorithm.

Table of contents

1 Introduction

The British health system severely disrupted [1], a blackout in the Ukrainian power grid [2], the attempted theft of over a billion US Dollars from the central bank of Bangladesh [3] – all of these events and their real-life consequences (people not receiving medical care, without power and a potential state bankruptcy) - were the result of cyberattacks against vulnerable it-systems. Our modern world depends on computers in various forms to perform a growing number of highly sensitive services. They control our hospitals and banks as well as industrial plants, cars, and airplanes. The interconnection between these devices becomes denser every year as, for example, electric generators and consumers become connected via the smart grid [4]. As more and more assets are controlled by computers and as (remote) access to these assets becomes easier via communication networks, the security of these devices [5] becomes crucial to the prosperity and stability of our society. Standard, hardwired integrated circuits perform most of these tasks. However, some rely, at least in parts, on Field Programmable Gate Arrays (FPGA) – commercial off-the-shelve chips that provide the capability to perform massively parallel computation while allowing the fine-grained reconfiguration of their logic circuits. This redefinition can take place as part of their ordinary operation (dynamic reconfiguration) or through maintenance updates of their configuration. Software security for standard server and client software has advanced in recent years, but other domains still lack awareness for security problems or sophisticated methods to engineer secure systems. FPGA based systems are one of these domains, and this thesis aspires to advance the security engineering efforts by adopting proven methods from the software domain while leveraging the unique attributes of modern FPGAs to improve their resilience. These insights are translated into a formal threat (meta-) model. This formalized description is further translated into other models, more suitable to identify potential vulnerabilities and adequate solutions while minimizing the workload of their users.

1.1 On the nature of IT-Security

A secure system can be defined as one that provides only the required functionality, whose functions operate as defined (in scope, the sequence of operation, and their timing). Only legitimate actors have access to functionality and data (an actor can be either a human or another technical system). Legitimate actors are those who require access to these assets in order to perform their designated and legitimate assignments. The different stakeholders may have different opinions about, e.g., what a necessary operation is or the legitimacy of specific actors. Therefore, a system can be considered secure by one stakeholder, while others consider it insecure. An attacker intends to gain more insight into and control over the system, forcing it to expose more functionality or data as specified, to perform operations that are either not defined appropriately or to use functions in ways unintended or unauthorized by the manufacturer, operator or legitimate user. The attacker achieves this through the abuse of existing functionality, by adding, removing, or altering functions. Misleading, misdirecting, or forcing legitimate actors (e.g., through social engineering [6] or blackmail) is also a common way to work around technical restrictions. A secure system must provide a sufficient level of resilience against these threats as well as the capability to recover from a successful attack.

Two forces determine the resilience of the system against attacks. The pressure exerted against it by the attacker and its capability to withstand this pressure. The attacker's time, knowledge, financial, and technical resources, as well as their motivation, determines the pressure against the system. Their access to the target and the connectivity between them must be considered as well. The resilience of the system is the result of the architect's skills, the quality of the implementation, and the determination of its stakeholders (e.g., developer, administrator, owner) to defend it against threats. These stakeholders change throughout a

systems lifetime as the parts of the system are designed, developed, integrated, manufactured, in operation, and finally decommissioned. The current set of stakeholders may not be able to mitigate every security problem. An ordinary operator, for example, may not be able to protect a system against an attack without a vendor-provided patch. The diverging interests between stakeholders must be taken into account as well as, for example, the owner of a device might want to exhaust its functionality fully. At the same time, the manufacturer has a strong motivation to restrict the user's access to protect, for example, his trade secrets or the intellectual property of a third party. The architecture and implementation of the systems, and therefore its capability to resist attacks, is more static and the removal of security bugs often expensive and potentially error-prone in complex systems (Adobes Acrobat and Oracles Java provide an insightful case study for this phenomenon in the personal computer domain [7]). The outside pressure, on the other hand, can change dramatically in a short time. Modern IT-systems have to be designed with the best knowledge about present attacks and potential future threats to resist this dynamic threat landscape. They must further expose as little functionality as possible to a potential attacker; they have to be built with appropriate security margins to withstand attacks before flaws can be fixed and finally must inhibit the capability to adapt to a changing threat landscape throughout their lifetime, including the recovery from a successful attack.

The failed copy protection scheme of the PlayStation 3 [8] is an example of a rapid change in the threat landscape. For almost four years, this copyright protection mechanism remained unbroken, while those of competitors like Microsoft (Xbox 360) and Nintendo (Wii) where broken within months. This security streak, however, was not owed to the strength of the defense mechanism but that users with the required skillset lacked the motivation to attack it. The threat environment changed when Sony decided to remove the "Other OS "option, which allowed home developers to run their self-developed software under a customized version of Linux (the feature was deemed a security risk by Sony.) Once users with the required skillset had a motivation to attack the system, the pressure against the PlayStation 3 security system rose, and it was broken within twelve months. Similar lessons can be learned from the Stuxnet virus attacking SCADA (Supervisory Control and Data Acquisition) systems for industrial environments [9]. SCADA security was not a priority of the stakeholders; neither manufacturers, industry users, or politicians regarded it as a pressing issue before the discovery of Stuxnet. Therefore, SCADA systems provided little resilience against targeted attacks. The zero-day exploits Stuxnet used against Microsoft Windows were regarded as highly sophisticated and costly to develop - while the actual targets, Siemens S7 PLC (Programmable Logic Controller), presumably used in an Iranian nuclear facility, had a significantly lower resilience [10]. Both the PlayStation 3 and the S7 security breach came not wholly unexpected. Previous (more or less successful) attacks targeted these systems (e.g. [11]) and the exploited vulnerabilities (improper application of cryptographic algorithms in the PlayStation case, bad design, and coding practices for the S7) were avoidable through a more advanced security development process.

The dynamic and often fuzzy nature of IT-Security makes the creation of a one hundred percent secure system a noble but unrealistic goal in the real world. Maximizing the resilience against threats and the possibility to recover from many plausible attacks is a more reasonable approach. The improvement of Microsoft software in the last ten years, mainly driven by its Security Development Process [12] (SDL) and the advancement made in securing web browsers through improved architectures [13], provide a reason for optimism. Knowledge from these domains can provide a better insight into the mechanisms of attack and defense and inspire security research in other domains as well. Analyzing why these tools and processes improved the security in their respective domain and transferring them to the domain of FPGA based designs is, therefore, a legitimate approach.

1.2 FPGA Security – motivation, current state and challenges

This section provides an overview of the motivation and challenges in advancing the security of FPGA based systems. The focus of this thesis lies on SRAM (Static Random-Access Memory) based FPGAs, but this section includes a short comparison between SRAM and flash-based FPGAs under security aspects as well. Parts of this section are based on research previously published in [14] and [15].

1.2.1 The motivation to advance FPGA based designs security

This section examines why the security risks of an unsecured FPGA design are high and how improvements in the FPGA domain may lead to improvements in other areas.

FPGAs behave mostly like hardwired integrated circuits. Operating on the low level of the design, they control any functionality above them, including operating systems and applications. They can, unlike ordinary ICs, change their behavior to either adapt to new requirements or to subvert the security of the system. Enforcing the development and usage of secure software is futile if the hardware layer is insecure. When sensitive applications rely on the computing power and flexibility of an FPGA securing this component is therefore essential to the security of the overall system Among these, most sensitive, applications are Firewalls, and Intrusion Detection Systems (IDS) [16], Cryptographic (Co-)Processors and Software Defined Radio [17] (SDR).

Exploiting the FPGA's high flexibility and adaptability opens the opportunity for unique security solutions. Designs with an upgrade mechanism for the FPGA configuration can close security vulnerabilities as simple (or problematic) as the installation of the new firmware. Weak algorithms, e.g., cryptography, can be replaced by more reliable algorithms if the excess capacity of the FPGA is large enough. The system architect can also leverage the dynamic runtime reconfiguration of SRAM based FPGAs, as discussed later in this thesis, to limit the potential vulnerabilities exposed to an attacker

Improving the security of FPGA based design may provide further insights into the nature of resilience systems and result in tools and methods suitable to secure other domains. Future systems (either in the desktop server or embedded domain) may rely on CPUs (Central Processing Unit) with hundreds of processor cores connected through a flexible network-on-a-chip architecture. Knowledge gained through the advancement of FPGA security may ensure the secure and efficient utilization of these cores as well. Finally, the flexibility and interconnectivity of FPGA make them an interesting case study for the security of embedded systems in general.

The Trusted Computing Base in general and for FPGAs

The most common definition of the Trusted Computing Base is

"as the set of components (hardware, software, human, . . .) whose correct functioning is sufficient to ensure that the security policy is enforced, or, more vividly, whose failure could cause a breach of the security policy."

([18], 243)

That an element is part of the Trusted Computing Base does not mean that this element functions as defined or expected. Those elements that fail to meet their obligations are still entrusted with sensitive tasks but cannot be considered trustworthy. A prominent example of a crucial TCB component subverting the security in the PC world is the alleged theft of NSA (National Security Agency of the USA) secrets from a private computer through the antivirus software [19]. Both the NSA employee and his antivirus software were part of a trusted computing base. One failed to fulfill his obligations as he placed sensitive material on an unclassified system and the other component failed as it did not only scan for malicious codes but also

for us US federal government classification markers [20] embedded in the document (a third breach happened when this classified information about the leak was leaked to the press.) This course of events is not the only possible [21], but this way or another weaponized code found its way into a (presumably state-sponsored) hacker group, the Shadow Brokers and released into the world (presumably to humiliate their counterparts at the NSA.) This information was then used by another entity to launch one of the most devastating cyber attacks to this date – WannaCry. This notorious encryption Trojan caused damage of up to 4 billion US dollars [22] while making only 140,000 US dollars [23] in ransom. The exploit was also part of the NotPetya [24] worm that roiled the world shortly after WannaCry, probably the most devastating cyberattack to date.

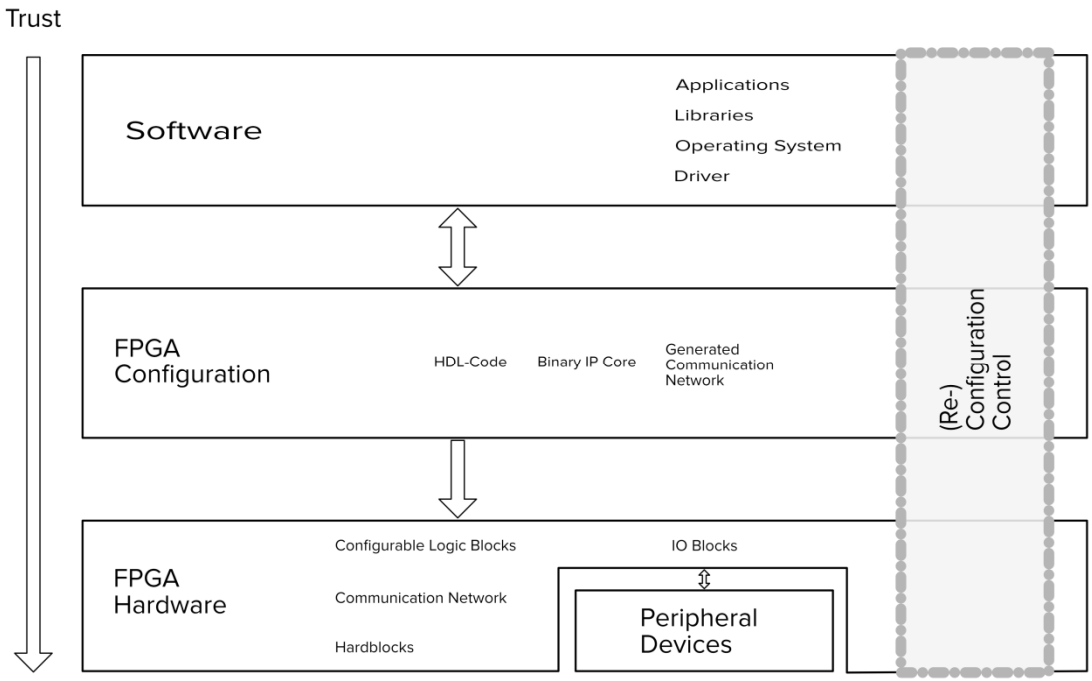


Figure 1 The FPGAs trusted computing base split into multiple layers

Figure 1 separates the Trusted Computing Base (TCB) of an FPGA based design in three vertical blocks. The basis of the TCB is the FPGA hardware itself, vendor-provided and configurable, but mostly a black box. The configuration on top of it defines its behavior if the FPGA performs its role as expected. The software layer has to trust both the FPGA configuration and the FPGA hardware as it has very few mitigation options against any malicious behavior of the FPGA itself. The Spectre security bug in Intel CPUs has demonstrated the severe risk and limited mitigation options of hardware-based flaws [25, 26].Each of these layers can be further separated into multiple components with different capabilities, different levels of trustworthiness, and often sourced through a diverse supply chain. The (Re-) configuration control is a particular component that runs vertically through these layers. Where the FPGA has only one, unchangeable, or static configuration, this unit can be rather simple, but its complexity grows when multiple, partial changes to the FPGA configuration occur over the system's runtime. Depending on the systems architecture, changes to the FPGA configuration can be triggered by a hardware component, part of the FPGA configuration, or the software.

1.2.2 Challenges in securing FPGA based designs

Several technical and economic aspects of FPGA characteristics pose a significant challenge towards the creation of efficient security solutions. One key factor is the limited number of FPGAs vendors in the market, with Xilinx and Intel (former Altera) holding almost 90 percent of the around 4 Billion Dollar market(2016) [27]. The limited number of vendors results in limited competition and lower incentives for innovative designs. New competitors entering the market require a significant amount of capital for design and production for their chips, as well as for the development of a toolchain to utilize them. Currently, no industrial-grade open-source toolchain exists, but only partial solutions like [28] or work in progress like [29]. The lack of an open development platform for all FPGAs complicates the development of security extensions for research purposes and industrial applications. The proprietary programming formats used by the vendors provide an additional challenge to the development of security tools and the auditing of existing designs for security flaws. Security solutions, suitable for industrial designs, therefore have to be built on top of or supplemental to existing toolchains. The growing complexity of FPGA designs incorporating third-party intellectual property (IP) and utilizing design paradigms like network-on-a-chip expose more functionality to potentially malicious actors and therefore require a higher investment in their security to achieve the same level of resilience. Finally, the FPGAs structure itself creates some challenges towards more secure designs. The heterogonous structure of FPGAs (two-dimensional programmable logic blocks, embedded memory elements, input-output components, flexible interconnections, and special purpose parts), the complex communication network between them and the parallel computation performed by them has to be considered, the exposure of sensitive components limited and the remaining threats mitigated.

Comparison of Flash and SRAM based FPGAs under security aspects

The main advantage of flash memory-based FPGAs over SRAM based FPGAs is that the configuration, once programmed, can remain inside the FPGA. The FPGA configuration is a valuable target for IP-theft or tampering, and having the configuration in one place (the nonvolatile storage in SRAM based FPGAs) instead of distributed within the FPGA simplifies this attack. There are, however, other ways an attacker might gain access to this valuable information, as explained in greater detail later. These common threats to the IP include theft by an employee, attacks against the code repository, and the hacking of the configuration deployment and upgrade mechanisms. The theft of the binary configuration enables further attacks. It is reasonable to assume that Flash-based FPGAs are as vulnerable to attacks against their residing configuration as SRAM based FPGAs as they differ only in the storage of their configuration. [30] presents a successful attack against a flash-based ProASIC3. The researchers were able to activate a backdoor that was placed in the FPGA itself through the application of an improved power analysis attack. The authors conclude that *"This way an attacker can extract all the configuration data from the chip, reprogram crypto and access keys, modify low-level silicon features, access unencrypted configuration bitstream or permanently damage the device. Clearly, this means the device is wide open to intellectual property (IP) theft, fraud, re-programming as well as reverse engineering of the design which allows the introduction of a new backdoor or Trojan. "*

It is reasonable to assume that the usage of flash memory-based FPGAs provides some additional protection against IP theft and manipulation while providing no advantage against other attack classes, like the exploit of design bugs or the integration of malicious code.

1.2.3 FPGA security-related work in academia and industry

This section provides a compact overview of security solutions in both the academic world and the industry to illustrate the current state of the art mechanisms and the perceived threats:

Applications	Academia	Industry
Cryptographic Algorithms	e.g. [31]	Plenty
Bitstream Security		[32],[33]
Bitstream Encryption		
Bitstream Authentication	[34]	[35]
Secure Boot	[36]	[37, 38]
IP Protection		
Cloning and IP theft prevention	[39–41]	[42–44]
Watermarking	[45]	
Secure IP-Exchange		[46, 47]
Secure Design		
Spatial Isolation of Elements	[48]	[49, 50]
Development Tools in General	[51, 52]	[53]
Secure Operation		
Secure Upgrade and Monitoring	[54]	[55]
Hardware-based information flow control		
Gate Level Information Flow Tracking	[56, 57]	

In conclusion, partial solutions for many security problems exist in both the academic and the commercial realm. The primary focus lies on the protection of intellectual property against theft. One of the standards to protect the IP is IEEE P1735 or, as the authors of [58] called it "*Standardizing Bad Cryptographic Practice.* "The authors found several security flaws and conclude that their finding "*…suggests that the standard requires a significant overhaul, and that IP-authors using P1735 encryption should consider themselves at risk.* "The number and quality of security flaws found in P1735 raise further questions about the reliability of standards and tools in general and in the niche market of FPGA development in particular.

1.3 Related work in adjacent domains

The approach described in this work operates on the intersection between IT security, model-driven development (MDD) model checking, and machine learning. Parts of this section were previously published in [59].

1.3.1 Model-driven development and security

Model-driven development [60] is a common technique to master the complexity of modern systems. General-purpose languages like SysML [61] (Systems Modeling Language) and AADL [62] (Architecture Analysis and Design Language) flatten the learning curve, advance best practices, and prevent vendor lock-in. However, to the author's knowledge, no standardized support for security analysis exists in either of these languages. Proposals like [63, 63] for AADL have been made but not widely adopted. Specialists often perform Threat Modeling [64] informally using tools as simple as paper, whiteboards, or general-purpose diagram software. Dedicated threat modeling tools include Microsoft Threat Modeling Tool [65], and OWASP (Open Web Application Security Project) Threat Dragon [66] provides a suitable structure and user interface but little automation.

1.3.2 Formal verification methods

Formal methods [67] are one crucial set of tools to ensure the quality of hard and software [67]. Model-checking [68] is a standard method to verify whether a finite state model of a system satisfies a set of formally stated properties. In the security-domain, model-checking can verify security protocols [69]. A more recent development is the application of mathematical proof assistants [70] to verify the security properties of sophisticated software [71]. Symbolic execution frameworks like Angr [72] and Triton [73] allow a more sophisticated security analysis of binary code [74] than conventional fuzzing [75] and taint analysis (e.g. [76]) tools. Many of these tools are, however, cumbersome to use, and their application is surprisingly limited. While user experience could be improved relatively easy other constraints (like NP-completeness of the underlying problem or the halting problem [77]) are insurmountable.

1.3.3 Machine learning

Machine learning (ML) has seen a boom in recent years, mainly powered by a trifecta of big data, parallel data processing power provided by GPUs (Graphics Processing Unit), and new ML architectures that utilize both. [78] discusses the application of machine learning for malware detection from a practitioner's perspective. [79] presents both a comprehensive review of the scientific literature and many research papers with applications ranging from the detection of malware to automatically generated penetration test plans. Machine Learning has also been used to generate malicious inputs with complexity beyond the conventional fuzzing [75] methods. [80] presents a Generative Adversarial Networks (GAN) [81, 82] based attack against fingerprint scanner. Researchers used a similar attack [83] against the computer vision system of an electric vehicle. Machine learning can also deceive a potential attacker [84]. In [83], the authors use Markov Decision Processes (MDP) to craft stealthy attack sequences against a cyber-physical system. MDPs can also detect attacks [85] or serve other defensive purposes. The Cyber Grand Challenge 2016 [86] demonstrated that modern computer programs are, in principle, able to detect, exploit, and patch previously unknown vulnerabilities in other cyber systems without human intervention. Analysts assume that military contractors and intelligence agencies do most research in this field, and therefore secret [87] Advances in other, more visible, domains like games can provide us with valuable insights about the capabilities and limits of current machine learning systems. Millstones for this progress include Chess with Deep Blue (Chess) [88], Jeopardy with Watson [89], Pacman [89] and diverse ATARI 2600 arcade games [90, 91], the board game Go [92], as well as the real-time strategy game Starcraft II [93] and Quake III [94]. Unresolved, but intensely researched problems, like autonomous driving, indicate the limits of the current technology when the problem space becomes too large.

Advances in other fields of statistics have provided us with methods to determinate casual effects [95, 96]in the absence of randomized experiments and the exploration of counterfactual scenarios. Progress in probabilistic programming [93, 97] has eased the usage of Bayesian statistics [98] to, e.g., incorporate knowledge about a system as informed priors.

The advancement of machine learning in recent years can and has improved both attacks and defenses of cyber-physical systems. The machine learning tools available today must be tailored to the application at hand and used with in-depth knowledge of their strength and limits [99].

1.3.4 Security engineering in the software and FPGA domain

Investments in software security by industry and the public sector have improved the resilience of client- and server-based software against attackers [100]. An analysis of the differences and similarities between conventional, software-centric, and FPGA based systems in general and under security aspects is presented here to provide further insight.

The FPGA market is, in comparison to the software domain, a niche market with a limited number of vendors, developers, tools, and buyers. The number of applications for FPGAs is still small in comparison to those in the software domain. From a technical standpoint, FPGAs provide the capability for massive parallel computation at a fine granularity, and sophisticated hardware description languages (HDLs) like VHDL (Very High Speed Integrated Circuit Hardware Description Language) and Verilog differ significantly from general-purpose programming languages like Java and C. There are also similarities. Exchangeable descriptions define the system behavior - program code in software, hardware description in the FPGA domain. In both domains, a single design can consist of a number of these descriptions from various sources and provide in multiple forms (e.g., binary, source code). Code written for ordinary processors might be executed on FPGAs as well, e.g., through standards like the Open Computing Language (OpenCL) [101]. The trend to computer-generated code, observable in both domains (e.g., using Matlab/Simulink [102]), improves coding efficiency, allows the creation of more intricate designs, and decreases understanding of the internals of the system. The human-written code can be made accessible for audit (e.g., peer review), and the limited processing power of the human brain limits its complexity. Code generators cannot annotate the crucial components of the code with helpful comments. Even rules can result in complex, generated code that is hard to assess.

Both domains share security-relevant attributes as well. The risk of flaws in general and a security-related flaw in particular increases with the complexity of the system. In both domains, encryption and obfuscation are used to protect intellectual property (the executable or firmware in the software domain and the configuration in the FPGA domain). The complexity of the source code supply chain poses a risk in both worlds as malicious or compromised participants in this supply chain may tamper the code on its way to the customer [103]. The large and often distributed teams used to create complex systems increase the risk of an insider attack. The variety of stakeholders (e.g., intellectual property owner, device owner, administrator, and manufacturer) during the systems lifecycle increases the risk of an attack. These stakeholders will often have competing interests, and this may result in tradeoffs (usability vs. encryption is a classic example of this problem) that decrease the security of the system.

The most apparent difference between the two domains is the lack of sophisticated security tools for FPGAs. The software developer can rely on a high number of tools for various tasks, IDApro [104] and Ghidra [105] disassemble and analyze binary code, static analyzers like Fortify[106] identify potential security flaws in the source code and a vast amount of fuzzing tools [75] enable the automatic and dynamic testing of code. Coding standards like the CERT (Computer Emergency Response Team) standards for programming languages [107] provide guidelines for [105]developers. Organizations like the Open Web Application Security Project (OWASP) [66]collect, improve, and propagate the collective knowledge about attacks, vulnerabilities, and counter-measurements in their respective fields. To the author's knowledge, no such tools, guidelines, and organizations exist to support the FPGA developer. The FPGA domain can benefit from some of these guidelines, taxonomy's, tools, and best practices. Selected entries from the Top 25 of the Common Weakness Enumeration (CWE), developed to provide a taxonomy for common

software security flaws, and are used here to illustrate the feasibility of this approach. As demonstrated in later chapters, FPGA configurations are well susceptible to those and other weaknesses already named and listed for the software domain, like (the following three quotes are from [108]):

"CWE-306: Missing Authentication for Critical Function

Description Summary

The software does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources."

This weakness is not limited to human actors (or users), any hardware component connected to the FPGA could make potentially dangerous requests, the mandatory authentication of this component (e.g., through a challenge-response protocol) mitigates this risk.

"CWE-829: Inclusion of Functionality from Untrusted Control Sphere

Description Summary

The software imports, requires, or includes executable functionality (such as a library) from a source that is outside of the intended control sphere."

The use of third-party IP cores and code generator exposes, as discussed in greater detail later, FPGA designs to the very same risk.

"CWE-807: Reliance on Untrusted Inputs in a Security Decision

Description Summary

The application uses a protection mechanism that relies on the existence or values of an input, but the input can be modified by an untrusted actor in a way that bypasses the protection mechanism."

FPGA based designs can suffer from this weakness as well, as illustrated here by an example. An FPGA implements an asymmetric cryptographic algorithm and utilizes an external hardware component to create cryptographically secure random numbers. If an attacker gains control over this random number generator and reduces the entropy of its output, the cryptographic operation inside the FPGA could fail as well and leave potentially sensitive material vulnerable.

A significant difference between these domains is the lack of a standard or quasi-standard security-aware development process. Software companies can refer to the Microsoft Security Development Lifecycle [109]or the BSIMM (Building Security In Maturity Model) benchmarking initiative [110] to kick-start and improve their security development process. Data gathered through these processes can be used to compare them and to gain further insight into the nature of secure systems[111]. To the author's knowledge, no such initiative or de facto standard exists for the FPGA domain.

It is reasonable to conclude that the tools and methods to create secure software are more advanced than those in the FPGA domain. The security engineering process of FPGA based designs can, therefore, be improved by applying methods and processes inspired by those in the software domain and adapted to not only incorporate FPGA specific characteristics fully but to leverage them for innovative security mechanisms.

1.4 Research gap and contribution of this thesis

One or more partial solutions for distinct security problems are available from either the academic or the commercial realm, but the tools and methods available and applied are less sophisticated and comprehensive than those in the software domain. This thesis focus lies on a high-level security analysis of the design, its components, and their interaction. The following contributions to advance the security engineering for FPGAs are claimed:

Contribution 1: Analysis and adaptation of appropriate security methods from the software domain into the FPGA domain

This thesis provides an analysis and comparison of the state of security engineering in the software and FPGA domain. A threat taxonomy for FPGAs, as well as system-centric threat modeling techniques, are presented. Building blocks for FPGA threat models are introduced to increase the efficiency and consistency of this process.

Contribution 2: Introduction of a domain-specific language suitable for the system-centric threat model of an FPGA architecture and the formal definition of its security policy

This thesis introduces FPGASECML - a domain-specific modeling language designed to formalize both the threat model of the FPGA architecture and the security policy. A proof of concept software to create and process these models is presented.

Contribution 3: Formal, model-based validation of an FPGA design security attributes

This thesis presents a method to validate the FPGASECML system model of a design against a security policy using formal temporal logic. A proof of concept implementation provides an automatic model to model transformation.

Contribution 4: Isolation of FPGA elements with different security levels through Partial Runtime Reconfiguration

This thesis proposes the application of Partial Runtime Reconfiguration to separate elements of different sensitivity and trustworthiness. The formal validation approach through model checking enables the creation and validation of a secure sequence of reconfigurations of the FPGA.

Contribution 5: Model-based weakness analysis through reinforcement learning

This thesis proposes the application of reinforcement learning to identify potential weaknesses and how an attacker may exploit them. A proof of concept implementation generates the required Reinforced learning model (based on Markov decision process-based) from any valid FPGASECML description.

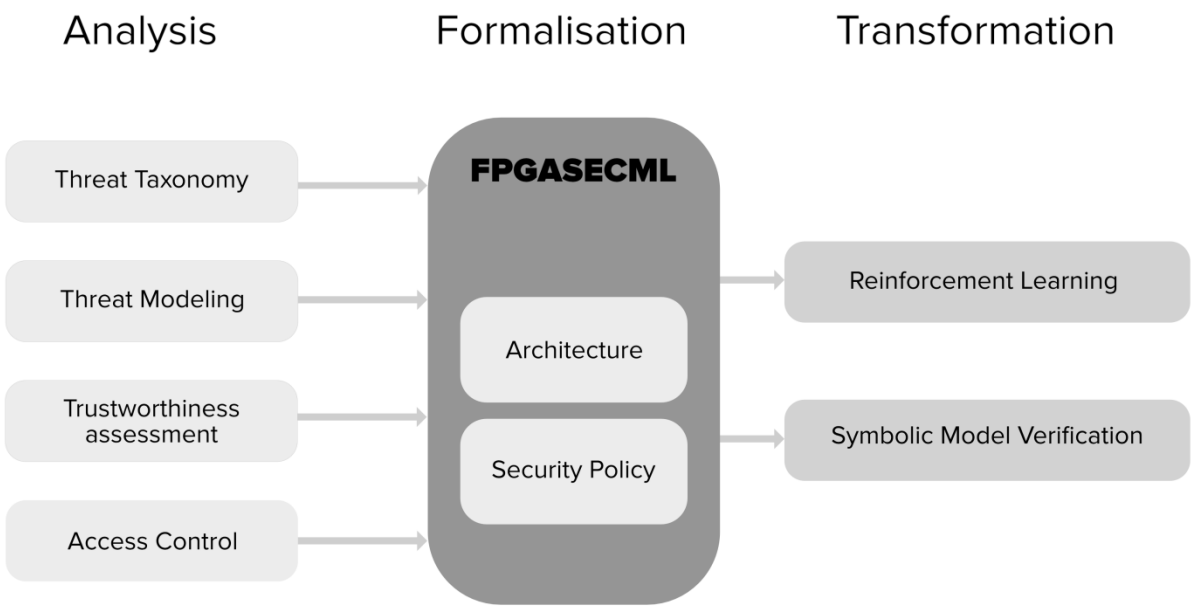


Figure 2 Structure of the thesis

1.5 Conclusion

Resilience against attacks is a crucial attribute of any IT system, including those relying on FPGAs. The existing and proposed security solutions in industry and academia mainly focus on the protection of intellectual property. The methods and processes for conventional IT systems are regarded as more advanced than those in the FPGA world, but it is possible to adapt some of these mechanisms to the FPGA domain. The following chapters (as outlined in Figure 2) present an analysis and adaptation of suitable security methods from the software domain into the FPGA domain, the formalization of the security challenge by introducing a domain-specific language, suitable to describe the designs architecture and the definition of its security policy. A workflow for the formal validation of the systems architecture model against an FPGA security policy is presented as well as a method to isolate FPGA elements with different security sensitivity levels through Partial Runtime Reconfiguration. Reinforcement learning can identify potential weaknesses in the design

2 Threat assessment and trustworthiness determination

Information about the threats a modern system will face throughout its lifetime will always be incomplete and fuzzy. There is also a considerable uncertainty of how well its multiple elements will withstand these threats while fulfilling their obligations. A structured approach to determine these two factors – threats and trustworthiness- can minimize this uncertainty and allow an efficient implementation of security mechanisms.

The first section of this chapter discusses the principal threat assessment methods used today and their suitability for the FPGA domain. It presents several threat archetypes against FPGA designs and how the threat landscape changes across the system lifecycle. The system to be created must be able to deal with these threats to be considered secure and trustworthy. The second section presents various methods and criteria to assess the trustworthiness of its components.

2.1 Assessing the threat of FPGA elements

The identification and classification of threats is the foundation for an effective defense against them. This section provides a framework for an FPGA specific threat classification, inspired by the extensive work already done in the software domain.

2.1.1 Common threat classification methods

Today two classifications for threats are commonly used: the CIA triad focusing on the security properties of a system and the Microsoft STRIDE approach analyzing the system from the attackers' point of view.

CIA Triad

The CIA classification focuses on the properties a secure system must maintain:

- **C**onfidentiality: hiding information from unauthorized actors
- **I**ntegrity: maintaining the assured and expected properties
- **A**vailability: providing the required services when requested

A system maintaining these attributes can further provide non-reputability. This fourth property assures the correct performance of an operation.

Microsoft STRIDE

Microsoft introduced STRIDE [74] as the attacker-centric alternative to the asset-centric CIA classification. STRIDE focuses on the attacker's intention; each letter of the mnemonic stands for a generic class of threats:

- **S**poofing identity: pretending to be someone or something else
- **T**ampering with: performing unauthorized changes
- **R**epudiation**:** hiding or denying that something happened
- **I**nformation disclosure: revealing sensitive information to unauthorized actors
- **D**enial of service: making a crucial element of, or the complete system, either temporary or permanently unavailable to legitimate actors
- **E**levation of privilege: gaining control over or getting access to the system or an element of it beyond the legitimate use

STRIDE does not aim to be a formal analysis method. It is often possible to attribute a single attack to more than one intention. STRIDE is, therefore, best suited to perform initial and nonformal threat analysis.

Comparison

A secure system should be able to maintain the CIA properties even under adversarial conditions. STRIDE focuses on the attacker's view and their goals and is, therefore, more suitable for threat models. Motivated by its success in the software domain, it serves as a basis for the generic threat model for FPGA based designs presented later.

2.1.2 Threat agent archetypes

A security assessment should identify those who pose a threat to the security of the system. This section discusses the classic threat agents first and presents FPGA specific threat agents second.

Traditional threat agents

Not all hackers, or more formally threat agents, are equal, and it is reasonable to separate them in different groups. The knowledge about them is often limited and fuzzy (in [18], page 367, the author bemoans the lack of an "*international standard burglar*"). The traditional hierarchy of attackers is:

- **Script Kiddies** execute prepared programs, shell scripts, and step-by-step instructions to achieve their goal. They lack any profound understanding of both the system they attack and the methods they are using. They are unable to customize their tools to a distinct target and to improve the performance of their tools.
- **Hackers/Crackers** have a deep technical understanding of the targeted system. They analyze IT-systems for weaknesses and have the competence to exploit them on their own. In the past, hackers have worked alone. Recent years have seen the rise of a more professional scene willing to collaborate to sell zero-day exploits, easy-to-use exploit packages, rentable botnets, and services like support and customization [112].
- **Organized Crime** has the financial and organizational means to perform attacks against well-protected high-value targets like banks [113] or credit card companies.
- **Foreign State/Intelligence Service** can invest significant resources in both fiscal and human resources. These capabilities allow them to attack cryptographic methods and hardware implementations beyond the reach of other parties. They can carry out cyber-attacks against well-secured facilities like the uranium enrichment facilities in Iran [114]. In this realm of "cloak and dagger "[115] operations, the real culprit is often hard to identify, even for other states [116]. Public information often remains fuzzy as, e.g., western intelligence services claim that Russian intelligence services are behind the Islamic State's hacking army [117]. Another example is North Korean alleged involvement in the attack against Sony Pictures. These assessments [118] where leaked by the NSA while hard evidence is either missing or remains classified for the foreseeable future.

This template serves well for a high-level analysis of general-purpose IT-systems. For a specific domain or a distinct application, a more detailed breakdown and analysis of the threat agents is often mandated. It is reasonable to discriminate attackers by their intention/motivation and their capabilities with regards to their budget, the available equipment, knowledge, and time:

- The **available time** can be crucial. Many attacks have a small window of opportunity, e.g., as a patch may remove the exploitable design flaw. Other attacks require extensive knowledge of a system and many rounds of trial and error before they succeed. Some attacks are only possible at a particular stage of the product lifecycle. Known attacks can become ineffective once detected and

mitigated, either by patches or through the introduction of additional security mechanisms (e.g., Firewalls).

- **Financial resources** available and the expected return on investment are crucial for attackers seeking financial gain through their evil deeds. Analyzing a complex system requires both expensive equipment and skilled engineers; a reasonable person or organization will, therefore, only attack a target worth the effort.
- The available **equipment** is also a significant factor. Some hardware attacks require expensive equipment like a focused ion beam microscope to dissect the die of a chip. Advances in hard- and software can lower the cost of entry and enable new attacks. A low budget or homebrew version may be less versatile [119] and not sophisticated enough for professional use but sufficient to carry out attacks against legacy systems. Alternative firmware can transform widely available and inexpensive devices into powerful hacking tools, like a mobile phone that becomes a GSM base station [120]. The same is valid for software tools rendering dedicated devices obsolete (as Ethereal, later Wireshark [121], did for network diagnoses).
- **Motivation** and **Incentives** can play a crucial role. Greed is often the motive for an attack but not the only one. The security system of the PlayStation 3 was, as mentioned above, not broken by criminals eager to sell pirated software but by enthusiastic hackers craving to restore the "Boot to Linux "feature removed by Sony (ironically for security reasons).
- **Knowledge** is also a crucial resource. Knowledge of the very system is often the critical component of a successful attack. An attacker also has to master the tools and mechanisms to carry out his attack.

FPGA specific threat agents

Applying the criteria from the last section to the FPGA domain, threat agents can be grouped in these, for this task more appropriate, categories:

- **Software Hackers** can detect and exploit software vulnerabilities. Their knowledge can range from low (script kiddie) to high (finding new flaws in the well-protected software system and inventing novel ways to exploit them). These threat agents do not rely on specialized hardware.
- **Hardware Tinkerer** can manipulate the peripheral hardware around the FPGA, either placing probes to gather information replacing elements or looking for side channels. Expensive equipment might be necessary for some attacks, but a USB oscilloscope and some probes could be sufficient for others. Attacking SMD devices or multilayer PCBs requires manual skills and the appropriate equipment. Physical tamper protection like glue can raise the stakes higher.
- **Digital Design Experts** with extensive knowledge about the design of digital circuits in general and FPGAs, in particular, can perform attacks against the FPGA configuration once it is retrieved. Attacks like these require extensive knowledge of not only the FPGA hardware and the tools chain, as well as access to this, often costly, software (a well-formulated search machine query could significantly reduce the cost of entry for the latter.)
- **Microsystems Engineers** capable of analyzing or manipulating the FPGA chip and storage elements holding either the FPGAs configuration or other sensitive data. They can retrieve this data from either the FPGA or its peripheral devices. Their skill requires either specialized equipment obtained at high cost or less sophisticated self-made solutions.

Software Hackers are not unusual these days, but the high equipment costs for an attack at the chip level makes the Microsystem Engineer capable of targeting the FPGA itself a much rarer opponent. The complexity of modern FPGA-chips, their gate size, and the cost of the required equipment must be taken into account as well. A reasonable assumption is that attacks against an FPGA this close to the hardware

are more expensive than attacks against legacy integrated circuits (IC) produced with a less versatile process and more straightforward design (a security advantage of FPGAs over hardwired ICs). Organizations, e.g., competitors, may combine the skills of multiple threat agents. Such organizations are the most dangerous threat if they combine several of these technical capabilities with the capability to coordinate these experts efficiently. Depending on the very system under analysis, a more detailed profile, considering the motivation and the available resources, for each of these classes may be created. In the absence of better intelligence, the generic threat agents introduced in this section should serve well for architecture-centric security analysis.

2.1.3 The FPGA threat landscape throughout the systems lifecycle

The threat landscape changes as the FPGA systems move from its development to the decommissioning phase. This section discusses multiple threats the system faces at each step of the lifecycle. This section builds upon and extends an analysis already published in [14]. The practitioner may extend the list of threats presented to fit a specific scenario and changes in the threat landscape.

Design and development

Moles planting malicious code inside the code, dissatisfied employees leaking trade secrets, and outside hacker tampering with the code repository are among the threats to be mitigated at this stage. Following the STRIDE mnemonic, these generic threats can be identified:

- **S**poofing the identity of a legitimate developer or supplier an attacker can access and alter the source code or the specification
- **S**poofing the identity of a crucial IT component, an attacker can sneak malicious information into the development process or retrieve sensitive information from it.
- **T**ampering with data: an attacker can change the systems source code or specification, either to add functionality to be exploited later or to weaken the systems resilience
- **T**ampering with the built tools: an attacker can sneak malicious code inside the product without changing the source code.
- **T**ampering with the IT infrastructure: an attacker can leak information about the product and sneak malicious code into the product.
- **R**epudiation**:** scrubbing or manipulating access logs to the code repository and other sensitive parts of the development infrastructure can hide an attack.
- **I**nformation disclosure: disclosing the source code, the specification, and information about the used IT infrastructure enables further attacks.
- **D**enial of service: an attacker could leave a legacy system in a vulnerable state by making its source code or the built system temporarily or permanently unavailable.
- **E**levation of privilege: legitimate user (developer, administrator, janitor) may escalate their access privileges to perform attacks (e.g., a janitor attacking the code repository)

The stakeholders at this stage can mitigate these threats by securing the crucial IT components (Server, Workstation, Network) and operations by technological and organizational means (as described in [122]).

Manufacturing

Complex supply chains and manufacturing processes, including the complete outsourcing of this step to contract manufacturers, create multiple potential vulnerabilities. The stakeholders must mitigate these threats by securing the manufacturing process and the supply chain, relying on security mechanisms and processes not unique to the FPGA domain and, therefore, outside the scope of this thesis.

Deployment, operation, and maintenance

Once the product has arrived at its destination, it must be installed, configured, and maintained. An attacker may exploit an opportunity provided by the system's physical location, connectivity, configuration, and or maintenance status. The stakeholders can mitigate these threats through the creation of a resilient system, shipped with a secure default configuration, and created with the ability to recover from a security breach. The methods presented in this thesis are focused on this effort.

Decommission

Decommissioned devices can provide valuable insights into the system's inner workings and may even contain sensitive data. The establishment of a secure decommissioning process mitigates these threats. Appropriate steps to decommission sensitive devices include the scrubbing of their content and their physical destruction [123]. The methods presented in this thesis support the critical steps of identifying the sensitive data and their location in the system.

2.2 Assessing the trustworthiness of a design element

The complete and formal verification of any nontrivial design, as performed on the SLE4 microkernel [124], could provide certainty whether a system satisfies the stated properties. Such formal proofs come with often prohibitive costs and are therefore of little use in most industrial designs. The dynamic and fuzzy threat landscape makes the development of security metrics [125] a challenging task, and their application can lead to a wrong sense of security (being 99% "secure "means little if an attacker finds a way to exploit the remaining 1%). However, if applied carefully and with knowledge of their limitations, a cautious expert can use security metrics to monitor and compare some aspects of a design's components. Relying on a single number to capture the security of unique designs, either complete or in parts, will not improve their security at best and provides a wrong sense of security at worst. Heuristics, deducted from experiences in software security, can be used to approximate the risk posed by the various design elements – the focus of this analysis lies on those elements that define the FPGAs behavior.

2.2.1 Trustworthiness indicators

This section discusses several quantitative and qualitative attributes of an element that can indicate its trustworthiness. Examples include the sourcing (distributors, vendors, developers) of the element, its complexity, the implementation layer, built-in security mechanisms, and which kind of testing and security analysis was performed on it. Appendix E provides a detailed discussion of these indicators.

2.2.2 Methods to formalize the security assessment

The use of checklists and decision trees supports inexperienced users, improves completeness and reproducibility of the assessment process, as well as the soundness of the assessment itself. Neither of these tools may provide the perfect answer for every case but only reasonable guidance through the most common problems.

Checklists

Checklists [126] can ensure a consistent security assessment for all elements. The trustworthiness indicator presented above, security standards, and recommendations (e.g., minimal length for an encryption key) can serve as a basis for this assessment. Personal signoff on checklists can increase the personal accountability of the assessor.

A checklist can be as simple as:

1. Is the source of the element trustworthy?
2. Was the element tested for security vulnerabilities?

3. Are there known security flaws within this element?

Decision Trees

Decision trees can guide and streamline the assessment process. They provide more flexibility than checklists, but the tree should not become too broad or deep as this decreases their usability. Decision trees can be either manually constructed or derived from previous security assessments through machine learning.

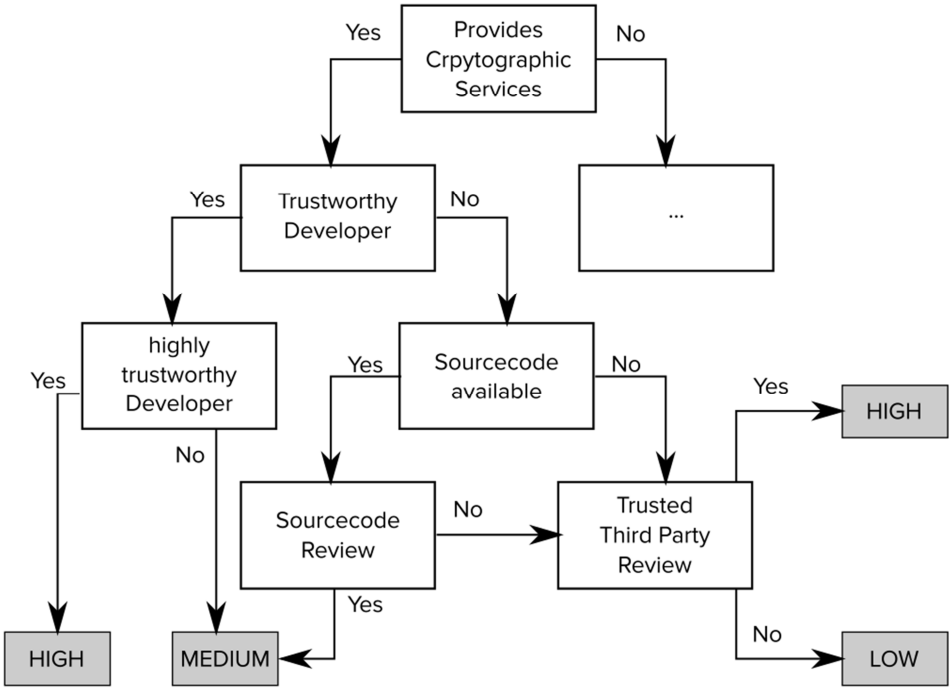


Figure 3 Fragment of a decision tree to assess the trustworthiness of an element (assuming that the external code review is more rigorous than the in-house analysis)

2.2.3 Data-driven security assessments through machine learning

Predictive maintenance [127, 128] is used in mechanical engineering to reduce repair costs and downtime while increasing the lifetime of industrial machines, wind turbines, and similar devices. This method utilizes statistical models to predict which components require maintenance based on the history of similar devices.

The trustworthiness assessment of a component is a nontrivial task that requires an extensive body of knowledge and experience. Where objective data and criteria are missing the influence of anecdotal evidence, personal bias, as well as subjective opinions, are often substantial. The lack of objective criteria makes it harder to achieve reliable, comparable, and replicable assessments. Machine (or statistical) learning methods [129] can decrease these effects and support the assessment process. Appendix E outlines such a machine learning solution. Supplementing human judgment with a statistical model provides several advantages. Statistical learning methods can identify and minimize human bias and therefore improve the consistency and reliability of the security assessment. Knowledge and experience codified in a mathematical model aides inexperienced assessors. The assessment process can be sped up by this approach, as the model can be run quickly in case of a security incident or any other change in the underlying database. Analyzing the generated (trained) model itself might provide further insights into the development process.

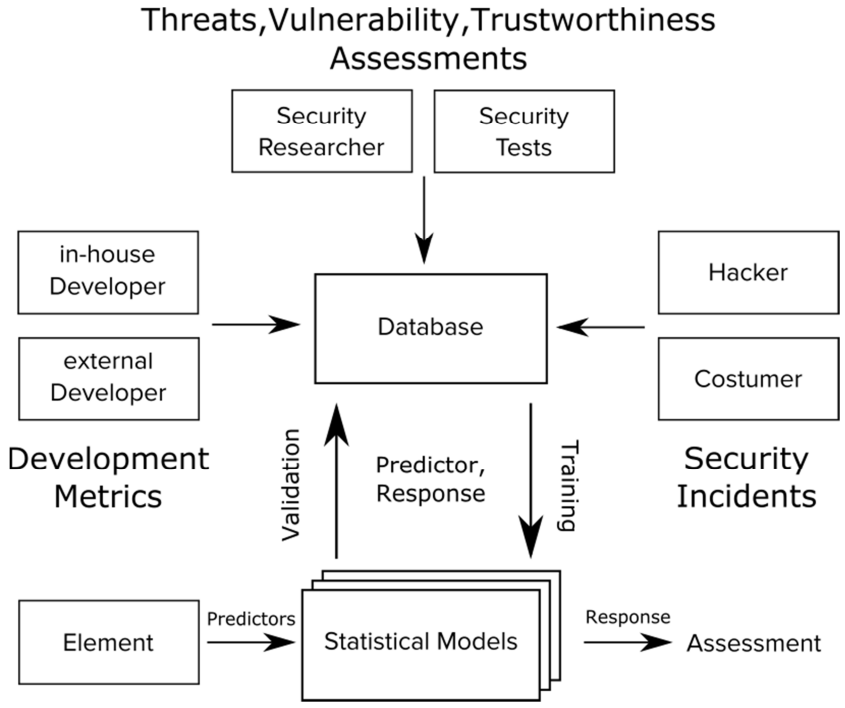


Figure 4 Components of a machine learning workflow

The primary challenge is the collection of meaningful and comparable data to create and verify the statistical models. Sampling the necessary data for machine learning is a challenge in many areas and the sensitive nature of IT security, and the niche characteristic of FPGA development is not helpful in this regard. To the author's knowledge, no open dataset (like [130]) is available, and neither the errata from manufacturers like Intel or AMD nor open-source projects provide the amount and depth necessary for such an analysis. It is, therefore, not possible to provide more than an abstract discussion of this topic here. Another challenge is the complexity of the development process, often distributed along a more or less extensive supply chain and deployment landscapes. Data sources can be grouped into three, coarse-grained classes: in-house, partners or associates, and third parties. Collecting the required data in-house should pose the fewest problems. Through persuasion, partners like suppliers and professional customers could provide at least some of the required data. Third parties like end customers may be even harder to convince. The same is true for competitors that could provide incident reports of attacks against their systems, as well as security researchers and intelligence services that may have information about existing vulnerabilities and developing threats. Legal restrictions like privacy and liabilities laws, as well as contractual obligations, are additional obstacles on the way to a suitable database. Data acquisition is, unfortunately, not the only challenge. Security incidents are often sudden and discrete events with potentially catastrophic results. This characteristic limits the use of statistical models but does not render them completely useless. Older elements may be more vulnerable, the given key length of a cryptographic algorithm provides less security over time, and a high error rate of a deployed device may either indicate an ongoing attack or a security risk due to questionable product quality. Statistical learning provides a valuable toolset to verify or reject these hypotheses as long as their limits and downsides [105], but the lack of available data makes them unsuitable for this thesis.

2.3 Conclusion

The creation of appropriate defense mechanisms requires a deep understanding of the system but also of the threat agents, especially with regards to their motivation and capabilities. The generic threat agents discussed in this chapter provide a suitable starting point for further analysis of both the design under

review and the threats against it. FPGAs based designs are, like any other IT system, exposed to different threat agents throughout their lifetime, urging the need for a suitable and adaptable security strategy for each step of the way. It is sensible to enumerate several generic threats for any FPGA based design using the STRIDE approach. The different stages of the lifecycle play a vital role in the security assessment. The design and development phase of the lifecycle lays the foundation for a secure and resilient system in every future stage.

Assessing the trustworthiness of the design elements and their supply chain, even if the result of this analysis is inherently fuzzy, is an important step to improve the security of the design. The partial automatization of the assessment process through machine learning appears promising as long as the user is fully aware of their pitfalls and restrictions. The lack of an extensive dataset for training and validation, however, blocks this avenue for research.

The next chapter is devoted to the creation of a threat model for the entire system.

3 On the threat modeling of FPGA based designs

A practical and efficient defense against attacks requires knowledge about the attacker, the assets worth defending, and the system containing them. This chapter discusses various ways of threat modeling and their applicability and limitations with regards to FPGA based designs. Parts of this chapter are based and built upon research previously published in [14].

3.1 The motivation for threat modeling

The general tendency to elaborate, interconnected systems, and the growing number [131] of attacks against IT-systems combined with the potential damage of these attacks motivate the search for better security solutions for FPGA based systems; cost efficiency dictates that this step should require as little resources as possible. Modern FPGA can contain logic circuits with the equivalent of several million gates. Standardized interfaces like AMBA [132] or Whisbone [133] simplify the integration of third-party IP cores. This incorporation of black-box components combined with new paradigms like Network-on-a-chip provides the base for more robust designs, but the complexity of these designs introduces new security challenges. Third-party elements can contain malicious code, endangering the security of the whole system. The installation of new functionality and the upgrade of existing blocks of reconfigurable logic over Wide Area Networks (e.g., the internet) extend the system defense perimeter way beyond the FPGA. The complexity of the FPGA designs makes the detection and mitigation of security-relevant bugs more challenging. Partial Runtime Reconfiguration further increases the risk of a security incident, as discussed in greater detail later.

Discarding these new opportunities is not an option either as it would reduce the capabilities of the FPGA significantly. Therefore, new approaches -balancing security, functionality, and efficiency- are necessary. These new methods must incorporate the knowledge about secure systems in general as well as the FPGA specific attributes and must empower their users to apply this knowledge to the very system to be created or reviewed. Threat modeling FPGA based designs is one crucial step to achieve this goal as this process creates and consolidates the knowledge necessary to identify and mitigate potential weaknesses throughout the design lifecycle.

3.2 Threat modeling basics

Threat agents (discussed in 2.1.2) require only one path to carry out their attack while the systems creator has to make sure that the easiest way to subvert the security of the system (the weakest link) is still too expensive for them. Locating and reinforcing this weakest link is crucial to the overall security of a system. Then the new weakest link has to be found and reinforced, and so on until the design has reached an acceptable level of (assumed) resilience. It is, therefore, necessary to analyze the threats, system weaknesses, and assets as well as the appropriate mitigation mechanism. Threat modeling is not restricted to any stage of the lifecycle but most efficient if completed before the implementation starts.

There are three approaches to Threat Modeling:

- **Asset Centric Threat Modeling** focuses on the system assets
- **Attacker Centric Threat Modeling** identifies the potential threat agents, their capabilities, and intentions (as discussed in chapter 2 and chapter 9)
- **Software Centric Threat Modeling** creates and analyses a model of the system to identify potential security vulnerabilities and if their proper mitigation. This approach is not limited to the software domain but can be used for FPGA based systems as well. It will be, therefore, referred to as **System Centric Threat Modeling** here.

All three views: assets, attackers, and the design of the system are essential to understanding the security risks and requirements of a design. System Centric Threat Modeling creates a simplified model of the design and its data flow, omits implementation details, and focuses on the threat modeling process on the interaction between the different elements. This model should contain the assets and their exposure to threat agents. It must further contain the security mechanisms used and where they are deployed to protect the assets.

This chapter determines and analyzes the required building blocks for system-centric threat modeling of FPGA based designs. The focus of this theses lies on a single FPGA configuration. Systems utilizing multiple FPGAs can be modeled by a distinct threat model for each FPGA, treating the others as peripheral devices. The model also omits details about the Hardware-Software interface.

3.3 Limits and constraints of existing threat modeling approaches

Any security analysis can only provide a snapshot of both the system under review and the threats it faces. The threat landscape is subject to change throughout the lifetime of the system, either through improved technology, incautious alterations of the system, or failing security mechanisms (e.g., a broken encryption algorithm). Therefore, an ongoing review of the threat model throughout all stages of the product lifecycle is necessary – a task simplified by using the predefined structures and building blocks introduced in the next chapter. Every model is only a simplified version of reality, but an incorrect model or one that omits essential details can be dangerous. A wrong or outdated model of the system may result in an incorrect security assessment. In [134], the authors describe a method to overcome this shortcoming for software, but to the author's knowledge, no commercial tool ever implemented this technique. The model's level of detail and the appropriate level of analysis have to be determined by the responsible stakeholder. Current threat model tools are designed for software systems [64] and lack many of the features necessary to analyze FPGA based architectures, e.g., support for Partial Runtime Reconfiguration (PRR). It is, therefore, necessary to extend and adapt them to the FPGA domain. Structuration and formalization of the model increase efficiency and enables further steps (like model-to-model translation) to improve the security of the design after the completion of the analysis and adaptation phase.

3.4 Structuring trust and threats

Systems security often breaks at the interfaces. The majority of attackers start at the periphery and moves towards central elements. It is, therefore, prudent to split the system into different layers and analyze the threats associated with each of them. This section provides an overview of the different roles an FPGA can perform, a generic set of threat layers as well as a discussion of trust boundaries.

3.4.1 The FPGAs role within the system

Analyzing the role of the FPGA within a system also provides first insights into the security requirements and threat landscape of the design. From the systems view, an FPGA can fulfill one or more of these roles:

- **Preprocessor:** The FPGA's massive parallel computing power can reduce the amount of data to be processed by a microcontroller or a DSP (Digital Signal Processor). Examples of such applications include software-defined radio [17]. Network Intrusion Detection systems use them to perform massive parallel operations on high-speed network traffic [135].
- **Coprocessor:** FPGAs are often used to complement conventional, hardwired processors. Examples for this configuration include the implementation of asymmetric cryptography like elliptic curve cryptography (ECC) in the FPGA [136], p 225.
- **Communication Hub:** FPGAs allow the parallel computation of data; they have a flexible communication infrastructure and contain hardware blocks for high-speed communication

protocols (hardwired IP). These characteristics enable them to serve as connectors between other processing components in complex and dynamic environments [16].

- **Standalone data processing:** FPGAs can also serve as standalone processing entities replacing DSPs, Microcontrollers, and other ICs.

Sensitive services must be performed as specified, especially if the FPGA is part of the Trusted Computing Base of the overall design. Where the FPGA is, for example, used to route sensitive data, the data flow must be correct, and no unauthorized actors must access sensitive information. It is reasonable to invest fewer resources (in analysis, design, implementation, and test) if the FPGAs services are not crucial to the security of the system.

3.4.2 Structuring threats and threat agents in layers

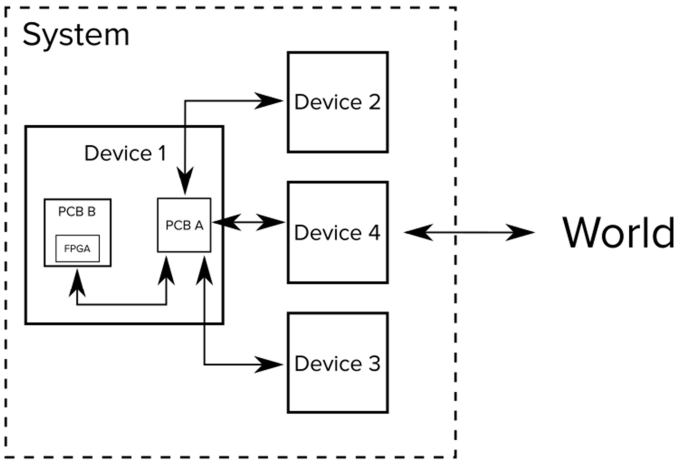


Figure 5 Schematic with threat layers, with Device 4 and PCB (Printed Circuit Board) A serving as interfaces between different threat layers

Dividing the system into multiple layers (Figure 5) eases the location of threat agents and simplifies the placement of appropriate security mechanisms. The FPGAs environment is split by knowledge, equipment, and physical location of threat agents:

- **World:** A growing number of embedded systems have a connection to Wide Area Networks and even the internet; Threat agents targeting this connection can attack the system from remote locations and maintain anonymity while causing widespread damage. Wire-based diagnostic interfaces to the system (like ODB-II in cars[137]) present another point of entry for hackers from this outside world. The risk of the latter case is not as high as the attacker requires physical access to the system. Threat mitigation has to consider all these threats. A physical enclosure (building or chassis) can prevent physical access to the system, and some form of the firewall may prevent unauthorized remote access through WANs.
- **System:** The integration of simple devices into complex systems creates new security challenges. Devices from different vendors may comply with different security requirements and standards. Replacing or tampering with devices in the system, e.g., through a supply chain attack, can endanger the security of the overall system as well as its FPGA component. Parts of the system may be interchangeable; there may be different configurations of the same system deployed.

- **Device:** The device itself may consist of multiple elements, from different vendors and with different levels of sensitivity. An attacker can target the communication interface between the printed circuit boards (PCB) if the device consists of multiple, interconnected PCBs. Shared resources like memory chips or the power supply provide a port of entry for the sophisticated attacker. Input sanitation, a device enclosure, and further security mechanisms like power filtering against power analysis attacks can mitigate these threats from the system.
- **Printed Circuit Board:** The PCB contains the FPGA, auxiliary circuits, and peripheral devices, as well as the storage elements for the FPGA configuration or other elements involved in the reconfiguration process. An attacker gaining access to the PCB may intercept messages, a replay of older transmissions, replace devices, or rewire the connection between them. A physical enclosure may protect the PCB. The deeper layers of the PCB may protect sensitive connections, and glue may be applied to protect sensitive ICs.
- **FPGA-Chip:** The FPGAs threat layer can be, as discussed in 1.2.1, further separated into three sub-layers:
    - The very hardware, static and vendor-provided
    - The configuration that can be changed to fit the applications need
    - An optional software layer that might run on top of an embedded processor (either hardwired or defined through the configuration)

The focus of the model presented here lies in the configuration layer of the FPGA, but it is worth remembering that a holistic approach to IT security is necessary to protect a system in the real world.

Depending on the system's complexity and the desired level of analysis, several layers can be merged, e.g., a system that contains a single PCB with the FPGA may omit the separation between System and Device. Multiple analyses with different points of view (e.g., for each device) may be necessary for complex systems. Threats against the diverse set of components within the FPGA itself and peripheral devices connected to it are discussed in greater length later in the next chapter.

3.4.3 Trust boundaries

Each non-trivial design consists of multiple elements with different levels of exposure to threat agents, level of criticality for the security of the system, and varying levels of trustworthiness and resilience. The border between elements considered more trustworthy than others are usually called trust boundaries. The threat layers presented above are generic trust boundaries, but they are not the only ones. The analyst can assess both sides of the trust boundary based on:

- The trust placed in the subsystem by storing confidential data or by entrusting it with sensitive operations.
- The exposure of the element to threat agents, as a component connected to the internet, will face more threats than a component deeply embedded in the inner layers of the design.
- The (assumed) resilience against intentionally malicious acts.

An attacker will try to gain control over more trusted elements by bypassing or breaking through these trust boundaries. Once in control over the inner circle, an attacker may try to leak sensitive information across the trust boundary into parts of the systems that are less trustworthy, may try to gain further control over the system or perform other malicious acts to solidify his influence over the system and remove traces to him.

Not all violations of trust boundaries are the direct result of a threat agent's activity. In side-channel-attacks, sensitive data passes trust boundaries; in most cases, without an attacker's intervention but through the intrinsic behavior of the system (e.g., power signature of different operations [138]).

The trust layers discussed earlier can be used to develop a first, coarse grain, definition of the system's trust boundaries. In some cases, however, a more nuanced analysis will be necessary. The FPGA configuration stored in a user-accessible SD-Card, for example, must be treated more carefully than a configuration stored in a soldered and glued read-only memory chip, even if both elements are part of the same layer (PCB). A secured (and trustworthy) connection between the FPGA and the manufacturer's internet server (e.g., for upgrades), on the other hand, is an example of communication across different layers that crosses only one trust boundary (if the VPN operates as intended.) The reconfiguration of FPGA resources can create a (temporal) trust boundary if the new configuration has different security attributes.

3.4.4 Attack vectors

Attack vectors are the interface or sequence of interfaces an attacker uses to carry out his attack. Identifying and removing or securing these attack vectors is one important technique to improve the security of the system.

3.4.5 Legitimate actors

Everything and everyone interacting with the system is a potential threat agent. Most of these actors will be other technical systems collaborating with the FPGA to perform a common task. The focus of this analysis lies on the FPGA, and this work, therefore, excludes human actors from any further consideration. The peripheral devices connected to the FPGA are the sole external actors examined (the behavior of the periphery can be, of course, controlled by one or more humans).

Identification and access restriction, even for legitimate actors, are vital methods to minimize the threat exposure of the system. It should further be able to ensure or validate the authenticity and integrity of its actors. Relevant information includes the position of the actor with regards to the trust boundaries, as well as the data and services provided or requested.

3.5 The attack surface of FPGA based designs

The term attack surface describes the potentially exploitable functions exposed to an attacker [139]. The term originally describes the potential vulnerability of a server or personal computer but can be applied to FPGA based systems as well.

3.5.1 Determining the attack surface of FPGA based designs

Determining the attack surface of a design requires knowledge about threat agents, the current and expected threats against FPGA based designs in general, and the very design under review. There are two major classes of attacks:

- Attacks from the outside: carried out through peripheral devices
- Attacks from the inside: carried out by through malicious parts of the configuration

This attack surface is dynamic and subject to change as the system and its environment changes. The threat landscape changes throughout the FPGAs lifecycle as different actors gain and lose access to the system. New attacks, circumventing defense mechanisms (e.g., physical protection), can expose additional functionality to an attacker. Updates or a change in the system parameters may close existing security holes or exposes new (exploitable) functionality.

It is necessary to assess the design from the viewpoints of the threat agents to determine the attack surface of FPGA based designs. An assessment like these requires knowledge about their position within the different threat layers (physical access, remote access, access to the FPGA or only to the system), their motivation, and the target assets. The system-centric threat model introduced later can be used to perform a

more detailed analysis using this intelligence (as well as any other appropriate assumption about potential threat agents). The primary goal of this assessment is to find any functionality and data exposed to a potential attacker and to develop efficient ways to mitigate the risks of this exposure.

3.5.2 Reducing the attack surface of FPGA based designs

Several commonly used design practices reduce the attack surface of the design. Each one of these principles requires detailed knowledge about the systems architecture, assets, data flow, and threat agents.

The design is as simple as possible: A simple design makes the presence of security bugs less and their detection more likely. Developers often use sophisticated designs to scare off attackers – an approach called security through obscurity. Increased knowledge of the system, advanced analysis tools, leaked specification, and design documents may decrease the effectiveness of this approach and increase the chances of an attacker. Further limiting factors of this approach is the limited knowledge about threats at the design and development phase, the life expectancy of the system and limited patch capabilities. Therefore, a simple design providing the same services poses a lower risk of security-relevant bugs and must, therefore, be considered superior to an intricate design.

Sensitive assets are protected. Knowledge of the present and future threats are limited, making a quantized analysis of the security properties hard. Determining the valuable assets of the system (data, processing elements, communication interfaces) qualitatively, however, is a feasible task. The design and its stakeholders must protect these sensitive elements (entrusted to perform critical services or data) against threats.

Elements are classified in different levels of trustworthiness and handled accordingly. Those elements considered as highly trustworthy can perform sensitive operations without further precautions, while those of low trustworthiness (e.g., encrypted IP core from an unknown vendor) mandate greater caution even if they only perform in low to medium level operations.

The exposure of sensitive elements is as small as possible. Access to any asset should be limited to an absolute minimum, even for legitimate actors. Threat agents could exploit any interface between the FPGA and the periphery and within the FPGA configuration. Limiting access denies an attacker insight into a system and reduces the risk of an escalation of privilege attack. Smaller interfaces are also simpler to verify, and this minimization of exposure also limits the damage when legitimate actors are compromised or engage in undesired actions.

Interaction between elements entrusted with tasks of different sensitivity is either prohibited or limited. It is reasonable to assume that critical elements are more thoroughly designed and tested than minor ones. Elements entrusted with tasks of low sensitivity should be considered as less trustworthy and not interact with elements performing tasks of high sensitivity.

Shared resources are sanitized before being used by an element with lower trustworthiness. Sharing resources between elements of different sensitivity and trustworthiness often creates problems. Partial Runtime Reconfiguration creates a new challenge as it allows the reuse of FPGA resources over time. The reconfiguration can result in the crossing of a trust boundary. Sanitization of the FPGA resources can limit the exposure of sensitive data to untrustworthy elements or of untrustworthy data to sensitive elements.

3.6 Conclusion

The term threat modeling comprises different approaches and techniques like asset-centric threat modeling, attacker centric threat modeling, and system-centric threat modeling. Experience from the software domain

indicates that system-centric threat modeling is best suited for developers as it enables them to identify the potential weak points of their design as well as the most suitable threat mitigation strategy by leveraging their superior knowledge about the system they create. The general-purpose tools developed for threat modeling in the software domain must be adapted and extended to meet the demands of the FPGA domain. The goal of the threat model process is, for both domains, the creation of a design that minimizes the attack surface and mitigates the remaining threats. The next chapter introduces the necessary components to create a system-centric threat model by either analyzing the utilized FPGA primitives or, for more complex designs, an abstract representation of its architecture. This abstract model then serves as the foundation for FPGASECML, the formal security model language introduced in chapters 6 and 7.

4 Building blocks for system-centric threat modeling

The systematic analysis of the design provides, as discussed in the previous chapter, a valuable insight into the threats and the required mitigation mechanisms. This chapter introduces the generic building blocks necessary to create practical and consistent threat models for FPGA based design. The starting point of this approach is the application or data flow centric threat modeling paradigm used, for example, in the Microsoft Security Development Lifecycle [140]. Parts of this section are based upon research previously published in [14].

Building blocks structure the process, speed up the analysis, and reduce the risk of missed threats and misunderstandings between stakeholders. Two approaches are presented: a threat model for the low-level components of the FPGA and a high-level model. This architectural model is more suitable for elaborate designs. The latter model is then used to assess the security implications of different FPGA communication paradigm as well as Partial Runtime Reconfiguration(PRR).

4.1 Choosing a suitable level of abstraction for the model

Each model has to find the best possible representation to solve the given problem. A too simplistic representation of reality will lead to an overly simplistic and presumably wrong assessment of the problem; a too complicated representation quickly becomes cluttered and thereby lacks the focus necessary to find an acceptable solution within a reasonable amount of time.

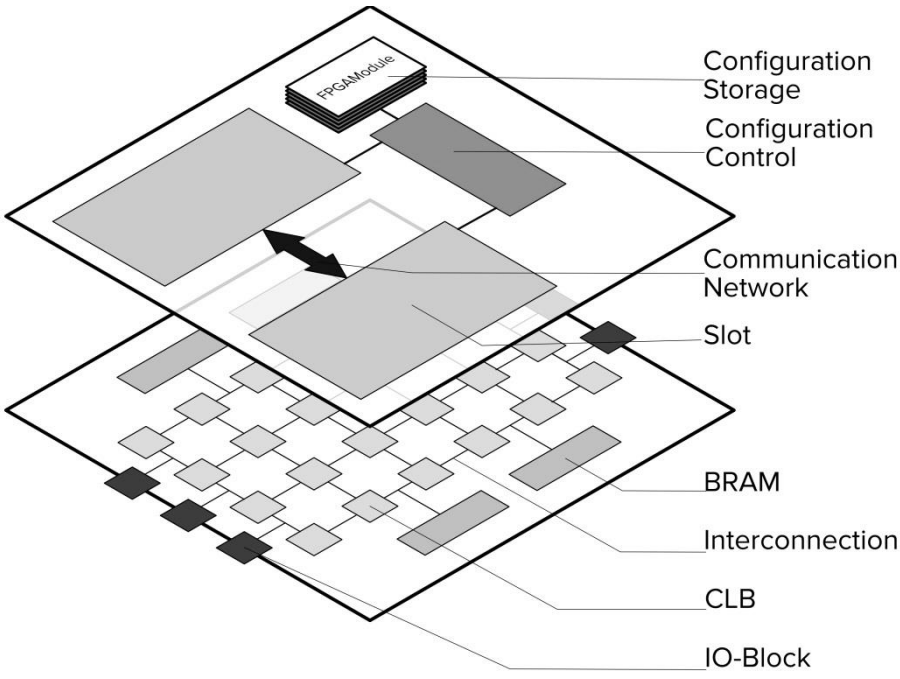


Figure 6 FPGA primitives and their abstract representation

4.1.1 Gate or primitive level analysis

Low-Level analysis of the actual primitives utilized has several advantages. No later element of the build chain can inject new bugs or malicious codes, physical attributes like time and power consumption can be analyzed as well as the security mechanisms implemented at the hardware level. However, there are also downsides to this approach. Not only is the synthesized configuration complex and challenging to assess,

but changes during the (often non-deterministic) synthesis process can change the behavior of the system, rendering the previous analysis obsolete and mandating a new round of low-level security analysis. Probes used to observe the internal communication may alter the behavior of the configuration and can create new security challenges, if not removed or protected. Documentation of the FPGAs configurations format may not be available or incomplete, tools to support the analyses unavailable or unsophisticated.

4.1.2 Source code analysis

The HDL code is far more accessible and may even be hardware independent. Code analysis technology from the software domain could be transferred into the FPGA world. However, the source code is not always available. The toolchains processing it may not be as trustworthy as desired, and the demand for efficient utilization of the FPGA results in an HDL-code with similar complexity and detail level than the binary configuration. It is also more efficient to perform the initial security analysis before any code before the start of the development phase as a later hardening of the system could require an extensive rework of the complete system.

4.1.3 System-level analysis

Any model is, at best, a simplified representation of a real object. In the worst case, it is a complete misrepresentation. The model may not contain the latest changes to the system, built tools could introduce new backdoors, and essential information might be missing. Nevertheless, there are more advantages to a system-level analysis than downsides. The stakeholders can complete the model-based analysis before the creation of the system; it is hardware independent and, if done correctly, focuses the attention of the analysis on the structural weaknesses and their mitigation. Detail level and usability, as well as expressiveness and hardware independence, have to be balanced to create a useful model (chapter 6), thorough review and roundtrip engineering could reduce the gap between the model and the actual system. Appendix B provides a lower level analysis of the FPGA primitives. The next section introduces the building blocks for a high-level analysis.

4.2 The system-centric security model and its building blocks

The heterogeneous structure of FPGAs, their many fields of application, and last but not least, their flexibility creates a multitude of serious threats that must be analyzed and adequately mitigated. The STRIDE mnemonic, already successfully tested in the software domain, proved to be suitable for this task. The user may amend or adapt the analysis performed in this section to improve its fit for a distinct group of FPGA hardware and designs. STRIDE is, however, an informal analysis method, not designed to create a logically consistent and mathematically verifiable model of the weaknesses of the system but to encourage the analyst to consider the multiple and various threats in an uncomplicated way. By aggregating low-level primitives into high-level elements, the threat model becomes more applicable to real-life designs. These blocks are proposed (explained in greater detail in Appendix C):

- **FPGAModules are** representing cohesive blocks of the reconfigurable logic accessible through one or more interfaces and (in designs that use runtime reconfiguration) interchangeable within specified limits. FPGAModules come in two forms:
    - **Processing Blocks (PB)**: operating within the FPGA
    - **IO Blocks (IOB)**: Processing Blocks with access to the periphery
- **Slots**: representing disjunction sets of FPGA primitives to be utilized by a distinct set of FPGAModules
- **Communication Networks**: connecting the Slots and thereby the FPGAModules that use them
- **Configuration Control and Storage** (CC): representing those crucial components entrusted with the configuration.

The next sections informally discuss these components, while chapter 7 introduces a domain-specific modeling language (FPGASECML) to describe these components and their security-relevant attributes formally.

4.3 Threat model representation of a single FPGAModule

The model organizes the information necessary to assess an FPGAModules role in the design into three categories:

- The a**ssets** they contain and that an attacker may target. They can be further dissected into:
  - Data they contain, their sensitivity and the type of access (read, write)
  - Services they provide and their sensitivity
- The s**ecurity mechanism** they deploy to protect their assets. They can mitigate attacks originating from other FPGAModules and (for IO Blocks only) from peripheral devices.
- The FPGA **resources** that utilize or access to fulfill their tasks (LUTs, BRAM, IO pins).

The representation of the FPGAModule should enable the analyst to assess its criticality and whether the deployed security mechanisms are adequate. The description may also include additional information like the trustworthiness or the mechanism deployed to protect the FPGAModules configuration might. There are multiple ways to collect and present this information from whiteboards to index cards to dedicated tools.

4.4 Threat modeling the communication between FPGAModules

This section provides an informal security analysis of different communication paradigm.

4.4.1 Point-to-point connection between FPGAModules

There is neither a shared Communication Network nor a central component controlling the communication between the elements. There are only point-to-point connections between the FPGAModules.

Graphical Model

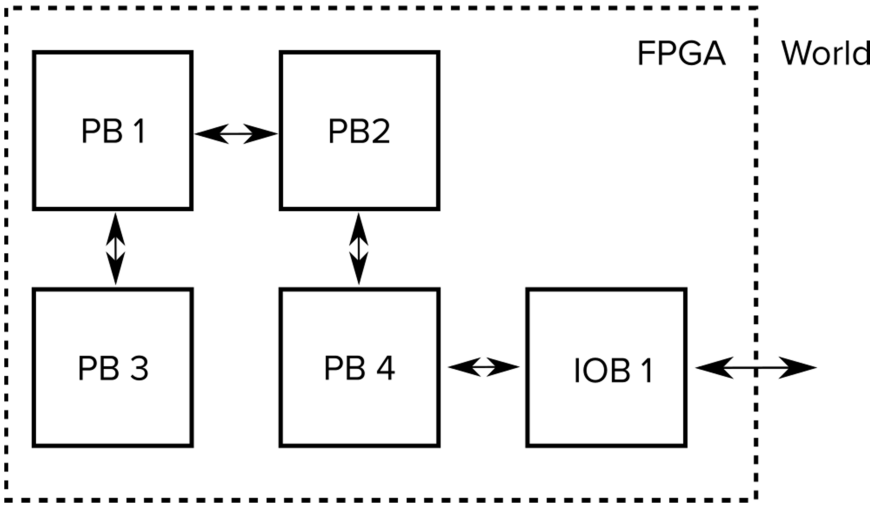


Figure 7 Graphic Model of an exemplary design with direct point-to-point connections between the FPGAModules

The graphical representation shows the design as a directed graph (Figure 7). The FPGAModules are the nodes in this representation, and the directed edges between them indicate the data flow between them.

Security Implications

This unstructured approach can hide sensitive elements within a "sea of elements" of lesser sensitivity. Reverse Engineering and targeted alteration of intricate, unstructured designs are more demanding than for the more structured approaches discussed later. However, a reverse engineer can exploit the following hints:

- Some elements require distinct hard blocks or access to peripheral devices.
- Floorplanning used to avoid conflicts and to reduce development time restricts the resource utilization of different elements.
- Timing constraints, necessary for the correct operation of circuits, restrict the utilization of FPGA resources that collaborate on a task
- Cryptographic algorithms like AES, DES, and RSA, rely on similar structures (e.g., Feistel ciphers, Substitution-Permutation Networks or distinct mathematical operations) and utilize similar primitives (shift registers, BRAMs) for storage and processing. An attacker targeting the cryptographic services of the FPGA will try to find these patterns in the register transfer logic.

Machine learning could support this process, and dedicated obfuscators may cloak these hints; randomized resource utilization could also prevent successful attacks against the current configuration of the FPGA.

4.4.2 Pipelining

ISO/IEC/IEEE 24765:2010 [141, 141] defines a pipeline as:

"a software or hardware design technique in which the output of one process serves as input to a second, the output of the second process serves as input to a third, and so on, often with simultaneity within a single cycle time"

In this context, a pipeline can be created by using the output of one FPGAModule as the input of the next.

Graphical Model

The model of the pipeline (Figure 8) is a series of FPGAModules, each connected exclusively with its predecessor and successor by (elementary) communication networks. The communication between each element of the pipeline is unidirectional.

Security Implications of this design paradigm

The separation of the design in multiple stages eases reverse engineering and the manipulation of sensitive elements. The interconnections between the elements could be the target of men-in-the-middle attacks like eavesdropping or data injection.

The sensitivity of the transmitted data can change as it passes through the stages of the pipeline. Data passing an input sanitization stage is less sensitive, while data leaving a decryption unit is more sensitive than the data entering this stage. The unidirectional data communication reduces the complexity and can, therefore, simplify the creation of a suitable security policy and the identification of the required security mechanisms.

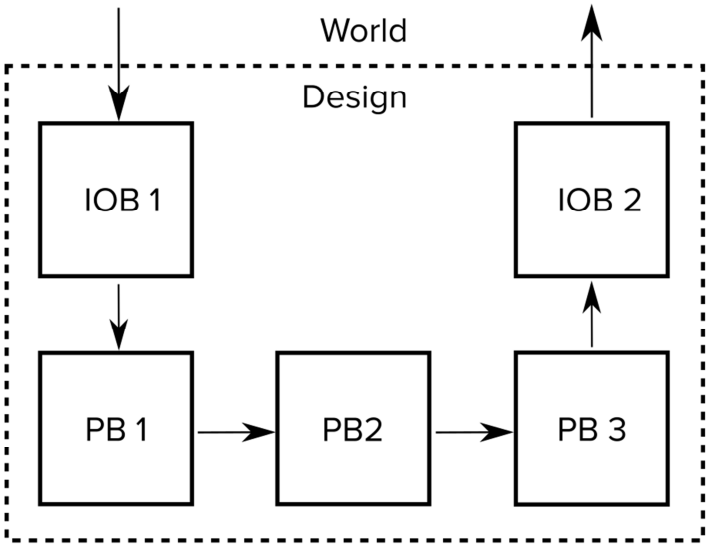


Figure 8 Model of a pipeline based design

4.4.3 Bus-based Designs

Development tools like Xilinx's EDK and quasi-standardized interfaces like the Advanced Microcontroller Bus Architecture (AMBA) make bus-based designs an efficient design paradigm for many FPGA designs. Bus-based designs allow the integration of compatible components from multiple sources, often relying on dedicated tools and industry standards.

Graphical Model

It is possible to depict a bus as a directed graph where the participants are all connected, but a dedicated bus-symbol makes the representation of this network type cleaner (Figure 9). Implementation constraints may limit access to other bus members; the model could reflect these constraints. One example of such a constraint is the implementation of a dedicated firewall integrated into the communication network. The model could reflect this constraint by splitting the bus representation into multiple communication networks, each one with only those FPGAModules able to communicate with each other.

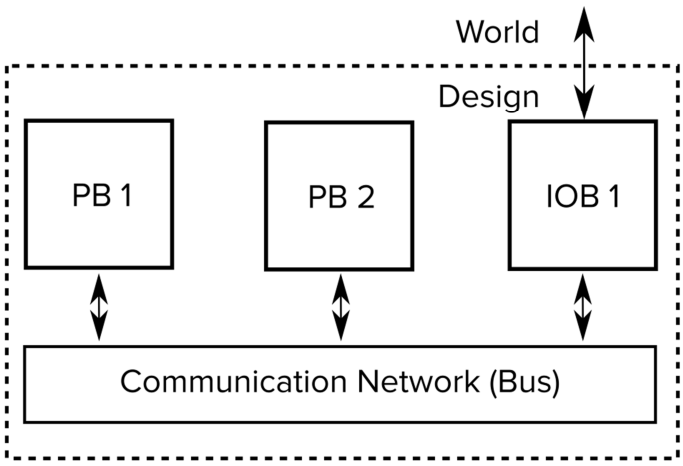


Figure 9 Model of a bus-based design

Security Implications of this design paradigm

The integration of third-party elements raises the risk of malicious code entering the design; this is true for both the generated Communication Network and the FPGAModules communicating over it. The communication network is also a potential target for men-in-the-middle attacks. The generated structure simplifies reverse engineering and malicious alterations. An attacker who lacks knowledge of the system as a whole can tamper with distinct elements or replace them together. By replacing a single, legitimate participant of the bus, an attacker may gain access to and control over the whole bus. He can gather information or inject malicious data into other network members and render the bus useless, e.g., by flooding it with useless information. The use of standardized interfaces and dedicated generators may foster the development of dedicated attack tools applicable to all designs utilizing these interfaces and tools.

4.4.4 Network on a Chip

Network on a chip [142] is another approach to tackle the growing complexity of chip designs. A hierarchical system of routers, defined in the FPGA configuration, connects the FPGAModules.

Graphical Model

The most straightforward model consists of the Processing Blocks sharing a single Communication Network (Figure 10). Elements can access all other members of the network in this simplified model. However, this approach put unnecessary constraints on the selection of suitable FPGAModules as not all nodes might able to communicate with each other. The network can be partitioned in several subnets connected via gateways, discussed later in greater detail, with corresponding filter rules reflecting blocked or technically impossible communication between FPGAModules.

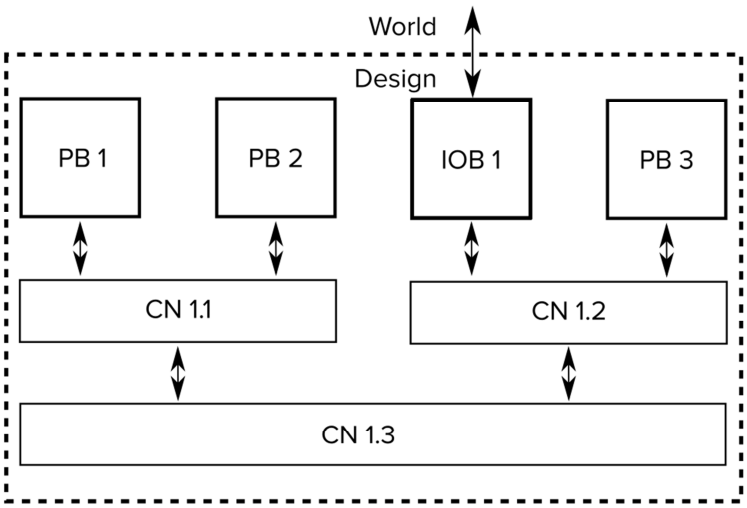


Figure 10 Network on a Chip consisting of different sub-networks (labeled as CN for Communication Network), each connecting two nodes

Security Implications of this design paradigm

The risk assessment for bus-based designs is, with minor alterations, applicable to the NoC paradigm. Data propagation is not as straightforward in NoCs as it is on buses. Depending on the implementation, not all members of the network might be able to communicate with each other, intercepting messages is harder due to the routing. These attributes of NoCs allow the creation of sub-networks mitigating the impact of a maliciously placed FPGAModule or router. The complexity of the communication network raises the capability level necessary for a successful attack. The increased complexity, on the other hand, raises the risk of an implementation error that can result in a security breach. Extensive testing of both the critical

connections and the efficiency of the compartmentalization is necessary to ensure the reliability and resilience of the NoC. The threat modeling method presented in this chapter helps to identify these crucial components.

4.4.5 Gateways

Gateways connect multiple networks. The data flow through this element may or may not be restricted. Only a simple gateway ruleset for filtering is to be expected, as the available computational space and memory are limited. Such a filtering mechanism can be implemented through a state machine accepting (whitelisting) or rejecting (blacklisting) input by predefined rules.

Graphical Model

A CN with one or more trust boundaries can be (Figure 11) used to picture gateways that can serve as a graphical representation of a gateway.

Security implications of gateways

Gateways can either increase the threats to the communication networks or decrease them, depending on their implementation. Gateways increase the threats to the system when they connect networks of different sensitivity and trustworthiness without appropriate restrictions. Gateways decrease the threats when they serve as a firewall, separating networks with high sensitivity or trustworthiness from those with lower sensitivity or trustworthiness while connecting only those FPGAModules that must communicate with each other. Incorrect specification and implementation of these restrictions (usually defined in a dedicated security policy) increase the risk for the system.

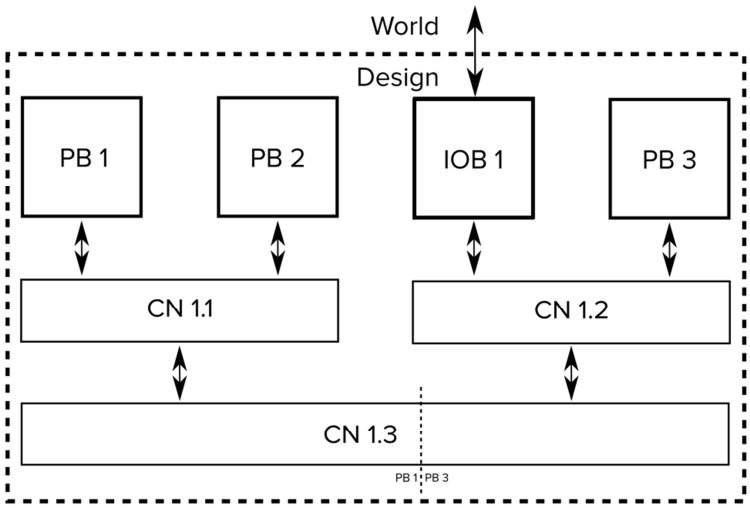


Figure 11 Communication Network with different Subnets, the dashed line indicates that CN1.3 restricts the communication between {PB1} and {PB3}

4.5 Modeling Partial Runtime Reconfiguration

Partial Runtime Reconfiguration introduces several new security challenges. Every additional element increases the complexity of the system and thereby the risk of a security flaw, as each element has its attack surface and may contain either malicious or exploitable functionality. The capacity of the FPGA is no longer the limiting factor for the design, as multiple configurations can share the same resources over time. Side

effects can arise from multiple FPGAModules sharing the same FPGA resource over time, as illustrated by the following example.

Example: A BRAM element serves as storage for a cryptographic key by configuration A and as a transmission buffer by configuration B later (Figure 12). Using the BRAM content (left by configuration A) for padding in configuration 2 leaks sensitive data to the outside world (similar to[143].)

Dividing the functionality into multiple parts simplifies the reverse engineering process as an attacker can identify and target those configurations performing sensitive operations. Comprehensive knowledge of the whole system is no longer required. Limiting the scope of the attacks to distinct, encapsulated elements reduces the risk of destroying the complete system. Reuse of modules or tools with known security flaws in different designs enables the reuse of a successful exploit and could motivate the development and proliferation of dedicated attack-toolkits. Attackers can also utilize the reconfiguration control for replacing legitimate modules with a malicious one. This approach can circumvent security features, to add new functionality, and to gather further information about the system.

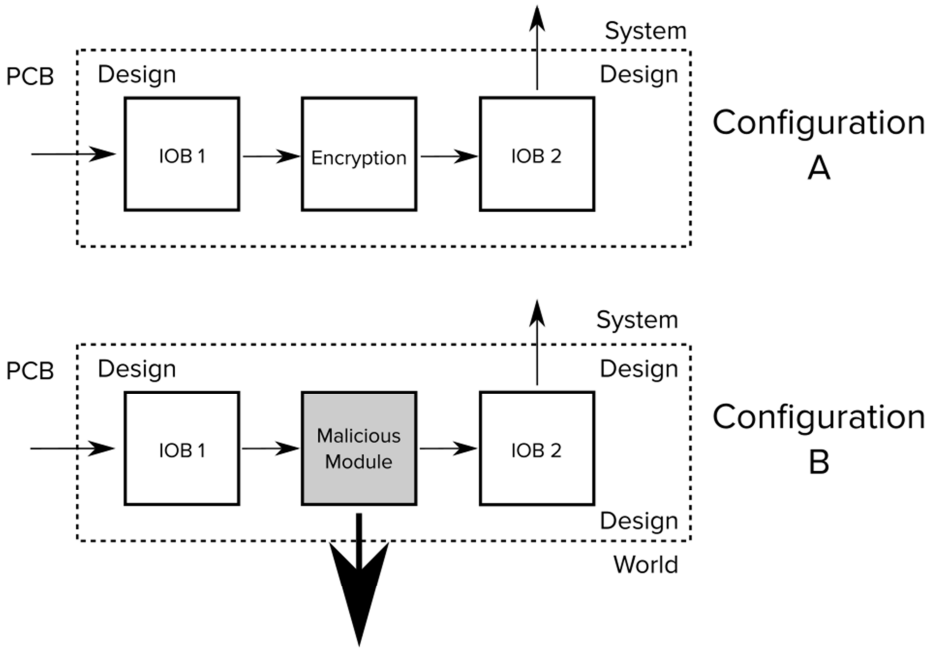


Figure 12 Leakage of unencrypted data into the threat layer world through a malicious module

The additional complexity makes the analysis more demanding; separating the design in two parts eases the threat modeling for PRR based designs:

- The static part of the FPGA configuration remains unchanged throughout all possible configurations.
- The dynamic part of the configuration is subject to reconfiguration.

The static part can, for example, contain:

- Configuration Control (CC) managing the (re)configuration process

- Communication Network (CN) providing the communication backend
- Additional IO Blocks (IOB) and Processing Blocks (PB) available throughout all possible configurations

The dynamic part of configurations consists of one or more FPGAModules that share the same (elementary) FPGA Resources over time (Figure 13). These shared resources are encapsulated into dedicated groups of primitives - the Slots. Slots shall have these attributes:

- Each of these Slots is a distinct group of FPGA primitives like IO elements and CLBs
- each of these primitives must be part of only one Slot
- a single FPGAModule can only occupy each Slot at the same time
- an FPGAModule must utilize the same Slot every time
- Communication Networks link the dynamically utilized Slots with each other, and with the static part of the configuration, they are not subject to reconfiguration.

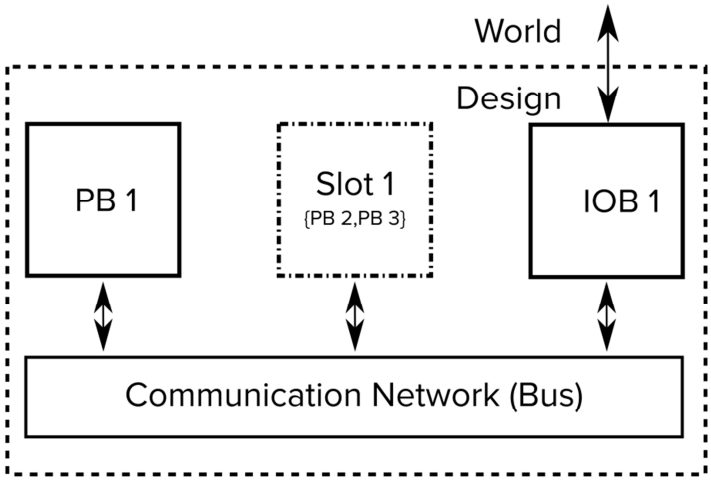


Figure 13 Model with one Slot used for Partial Runtime Reconfiguration, the names inside the bracket show the FPGAModules utilizing these Slots

It is possible to determine potentially dangerous combinations and avoid them by analyzing those FPGAModules sharing a Slot (consecutive) or a communication network (concurrent). This analysis requires a form of temporal logic to describe the reconfiguration process. Thirteen temporal rules can be used to describe the temporal relationship between two configurations A and B, using the temporal relations presented in [144], page 23:

- A meets B (B met-by A)
- A before B (B after A)
- A overlaps B (B overlapped-by A)
- A starts B (B started-by A)
- A during B (B contains A)
- A finishes B (B finished-by A)
- A equals B

In the context of PRR, the above temporal relationships mean that:

- The temporal relationship “finishes“ requires that configuration A overwrite all Slots utilized by configuration B
- The temporal relationships “overlaps“, “during“ and “equals“ that both configuration share no Slot
- The temporal relationships “starts“ and “finishes“ imply that A has control over the resource utilization of B, by, e.g., triggering the corresponding reconfiguration process

This section concludes with an example for each of the temporal relations (Figure 14). A light grey marks Configuration A while configuration B has a darker grey. The rows represent the Slots of the FPGA design; each column represents the state of the FPGA resources at the distinct time, Slots not utilized yet are marked by a dash.

A threat model considering all possible combinations of FPGAModules would be too complicated for large designs. Later chapters present a formal definition of the FPGA architecture, the security policy, and temporal relations between the different configurations to mitigate this problem.

4.6 Conclusion

System-centric threat modeling, already successfully applied in the software world, can be transferred into the FPGA domain. The systematic analysis of the design provides valuable insight into the threats, the vulnerable components, the resilience of the architecture, and the necessary security mechanisms. This approach speeds up the analysis and reduces the risk of missed threats and misunderstandings between stakeholders by offering standard building blocks structures. This chapter introduced the components necessary to perform system-centric threat modeling for FPGA based designs. Predefined blocks make the threat modeling of FPGA designs more accessible. This process can be executed in several ways, depending on the system's security requirements and complexity. For some uncritical and straightforward designs, a simple discussion between the developers using a napkin or a blackboard can be sufficient. An organization requiring or favoring a more formal approach may prefer standardized documents containing the threat model in written form. Chapter 7 presents FPGASECML, a text-based domain-specific language that allows a formalized, human-readable representation of this information that enables the automatic processing and transformation of the model.

	t1	t2	t3	t4
Slot 0	PB1	PB1	PB1	PB1
Slot 1	-	IOB2	IOB2	IOB2
Slot 2	-	PB3	IOB1	IOB1
Slot 3	-	-	PB4	PB4
Slot 4	-	-	-	-

a) A meets B

	t1	t2	t3	t4
Slot 0	PB1	PB1	PB1	PB1
Slot 1	-	IOB2	IOB2	PB8
Slot 2	-	PB3	PB3	PB9
Slot 3	-	-	IOB4	IOB4
Slot 4	-	-	PB4	PB4

b) Either **A overlaps B** or **A starts B**

	t1	t2	t3	t4
Slot 0	-	-	-	-
Slot 1	-	IOB4	IOB4	PB3
Slot 2	-	PB4	PB4	PB11
Slot 3	-	-	-	-
Slot 4	-	-	-	-

c) A finishes B

	t1	t2	t3	t4
Slot 0	-	IOB2	IOB2	-
Slot 1	-	PB3	PB3	-
Slot 2	-	PB4	PB4	-
Slot 3	-	IOB4	IOB4	-
Slot 4	-	PB4	PB4	-

d) A equals B

	t1	t2	t3	t4
Slot 0	-	-	IOB2	PB9
Slot 1	-	-	PB3	PB11
Slot 2	-	PB4	PB4	PB4
Slot 3	-	IOB4	IOB4	IOB4
Slot 4	-	PB4	PB4	PB4

e) A during B

Figure 14 Examples of different temporal relationships between two configurations

5 Modeling access restrictions in FPGA designs

The creation of an architecture-centric threat model provides valuable insight into the design vulnerabilities and the potential threats against it. This analysis, however, must be translated into tangible actions. Resilient designs must separate, vulnerable, highly sensitive parts from potentially hostile or unprotected components. A method to describe the mandated and unacceptable combinations is missing for FPGAs. Reframing this problem as an access control problem clarifies it and charts a path towards a workable solution. This chapter discusses the suitability of several universal access control paradigm for this task. It examines how access restrictions can reduce the design attack surface and which approach is best suited to formulate the access control rules. Chapters 6 to 8 discuss the formal definition of the security rules and ways to enforce them. Parts of this chapter are based and extend research previously published in [145].

5.1 Components of an access control system

Every access control system consists of these components:

- **Objects:** to be protected from unauthorized access.
- **Subjects:** seeking access to an object.
- **Privileges:** defining the kind of access granted or denied.
- **Rules:** formal statements granting or denying subjects access privileges to objects.
- **Rule Enforcement:** ensures that the design is and stays in compliance with the access control rules.

An access control mechanism can also include additional components, for logging purposes**,** or a monitor to detect, report, or stop potentially malicious operations.

Access privileges can be granted to subjects by their functional requirements, trustworthiness, the task they perform, and the security demands of the design. Tighter restrictions reduce the attack surface while increasing the risk that legitimate subjects cannot perform necessary tasks due to a lack of access privileges.

The enforcement of the access control can be either at the discretion of the object owner (Discretionary Access Control- DAC) or a mandatory part of the system (Mandatory Access Control – MAC). The access control mechanism is critical to the security of the system, as it is very likely that the system is compromised after the attacker circumvented it.

5.2 Generic subjects and their attributes in FPGA based designs

Knowledge about who (or what) is accessing the assets is the first step to create an access control system. For FPGAs, these general subjects can be identified:

- **Human** interaction with FPGAs requires a periphery device like a keyboard or a touchscreen. **Peripheral Devices** and their respective IO Blocks invoking services from or providing services for the FPGA
- **FPGAModules** utilize the FPGAs resources to communicate with other FPGAModules and the periphery.

Techniques to control the access of humans to technical systems are well established and documented. Therefore, human interactions with the system are of no further concern for the analysis as it focuses on the peripheral devices utilized for this process.

5.3 FPGAs generic objects and their privileges

In any access control system, subjects can only access objects if they have the right privileges. This section discusses the FPGA primitives (like lookup tables and BRAM) and their privileges. A working access control system may rely only on a subset of these (object, privileges)-combinations.

5.3.1 Slots - the FPGA resources

Subjects can **utilize** and **reconfigure** the FPGA resources. The utilization privilege can be further broken down into **computation**, **communication**, **data storage,** and **data retrieval**. A **readback** operation retrieves the current content of configurable FPGA resources.

5.3.2 FPGAModules – blocks of reconfigurable logic

The access privileges to FPGAModules include **request service**, **provide service**, **store data,** and **retrieve data**. They may also enable subjects to gain **access to** other elements.

5.3.3 IO Blocks – FPGAModules with access to the periphery

The access privileges to IO Blocks include the privileges for FPGAModules and **provide access to** Peripheral Devices (PD). This access could be separated further into: **read from** PD, **write to** PD, **request service from** PD, and **provide service to** PD.

5.3.4 Communication Networks

Subjects can **access** the Communication Network to **send data to** and **receive data from** other participants

5.3.5 Configuration Control and Storage

Subjects may want to trigger a reconfiguration, update an existing configuration, install a new configuration and retrieve the configuration

5.4 Minimal (Object, Subject, Privilege)-set

It is possible to constrain the behavior of the FPGA by defining which FPGAModule (subject) can utilize (privilege) the FPGAs Slots (object). Access to the communication networks has to be defined at the Slot level to achieve this.

5.5 Modeling access control rules

This section discusses the suitability of the most common access control paradigms to the FPGA domain. [18] and [146] provide a domain agnostic discussion of most access control paradigms discussed here.

5.5.1 Multilevel security

In multilevel security systems, the subjects are classified into multiple levels reflecting their trustworthiness and sensitivity. The most prominent representatives of the multilevel security paradigm are the Bell-LaPadula[147] and the Biba-Model[148].

Systems following the Bell- LaPadula enforce two simple rules:

- no element must read data from an element with higher rank
- no element must write data to an element with lower rank

Reversing the rules of the Bell- LaPadula model the Biba model states that

- no element must write data to an element with higher rank
- no element must read data from an element with lower rank

Bell-LaPadula maintains the confidentiality of the system's data while neglecting their integrity. Biba maintains integrity but neglects the confidentiality property. Both systems are classic examples of mandatory access control systems.

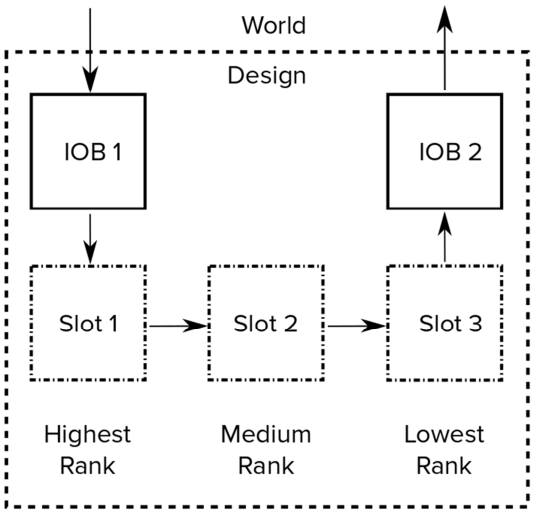


Figure 15 Architecture of Slots for Partial Runtime Reconfiguration ensuring the Biba models security properties, as no module with a lower rank can write data to a module with a higher rank

One application of these paradigms is a pipeline (Figure 15). The system architect can prevent a violation of the rules stated above by separating modules in different classes and making sure that only one module of each class is present at the time. The data is propagated unidirectional from one stage of the pipeline to the next, either rising or dropping in rank.

In a design with Partial Runtime Reconfiguration, the data may stay in place as elements with successively higher (or lower) rank are utilizing the same resource Slot (Figure 16).

The multilevel access paradigm is easy to implement, but the lack of bidirectional communication limits its usage significantly.

5.5.2 Access Control Matrix

An Access Control Matrix (ACM) is a two-dimensional table that stores the access privileges of each object to each subject (Figure 17). The advantage of an ACM is that every combination of subjects and objects is defined. The main problem with Access Control Matrices is their size, as every combination of subject and object has to be defined. An ACM for an FPGA design might contain each utilized FPGA resource as a subject, and requires a privilege assignment for each FPGAModule; this increases the complexity (and memory consumption when used for access control enforcement) of the access control rules. The application of ACMs to FPGA based designs should, therefore, be limited to simple designs, designs with a dense interaction between subjects and objects, and those rare cases where any missing privilege assignment to a (Subject, Object)-combination poses a severe threat.

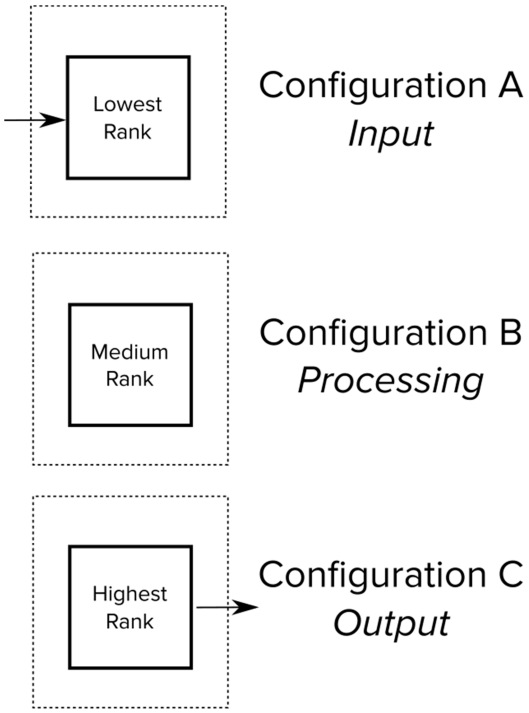


Figure 16 Bell-LaPadula-"Pipelining" through Partial Runtime Reconfiguration

	BRAM1	Slot1	Slot2	Configuration Storage
PB 1	read,write	utilize	none	none
IOB 1	none	none	utilize	none
PB 2	read	utilize	none	none
Config. Control	none	reconfigure	reconfigure	read

Figure 17 Access Control Matrix with a column for each object, and a row for each subject

5.5.3 Access Control List

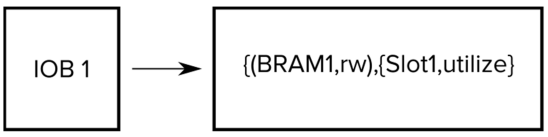


Figure 18 Access Control List with one entry granting the IOBlock IOB1 read and write access to BRAM1 and the right to use Slot1

In Access Control Lists (ACL), each object has a list of their respective subjects and their privilege (Figure 18). Avoiding the overhead of ACMs ACLs are more suitable for designs with a less dense (Subject, Object)-interaction.

5.5.4 Role-Based Access Control

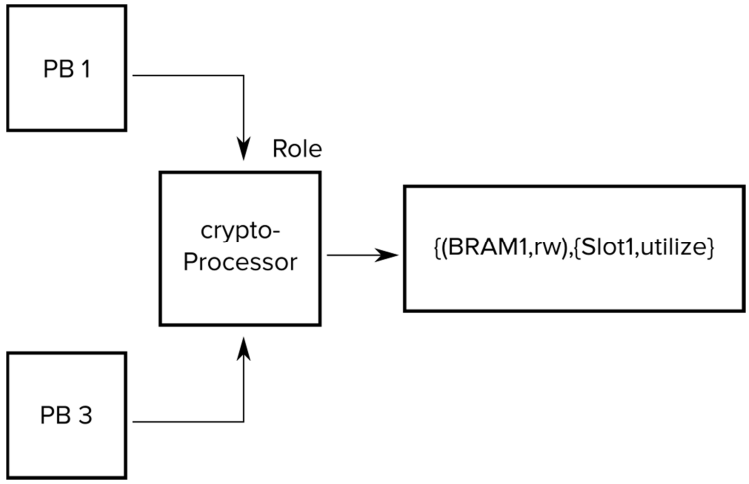


Figure 19 Role-based access control description with two subjects and one role

The Role-Based Access Control (RBAC) paradigm is best suited for complex systems with multiple objects performing similar tasks (or roles Figure 19). By assigning the access rules to roles and one or more roles to each object, RBAC provides flexibility and allows the reuse of rules. Roles for FPGA elements can be cryptographic operations, reconfiguration control, or being a gateway to a peripheral device. RBAC provides little to no advantage for small designs or those whose elements perform disjunctive tasks.

5.5.5 Attribute-Based Access Control

Less common than the other access control methods, attribute-based access control provides a method to describe the interaction between objects and subjects without enumerating each of these elements but by using their attributes.

A definition of Attribute-Based Access Control (ABAC) is:

"a logical access control methodology where authorization to perform a set of operations is determined by evaluating attributes associated with the subject, object, requested operations, and, in some cases, environment conditions against policy, rules, or relationships that describe the allowable operations for a given set of attributes."

[149]

ABAC allows a very abstract and generalized definition of the access control policy but makes it more challenging to find and resolve violations. An analyst might not find a suitable set of attributes for all fields of applications, primarily if the framework must operate in very diverse contexts, but for this domain, ABAC provides a reasonable approach. The trustworthiness indicators presented earlier (section 2.2), on the one hand, and the sensitivity of the operation on the other support ACAB in the formal definition introduced later.

5.6 Access control strategies

This section discusses several strategies to restrict access to the communication network (concurrent access) and FPGA resources (consecutive access).

5.6.1 Concurrent access - communication networks

Multiple strategies can be applied to constrain the participation of communication networks (CN):

- Only those elements required for the sensitive operation shall be present in the Communication Network (to reduce the internal attack surface)
- No IO Block (IOB) shall be present during a sensitive operation (to reduce the external attack surface)
- The design shall remove all elements violating the security policy before the sensitive operation starts
- Sensitive modules shall be removed from the CN as soon as possible to minimize their exposure to (potentially) hostile communication partners

5.6.2 Consecutive access - resource utilization

Only trustworthy elements should utilize FPGA Slots that could contain sensitive data. The system architect can constrain the usage of these memory elements (either internal or external) through the following strategies:

- FPGAModules shall not utilize distinct memory elements with insufficient trustworthiness
- FPGAModules shall not utilize distinct memory elements with lower trustworthiness unless the Slot is proper sanitized

The FPGA should not contain any sensitive data longer than necessary. The design can achieve this, for example, by removing all data touched by a sensitive operation after their completion:

- Sensitive data shall be removed immediately after use
- Sensitive data shall be removed as soon as possible
- Sensitive data shall be encrypted, and only trustworthy elements shall have the key

This approach could lead to conflicts with other non-functional requirements like reliability and performance that must be resolved.

5.7 Conclusion

The conventional access control paradigms allow the formulation of FPGA specific access control rules. Choosing the most suitable paradigm depends on the complexity and security requirements of the design. The Bell-LaPadula or Biba paradigm might be appropriate for a simple, pipeline-based approach, while a more complex design may profit from the higher flexibility of the access control list or access control matrices. Chapter 7 introduces a domain-specific model language whose access control rules follow the Attribute-Based Access Control paradigm.

6 Formalizing the FPGA security model

A model is always a simplified version of reality; the question is whether it is a useful tool to solve the problem at hand. A precise definition of its purpose, constraints, and limits promotes the creation of a useful model. The method introduced in the following chapters formalizes the analytical work presented in the previous chapters. This formalization transforms the ambiguous task of "create a secure FPGA design" into a set of formal statements, stated in a domain-specific modeling language. The associated workflow provides not only the automatic validation of the architecture's security properties against the rules of the security policy. It creates additional value by making the search for a design with the mandated level of security more accessible.

Any useful FPGA security model must contain these aspects of FPGA based designs:

- The static architectural description. The focus of this part lies on the FPGAModules, their assets and security mechanisms, how they interact with each other, and the FPGA resources they utilize.
- The dynamic reconfiguration description defines the changes in the FPGA configuration over time.
- The security policy that defines the required isolation and permitted interaction between the multiple elements of the design.

This chapter illustrates the advantages of a formalized approach through uses cases; it presents the requirements for a suitable model and a high-level description of the mandatory components. Parts of these chapters are based on previously published research [150].

6.1 Models, metamodels, and meta-metamodels

Model-driven engineering [60] knows multiple layers of model abstraction (Figure 1). Each FPGA security model (M1) is an abstract representation of a real-life design (M0). The definition of all valid models is the metamodel (M2), and any further abstraction of a set of metamodels is the meta-meta model (M3). This meta-meta model often has a recursive definition, removing the need for further layers (M4-M∞.)

The structure of the proposed metamodel for FPGA security combines the system-centric threat model and the access control paradigm discussed in the previous chapters. It provides a framework for a simple and unambiguous description and allows the natural transformation into other models.

6.2 Metamodel requirements

The metamodel has to satisfy several requirements to be useful. This section presents the most important of them.

6.2.1 Architecture-centric modeling

The model should only contain information necessary for its task and omit unnecessary details about, e.g., implementation, the supply chain, or details about the FPGA primitive utilized.

Assumptions and constraints

These limitations, assumptions, and constraints where made to create a suitable model of the FPGA architecture:

- All elements meet their physical requirements (like power consumption, timing)
- Attacks utilizing physical attributes are beyond the scope of the model

- The model does not contain the class of auxiliary devices
- The model reflects the system as it was created and maintained
- A peripheral device is everything outside the very FPGA Primitives (IO-Blocks, LUT, and BRAM); this includes hard IP blocks like processors or communication interfaces.

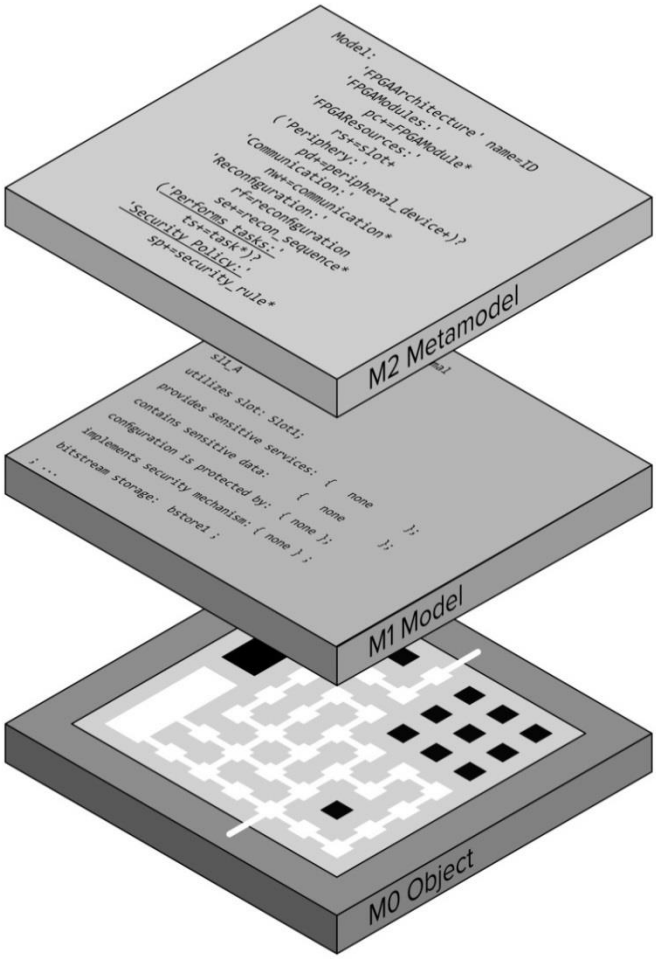


Figure 20 Objects, models and metamodels

6.2.2 Focused grammar but support for reuses and extensions

To overcome the limits imposed requirement introduced in 6.2.1, the model should be extendable through the reuse of, e.g., the metamodel and by sharing unique identifiers across multiple models. These auxiliary models could provide supplementary information about the supply chain of an FPGAModule or the capability and motivation of the various threat agents (more in Appendix B).

6.2.3 Support for Model to Model transformation

It is necessary to translate its content into other representations (as demonstrated in later chapters) to retrieve useful information from the descriptive model. This model to model translation enables the user to solve FPGA specific security challenges in other, more general, domains.

6.3 Metamodel components

Each FPGA security model consists of these three parts:

- **Architecture description**: the components of the design and how they interact with each other:
  - the utilized FPGA primitives, grouped into Slots
  - the FPGAModules with their respective security attribute
  - the communication infrastructure defined through one or more networks
- **Reconfiguration description**: when and how the FPGAs configuration changes
- **Security policy:** the mandated or prohibited interaction between FPGAModules

6.3.1 FPGA Architecture

The architecture component of the metamodel reflects the threat model building blocks from 4.2 (and Appendix C). It consists of these three parts: the parts of the configuration that perform computations, the FPGAModules, the abstract representation of the FPGAPrimitives (Slots) they utilize to perform their task, and the communication infrastructure connecting them.

FPGA Primitives - Slots

Slots represent groups of FPGAPrimitives that are utilized *en bloc*. The model will not contain which FPGAPrimitives are in the Slot, except for BRAM. This omittance simplifies the model, put the focus on the architecture, and increases the hardware independence of the model.

Reconfigurable Logic - FPGA Modules

A set of FPGAModules represents the FPGA configuration. FPGAModules are loosely coupled blocks of (tightly connected) reconfigurable logic representing a distinct set of operations; their description must contain a list of their security-relevant assets (data stored and services provided), the security mechanisms implemented and the FPGA resources that they use.

Communication infrastructure

A directional graph between the Slots represents the communication infrastructure (CI). An FPGAModule based communication model is not expedient as a malicious module utilizing a Slot with access to the communication infrastructures could nonetheless communicate with other FPGAModules sharing the same network (or subgraph). The communication infrastructure is not subject to reconfiguration.

6.3.2 Reconfiguration

There are two ways to describe the reconfiguration process: describing the temporal relationships between the FPGAModules or the reconfiguration events that indicate the (partial) reconfiguration of the FPGA. This section presents descriptions for both approaches and a comparison of their respective advantages and disadvantages.

Configuration based temporal descriptions

The temporal relationship rules (Figure 14) can describe the direct relationship between FPGAModules. A possible description could look like this: {Configuration A after Configuration B} AND {Configuration C after Configuration B}, where each configuration contains the current utilization of each Slot. The problem with this solution is that a sophisticated design may contain too many different configurations to enumerate and order them all.

Reconfiguration events

Reconfiguration events represent a point in time where a set of FPGAModules enters (or utilizes) the FPGA fabric. A new configuration must not change the state of each Slot as FPGAModules may only use a subset of these Slots. Therefore, the sequence of reconfiguration events, starting with the initialization event evInit, is necessary to determine the state of each Slot. The event bases approach limits the expressiveness of the model in regards to the actual configuration while simplifying the description of the reconfiguration process. Reconfiguration events take place consecutively and not parallel. The temporal relation 'after' is, therefore, the only one necessary to fully describe the relationship between reconfiguration events. The sequence of configuration events can be modeled as a simple, ordered list like: ({evInit,event1,event2,…,eventN})

Configuration based description vs. event-based description

An event-based description provides a simpler structure and better readability of the actual reconfiguration process but provides no direct information about the actual state of the FPGA. Using the temporal relationship between all possible combinations of FPGAModules provides this information but is better automatically generated than manually entered. The event-based approach is therefore considered as more suitable to describe the dynamic behavior of the design while the configuration based description is more suitable to describe and validate the security policy.

6.3.3 Security Policy

The security policy must capture the mandated and invalid interactions between the FPGAModules. This section discusses the verifiable properties and intangible attributes, the potential objects for access control based security rules, and the level of abstraction that can describe these rules. The subject, object, privilege set is kept as small as possible.

Verifiable properties of a secure system

Each model is a simplified representation of reality and has, therefore, only limited validity. The model presented later can verify whether a design satisfies security properties like:

- **Data flow between elements with conflicting security attributes is restricted.** The design should expose no element involved in a security-sensitive operation to an element with insufficient trustworthiness. Security attributes express both trustworthiness and sensitivity. Security rules describe unacceptable and mandatory combinations of elements with distinct security attributes.
- **Elements with low trustworthiness are isolated from those with higher trustworthiness.** Elements of lower trustworthiness inhibit a higher risk of compromising the security of the system than those considered more trustworthy. Chapter 2 provides a detailed discussion of the assessment process.
- **The exposure of the design to peripheral threats is restricted.** The design should expose as few functionalities as possible to the outside world. This minimization limits the risk of a successful attack from the periphery. This paradigm is of particular importance to the times where security-sensitive operations take place.
- **No reconfiguration results in an insecure state.** No reconfiguration must result in a violation of the system properties introduced above (this property is only applicable to systems using Partial Runtime Reconfiguration.)

Intangible attributes

There are specific characteristics of any design that can be neither precisely quantified nor verified. They are the result of the analyst's judgment. Qualifying these attributes with high, medium, and low can be considered sufficient for most designs.

- **Trustworthiness:** The trustworthiness of an element depends on multiple factors. The trustworthiness of the developer, the complexity of the code, and the scrutiny of the code review process are relevant indicators. The analyst can make this classification, as discussed in Chapter 2, by examining the element itself, its development process, the supply chain relative to the threat landscape it faces.
- **Sensitivity:** The sensitivity of data or service cannot be derived from a mathematical formula or deducted from logical statements. The impact of a security breach has to be manually assessed for this assignment. Intangibles will dominate this assessment, and different stakeholders may offer different assessments based on their respective interests – it is then up to the analysts to find the right balance between these conflicting views.
- **The resilience of a defense mechanism:** Threat mitigation requires the implementation of suitable security mechanisms. Based on experience and threat intelligence, these mechanisms can be considered as appropriate, or not – a quantitative assessment is hard as the threat landscape is dynamic and knowledge about the threat agent's motivation and capabilities limited. The assessment is further complicated if details about the strength of these security mechanisms are missing, incomplete or wrong (e.g., if provided by a not that trustworthy third party).

Access control objects

The security policy is enforced by restricting the resource utilization of FPGAModules and the communication between them.

Resource utilization

Defining and enforcing the access to critical components is, as stated earlier, an essential mechanism to improve the security of an IT system. Access control is crucial for FPGA designs with dynamic reconfiguration where the design complexity is inherently higher, and no reconfiguration should result in an insecure state. Constraining the resource utilization is necessary to avoid the reuse of FPGA primitives containing sensitive data by elements with insufficient trustworthiness. Restricting the configuration flow means restricting the access to the communication networks by denying untrustworthy elements access to the other elements of the network. Security restriction may be eased or lifted after a highly sensitive operation is completed to improve the efficiency of the design. New slots may be added to the design to compartmentalize sensitive operations.

Communication

IO Blocks and Processing Blocks connected through a communication network can transfer data to and invoke operations at each other. These interactions may exceed expected and legal operations if an FPGAModule is under the influence of a threat agent. Access to the communication networks must be limited to restrict this cooperation to trustworthy elements only. These constraints must be derived from the security requirements of the design and reflected in its architecture. Restricting the data flow between Slots or the spatial separation of networks might be necessary where resource utilization control alone is not sufficient.

Different scopes of security rules

Security rules can have different scopes:

- **Security Attributes**: the scope of security rules can be limited to all FPGAModules with a distinctive set of security attributes.
- **Programmable Logic - IO Blocks and Processing Blocks**: the scope of security rules can be limited to specific FPGAModule (IO Block or Processing Blocks) addressed explicitly through their unique identifier. This approach is less flexible than the indirect address via security attributes but simplifies the process for simple designs and unique FPGAModules
- **Utilized resources – FPGA Primitives**: the scope of security rules can be limited to particular Slots, addressed explicitly through its unique identifier.

6.4 Conclusion

Formalizing the architectural model and the security policy removes ambiguity and provides a solid base for further analysis, both manual and automatic. The domain-specific description language FPGASECML language introduced in the next chapter implements these requirements and provides a framework for further analysis and processing.

7 FPGASECML - A domain-specific language for FPGA security models

There are multiple ways to satisfy most of the requirements laid out in the previous chapter – a simple spreadsheet in excel, a Viso diagram, or a UML profile – each with their advantages and disadvantages. This chapter introduces the text-based domain-specific modeling language (DSL) that makes it easy to formulate the model, read and interface it with other tools. The DSL called FPGASECML, its various components, and its application are discussed, and an example demonstrates the capabilities of the metamodel and the proof of concept implementation. A discussion of current restrictions and possible extensions concludes the chapter.

Appendix D presents the grammar as well as the validation rules and semantic of this language in greater detail and may serve the reader as a reference in this chapter and the next.

7.1 Implementation

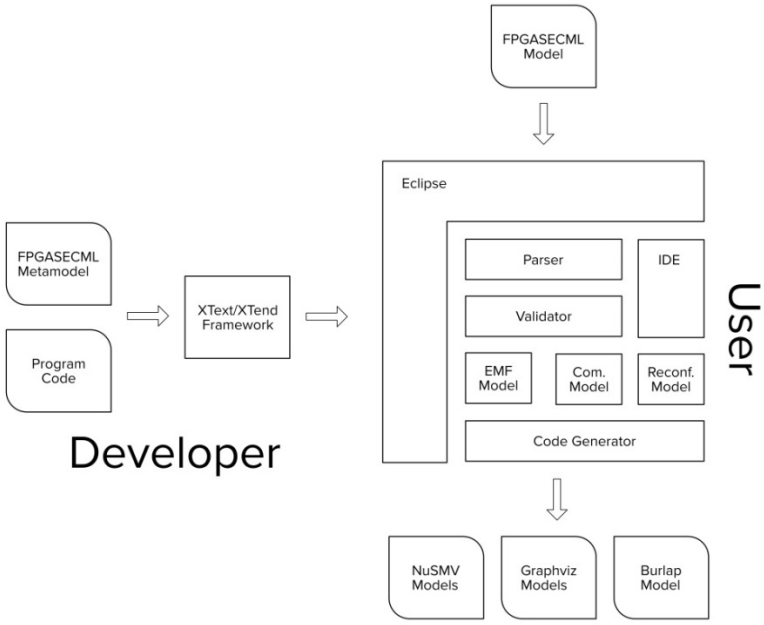


Figure 21 Simplified architecture of the FPGASECML proof of concept implementation

The model language uses a context-free grammar (Figure 21). This approach increases the readability of the model while serving as a base for automated processing (as discussed in the next chapters.) The implementation relies on Xtext/Xtend [151], a DSL development toolkit that generates the necessary code to parse and process context-free grammars in Java. The generated parser/lexer uses Antlr [152]. The processed model is accessible through the Eclipse Modeling Framework (EMF) [153] compatible classes. Xtend is a Java Virtual Machine (JVM) based language with specific improvements over conventional Java. Xtext is a toolkit mainly developed for the Eclipse IDE (Figure 22), but its code can also be used in IDEA or as a standalone program with a command-line interface. Graphviz's [154] dot-notation allows the creation of diagrams without IDE lock-in. The graph models used for these diagrams are also exported as GraphML-files [155] for further processing, for example, in R [156]. The proof of concept

implementation supports the automatic model to model transformation into NuSMV-Models (Chapter 9) and Burlap-models (Chapter 11).

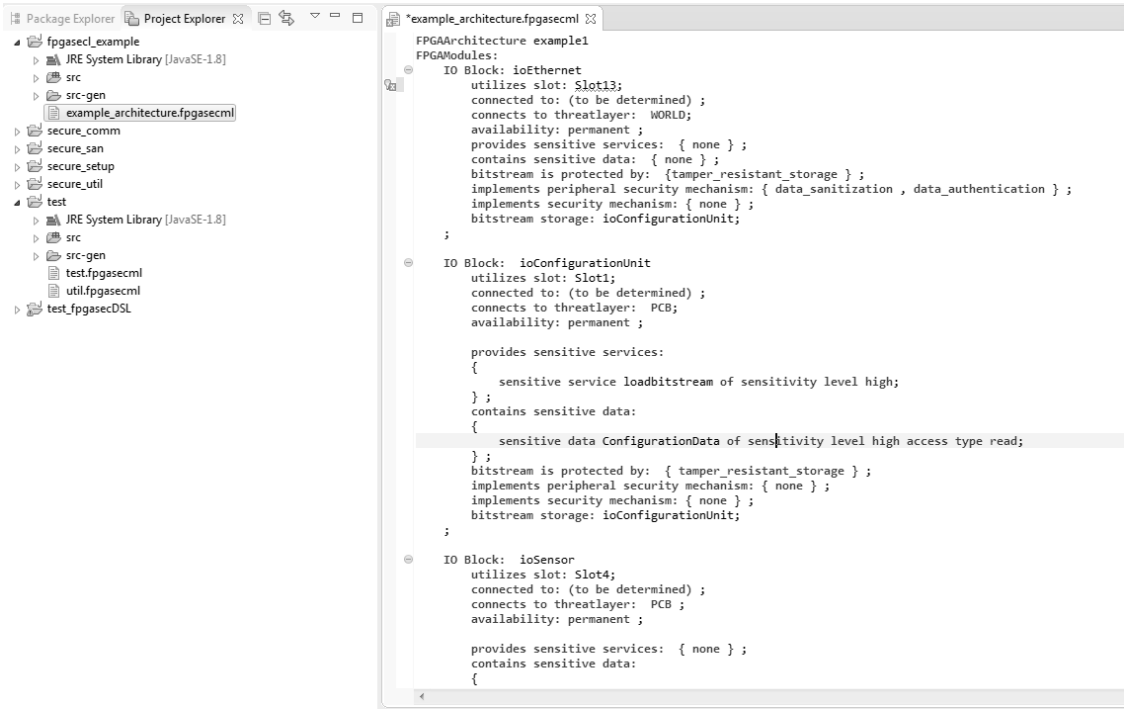


Figure 22 FPGASECML Project in Eclipse

The implementation follows the "*loose grammar strict, validation*"-a paradigm to keep the grammar simple. The validator imposes further restrictions on the (grammatically valid) model to achieve this goal.

7.2 Overview

A model must contain at least one Slot, and each Slot must contain at least one FPGAModule as well as evInit the reconfiguration event marking the initial configuration.

```
/*
      This is a minimal but still valid
      FPGASECML model
*/
FPGAArchitecture minimal // this is a comment

FPGAModules:
Processing Block:
            sl1_A
                  utilizes slot: Slot1;
                  provides sensitive services:
                        {   none    };
                  contains sensitive data:
                        {   none    };
                  bitstream is protected by:  { none };
                  implements security mechanism: { none } ;
                  bitstream storage:  bstore1 ;
            ;
```

```
FPGAResources:
      Slot Slot1 {};
Communication:
Reconfiguration:
Events:
      Event evInit loads: { sl1_A };
Security Policy:
// no security policy defined yet
```

The name of the Slot (*Slot1)* and its initial FPGAModule (*sl1_A*) are arbitrary. The security policy can be left empty; comments follow the well known C/C++ style.

7.3 System description

FPGAModules can be either:

- **Processing Blocks**: accessing data, providing services to other components
- **IO Blocks**: connecting the design with the periphery

FPGAResources include:

- **Slots**: representing disjunction sets of FPGA primitives to be utilized by a distinct set of FPGAModules
  - **Memory:** storage elements within a Slot who can be utilized by one or more FPGAModule

The Communication section contains the different networks that can be either:

- **Buses**: where each Slot has can send and receive data from all other Slots
- **Networks**: providing an elaborate description of the communication network

An optional periphery section allows the description of those (external) devices connected to the FPGA.

Dynamic Reconfiguration

The modeler can define the design's reconfiguration behavior through:

- **Reconfiguration Events**: indicate the re-utilization of at least parts of the FPGA with a distinct set of FPGAModules
- **Reconfiguration Sequences** (optional): independent sequences of reconfiguration events, subject to further security analysis

7.4 Security Policy

The security policy is a set of security rules that must be met by the architecture. Each security rule defines a prohibited (or required) combination of two sets of FPGAModules with diverging security attributes(see D.4.1.3). The system is compliant with the security policy if it complies with all of its rules. The definition of the security policy reflects the attribute-based access control paradigm discussed in chapter 5 (and the trust boundaries discussed in 3.4.3).

Each rule defines either a mandated or prohibited consecutive or concurrent combination of two sets of FPGAModules (a 'prohibit'-security rule represents a trust boundary.) Two classes of security rules can be

identified, one constraining the consecutive utilization for each Slot, and the other one constraining the contemporary utilization of all FPGA Slots:

- **Constrained consecutive utilization** restricts the consecutive utilization of each Slot with FPGAModules of two different sets of attributes A and B.
- **Constrained contemporary utilization** prohibits or requires the simultaneous presence of or communication between any FPGAModules with the attribute set A and FPGAModules with the attribute set B.

Chapter 8.3 contains a detailed discussion of the different forms of security rules and their application.

7.5 Example

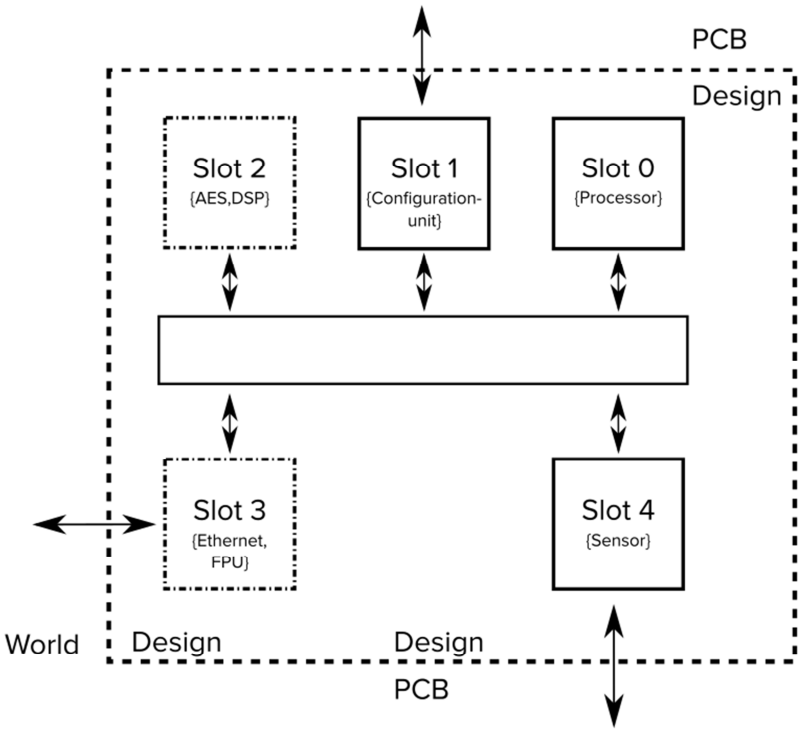


Figure 23 Schematic of example1's architecture

A simple example (Figure 23) illustrates the FPGASECML workflow:

An FPGA performs two tasks:

1. Bidirectional data exchange with a remote server. An AES core encrypts and authenticates [157] the data flow. An Ethernet interface provides connectivity to the outside world.
2. Collect and process data from a single sensor. The sensor and FPGA share one PCB.

Partial Runtime Reconfiguration improves the resource utilization of the FPGA. An external device performs the reconfiguration process (on the same PCB as the FPGA), the same device (Configurationunit) also contains the bitstream. A soft-core IP processor triggers all reconfiguration events.

7.5.1 Architectural description

For the sake of brevity, only a short fragment of the FPGASECML model is presented here (Appendix G contains the unabridged model.)

```
FPGAArchitecture example1
FPGAModules:

Processing Block: pbProcessor
      utilizes slot: Slot0 ;
      provides sensitive services:  {
            sensitive service encryptedconnection
                  of sensitivity level high ;
            sensitive service reconfigurationcontrol
                  of sensitivity level high ;
            } ;
      contains sensitive data:  {
            sensitive data sharedsecret
                  of sensitivity level high
                  access type read ;
            sensitive data software
            of sensitivity level high
            access type read ;
      } ;
      bitstream is protected by:  { tamper_resistant_storage } ;
      implements security mechanism: { none } ;
      bitstream storage: ioConfigurationUnit ;
      ;

IO Block:  ioConfigurationUnit
      utilizes slot: Slot1;
      (…)
      ;

Processing Block: pbAESUnit
      utilizes slot: Slot2  ;
      provides sensitive services:  {
            sensitive service encryption
                  of sensitivity level high  ;
            sensitive service decryption of sensitivity level high ;
      } ;
      (…)
      ;

Processing Block: pbDSP
      (…)
      ;
Processing Block: pbFPU
      (…)
      ;

IO Block: ioEthernet
      (…)
```

```
      ;
IO Block:  ioSensor
      (…)
      ;

(…)

FPGAResources:
      Slot Slot0 { };
      Slot Slot1 { };
      (…)

Communication:
      Bus nwBUS1 {  Slot0, Slot1, Slot2 , Slot3 , Slot4  } ;

Reconfiguration:
      Events:
            Event evInit loads:
                  {ioConfigurationUnit,pbProcessor,pbInitS2,
                  pbInitS3,pbInitS4};
            (…)
```

7.5.2 Architecture analysis

The FPGASECML implementation creates several charts to speed up the analysis process. The tool generates each of these charts in the text-based .dot-format. This format can be further processed with the established Graphviz toolset [154] (or other tools like Gephi [158] and libraries like NetworkX [159] for Python.)

Communication infrastructure

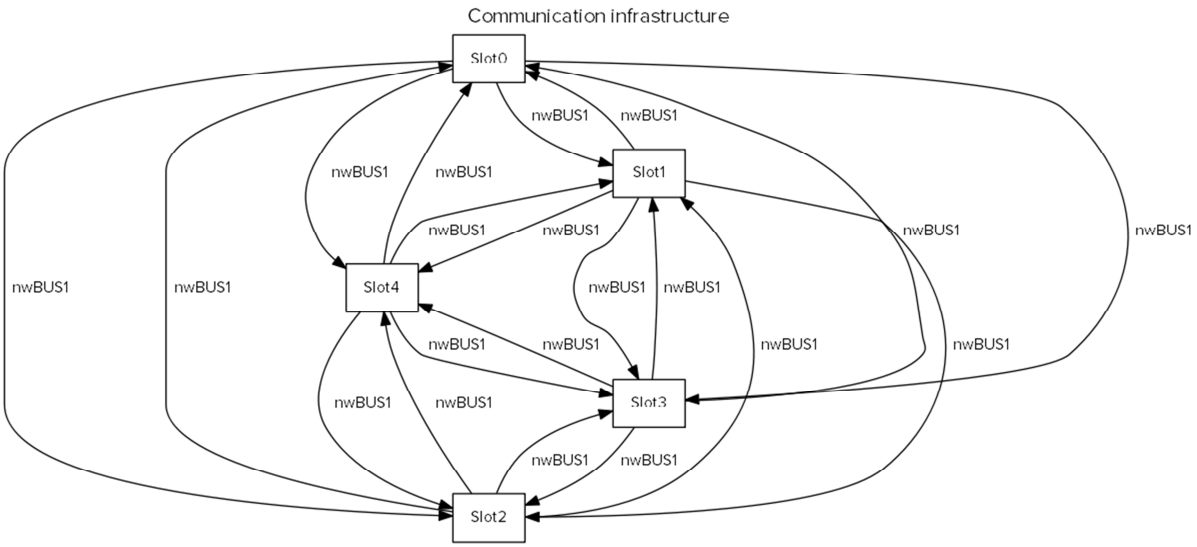


Figure 24 Communication infrastructure of the design

Each FPGAModule has read and write access to all other FPGAModules as all Slots share the same bus (Figure 24).

Threatlayer exposure

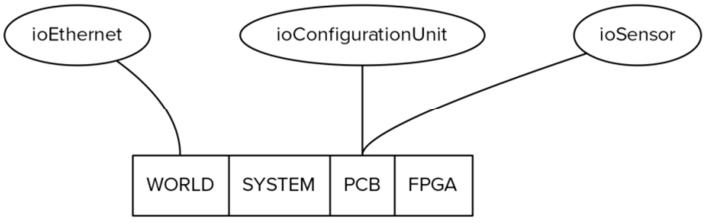


Figure 25 IO Blocks of the design and their respective threatlayer

The FPGAModule ioEthernet exposes (when present) the design to the threat layer WORLD, the modules ioConfigurationUnit, and ioSensor expose it to the threatlayer PCB (Figure 25).

Slot memory

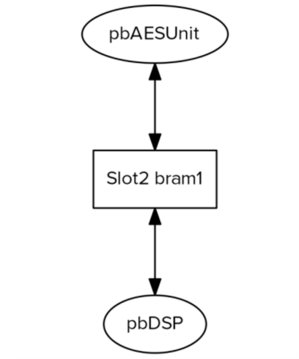


Figure 26 Access to Slot memories

The modules pbAESUnit and pbDSP have read and write access to the only Slot memory bram1 in Slot2 (Figure 26).

Reconfiguration

Even this small example results in a somewhat complicated reconfiguration flow (Figure 27). The reconfiguration flows with and without the 'triggered by' statement is identical as the processor initiates all reconfiguration events – and this FPGAModule is present in all configurations.

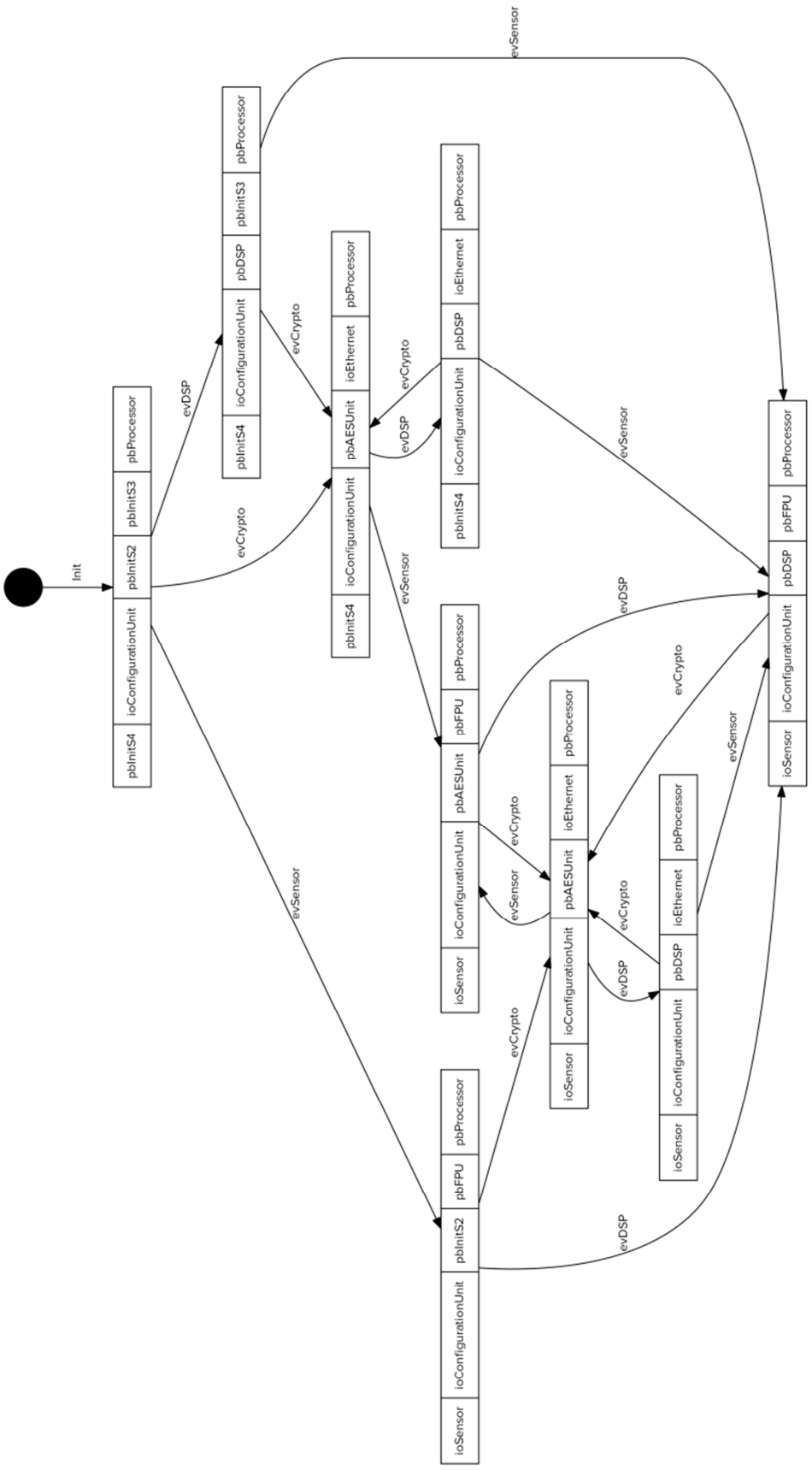


Figure 27 Reconfiguration flow

7.5.3 Security Policy

The security policy shall satisfy these four criteria:

1. No FPGAModule but pbProcessor shall communicate with the ioConfigurationUnit
2. No FPGAModule shall utilize Slot0 except pbProcessor
3. Encrypted communication must take place before any sensor operation
4. The FPGA primitives utilized by the encryption unit shall not be reutilized by any IO Block to prevent the leakage of sensitive data

This FPGASECML fragment reflects these criteria:

```
Security Policy:
      //1.  No element but pbProcessor shall communicate
      //    with the ioConfigurationUnit
      Rule rule1_communication :
      prohibits communication of{
            FPGAModule: { ioConfigurationUnit };
      }
      with
      {
            NOT FPGAModule: { pbProcessor};
      }
      ;


      // 2. No element shall utilize Slot 0 except pbProcessor
      Rule rule2_peprocessor_after :
      prohibits utilization of
      {
            NOT FPGAModule: { pbProcessor};
      }
      after
      {
            FPGAModule: { pbProcessor};
      }
      ;

      (…)
```

The complete security policy can be found in Appendix G

7.5.4 Security analysis

The design is simple enough to perform ad-hoc analysis of the security policy:

No FPGAModule shall utilize Slot 0 but pbProcessor

The design complies with the rule as no FPGAModule other than pbProcessor utilizes this Slot.

No element except pbProcessor shall communicate with the pcConfigurationUnit

The design violates this rule. ioConfigurationUnit utilizes Slot1, and Slot1 is a member of bus nwBUS1, together with Slot0, Slot2, Slot3, Slot4. Therefore all FPGAModules can send data to or receive data from the ioConfigurationUnit.

Encryptedcommunication must take place before any sensor operation

The design satisfies this rule if evCrypto happens before evSensor.

To prevent the leakage of sensitive data the FPGA primitives utilized by the encryption unit shall not be reutilized by any IO Block

The design satisfies this rule as pbAESUnit shares a Slot with pbDSP and pbInitS2, and none of those FGPGAModules is an IO Block.

7.5.5 Revised and compliant design

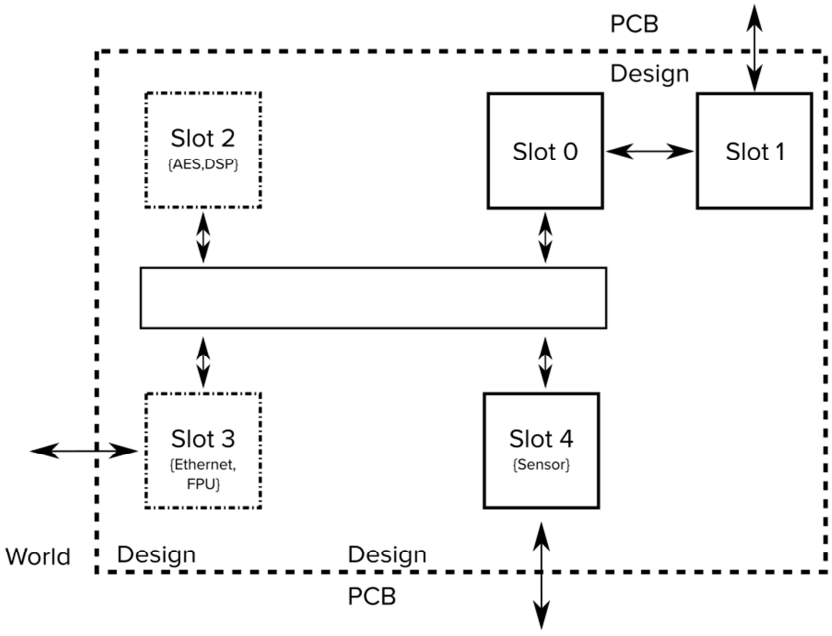


Figure 28 Schematic of the rule1_communication-compliant design

This section demonstrates how the system has to be revised to make it compliant with the security policy (Figure 28).

No element except pbProcessor shall communicate with the pcConfigurationUnit

Removing Slot1 from the bus (Figure 29) and the introduction of a dedicated communication network between Slot0 and Slot1 resolves this problem. The changed communication infrastructure is:

```
Communication:
      Bus     nwBUS1 {  Slot0, Slot2 , Slot3 , Slot4  } ;
      Network reconf { {Slot1} <-> {Slot0} };
```

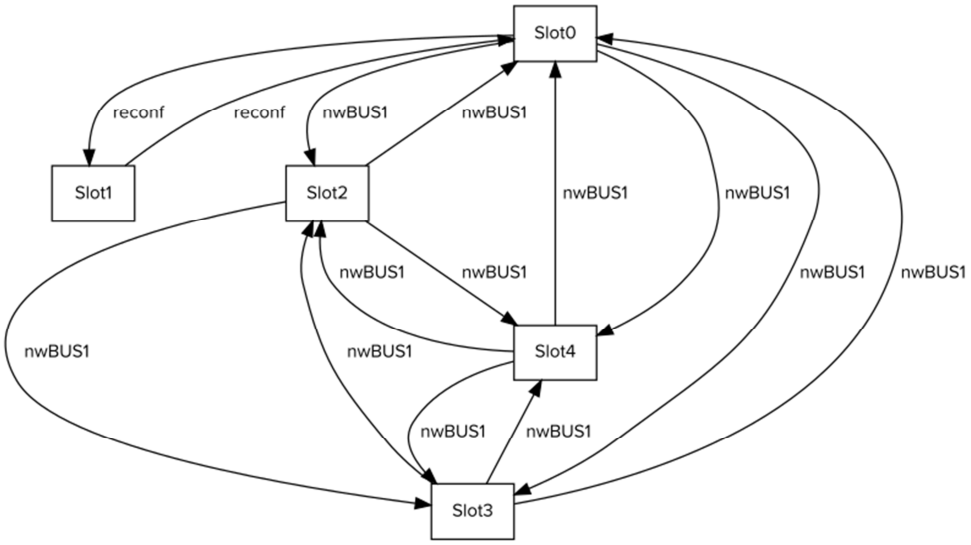


Figure 29 Graph representation of the compliant communication infrastructure

Encrypted communication must take place before any sensor operation

A suitable schedule (e.g., {evInit,evCrypto,evSensor}) resolves this violation. Ensuring that evSensor can only be triggered by an element that is the only member of the evCrypto configuration is another way to reach this goal:

```
Event evSensor triggered by { pbAESUnit } loads: { ioSensor,pbFPU } ;
```

Figure 30 shows the policy compliant reconfiguration flow.

7.5.6 Conclusion

The formal description of the FPGA architecture (FPGAModules, communication networks, and reconfiguration flow), as well as the security policy, promotes a precise definition of the assets and their possible, mandated and prohibited interactions. It removes ambiguity from the threat model while providing a blueprint for the implementation. This approach enables, as described in the following chapters, the rigorous validation of the security properties. The domain-specific language may be supplemented through additional rules when necessary. Model to model translations allows the automatic adaptation of the model content to solve a given problem in another domain. Translations include the dot-models used to create the graph diagrams shown in this chapter as well as the more sophisticated translations presented in section 8.3 (automatic validation of the design with NuSMV-Models) and chapter 9 (weakness analysis with Burlap-Models).

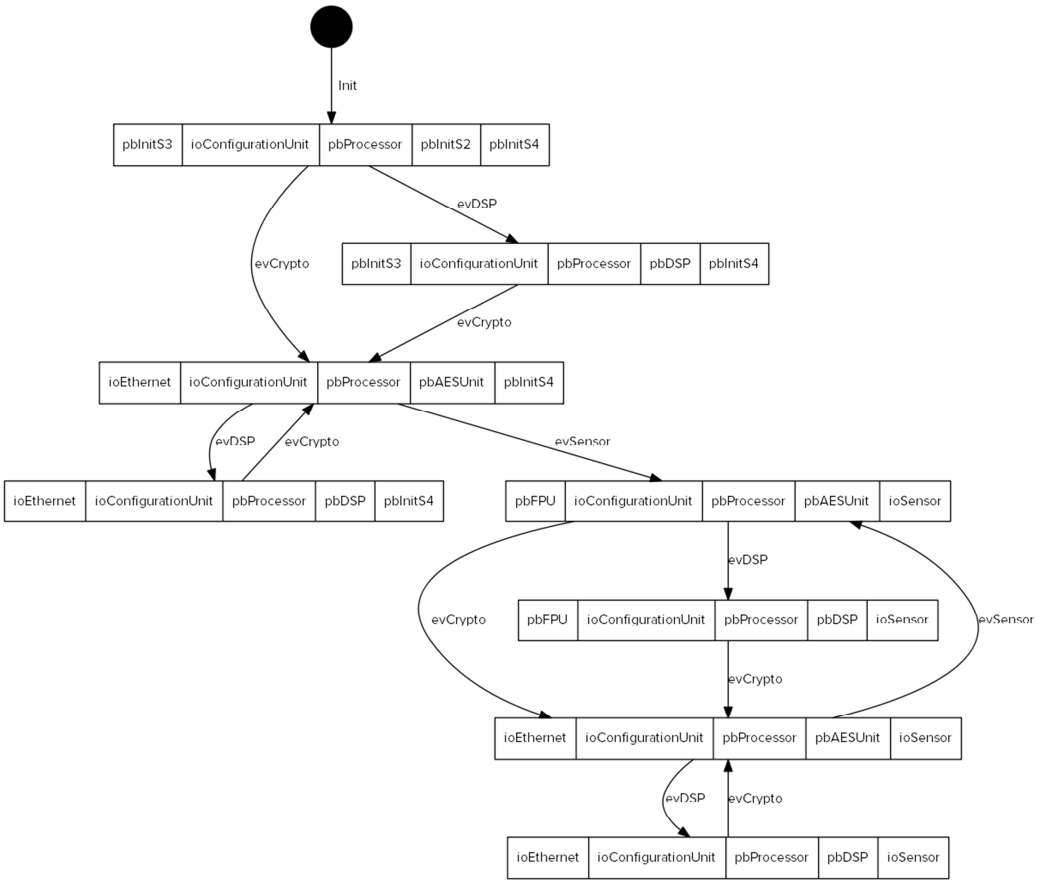


Figure 30 rule3 compliant reconfiguration flow by asserting that evSensor is triggered by pbAESUnit

8 Validation of the designs security properties

Simple designs, like the one presented in the previous chapter, can be validated in an unstructured and informal way. With a higher complexity of the design and security policy, however, comes a higher risk of undetected or overlooked security violations. This chapter discusses the structured validation of the architectural description against the different classes of security rules, either manually or automatically. The first section presents a number of scenarios to illustrate the use cases for the rules; the second discusses the manual validation of a single configuration state as well as the validation of systems with Partial Runtime Reconfiguration. The third section presents a workflow for automatic validation via model to model transformation and model checking, while the third section discusses strategies to remove violations against the security policy.

In the following, the term FPGAModule set (or simply set) refers to the list of all FPGAModules of a design or a Slot with distinct security attributes as defined through a security attribute query (as described in greater detail in D.4.1.3).

8.1 Security rules use cases

This section presents five different use cases for security rules and their field of application. The corresponding FPGASECML models for each scenario can be found in Appendix F.

8.1.1 Resource Sanitization

Removing sensitive data earlier reduces the risk of leakage of information attacks. Another module must perform this task when the sensitive FPGAModule has no control over its removal from the state (preemptive multitasking) or cannot perform the demanded sanitization. Each module providing this service must, therefore, have an appropriate security attribute. Additional modules, providing the only sanitization for a single Slot, may be added to the design. An FPGA module with the demanded sanitization attribute should remove all sensitive information from the Slot.

Scenario

A given design uses a single Slot. FPGAModule A provides an encryption service; a BRAM stores the symmetric key. A potentially malicious module B, utilizing the same BRAM after module A, could access this secret key either as a whole or in parts (modules utilizing the Slot between modules A and B could override parts of the memory.) Module SAN ensures the sanitization of the Slot resource.

8.1.2 Secure setup

This rule ensures that the appropriate preparation of the Slot before any sensitive operation takes place. It can be either used to remove potentially hazardous data from the system or to perform a particular setup procedure (e.g., to establish a secure communication channel) that provides a secure environment.

Scenario

A secure setup procedure, performed by Module sl1_CommSetup, must be completed before Module C can communicate securely with the world.

8.1.3 Consecutive exclusion

These rules prohibit the presence of FPGAModules with one set of attributes after FPGAModules with another set of attributes utilized the same Slot.

Scenario

The FPGAModules A, B, and C share the same Slot and operate on the same block ram. A encrypts the data; module B generates the Message Authentication Code, and module C transmits the encrypted and authenticated data to the outside world. FPGAModule A (accessing unencrypted data) is considered the most sensitive operation and the most trustworthy implementation, FPGAModule B has lower trustworthiness than A, and C has the lowest trustworthiness. To prevent module C from tampering with or transmitting unencrypted and unauthenticated data, C must be utilizing the Slot only after B, and A. B must utilize the Slot only after A to avoid the authentication of unencrypted data.

8.1.4 Concurrent exclusion

It may become necessary to isolate sensitive modules of (presumable) high trustworthiness from modules with lower trustworthiness. This technique reduces the negative influence (e.g., through attacks against a common communication network) of potentially malicious modules to delicate tasks.

Scenario

An FPGA design includes modules performing operations of high sensitivity (Modules A, D, and E) and modules with only medium trustworthiness (Modules B, C). The design must prevent modules with low trustworthiness from endangering sensitive operations. It is further assumed that the mere presence of these modules endangers the security of the system. Therefore, modules of the groups {A, D, E}, and {B, C} must never share the FPGA at the same time. Module F is regarded as highly trustworthy and poses neither a threat to the high sensitivity objects nor is its operation critical enough to cause serious harm if attacked.

8.1.5 Communication exclusion

The global concurrent exclusion rule might put unnecessary strict constraints on the schedule. It is often reasonable to limit the scope of these rules to those FPGAModules who could interact with each other through a communication network (see D.2.3).

Scenario

An FPGA design consists of two independent sections. The first section performs operations of lower security concerns and consists of Slot 3. The two modules {G, H} that utilize this Slot are of low and medium trustworthiness, respectively. In the second, the more sensitive part has modules performing operations of high sensitivity with high trustworthiness (Modules A, D, and E) and modules with only medium trustworthiness (Modules B, C). The design must prevent modules with low trustworthiness from endangering sensitive operations. Mixed utilization of the Slots is permitted as long as they cannot communicate with each other. Therefore, the modules of the groups {A, D, E}, and {B, C, F, G} must never be able to communicate with each other. Module F is regarded as highly trustworthy and poses neither a threat to the high sensitivity objects nor is its operation critical enough to cause serious harm if attacked.

8.2 Manual validation of the security rules

This section demonstrates how to manually validate the compliance of the single state and the flow of reconfigurations with the security policy. This section also introduces the different forms of rules that can make up the security policy – a detailed discussion of the underlying grammar can be found in Appendix D.

8.2.1 Validation of a single FPGA state

FPGAModules utilize Slots (groups of FPGA primitives) to perform their operations. The communication infrastructure (utilizing FPGA Resources themselves, but the model omits this detail) connects the FPGAModules. The current resource utilization of the FPGA primitive with these FPGAModules forms its current "state." The following workflow validates the security rules for communication and resource

utilization for a single, given state of the FPGA. A design is compliant if, and only if, it satisfies all rules of the security policy.

Validating communication restriction rules

Security rule of the shape restrict the data flow between two sets of FPGAModules:

```
Rule name:
            prohibits
            communication of
                  FPGAMODULE_SET_A
            with
                  FPGAMODULE_SET_B
            ;
```

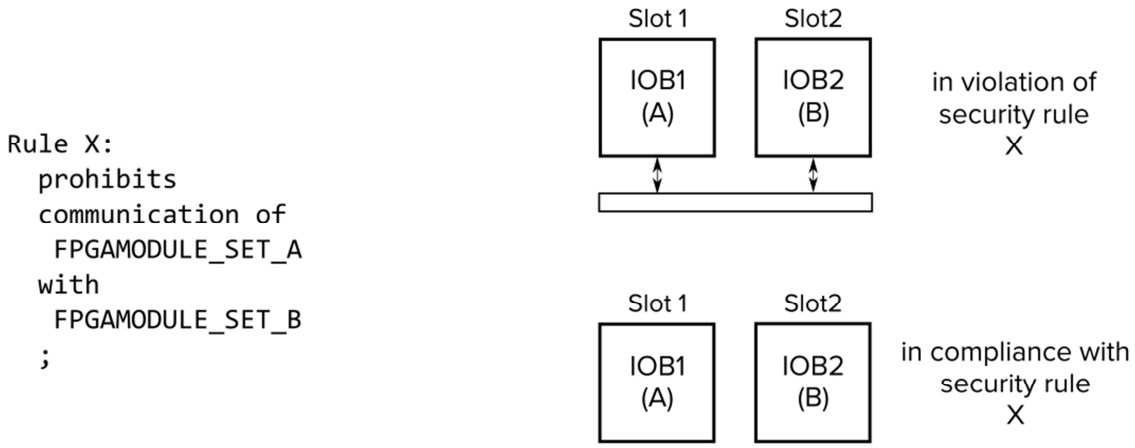


Figure 31 Two designs, one is violating the communication restriction rule and one in compliance. The letter in brackets indicates the corresponding FPGAModule set.

These rules ensure that no FPGAModule with one set of attributes can communicate with one or more FPGAModules with an antagonistically set of attributes (Figure 31). The security rule is satisfied if and only if:

```
For each pair
            (Element_A of FPGAMODULE_SET_A, Element_B of FPGAMODULE_SET_B):
            There is no edge in the communication model with
                  Slot(Element_A) as source and Slot(Element_B) as target
```

Validating communication requirement rules

```
Rule name:
            requires
            communication of
                  FPGAMODULE_SET_A
            with
                  FPGAMODULE_SET_B
```

```
            ;
```

These rules are satisfied if, and only if, a communication network connects at least one FPGAModule of set A and one FPGAModule of set B.

```
For any pair
            (Element_A of FPGAMODULE_SET_A, Element_B of FPGAMODULE_SET_B):
            There is at least one edge in the communication model with
                  Slot(Element_A) as source and Slot(Element_B) as target
```

Validating resource utilization restrictions

```
Rule X:
  prohibits
  presence of
   FPGAMODULE_SET_A
  and
   FPGAMODULE_SET_B
  ;
```

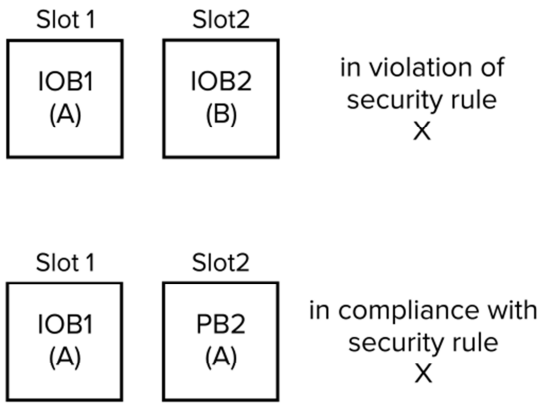


Figure 32 Two designs, one is violating the utilization restriction rule and one in compliance. The letter in brackets indicates the corresponding FPGAModule set.

Validating the resource utilization restrictions means to ensure that the FPGA state does not contain FPGAModules with two antagonistically set of attributes at the same time (Figure 32). Rules of this shape introduce these restrictions:

```
Rule name:
            prohibits
            presence of
                  FPGAMODULE_SET_A
            and
                  FPGAMODULE_SET_B
            ;
```

The rule is satisfied if and only if:

```
If FPGAModule of SET_A is present:
      No FPGAModule of SET_B is present.
```

Validating resource utilization requirements

Rules of this shape mandate the presence of at least one module from both sets if there is at least one module from each set.

```
Rule name:
            requires
            presence of
                  FPGAMODULE_SET_A
            and
                  FPGAMODULE_SET_B
            ;
```

These rules mandate that if one FPGAModule of Set A is present, another FPGAModule of set B must be present as well (a suitable definition of the reconfiguration events and their respective FPGAModule list is often a better way to ensure the simultaneous presence of FPGAModules.)

Validating consecutive utilization rules

Security rules restricting consecutive utilizations must only be evaluated for systems with Partial Runtime Reconfiguration as a single FPGA state has neither a past nor future.

8.2.2 Validating Partial Runtime Reconfiguration

The current resource utilization is its state, reconfiguration changes (part) of the configuration, and thereby the state of the FPGA. Validating the reconfiguration flow means validating a sequence of consecutively visited states, starting with its initial state:

$$State_{old} \xrightarrow{Reconfiguration} State_{new} \quad (8.1)$$

The reconfiguration flow of the system can be represented as a directed graph with the different states as nodes and the reconfiguration events as edges. A graphic representation of these graphs, as created by the FPGASECML proof of concept implementation (like Figure 30), can be used to manually validate designs with a limited number of reconfigurable resources and reconfiguration events. More intricate designs can benefit from this approach as well, but the formal workflow, introduced later in this chapter, is more suitable for them.

Validation of the concurrent rules of the policy

```
state_new is compliant with the concurrent security rules
```

No FPGAModule may join a communication network if its presence would violate a security rule. The graph representation of the system makes it possible to check each state independently for compliance. The transition into these, non-compliant states, must be either prevented *a priori* through a smart design, blocked by the reconfiguration control or at least detected and handled appropriately (e.g., transition of the system into a failure mode) – a combination of these approaches adds defense in depth where necessary.

Validating the consecutive utilization restrictions rules of the policy

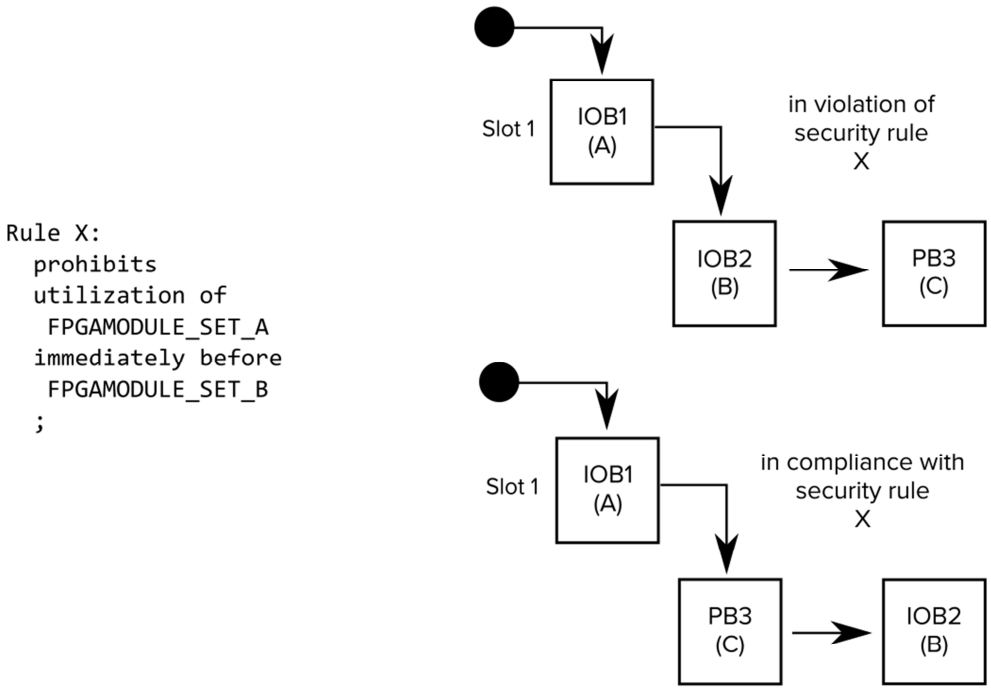


Figure 33 Two reconfiguration flows - one violating the immediate consecutive utilization restriction rule and one in compliance with the rule. The letter in brackets indicates the corresponding FPGAModule set.

An FPGAModule should reutilize no Slot that could contain sensitive data unless the resource (containing the sensitive data) is sanitized or the new FPGAModule is considered sufficiently trustworthy.

```
Rule name:
      prohibits
      utilization of
            FPGAMODULE_SET_A
      immediately?(after|before)
            FPGAMODULE_SET_B
      ;
```

As FPGAMODULE_SET_A after FPGAMODULE_SET_B is equivalent to FPGAMODULE_SET_B before FPGAMODULE_SET_A only the before clause must be considered:

```
For each FPGAModule in the new state:
            a) the new state itself would not violate a security rule
      and
            b) previous utilization of the resource does not prohibit the
               resource utilization through the new FPGAModules
```

Condition a) is fulfilled if the state created through the reconfiguration complies with the security policy. Condition b can come in two forms. The rule can only impose restrictions on the immediate predecessor's Slot utilization (Figure 33):

```
If Slot sl contains FPGAModule from SET_B:
For each states s that is a direct predecessor of state_new:
            sl does not contain any FPGAModule from SET_A
```

Alternatively, (Figure 34) the complete history of the Slot utilization has to be examined:

```
If Slot sl contains FPGAModule from FPGAMODULE_SET_B:
Initial state:
            does not contain FPGAModule from FPGAMODULE_SET_A
      and
            For every path p between the initial state and
            the current state:
                  p does not contain any FPGAModule from FPGAMODULE_SET_A
```

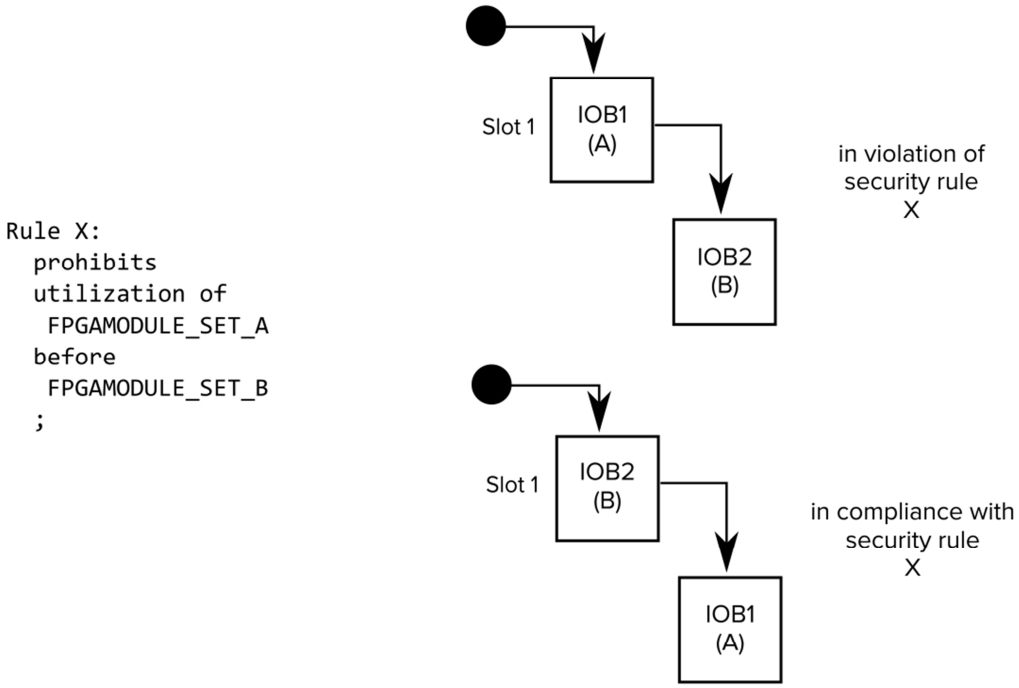


Figure 34 Two designs, one violating the consecutive utilization restriction rule and one in compliance

This path can be retrieved through a graph traversal, but the potential presence of cycles within the graph complicates this assessment. The following algorithm is a suitable alternative for a quick, manual assessment as it rejects all invalid states (at the price of rejecting valid states):

```
For each Slot in the model:
      create a list of FPGAModules utilizing the Slot

For each of these lists:
      Check if two or more FPGAModules in the list have an
      antagonistically attribute set (ignoring the immediate keyword)
```

Another method to deal with potential cycles is to collapse the strongly connected components of the graph. This simple algorithm can be applied to each of these components. Each member of this component can utilize the FPGA before or after the other. A second step analyzes the reconfiguration flow between these "supernodes."

Validating the mandatory utilization rules of the policy

Mandatory utilization works analogous to the restriction-case but for each Slot that contains an element of FPGAMODULE_SET_B either the direct predecessors (immediate-clause) or any path between the initial state and state $_{new}$ must contain at least one element of FPGAMODULE_SET_A (the simplified approach at the end of the last section could accept invalid combinations.)

8.2.3 Conclusion

It is possible to validate the security policy manually, one rule after the other. The reconfiguration graphs generated by the FPGASECML proof of concept implementation can aid this task. It is also possible to write a dedicated tool for the verification process. The next section presents a different approach by transforming the FPGASECML model into a model that is verifiable by well-established (and therefore trustworthy) model checking software.

8.3 Automatized validation of the security policy through model checking

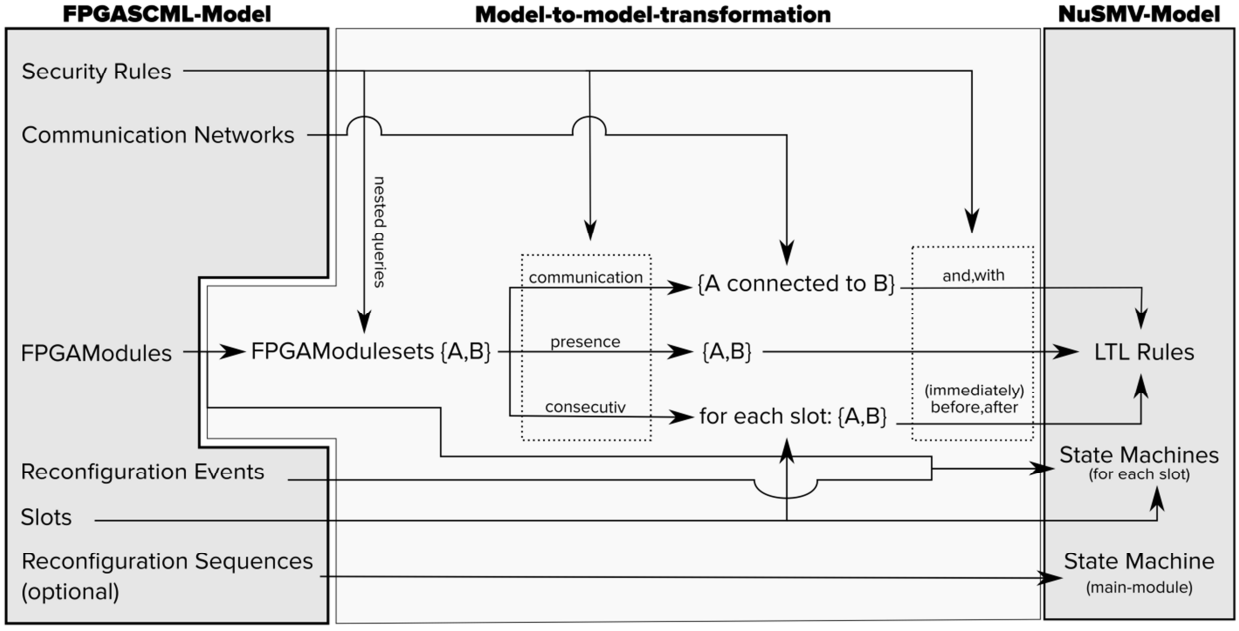


Figure 35 Transformation flow from the FPGASECML to the NuSMV-Model

The manual validation of the policy, however, remains a tedious and error-prone endeavor. This section builds upon the analytic and descriptive methods introduced earlier and applies the well-proven methods model-to-model transformation and model checking (via the open-source tool NuSMV) to them (Figure 35). The automatization of this process reduces the risk of an erroneous manual model transformation and relieves the user from the numb computations present in the last section. It is acceptable that the model checking software will mark secure sequences as insecure as long as it rejects all insecure sequences.

8.3.1 Workflow

The workflow (Figure 35) introduced here transforms the vague challenge of "create a secure system architecture" into the precise task of validating the model's properties against a set of logical statements. For

this purpose, the software transforms the reconfiguration flow into a set of state machines, and the security policy into a set of linear temporal logic (LTL) rules. A schedule is secure if the sequence of configurations utilizing the FPGA resources satisfies formally stated the mandated isolation and interaction between elements with different sets of security attributes. The methods and tools introduced here can perform two tasks: the validation of a given reconfiguration sequence and the generation of such a sequence (if it exists.)

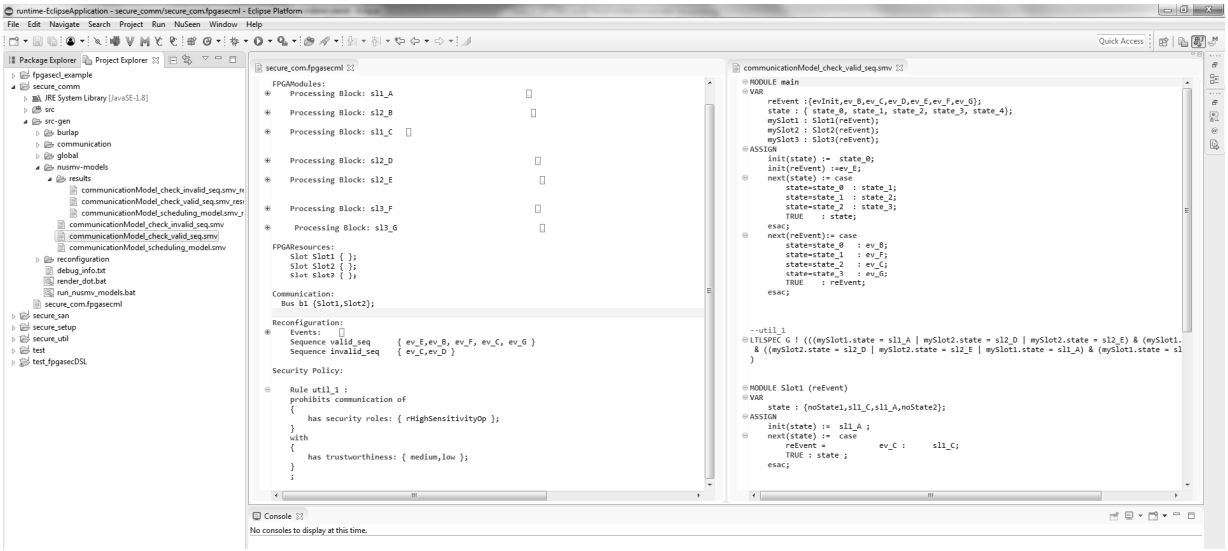


Figure 36 Eclipse IDE with FPGASECML-Model (left) and the generated NuSMV Model (right)
(Appendix H.1 discusses the proof of concept software in greater detail)

8.3.2 Assumptions and restrictions

The workflow has, like every model and tool, certain limits and constraints that must be taken into account.

Security

The strength of this approach relies on the validity of these assumptions:

1. Isolating critical and trustworthy operations from less critical or less trustworthy operations improve the security of the system.
2. Constraining the presence of all reconfigurable elements at any given time achieves the required level of isolation can be
3. An exhaustive set of attributes can express the security properties of each element
4. A finite set of rules can describe the required isolation between elements with different security characteristics
5. The protection of the reconfiguration process itself is sufficient.
6. The model to model transformation does not introduce any new errors

Reconfiguration

The reconfiguration process is also slightly constrained:

1. The generated NuSMV model does not consider the 'triggered by' statements. The current configuration of the FPGA does not restrict or influence the selection of the next reconfiguration.
2. The generated schedule does not consider any functional requirements but uses every FPGAModule at least once.

The 'triggered by' statements (see Appendix D.3) were not incorporated into the model as an attacker may circumvent these (soft) restrictions. The model generator could be extended, when necessary, to include them.

8.3.3 A short discussion of Model-checking

This section provides a short overview of the temporal logic used to describe the model properties and the model checking software NuSMV (new symbolic model verifier) used to validate this model.

Linear Temporal Logic basics

There are three common forms to describe temporal logic [160]. Computation Tree Logic (CTL) is based on a tree representation of the temporal properties, Linear Temporal Logic (LTL) based on a linear representation of these attributes and CTL* a superset of both LTL and CTL. CTL* could not be used as the selected model checking software (NuSMV, [161] [162]) does not support it, and a CTL based description was regarded as less intuitive for this purpose while providing no significant advantage over LTL.

The NuSMV-models use these temporal modal LTL-operators:

- **G** Φ Globally the term Φ has to be valid in all states
- **X** Φ NeXt the next state has to satisfy the term Φ
- **F** Φ Finally one of the next states has to satisfy Φ
- **Y** Φ Yesterday the previous state has to satisfy Φ
- **H** Φ Historically one of the previous states has to satisfy Φ

The NuSMV models use these logical connectives:

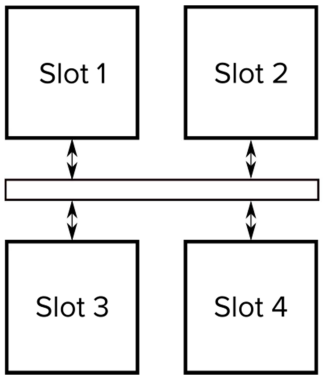
- Φ **->** ψ implication if Φ is true, ψ has to be true as well
- **!** Φ negation of the term Φ must not be true
- Φ **&** ψ logical and Φ and ψ have to be true
- Φ **|** ψ logical or either Φ or ψ has to be true

The reliability of the model checking software is a critical factor. NuSMV is a symbolic model verifier that is well established for research and development tasks and available under an open-source license (LGPL v2.1).

NuSMV tries to contradict the validity of the model properties by finding a counterexample. This feature is used, as discussed later in this section, to find a valid sequence of reconfiguration events by forcing NuSMV to contradict the negated security rules. These schedules often contain reconfiguration events that do not change the configuration. The user should, therefore, consider an automatically generated schedule as a starting point only. LTL rule starts with the keyword LTLSPEC in NuSMV.

8.3.4 The FPGA architecture representation for model checking

Model-checking requires the transformation of the abstract FPGAs design description into a computable model. Aggregating low-level resources (like CLBs) to be reconfigured *en bloc*, into higher-level entities (Slots) reduces the number of possible states and mitigates the combinatorial explosion problem typical for model checking problems. In this model, state machines represent the reconfigurable resources of the FPGA. The different states represent the different FPGAModules that can use a Slot (Figure 37).



Resource	Configuration	
	evInit	Event 1
Slot 1	PB 1	PB 5
Slot 2	PB 2	PB 2
Slot 3	PB 3	PB 3
Slot 4	PB 4	PB 4

Figure 37 Four Slots and their utilization during two events

The state machines accept reconfiguration events as sole input and generate no output. They operate independently of each other. They remain in their present state if they are not affected by a new reconfiguration event.

Slot 2 from the FPGASECML example in 7.5 is converted into this NuSMV-state machine:

```
MODULE Slot2 (reEvent)
VAR
      state : {noState1,pbInitS2,pbAESUnit,pbDSP,noState2};
ASSIGN
      init(state) :=    pbInitS2 ;
      next(state) :=    case
            reEvent =        evCrypto :  pbAESUnit;
            reEvent =           evDSP :      pbDSP;
            TRUE : state ;
      esac;
```

The state-entries noState1 and noState2 are generated for technical reasons and do not influence the validation process.

8.3.5 Security rule to LTL-Rule conversion table

All valid security policy rules follow this pattern:

```
Rule name:
            (requires|prohibits)
            (utilization|presence|communication) of
                  FPGAMODULE_SET_A
            (immediately)?(and|with|after|before)
                  FPGAMODULE_SET_B
            ;
```

Every valid security rule has a corresponding LTL rule. The rules for these conversions are (if both sets (A and B) are not empty):

Type	Keyword		LTL-Rules	
			Prohibit	Requires
Concurrent	communication*		G!((A1|A2…) & (B1|B2…))	G((A1|A2…)& (B1|B2…))
	Presence			
Consecutive (to be generated for each Slot)	After		G((B1|B2…) -> !F(A1|A2…))	G((B1|B2…) -> F(A1|A2…))
		immediately	G((B1|B2…) -> !X(A1|A2…))	G((B1|B2…) -> X(A1|A2…))
	Before		G((B1|B2…) -> H! (A1|A2…)) or G((A1|A2…) -> !F(B1|B2…))	G((B1|B2…) -> H(A1|A2…)) or G((A1|A2…) ->F(B1|B2…))
		immediately	G((B1|B2…) -> Y! (A1|A2…)) or G((A1|A2…) -> !X(B1|B2…))	G((B1|B2…) -> Y(A1|A2…)) or G((A1|A2…) -> X(B1|B2…))

*), each edge of the communication model generates one of these rules. FPGAModules that cannot communicate with each other are thereby not considered in security rules with the communication qualifier

If the sets A or B or both are empty:

Type	LTL-Rules			
	Prohibit	Requires		
concurrent	A or B empty: (TRUE)	(A is empty) and (B is empty)		(TRUE)
		Either (A is empty) or (B is empty)		(FALSE)
consecutive		after	B is empty	(TRUE)
			(B is not empty) and (A is empty)	(FALSE)
		before	B is empty	(TRUE)
			(B is not empty) and (A is empty)	(FALSE)

Appendix F provides an example for each of the scenarios outlined at the beginning of this chapter.

8.3.6 Conclusion

Model-checking can be used to verify if, and to a certain degree when, a design complies with the stated security policy. Transferring the architecture into a suitable, state machine based model and the security policy into verifiable temporal logic properties can be, as demonstrated in this chapter, automated. This step mitigates one source of error. Invalid assumptions about the effectiveness of security mechanisms, and the incorrect assignment of the security attributes, attacks against lower levels of the design or faulty

implementation of the elements remain a danger to the system. The model-based validation of the security attributes does, therefore, not replace extensive testing and Donald E.Knuth's famous warning *"beware of bugs in the above code; I have only proved it correct, not tried it."* [163] applies to this problem just as well.

8.4 Resolving security policy violations

Modeling the system, formulating the security policy, and determining the potential vulnerabilities, is of high importance but does not improve the resilience of the system by itself. The final section of this chapter discusses several strategies to create a secure system within the scope of the model.

8.4.1 Security aware resource assignment

The system architect can isolate FPGAModules with conflicting security properties through the introduction of new communication networks or Slots. These new resources could be reserved exclusively for those FPGAModules involved in sensitive operations or considered untrustworthy.

8.4.2 Implementing additional security mechanisms

Implementing additional security mechanisms, like resource sanitization, in FPGAModules improves their (assumed) resilience and, therefore, trustworthiness. The information derived from the model itself can limit the waste of precious resources by strengthening only the most vulnerable or exposed components (weakest link property of security) of the design. The reinforcement learning-based method (Chapter 9) can aid this search for the weakest link.

8.4.3 Blocking of events (or states)

The configuration control could block reconfiguration events violating the security rules during runtime. This strategy requires a single "enforcer" of the security policy, capable of detecting and blocking any invalid reconfiguration request. The FPGA might not be able to fulfill its obligations and may end up in a complete deadlock as a side effect of this blocking mechanism.

8.4.4 Security aware scheduling

Finding and enforcing a valid sequence of configurations that complies with both the security policy and the mandatory temporal relations between the configurations resolves some violations of the security policy. The system architects could create this scheduling either manually or, as discussed earlier in this chapter, let a suitable tool compute the schedule. A strategy for simple scenarios is to schedule configurations with low sensitivity first and those with higher sensitivity later to prevent untrustworthy modules from reading sensitive data or to going from a higher sensitivity to lower sensitivity to prevent modules of low trustworthiness from tampering with the state(as described in 5.5.1).

8.5 Conclusion

This chapter presented a structured and formalized description of the design and its security policy. This formalized approach allows the validation of the former against the latter. Different security rules require different verification steps. The automatic validation of the design simplifies the validation process, reduces the risk of an error, and allows a quick adaptation and revalidation of either the design or the policy.

9 Model-based vulnerability analysis through reinforcement learning

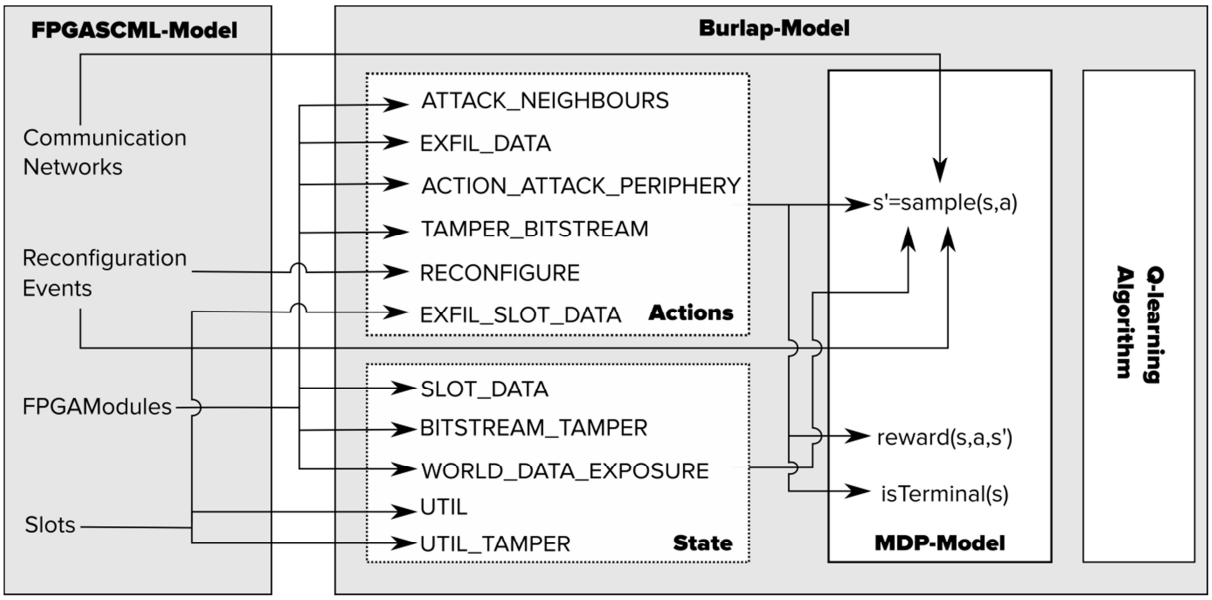


Figure 38 Transformation flow from the FPGASECML to the BURLAP based reinforcement learning model

A successful attack against any non-trivial system consists of a sequence of consecutively executed actions (often called attack vector.) A reasonable attacker will try to find a sequence that minimizes his costs and risk (of detection and failure) and maximizes his reward. Identifying one or more of these plausible attack sequences can help the system architect to identify and mitigate potential weaknesses in the system. This chapter examines how the search for successful attack sequences against an FPGA design is a reinforcement learning [164, 165] problem. It demonstrates how a system modeled in the FPGASECML (Chapter 7) becomes a Markov decision process (MPD) based model processable through well-established reinforcement learning algorithms (Figure 38). The limits and challenges of this approach are discussed, as well as possible steps towards more advanced reinforcement learning solutions proposed. This chapter presents a proof of concept implementation based on the FPGASECML eclipse plugin introduced earlier and the Java library BURLAP. Parts of this chapter were previously published in [59].

9.1 MDP-Representation of the FPGA model

This section presents the basic terminology of Markov decision processes, the simplifications made for the MDP based model and its components – state, action, reward, and terminal states.

9.1.1 Markov decision process

This section provides a short overview of Markov decision processes (Figure 39). Each MDP consist of these elements:

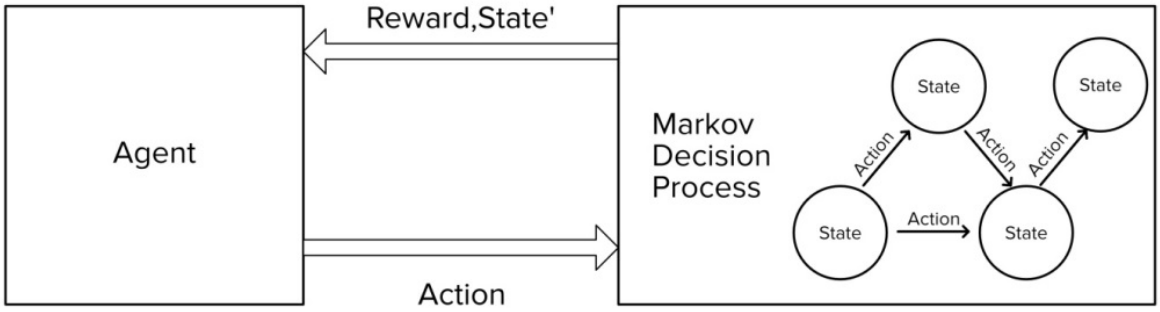


Figure 39 Schematic of the Markov decision process-based model

- **States:** the current state of the MDP contains all information necessary to determine in combination, with an action, the next state of the model.
- **Actions:** to change the current state, the agent can choose from a finite set of actions. By executing this action, the model transitions from the state s to state s'. All actions are atomic and require the same amount of time.
- $\boldsymbol{P_a(s,s')}$: The probability that s' becomes the next state if the agent chooses action a in state s. The states s and s' can be identical. In a deterministic MDP $P_a(s,s')$ is 1 for a single action (a) and zero for all others.
- **Reward** or $R_a(s,s')$**:** Rewards signal the agent's desirable and undesirable state transitions. The reward depends on the current s and the next state s'. The agent receives the reward immediately after the execution of the action. Rewards can be negative (punishment) as well and, not all transitions may result in a reward.
- γ or **Gamma:** a discount factor for future rewards. Gamma has a value between 0 and 1. A typical value for gamma is 0.7.
- ϵ or **Epsilon:** instructs the agent how often a random action should be used instead of the most efficient one (exploration-exploitation tradeoff). Epsilon must be a positive number smaller or equal to 1

The goal of the agent is to find a policy (also called π.) The policy is the combination of states and actions that maximizes the reward an agent can collect from a given start state. The optimal policy can be determined either through a single run of an algorithm (planning) or through multiple interaction sequences (episodes) between the agent and the environment (learning)

Metrics

Many of these algorithms rely on two metrics to assess a state from an agent's perspective: its value and its quality. Both use the γ- factor (between 0 and 1) to discount future rewards.

The value of the state is:

$$V_{i+1}(s) := \quad \max_a \{\sum_{s'} P_a(s,s')(R_a(s,s') + \gamma V_i(s'))\} \quad (9.1)$$

The Q-value (quality) of any state,action-tuple is defined as:

$$Q(s,a) := \sum_s' P_a(s,s')\big(R_a(s,s') + \gamma V_i(s')\big) \quad (9.2)$$

Q-learning

The proof of concept implementation uses q-learning, a reinforcement learning algorithm where the agent keeps tab of the Q-values he has already encountered and takes his next action based on this experience.

1. Initialize the q-table
2. Initialize the state s with the start state
3. Repeat until either a final state or the maximal number of actions permitted is reached:
   3.1. Take action a in the current state s either to
      3.1.1. explore: select at random
      3.1.2. exploit: select a to maximize the expected Q(s, a)
   3.2. Observe the outcome of this action: the new state s' and the reward r
   3.3. Update the entry for $Q(s_t,a_t)$ according to the learning rate α and the discount factor γ:

$$Q(s_t,a_t) \leftarrow (1-\alpha)*Q(s_t,a_t)+\alpha*((r_t*\gamma)*\max_a Q(s_{t+1},a)) \quad (9.3)$$

4. Repeat the steps 2 and 3 for a given number of episodes

The size of the q-table may increase over time when it adds newly encountered states.

9.1.2 Simplifications and constraints

For the sake of argument (and limited computing power), the MDP model omits several details from the FPGASECML metamodel presented earlier. The multiple threat layers of the model have collapsed into a single "WORLD"-node. The attacker has access to all peripheral devices and bitstream storage, independent of their location. The actions in the MDP-model represent a whole class of attacks; several operations will be necessary to execute each of these actions in the real world. It is assumed that the success rate is 100% for each action (deterministic MDP). The MDP model does not consider internal and external defense mechanisms of the FPGAModules. The attacker has complete knowledge of the systems state, and this state solely depends on the actions of the attacker. Some of these constraints can be lifted through a more complicated state representation, while others require more sophisticated methods discussed later in this chapter.

9.1.3 State representation

In a deterministic MDP, the next state chosen solely depends on the current state and the action the agent takes. The state representation must, therefore, not only store the relevant information about the current utilization of the FPGA but all relevant effects of previously executed actions.

FPGA

The generator creates these variables for each Slot:

- UTIL_*slotname* (Integer): the current utilization of the FPGA, represented by the respective FPGAModules unique identifier.
- UTIL_TAMPER_*slotname* (Boolean): true indicates that the module utilizing the Slot is compromised.
- SLOT_DATA_*slotname* (Integer): represents the source FPGAModule of the data (identified through its unique identifier.)

Periphery

The generator creates these variables for FPGAModule::

- BITSTREAM_TAMPER_*fpgamodule* (Boolean): is true if the bitstream storage of the given FPGAModule is under the control of the attacker

- WORLD_DATA_EXPOSURE_*fpgamodule* (Boolean): is true if sensitive data of the given FPGAModule is exfiltrated into the world

9.1.4 Action

An agent can choose from a set of predefined actions to gain and extend control over the system:

- RECONFIGURE_*eventname* changes the state of the FPGA as defined by the given reconfiguration event description.
- ATTACK_NEIGHBOURS_*fpgamodulename* forces a previously hacked FPGAModule to attack all modules he has write-access to, enabling the attacker to exercise total control over them as well. Reconfiguration of these Slots with untampered modules removes the tampered flag. The state remains unchanged if this action is used against an FPGAModule not under the attackers' control.
- ATTACK_PERIPHERY_*ioblockname* forces a (present) IO Block under the control of the attacker. The system remains unchanged if the IO Block is not present. Reconfiguration of these Slots with untampered modules removes the tampered flag.
- EXFIL_DATA_fpgamodulename forces a previously hacked FPGAModule to send its sensitive data to all other modules which have read access (outgoing edges in the communication model) to it.
- TAMPER_BITSTREAM_*fpgamodulename* tampers the FPGAModules storage and forces it under its control, FPGAModules already in the FPGA are unaffected by this change.
- EXFIL_SLOT_DATA_slotname forces the FPGAModule utilizing the given Slot if already tampered with, to send its data to all Slots who can read from data from this Slot. The data is also exfiltrated into the "World" if an IO Block utilizes the Slot.

The actions reflect three of the five STRIDE criteria implicitly:

Target	Action	STRIDE		
		Tampering with	Information Disclosure	Elevation of privilege
Configuration	RECONFIGURE	FPGA state		
FPGAModules	ATTACK_NEIGHBOURS	neighboring FPGAModules		of control over other FPGAModules
	ATTACK_PERIPHERY	IO Block		of control over the IO BLock
	EXFIL_DATA		of data stored within the FPGAModule	of access to the data
	TAMPER_BITSTREAM	… configuration		
Slot	EXFIL_SLOT_DATA		of data stored within the Slot	of access to the data of transmitting the data

Spoofing identity, repudiation, and denial of service attacks might be necessary to carry out one of these actions.

9.1.5 Terminal states

Terminal states are, by default, all states where the data of each FPGAModule is exfiltrated into the world (WORLD_DATA_EXPOSURE is true for every FPGAModule). The code generator does not validate whether a terminal state is reachable.

9.1.6 Reward

Designing an appropriate reward function for a complex reinforcement learning problem is a nontrivial task. This section presents four strategies to create an efficient reward function.

Rewards based on the perceived value of a security rule violation

This strategy awards each security rule a value relative to the (assumed) consequences of the violation. Each action resulting in the violation of the rule receives this previously defined reward.

Reward-based on the perceived value of the state s' for the attacker

This strategy focuses on the question: „What value does a state create for the attacker?“

These questions can serve as a base for this assessment:

- Which tampered FPGAModules are currently present?
- What is the value of each of these FPGAModules? (services provided, data weighted by their sensitivity and kind of access)
- What sensitive data has already been exfiltrated and is currently stored in other Slots?
- Which data has already been exfiltrated to the outside WORLD?

An agent will try to maximize the reward awarded to him. This strategy encourages him to attack high-value targets and spread their data as far as possible. It does not necessarily encourage the agent to move towards a terminal state.

Reward-based on the perceived value change between states s and s'

A refinement of the previous strategy is to encourage the agent to move towards states of higher value by subtracting the value of the state they are leaving.

Reaching a terminal state is rewarded; all other steps are punished.

The most natural solution is to withhold the reward until the agent encounters a terminal state. This strategy can be improved by penalizing (negative reward) each action taken until this terminal state. The penalty encourages the agent to create an effective security policy. The disadvantage of this strategy is that the agent receives no information about desirable states until it reaches a terminal state, complicating the policy search in complex systems and returning no useful information where the terminal states are unreachable. A refinement of this strategy that penalizes distinct actions by imposing additional costs on the agent is discussed later in this chapter.

9.2 Implementation

The proof of concept implementation transforms a valid FPGASECML-model into Java code compatible with the Brown-UMBC reinforcement learning and Planning (BURLAP) library [166] in the version 3.0.1. A reimplementation in, e.g., C++, Rust, or Go should lead to better performance (if there is no demand for the algorithmic flexibility provided by BURLAP)

9.2.1 Deterministic MDP

The generated code for the deterministic MDP model includes:

- **State-**Class, representing the MDP state defined above
- The **sample(State s, Action a)**- function to determine s'.
- The **isTerminal (State s)** function to determine whether a state is terminal or not.
- The **reward(State s, Action a, State sprime)** function is calculating the reward for a given transition. Two auxiliary functions to calculate the attackers' value of the data exposed to the outside world and of the rest of the state, respectively. The default reward function generated awards 10.000 to the agent when a terminal state is reached and punishes the Agent for every other action with -1.000.
- A factory class **FPGADomainFactory.** This class generates a Single Agent Domain (SADomain) to be used by the reinforcement learning (or planning) algorithms.
- The main routine to find a suitable policy for this model through the q-learning algorithm.

- The code further generates an R-file [167] to aid the statistical evaluation of the training process.

9.2.2 Manual extension of the generated code

The user can extend the generated JAVA code to, e.g., transform it into a probabilistic MDP. With probabilistic transitions activated, an action can either succeed, and the system will enter a new state, or it can fail, which means that the system will remain in the original state; there are no side effects or partial successes. The user can define the success rate can either for the whole action class or a distinct action (the code generator creates the necessary constants by default, and the user may change them before the execution of the model). Executing an action leads to a new state if in the function sample (State s, Action a) a random number (retrieved through Math. Random()) is smaller than the defined success rate. It is more efficient to measure the success rate relative to the action most likely to succeed instead of absolute numbers. The learning rate of the reinforcement learning algorithm has to be reduced appropriately once this option is activated. A more sophisticated approach to describe probabilistic MDPs is the definition of "Reinforcement Scenarios" in FPGASECML in the next sections.

9.2.3 FPGASECML defined scenarios

This deterministic model is simple and requires no further input from the FPGASECML definition, but every change of the MDPs requires a change of either the generated code or of the code generator. This is problematic as a meaningful analysis requires multiple experiments with different parameters. Each scenario is a combination of MDP parameters like the reward, the success likelihood, and the costs of the different actions. It also contains hyperparameters, like the learning rate. FPGASECML scenarios allow the independent definition and evaluation of these parameters. All scenarios share a set of parameters like γ and the exploration-exploitation tradeoff ϵ. Appendix D provides a detailed description of the FPGASECML-grammar. Appendix H provides an example of this workflow.

9.3 Limits and restrictions and extensions of the MDP based analysis

The MDP based model presented here relies on certain constraints that could be lifted by supplementing the presented model.

Hierarchical learning [168] could combine this high-level model with the low-level activities needed to execute the strategy. It is also plausible that the attacker does not know the state of the system but has to guess it from noisy indicators, like side channels [138, 169]. A Partially Observable MDP (POMDP [170] can mimic this behavior - BURLAP provides minimal support for them-) Transforming the MDP into a stochastic game allows the integration of multiple actors (e.g., an active defense mechanism.)

Navigating the vast search space remains the main problem of reinforcement learning. Limiting the content of the state and the number of actions eases this problem but restricts the expressiveness of the model. Constraining the search space through transition probabilities and costs is another method but increases the number of experiments. A functional approximation of the QTable could provide an avenue towards an improved RL based weakness analysis. One candidate for this approximation is deep neural networks, whose general feasibility has been demonstrated by the DeepQ [91] algorithm. This more recent technique replaces the QTable with a neuronal network that approximates the value of the given state, action combination. Suitable neural networks mitigate the state explosion problem and should be able to detect a pattern. They may, therefore, be used to extract and abstract information from a feature-rich state, action representation in the same way convolution networks extract information, e.g., from a picture [81].

Policy shaping [171] or reward learning [90] provide other avenues to explore. A sufficiently competent and autonomous reinforcement learning system could be pitted against real-life systems (preferably in a laboratory environment) for further refinement without human interference (similar to AlphaZero [172].) These proposals require at least a rudimentary implementation of the design, and the insight gained from attacking this system could be transferred to other systems still in the design and analysis phase.

Finding the best parameters for the reward and success rates represents an additional challenge. Domain experts may have different assessments of the threat landscape or varying confidence in their assumptions. Running a large number of scenarios with different parameters also increases the chance to find an efficient and robust solution. In later stages of the development cycle, data from penetration tests and security incidents could verify the validity of the assumptions made. To simplify the creation of scenarios, the point values for cost and success rate could be replaced with distributions (similar to hierarchical learning in probabilistic programming) that serve as the source for the actual values of each run.

9.4 Further opportunities and challenges

The vastness of the search space to be processed to find a (sufficiently) optimal solution remains the central problem of reinforcement learning. Limiting the search space by constraining the content of the state and the number of actions eases this problem but limits the validity and usefulness of the model. The still exponential growth of computing power and on-demand cloud-based solutions may provide one building block towards more realistic models in the future. Functional approximations of the Q-Function could provide another avenue towards an improved RL based weakness analysis. Deep neural networks, whose general feasibility has been demonstrated by the DeepQ [91] algorithm, may be used to extract the appropriate information from a feature-rich state/action representation in the same way convolutional networks extract information from a picture [81]. Deep neural networks have also shown a (limited) capability of transfer learning, a feature that may allow the user to utilize a neural network pre-trained on one system to learn the Q-function of another, shortening the learning phase in the process. The necessary training and validation data could be gathered during penetration tests first and later refined through policy-shaping or, once the system is sufficiently competent, by pitting it against real-life systems (preferably in a laboratory environment) for further refinement without human interference. It is also possible that other methods, like evolutionary strategies [173], will enable us to solve this (and similar) problems in the future.

9.5 Conclusion

Reinforcement learning can identify potential weaknesses and the steps an attacker may take to exploit them. The developer can start the search for weaknesses before the implementation of the actual system. The model-based approach enables the parallelization of the search process by using multiple instances. It is possible to transform any correct FPGASECML based description into the underlying Markov decision process. The automatic model-to-model translation reduces the developers' workload, decreases the risk of errors, and keeps both models in sync. Probabilistic transitions allow the user to evaluate his assumptions about the strength of attack and defense. Adding more state information as well as actions to the MDP model enhances its expressiveness, but the computational costs may exceed the value gained. Adding more advanced techniques like Options, Partially Observable MDPs, and stochastic games to the generated code can increase the functionality of the model, but only where the circumstances justify the significant increase in complexity. Further advances in the field of reinforcement learning, maybe in combination with Deep Learning, could enable the creation of an entirely automated penetration tester, able to find and exploit weaknesses in real-life systems. The DSL based approach introduced here could be of use to bootstrap such as system by providing structured information about the systems assets and the interaction between them.

10 Conclusion

Resilience against attacks is a crucial attribute for any IT system, including those relying on FPGAs. The methods and processes for conventional IT systems are more advanced than those in the FPGA world where the primary focus of manufacturers and security research has lied on the protection of intellectual property. Proven techniques of the software industry can be transferred into the FPGA domain to close the gap, and newly developed techniques can leverage the unique features of FPGAs to increase the security of the system. This thesis presented an analysis and adaptation of appropriate security methods from the software domain into the FPGA world. A method to formalize the FPGA security challenge via FPGASECML based models were presented. FPGASECML is a domain-specific language, suitable for a system-centric threat modeling and the formal definition of a security policy. The formal validation of an FPGA security policy where introduced as well as a method to isolate FPGA elements with different security sensitivity through Partial Runtime Reconfiguration. A reinforcement learning-based method was used to gain further insight into the weakness of the system and how an attacker may exploit them. The remainder of this chapter provides a more comprehensive summary.

Analyzing the existing security features for FPGA based designs leads to the conclusion that solutions for many security challenges in FPGA based designs problems are available. However, to the author's knowledge, no integrated development process to develop secure FPGA based designs exist.s The creation of appropriate defense mechanisms requires intelligence about the threat agents, especially their motivation and capabilities. The generic threat agents discussed in this thesis provide a starting point for further analysis of both the design under review and the threat agents against it. FPGA based designs are, like any other IT system, exposed to different threat agents throughout the systems lifetime, urging the need for a suitable and adaptable security strategy. Assessing the trustworthiness of the supply chain and the resulting design elements, even if the result of this analysis is inherently fuzzy, is an important step to improve the security of the design. This thesis discusses the adaptation of system-centric threat modeling to the FPGA domain. The systematic analysis of the design, based on the STRIDE concept, provides valuable insight into the design threats and the required counter mechanisms. The process speeds up the analysis and reduces the risk of missed threats and misunderstandings by relying on prepared building blocks structures. The most common access control paradigms can model access control rules in FPGA designs. Choosing the appropriate design paradigm depends on the complexity and security requirements of the design. For a simple, pipeline based design, the Bell-LaPadula or Biba paradigm might be appropriate, while a more complex design requires an access control list or matrix to create an enforceable rule. The formal definition of the FPGA architecture, the temporal relationship between the configurations and the security rules, promotes a precise definition of the assets and their interaction, removes ambiguity from the threat model, provides a blueprint for the implementation and allows an automatized validation of the security rules. The domain-specific language supports standard access control paradigms like Access Control Lists, RBAC, or Attribute-Based Access Control.

The formal description of the FPGA architecture (FPGAModules, communication networks, and reconfiguration process) and the security policy promotes a precise definition of the assets and their possible, allowed, and prohibited interactions. It removes ambiguity from the threat model while providing a blueprint for the implementation. It also allows the rigorous validation of the security policy, as described in chapter 8.

Model-checking can be applied to verify if, and to a certain degree when, a design complies with the stated security policy. Transferring the architecture into a suitable model and the security policy into verifiable logic

properties can be, as demonstrated in this thesis, automated, mitigating one source of error. The successful validation of the model can be counteracted by wrong assumptions about the effectiveness of security mechanisms and the wrong assignment of the security attributes, attacks against lower levels of the design, or a faulty implementation. The challenge of determining and assigning appropriate security attributes and their acceptable and unacceptable interactions remains a task best left to experienced humans.

Reinforcement learning can identify potential weaknesses and the steps an attacker may take to exploit them. To achieve this goal, an FPGASECML Model can be transformed into an MDP model, solvable with standard algorithms like q-learning. Adding more state information and actions to it improves the expressiveness of the MDP models, but the computational costs may exceed the value gained. It is possible to extend the generated code to use more advanced techniques like Options, Partially Observable Markov Decision Processes, and stochastic games. Further advances in the field of reinforcement learning, perhaps in combination with Deep Learning, could lead to the creation of an entirely automated penetration tester, able to find and exploit weaknesses in real-life systems.

FPGAs share some threats and weaknesses with conventional IT-system and knowledge about the successful and failed attempts in the vastly more significant software domain can create better tools and techniques for them. However, FPGA also provides the opportunity to create innovative security solutions by leveraging the dynamic change of their configuration to limit their exposure to threats and to lead attackers astray. Finally, some of the methods developed for this thesis may be useful beyond the FPGA world as the attribute-based access control approach to security policies that could be used for complicated Internet of things (IoT) configurations or the reinforcement learning-based attacker model that may be generalized to test a broader range of IT-systems.

Appendix A Glossary

ABAC	Attribute Based Access Control
ACM	Access Control Matrix
AES	Advanced Encryption Standard
AMBA	Advanced Microcontroller Bus Architecture
BRAM	BlockRAM - volatile memory component of an FPGA
CAN	Controller Area Network
CC	Configuration Control and Storage
CIA	Mnemonic for confidentiality, integrity and availability
CM	Configuration Management
CRC	Cyclic Redundancy Check
CSE	Configuration Storage Elements
CTL	Computation Tree Logic
CLB	Configurable Logic Block
DAC	Discretionary Access Control
DMA	Direct Memory Access
DRAM	Distributed RAM – CLBs configured as memory elements
DoS	Denial-of-service attack
ECB	Electronic Code Book – unsecure mode of operation for a block cipher
FPGA	Field Programmable Gate Arrays
GSM	Global System for Mobile Communications
GLM	Generalized linear model
IP	Intellectual Property
JTAG	Joint Test Action Group
LOC	Lines of Code
LTL	Linear Temporal Logic
MDP	Markov Decision Process
MAC	Mandatory Access Control

NoC	Network on a chip
NSA	National Security Agency
POMDP	Partially Observable MDP
PCB	Printed Circuit Board
PLC	Programmable Logic Controller
PRR	Partial Runtime Reconfiguration
PUF	Physical Unclonable Functions
RIPE-MD160	RACE Integrity Primitives Evaluation Message Digest
RISC	Reduced Instruction Set Computer
SCADA	Supervisory control and data acquisition
SHA	Secure Hash Algorithm
SMD	Surface-mounted device
SoC	System-on-a-Chip
SRAM	Static random-access memory
STRIDE	Mnemonic for Spoofing of user identity, Tampering, Repudiation, Information disclosure, Denial of service (D.o.S), Elevation of privilege
TCB	Trusted Computing Base
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
WORM	Write once read many
RNG	Random Number Generator

Appendix B The low-level threat model for FPGAs

This section presents a systematic security analysis of low-level components inside the FPGA and its periphery. Further, more abstract, security analysis of every FPGA based design can use the analysis of these low-level components as a foundation.

B.1 Peripheral Devices, the FPGAs environment

This subsection introduces the building blocks to threat model the peripheral devices connected to the FPGA. First, the security concerns for peripheral devices, in general, are discussed, followed by a more detailed nuanced analysis of the different classes of peripheral devices. Where a peripheral device combines the function of multiple classes, it is recommended to threat model the peripheral device for each of these classes.

B.1.1 Peripheral Devices

Peripheral devices are all devices directly connected to the FPGA. An attacker may try to:

- Replace the device
- Bypass the device
- Alter the behavior of the device
- Attack the communication between the device and the FPGA

General security mechanisms for all peripheral devices include:

- Physical protection against attacks (e.g., through enclosure or glue)
- Trustworthy development process and developers
- Trustworthy supply chain from the developer over manufacturing to the shipment of the final device (Counterfeited microchips pose a severe risk [174, 175] that must be mitigated appropriately.)
- Authentication of the identity of the device and data transmitted or received
- Encryption of any sensitive data stored and transmitted
- Reliable removal of sensitive data (as soon as possible)
- Identification of its current security state (e.g., ok, under attacker, tampered with)

Peripheral devices classes are auxiliary devices, memory elements, communications interfaces, and processing devices. A single device can also combine the attributes of several of these generic classes.

B.1.2 Auxiliary Devices

Those devices necessary for a stable operation of the FPGA but without any further functionally can be subsumed into a class of auxiliary devices:

- **Power Supply**: providing a stable power supply for modern FPGAs is a non-trivial task. Different voltage rates are required, and multiple IO interfaces have different power requirements. Gaining control over the power supply (either of the overall system, the device, or the FPGA) enables an attacker to perform a denial of service attack or a side-channel attack. Complex power management systems, providing services like energy save modes, increase the attack surface. They must, therefore, be treated with the appropriate caution.

- **Oscillator**: provide the clock for the FPGAs operation. Speeding up the oscillator can improve the likelihood of a glitch and can lead to a temporary or persistent DoS attack (e.g., through thermal overload). Reducing the frequency of the oscillator, on the other hand, can ease reverse engineering and side-channel attacks (oscilloscope sampling rate vs. device frequency)

These devices have, if they perform their task as specified, only a marginal influence on the systems behavior and attacks against them mainly support or enable other low-level attacks like timing or power analysis, deliberately omitted from the formal model introduced in chapter 7.

B.1.3 External Memory

Almost all non-trivial tasks require external memory chips to store data temporarily or persistent. The STRIDE approach leads to these main threats against and from memory devices:

- **Spoofing identity:** By replacing or bypassing the device of the memory, an attacker can inject malicious code or data into the system.
- **Tampering with data**: Altering the data that the data processes processed (this includes code to be processed by a CPU) changes the behavior of the system.
- **Repudiation**: Data on the device (e.g., a log file) could be deleted to hide an attack.
- **Information disclosure**: Information, otherwise protected by design, could be retrieved from the memory chip itself.
- **Denial of service**: Denying access to the memory cripples or disables all services and functionality relying on this data.
- **Elevation of privilege**: An attacker can bypass existing security mechanisms and perform actions denied to a legitimate actor by tampering with the memory either directly or via the access control mechanism. This threat is not limited to the content of the memory itself as it could include, for example. The addition of illegal entries to an access control database, for example, grants an attacker further control over the system.

These attributes are of particular importance from a security perspective:

- **Volatility**: A reboot disrupts any attack tampering with data in a volatile memory area. Tampering with persistent data like a flash memory creates a more persistent threat. Detecting (nonvolatile) attacks against flash memories, on the other hand, is more straightforward than detecting attacks against volatile memory, where a simple power off removes any traces of the attack [176]. The secure removal of sensitive data, for example, during the decommissioning process, represents another threat limited to nonvolatile memory only.
- **User access**: Standard, user-replaceable memory elements like USB Sticks or SD Card can contain configuration files and other persistent and sensitive data. Their removability makes these memory devices easy to use points of entry. User removable SDRAMs can be used to perform a cold boot attack [177] by removing them from their socket and reading their content in another device, an attack unlikely to succeed with soldered memory elements.
- **Cryptographic Services:** Some memory devices provide cryptographic services like encryption or authentication. The sensitive nature of these services requires a careful investigation into the trustworthiness of their implementation and their correct configuration and integration.
- **Data flow**: An SD card used to store data of low sensitivity like a log file (maybe even in write-only mode) poses a significantly lower risk than a memory device storing the FPGA configuration or any other data to be processed by the FPGA.

- **Content**: The type of data stored is of high importance. A single device may hold data of different levels of sensitivity, like the FPGA configuration and logfiles at the same time. Where the memory element contains data of different levels of sensitivity, each of these data sets should be considered individually, e.g., by creating a table of the memory elements content and their respective sensitivity.

B.1.4 Communication Interface

Communication Interfaces provide gateways to and from the various threat layers. It is, therefore, vital to understand the corresponding endpoints and their location within these layers. A Gigabit Ethernet PHY chip, in combination with an FPGA IP-Core, can provide access to the internet or the backplane of the system. The location of the endpoint(s) determines the exposure of the system to threat agents.

The main threats against and from communication interface are:

- **Spoofing the identity of the communication interface:** an attacker can circumvent the security mechanisms implemented in the interface and perform further men-in-the-middle attacks by replacing the device itself.
- **Spoofing the identity of the communication partner:** an attacker can access the data sent from the FPGA and inject data into the FPGA.
- **Tampering with data or the communication interface:** An attacker can influence the behavior of the receiver by altering data transmitted through the interface.
- **Repudiation: D**eleting log files and buffers on the device can hide an ongoing or past attack.
- **Information disclosure:** Information sent to the FPGA can be exposed either through an attack against the element or a man-in-the-middle attack during the data transfer. An attacker gaining control over the FPGA configuration can reveal sensitive information to unauthorized actors via the communication interface.
- **Denial of service:** Jamming the communication via the interface can make the FPGA and its communication partner unavailable for other tasks. A threat agent can perform such an attack at any communication layer.
- **Elevation of privilege:** By gaining control over the device or by replacing it with a device under its control, an attacker can bypass existing security mechanisms and perform actions otherwise denied to him.

These security-relevant attributes of communication network require particular attention:

- **Data:** how sensitive is the data passed through this interface?
- **Trust Boundaries:** does data to and from the peripheral device cross trust boundaries?
- **IO Sanitization:** is the data entering or leaving the interface validated?
- **Authentication:** is it possible to authenticate the other endpoints and the transmitted data?
- **Encryption:** is data encryption demanded and implemented?

B.1.4.1 Processing Devices

Processing Devices like microcontrollers or DSPs perform computations, transfer data, and control information to the FPGA or receive them from it. These threats can be identified using the STRIDE approach:

- **Spoofing identity:** of the processing device, an attacker can gather information (sensitive data, reverse engineering of the control flow) and inject malicious data into the FPGA or extract sensitive data exchanged between the peripheral device and the FPGA.
- **Tampering with the device:** An attacker can tamper with either the FPGA or the processing device by tampering with the data exchanged via this device.
- **Information disclosure:** A glitch or attack against the processing device, as well as a man-in-the-middle attack can expose sensitive data.
- **Denial of service:** A DoS attack against processing devices providing essential, security-sensitive services could either render the whole system useless or force it to forgo the services provided by them. The latter case could leave the FPGA without the mandated security mechanisms.
- **Elevation of privilege:** Gaining unauthorized access to the processing devices, an attacker can bypass existing security mechanisms and perform actions denied to a legitimate actor.

From a security perspective, the following attributes are essential:

- **Provided services**: It is essential to identify the security-sensitive services rendered to the system and FPGA. When services provided by the processing devices are crucial to the FPGAs security (direct or indirect), the processing devices must provide these services as required and show the mandated resilience against attacks. Such services include cryptographic operations, input sanitization, authentication, or monitoring tasks.
- **Connectivity:** a processing device that is collaborating with other elements of the design can provide a connection across trust boundaries.
- **Flexibility**: altering the behavior of the device can endanger the security of the FPGA (and other devices connected to it). It is, therefore, reasonable to consider processing devices with hardwired behavior as more resilient against outside threats and less patchable than software-defined processing devices.
- **Complexity/attack surface:** it is reasonable to assume that the risk of security-related bugs increases with the complexity of the system. The analyst should, therefore, treat sophisticated processing devices entrusted with sensitive tasks with more caution than simpler processing devices performing the same tasks
- **Security mechanisms**: the device may have one or more security mechanisms implemented in the hard- or the software, to fend off attacks.

B.2 FPGA Configuration - Storage and Management

The configuration of the FPGA defines its behavior and is, therefore, among the most sensitive components of every FPGA based design. The system must protect all elements entrusted with the configurations storage accordingly. All elements outside the FPGA are the periphery. The programming path from the configuration to the FPGA can run through one or more peripheral devices, be hidden inside the very FPGA package (if there is an internal nonvolatile memory like in the Spartan 3AN), or a combination of both. The security level provided by the multiple programming paths may vary.

B.2.1 FPGA Hardware

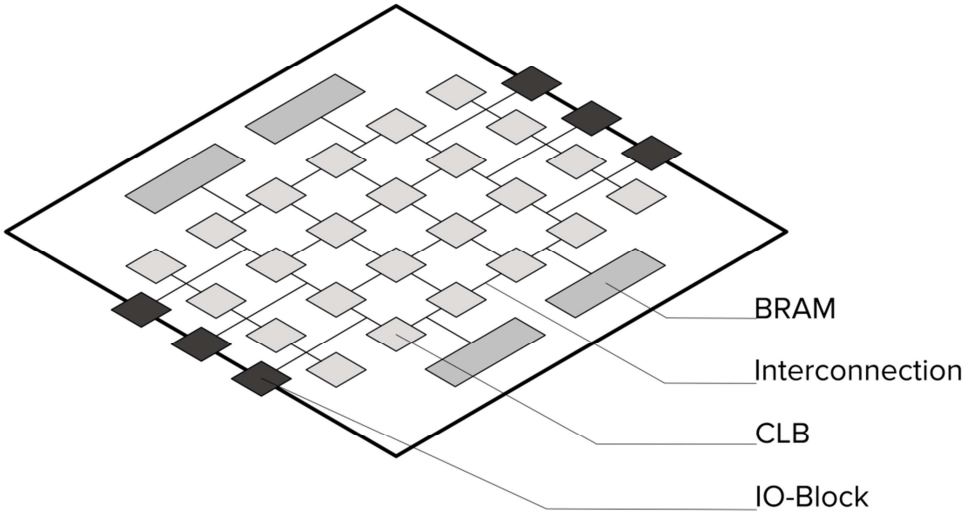


Figure 40 FPGA schematic with typical components

The FPGA (Figure 40) package protects the elements discussed in this section, and a direct attack against them will, therefore, be expensive. Attacking them indirectly via the configuration process is more feasible. This analysis is, therefore, performed to understand the implications of maliciously crafted or altered configurations.

B.2.2 Configurable Logic Blocks

Modern FPGAs can contain logic circuits with the equivalent complexity of several million gates [178]. This user-defined computation is performed primarily by a grid of Configurable Logic Blocks (CLB). These are, in general, lookup-tables with additional functionality to communicate with other elements. Their application is not limited to binary logical operations as they can serve as shift registers, distributed ram, and connectors between different parts of the design (e.g., in designs with Partial Runtime Reconfiguration) as well.

- **Spoofing the identity** of CLBs could be performed by micro probing the FPGA fabric, but due to the size of the elements, such an attack appears very unlikely.
- **Tampering with data:** changing the configuration of a CLB changes the behavior of the element or the data stored in the CLB.
- **Repudiation:** An attacker could destroy the evidence of the security breach by wiping the configuration of the element or by restoring its original state. If the compromised CLBs perform logging or monitoring purposes, traces of malicious transactions may be suppressed or removed.
- **Information disclosure:** Retrieving the CLB configuration can provide the attacker either with their logical function or the memory content stored within them (distributed memory). Getting access to the content of all CLBs and the programmable interconnections discloses the configuration.
- **Denial of service:** Alterations of the CLB can disrupt the operation they participate in and even lead to their physical destruction [179]. These changes render the FPGA, at least in parts, temporarily or permanently useless.

- **Elevation of privilege**: Tampering with CLBs used for security services (e.g., secure configuration or input sanitization) can disable these security mechanisms, enabling the attacker to perform actions otherwise denied.

B.2.3 IO-Blocks

IO-Blocks provide the physical connection to the outside world. Their electrical characteristic is configurable to match the physical requirements of the peripheral device. They are, by their pins, exposed to the outside world. Physical tampering with IO Blocks requires less effort than with other FPGA components due to this exposure to the PCB threatlayer.

- **Spoofing the identity of the peripheral device:** An attacker can inject malicious data into the FPGA by taking the place of a peripheral device connected to it.
- **Tampering with data:** An attacker can change or suppress data to and from the peripheral device.
- **Information disclosure:** Taping the IO Pin, (otherwise unprotected) communication between the FPGA and its periphery. IO pins can be targeted form inside the FPGA or at the PCB level.
- **Denial of service**: A deliberate misconfiguration of the IO Block can lead to the physical destruction of either the IO Pins of the FPGA, the periphery, or both. An attacker can also try to destroy the surrounding CLBs and Interconnections by applying external power to the IO Pin (similar to[180].)
- **Elevation of privilege:** An attacker can perform actions denied to legitimate users through a man-in-the-middle attack between FPGA and periphery

B.2.4 Embedded Hard Blocks

Embedded Hard Blocks are integrated into FPGAs to perform often required tasks, like multiplications, more efficiently than a block of CLBs. Their behavior is, to a certain degree, reconfigurable, but they are not as versatile as CLBs. FPGAs can include a variety of embedded hard blocks like:

- **Embedded Processors** like the PowerPC, ARM or other RISC derivate
- **Communication Interfaces,** e.g., PCI Express and Gigabit Ethernet
- **Multiply/Add** and **DSP-Blocks** to speed up mathematical operations
- **BlockRAM** to store small to medium amounts of data inside the FPGA

Embedded processors can be, under security aspects, treated as processing devices in the last section and communication interfaces like the peripheral devices. Embedded inside the FPGA, they provide a higher resilience against physical manipulation as an attacker must gain access to the FPGA itself (while other peripheral devices are more exposed to attackers.) Multiply/Add, and DSP-Blocks provide lesser flexibility as CLBs and represent no higher risk from a security perspective. It is, therefore, reasonable to treat them in the same way as CLBs.

B.2.5 BlockRam

BlockRam is used to store medium-sized amounts of data inside the FPGA. Applying the STRIDE mnemonic leads to these threats:

- **Tampering with data of the block ram:** An attacker can influence the behavior of the FPGA by altering the data stored in the BlockRam. Possible attacks here include the injection of shellcode

for an embedded processor or the zeroing of a cryptographic key to render the encryption mechanism useless.

- **Repudiation:** An attacker can remove traces of an attack or a legitimate transaction through changes in the BRAMs content.
- **Information disclosure**: An attacker with read-access to the BRAM can reveal sensitive data, like cryptographic keys or program code, otherwise protected through the design.
- **Denial of service**: An attacker can inject maliciously crafted content into the block ram to disable parts of the configurations by, e.g., injecting malicious code in an embedded processor. - a simple endless loop can be sufficient.
- **Elevation of privilege:** An attacker can bypass the existing security mechanism, like access control, and perform actions denied to the legitimate user by directly altering the content of the BRAM.

B.2.6 Auxiliary Blocks

Auxiliary Blocks are those elements that are necessary for a stable operation of the FPGA but with minimal influence over the systems services and functions (if they operate within their specification.)

Clock management

FPGA requires a stable clock with a frequency appropriate to their configuration. Clock management elements inside the FPGA fulfill this task, often utilizing external oscillators.

- **Spoofing identity:** Taking the place of a legitimate user of the clock management (e.g., a processor or a dedicated monitoring element), an attacker may lay the groundwork for further attacks.
- **Tampering with the configuration:** An attacker can change the behavior of the circuit and enable further attacks by tampering with the clock management.
- **Repudiation:** An attacker can try to avoid detection by a monitoring system through a change in clock management.
- **Information disclosure**: speeding the clock up or slowing it down can leak information about the system to the outside world. The results of such manipulation include glitches occurring at a higher frequency or side-channel attacks enabled by lower frequencies.
- **Denial of service:** Tampering with the system clock can, as mentioned above, lead to a malfunction of the system.
- **Elevation of privilege**: Glitches occurring at clock frequencies outside the specified range may enable an elevation of privilege attack, like the access of memory segments protected under normal circumstances.

Debugging interface

It is the nature of debugging interfaces that they provide access to the most sensitive points of the system. An interface like JTAG (Joint Test Action Group) can be used not only to get information from the system under test but to alter their configuration as well as data stored within them. An attacker gaining access to the debug interface poses, therefore, a grave threat:

- **Spoofing identity** (of a legitimate actor, e.g., developer or maintenance expert): An attacker can impersonate a legitimate actor (e.g., developer or maintenance expert) and abuse the debug interface to gain further insight into or control over the system

- **Tampering with data:** An attacker can alter internal data directly through the debug port
- **Tampering with the FPGAs configuration or software:** An attacker can change the behavior of the FPGA, for example, change sensitive data through malicious alterations to the configuration or software.
- **Repudiation:** Using the debug port, an attacker can remove traces of an attack or a legitimate transaction.
- **Information disclosure:** The debugging port can expose sensitive information (data, status, and configuration) to unauthorized actors with access.
- **Denial of service:** injecting maliciously crafted bitstreams or data into the FPGA can lead to either or partial or complete unavailability of services. Reconfiguring or disabling the programming path via the debugging interface is another threat to the availability of the FPGAs services.

B.2.7 Programmable Interconnections

Programmable interconnections within the FPGA provide the flexible communication infrastructure between the components described above. The majority of the FPGA fabric consists of these interconnections [181]. STRIDE can be used to identify these generic threats:

- **Spoofing Identify:** An attacker can spoof the identity of an internal or external (connected via IO Pins) communication partner by rerouting the FPGA network.
- **Tampering with the interconnection configuration:** A malicious actor can alter the source and destination of a connection by changing the interconnections configuration.
- **Repudiation:** An attacker can remove traces of an attack or a legitimate transaction by bypassing distinct CLBs and thereby the service they provide (logging, monitoring, or a watchdog service).
- **Information disclosure:** Rerouting interconnects can expose sensitive information to either an unauthorized element inside or outside (via an IO Block) the FPGA.
- **Denial of service**: A deliberate misconfiguration of the FPGA's internal communication network interrupts the tasks relying on the affected interconnects. This interruption can be either temporary or permanent.

Elevation of privilege: An attacker gains further control over the system and performs actions denied to legitimate users by rerouting the interconnections to an element (internal or external) under his control

Appendix C High-level security analysis of intricate FPGA designs

This section builds upon the security analysis presented in the previous section and introduces the tools for a more abstract threat analysis of the system (Figure 41). This analysis focuses on the various programmable logic modules that are using the FPGAs primitives to perform their computations. This approach is more suitable for the system-centric threat analysis of complex systems. The model creator can use instances of these primary components:

- **FPGAModules are** representing cohesive blocks of the reconfigurable logic accessible through one or more interfaces and (in designs that use runtime reconfiguration) interchangeable within specified limits. FPGAModules come in two forms:
    - Processing Blocks: operating within the FPGA
    - IO Blocks: Processing Blocks with access to the periphery
- **Communication Networks**: connecting the FPGAModules
- **Configuration Control and Storage** (CC): representing those crucial components entrusted with the configuration.

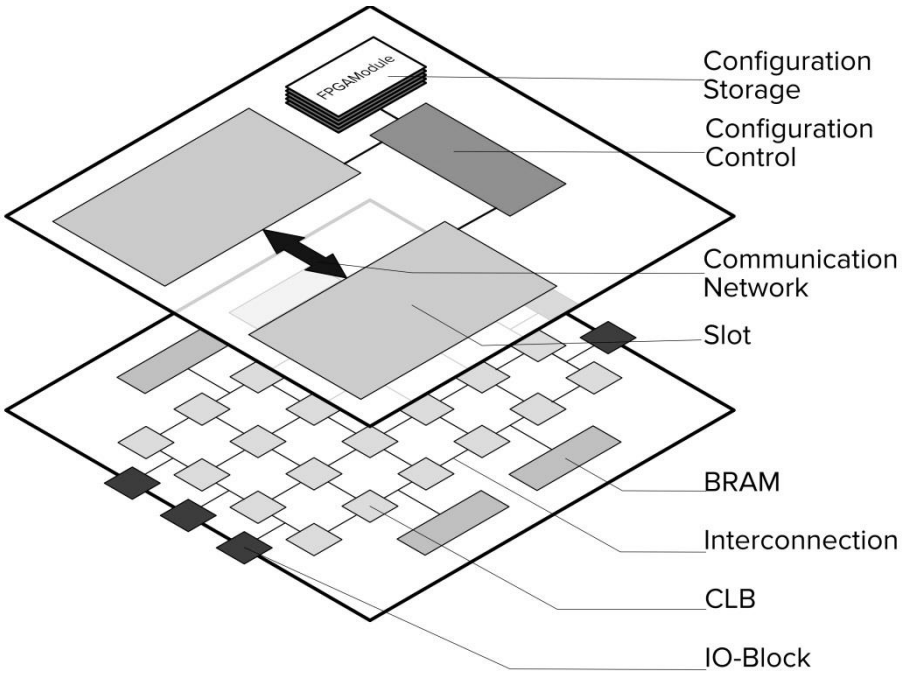


Figure 41 FPGA primitives and their abstract representation

C.1 Configuration Control and Storage (CC)

The Configuration Control and Storage (CC) subsystem is responsible for the complete or partial reconfiguration of the FPGA. CC receives a (re)configuration command and loads either a complete or partial configuration from a defined programming path (e.g., Flash, JTAG) into the FPGA. The storage elements holding the configurations are part of this component as well as the upgrade mechanisms and security mechanisms like encryption, authentication deployed to protect the configuration. Integrating all elements that participate in the reconfiguration process into a single component simplifies the model, but it

is also possible to create a distributed representation of this component. This distributed representation stores the storage and security mechanism information within each FPGAModule description.

C.1.1 Implementation

The configuration storage can be a part of the FPGA package (e.g., Xilinx Spartan3AN), inside the FPGA (flash ram based FPGAs) or - in most designs- outside the FPGA in an external nonvolatile storage element like a flash memory chip.

The configuration control can be a part of the FPGA configuration, software running on top of an embedded processor (hard block or IP-core), or an external device controlling the FPGA. The role of the FPGA can, therefore, be either:

1) **Master**: the FPGA controls the configuration process
2) **Slave**: an external device, e.g., a JTAG interface or a microcontroller, controls the configuration process of the FPGA

Multiple entities at different stages of the systems lifecycle may perform Configuration Control functionality. A system may utilize a JTAG port at the development stage, a simple hardware connection during regular operation, and an external microcontroller, as part of a maintenance kit, for upgrades and diagnoses. A threat model may omit some of these options for the sake of simplicity, as most threat models are created to avoid security problems of the system deployed in the field.

C.1.2 Threats

Tampering with the CC, an attacker could load configurations that are not trustworthy, bypass built-in security mechanisms (like encryption and authentication), or forces the (either partial or complete) reconfiguration into a less secure state.

Using the STRIDE approach these threats against the configuration storage: can be identified

- **Spoofing the identity of the configuration storage:** An attacker can inject a malicious configuration into the system by replacing the storage device.
- **Tampering with data of the configuration storages:** An attacker can alter the behavior of the FPGA by tampering with the configuration.
- **Repudiation:** An attacker can hide an attack by removing his maliciously crafted configuration from the storage or by restoring the original state of the configuration after an attack.
- **Information disclosure of the configuration storage:** Reading the content of the configuration storage reveals this sensitive content to the attacker.
- **Denial of service:** Making the configuration storage unavailable prevents the FPGA from (re-)configuration, rendering it useless or leaving it in a vulnerable state**.**
- **Elevation of privilege:** An attacker can perform actions denied to the regular user, by replacing or altering an existing configuration.

Threats against the configuration management include:

- **Spoofing the identity of a configuration management component:** An attacker can inject a configuration of his choice in the FPGA, bypass security mechanisms like bitstream authentication and encryption and reveal the configuration files to unauthorized actors.

- **Tampering with the configuration management:** An attacker can force the FPGA into a reconfiguration, either to load a maliciously crafted configuration into the FPGA or to exploit side effects of insecure reconfigurations (see 4.5)
- **Repudiation:** Clearing log files or restoring the original configuration of the CM after an attack allows a hacker to cloak his activities.
- **Information disclosure:** an attacker can learn which configurations are available, the storage and usage of the data by gaining access to the configuration management.
- **Denial of service attack:** A successful DoS attack prevents the FPGA from (re-)configuration, rendering it useless or leaving it in a vulnerable state**.**
- **Elevation of privilege: a**n attacker can perform actions denied to the regular user by forcing the (re)configuration of the FPGA with a configuration of his choice.

C.1.3 Threat Agents

Using the taxonomy introduced earlier, these threat agents and their activities can be identified:

- **Software Hackers:** attacking the software in control of the configuration
- **Hardware tinkerer:** tampering with the hardware along the programming path.
- **Digital Design Experts:** tampering with the configuration inside their storage element or the FPGA
- **Microsystems Engineer** retrieving the configuration from the configuration storage for further analysis

C.1.4 Security Mechanisms

Security Mechanisms can protect the configuration of the FPGA from theft, unauthorized changes, and malicious replacement:

Security Property	Mechanism	
Confidentiality	Symmetric or asymmetric encryption of the configuration	Tamper Resistant Storage
Integrity	Message Authentication Code	
Availability	Redundancy Safety margins to handle the excessive traffic	

Many FPGAs provide the necessary support for the cryptographic protocols, for example [182]

C.2 Processing Blocks

Processing Blocks (PB) represent a distinct configuration of low-level FPGA primitives. These blocks perform distinct tasks together, are utilized at the same time, and are tightly connected. The utilization of

FPGA resources can be either static or dynamic. Multiple dynamic PBs can share the same FPGA resource over time.

C.2.1 Implementation

Processing Blocks consists of a distinct set of Configurable Logic Blocks, Embedded Hard Blocks and the Interconnections between them. Those parts of the PB serving as interfaces expose it to threats by other FPGAModules and deserve special attention.

C.2.2 Threats

An attacker who controls the PB can receive data intended for other design elements, replace functionality, or disrupt the operation of the overall system by sending maliciously crafted messages. The threats posed by a single Processing Block strongly depend on its role and position within the configuration, e.g., the sensitivity of the services it provides and its connectivity to other elements.

C.2.3 Threat Agents

- **Software Hackers** attacking –where available- software running on top of the configuration
- **Digital Design Experts are** tampering with the configuration of the Processing Blocks.
- **Microsystems Engineer** probing or manipulating the content of the PB (an expensive process, unlikely to succeed unless previous knowledge indicates a valuable target)

C.2.4 Security Mechanisms

Additional security mechanisms should protect the Processing Block and the elements that interact with him where the architecture cannot provide the required level of security. Generic security mechanisms to protect a Processing Block and the data entrusted to it include:

Security Property	Mechanism
Confidentiality	Symmetric or asymmetric encryption of the data Obfuscation of sensitive data stored inside the FPGA
Integrity	Message Authentication Code of data received and sent Sanitation of the used FPGAResources Validation of received data
Availability	Redundancy Safety margins to handle the excessive traffic DoS-detection and filtering Economic message handling

C.3 IO Blocks

An IO Block (IOB) provides access to the periphery outside the FPGA. They can be gateways to networks (local or global), storage elements (volatile and nonvolatile), or provide additional processing services implemented in external devices (e.g., to a microcontroller). IO Blocks are, therefore, Processing Blocks with additional access to services and data from beyond one or more trust boundaries.

C.3.1 Implementation

IO Blocks utilize the same resources as Processing Block plus the low-level IO component (the low-level IO-Blocks or other, dedicated hard blocks), providing access to the periphery.

C.3.2 Threats

Peripheral Connectors are natural locations for trust boundaries as they are gateways between the FPGA and the outside world. An attacker can perform, e.g., a DoS attack or an escalation of privileges attack by gaining access to their periphery.

C.3.3 Threat Agents

- **Digital Design Experts are** analyzing and tampering with the FPGA configuration of the IO Block.
- **Microsystems Engineer** probing the internals of the Peripheral Device and the FPGA
- **Hardware tinkerer** tampering with the external peripheral device, probing the IO pins
- **Software Hacker** attacking software running on the peripheral device

C.3.4 Security Mechanisms

IO Blocks have a connection to the outside world, and the threat agents associate with it and therefore require an additional set of security mechanisms that operate between the FPGA and its periphery. These security mechanisms include:

Security Property	Mechanism
Confidentiality	Symmetric or asymmetric encryption of the data
Integrity	Message Authentication Code of data received and sent Sanitation of the used FPGAResources Validation of received data
Availability	As identified for Processing Blocks plus: Overvoltage protection Physical protection of the communication

C.4 Communication networks

The communication networks (CN) connect the different FPGAModules. CNs can vastly vary in complexity and functionality. A simple CN can consist of bus macros [183] while complex CNs implement a bus or network-on-a-chip design paradigm. Additional CNs can either simplify the task, to meet non-functional requirements like multiple quality of service levels, or to improve the security of the design through compartmentation. Different CNs can be connected explicitly through gateways or implicitly

through common FPGAModules. Networks across trust boundaries have to be treated with the appropriate care to avoid threat agents or security breaches on one side tampering with the security of the other side.

Integration tools and code generators, like Xilinx's EDK [184], are often utilized to create CNs. This code generation is not limited to the very communication network but building blocks for bus access and control. The use of compromised or flawed tools for this task could endanger the overall security of the design.

Participants to the CN can be classified in masters able to control other participants of the CN, slaves controlled by others and peers who have neither control over other participants, nor are they controlled by them. This discrimination seems to be less critical for a security analysis as a compromised "slave" component might not comply with the mandated separation of control.

C.4.1 Implementation

Communication Networks are, in general, formed by a combination of programmable interconnects and Configurable Logic Blocks that may be supplemented by hard-blocks like BRAMs.

C.4.2 Threats

An attacker could manipulate the CN to redirect the communication, bypass security mechanisms (like firewalls) and services (like encryption), and either intercept, redirect or alter the transmitted data. This manipulation must not be restricted to the very design under inspection but can be carried out by compromising the tools used to create the CN.

C.4.3 Threat Agents

- **Software Hackers** hiding malicious functionality within the code generators or integration tools utilized to generate the CN
- **Digital Design Experts** are tampering with the Communication Network configuration.
- **Microsystems Engineer** probing communication through the Communication Networks

Appendix D FPGASECML grammar and validation

This appendix provides a short overview of FPGASECML grammar.

D.1 Xtext syntax 101

The parser of the DSL uses XText, and the formal definition presented in this appendix follows this syntax. [185] presents a short tutorial (it takes about 15 minutes) for Xtext. This section provides the most important grammatical rules to make the following walkthrough more accessible:

- **'abc'** String
- **a?** Optional element a
- **a+** One or more a's
- **a*** Zero or more a's
- **(a|b|c)** Either subrule a,b or c must be satisfied

D.1.1 Rules

Each grammar rule follows the pattern:

```
rulename:
      RULES
      ;
```

D.1.2 References

Variable assignments define how to access the parsed content in the EMF classes:

```
var=rule
```

This statement declares that a single object (representing the content of 'rule' to rule is assigned to the variable var

```
var+=rule* or var+=rule+
```

These statements declare that the variable var represents a list of objects (each representing the content of one 'rule' statement).

```
(var?=rule|'terminal')
```

This statement will instruct Xtext to create a boolean variable that is false if the parser finds the string 'terminal' at this part of the grammar.

D.1.3 ID

The token ID is used for any alphanumerical string and predefined by Xtext.

D.1.4 Name

```
var=[rulename]
```

The variable name is reserved for unique identifiers of an object. Any Xtext rule in the metamodel can reference to this object using square brackets:

```
IOBlock :
      'IO Block:' name=ID
      'utilizes slot:'  slotid=[Slot] ';'
```

This statement defines that each IO Block must have a unique alphanumeric identifier after the declaration 'IO Block' and that a valid statement must reference to a well-defined (FPGA) Slot in this model. It is not necessary to declare the Slot before IO Block. Xtext automatically generates the corresponding validation and scoping code.

D.1.5 Qualified Name

Elements declared within another element must be addressed with their qualified names. FPGASEC uses the common dot notation and therefore defines qualified names as:

```
QualifiedName:
  ID ('.' ID)*
;
```

The following term, for example, refers to a model element mem1, defined within the element Slot1:

```
Slot1.mem1
```

D.1.6 Enum

Enums are terminal nodes whose value can take one of the multiple strings. They are easy to validate. The values for these strings are either identical with the given rule or match an explicitly assigned string.

The security_featureENUM, for example, can be either: "encryption," „authentication," "random," or "rand":

```
enum security_featureENUM
      :
      encryption | authentication | randomization=„random"
      |randomization=„rand."
      ;
```

D.2 Architecture

FPGASECML formalizes the analysis presented earlier in this thesis - its architecture descriptions reflect the threat modeling blocks introduced in Appendix C, and the security policy implements the ABAC paradigm discussed in 5.5.5. Each valid security model contains both FPGA architecture and security policy following this scheme:

```
Model:
      'FPGAArchitecture' name=ID
      'FPGAModules:'
            fpgaModules+=FPGAModule+
      'FPGAResources:'
            fpga_primitives+=slot+
      ('Periphery:'
            periphery+=peripheral_device+)?
      'Communication:'
            communication_infrastructure+=communication*
      'Reconfiguration:'
            reconfiguration=reconfiguration
            rsequences2validate+=recon_sequence*
      ('Performs tasks:'
            tasks+=task*)?
      'Security Policy:'
            security_policy+=security_rule*
;
```

D.2.1 FPGAModules – IO Elements and Processing Blocks

An FPGAModule can come in one of two manifestations:

```
FPGAModule:
      IO Block | Processing Block
;
```

Processing Blocks represent a distinct set of user-defined logic that performs a defined set of operations using the preassigned FPGA resources. IO Blocks share all attributes of Processing blocks and add some periphery specific information to it. Xtext creates three classes out of this rule::

- FPGAModule that contains all attributes common to IO Blocks and Processing Blocks
- IO Block that inherits from FPGAModule and contains the periphery specific information
- Processing Block that also inherits from FPGAModule but does not contain any further information.

D.2.1.1 Security attributes

Security attributes include the systems assets, security mechanisms, and information about the storage location and defense mechanism of the configuration. These attributes include the security-sensitive services provided and the sensitive data processed or stored by them, the security mechanisms to protect its operation as well as those deployed to protect its bitstream (the model calls the binary FPGA configuration

bitstream, following Xilinx terminology) and finally the bitstreams storage element itself. IO Blocks and Processing Blocks share the same set of primary security attributes presented here; the IO Block specific security attributes are presented later in this section.

D.2.1.2 Sensitive services

The description of each FPGAModule should contain a list of the sensitive services qualified by their security (sensitivity) level.

```
sensitive_service:
      'none'
      | ('sensitive service'serviceid=ID 'of sensitivity level.'
          securityLevel=security_levelENUM';')
      ;

enum security_levelENUM:
      low|medium|high
      ;
```

For most applications, the granularity provided by the default levels (low, medium, and high) is sufficient, but additional levels may be to the security_levelENUM when necessary.

D.2.1.3 Sensitive data

```
sensitive_data:
'sensitive data' dataID=ID 'of sensitivity level'
            securityLevel=security_levelENUM
            ('access type' accessType=dataAccesstypeENUM)
            ('stored in memory' mem+=[Slot_memory | QualifiedName])?
            ';'
;

enum dataAccesstypeENUM:
      read|write|rw|internal
;
```

The sensitive data rule allows the enumeration of security-relevant data, the access privileges exercised by the FPGAModule, and, optional, the Slot memory element used to store it. To model direct memory access (DMA), the data storage information can include memory from other Slots.

(a) Security mechanism

```
enum security_mechanismENUM:
      input_validation | data_encryption | data_authentication
      | resource_sanitization | data_sanitization
;
```

An element may (or should) implement specific security mechanisms to improve its resilience against attacks. Supported mechanisms are:

- **input_validation**: the FPGAModule shall validate all received data
- **data_encryption**: the FPGAModule shall be encrypted all sent data
- **data_authentication**: the FPGAModule shall authenticate all data sent or received
- **resource_sanitization**: all memory resources of the used Slot will get a neutral value during the reconfiguration process (e.g., by setting each bit of a BlockRam to zero). Resource sanitization does not affect memory elements accessed via DMA and, therefore, outside the utilized Slots.

It is possible to extend the list, if necessary. The current version of FPGASECML requires a minor change in the grammar, but a future version might provide a mechanism for user-defined or model-specific security mechanisms.

(b) Bitstream security

```
enum bitstream_security_featureENUM
      :
      bitstream_encryption | bitstream_authentication
      | bitstream_randomization | tamper_resistant_storage
;
```

The architects may deploy one or more of these security mechanisms to protect the binary FPGA configuration file of the FPGAModule from unauthorized access:

- **bitstream_encryption**: the design must encrypt the bitstream to protect the confidentiality of the FPGAModule
- **bitstream_authentication**: the design must provide a mechanism to validate the integrity of the FPGAModule
- **bitstream_randomization**: the resource utilization shall be randomized to protect the integrity of the FPGAModule against targeted alteration while they reside in the FPGA
- **tamper_resistant_storage**: the bitstream storage must protect the integrity of the FPGAModule against attacks.

Extensions to the list could be made to reflect the introduction of new security mechanisms.

D.2.1.4 Processing Blocks

Processing Blocks can store data, communicate with other FPGAModules, and provide services for the system.

```
Processing Block :
      'Processing Block:' name=ID
      'utilizes slot:'  slotid=[Slot] ';
      ('provides sensitive services:''{'
            sensitive_services+=sensitive_service+
      '}'';')
      ('contains sensitive data:' '
            {' sensitive_data+=sensitive_data+'}'';')
                  'bitstream is protected by:''{'
                        ((bsf+=bitstream_security_featureENUM
```

```
                  (','bsf+=bitstream_security_featureENUM)*)|'none')
            '}'';')
      ('implements security mechanism:'
            '{'((sec_mech+=security_mechanismENUM
            (','sec_mech+=security_mechanismENUM)*)|'none')'}'';')
      ('bitstream storage:' bitstreamstorage=ID ';')
';'
;
```

D.2.1.5 IO Blocks

Processing Blocks are only exposed to other FPGAModules but IO Blocks provide an additional connection to the outside world and are thereby exposed to other threat agents. Their description contains therefore the same information as Processing Blocks (these parts are marked in italic) plus periphery specific information:

```
IOBlock :
      'IO Block:' name=ID
      'utilizes slot:'  slotid=[Slot] ';'
      'connected to:'
            ( '(to be determined)' | '{' pd+=peripheral_device
            (','pd+=peripheral_device+ )*  '}')  ';'
      ('connects to threatlayer:' threatlayer=threatlayerENUM ';')?
      ('availability:' availability=availabilityENUM ';')
      ('provides sensitive services:''{'
            sensitive_services+=sensitive_service+ '}'';')
      ('contains sensitive data:' '{'
            sensitive_data+=sensitive_data+'}'';')
      ('bitstream is protected by:''{'
            ((bsf+=bitstream_security_featureENUM
            (','bsf+=bitstream_security_featureENUM)*)|'none') '}'';')
      ('implements peripheral security mechanism:'
            '{'((peripheral_sec_mech+=security_mechanismENUM
            ('|'peripheral_sec_mech+=security_mechanismENUM)*)
            |'none')'}' ';')?
      ('implements security mechanism:'
            '{'((sec_mech+=security_mechanismENUM
            (','sec_mech+=security_mechanismENUM)*)|'none')'}'';')
      ('bitstream storage:' bitstreamstorage=ID ';')
      ';'
;
```

The security model focuses on the FPGA design and data flow within them. It is, therefore, reasonable to consider only data stored inside the FPGA and functionality available through the IOBlock.

```
enum availabilityENUM returns ID:
      permanent | temporary | user_removable
;
```

A peripheral device might be either temporary or permanently available. Temporary availability indicates that it is removable and might, therefore, be more comfortable to manipulate. Further security concerns could arise when the required device is not available when expected.

(a) IO Block specific security attributes

Processing Blocks and IO Blocks share the same set of essential security attributes, but only IO Blocks have specific, periphery specific security attributes.

Threat Layer

```
enum threatlayerENUM:
      FPGA="FPGA"|PCB="PCB"|System="SYSTEM"|World="WORLD"
;
```

The different layers of the FPGAs environment are, as discussed in 3.4.2, populated by different threat agents - each one posing different security risks to the system. The enum may be extended to provide a more nuanced representation of the threat landscape.

Peripheral Security Mechanism

Additional security mechanisms might be necessary to prevent the threats agents in the periphery from endangering the FPGAs security. The list of peripheral security mechanisms is syntactically identical to the FPGAModule security mechanism list, but defense mechanisms listed in this category operate exclusively between the Peripheral Device and the FPGA. A valid enumeration of peripheral security mechanisms must not contain resource_sanitization as this mechanism operates only on the utilized FPGA resources and does not operate on sent or received data.

D.2.2 FPGA Resources

The FPGA Resource section is reduced to the very essentials to maximize the hardware independence of the model.

D.2.2.1 Slots

```
Slot  :
      'Slot' name=ID
      '{' var+=Slot_memory* '}'
      ';'
;
```

Slots represent distinct groups of disjoint FPGA primitives. This grouping of a set of FPGA primitives like CLBs and BRAM into Slots simplifies the analysis of designs with Partial Runtime Reconfiguration. The

information about the very set of primitives that make up the Slot is it is not required for the security analysis and therefore omitted from the model.

D.2.2.2 Memory

```
Slot_memory:
      'Memory' name=ID';'
;
```

Data stored in FPGA primitives (BRAM or CLBs configured as DRAM) poses a unique security risk as unauthorized or untrustworthy FPGAModule utilizing this Slot could gain read or write access to them. They could thereby access sensitive data or exercise undue control over sensitive FPGAModules reutilizing this memory element later.

D.2.2.3 Peripheral Devices

The periphery section contains information about the world outside the FPGA:

```
peripheral_device:
      'Peripheral Device' name=ID
      iop=ioport?
      ('resides at threatlayer:' threatlayer=threatlayerENUM ';')?
      ('provides sensitive services:'
            '{' sensitive_services+=security_levelENUM
            (','sensitive_services+=security_levelENUM)* '}'';')?
      ('contains sensitive data:' '{'
            sd+=sensitive_data (',' sd+=sensitive_data )* '}'';')?
      '{' var+=Slot_memory* '}'
      ';'
;
```

This part of the DSL has a purely informational character. The proof of concept software performs no further evaluation of this segment. A future version may incorporate this information into the Security Policy (via the query presented later) or mandate that each FPGAModules bitstream storage is a fully described Peripheral Device.

D.2.3 Communication infrastructure

```
communication:
      bus | network
;
```

The communication infrastructure is static and a set of directed connections between the subset of the designs Slots. Slot A has read access to Slot B if A is the source of an edge that has B as its target. Slot B has read access to Slot A. FPGASECML supports two forms of communication descriptions:

- Buses
- Networks

The implementation details, as well as the resources used, are omitted from the model for both rules.

D.2.3.1 Bus

```
bus   :
      'Bus' name=ID ('{' slotid+=[Slot] (','slotid+=[Slot])*'}')';'
;
```

Every member has bidirectional (read and write) access to every other bus member on a bus.

D.2.3.2 Networks

Networks are explicitly defined directional graphs with Slots as nodes. This rule provides more flexibility than the Bus-description introduced earlier. Networks also provide the capability to connect two or more buses. A dedicated filter clause, excluding connections between distinct Slots, can be used to model gateways with firewall functionality.

Their model representation follows these rules:

```
network:
      'Network' name=ID '{' nc=network_connection '}' cf=comfilter?
      ';'
;

network_connection:
      left=network_sub
            (write?='-->'|read?='<--'|bid?='<-->')
      right=network_sub
;

comfilter:
      'without' '{' nc+=network_connection (',' nc+=network_connection)*'}'
;

network_sub:
      '{' node+=network_node (','node+=network_node)*'}'
;

network_node:
      (Slot=[Slot]|'Bus' bus=[bus])
;
```

Besides these grammatical rules, a valid com(short for communication) filter must only contain Slots as network nodes.

D.2.3.3 The communication infrastructure as a graph

The model representation of the communication infrastructure is more suitable for declaration than evaluation. Further processing requires its transformation into a single, directed graph with the Slots as nodes and the supported data flow represented through directed edges.

(a) Graph grammar

The (theoretical) grammar for this consolidated graph representation is:

```
communication:
      cp+=comm_pair*
;
comm_pair:
      ([bus]|[network])? a=[Slot] '->' b=[Slot] ';'
;
```

These rules are not part of the current FPGASECML language. The software could derive such a representation during the parser run to create, for example, a visual representation using an EMF compatible diagram framework.

(b) Representation in the proof-of-concept software

The actual communication model of the PoC implementation uses the DefaultDirectedGraph class of the JGraphT [186] library.

D.3 Modeling Partial Runtime Reconfiguration

The dynamic segment of the model contains two sets of rules. Reconfiguration events overwrite, if they are triggered, the configuration of the FPGA with their given set of FPGAModules. The 'triggered by' statement allows us to assign the privilege to initiate this reconfiguration to a set of FPGAModules. An event that lacks this description is assumed to be caused by an external element. Reconfiguration sequences allow the validation of one or more independent sequences of reconfiguration events, each starting with the FPGA in its initial state. The dedicated event 'evInit' tags the initial configuration.

D.3.1 Event based description of PRR

```
event:
      'Event' name=ID
            ('triggered by' '{'
                  trigger+=[FPGAModule] (',' trigger+=[FPGAModule])*
            '}')?
      'loads:' '{'
            fpgamodules+=[FPGAModule](','fpgamodules+=[FPGAModule])*
            '}' ';'
;
```

A reconfiguration event indicates (or initiates – depending on the point of view) a change in the utilization of at least one FPGA Slot. The 'triggered by'-statement indicates that only a limited number of FPGAModules can initiate the reconfiguration process (opposite to, e.g., an external event) associated with this event. An event can change the content of multiple Slots at the same time, and an FPGAModule can be part of multiple configuration events.

D.3.2 Validation

A valid model requires at least an initial configuration, defining the content of each Slot at startup. This event must be called evInit. Each reconfiguration event must not contain more than one FPGAModule per Slot.

D.3.3 Partial Runtime Reconfiguration as a graph

The event-based approach is more suitable for declaration than processing. This representation is, therefore, transformed into a graph-based reconfiguration model for further use. In this model, the changed utilization of the FPGA (its state) are the nodes, and the reconfiguration events represent the edges between them. The initial configuration (indicated by evInit) serves as the root node (Figure 27.)

(a) Graph grammar

The grammar for the nodes recon_state is defined as:

```
recon_state:
      '{' rp+=state_pair+ '}'
;

state_pair:
      Slot=[Slot] ':' util=[FPGAModule]
;
```

A valid recon_state describes the utilization of each Slot. The edges between the states can be represented as:

```
recon_pair:
      orig=recon_state ':' event=[def::event]  ':' dest=recon_state
;
```

Where necessary, e.g., to create a visual representation using an EMF compatible diagram framework, such a representation could be derived during the parser run.

(b) JGraphT representation

The actual reconfiguration model of the PoC implementation is based on the DefaultDirectedGraph class of the JGraphT [186] library and generated during the parser run. The software generates two graphs, one containing all possible reconfiguration flows, the second constraining the flow to those reconfiguration events satisfying 'triggered by' statements (events with these statements can only be triggered if at least one FPGAModule with trigger privilege is present.)

D.3.3.2 Event sequences

```
recon_sequence:
      'Sequence' name=ID '{' eventr+=[event] (',' eventr+=[event])*  '}'
;
```

It is possible to include one or more reconfiguration sequences into the model for further analysis. The event sequences are independent of the reconfiguration graph.

D.4 The formal definition of the security policy

The security policy is a set of security rules that must be met by the architecture of the design. Each security rule defines a prohibited (or required) combination of two sets of FPGAModules with different sets of attributes. The system is compliant with the security policy if it complies with all of its rules. The definition of the security policy reflects the attribute-based access control paradigm discussed in chapter 5 (and the trust boundaries discussed in 3.4.3).

D.4.1 Security rules

The grammar for a security rule is defined as:

```
security_rule:
      'Rule' name=ID ':'
      (prohibit?='prohibits'|require?='requires')
      (util?='utilization'|pres?='presence'|com?='communication')'of'
            nested_sec_attr_A = nested_security_attribute_query
      ((im?='immediately')?  (before?='before'|after?='after'|'with'))
             nested_sec_attr_B = nested_security_attribute_query
      ';'
;
```

Each rule defines a valid consecutive or concurrent combination of two sets of FPGAModules. Their security attributes (see D.4.1.3) define each set. A 'prohibit'-security rule represents a trust boundary. Two classes of security rules are available, one constraining the consecutive utilization for each Slot, and the other one constraining the contemporary utilization of all FPGA Slots.

D.4.1.1 Constrained consecutive utilization

```
Rule name:
            (requires|prohibits)
            utilization of
                  FPGAMODULE_SET_A
            immediately (before|after)
                  FPGAMODULE_SET_B
;
```

This type of rule restricts the consecutive utilization of a Slot with FPGAModules of two different sets of attributes A and B.

D.4.1.2 Constrained contemporary utilization

```
Rule name:
            (requires|prohibits)
            (presence |communication) of
                  FPGAMODULE_SET_A
            (and|with)
                  FPGAMODULE_SET_B
;
```

This clause prohibits or requires the simultaneous presence of or communication between any FPGAModules with the attribute set A and FPGAModules with the attribute set B. The smart design of the reconfigurations events is often a better way to achieve this. Security rules constraining the communication do not consider the direction of the data flow, but only the possible exchange of data in one direction or the other.

Section 8.3 contains a detailed discussion of the different forms of security rules and their application.

D.4.1.3 Querying FPGAModules

Enforcement of the security rules requires the creation of the respective FPGAModule sets. The security_attribute_query-clause is used to describe the query of a single security-relevant attribute. The nested_security_attribute_query clause allows more powerful searches by combining these simple queries through logical terms.

(a) Security attribute query

This clause is used to determine which FPGAModules match the given attributes, for example the sensitivity of the services they perform. Lists of element are substituted through a stub instead of their real representation to improve the readability of the grammar. The xtext definition "element (‚element)*" becomes *elementlist*. FPGAModules can be queried by these attributes:

```
security_attribute_query:
      (
      ('FPGAModule:' '{' FPGAModulelist'}' ';')
      |('utilizes slot:'  '{' Slotlist '}' ';')
      |('connects to threatlayer:' '{' ( ThreatLayerlist | 'ANY') '}'';')
      |('availability:''{' Availabiltylist '}'';')
      |('provides sensitive services:' '{' SensitiveServicesList'}' ';' )
      |('sensitive data:'  '{' Datadiscriminatorlist'}'';')
      |('bitstream is protected by:'
            '{' Bitstreamsecurityfeaturelist | 'ANY') '}'';')
            |('implements peripheral security mechanism:'
       '{'  peripheral security mechanism list | 'ANY')'}' ';')
      |('implements security mechanism:'
            '{' Securitymechanismlist  | 'ANY') '}'';')
      |('bitstream storage:' '{' Bitstreamstoragelist '';')
      |('has security roles:''{' Rolelist '}'';')
      |(('at least one')? 'security role contains:'
```

```
            '{' role_regex=STRING '}'  ';')
      |('reconfiguration event:' '{' Eventlist '}' ';')
      |('connected to peripheral device:' '{' Peripheraldevicelist '}'';')
      |('has trustworthiness:''{' Trustworthinesslist)? '}'';')
      )
;
```

Querying sensitive data is a little bit more complicated than the rest and defined as:

```
sensitive_dataDiscriminator:
      accessType=dataAccesstypeENUM
      'access to.'
       (('sensitivity level' securityLevel=security_levelENUM)
            | ('Slot memory' mem=[Slot_memory | QualifiedName] )
)
';'
;
```

An FPGAModule satisfies the query if it contains at least one of the attributes given in the attribute list or contains an arbitrary attribute (the 'ANY-Keyword indicates this). All queries expect a complete match with the attribute except for "at least one security role contains:" that matches the roles of the FPGAModules with the given regular expression string. The query:

```
at least one security role contains: { "encryption_.*" };
```

matches all FPGAModules with at least one role that starts with the string "encryption_."

(b) Nested security attribute query

Nested security attributes query are built upon the security_attribute_query and combine them with logical operators:

```
nested_security_attribute_query:
      aqOR
;
aqOR  returns nested_security_attribute_query:
       aqAnd ({aqOr.left=current} "OR" right=aqAnd)*
;
aqAnd  returns nested_security_attribute_query:
      aqPrimary ({aqAnd.left=current} "AND" right=aqPrimary)*
;

aqPrimary returns nested_security_attribute_query:
      aqAtomic
      | '{' nested_security_attribute_query '}'
      | {aqNot} not?="NOT" expression=aqPrimary
      ;
```

```
aqAtomic returns nested_security_attribute_query:
      {security_attribute_query}  value=security_attribute_query
;
```

Evaluation of this sub-rule results in either a set of those FPGAModules that satisfies the query or an empty set. The proof of concept implementation iterates through the list of all FPGAModules and returns the set of them that satisfy the given nested security attribute query.

A nested security query could look like this:

```
{
      NOT { FPGAModule: { ioEthernet , pbAESUnit }; }
      AND { utilizes slot: { Slot1 }; }
}
OR
      { utilizes slot: { Slot2 };}
```

This nested query returns a list of all FPGAModules that utilize Slot2 and all from Slot1 except for ioEthernet and pbAESUnit (assuming that those two modules use Slot1.)

D.4.2 Reinforcement Learning Scenarios

The instruction to create Probabilistic MDP scenarios is optional and must be declared immediately after the security policy. The scenario declaration consists of two parts – the general definition of the parameters that are valid for all scenarios and the definition of the different scenarios.

D.4.2.1 General Parameters

All subsequently declared scenarios share these parameters:

```
'Security Policy:'
      security_policy+=security_rule*
('Probabilistic Reinforcement Learning:'
('Save episodes with fewer timesteps than'':'
                  rl_tstps_to_save=INT ';')?
      'Episodes'':' rl_episodes=INT';'
      'Epsilons' ':' '{' rl_eps+=float (',' rl_eps+=float)* '}'
      (rl_scenarios+=rl_scenario)*)?
```

The first option allows the user to save all episodes with fewer steps in a terminal state as defined - a feature to determine if there is, e.g., more than one unique solution within a reasonable time. The Episodes integer value defines the number of scenario restarts in each run while the final, general, option determines how many different runs are made for each scenario and which epsilon is to be used for each run.

D.4.2.2 Scenario description

A scenario is a set of goals (terminal conditions), rewards (awarded for reaching the terminal states and for any other action), the learning rate and a set of rules to define and modify the success rate of the predefined actions (as presented in 9.1.4.)

D.4.2.3 General Parameters

```
rl_scenario:
      'Scenario' name=ID
      ('Probabilistic' 'MDP' pmdp_off?='off' ';')?
      'Goal:'  goals=rl_goals
      'Reward:'
            'final' ':' final_state_reward=float ';'
            'other' ':' other_reward=float ';'
      'Actions:'
            'default' ('success' 'rate')? ':'
                  default_success_rate=float ';'
            (actions+=rl_action)*
            (r_actions+=rl_reconfiguration_action)*
            (s_exf_actions+=rl_exfiltrate_slot_action)*
';'
;
```

The scenario description includes an option to run the scenario as a deterministic MDP (e.g., to determine a baseline), the Goals that must be fulfilled before the MDP can terminate, and the reward the agents received for reaching a terminal state and any other action along the way.

```
rl_goals:
       'exfiltrate' (rl_goal_all?='all'|
            '{' fmodule+=[FPGAModule] (',' fmodule+=[FPGAModule])* '}' )
';'
;
```

Scenarios are only an extension to the deterministic MDPs presented earlier, and therefore only support the exfiltration of data from the FPGAModules.

D.4.2.4 Actions and their costs, success rate

The actions of the MDP can be separated into three classes: those targeting the reconfiguration control, a distinct FPGAModule, and those aimed at the Slots of the FPGA architecture.

```
rl_reconfiguration_action:
      'action' 'RECONFIGURE' event=[event] 'rate' success_rate=float
      ('cost' '=' cost=float)?
';'
;
```

This set of rules allows the user to define the success rate and costs of a reconfiguration through the attacker.

```
rl_exfiltrate_slot_action:
      'action' 'EXFILTRATE''SLOT' slot=[slot] 'rate' success_rate=float
('cost' '=' cost=float)?  ';'
;
```

These set of rules allows the user to define the success rate and costs of exfiltration data from a slot memory.

```
rl_action:
      'action' action=rl_default_actionENUM
      'success'(( 'rate''=' success_rate=float)
      |(sfs+= rl_success_factor*))
      ';'
;
```

This rule allows the user to define the success rate of an action against an FPGAModule either within the FPGA (ATTACK_NEIGHBOURS, ATTACK_PERIPHERY, EXFIL_DATA) or outside (TAMPER_BITSTREAM). The success rate can be defined directly for all attacks of this class. They could also be calculated through a group of factors defined for a set of FPGAModules. The same queries used for the security policy defines these sets as well.

```
rl_success_factor:
      'factor' success_factor=float
'against'
      '{' nested_sec_attr = nested_security_attribute_query  '}'
';'
;
```

The success rate for each action and FPGAModule is calculated by multiplying the default success rate with all relevant success factors. The calculated success rates are finally divided by the highest calculated success rate to fit them in a range between zero and one.

Appendix E Data-driven trustworthiness assessments

This appendix provides a short overview of the building blocks for a data-driven trustworthiness assessment system for FPGAs.

E.1 Dataset of potential trustworthiness indicators

The quality of the model depends on the quality and quantity of the available data. Collecting and preparing security-related data is an essential first step. This section describes the challenges involved in this step, possible sources and proposes a structure to collect and store data. The statistical methods proposed in this section can process both qualitative data (like lines of code) and quantitative information (like Vendor A, Vendor B)

E.2 Data structure

This section outlines the three main categories of data.

E.2.1 Element implementation

The attributes of the element itself are of the highest interest for the security assessment:

- Is it implementing an existing standard or an in-house specification?
- How often is this element in use? Is the element a reusable component, or was it tailored for one application?
- How complex is it? The assessor may rely on metrics as simple as Lines of Code (LOC) or more sophisticated forms.
- In which form is it available? VHDL or Verilog code, for example, is more accessible to screen for possible backdoors than elements only available in binary form.

E.2.2 Historical data

The history of the element can provide further insights:

- Has the element a trustworthiness assessment? By whom?
- When was it created, and by whom?
- How good was the development process at that time and in that organization?
- The development and test process used which tools?
- How rigorous was the testing? How many errors did the tester find?
- How are its maintenance status and history? (refactoring, bug fixing)
- Were there previous security incidents?
- Are there known problems or vulnerabilities?
- Are there other versions or configurations of this element in use?

E.2.3 Related Information

Related information can be used to supplement or substitute direct information about the element as successful attacks against one implementation may indicate problems for other implementations:

- Which development tools where used? Which Version of them?

- Does it implement a specific standard or specification?
- Are there known security problems with the tools, specifications, and standards?

E.3 Model

Three steps are necessary to create a machine learning model:

1. Construction of a function f
2. Training (or parametrization9 of this function
3. Validation of fs performance against a set of validation data

Libraries like Scikit-learn [187] for Python or dedicated statistic programs like R [167] provide the necessary functionality to conduct these steps for many different functions and parameters with a few lines of code.

The models should not be too complicated to ease their interpretation and validation. That means that the used function f should be kept simple. It is assumed that a sufficient number of (Y, X) pairs are available to train f. Training minimizes the error between the provided and the calculated response. The model might perform only well on the training data; this renders its results useless for unknown datasets and creates an inappropriate level of confidence in the assessment. This phenomenon, called overfitting, can be avoided by validating the model against a set of data not used in the training set. Some models require the usage of a designated validation set (a split of 80/20 between training and validation data is typical). The methods proposed in this section support cross-validation, the alternating use of data for training and validation, which avoids this waste of data.

E.3.1 Functions

Suitable statistical learning methods include generalized linear models like linear or logistic regression as well as trees based functions. The deployment of these relatively simple methods eases the plausibility check of the model and reduces the risk of overfitting.

E.3.1.1 Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n \text{ (E.1)}$$

Linear Regression (formula E.1) assumes a linear relationship between the given set of predictors X and the response Y. Parameters β are chosen In the training phase to minimize the error between the given and the computed response. Predictors can be preprocessed by mathematical operations like logarithm or a polynomial operation – the relationship between predictor and response remains linear. Qualitative variables (Developer A, B) can be represented as binary encoded dummy variables (A=0, B=1).T he influence of different predictors can be easily determined from their respective parameters (given that problems like collinearity have been adequately addressed.) The quality of the model can be verified either by a separate validation data set or cross-validation.

E.3.1.2 Logistic Regression

$$\log\left(\frac{P(X)}{1-P(X)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n \text{ (E.2)}$$

Logistic Regression (formula E.2) can be used to calculate the odds that a set of predictors belong to a given class or not. Multinomial regression supports multiple, independent classes, and ordinal regression accepts

multiple classes with a distinct order (High=3, Medium=2, Low=1) between them [188]. Either a validation set or cross-validation must be used to verify the quality of the model.

E.3.1.3 Regression and Classification Trees

Trees based methods can be used for both regression and classification problems. Both decision and regression trees can be created automatically through software and validated by using either a validation set or cross-validation. Multiclass regression is easy to implement, and the graphical representation of the tree makes it easier to explain and to analyze as models based on linear or logistic regression. Small changes in the underlying database can result in vastly different trees.

E.3.1.4 Further options

Neuronal networks [81] are the state of the art technology for many machine learning problems, but their application appears counterproductive in this context. Neuronal Networks require a large number of data for training and validation, they are much harder to interpret, and their flexibility increases the risk of overfitting. Ensembles [189], combining multiple (simpler) sub-models, can improve the overall performance of the model, but they increase its complexity and decrease its interpretability. Unsupervised learning methods like k-Means could be used to find elements with similar attributes but allow no classification or regression of the result. Advances in reinforcement learning may result in software that can automatically test the resilience of the software, easing or removing the reliance on metadata. Chapter 9 presents a novel method that applies reinforcement learning to find potential weaknesses in FPGAs architecture.

E.3.1.5 Response

The responses can be either an explicit trustworthy assessment or information beneficial to the assessment.

(a) Classification model

A classification model may return trustworthy classes like:

- Trustworthy either {yes, no}
- Trustworthy either {High, Medium, Low}.

(b) Regression model

Finding the appropriate response to the regression problem is more challenging; candidates for the regression models response include:

- Number of security-relevant bugs
- Number of security incidents
- Accumulated costs of security incidents
- An abstract security score (e.g., 0-100 low trustworthiness, 100-199 medium, more than 200 high)

E.3.1.6 Predictors

Selecting the right subset of predictors from the available dataset is a process that requires multiple rounds of selection, training, validation, and model comparison. Methods for the partial automation of this process

exist. In some cases, the predictors may require a certain amount of preprocessing like normalization or the application of nonlinear operations.

E.4 Conclusion

Assessing the trustworthiness of the design elements and their supply chain, even if the result of this analysis is inherently fuzzy, is an important step to improve the security of the design. The partial automatization of the assessment process through machine learning appears promising but requires an extensive dataset for training and validation as well as cautious users, fully aware of their pitfalls and restrictions.

Appendix F Examples for FPGASECML formal verification

This section presents five different use cases for security rules, their field of application, the formulation of the policy as well as their representation in LTL. An example of each class is presented.

F.1 Resource Sanitization

Removing sensitive data earlier reduces the risk of leakage of information attacks. Another module must perform this task when the sensitive FPGAModule has no control over its removal from the state (preemptive multitasking) or cannot perform the demanded sanitization. Each module providing this service must, therefore, have an appropriate security attribute. Additional modules, providing the only sanitization for a single Slot, may be added to the design. An FPGA module with the demanded sanitization attribute should remove all sensitive information from the Slot.

F.1.1 Security rules

Rules of this shape represent the sanitization security requirement:

```
Rule name:
            requires utilization of
                  implements security mechanism:
                  { resource_sanitization } ;
            immediately after
                  ATTRIBUTES_REQUIRING_RESOURCE_SANITIZATION
;
```

It is also possible to make further restrictions as a system architect might, e.g., only entrust highly trustworthy FPGAModules with this sensible task.

F.1.2 LTL pseudocode

```
For each Slot in the architecture:
    G ({at least one module in this Slot demanding sanitization}
      -> X {at least one module in this Slot providing sanitization})
```

An FPGAModule removing sensitive data must be scheduled immediately after each module demanding sanitization. Both FPGAModules must utilize the same Slot, the code generator, therefore, iterates through each Slot and generates the appropriate LTL rule. It does not generate a rule if no module requires sanitization for this Slot, and the LTL rule is a hard (FALSE) if no FPGAModule can satisfy the demand.

F.1.3 Scenario

A given design uses a single Slot. FPGAModule A provides an encryption service; a BRAM stores the symmetric key. A potentially malicious module B, utilizing the same BRAM after module A, could access this secret key either as a whole or in parts (modules utilizing the Slot between modules A and B could override parts of the memory.) Module SAN ensures the sanitization of the Slot resource.

F.1.4 FPGASECML-Model

```
FPGAArchitecture sanitizationModel

FPGAModules:

      Processing Block: sl1_A
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none };
                  bitstream is protected by:    { bitstream_encryption };
                  implements security mechanism:{ none } ;
                  bitstream storage:            bstore1 ;
                  trustworthiness:              { high };
                  assigned security roles:      { rCryptography  };
                  ;

      Processing Block: sl1_B
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none};
                  bitstream is protected by:    { bitstream_encryption };
                  implements security mechanism:{ none } ;
                  bitstream storage:            bstore1 ;
                  trustworthiness:              { low };
                  assigned security roles:      { none };
                  ;

      Processing Block: sl1_C
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none};
                  bitstream is protected by:    { bitstream_encryption };
                  implements security mechanism:
                                                { resource_sanitization };
                  bitstream storage:            bstore1;
                  trustworthiness:              { medium };
                  assigned security roles:      { none };
                  ;

      Processing Block: sl1_SAN
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none};
                  bitstream is protected by:    { bitstream_encryption };
                  implements security mechanism:
                                                { resource_sanitization };
                  bitstream storage:            bstore1 ;
                  trustworthiness:              { high };
                  assigned security roles:      { none };
                  ;
```

```
FPGAResources:
      Slot Slot1 { };

Communication:

Reconfiguration:
      Events:
                  Event evInit      loads: { sl1_A };
                  Event ev_B        loads: { sl1_B };
                  Event ev_C        loads: { sl1_C };
                  Event ev_SAN      loads: { sl1_SAN };

      Sequence valid_seq {ev_SAN,ev_B, ev_C}
      Sequence invalid_seq {ev_B,ev_SAN,ev_B, ev_C}

Security Policy:

      Rule santest :
      requires utilization of
      {
            implements security mechanism: { resource_sanitization } ;
            AND has trustworthiness: { high  };
      }
      immediately
      after
      {
            has security roles: { rCryptography };
      }
      ;
```

F.1.5 NuSMV Model to determine a valid schedule

This NuSMV model returns, if possible, a valid reconfiguration sequence that uses each FPGAModule at least once.

```
VAR
      reEvent : {evInit,ev_B,ev_C,ev_SAN};
      mySlot1 : Slot1(reEvent);

LTLSPEC ! (
--santest
G ((mySlot1.state = sl1_A) -> X (mySlot1.state = sl1_SAN))
-- force the utilization of each module
 & F( mySlot1.state=sl1_A ) & F( mySlot1.state=sl1_B ) & F(
mySlot1.state=sl1_C )
 & F( mySlot1.state=sl1_SAN ))


MODULE Slot1 (reEvent)
VAR
      state : {noState1,sl1_C,sl1_B,sl1_A,sl1_SAN,noState2};
```

```
ASSIGN
      init(state) :=    sl1_A ;
      7next(state) :=   case
            reEvent =           ev_C :      sl1_C;
            reEvent =           ev_B :      sl1_B;
            reEvent =         ev_SAN :    sl1_SAN;
            TRUE : state ;
      esac;
```

Running this model through NuSMV leads to this result:

```
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:51 2015)
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:51 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification !(((( G (mySlot1.state = sl1_A ->  X mySlot1.state =
sl1_SAN) &  F mySlot1.state = sl1_A) &  F mySlot1.state = sl1_B) &  F
mySlot1.state = sl1_C) &  F mySlot1.state = sl1_SAN)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    reEvent = ev_SAN
    mySlot1.state = sl1_A
  -> State: 1.2 <-
    mySlot1.state = sl1_SAN
  -> State: 1.3 <-
    reEvent = ev_C
  -> State: 1.4 <-
    reEvent = ev_SAN
    mySlot1.state = sl1_C
  -> State: 1.5 <-
    reEvent = ev_B
    mySlot1.state = sl1_SAN
  -> State: 1.6 <-
    reEvent = ev_SAN
    mySlot1.state = sl1_B
  -- Loop starts here
```

```
  -> State: 1.7 <-
    mySlot1.state = sl1_SAN
  -> State: 1.8 <-
```

This sequence satisfies the security policy as sl1_A is followed immediately by sl1_SAN, but the frequent utilization of sl1_SAN indicates room for (manual) optimization. The system architect could now use this schedule, optimize it, and add it to the model as a reconfiguration sequence and use the model to verify the optimized schedule.

A.1 Secure setup

This rule ensures that the appropriate preparation of the Slot before any sensitive operation takes place. It can be either used to remove potentially hazardous data from the system or to perform a particular setup procedure (e.g., to establish a secure communication channel) that provides a secure environment.

F.1.6 Security rules

The security rule for secure setup requirements follows this pattern:

```
Rule name:
            requires
                  FPGAMODULES_PROVIDING_SECURE_SETUP
            immediately before
                  FPGAMODULES_REQUIRING_SECURE_UPDATE
;
```

F.1.7 LTL Pseudocode

```
For each Slot in the architecture:
      G ((at least one module in this Slot demanding secure setup)
            -> Y (at least one module in this Slot providing secure setup))
```

F.1.8 Scenario

A secure setup procedure, performed by Module sl1_CommSetup, must be completed before Module C can communicate securely with the world.

F.1.9 FPGASECML-Model

```
FPGAArchitecture secureSetupModel

FPGAModules:
      IO Block: sl1_C
                  utilizes slot: Slot1;
                  connected to: (to be determined) ;
                  connects to threatlayer:  WORLD ;
                  availability: permanent ;
                  provides sensitive services: { none };
```

```
                  contains sensitive data:  { none };
                  bitstream is protected by:
                        { bitstream_encryption, tamper_resistant_storage };
                  implements peripheral security mechanism:
                        { data_sanitization,data_authentication} ;
                  implements security mechanism: { none } ;
                  bitstream storage: test ;
                  trustworthiness: { high };
                  assigned security roles: { rSensitiveCommunication  };
            ;

      Processing Block: sl1_A
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none };
                  bitstream is protected by:
                        { bitstream_encryption, tamper_resistant_storage };
                  implements security mechanism: { none } ;
                  bitstream storage:  bstore1 ;
                  trustworthiness: { medium };
                  assigned security roles: { none };
            ;

      Processing Block: sl1_B
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none };
                  bitstream is protected by:
                        { bitstream_encryption, tamper_resistant_storage };
                  implements security mechanism:
                        { resource_sanitization } ;
                  bitstream storage: TEST1;
                  trustworthiness: { medium };
                  assigned security roles: { none };
            ;


      Processing Block: sl1_CommSetup
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none };
                  bitstream is protected by:    { none };
                  implements security mechanism:{ none };
                  bitstream storage: bstore1 ;
                  trustworthiness: { high };
                  assigned security roles: {  rSecureCommSetup };
            ;

FPGAResources:
      Slot Slot1 { };

Communication:
```

```
Reconfiguration:
      Events:
                  Event evInit            loads: { sl1_A };
                  Event ev_B              loads: { sl1_B };
                  Event ev_C              loads: { sl1_C };
                  Event ev_ComS           loads: { sl1_CommSetup  };

      Sequence valid_seq      {ev_ComS,ev_C, ev_B}
      Sequence invalid_seq    {ev_C,ev_B }

Security Policy:

      Rule secure_setup_test :
      requires utilization of
      {
            has security roles: { rSecureCommSetup };
      }
      immediately
      before
      {
            has security roles: { rSensitiveCommunication };
      }
      ;
```

F.1.9.1 NuSMV model to validate the sequence [ev_ComS,ev_C, ev_B]

```
MODULE main
VAR
      reEvent :{evInit,ev_B,ev_C,ev_ComS};
      state : { state_0, state_1, state_2};
      mySlot1 : Slot1(reEvent);
ASSIGN
      init(state) :=    state_0;
      init(reEvent) :=ev_ComS;
      next(state) := case
            state=state_0  : state_1;
            TRUE  : state;
      esac;
      next(reEvent):= case
            state=state_0     : ev_C;
            state=state_1     : ev_B;
            TRUE  : reEvent;
      esac;

--secure_setup_test
LTLSPEC G ((mySlot1.state = sl1_C) -> Y (mySlot1.state = sl1_CommSetup))

MODULE Slot1 (reEvent)
VAR
      state : {noState1,sl1_A,sl1_C,sl1_CommSetup,sl1_B,noState2};
```

```
ASSIGN
      init(state) :=    sl1_A ;
      next(state) :=    case
            reEvent =           ev_C : sl1_C;
            reEvent =         ev_ComS : sl1_CommSetup;
            reEvent =           ev_B : sl1_B;
            TRUE : state ;
      esac;
```

Validated by NuSMV:

```
-- specification  G (mySlot1.state = sl1_C ->  Y mySlot1.state =
sl1_CommSetup)  is true
```

F.1.9.2 NuSMV model to evaluate the invalid sequence {ev_C,ev_B}

The sequence evInit,evC, evB violates the security policy. Adapting the main-Module to the new sequence results in:

```
MODULE main
VAR
      reEvent :{evInit,ev_B,ev_C,ev_ComS};
      state : { state_0, state_1};
      mySlot1 : Slot1(reEvent);
ASSIGN
      init(state)   := state_0;
      init(reEvent) := ev_C;
      next(state)   := case
            TRUE    : state;
      esac;
      next(reEvent):= case
            state=state_0     : ev_B;
            TRUE  : reEvent;
      esac;
```

NuSMV confirms this suspicion:

```
-- specification  G (mySlot1.state = sl1_C ->  Y mySlot1.state =
sl1_CommSetup)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    reEvent = ev_C
    state = state_0
    mySlot1.state = sl1_A
  -> State: 1.2 <-
    reEvent = ev_B
```

```
    mySlot1.state = sl1_C
  -- Loop starts here
  -> State: 1.3 <-
    mySlot1.state = sl1_B
  -> State: 1.4 <-
```

F.2 Consecutive exclusion

These rules prohibit the presence of FPGAModules with one set of attributes after FPGAModules with another set of attributes utilized the same Slot.

F.2.1 Security rules

Consecutive exclusion is enforced with the terms:

AFTER-Clause:

```
      Rule name :
      prohibits utilization of { ATTRIBUTE_SET_A }
      after                    { ATTRIBUTE_SET_B };
```

BEFORE-Clause:

```
Rule name :
      prohibits utilization of { ATTRIBUTE_SET_A }
      before                   { ATTRIBUTE_SET_B };
```

F.2.2 LTL pseudocode

AFTER-Clause:

```
For each Slot s:
      G({any FPGAModule with ATTRIBUTE_SET_B utilizing s }
      ->!F{ any FPGAModule with ATTRIBUTE_SET_A utilizing s })
```

BEFORE-Clause:

```
For each Slot s:
      G({any FPGAModule with ATTRIBUTE_SET_B utilizing s }
      ->H!{ any FPGAModule with ATTRIBUTE_SET_A utilizing s })
Or
      G({any FPGAModule with ATTRIBUTE_SET_A utilizing s }
      ->!F{ any FPGAModule with ATTRIBUTE_SET_B utilizing s })
```

F.2.3 Scenario

The FPGAModules A, B, and C share the same Slot and operate on the same block ram. A encrypts the data; module B generates the Message Authentication Code, and module C transmits the encrypted and authenticated data to the outside world. FPGAModule A (accessing unencrypted data) is considered the most sensitive operation and the most trustworthy implementation, FPGAModule B has lower trustworthiness than A, and C has the lowest trustworthiness. To prevent module C from tampering with or transmitting unencrypted and unauthenticated data, C must be utilizing the Slot only after B, and A. B must utilize the Slot only after A to avoid the authentication of unencrypted data.

F.2.4 FPGASECML-Model

```
FPGAArchitecture utilizationMODEL

FPGAModules:
      Processing Block: sl1_INIT
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none };
                  bitstream is protected by:    { none };
                  implements security mechanism:{ none } ;
                  bitstream storage:            bstore1 ;
                  trustworthiness:              { high };
                  assigned security roles:      { rINIT};
                  ;

      Processing Block: sl1_A
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none };
                  bitstream is protected by:    { none };
                  implements security mechanism:{ none } ;
                  bitstream storage:            bstore1 ;
                  trustworthiness:              { high };
                  assigned security roles:      { rHighSensitivityOp };
                  ;

      Processing Block: sl2_B
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none };
                  bitstream is protected by:    { none };
                  implements security mechanism:{ none };
                  bitstream storage:            bstore1;
                  trustworthiness:              { medium };
                  assigned security roles:      { rMediumSensitivityOp };
                  ;

      Processing Block: sl1_C
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none };
                  bitstream is protected by:    { none };
```

```
                  implements security mechanism:{ none } ;
                  bitstream storage:            bstore1;
                  trustworthiness:              { low };
                  assigned security roles:      {  rLowSensitivityOp };
                  ;

FPGAResources:
      Slot Slot1 { };


Communication:
Bus b1 {Slot1};

Reconfiguration:
      Events:
                  Event evInit      loads: { sl1_INIT };
                  Event ev_A        loads: { sl1_A };
                  Event ev_B        loads: { sl2_B };
                  Event ev_C        loads: { sl1_C };

      Sequence valid_seq      { ev_A,ev_B, ev_C }
      Sequence invalid_seq    { ev_C,ev_B, ev_A }

Security Policy:

      Rule util_1 :
      prohibits utilization of
      {
            has trustworthiness: { high };
      }
      after
      {
            has trustworthiness: { medium,low };
      };

      Rule util_2 :
      prohibits utilization of
      {
            has trustworthiness: { medium,high };
      }
      after
      {
            has trustworthiness: { low };
      };

      Rule util_3 :
      prohibits utilization of
      {
            has trustworthiness: { low };
      }
      before
      {
```

```
            has trustworthiness: { medium,high};
      }
```

This policy mixes before and after-clauses to demonstrate the transformation in both cases. The readability would benefit from a uniform use of the after clause.

F.2.4.1 NuSMV Model to determine a valid schedule

This NuSMV model returns, if possible, a valid reconfiguration sequence that uses each FPGAModule at least once.

```
MODULE main
VAR
      reEvent : {evInit,ev_A,ev_B,ev_C};
      mySlot1 : Slot1(reEvent);

LTLSPEC ! (
--util_1
G ((mySlot1.state = sl2_B | mySlot1.state = sl1_C) ->  !F (mySlot1.state =
sl1_INIT | mySlot1.state = sl1_A))

&
--util_2
G ((mySlot1.state = sl1_C) ->  !F (mySlot1.state = sl1_INIT | mySlot1.state
= sl1_A | mySlot1.state = sl2_B))

&
--util_3
G ((mySlot1.state = sl1_INIT | mySlot1.state = sl1_A | mySlot1.state =
sl2_B) -> H!(mySlot1.state = sl1_C))

-- force the utilization of each module
 & F( mySlot1.state=sl1_INIT ) & F( mySlot1.state=sl1_A ) & F(
mySlot1.state=sl2_B )
 & F( mySlot1.state=sl1_C ))

MODULE Slot1 (reEvent)
VAR   state : {noState1,sl2_B,sl1_INIT,sl1_A,sl1_C,noState2};
ASSIGN
      init(state) :=    sl1_INIT ;
      next(state) :=    case
            reEvent =           ev_B :      sl2_B;
            reEvent =           ev_A :      sl1_A;
            reEvent =           ev_C :      sl1_C;
            TRUE : state ;
      esac;
```

This NuSMV model returns, if possible, a valid reconfiguration sequence that uses each module at least once.

```
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    reEvent = evInit
    mySlot1.state = sl1_INIT
  -> State: 1.2 <-
    reEvent = ev_A
  -> State: 1.3 <-
    reEvent = ev_B
    mySlot1.state = sl1_A
  -> State: 1.4 <-
    reEvent = ev_C
    mySlot1.state = sl2_B
  -- Loop starts here
  -> State: 1.5 <-
    mySlot1.state = sl1_C
  -- Loop starts here
  -> State: 1.6 <-
  -> State: 1.7 <-
```

F.2.4.2 NuSMV module to validate the sequence {ev_C,ev_B, ev_A }

The model with the reconfiguration event sequence {ev_C,ev_B, ev_A } does not satisfies the security policy:

```
-- specification  G ((mySlot1.state = sl2_B | mySlot1.state = sl1_C) -> !(
F (mySlot1.state = sl1_INIT | mySlot1.state = sl1_A)))  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    reEvent = ev_C
    state = state_0
    mySlot1.state = sl1_INIT
  -> State: 1.2 <-
    reEvent = ev_B
    state = state_1
    mySlot1.state = sl1_C
  -> State: 1.3 <-
    reEvent = ev_A
    mySlot1.state = sl2_B
  -- Loop starts here
  -> State: 1.4 <-
    mySlot1.state = sl1_A
  -- Loop starts here
  -> State: 1.5 <-
  -> State: 1.6 <-

-- specification  G (mySlot1.state = sl1_C -> !( F ((mySlot1.state =
sl1_INIT | mySlot1.state = sl1_A) | mySlot1.state = sl2_B)))  is false
-- as demonstrated by the following execution sequence
```

```
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 2.1 <-
    reEvent = ev_C
    state = state_0
    mySlot1.state = sl1_INIT
  -> State: 2.2 <-
    reEvent = ev_B
    state = state_1
    mySlot1.state = sl1_C
  -> State: 2.3 <-
    reEvent = ev_A
    mySlot1.state = sl2_B
  -- Loop starts here
  -> State: 2.4 <-
    mySlot1.state = sl1_A
  -> State: 2.5 <-

-- specification  G (((mySlot1.state = sl1_INIT | mySlot1.state = sl1_A) |
mySlot1.state = sl2_B) ->  H !(mySlot1.state = sl1_C))  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 3.1 <-
    reEvent = ev_C
    state = state_0
    mySlot1.state = sl1_INIT
  -> State: 3.2 <-
    reEvent = ev_B
    state = state_1
    mySlot1.state = sl1_C
  -> State: 3.3 <-
    reEvent = ev_A
    mySlot1.state = sl2_B
  -- Loop starts here
  -> State: 3.4 <-
    mySlot1.state = sl1_A
  -> State: 3.5 <-
```

F.3 Concurrent exclusion

It may become necessary to isolate sensitive modules of (presumable) high trustworthiness from modules with lower trustworthiness. This technique reduces the negative influence (e.g., through attacks against a common communication network) of potentially malicious modules to delicate tasks.

F.3.1 Security Rules

Rules of this shape force the design to exclude the concurrent present of FPGAModules from set A and B:

```
Rule name:
      prohibits
```

```
            presence of
                  FPGAMODULE_SET_A
            and
                  FPGAMODULE_SET_B
;
```

F.3.2 LTL Pseudocode

Each rule of the security policy that satisfies the above shape becomes an LTL rule of the shape:

```
G!({any module from   FPGAMODULE_SET_A}
& { any module from   FPGAMODULE_SET_B})
```

For all valid states: modules with attribute A and modules with attribute B must never be present at the same time.

F.3.3 Scenario

An FPGA design includes modules performing operations of high sensitivity (Modules A, D, and E) and modules with only medium trustworthiness (Modules B, C). The design must prevent modules with low trustworthiness from endangering sensitive operations. It is further assumed that the mere presence of these modules endangers the security of the system. Therefore, modules of the groups {A, D, E}, and {B, C} must never share the FPGA at the same time. Module F is regarded as highly trustworthy and poses neither a threat to the high sensitivity objects nor is its operation critical enough to cause serious harm if attacked.

F.3.4 FPGASECML-Model

```
FPGAArchitecture concurrentModel

FPGAModules:
      Processing Block: sl1_A
                  utilizes slot: Slot1;
                  provides sensitive services:  { none };
                  contains sensitive data:      { none };
                  bitstream is protected by:    { none };
                  implements security mechanism:{ none } ;
                  bitstream storage:            bstore1 ;
                  trustworthiness:              { high };
                  assigned security roles:      { rHighSensitivityOp };
                  ;

      Processing Block: sl2_B
                  utilizes slot: Slot2;
                  provides sensitive services:        { none };
                  contains sensitive data:            { none };
                  bitstream is protected by:          { none };
                  implements security mechanism:      { none };
                  bitstream storage:                  bstore1;
                  trustworthiness:                    { medium };
```

```
                assigned security roles:            { none };
                ;

      Processing Block: sl1_C
                utilizes slot: Slot1;
                provides sensitive services:  { none };
                contains sensitive data:      { none };
                bitstream is protected by:    { none };
                implements security mechanism:{ resource_sanitization };
                bitstream storage:            bstore1;
                trustworthiness:              { medium };
                assigned security roles:      { none };
                ;


      Processing Block: sl2_D
                utilizes slot: Slot2;
                provides sensitive services:  { none };
                contains sensitive data:      { none };
                bitstream is protected by:    { none };
                implements security mechanism:{ none };
                bitstream storage:            bstore1;
                trustworthiness:              { high };
                assigned security roles:      { rHighSensitivityOp };
                ;

    Processing Block: sl2_E
                utilizes slot: Slot2;
                provides sensitive services:  { none };
                contains sensitive data:      { none };
                bitstream is protected by:    { none };
                implements security mechanism:{ none };
                bitstream storage:            bstore1 ;
                trustworthiness:              { high };
                assigned security roles:      { rHighSensitivityOp };
                ;


    Processing Block: sl1_F
                utilizes slot: Slot1;
                provides sensitive services:  { none };
                contains sensitive data:      { none };
                bitstream is protected by:    { bitstream_encryption };
                implements security mechanism:{ none } ;
                bitstream storage:            bstore1 ;
                trustworthiness:              { high };
                assigned security roles:      { rLowSensitivityOp  };
                ;


FPGAResources:
      Slot Slot1 { };
```

```
      Slot Slot2 { };

Communication:
  Bus b1 {Slot1,Slot2};

Reconfiguration:
      Events:
                  Event evInit      loads: { sl1_A, sl2_D };
                  Event ev_B        loads: { sl2_B };
                  Event ev_C        loads: { sl1_C };
                  Event ev_D        loads: { sl2_D };
                  Event ev_E        loads: { sl2_E };
                  Event ev_F        loads: { sl1_F };
      Sequence valid_seq            { ev_E,ev_F,ev_B,ev_C}
      Sequence invalid_seq          { ev_C,ev_C }

Security Policy:

      Rule util_1 :
      prohibits presence of
      {
            has security roles: { rHighSensitivityOp };
      }
      and
      {
            has trustworthiness: { medium,low };
      }
      ;
```

F.3.5 NuSMV Model to determine a valid schedule

This NuSMV model returns, if possible, a valid reconfiguration sequence that uses each module at least once.

```
MODULE main
VAR
      reEvent : {evInit,ev_B,ev_C,ev_D,ev_E,ev_F};
      mySlot1 : Slot1(reEvent);
      mySlot2 : Slot2(reEvent);

LTLSPEC ! (
--util_1
G ! ((mySlot1.state = sl1_A | mySlot2.state = sl2_D
    | mySlot2.state = sl2_E)
& (mySlot2.state = sl2_B | mySlot1.state = sl1_C))

-- force the utilization of each module
 & F( mySlot1.state=sl1_A ) & F( mySlot2.state=sl2_B )
 & F( mySlot1.state=sl1_C ) & F( mySlot2.state=sl2_D )
 & F( mySlot2.state=sl2_E ) & F( mySlot1.state=sl1_F )
)
```

```
MODULE Slot1 (reEvent)
VAR
      state : {noState1,sl1_C,sl1_A,sl1_F,noState2};
ASSIGN
      init(state) :=    sl1_A ;
      next(state) :=    case
            reEvent =           ev_C :      sl1_C;
            reEvent =           ev_F :      sl1_F;
            TRUE : state ;
      esac;

MODULE Slot2 (reEvent)
VAR
      state : {noState1,sl2_D,sl2_E,sl2_B,noState2};
ASSIGN
      init(state) :=    sl2_D ;
      next(state) :=    case
            reEvent =           ev_E :      sl2_E;
            reEvent =           ev_B :      sl2_B;
            reEvent =           ev_D :      sl2_D;
            TRUE : state ;
      esac;
```

The NuSMV run for the model returned this schedule:

```
-- specification !(((((( G !(((mySlot1.state = sl1_A | mySlot2.state =
sl2_D) | mySlot2.state = sl2_E) & (mySlot2.state = sl2_B | mySlot1.state =
sl1_C)) &  F mySlot1.state = sl1_A) &  F mySlot2.state = sl2_B) &  F
mySlot1.state = sl1_C) &  F mySlot2.state = sl2_D) &  F mySlot2.state =
sl2_E) &  F mySlot1.state = sl1_F)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    reEvent = ev_D
    mySlot1.state = sl1_A
    mySlot2.state = sl2_D
  -> State: 1.2 <-
    reEvent = ev_F
  -> State: 1.3 <-
    reEvent = ev_B
    mySlot1.state = sl1_F
  -> State: 1.4 <-
    reEvent = ev_E
    mySlot2.state = sl2_B
  -> State: 1.5 <-
    reEvent = ev_B
    mySlot2.state = sl2_E
  -> State: 1.6 <-
    reEvent = ev_D
```

```
  mySlot2.state = sl2_B
-> State: 1.7 <-
  reEvent = ev_B
  mySlot2.state = sl2_D
-> State: 1.8 <-
  reEvent = ev_C
  mySlot2.state = sl2_B
-- Loop starts here
-> State: 1.9 <-
  reEvent = ev_B
  mySlot1.state = sl1_C
-- Loop starts here
-> State: 1.10 <-
-> State: 1.11 <-
  reEvent = ev_F
-> State: 1.12 <-
  reEvent = evInit
  mySlot1.state = sl1_F
-> State: 1.13 <-
  reEvent = ev_E
-> State: 1.14 <-
  reEvent = ev_B
  mySlot2.state = sl2_E
-> State: 1.15 <-
  reEvent = ev_D
  mySlot2.state = sl2_B
-> State: 1.16 <-
  reEvent = ev_B
  mySlot2.state = sl2_D
-> State: 1.17 <-
  reEvent = ev_C
  mySlot2.state = sl2_B
-- Loop starts here
-> State: 1.18 <-
  reEvent = ev_B
  mySlot1.state = sl1_C
-> State: 1.19 <-
```

F.4 Communication exclusion

The global concurrent exclusion rule might put unnecessary strict constraints on the schedule. It is often reasonable to limit the scope of these rules to those FPGAModules who could interact with each other through a communication network (see D.2.3).

F.4.1 Security rule

The term prohibits communication between FPGAModules from Set A and Set B:

```
Rule name:
            prohibits
            communication of
                  FPGAMODULE_SET_A
            with
                  FPGAMODULE_SET_B
;
```

F.4.2 LTL pseudocode

```
For each edge e in the communication graph:
  G!({ any module from FPGAMODULE_SET_A utilizing source_slot_of(e)}
  & {any module from FPGAMODULE_SET_B utilizing target_slot_of(e) })
```

F.4.3 Scenario

An FPGA design consists of two independent sections. The first section performs operations of lower security concerns and consists of Slot 3. The two modules {G, H} that utilize this Slot are of low and medium trustworthiness, respectively. In the second, the more sensitive part has modules performing operations of high sensitivity with high trustworthiness (Modules A, D, and E) and modules with only medium trustworthiness (Modules B, C). The design must prevent modules with low trustworthiness from endangering sensitive operations. Mixed utilization of the Slots is permitted as long as they cannot communicate with each other. Therefore, the modules of the groups {A, D, E}, and {B, C, F, G} must never be able to communicate with each other. Module F is regarded as highly trustworthy and poses neither a threat to the high sensitivity objects nor is its operation critical enough to cause serious harm if attacked.

F.4.4 FPGASECML-Model

```
FPGAArchitecture communicationModel

FPGAModules:
            Processing Block: sl1_A
                  utilizes slot: Slot1;
                  provides sensitive services:        { none };
                  contains sensitive data:            { none };
                  bitstream is protected by:          { none };
                  implements security mechanism:      { none } ;
                  bitstream storage:                  bstore1 ;
                  trustworthiness:                    { high };
                  assigned security roles:      { rHighSensitivityOp };
                  ;

      Processing Block: sl2_B
                  utilizes slot: Slot2;
                  provides sensitive services:        { none };
                  contains sensitive data:            { none };
                  bitstream is protected by:          { none };
```

```
                implements security mechanism:      { none };
                bitstream storage:                  bstore1;
                trustworthiness:                    { medium };
                assigned security roles:            { none };
                ;

    Processing Block: sl1_C
                utilizes slot: Slot1;
                provides sensitive services:        { none };
                contains sensitive data:            { none };
                bitstream is protected by:          { none };
                implements security mechanism:      { none } ;
                bitstream storage:                  bstore1;
                trustworthiness:                    { medium };
                assigned security roles:            { none };
                ;


    Processing Block: sl2_D
                utilizes slot: Slot2;
                provides sensitive services:        { none };
                contains sensitive data:            { none };
                bitstream is protected by:          { none };
                implements security mechanism:      { none };
                bitstream storage:                  bstore1;
                trustworthiness:                    { high };
                assigned security roles:      { rHighSensitivityOp };
                ;

  Processing Block: sl2_E
                utilizes slot: Slot2;
                provides sensitive services:        { none };
                contains sensitive data:            { none };
                bitstream is protected by:          { none };
                implements security mechanism:      { none };
                bitstream storage:                  bstore1 ;
                trustworthiness:                    { high };
                assigned security roles:      { rHighSensitivityOp };
                ;

  Processing Block: sl1_F
                utilizes slot: Slot1;
                provides sensitive services:        { none };
                contains sensitive data:            { none };
                bitstream is protected by:          { none };
                implements security mechanism:      { none } ;
                bitstream storage:                  bstore1 ;
                trustworthiness:                    { high };
                assigned security roles:      { rLowSensitivityOp };
                ;
```

```
    Processing Block: sl3_G
                  utilizes slot: Slot3;
                  provides sensitive services:        { none };
                  contains sensitive data:            { none };
                  bitstream is protected by:          { none };
                  implements security mechanism:      { none } ;
                  bitstream storage:                  bstore1 ;
                  trustworthiness:                    { low };
                  assigned security roles:            { none };
                  ;

     Processing Block: sl3_H
                  utilizes slot: Slot3;
                  provides sensitive services:        { none };
                  contains sensitive data:            { none };
                  bitstream is protected by:          { none };
                  implements security mechanism:      { none } ;
                  bitstream storage:                  bstore1 ;
                  trustworthiness:                    { medium };
                  assigned security roles:            { none };
                  ;

FPGAResources:
      Slot Slot1 { };
      Slot Slot2 { };
      Slot Slot3 { };


Communication:
  Bus b1 {Slot1,Slot2};

Reconfiguration:
      Events:
                  Event evInit      loads: { sl1_A, sl2_D,sl3_G };//
triggered by peripheral device or processing Block transition to
configuration IDentifier
                  Event ev_A        loads: { sl1_A };
                  Event ev_B        loads: { sl2_B };
                  Event ev_C        loads: { sl1_C };
                  Event ev_D        loads: { sl2_D };
                  Event ev_E        loads: { sl2_E };
                  Event ev_F        loads: { sl1_F };
                  Event ev_G        loads: { sl3_G };
                  Event ev_H        loads: { sl3_H };

      Sequence valid_seq      { ev_E,ev_H,ev_F,ev_G,ev_B,ev_C}
      Sequence invalid_seq    { ev_H,ev_C }
      Sequence valid_seq2     { ev_H,ev_G }

Security Policy:
```

```
      Rule util_1 :
      prohibits communication of
      {
            has security roles: { rHighSensitivityOp };
      }
      with
      {
            has trustworthiness: { medium,low };
      }
      ;
```

F.4.5 NuSMV Model to validate the sequence { ev_E,ev_H,ev_F,ev_G,ev_B,ev_C}

This NuSMV model validates the reconfiguration sequence { ev_E,ev_H,ev_F,ev_G,ev_B,ev_C} The modules F and H are not constrained by any security rule and are therefore not incorporated inti the LTL rule:

```
MODULE main
VAR
      reEvent :{evInit,ev_A,ev_B,ev_C,ev_D,ev_E,ev_F,ev_G,ev_H};
      state : { state_0, state_1, state_2, state_3, state_4, state_5};
      mySlot1 : Slot1(reEvent);
      mySlot2 : Slot2(reEvent);
      mySlot3 : Slot3(reEvent);
ASSIGN
      init(state) :=    state_0;
      init(reEvent) :=ev_E;
      next(state) := case
            state=state_0  : state_1;
            state=state_1  : state_2;
            state=state_2  : state_3;
            state=state_3  : state_4;
            TRUE  : state;
      esac;
      next(reEvent):= case
            state=state_0     : ev_H;
            state=state_1     : ev_F;
            state=state_2     : ev_G;
            state=state_3     : ev_B;
            state=state_4     : ev_C;
            TRUE  : reEvent;
      esac;

--util_1
LTLSPEC G ! ((
(mySlot1.state = sl1_A) & (mySlot2.state = sl2_B))--b1 |
((mySlot2.state = sl2_D | mySlot2.state = sl2_E) & (mySlot1.state =sl1_C))-
-b1
)
```

```
MODULE Slot1 (reEvent)
VAR
      state : {noState1,sl1_F,sl1_C,sl1_A,noState2};
ASSIGN
      init(state) :=    sl1_A ;
      next(state) :=    case
            reEvent =           ev_F :      sl1_F;
            reEvent =           ev_C :      sl1_C;
            reEvent =           ev_A :      sl1_A;
            TRUE : state ;
      esac;


MODULE Slot2 (reEvent)
VAR
      state : {noState1,sl2_B,sl2_E,sl2_D,noState2};
ASSIGN
      init(state) :=    sl2_D ;
      next(state) :=    case
            reEvent =           ev_B :      sl2_B;
            reEvent =           ev_E :      sl2_E;
            reEvent =           ev_D :      sl2_D;
            TRUE : state ;
      esac;


MODULE Slot3 (reEvent)
VAR
      state : {noState1,sl3_G,sl3_H,noState2};
ASSIGN
      init(state) :=    sl3_G ;
      next(state) :=    case
            reEvent =           ev_G :      sl3_G;
            reEvent =           ev_H :      sl3_H;
            TRUE : state ;
      esac;
```

Running the model through NuSMV confirms the validity of the sequence:

```
-- specification G !((mySlot1.state = sl1_A & mySlot2.state = sl2_B) |
((mySlot2.state = sl2_D | mySlot2.state = sl2_E) & mySlot1.state = sl1_C))
is true
```

F.5 LTL rules of the FPGASECML-example

The example presented in 7.5 is converted into these LTL-Rules:

```
--rule1_communication
LTLSPEC G ! (
((mySlot1.state = ioConfigurationUnit) &
(mySlot2.state = pbAESUnit | mySlot2.state = pbDSP
                  | mySlot2.state = pbInitS2))--nwBUS1
 | ((mySlot1.state = ioConfigurationUnit) &
(mySlot3.state = ioEthernet | mySlot3.state = pbFPU
      | mySlot3.state = pbInitS3))--nwBUS1
 | ((mySlot1.state = ioConfigurationUnit) &
      (mySlot4.state = ioSensor | mySlot4.state = pbInitS4))--nwBUS1
)--rule2_peprocessor_after
LTLSPEC (TRUE)

--rule2_preprocessor_before
LTLSPEC (TRUE)

--rule3_cfEncryptedcommunication_Before_cfSensor
LTLSPEC G ((mySlot3.state = ioEthernet) ->! H (mySlot3.state = pbFPU))

--rule4_IOBLOCKafterAES
LTLSPEC (TRUE)
```

Smart scheduling can only resolve the violation of Rule 3. Rule 1 requires, as mentioned earlier, a dedicated network between Slot0 and Slot1 after which this rule becomes:

```
--rule1_communication
LTLSPEC (TRUE)
```

The security policy compliant schedule can be derived after the communication infrastructure is changed:

```
-- specification !((((((((((((TRUE & TRUE) & TRUE) &  G (mySlot3.state =
pbFPU -> !( F mySlot3.state = ioEthernet))) & TRUE) &  F mySlot3.state =
ioEthernet) &  F mySlot1.state = ioConfigurationUnit) &  F mySlot4.state =
ioSensor) &  F mySlot0.state = pbProcessor) &  F mySlot2.state = pbAESUnit)
&  F mySlot3.state = pbFPU) &  F mySlot2.state = pbDSP) &  F mySlot2.state
= pbInitS2) &  F mySlot3.state = pbInitS3) &  F mySlot4.state = pbInitS4)
is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    reEvent = evInit
    mySlot0.state = pbProcessor
    mySlot1.state = ioConfigurationUnit
    mySlot2.state = pbInitS2
```

```
    mySlot3.state = pbInitS3
    mySlot4.state = pbInitS4
  -> State: 1.2 <-
    reEvent = evDSP
  -> State: 1.3 <-
    mySlot2.state = pbDSP
  -> State: 1.4 <-
    reEvent = evCrypto
  -> State: 1.5 <-
    reEvent = evSensor
    mySlot2.state = pbAESUnit
    mySlot3.state = ioEthernet
  -- Loop starts here
  -> State: 1.6 <-
    reEvent = evInit
    mySlot3.state = pbFPU
    mySlot4.state = ioSensor
  -- Loop starts here
  -> State: 1.7 <-
  -> State: 1.8 <-
```

Appendix G FPGASECMLcode for the introductory example

This appendix presents the FPGASECML code for the introductory example in chapter 7

```
FPGAArchitecture example1
FPGAModules:
Processing Block: pbProcessor
      utilizes slot: Slot0 ;
      provides sensitive services:  {
            sensitive service encryptedconnection
                  of sensitivity level high ;
            sensitive service reconfigurationcontrol
                  of sensitivity level high ;
            } ;
      contains sensitive data:  {
            sensitive data sharedsecret
                  of sensitivity level high
                  access type read ;
            sensitive data software
            of sensitivity level high
            access type read ;
      } ;
      bitstream is protected by:  { tamper_resistant_storage } ;
      implements security mechanism: { none } ;
      bitstream storage: ioConfigurationUnit ;
      ;

IO Block:  ioConfigurationUnit
      utilizes slot: Slot1;
      connected to: (to be determined) ;
      connects to threatlayer:  PCB;
      availability: permanent ;
      provides sensitive services:
      {
            sensitive service loadbitstream
            of sensitivity level high;
      } ;
      contains sensitive data:
      {
            sensitive data ConfigurationData
            of sensitivity level high
            access type read;
      } ;
      bitstream is protected by:  { tamper_resistant_storage } ;
      implements peripheral security mechanism: { none } ;
      implements security mechanism: { none } ;
      bitstream storage: ioConfigurationUnit;
      ;

Processing Block: pbAESUnit
      utilizes slot: Slot2  ;
```

```
      provides sensitive services:  {
            sensitive service encryption
                  of sensitivity level high  ;
            sensitive service decryption of sensitivity level high ;
      } ;
      contains sensitive data:  {
            sensitive data aeskey of
                  sensitivity level high
                  access type write;
            sensitive data aesplaintext of
                  sensitivity level high
                  access type rw
                  stored in memory Slot2.bram1;
            sensitive data aesciphertext
                  of sensitivity level medium
            access type rw ;
      } ;
      bitstream is protected by:  { tamper_resistant_storage } ;
      implements security mechanism: { none } ;
      bitstream storage: ioConfigurationUnit ;
      ;

Processing Block: pbDSP
      utilizes slot: Slot2  ;
      provides sensitive services:  { none } ;
      contains sensitive data:  {
            sensitive data sensordata
            of sensitivity level high
            access type rw
            stored in memory Slot2.bram1;
            };
      bitstream is protected by:  { tamper_resistant_storage } ;
      implements security mechanism: { none } ;
      bitstream storage: ioConfigurationUnit;
      ;

Processing Block: pbFPU
      utilizes slot: Slot3  ;
      provides sensitive services:  { none } ;
      contains sensitive data:  { none } ;
      bitstream is protected by:  { tamper_resistant_storage } ;
      implements security mechanism: { none } ;
      bitstream storage: ioConfigurationUnit;
      ;

IO Block: ioEthernet
      utilizes slot: Slot3;
      connected to: (to be determined) ;
      connects to threatlayer:  WORLD;
      availability: permanent ;
      provides sensitive services:  { none } ;
      contains sensitive data:  { none } ;
```

```
      bitstream is protected by:
            {tamper_resistant_storage } ;
      implements peripheral security mechanism:
            { data_sanitization , data_authentication } ;
      implements security mechanism: { none } ;
            bitstream storage: ioConfigurationUnit;
      ;
IO Block:  ioSensor
      utilizes slot: Slot4;
      connected to: (to be determined) ;
      connects to threatlayer:  PCB ;
      availability: permanent ;
      provides sensitive services:  { none } ;
      contains sensitive data:
      {
            sensitive data sensordata
            of sensitivity level low
            access type read;
            } ;
      bitstream is protected by:
            { tamper_resistant_storage } ;
      implements peripheral security mechanism: { none } ;
      implements security mechanism: { none } ;
      bitstream storage: ioConfigurationUnit;
      ;

Processing Block: pbInitS2
      utilizes slot: Slot2  ;
      provides sensitive services:  { none } ;
      contains sensitive data:  { none } ;
      bitstream is protected by:  { tamper_resistant_storage } ;
      implements security mechanism: { none } ;
      bitstream storage: ioConfigurationUnit;
      ;

Processing Block: pbInitS3
      utilizes slot: Slot3  ;
      provides sensitive services:  { none } ;
      contains sensitive data:  { none } ;
      bitstream is protected by:  { tamper_resistant_storage } ;
      implements security mechanism: { none } ;
      bitstream storage: ioConfigurationUnit ;
      ;

Processing Block: pbInitS4
      utilizes slot: Slot4  ;
      provides sensitive services:  { none } ;
      contains sensitive data:  { none } ;
      bitstream is protected by:  { tamper_resistant_storage } ;
      implements security mechanism: { none } ;
      bitstream storage: ioConfigurationUnit;
      ;
```

```
FPGAResources:
      Slot Slot0 { };
      Slot Slot1 { };
      Slot Slot2 { Memory bram1; };
      Slot Slot3 { };
Slot Slot4 { };

Communication:
      Bus nwBUS1 {  Slot0, Slot1, Slot2 , Slot3 , Slot4  } ;

Reconfiguration:
      Events:
            Event evInit loads:
                  {ioConfigurationUnit,pbProcessor,pbInitS2,
                  pbInitS3,pbInitS4};
            Event evCrypto triggered by { pbProcessor }
                  loads: { ioEthernet,pbAESUnit };
            Event evSensor triggered by  { pbProcessor }
                  loads: { ioSensor,pbFPU } ;
            Event evDSP triggered by  { pbProcessor }
                  loads: {pbDSP};
Security Policy:
      //1.  No element but pbProcessor shall communicate
      //    with the ioConfigurationUnit
      Rule rule1_communication :
      prohibits communication of{
            FPGAModule: { ioConfigurationUnit };
      }
      with
      {
            NOT FPGAModule: { pbProcessor};
      }
      ;

      // 2. No element shall utilize Slot 0 except pbProcessor
      Rule rule2_peprocessor_after :
      prohibits utilization of
      {
            NOT FPGAModule: { pbProcessor};
      }
      after
      {
            FPGAModule: { pbProcessor};
      }
      ;

      Rule rule2_preprocessor_before :
      prohibits utilization of
      {
            NOT FPGAModule: { pbProcessor };
      }
```

```
      before
      {
            FPGAModule:  { pbProcessor };
      }
      ;

      //3.  Encrypted communication must take place before sensor operation
      Rule rule3_cfEncryptedcommunication_Before_cfSensor   :
      prohibits utilization of{
            reconfiguration event: {  evSensor };
      }
      before
      {
            reconfiguration event: {  evCrypto };
      }
      ;

      //4.  To prevent the leakage of sensitive data the FPGA primitives
      //    utilized by the encryption unit shall not be reutilized by
      //    any IO Block
      Rule rule4_IOBLOCKafterAES:
      prohibits utilization of{
            connects to threatlayer: { ANY };
      }
      after
      {
            FPGAModule: { pbAESUnit };
      };
```

Appendix H Reinforcement Learning Example

This section introduces a small example to illustrate how reinforcement learning can be used to identify weaknesses in the design.

H.1 Reinforcement Learning Scenarios

An FPGA design (Figure 42) consists of three Slots with two FPGAModules each:

- Slot1: A, C
- Slot2: B, D
- Slot3: E, F

FPGAModule A is the only IO Block. The run ends when the agent has exfiltrated the sensitive data of all 6 FPGAModules into the outside world. An agent gets a reward of 10.000 for reaching a terminal state and -1.000 for every other action.

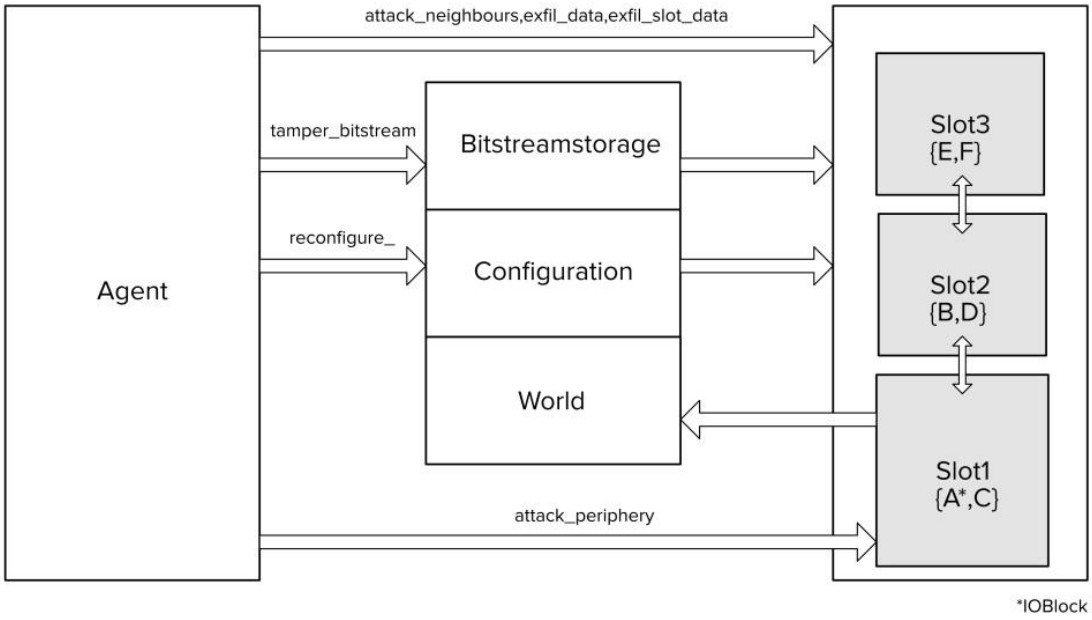


Figure 42 Schematic of the FPGASECML to burlap example

H.2 Reinforcement Learning Scenarios

Meaningful analysis requires multiple experiments with different parameters. Each scenario is a combination of MDP parameters like the reward, the success likelihood, and the costs of an action. They also include hyperparameters, like the learning rate. FPGASECML allows the definition of scenarios that allow the independent definition and evaluation of these parameters. All scenarios share a set of parameters like γ and the exploration-exploitation trade-off ε. The example in the next section illustrates the application of scenarios.

H.2.1 Scenario 1: Baseline

Scenario 1 establishes a baseline of the model through a deterministic MDP (the success rate of every action is 1) and a fixed reward/punishment assignment. The agent gets a reward of 200.000 for reaching a terminal state and a punishment of -1.000 for every other action. All three runs return a minimal solution with 24 actions, the histogram (Figure 44) of all three actions shows that the epsilon of 0.1 has the most episodes with the minimal steps followed by 0.05 (low exploration rate requires more steps to find a solution) and 0.25 (too much exploration hinders efficient exploitation of the available data.) The learning process starts with episodes well over 2.000 steps (Figure 43) and provides better results as the Qtable gets filled. For an ϵ of 0.1 the first minimal solution is found after just 303.323 Episodes - indicating that the agent has either found the best possible solution or is stuck in a local minimum with little chances to break out.

It is also notable that all three solutions make extensive use of the tamper_bitstream-action, but what if this action is unlikely to succeed or prohibitively expensive?

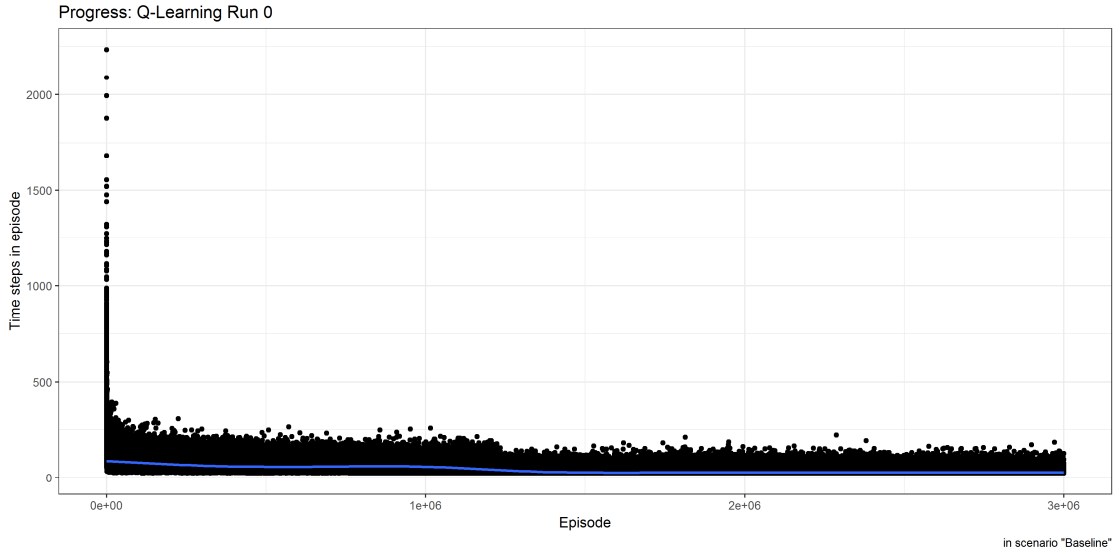


Figure 43 Learning progress for the first run of the baseline scenario with an ϵ of 0.1 , each dot represents one episode

H.2.2 Scenario 2: Tamper resistance storage prevents most attacks}

The second scenario assumes that only one in a thousand attacks against tamper-resistant bitstream storage is successful (relative to all other attacks as success factors are normalized.) The three runs return a minimal sequence of 25 actions each, and all three of them avoid the tamper_bitsteam actions altogether. The histogram (Figure 45) shows a smaller spread than in the first scenario. The median of the episodes for each run shrunk from (29,39,44) to (28,26,35).

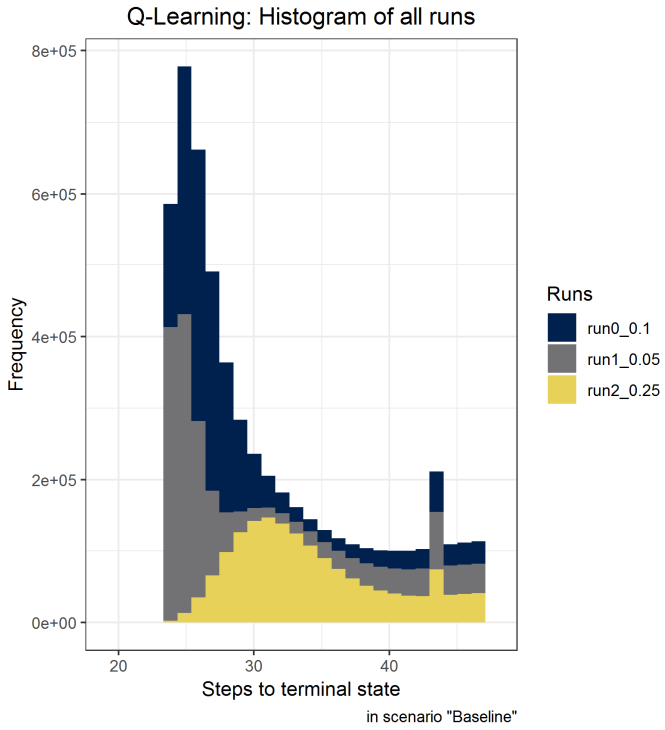


Figure 44 Histogram for the baseline scenario with an ϵ of 0.1. 0.05 and 0.25

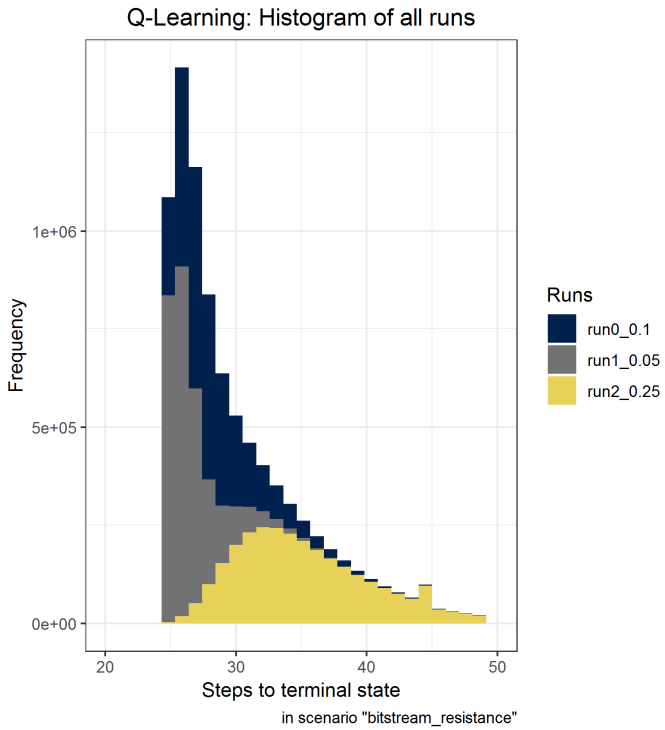


Figure 45 Histogram for all three runs of the second scenario (truncated on the right)

H.2.3 Scenario 3: Faulty Encryption

It is assumed that weaker tamper protection of the bitstream storage and weakness in the bitstream encryption mechanism increases the success rate of tamper_bitstream actions. The minimal sequences

returned require 27 steps for epsilon of 0.1, 26 for 0.05, and 30 for 0.25 - a significantly worse performance than in the previous scenarios. The histogram of all three runs also skews much more to the right than those of the above scenarios. Worse, the three attack sequences bitstreams tampered with 4 of the 6. The success rate of one in a thousand was, apparently, low enough to discourage the agent from using these actions while the "one in five" -chance is too weak to achieve a similar effect. The low success rate is, however, strong enough to prolong the learning process as the median number of steps per episode rises from previously (28, 26, 35) to (154, 155, 152). The learning progress (Figure 46) is much noisier than it was in the baseline scenario (Figure 43.)

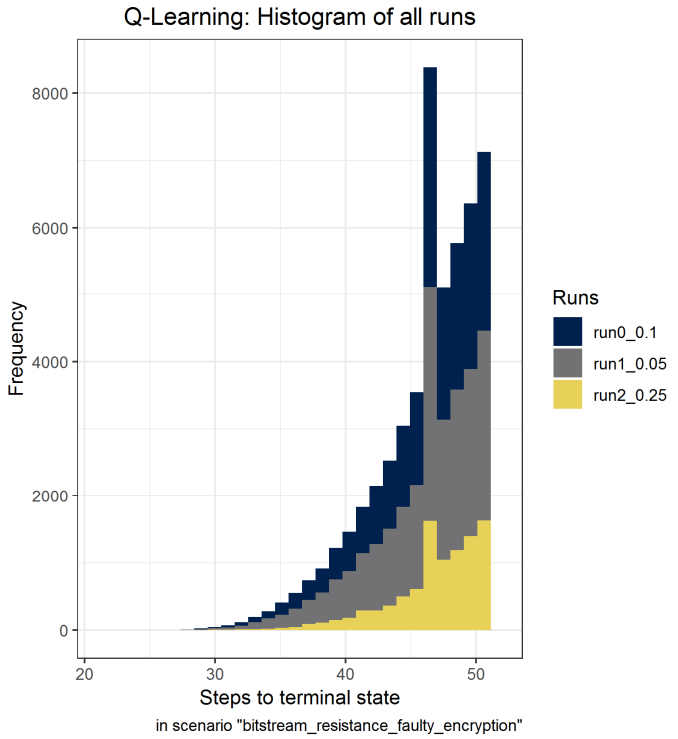


Figure 46 Histogram for all three runs of the third scenario (truncated on the right)

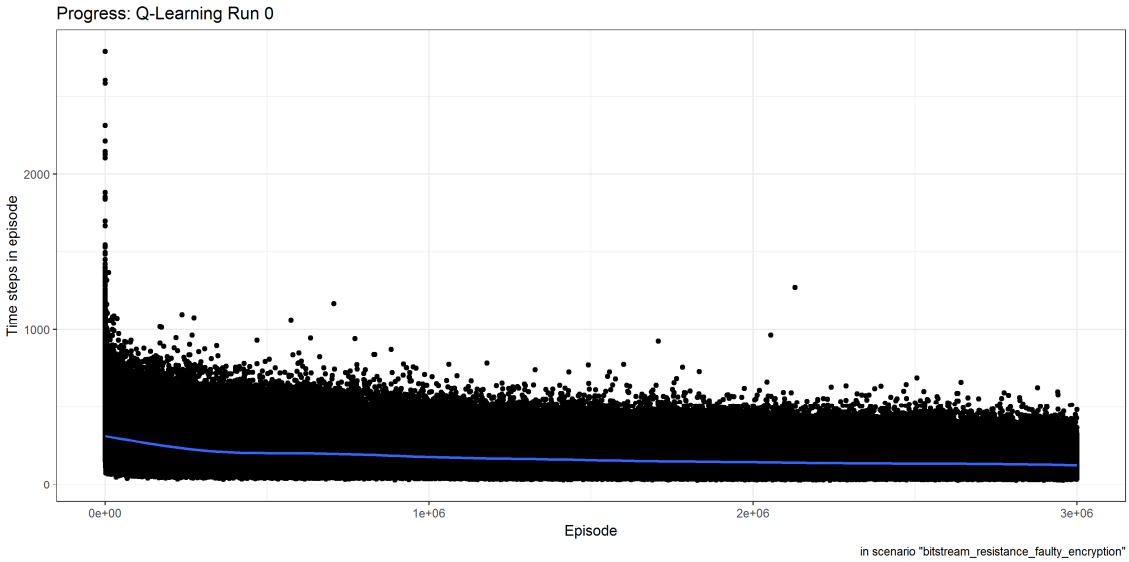


Figure 47 Learning progress for the first run of the fourth scenario, each dot represents one episode

H.2.4 Scenario 4: Physical attacks against storage devices are expensive

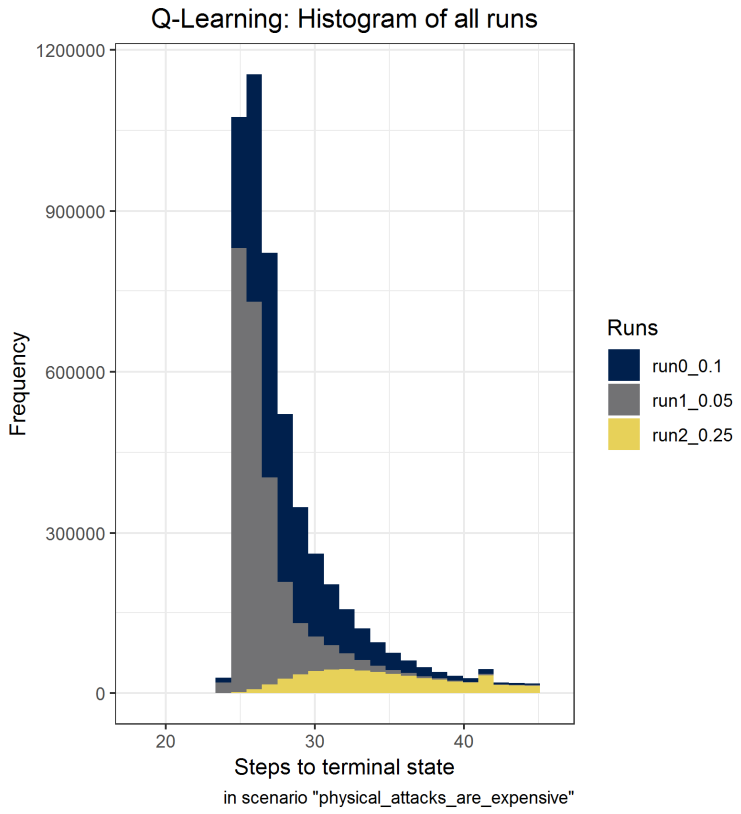


Figure 48 Histogram for all three runs of the fourth scenario (truncated on the right)

Not all attacks require the same amount of resources - some take longer than others; a few require costly equipment and highly specialized threat agents. Reducing the success rate of an attack makes it implicitly more expensive (as, on average, more attacks are necessary before succeeding), but with several million trials per run, there is a good chance that at least one sequence of highly improbable actions will succeed at least once. A real attacker might not want to take this risk. The MDP is converted back into a DMDP, and the tamper_bitstream action gets an additional cost weight of 100.000 (additional to the default negative reward of -1.000.) The minimal sequence for the epsilons 0.1 and 0.25 is 24. The low exploration rate of 0.05 has led to a global minimum of 23. It is also notable that this global minimum required 2.543.660th episode and that the agent tampers with two bitstream storages while the other two use this expensive operation only once. Imposing even higher costs (bitstream_tampering_extremely_expensive) on the tamper_bitstream operation had no positive effect, with all three runs returning a sequence of the length 24.

H.2.5 Results

The approach presented here found a reasonable solution to this problem within a feasible amount of computations. The noisy progress of the RL algorithm makes it impossible to determine whether this minimum is local or global. Scenarios can help to assess the impact of high costs and low success rates and to find better solutions by constraining the size of the search space. A plausibility check of the results (here performed on the generated sequence) is mandatory, as for all machine learning methods. The number of training episodes required for this small example indicates challenges for intricate designs and models with a richer state, action representation. The memory consumption of the program, presumably driven by the expanding Qtable, supports this

assumption. Finding the appropriate cost/reward structure remains a challenge, and any analysis should include multiple scenarios with different cost and success rate values.

Appendix I Table of Figures

Appendix J References

[1] National Audit Office (NAO), *Investigation: WannaCry cyber attack and the NHS*. [Online]. Available: https://www.nao.org.uk/report/investigation-wannacry-cyber-attack-and-the-nhs/ (accessed: Apr. 7 2018).

[2] BBC, *Ukraine power cut 'was cyber-attack'*. [Online]. Available: http://www.bbc.com/news/technology-38573074 (accessed: Apr. 7 2018).

[3] *Exclusive: Bangladesh probes 2013 hack for links to central bank heist*. [Online]. Available: https://www.yahoo.com/news/exclusive-bangladesh-probes-2013-hack-links-central-bank-201427456--sector.html?ref=gs (accessed: Apr. 7 2018).

[4] R. Leszczyna, *Cybersecurity in the Electricity Sector: Managing Critical Infrastructure,* 1st ed. Cham, Cham: Springer International Publishing; Springer, 2019.

[5] A. Gupta, *The IoT hacker's handbook: A practical guide to hacking the Internet of Things*. [New York, New York], New York, NY: Apress; Distributed by Springer Science+Business Media New York, 2019.

[6] C. Hadnagy, *Social engineering: The art of human hacking*. Hoboken, NJ: Wiley, 2011. [Online]. Available: http://site.ebrary.com/lib/academiccompletetitles/home.action

[7] Microsoft, *Have you checked the Java?* [Online]. Available: http://blogs.technet.com/b/mmpc/archive/2010/10/18/have-you-checked-the-java.aspx

[8] bushing, marcan, segher, sven, *Console Hacking 2010: PS3 Epic Fail*. [Online]. Available: http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf

[9] N. Falliere, L. O. Murchu, and E. Chien, *W32.Stuxnet Dossier*. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

[10] K. Zetter, *Hard-Coded Password and Other Security Holes Found in Siemens Control Systems*. [Online]. Available: http://www.wired.com/threatlevel/2011/08/siemens-hardcoded-password/

[11] *How the PS3 hypervisor was hacked | rdist on WordPress.com*. [Online]. Available: https://rdist.root.org/2010/01/27/how-the-ps3-hypervisor-was-hacked/ (accessed: Mar. 3 2018).

[12] Micrososft, *Benefits of the SDL*. [Online]. Available: http://www.microsoft.com/security/sdl/about/benefits.aspx

[13] C. Reis, A. Barth, and C. Pizano, "Browser security: lessons from Google Chrome," *Commun. ACM*, vol. 52, no. 8, 45-49, 2009. [Online]. Available: http://doi.acm.org/10.1145/1536616.1536634

[14] M. Kucera and M. Vetter, "FPGA Rootkits: - hiding malicous code inside the hardware," in *Solutions in Embedded Systems (WISES '07) Solutions in Embedded Systems (WISES '07)*, 2007.

[15] Michael Vetter, "Security Implications of FPGAs in Embedded Systems," University of West Bohemia in Pilsen, Plzeň.

[16] NetFPGA, *NetFPGA*. [Online]. Available: http://netfpga.org/

[17] P. Ekas, *Keys to reconfigurable SDR system design*. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1271957

[18] R. J. Anderson, *Security engineering: a guide to building dependable distributed systems*. Indianapolis, Ind: Wiley, 2008. [Online]. Available: http://www.loc.gov/catdir/enhancements/fy0827/2008006392-d.html

[19] Ellen Nakashima, *Israel hacked Kaspersky, then tipped the NSA that its tools had been breached*. [Online]. Available: https://www.washingtonpost.com/world/national-security/israel-hacked-kaspersky-then-tipped-the-nsa-that-its-tools-had-been-breached/2017/10/10/d48ce774-aa95-11e7-850e-2bdd1236be5d_story.html?utm_term=.70f7aadfad78 (accessed: Feb. 28 2018).

[20] NICOLE PERLROTH, *How Antivirus Software Can Be Turned Into a Tool for Spying*. [Online]. Available: https://www.nytimes.com/2018/01/01/technology/kaspersky-lab-antivirus.html (accessed: Feb. 28 2018).

[21] T. Fox-Brewster, *Is It Lights Out For Kaspersky After Latest NSA Disaster?* [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2017/10/06/kaspersky-nsa-russia-hack-mess/ (accessed: Apr. 4 2018).

[22] *"WannaCry" ransomware attack losses could reach $4 billion*. [Online]. Available: https://www.cbsnews.com/news/wannacry-ransomware-attacks-wannacry-virus-losses/ (accessed: Apr. 4 2018).

[23] K. Collins, *The hackers behind the WannaCry ransomware attack have finally cashed out*. [Online]. Available: https://qz.com/1045270/wannacry-update-the-hackers-behind-ransomware-attack-finally-cashed-out-about-140000-in-bitcoin/ (accessed: Apr. 4 2018).

[24] A. Greenberg, *Sandworm: A new era of cyberwar and the hunt for the Kremlin's most dangerous hackers*. New York: Doubleday, 2019.

[25] P. Kocher *et al.,* "Spectre Attacks: Exploiting Speculative Execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[26] M. Lipp *et al.,* "Meltdown: Reading Kernel Memory from User Space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[27] *And the Winner of Best FPGA of 2016 is… | EE Times*. [Online]. Available: https://www.eetimes.com/author.asp?section_id=36&doc_id=1331443 (accessed: Mar. 18 2018).

[28] R. K. Soni, N. Steiner, and M. French, "Open-Source Bitstream Generation," in *Proceedings of the 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, 2013, 105-112 numPG - 8. [Online]. Available: http://dx.doi.org/10.1109/FCCM.2013.45

[29] *SymbiFlow - the GCC of FPGAs*. [Online]. Available: https://symbiflow.github.io/ (accessed: Feb. 3 2020).

[30] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Proceedings of the 14th international conference on Cryptographic Hardware and Embedded Systems*, 2012, 23-40 numPG - 18. [Online]. Available: www.cl.cam.ac.uk/~sps32/ches2012-backdoor.pdf?

[31] F. Rodriguez-Henriquez, *Cryptographic algorithms on reconfigurable hardware*. New York, NY: Springer, 2006.

[32] M. Smerdon, *Security Solutions Using Spartan-3 Generation FPGAs*. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp266.pdf

[33] Intel, "AN 556: Using the Design Security Features in Intel FPGAs,"

[34] S. Drimer, "Authentication of FPGA bitstreams: why and how," in *Applied Reconfigurable Computing*, 2007, pp. 73–84. [Online]. Available: http://www.cl.cam.ac.uk/~sd410/papers/bsauth.pdf

[35] I. Xilinx, "Using Encryption to Secure a 7 Series FPGA Bitstream Application Note (XAPP1239)," [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp1239-fpga-bitstream-encryption.pdf

[36] T. Eisenbarth, T. Güneysu, C. Paar, A.-R. Sadeghi, D. Schellekens, and M. Wolf, "Reconfigurable trusted computing in hardware," in *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, 2007, pp. 15–20. [Online]. Available: http://www.crypto.ruhr-uni-bochum.de/imperia/md/content/texte/publications/conferences/fpga_tpm.pdf

[37] Microsemi, "Overview of Secure Boot with Microsemi SmartFusion2 FPGAs," [Online]. Available: http://www.microsemi.com/document-portal/doc_view/132874-overview-of-secure-boot-with-microsemi-smartfusion2-fpgas

[38] I. Xilinx, "Secure Boot of Zynq-7000 All Programmable SoC Application Note (XAPP1175)," [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp1175_zynq_secure_boot.pdf

[39] L. Bossuet, G. Gogniat, and W. Burleson, "Dynamically configurable security for SRAM FPGA bitstreams," in *18th International Parallel and Distributed Processing Symposium*, 2004, p. 146.

[40] E. Simpson and P. Schaumont, "Offline Hardware/Software Authentication for Reconfigurable Platforms: 8th international workshop, Yokohama, Japan, October 10-13, 2006 ; proceedings," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, Berlin [u.a.]: Springer, 2006, pp. 311–323. [Online]. Available: http://www.springerlink.com/content/f76px727174j2527

[41] T. Kean, "Cryptographic rights management of FPGA intellectual property cores," in *FPGA '02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, 2002, pp. 113–118. [Online]. Available: http://www.algotronix.com/content/security%20fpga2002.pdf

[42] Altera, *FPGA design security solution using MAX II devices*: Altera Corp. [Online]. Available: http://www.altera.com/literature/wp/wp_m2dsgn.pdf

[43] K. Chapman, *Low Cost Design Authentication for Spartan-3E FPGAs*. [Online]. Available: http://www.xilinx.com/products/boards/s3estarter/files/s3esk_authentication.pdf

[44] K. Chapman, *Reading Spartan-3A Device DNA: A Reference Design for the Spartan-3A Starter Kit*. [Online]. Available: http://www.xilinx.com/products/boards/s3astarter/files/s3ask_dna_reader.pdf

[45] Jain, Adarsh K., Yuan, Lin, Pari, Pushkin R., and Qu, Gang, "Zero overhead watermarking technique for FPGA designs," in *GLSVLSI '03: Proceedings of the 13th ACM Great Lakes symposium on VLSI*, 2003, pp. 147–152.

[46] P. J. Ashenden and J. Lewis, *VHDL-2008: just the new stuff*. Amsterdam [u.a.]: Elsevier/Morgan Kaufmann, 2008.

[47] *IEEE 1735-2014 - IEEE Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP)*. [Online]. Available: https://standards.ieee.org/findstds/standard/1735-2014.html (accessed: Feb. 13 2018).

[48] T. Huffmire *et al.,* "Moats and Drawbridges: An Isolation Primitive for Reconfigurable Hardware Based Systems," in *Proceedings of the 2007 IEEE 2007using tusing*, 281-295.

[49] Xilinx, "Xilinx XAPP1085 7 Series Isolation Design Flow Lab Using ISE Design Suite 14.4," [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp1085-7s-isolation-design-flow-ise-14-4.pdf

[50] Xilinx, *Isolation Design Flow*. [Online]. Available: http://www.xilinx.com/applications/isolation-design-flow.html (accessed: Jun. 16 2016).

[51] D. Koch, C. Beckhoff, and J. Teich, "ReCoBus-Builder; A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAS," in *2008 International Conference on Field Programmable Logic and Applications*, 2008, pp. 119–124.

[52] C. Beckhoff, D. Koch, and J. Torresen, "Go Ahead: A Partial Reconfiguration Framework," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, 2012, pp. 37–44.

[53] I. Xilinx, "Xilinx XAPP290 Difference Based Partial Reconfiguration Application Note," [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp290.pdf

[54] S. Drimer and M. G. Kuhn, "A Protocol for Secure Remote Updates of FPGA Configurations," in *Reconfigurable Computing: Architectures, Tools and Applications, 5th International Workshop, ARC 2009*, 2009, pp. 50–61.

[55] Xilinx, *SECURITY MONITOR IP*. [Online]. Available: http://www.xilinx.com/support/documentation/product-briefs/security-monitor-ip-core-product-brief.pdf (accessed: Jun. 16 2016).

[56] W. Hu *et al.,* "Gate-Level Information Flow Tracking for Security Lattices," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 1, 2:1-2:25, 2014, doi: 10.1145/2676548.

[57] W. Hu, B. Mao, J. Oberg, and R. Kastner, "Detecting Hardware Trojans with Gate-Level Information-Flow Tracking," *Computer*, vol. 49, no. 8, pp. 44–52, 2016, doi: 10.1109/MC.2016.225.

[58] A. Chhotaray, A. Nahiyan, T. Shrimpton, D. Forte, and M. Tehranipoor, "Standardizing Bad Cryptographic Practice," in *CCS'17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security ; Oktober 30 - November 3, 2017, Dallas, TX, USA*, Dallas, Texas, USA, 2017, pp. 1533–1546.

[59] M. Vetter, "MODEL-BASED SECURITY ANALYSIS OF FPGA DESIGNS THROUGH REINFORCEMENT LEARNING," *Acta Polytech*, vol. 59, no. 5, pp. 518–526, 2019, doi: 10.14311/AP.2019.59.0518.

[60] M. Brambilla, J. Cabot, and M. Wimmer, *Model-driven software engineering in practice*: Morgan & Claypool Publishers, 2017.

[61] L. Delligatti, *SysML distilled: A brief guide to the systems modeling language*: Addison-Wesley Professional, 2014.

[62] Julein Delange, *AADL IN PRACTICE*. [S.l.]: REBLOCHON DEVELOPMENT CO, 2017.

[63] Robert Ellison, Allen Householder, John Hudak, Rik Kazman, Carol Woody, *Extending AADL for Security Design Assurance of Cyber-Physical Systems*. [Online]. Available: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2015_005_001_449522.pdf (accessed: Feb. 7 2019).

[64] F. Swiderski and W. Snyder, *Threat modeling*. Redmond, Wash: Microsoft Press, 2004. [Online]. Available: http://www.gbv.de/dms/bowker/toc/9780735619913.pdf

[65] Mircrosoft, *Getting Started - Microsoft Threat Modeling Tool - Azure*. [Online]. Available: https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool-getting-started (accessed: Jun. 24 2019).

[66] OWASP, *Open Web Application Security Project (OWASP)*. [Online]. Available: https://www.owasp.org/

[67] F. Nielson and H. R. Nielson, *Formal methods: An appetizer*: Springer Publishing Company, Incorporated, 2019.

[68] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds., *Handbook of Model Checking*. Cham: Springer International Publishing, 2018.

[69] D. Basin, C. Cremers, and C. Meadows, "Model Checking Security Protocols," in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds., Cham: Springer International Publishing, 2018, pp. 727–762. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8_22

[70] T. Nipkow and G. Klein, *Concrete Semantics: With Isabelle/HOL*. Cham, s.l.: Springer International Publishing, 2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10542-0

[71] G. Klein, J. Andronick, M. Fernandez, I. Kuz, T. Murray, and G. Heiser, "Formally verified software in the real world," *Commun. ACM*, vol. 61, no. 10, pp. 68–77, 2018, doi: 10.1145/3230627.

[72] Y. Shoshitaishvili *et al.,* "SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis," in *IEEE Symposium on Security and Privacy*, 2016.

[73] F. Saudel and J. Salwan, Eds., *Triton: A Dynamic Symbolic Execution Framework*: SSTIC, 2015.

[74] D. Andriesse, *Practical Binary Analysis*. San Francisco, CA: No Starch Press Incorporated, 2018.

[75] M. Sutton, A. Greene, and P. Amini, *Fuzzing: Brute force vulnerabilty discovery*. Upper Saddle River, N.J.: Addison-Wesley, 2007. [Online]. Available: http://proquest.safaribooksonline.com/9780321446114

[76] V. P. Kemerlis, G. Portokalidis, K. Jee, and A. D. Keromytis, "Libdft: Practical Dynamic Data Flow Tracking for Commodity Systems," in *Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, 2012, pp. 121–132. [Online]. Available: https://doi.org/10.1145/2151024.2151042

[77] N. S. Yanofsky, *The outer limits of reason: What science, mathematics, and logic cannot tell us*. Cambridge, Mass.: The MIT Press, 2013.

[78] J. Saxe and H. Sanders, *Malware Data Science: Attack Detection and Attribution*. San Francisco, CA: No Starch Press Incorporated, 2018.

[79] S. Parkinson, A. Crampton, and R. Hill, *Guide to Vulnerability Analysis for Computer Networks and Systems: An Artificial Intelligence Approach*: Springer, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-92624-7

[80] P. Bontrager, A. Roy, J. Togelius, N. Memon, and A. Ross, "DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution," Accessed: May 14 2019. [Online]. Available: https://arxiv.org/pdf/1705.07386.pdf

[81] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge, Massachusetts, London, England: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org/

[82] I. J. Goodfellow *et al.,* "Generative Adversarial Networks," *arXiv.org*, 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[83] Tencent Keen Security Lab, *Autopilot*. [Online]. Available: https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf (accessed: May 14 2019).

[84] E. Al-Shaer, J. Wei, K. W. Hamlen, and C. Wang, *Autonomous Cyber Deception: Reasoning, Adaptive Planning, and Evaluation of HoneyThings*. Cham: Springer International Publishing, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-02110-8

[85] H. Koduvely, *Anomaly Detection through Reinforcement Learning*. [Online]. Available: http://blog.zighra.com/anomaly-detection-and-reinforcement-learning (accessed: Aug. 20 2020).

[86] Darpa, *Cyber Grand Challenge (CGC)*. [Online]. Available: https://www.darpa.mil/program/cyber-grand-challenge (accessed: May 15 2019).

[87] *Mayhem, the tech behind the DARPA Grand Challenge winner, now used by the Pentagon - CyberScoop*. [Online]. Available: https://www.cyberscoop.com/mayhem-darpa-cyber-grand-challenge-dod-voltron/ (accessed: Feb. 7 2019).

[88] Chandrasekaran and Rajiv, *WashingtonPost.com: Deep Blue Defeats Kasparov in Game 2*. [Online]. Available: https://www.washingtonpost.com/wp-srv/tech/analysis/kasparov/kasparov.htm? (accessed: Jun. 24 2019).

[89] N. Jones, *Quiz-playing computer system could revolutionize research*. [Online]. Available: https://www.nature.com/news/2011/110215/full/news.2011.95.html (accessed: May 15 2019).

[90] H. van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, "Hybrid Reward Architecture for Reinforcement Learning," *CoRR*, abs/1706.04208, 2017.

[91] V. Mnih *et al.,* "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.

[92] *AlphaGo: using machine learning to master the ancient game of Go*. [Online]. Available: https://blog.google/topics/machine-learning/alphago-machine-learning-game-go/ (accessed: Dec. 13 2017).

[93] O. Martin, *Bayesian Analysis with Python: Introduction to Statistical Modeling and Probabilistic Programming Using PyMC3 and ArviZ, 2nd Edition,* 2nd ed. Birmingham: Packt Publishing Ltd, 2018. [Online]. Available: https://ebookcentral.proquest.com/lib/gbv/detail.action?docID=5626694

[94] Max Jaderberg, Wojciech Marian Czarnecki, Iain Dunning, Thore Graepel, and Luke Marris, *Capture the Flag: the emergence of complex cooperative agents | DeepMind*. [Online]. Available: https://deepmind.com/blog/capture-the-flag-science/ (accessed: Jun. 25 2019).

[95] J. Pearl and D. Mackenzie, *The book of why: The new science of cause and effect*, 2018.

[96] J. Pearl, "Causal inference in statistics: An overview," *Statist. Surv.*, vol. 3, no. 0, pp. 96–146, 2009, doi: 10.1214/09-SS057.

[97] C. Davidson-Pilon, *Bayesian methods for hackers: Probabilistic programming and Bayesian methods*. New York: Addison-Wesley, 2016.

[98] W. Kurt, *Bayesian statistics the fun way: Understanding statistics and probability with Star Wars, LEGO, and rubber ducks*, 2019.

[99] François Chollet, *On the Measure of Intelligence*. [Online]. Available: https://arxiv.org/abs/1911.01547 (accessed: Feb. 3 2020).

[100] W. Ashford, *Microsoft: Is computing more trustworthy 10 years on?* [Online]. Available: http://www.computerweekly.com/feature/Microsoft-Is-computing-more-trustworthy-10-years-on (accessed: Jun. 29 2016).

[101] Altera, *Implementing FPGA Design with the OpenCL Standard*. [Online]. Available: http://www.altera.com/literature/wp/wp-01173-opencl.pdf

[102] MathWorks, *FPGA Design and Codesign - MATLAB & Simulink Solutions*. [Online]. Available: http://de.mathworks.com/solutions/fpga-design/ (accessed: Jun. 21 2016).

[103] R. J. Ellison, J. B. Goodenough, C. B. Weinstock, and C. Woody, "Evaluating and Mitigating Software Supply Chain Risks," [Online]. Available: http://www.sei.cmu.edu/reports/10tn016.pdf

[104] C. Eagle, *The IDA pro book: The unofficial guide to the world's most popular disassembler,* 2nd ed. San Francisco, CA: No Starch Press, 2011. [Online]. Available: http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10496688

[105] NSA, *Ghidra*. [Online]. Available: https://ghidra-sre.org/ (accessed: Feb. 3 2020).

[106] B. Chess and J. West, *Secure programming with static analysis*. Upper Saddle River, NJ [u.a]: Addison-Wesley, 2007. [Online]. Available: http://www.loc.gov/catdir/toc/ecip0713/2007010226.html

[107] CERT, *CERT Secure Coding Standards*. [Online]. Available: https://arxiv.org/pdf/1802.08842.pdf

[108] *CWE - Common Weakness Enumeration*. [Online]. Available: https://cwe.mitre.org/index.html (accessed: Jun. 16 2016).

[109] *Microsoft Security Development Lifecycle*. [Online]. Available: https://www.microsoft.com/en-us/sdl/ (accessed: Jun. 16 2016).

[110] *Building Security In Maturity Model*. [Online]. Available: https://www.bsimm.com/ (accessed: Jun. 16 2016).

[111] L. Ben Othmane, G. Chehrazi, E. Bodden, P. Tsalovski, and A. D. Brucker, "Time for Addressing Software Security Issues: Prediction Models and Impacting Factors," *Data Science and Engineering*, vol. 2, no. 2, pp. 107–124, 2017, doi: 10.1007/s41019-016-0019-8.

[112] D. Emm, *The malware business*. [Online]. Available: https://www.virusbulletin.com/uploads/pdf/conference_slides/2008/DavidEmm-VB2008.pdf (accessed: Jun. 16 2016).
[113] A L Johnson, *SWIFT attackers' malware linked to more financial attacks*. [Online]. Available: http://www.symantec.com/connect/blogs/swift-attackers-malware-linked-more-financial-attacks (accessed: Jun. 29 2016).
[114] D. E. Sanger, "Obama Order Sped Up Wave of Cyberattacks Against Iran," *New York Times*, 01.06, 01.06. http://www.nytimes.com/2012/06/01/world/middleeast/obama-ordered-wave-of-cyberattacks-against-iran.html?pagewanted=all&_r=0.
[115] J. T. Richelson, *A century of spies: intelligence in the twentieth century*. New York [u.a.]: Oxford Univ. Press, 1997.
[116] T. Rid and B. Buchanan, "Attributing Cyber Attacks," *Journal of Strategic Studies*, vol. 38, 1-2, pp. 4–37, 2014, doi: 10.1080/01402390.2014.977382.
[117] John Schindler, *False Flags: The Kremlin's Hidden Cyber Hand*. [Online]. Available: http://observer.com/2016/06/false-flags-the-kremlins-hidden-cyber-hand/ (accessed: Jun. 19 2016).
[118] David E. Sanger and Martin Fackler, *N.S.A. Breached North Korean Networks Before Sony Attack, Officials Say*. [Online]. Available: http://www.nytimes.com/2015/01/19/world/asia/nsa-tapped-into-north-korean-networks-before-sony-attack-officials-say.html?_r=0 (accessed: Jun. 21 2016).
[119] P. Maier and C. Nohl, *Low-Cost Chip Microprobing*. [Online]. Available: http://media.ccc.de/browse/congress/2012/29c3-5124-en-low_cost_chip_microprobing_h264.html
[120] S. Krempl, Ed., *29C3: Budget mobile turns into GSM base station*, 2012. [Online]. Available: http://www.h-online.com/open/news/item/29C3-Budget-mobile-turns-into-GSM-base-station-1775204.html
[121] *Wireshark*. [Online]. Available: https://www.wireshark.org/ (accessed: Jun. 29 2016).
[122] Chris Crowley, *What Your SecOps Team Can (and Should) Do*.
[123] L. Harding, *Footage released of Guardian editors destroying Snowden hard drives*. [Online]. Available: https://www.theguardian.com/uk-news/2014/jan/31/footage-released-guardian-editors-snowden-hard-drives-gchq (accessed: Jul. 4 2017).
[124] G. Klein *et al.,* "seL4: Formal Verification of an OS Kernel," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, 2009, pp. 207–220. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629596,%20Titel%20anhand%20dieser%20DOI%20in%20Citavi-Projekt%20%C3%BCbernehmen
[125] A. Jaquith, *Security metrics: Replacing fear, uncertainty, and doubt,* 1st ed. Boston Mass. u.a.: Addison-Wesley, 2007.
[126] A. Gawande, *The checklist manifesto: How to get things right,* 1st ed. New York, NY: Picador, 2010.
[127] Dominic Basulto, *How artificial intelligence could lead to self-healing airplanes*. [Online]. Available: https://www.washingtonpost.com/news/innovations/wp/2015/10/06/how-artificial-intelligence-could-lead-to-self-healing-airplanes/?utm_term=.0f98b7b643aa (accessed: Feb. 10 2018).
[128] Andrew Russell and Lee Vinsel, *Opinion | Let's Get Excited About Maintenance!* [Online]. Available: https://www.nytimes.com/2017/07/22/opinion/sunday/lets-get-excited-about-maintenance.html (accessed: Feb. 10 2018).
[129] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning: With applications in R,* 6th ed. New York, Heidelberg, Dordrecht, London: Springer, 2015.
[130] *ML2016 Cyber Security Attack Defender*. [Online]. Available: https://www.kaggle.com/c/ml2016-cyber-security-attack-defender (accessed: Mar. 29 2018).

[131] Symantec, "Internet Security Threat Report VOLUME 21, APRIL 2016," [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf?aid=elq_&om_sem_kw=elq_16196994&om_ext_cid=biz_email_elq_&elqTrackId=283a3acdb3ff42f4a70ab5a9f236eb71&elqaid=2902&elqat=2

[132] *AMBA Open Specifications: The de facto standard for on-chip communication.* [Online]. Available: http://www.arm.com/products/system-ip/amba/amba-open-specifications.phptm

[133] OpenCores, *SoC Interconnection: Wishbone.* [Online]. Available: http://opencores.org/opencores,wishbone (accessed: Aug. 20 2020).

[134] M. Abi-Antoun, D. Wang, and P. Torr, "Checking threat modeling data flow diagrams for implementation conformance and security," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, 393-396 numPG - 4. [Online]. Available: http://doi.acm.org/10.1145/1321631.1321692

[135] A. Mitra, W. Najjar, and L. Bhuyan, "Compiling PCRE to FPGA for accelerating SNORT IDS," in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, 2007, 127-136 numPG - 10. [Online]. Available: http://doi.acm.org/10.1145/1323548.1323571

[136] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. New York, NY [u.a.]: Springer, 2004. [Online]. Available: http://www.gbv.de/du/services/toc/bs/37088700x

[137] S. Checkoway *et al.,* "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX conference on Security*, 2011, 6-6 numPG - 1. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028073%20acmID%20%20-%20%202028073

[138] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Boston, MA: Springer Science+Business Media, LLC, 2007. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-38162-6

[139] *The Attack Surface Problem.* [Online]. Available: http://www.sans.edu/research/security-laboratory/article/did-attack-surface (accessed: Jun. 16 2016).

[140] M. Howard and S. Lipner, *The security development lifecycle: SDL, a process for developing demonstrably more secure software*. Redmond, Wash: Microsoft Press, 2006. [Online]. Available: http://www.gbv.de/dms/bowker/toc/9780735622142.pdf

[141] *ISO/IEC/IEEE International Standard - Systems and software engineering - Vocabulary.*

[142] Altera, "Applying the Benefits of Network on a Chip Architecture to FPGA System Design," [Online]. Available: https://www.altera.com/en_US/pdfs/literature/wp/wp-01149-noc-qsys.pdf

[143] Ofir Arkin and Josh Anderson, "EtherLeak: Ethernet Frame Padding Information Leakage," [Online]. Available: https://ofirarkin.files.wordpress.com/2008/11/atstake_etherleak_report.pdf

[144] K. R. Apt, *Principles of constraint programming*. Cambridge: Cambridge University Press, 2010.

[145] M. Kucera and M. Vetter, "A Generic Framework to Enforce Access Control in FPGAs with Dynamic Reconfiguration," in *Software Engineering and Applications - 2007*, 2007. [Online]. Available: http://www.actapress.com/PaperInfo.aspx?PaperID=32117&reason=500

[146] M. Schumacher, *Security patterns: integrating security and systems engineering*. Chichester [u.a.]: Wiley, 2006.

[147] David E. Bell, Leonard J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," [Online]. Available: http://csrc.nist.gov/publications/history/bell76.pdf

[148] K. J. Biba, "Integrity Considerations for Secure Computer Systems," [Online]. Available: http://seclab.cs.ucdavis.edu/projects/history/papers/biba75.pdf

[149] V. C. Hu *et al., Guide to Attribute Based Access Control (ABAC) Definition and Considerations*: National Institute of Standards and Technology, 2014. Accessed: Jun. 16 2016. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf

[150] M. Kucera and M. Vetter, "On Secure Resource Utilization in FPGAs with Partial Runtime Reconfiguration," in *Future Dependable Distributed Systems, 2009 Software Technologies for*, 2009, pp. 20–24.

[151] L. Bettini, *Implementing domain-specific languages with Xtext and Xtend: Learn how to implement a DSL with Xtext and Xtend using easy-to-understand examples and best practices*. Birmingham, UK: Packt Publishing, 2016. [Online]. Available: http://proquest.tech.safaribooksonline.de/9781786464965

[152] T. Parr, *The definitive ANTLR reference: building domain-specific languages*. Raleigh, NC [u.a.]: Pragmatic Bookshelf, 2007.

[153] R. Gronback, *Eclipse Modeling Project*. [Online]. Available: https://www.eclipse.org/modeling/emf/ (accessed: Dec. 15 2017).

[154] *Graphviz - Graph Visualization Software*. [Online]. Available: https://www.graphviz.org/ (accessed: Jan. 22 2018).

[155] *The GraphML File Format*. [Online]. Available: http://graphml.graphdrawing.org/ (accessed: Feb. 26 2018).

[156] Katherine Ognyanova, "Network visualization with R," [Online]. Available: http://www.kateto.net/wp-content/uploads/2015/06/Polnet%202015%20Network%20Viz%20Tutorial%20-%20Ognyanova.pdf

[157] A. Canteaut, K. Viswanathan, D. A. McGrew, and J. Viega, Eds., *The Security and Performance of the Galois/Counter Mode (GCM) of Operation: Progress in Cryptology - INDOCRYPT 2004*: Springer Berlin Heidelberg, 2005.

[158] M. Bastian, S. Heymann, and M. Jacomy, *Gephi: An Open Source Software for Exploring and Manipulating Networks*. [Online]. Available: http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154

[159] *NetworkX*. [Online]. Available: https://networkx.github.io/ (accessed: Feb. 6 2020).

[160] M. Huth and M. Ryan, *Logic in computer science: Modelling and reasoning about systems,* 2nd ed. Cambridge: Cambridge Univ. Press, 2011. [Online]. Available: http://reference-tree.com/book/logic-in-computer-science-modelling-and-reasoning-about-systems?utm_source=gbv&utm_medium=referral&utm_campaign=collaboration

[161] A. Cimatti *et al.,* "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking," in *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, 2002.

[162] R. Cavada *et al., NuSMV 2.6 User Manual*. [Online]. Available: http://nusmv.fbk.eu/NuSMV/userman/v26/nusmv.pdf (accessed: Jun. 16 2016).

[163] *Knuth: Frequently Asked Questions*. [Online]. Available: https://www-cs-faculty.stanford.edu/~knuth/faq.html (accessed: Feb. 10 2018).

[164] F. AKHTAR, *Practical Reinforcment Learning*. [S.l.]: PACKT PUBLISHING LIMITED, 2017.

[165] C. Isbell and M. Littman, *Reinforcement Learning - Udacity*. [Online]. Available: https://classroom.udacity.com/courses/ud600 (accessed: Dec. 13 2017).

[166] *BURLAP*. [Online]. Available: http://burlap.cs.brown.edu/ (accessed: Jan. 24 2018).

[167] *R: The R Project for Statistical Computing*. [Online]. Available: https://www.r-project.org/ (accessed: Jun. 20 2017).

[168] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, 1-2, pp. 181–211, 1999, doi: 10.1016/S0004-3702(99)00052-1.

[169] unkw, *Meltdown and Spectre*. [Online]. Available: https://meltdownattack.com/ (accessed: Feb. 9 2018).

[170] Xilinx, *Partial Reconfiguration in the ISE Design Suite*. [Online]. Available: https://www.xilinx.com/products/design-tools/partial-reconfiguration.html (accessed: Aug. 24 2016).

[171] S. Griffith, K. Subramanian, J. Scholz, C. L. Isbell, and A. Thomaz, "Policy shaping: Integrating human feedback with reinforcement learning," in *In Advances in Neural Information Processing Systems*, 2013.

[172] *AlphaZero: Shedding new light on the grand games of chess, shogi and Go | DeepMind*. [Online]. Available: https://deepmind.com/blog/alphazero-shedding-new-light-grand-games-chess-shogi-and-go/ (accessed: Jun. 25 2019).

[173] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari," Accessed: Mar. 18 2018. [Online]. Available: https://arxiv.org/pdf/1802.08842.pdf

[174] D. Lim, *Counterfeit Chips Plague U.S. Missile Defense*. [Online]. Available: https://www.wired.com/2011/11/counterfeit-missile-defense (accessed: Jun. 17 2016).

[175] M. B. Kelley, *Counterfeit Chinese Microchips Are Getting So Good They Can't Be Identified* (accessed: Jun. 17 2016).

[176] *PowerShell Injection with Diskless Payload Persistence and Bypass Techniques - Binary Defense*. [Online]. Available: https://www.binarydefense.com/powershell-injection-diskless-persistence-bypass-techniques/ (accessed: Jun. 23 2017).

[177] J. A. Halderman *et al.,* "Lest we remember: Cold boot attacks on encryption keys," in *In USENIX Security Symposium*, 2008.

[178] *How many ASIC Gates does it take to fill an FPGA? – Breaking The Three Laws*. [Online]. Available: https://blogs.synopsys.com/breakingthethreelaws/2015/02/how-many-asic-gates-does-it-take-to-fill-an-fpga/ (accessed: Feb. 10 2018).

[179] I. Hadizc, S. Udani, and M. S. Smith, "FPGA Viruses," *Lecture Notes in Computer Science*, vol. 1673, pp. 291–300, 1999. [Online]. Available: www.cis.upenn.edu/~boosters

[180] Dan Goodin, *"USB Killer" flash drive can fry your computer's innards in seconds*. [Online]. Available: https://arstechnica.co.uk/information-technology/2015/10/usb-killer-flash-drive-can-fry-your-computers-innards-in-seconds/ (accessed: Aug. 28 2017).

[181] C. Maxfield, *The design warriors's guide to FPGAs: devices, tools and flows*. Amsterdam [u.a.]: Elsevier [u.a.], 2004. [Online]. Available: http://www.loc.gov/catdir/toc/els051/2004557408.html

[182] I. Xilinx, "Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream Application Note (XAPP1267)," [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp1267-encryp-efuse-program.pdf

[183] Xilinx, *Two Flows for Partial Reconfiguration: Module Based or Difference Based*. [Online]. Available: http://www.xilinx.com/bvdocs/appnotes/xapp290.pdf

[184] Xilinx, *Platform Studio and the Embedded Development Kit (EDK)*. [Online]. Available: http://www.xilinx.com/products/design-tools/platform.html (accessed: Jun. 21 2016).

[185] S. Efftinge and M. Spoenemann, *Xtext - 15 Minutes Tutorial*. [Online]. Available: https://www.eclipse.org/Xtext/documentation/102_domainmodelwalkthrough.html (accessed: Feb. 1 2018).

[186] B. Naveh, *Welcome to JGraphT - a free Java Graph Library*. [Online]. Available: http://jgrapht.org/ (accessed: Dec. 15 2017).

[187] *scikit-learn: machine learning in Python — scikit-learn 0.18.1 documentation*. [Online]. Available: http://scikit-learn.org/stable/ (accessed: Jun. 20 2017).

[188] *How to use Multinomial and Ordinal Logistic Regression in R ?* [Online]. Available: https://www.analyticsvidhya.com/blog/2016/02/multinomial-ordinal-logistic-regression/ (accessed: Jun. 20 2017).

[189] Tavish Srivastava, *Basics of Ensemble Learning Explained in Simple English.* [Online]. Available: https://www.analyticsvidhya.com/blog/2015/08/introduction-ensemble-learning/ (accessed: Jun. 23 2017).

Appendix K Publications of the Author

K.1 Journal papers and book chapters

- Vetter, Michael (2019): MODEL-BASED SECURITY ANALYSIS OF FPGA DESIGNS THROUGH REINFORCEMENT LEARNING. In: Acta Polytech 59 (5), S. 518–526. DOI: 10.14311/AP.2019.59.0518.
- H. de Meer; M. Diener; R. Herkenhöner; M. Kucera ; M. Niedermeier; A. Reisser ; G. Schryen ; M. Vetter; T. Waas ; E. Yasasin (2013, in German)
  Sicherheitsherausforderungen in hochverteilten Systemen. In: PIK - Praxis der Informationsverarbeitung und Kommunikation. Band 36, Heft 3,
- Kucera, Markus; Vetter, Michael (2009): FPGA-Rootkits. In: Natividad Martínez Madrid und Ralf E.D Seepold (Hg.): Intelligent Technical Systems.Dordrecht: Springer Netherlands

K.2 Workshop Proceedings

- F. Lutz; M. Vetter; T. Waas; M. Kucera (2011): Load balancing and aggregation of multiple ADSL links. In: Intelligent Solutions in Embedded Systems (WISES), 2011 Proceedings of the Ninth Workshop on,
- Markus, Kucera; Vetter, Michael (2009): On Secure Resource Utilization in FPGAs with Partial Runtime Reconfiguration. In: Software Technologies for Future Dependable Distributed Systems
- Kucera, Markus; Vetter, Michael (2007): FPGA Rootkits. - hiding malicous code inside the hardware. In: Solutions in Embedded Systems (WISES '07) Proceedings of the Fith Workshop on .
- Vetter, Michael: On symmetric cryptography with FPGAs. in Počítačové architektury & diagnostika (PAD) 2009,
- Kucera, Markus; Vetter, Michael (2007):
  A Generic Framework to Enforce Access Control in FPGAs with Dynamic Reconfiguration. In: Software Engineering and Applications - 2007:ActaPress.
- Vetter, Michael: On symmetric cryptography with FPGAs. in Počítačové architektury & diagnostika (PAD) 2009,
- Kucera, Markus; Vetter, Michael (2007):
  A Generic Framework to Enforce Access Control in FPGAs with Dynamic Reconfiguration. In: Software Engineering and Applications - 2007:ActaPress.

K.3 Technical Report

- Vetter, Michael (2009):Security Implications of FPGAs in Embedded Systems, Technical Report No. DCSE/TR-2009-07