Thwarting Software Attacks on Data-intensive Platforms with Configurable Hardware-assisted Application Rule Enforcement

Mohammed M. Farag, Lee W. Lerner, and Cameron D. Patterson
Wireless@VT
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA 24061 U.S.A.
Email: {mmorsy, lwl, cdp}@vt.edu

***Abstract*—Security is difficult to achieve on general-purpose computing platforms due to their complexity, excess functionality, and resource sharing. An alternative is the creation of a Tailored Trustworthy Space for the system or application class of interest. We focus on data-intensive computing systems using reconfigurable hardware to implement streaming operations, and provide security assurances that are independent of application software, middleware, or operating system integrity and correctness. All interaction between software and the dataflow hardware passes through an automatically synthesized and formally verified hardware controller incorporating enforcement and real-time monitoring of application-specific rules. Abstractions provided by the Bluespec high-level language assist in the translation of domain-specific policy rules to synthesized logic. For the cognitive radio example used, hardware-enforced policies include physical layer rules such as sanctioned spectrum usage. Policy changes cause the secure generation and transfer of a new controller-wrapped datapath hardware plug-in. Datapath dynamic block swaps and cryptographic operations are managed entirely by the hardware controller rather than software drivers. Design for performance and design for security are therefore simultaneously addressed since the datapath is configured and monitored at hardware speeds, and software has no access to datapath configurations and cryptographic keys.**

I. Introduction

There is a trickle-down of architectural advances from high performance desktop and server computing platforms to embedded computer systems. Larger address spaces, more effective caches, RISC instruction sets, multi-gigabit serial I/O, multi-core and heterogeneous architectures are commonly used to satisfy performance and power constraints on systems performing computations on streaming data. However, embedded system adoption of desktop platform security mechanisms tends to lag adoption of performance techniques. Several reasons are usually given: power and cost concerns, a belief that software-based attacks are less likely compared to desktop platforms, and a focus on physical attacks. As embedded platforms are increasingly networked, the primary threat will likely shift to malware.

The trusted computing technologies used in large-scale, personal, and later embedded systems have the following progression:

1) *SLS: Software limits access to software and data.* Software is structured in layers, and application or user requests requiring services from a more privileged layer are vetted by software-implemented processes. The goal is to separate layers with robust APIs that cannot be circumvented. Java virtual machines and packet filters are examples of software stratification. Attacks generally have executable code masquerade as data, and use vulnerabilities such as buffer overflows to execute the data.
2) *SLH: Software limits raw hardware resource access.* Innermost software layers have exclusive responsibility for allocating and managing hardware resources such as CPUs, memory, and peripherals. Examples of layers interacting directly with hardware are virtual machines and operating systems. Access to hardware does not necessarily imply access to (possibly encrypted) software and data. Direct programmatic access to hardware is different from physical access to hardware, which introduces the possibility of side-channel and probing attacks.
3) *HLS: Hardware limits access to software and data.* Static hardware units, possibly controlled by the innermost and trusted software layer, assist in the separation of software processes and layers. Examples are memory management units (MMUs), Trusted Platform Modules (TPMs) [1], protected execution and launch portions of Intel's Trusted Execution Technology (TXT) [2], and the use of eTokens. Although hardware provides enforcement, trust in supervisory software may still be needed.
4) *HLH: Hardware limits raw hardware resource access.* Static hardware controllers have exclusive responsibility for managing hardware-implemented processes or channels, and can deny requests from software at any layer. Examples are the protected input and graphics parts of Intel's TXT, Intel's Virtualization Technology [3], and hardware firewalls.

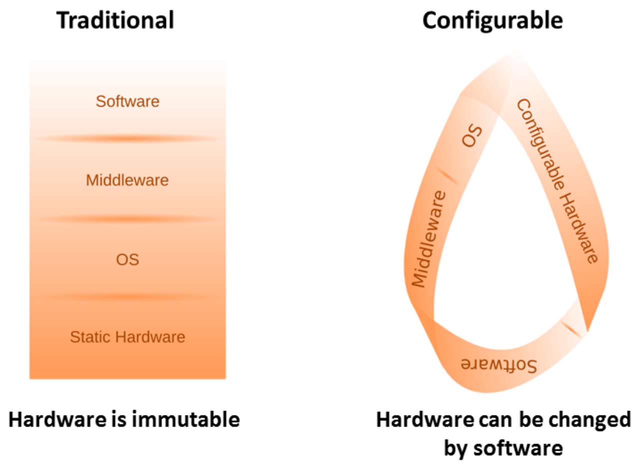

Fig. 1. Static versus reconfigurable platforms.

The programmability and performance of reconfigurable hardware suits data-intensive embedded computing applications. As shown in Fig. 1, traditional computing platforms have a fixed trust hierarchy consisting of static hardware, operating system, middleware, and application software. However, trust inheritance becomes complicated in configurable platforms when software updates the underlying hardware structure. "In hardware we trust" is no longer axiomatic since the hardware can be modified to violate specific policies. Current practice places dynamic hardware configuration under the control of application-level software, and even proposed OS-managed reconfiguration remains an SLH solution in the above taxonomy. Dynamic hardware blocks are application-tailored and potentially untrusted. Software modification of hardware structure is analogous to self-surgery, and independent hardware should provide oversight rather than rely solely on the correctness and integrity of application software and circuits. We insert a hardware-implemented, application-specific controller and monitor on the boundary between static hardware (which hosts software) and dynamic application hardware. This HLH approach retains most of the flexibility of application software directly controlling dynamic hardware, while enhancing trustworthiness, performance, and hardware abstraction.

The remainder of the paper is structured as follows: Section II surveys existing and proposed approaches to enhancing platform security. Section III describes an architecture combining a reconfigurable datapath with a synthesized controller to enforce and monitor application-specific rules. The use of the platform and associated tools are illustrated in Section IV with a cognitive radio application. Finally, Section V draws conclusions.

II. Existing Security Approaches

Current support for secure computing uses fixed hardware to enforce resource separation between applications. The Trusted Computing Group (TCG) developed a trusted platform (TP) specification to provide consistent behavior for a specific purpose using software and hardware enforcement [1]. Root of Trust for Management (RTM), Trusted Platform Module (TPM), and Trusted Software Stack (TSS) components provide three basic features: protected capabilities, attestation, and integrity measurement and reporting. Microsoft's Next Generation Secure Computing Base (NGSCB) is a software architecture exploiting the security provided by a TPM. The NGSCB consists of two kernels: an untrusted mode kernel, and a trusted NEXUS mode kernel that provides a secure environment for trusted code [4]. Intel's TXT uses processor enhancements, a TPM, operating system extensions such as the NGSCB, and enabled applications to protect sensitive information from software-based attacks [2]. ARM's TrustZone processor extensions support normal and secure environments, with a monitor mode providing robust context switches [5]. In all of these commercial examples, the sole focus is software separation, the execution model does not consider hardware adaptability, and there is excessive reliance on software correctness and integrity [6].

Reconfigurable hardware has been used to implement policy-driven memory protection mechanisms [7]. This work develops an access policy language to describe fine-grained memory separation of modules on an FPGA. A policy compiler converts the specified memory access policies into enforcement hardware modules. Reconfigurable hardware has been used to implement a TPM [8]. Modifications to existing FPGA architecture are required, including updates to the AES core bitstream decryptor and adding an on-chip non-volatile memory. An FPGA has been augmented with a Trust Block consisting of a TPM, a secure ROM storing FPGA configuration data, and switch logic used to configure the FPGA solely from secure ROM during system boot [9]. Unfortunately, these efforts do not target platforms where the hardware structure potentially changes during operation.

Reconfigurable systems often use third party IP cores. Although ideally these cores would be verified by a trusted party, cost and source code requirements can make such a development model impractical. Reliance on off-the-shelf IP modules provided by multiple vendors with different levels of trust introduces serious security concerns [10]. A moat and drawbridge model provides spatial module isolation and statically verifiable communication flow [11], but does not address modules with self-contained trojan horses.

III. Minimizing Software and IP Trust in a Reconfigurable Platform

Separation is a fundamental tool in secure system design and should be used in reconfigurable platforms with hardware and software interactions. In such platforms, software control of hardware configuration introduces new security concerns compared to static hardware systems. System trust can be enhanced by enforcing application-specific access control policies using either software or hardware. Hardware, which has greater tamper resistance and is better suited to formal analysis than software, provides policy oversight in the Configurable

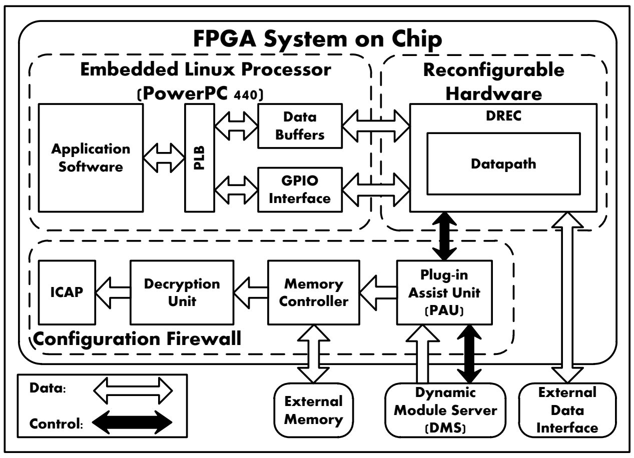

Fig. 2. CHARE prototype platform.

Hardware-assisted Application Rule Enforcement (CHARE) platform.

As shown in Fig. 2, the CHARE architecture has four major components: an embedded application processor, a reconfigurable controller-wrapped datapath, dedicated hardware to securely configure the datapath, and secure access to a shared configuration server. Fig. 3 shows the separate regions allocated to these components on the Xilinx Virtex-5 FX130T FPGA used in the initial prototype, with controlled communication between regions. Xilinx's Embedded Development Kit (EDK) connects one of the embedded PowerPC 440 processors to peripherals over a Processor Local Bus (PLB). The processor runs real-time Linux for data-intensive applications implemented with both software and custom hardware. A general-purpose I/O (GPIO) control interface stores datapath update request parameters while buffers transfer data between software and the reconfigurable hardware. There are few static routes crossing the dynamic region since most of the I/O signals connected to the two processors reside in the leftmost I/O banks. Roughly 70% of the chip area, including all 320 DSP slices and the majority of the Block RAM, is allocated to the reconfigurable plug-in region.

The reconfigurable hardware block consists of a datapath wrapped in a Datapath Rule Enforcement Controller (DREC). Parameterized IP cores are connected to implement streaming algorithms in domains such as DSP, communication, and video. The DREC is a hardware-based finite state machine responsible for checking that software-issued datapath update requests conform to policy rules embedded in the DREC. Software-visible datapath update registers contained in the GPIO bus interface are not directly connected to the datapath, and may not even have a one-to-one correspondence with actual datapath parameters. Invalid update requests return an error, while requests conforming to policy rules can result in a parameter update, individual module swaps, or a complete datapath plug-in replacement.

The DREC also serves as a datapath hardware abstraction layer. This has two advantages: software interaction with

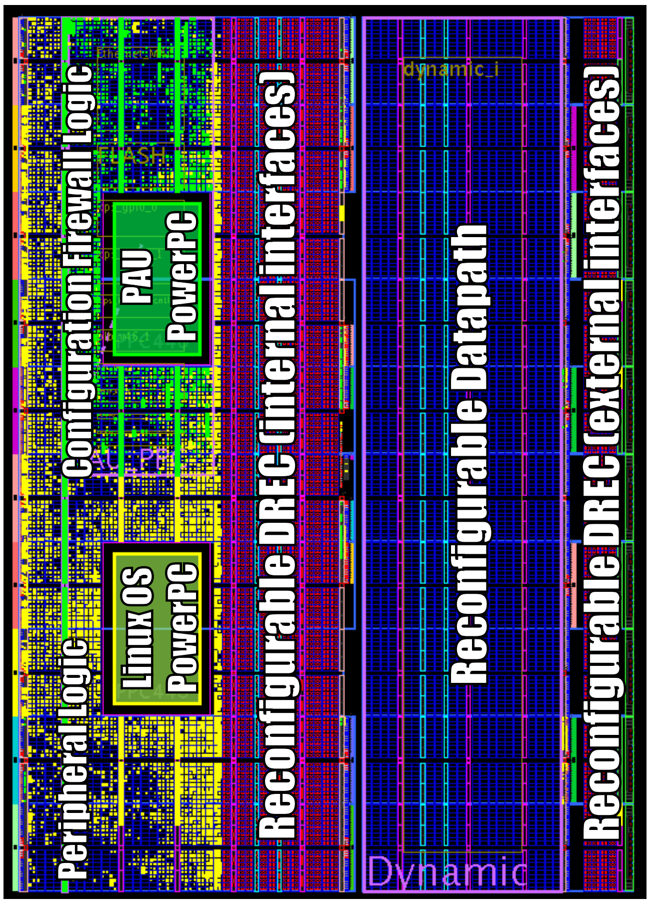
Peripheral Logic
Configuration Firewall Logic
PAU PowerPC
Linux OS PowerPC
Reconfigurable DREC (internal interfaces)
Reconfigurable Datapath
Reconfigurable DREC (external interfaces)
dynamic_i
Dynamic

Fig. 3. CHARE floorplan on a Xilinx Virtex-5 FX130T FPGA.

the hardware is simplified to enhance portability, and the datapath's detailed implementation is not revealed to software. For example, software is oblivious to the possible use of reconfiguration for swapping cores in response to datapath update requests. The hardware model presented to software restricts the set of control capabilities to that of a virtual ASIC implementing only the configuration options currently authorized. An additional advantage of custom hardware plug-ins is the provision of software-independent cryptographic services. For example, data may be automatically encrypted or decrypted by the plug-in using a key embedded in the datapath controller. Configurable hardware is generally more efficient than software for cryptographic algorithms [12], yet can be changed as readily as software implementations. Session keys may be used as a means of imposing expirations on particular capabilities.

DREC logic is never updated independently of the datapath, and any policy updates necessitate a complete plug-in replacement. Plug-ins may include monitors to check the operation of individual cores that may be untrusted or subject to single-event upsets. Detection of anomalous behavior signals the DREC to reload the plug-in if an upset occurred or a trojan horse was enabled. New plug-ins are securely (and perhaps wirelessly) transferred from a remote, shared and trusted Dynamic Module Server (DMS). Server-class hardware suits the time- and memory-intensive EDA tools required to generate new FPGA configurations; these tools exceed the resources available in embedded platforms. The DMS runs the PATIS tools to accelerate hardware plug-in implementation through the automatic parallel application of standard implementation tools [13].

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

IV. A Cognitive Radio Example

The need for flexibility and better spectrum management gave impetus to the cognitive radio (CR). A CR is a smart software defined radio that adapts its configuration based on perceived changes in its environment [14]. By sensing the spectrum, a CR detects and leverages opportunities by tuning communication variables such as the waveform, transmission power, modulation scheme, and operation frequency. Reconfigurable hardware suits both the performance and flexibility required by a CR [15]. To ensure integrity of the shared spectrum resource, the CHARE platform provides hardware oversight of physical-layer radio policies.

A simple digital AM transceiver with variable parameters is selected to demonstrate the application of CHARE to a CR platform. The AM transceiver architecture shown in Fig. 4 implements:

$$Y(t) = A_0 S(t) \cos(\omega_0 t + \phi), \tag{1}$$

where carrier frequency ω_0 and gain A_0, the two transmission parameters, are updated by issuing software update requests based on the radio's interaction with its environment. Xilinx's System Generator for DSP is used to construct the datapath. The carrier frequency is adjusted by modifying a Direct Digital Synthesizer (DDS) core parameter, while gain is adjusted by modifying an input to a multiplier core.

A. Policy Synthesis and Enforcement

Policies are derived from specifications of expected system behaviors. Specifications may describe, for example, expected software or hardware interactions with physical processes. Modern trust solutions fail to ensure that software functions, such as the embedded cognitive engine program in this CR

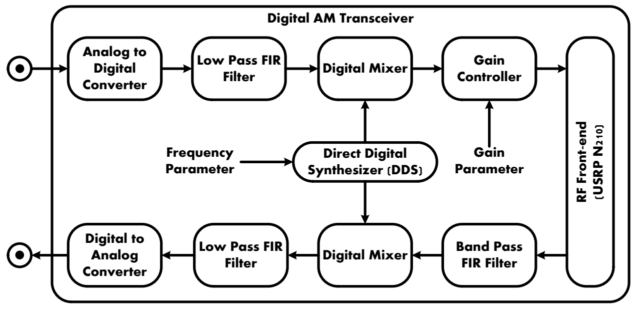

Fig. 4. Digital AM transceiver architecture.

example, conform to the policies in all instances without introducing additional, unknown behaviors. The CHARE platform demonstrates that policies can, however, be compiled into a hardware root-of-trust that is tightly integrated with a datapath to prevent undesired outside influence and erroneous internal behaviors.

In the CR example, configuration update requests are issued by the software-based cognitive engine to the reconfigurable hardware datapath. Policies are used to describe specifications that the datapath must always conform to, regardless of outside influence. Datapath update requests are specified as a set of registers which store update values for the DREC to evaluate against independently supplied policies, and update procedures which instruct the DREC on how to apply the updates to the datapath. Policies are compiled into satisfiability groupings, which are inferred from policy types. These groupings validate update requests and perform online monitoring of the datapath. When all policies are satisfied, the DREC performs the datapath update procedure associated with an update request.

Table I illustrates policies applied to a CR datapath. The policies are initially specified as a collection of the signal frequency and gain updates that can be made at certain times. Policies are typically associated with datapath update registers, parameters, or signals. Registers corresponding to policies `P1` through `P6` are shown in the rightmost three columns of Table I. The CHARE compiler synthesizes policy specifications into declarative assertions in the form of Bluespec SystemVerilog (BSV) rules that make up a policy set within the body of the DREC [16]. Targeting BSV enables efficient and verifiable generation of hardware transactors. BSV's atomicity and expressivity also aid policy specification, and provide abstractions familiar to software-oriented datapath designers. Atomicity ensures that rules are entirely executed as they are always enabled in the DREC. The CHARE compiler leverages Bluespec to manage potential interactions between rules through the automatic generation of arbitration and scheduling logic.

The CHARE compiler translates registers associated with a datapath update request into a BSV action method enabled when the request occurs. Any number of update requests may be used with all or a subset of the datapath registers. In this application, a single update request named `UR1` is specified which requires the DREC to check the entire set of datapath

TABLE I
POLICIES WITH CORRESPONDING UPDATE REGISTERS.

Policy	Type/Grouping	Frequency Range (MHz)	Maximum Gain (dBm)	Time Range
P1	Allow/G1	2.0-3.0	-10	0100-0200
P2	Allow/G1	2.0-3.0	-15	0300-0400
P3	Allow/G1	2.0-3.0	-10	0500-0600
P4	Allow/G1	3.5-4.5	-10	0700-0800
P5	Allow/G1	3.5-4.5	-15	0900-1000
P6	Monitor/G2	-	-	0000-1800

```
update_request(UR1)

update_registers(FixedPoint#(0,16) freq,
                 Int#(32) gain,
                 UInt#(12) time)

update_procedure(UP1)
```

Fig. 5. Update request format.

registers for policy conformance, as seen in Fig. 5. UR1 is used to send updates to the freq and gain datapath update registers.

Enabling the update method causes the DREC controller to evaluate the datapath register values corresponding to frequency, gain, and time against the policies P1 through P6. Because update requests are not linked directly to policies, every request must conform to all groupings of policy rules. Unlike DARPA's neXt Generation platform, policies cannot be added, removed, or deactivated without performing a complete DREC hardware replacement [17]. For applications where only a subset of policies are active at a given time, the remote DMS issues new DREC-wrapped datapath plug-ins.

Policies P1 through P5 are all of type Allow and are grouped into a single satisfiability BSV rule G1 that disables further DREC processing of the update request if none of the policies are satisfied. Update requests that do not satisfy any of the Allow policies are always rejected. The DREC can provide feedback as to which policies are not satisfied in a given update request, which may be useful for software learning or strategy development. Policies may also be hidden from software generating the update requests, in which case only a DREC accept or deny response is returned.

Policies can also act as monitors of datapath update registers and datapath parameters. Policy P6 is specified as type Monitor and is used to observe the time register. P6 asserts that the time register will always fall within 0000 to 1800. When the time falls outside of this range, the policy's satisfiability, G2, is no longer met and a response action is triggered in the DREC. This action could be used to disable datapath modules for various reasons, for example to save power in certain sections of a satellite's orbit, to enforce a mandatory offline built-in self-test cycle, or to expire IP after some period of system operation.

Monitors can also be placed on module interfaces or internal signals when specified as continuous or dynamic assertions. Monitors may be used in this manner to provide assurances on untrusted or poorly understood IP interfaces. Continuous assertions are specified directly in the DREC module and checked at each clock cycle. Similar to policies on datapath update registers, dynamic assertions are compiled to BSV rules in the DREC and evaluated whenever the rules are enabled. An optional guard may be placed on a rule containing dynamic assertion logic causing the rule to be enabled only when some condition is met [16], [18].

The CHARE compiler can also be extended to support commonly used libraries of assertions. For example, assertions found in the Acellera Standard Open Verification Library (OVL) are compiled to the DREC using the BSV interfaces and wrappers found in the OVLAssertions package [19]. Incorporating such monitors increases run-time observability which helps alleviate design verification and fault tolerance concerns.

Update requests are linked to update procedures that perform datapath plug-in reconfigurations or parameter adjustments on dynamically reconfigurable modules. When a plug-in reconfiguration is requested, the update procedure is written simply as an update code that the DREC passes to the PAU. If a datapath module parameter update is performed, the update procedure is written as the succession of values that the DREC must apply to registers representing datapath parameter values. The CHARE compiler synthesizes these procedures into sequences of BSV actions. In this application, a datapath parameter update procedure UP1 is associated with update request UR1. UP1 is specified as a series of parameter values that are written in order to phase shift the datapath's DDS modules and apply a new coefficient to the gain controller. The datapath parameter values written in UP1 do not correspond directly to datapath update registers used in the request and are not visible to the source of the request. Simultaneous, non-conflicting update procedures are parallelized in the DREC controller through the use of BSV par blocks.

Integration of the DREC with the underlying datapath is achieved with the Bluespec ImportBSV package. The DREC wraps the datapath, either applying procedures to interface signals or simply passing them through to data buffers or external interfaces. Applying policies on all datapath interfaces helps to ensure external devices cannot cause datapath updates. The Bluespec compiler is then used to generate RTL Verilog containing the original datapath and integrated DREC. A black-box version of the datapath is used during compilation to reduce implementation time and ensure Bluespec does not introduce internal modifications. The generated RTL and original datapath are then processed by the PATIS and ISE PR implementation flow to produce a device programming bitstream. Any logic overhead incurred using BSV is acceptable for many applications as the resultant implementation is often more timing efficient than what would be produced using conventional HDL [20], [21].

B. Operational Verification

A trustworthy security solution requires rigorous verification to ensure defined and correct behavior in all situations. Rather than rely solely on functional simulation to validate a solution, formal verification is also used to provide a correctness proof. The two main elements of formal verification are an accurate design model and precise specifications of required properties.

Operation of CHARE subsystems are formally verified through both policy-dependent and policy-independent property checking. This is accomplished using compositional model checking, which relies on a simple set of proof techniques and a domain-specific strategy [22]. The goal of this strategy is to reduce the verification of a large system to smaller and more tractable sub-modules. A proof system supporting this approach generates verification subgoals discharged by Cadence's SMV symbolic model checker. In such a verification framework, the DREC is modeled as a finite state machine with an access state corresponding to the BSV update request action method. The access state directly enforces spectrum access policies given in Table I by responding to an update request `UR1` and checking its conformance to policies. An update response is specified as a propositional formula capturing policies `P1` through `P6`. The update procedure `UP1` is granted for requests satisfying policy constraint predicates.

The DREC security specifications include safety and liveness properties. Liveness properties ensure a response for every update request occurs without deadlocks or livelocks, whereas safety properties guarantee that policies are correctly applied [23]. All datapath update requests should eventually have a response, an update request that complies with policy rules should be granted access, and an update request that does not comply with policy rules should be denied. These security properties have the following form in first-order temporal logic:

$$\begin{aligned}&\mathbf{always}\,(\mathsf{request} \rightarrow \mathbf{eventually}\,\mathsf{respond}),\\&\mathbf{always}\,(\mathsf{valid\_request} \rightarrow \mathbf{eventually}\,\mathsf{grant}),\\&\mathbf{always}\,(\mathsf{invalid\_request} \rightarrow \mathbf{immediately}\,\mathsf{deny}).\end{aligned}$$

V. Conclusions

Most of the major advances in computer system security rely on hardware for enforcement. Data-intensive embedded platforms may require a hardware reorganization capability, introducing development complications and new types of security threats including zero-day attacks. CHARE automatically generates application-specific dynamic hardware plug-ins consisting of controller-wrapped datapaths. Software drivers are simplified by the controller's encapsulation and abstraction of the datapath structure. Hardware augments software monitoring and enforcement through the synthesis and formal verification of a controller incorporating datapath policy rules. As a result, CHARE simultaneously addresses the performance, power, developer productivity, and security requirements of high-throughput, reconfigurable platforms.

Acknowledgment

This work is part of the AUSTIN project supported by the National Science Foundation, Grant No. CNS-0910557.

References

[1] Trusted Computing Group, 2010, http://www.trustedcomputinggroup.org.

[2] Intel Corp, "Intel Trusted Execution Technology Overview."

[3] ——, "Intel Virtualization Technology Overview."

[4] M. Peinado, Y. Chen, P. England, and J. Manferdelli, "NGSCB: A Trusted Open System," in *Information Security and Privacy*, ser. Lecture Notes in Computer Science, 2004, vol. 3108, pp. 86–97.

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

[6] S. Schoen, "Trusted Computing: Promise and Risk," October 2003, http://www.eff.org/wp/trusted-computing-promise-and-risk.

[7] T. Huffmire, T. Sherwood, R. Kastner, and T. Levin, "Enforcing memory policy specifications in reconfigurable hardware," *Computers and Security*, vol. 27, no. 5-6, pp. 197–215, 2008.

[8] T. Eisenbarth, T. Güneysu, C. Paar, A.-R. Sadeghi, D. Schellekens, and M. Wolf, "Reconfigurable trusted computing in hardware," in *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*, 2007, pp. 15–20.

[9] B. Glas, A. Klimm, O. Sander, K. Müller-Glaser, and J. Becker, "A system architecture for reconfigurable trusted platforms," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2008, pp. 541–544.

[10] T. Huffmire, B. Brotherton, N. Callegari, J. Valamehr, J. White, R. Kastner, and T. Sherwood, "Designing secure systems on reconfigurable hardware," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, pp. 44:1–44:24, July 2008.

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

[12] C. Patterson, "High performance DES encryption in Virtex FPGAs using JBits," in *8th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2000, pp. 113–121.

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

[14] Y.-C. Liang, H.-H. Chen, J. Mitola, P. Mahonen, R. Kohno, J. Reed, and L. Milstein, "Guest Editorial - Cognitive Radio: Theory and Application," *Selected Areas in Communications, IEEE Journal on*, vol. 26, no. 1, pp. 1–4, Jan. 2008.

[15] J. Lotze, S. A. Fahmy, J. Noguera, L. Doyle, and R. Esser, "An FPGA-based cognitive radio framework," in *Signals and Systems Conference (ISSC 2008)*, Jun. 2008, pp. 138 –143.

[16] *Bluespec SystemVerilog Reference Guide*, Bluespec, Inc., October 2009, http://www.bluespec.com.

[17] D. Elenius, G. Denker, and D. Wilkins, "XG Policy Architecture," April 2007, http://www.csl.sri.com/projects/xg/.

[18] M. Pellauer, M. Lis, D. Baltus, and R. Nikhil, "Synthesis of synchronous assertions with guarded atomic actions," in *Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE'05)*, 2005, pp. 15–24.

[19] *Accellera Standard OVL V2 Library Reference Manual*, Accellera, Inc., July 2010, http://www.accellera.org.

[20] A. Agarwal, "Comparison of high-level design methodologies for algorithmic IPs: Bluespec and C-based synthesis," Master's thesis, MIT, Feb 2009.

[21] J. Simsa and S. Singh, "Designing hardware with dynamic memory abstraction," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2010, pp. 69–72.

[22] K. L. McMillan, "A methodology for hardware verification using compositional model checking," *Sci. Comput. Program.*, vol. 37, pp. 279–309, May 2000.

[23] V. Hu, R. Kuhn, T. Xie, and J. Hwang, "Model checking for verification of mandatory access control models and properties," *International Journal of Software Engineering and Knowledge Engineering*, 2010.