This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2638439, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Hiding Hardware Trojan Communication Channels in Partially Specified SoC Bus Functionality

Nicole Fern*‡, Ismail San†, Çetin Kaya Koç*, and Kwang-Ting (Tim) Cheng*‡
*University of California, Santa Barbara
Email: {nicole, timcheng}@ece.ucsb.edu, koc@cs.ucsb.edu
†Anadolu University, Eskişehir, Turkey
Email: isan@anadolu.edu.tr
‡Hong Kong University of Science and Technology

***Abstract*—On-chip bus implementations must be bug-free and secure to provide the functionality and performance required by modern SoC designs. Regardless of the specific topology and protocol, bus behavior is never fully specified, meaning there exist cycles/conditions where some bus signals are irrelevant, and ignored by the verification effort. We highlight the susceptibility of current bus implementations to Hardware Trojans hiding in this partially specified behavior, and present a model for creating a covert Trojan communication channel between SoC components for any bus topology and protocol. By *only* altering existing bus signals during the period where their behaviors are *unspecified*, the Trojan channel is very difficult to detect. We give Trojan channel circuitry specifics for AMBA AXI4 and APB, then create a simple system comprised of several master and slave units connected by an AXI4-Lite interconnect to quantify the overhead of the Trojan channel and illustrate the ability of our Trojans to evade a suite of protocol compliance checking assertions from ARM. We also create an SoC design running a multi-user Linux OS to demonstrate how a Trojan communication channel can allow an unprivileged user access to root-user data. We then outline several detection strategies for this class of hardware Trojan.**

I. Introduction

Hardware Trojans are a concern for both semiconductor design houses and the U.S. government [1]. The design, manufacturing, testing, and deployment of silicon chips involves many parties. If a single party involved deems it advantageous to insert malicious functionality into the chip, referred to as *Hardware Trojans*, the consequences can be catastrophic.

Hardware Trojans may be inserted into the system specification, high-level models, RTL code, gate level net list, circuit layout, or circuit mask for a given design. Trojan behavior ranges from denial of service attacks such as premature aging and bus deadlock to subtler attacks which attempt to gain undetected privileged access on a system, leak secret information through side channels, or weaken random number generator output [2].

This work focuses on Trojans in SoC on-chip buses. The ability to manipulate the bus system is extremely valuable to an attacker since the bus controls communication between critical system components. A denial of service Trojan halting all bus traffic can render an entire SoC useless. Any information transferred to/from main memory, the keyboard, system display, network controller, etc. can be passively captured or actively modified by Trojan circuitry inserted in the interconnect.

There exist many different bus protocols designed to optimize different design parameters such as area/timing overhead, power consumption, and performance [3]. Regardless, all protocols employ signals to mark when valid bus transactions occur and handshakes to provide rate-limiting capabilities, meaning valid and idle bus cycles can be clearly differentiated. While bus protocols clearly define the desired values for each data or control signal during *valid* transactions, the values of these signals during idle cycles are **unspecified** and largely ignored by bus protocol checkers, formal verification properties, and scrutiny during simulation-based verification. Trojan behavior during these cycles will not be detected by traditional verification methodologies.

The Trojans we propose in this work operate entirely within idle bus cycles, with the goal being to provide a covert communication channel built upon existing bus infrastructure. This Trojan channel can be used to connect Trojan components spread across the SoC in addition to enabling information leakage from legitimate components not possible in the original design. Unlike previously proposed bus Trojans, which lock the system bus, modify bus data, and allow unauthorized bus transactions [4], [5], our Trojans never hinder normal bus functionality or affect valid bus transactions.

In Section II we review the current solutions addressing bus architecture security issues, and motivate why these are not adequate for detecting bus Trojans hiding in partially specified bus functionality. Section III-A outlines the threat model, Section III-B introduces the Trojan channel model and circuitry, and Section IV provides complete details for AMBA AXI4 and APB. The overhead of creating a 2-way information leakage channel between slaves with varying channel parameters in an AXI4-Lite interconnect is explored in Section V, then in Section VI a Trojan channel is inserted in a full SoC system running multi-user Linux to demonstrate how a malicious unprivileged software program can access root-user data. Several detection methodologies are outlined in Section VII, and Section VIII summarizes our results and contributions.

Copyright (c) 2015 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending an email to pubs-permissions@ieee.org.

0278-0070 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2638439, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

II. Related Work

A. Bus Security

The following are bus security issues being addressed in literature and industry:

1) Malicious snooping of bus data
2) Enforcing bus slave access control policies
3) Deadlock prevention (malicious and accidental)
4) Data integrity, data tampering prevention

Previously proposed bus Trojans include denial of service attacks accomplished by indefinitely asserting the LOCK signal in one of the bus masters or the WAIT signal in a bus slave, observing bus transactions between other components, corrupting bus data, and allowing a master to access forbidden address ranges [4], [5].

In [5], the authors present a secure AHB bus architecture to detect the above mentioned Trojans at runtime. A watchdog timer is added to detect bus deadlock, and to prevent snooping multiplexors are added on all data lines to zero the lines visible to components uninvolved in the current transaction, however this additional circuitry was shown to have significant impact on the maximum bus operation frequency.

Encryption of bus data [6], [7] has been proposed as a method to prevent bus snooping. Key maintenance, along with the overhead of encryption circuitry limits the widespread adoption of this countermeasure. While encryption of bus data prevents snooping, it does not prevent the existence of a Trojan communication channel.

To prevent illegal peripheral access, [5] adds registers holding the allowable access ranges for each bus master ensuring unauthorized requests are blocked and recorded. ARM TrustZone Controllers are commercial IP blocks which provide access control mechanisms to memory regions and bus peripherals and are compatible with other ARM bus IP and AMBA protocols [8].

Both these measures monitor *valid* bus transactions for violations. Since our proposed Trojans never modify existing or create new valid bus transactions, these countermeasures will not detect communication on the Trojan channel. Moreover, neither of these countermeasures address rouge communication between 2 slaves.

Extensive research on formal verification of bus protocols has been performed to ensure deadlock avoidance and fairness [9], [10], [11]. The properties checked using formal methods can be re-used during protocol compliance checking of specific bus implementations using either formal or simulation based methods. The availability of commercial compliance checking verification IP (ex. [12] for AMBA protocols) and pre-packaged SystemVerilog assertions suites [13] illustrate the importance of verifying the correctness of *specified* bus functionality.

During idle bus cycles, when VALID signals are de-asserted, there are no properties/assertions to capture what the correct behavior is, because it is not relevant to the protocol. Our proposed Trojans exploit this fact, and operate exclusively during these cycles to avoid violating assertions or detection during property checking.

B. Hardware Trojan Detection

Trojans with Rare Triggering Conditions: Many Trojans proposed in literature hide from the verification effort by only performing malicious functionality under extremely rare triggering conditions. Detection methods targeting this Trojan type identify “almost unused” logic, where rareness is quantified by an occurrence probability threshold. This probability is either computed statically using approximate boolean function analysis [14], [15] or based on simulation traces [16], [17].

The Trojans we propose in this work only modify signals under conditions during which they are *unspecified*, and to be detected by the existing methods, the occurrence of such conditions must be sufficiently rare. We argue that this is seldom the case. For example, our proposed Trojan communication channel can be used to snoop data destined for Slave A by placing data from valid writes to Slave A into a FIFO from which the data is read and leaked to Slave B’s bus interface whenever the channel is idle. The FIFO write condition is a valid data transfer to Slave A, and the leakage condition causing the data to appear at Slave B’s bus interface is an idle channel, neither of which are inherently “rare” conditions.

Detection Through Side-Channel Fingerprinting: Because our Trojans do not rely on rare triggering conditions and may be active frequently, differences in side-channel parameters such as power consumption and delay exist if a circuit infested with a Trojan channel is compared with a Trojan-free version. A large number of Trojan detection methods exist which use these differences (ex. in power consumption [18] and path delay [19]) to detect Trojan circuitry.

One requirement for most side-channel fingerprinting methods is a set of Trojan-free chips used to measure a “golden” side-channel fingerprint for comparison against the fingerprint measured for the chips suspected of containing Trojan circuitry. Even techniques which do not require a small population of “golden” chips, such as [20], still rely on having a golden Spice-level model to extract the Trojan-free fingerprint. In our threat model, given in Section III-A, we assume that the Trojan communication channel can be inserted in the RTL code and all subsequent stages in the design lifecycle, meaning it is possible that no golden RTL, gate-level, Spice-level models, or golden chips exist.

Trojans in Unspecified Functionality: Unspecified functionality is defined in [21] as incompletely specified state transition and output functions, given a digital system specified as a finite-state machine (FSM). The detection methodology proposed in [21] requires analysis on a *symbolic* representation of the design state space and the manual labeling of protected v. non-protected symbolic states.

The authors in [22] introduce a class of Trojans which leak information by only modifying RTL don’t care bits, and use combinational equivalence checking techniques to differentiate between don’t cares which can be exploited by an attacker to leak information and those which are harmless and should remain in the design for optimization during synthesis. A drawback of this technique is that only unspecified functionality captured by don’t care bits can be analyzed.

Using mutation testing, which is applicable broadly to FSM, C, SystemC, TLM, RT, and gate-level models, [23] builds upon

0278-0070 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2638439, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

the ideas presented in [22] to identify dangerous unspecified functionality in any type of design, including bus systems. Mutant simulation and analysis is expensive, but this process is necessary if one cannot identify dangerous unspecified functionality directly by inspection. Since bus systems are characterized by well-defined protocols and set of common topologies, our work directly presents a general model for dangerous unspecified bus functionality.

III. Trojan Communication Channel

There are many bus standards, providing the ability to optimize with respect to area/timing overhead, power consumption, and performance parameters [3]. For example the AXI [24] and APB [25] protocols from the ARM AMBA bus architecture target low-latency/high-throughput and low-speed/low-power respectively, and a similar pair of standards (the PLB and OPB protocols) exist in the IBM CoreConnect bus architecture [26].

Common among all standards are control signals marking when valid bus transactions occur. During idle cycles, the value of many control and data signals are unspecified, allowing a powerful Trojan communication channel to be built using the existing bus infrastructure. This section first gives our threat model, then details how to insert such a channel for any bus topology and protocol.

A. Threat Model

Since a covert communication channel is useless without a sender and receiver of information, we assume that at least one component connected to the system bus contains a Trojan utilizing the information received on the channel, and that there is another Trojan to either leak data from the component it resides in or snoop bus data otherwise not visible to the receiver and send it over the channel.

Although it is possible for the Trojan to create new bus transactions adhering to the bus protocol during unused cycles, verification infrastructure often includes bus checkers which count and log all valid bus transactions. For this reason, our proposed Trojans do not suppress, alter, or create valid bus transactions, but instead re-use existing bus protocol signals to define a new "Trojan" bus protocol allowing communication between different malicious components across the SoC.

Trojan Insertion Stage: It is assumed the Trojans are inserted in the RTL code or higher-level model, meaning no golden RTL model exists to aid in Trojan detection at later stages in the design cycle. While is it theoretically possible for a Trojan channel to be inserted by an adversary during fabrication, the amount of extra logic required (while only a small fraction of the total design area) is prohibitive. A complex SoC requires hundreds of engineers to design and test, and relies on third party IP and tools to meet time to market demands. A single rouge design engineer or malicious 3rd party IP or CAD tool vendor has the potential to implement a Trojan communication channel pre-silicon.

B. Trojan Channel Components

The structure and size of the Trojan communication channel circuitry depends on the following:

1) **Bus Topology:** Determines necessity of FIFO and extra Leakage Conditions Logic at receiver interface
2) **Bus Protocol:** Defines Leakage Conditions Logic and selection of signal(s) to mark valid Trojan transactions
3) **Trojan Channel Connectivity:** Channel can be one-way or bi-directional, contain an active or snooping sender, and involve information leakage between two masters, two slaves, or a master and a slave
4) **Data Width of Trojan Channel (k):** number of bits leaked during a Trojan transaction
5) **FIFO Depth (d):** FIFO used to buffer Trojan channel data if the receiver is busy accepting valid bus transactions

Bus topology and protocol are selected by the system designer, whereas Trojan channel connectivity is chosen by the attacker. Data width (k) and Trojan FIFO depth (d) are parameters selected by the attacker to trade-off performance and overhead of the Trojan channel.

The black-colored components in Figure 1a are necessary to implement a Trojan communication channel for a shared bus topology, which is shown in Figure 1b. For this case, the Data and Control lines from the sender component are directly visible at the receiver. The red-colored components in Figure 1a show the extra circuitry required to implement the channel in an interconnect with a MUX based topology, which is shown in Figure 1c.

The sender and the receiver can be any master or slave component on the interconnect. The goal of the Trojan channel is to use *only pre-existing* interconnect interfaces to pass data from the sender to the receiver. For example, the line labeled Data in Figure 1a on the sender's side could be the write data or read/write address port if the sender is a bus master and the read data port if the sender is a bus slave and vice versa for the Data on the receiver's side.

Since the Trojan data is transmitted using the same lines as normal bus traffic, additional signaling must mark when valid Trojan data is being transmitted. These signals are labeled as Control in Figure 1a, and like the Trojan data, are mapped to pre-existing data/address/control signals, meaning no additional interface ports are created.

The Leakage Conditions Logic is protocol dependent and examines signals at the sender's interconnect interface to determine when it is "safe" to replace the original bus signal values with Trojan values.

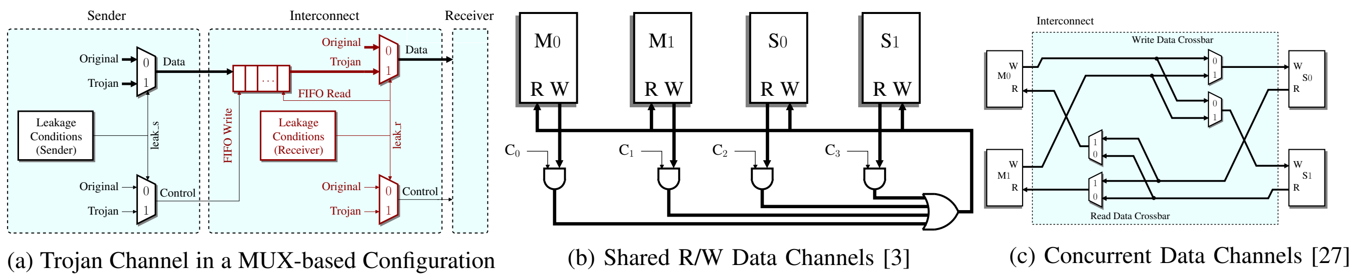
C. Topology Dependent Trojan Channel Properties

All bus signals can be classified as address, data, or control signals, and additionally classified as belonging to read and/or write functionality. The interconnect topology specifies the degree of parallelism between the different categories of bus signals, and the connectivity between masters and slaves [3].

Figures 1b and 1c show the read and write data channels for topologies sitting at opposite ends of the area efficiency and channel throughput trade-off. Figure 1b is the most area

0278-0070 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2638439, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



(a) Trojan Channel in a MUX-based Configuration (b) Shared R/W Data Channels [3] (c) Concurrent Data Channels [27]

Fig. 1: Bus Communication Topologies and a Trojan Channel

efficient, but can only support a single transaction at a time, whereas Figure 1c contains significantly more circuitry, but can support multiple simultaneous transactions.

In Figure 1b, all read and write transactions are visible to all bus components, meaning no Trojan circuitry is required to simply snoop bus data. If a Trojan bus component wishes to send information, the black-colored circuitry inside the sender block of Figure 1a is required.

In Figure 1c, data is not visible to a component uninvolved in the transaction. Unlike Figure 1b, forming a channel between two slaves or two masters requires extra circuitry inside the interconnect, shown in red in Figure 1a.

Because the signals at the sender's interconnect interface are not visible at the receiver's interface and vice versa, new leakage conditions are required, which monitor the receiver's interface and determine when it is safe to leak data without altering valid bus transactions. Signals available at the receiver's interface must also be selected to implement the Data and Control lines. The FIFO is necessary because leakage conditions at the sender and receiver may not occur simultaneously.

D. Protocol Dependent Trojan Channel Properties

The specifics of the Leakage Conditions Logic, which produces *leak_s* and *leak_r*, and the selection of Data and Control signals depend on the bus protocol used. Because of the similarities between various bus protocols, a general procedure for determining the Leakage Conditions Logic and the selection of Data and Control signals can be given.

1) ***Data Signal Selection****:* In order to remain stealthy, the Trojan cannot create additional signals to transmit data, and must send data via pre-existing signals in the bus protocol. Being that the primary purpose of a bus is to transmit data, all bus protocol/topology combinations have signals that are suitable for sending/receiving Trojan data.

In a protocol with separate read and write data signals, selection depends on if the Trojan Sender/Receiver resides in a master or slave component, since masters drive write data and observe read data signals, and vice versa for slave components. If the Trojan Sender resides in a master component, the read and write address signals can also be used to send Trojan data.

2) ***Leakage Conditions Logic****:* Since pre-existing bus signals are used to transmit Trojan data, logic ensuring that normal bus operation is not compromised by the Trojan is necessary. The Leakage Conditions Logic examines protocol control signals to identify when Trojan Data signals are not being used to transmit *valid* data, and have unspecified values.

Every bus protocol clearly defines the conditions for which data, address, and error reporting signals are valid. Some protocols, such as AXI4, designate a "valid" signal for each data channel, while others such as APB use the current state within the protocol to identify which signals are valid.

leak_s is set when the Trojan Sender has data to transmit and the Data signals are not involved in a valid transaction. If the Trojan Sender is leaking valid bus transactions instead of actively sending information, then *leak_s* is not needed. *leak_r* is set when there are items in the Trojan FIFO and the Data signals at the receiver interface are not currently involved in a valid transaction.

3) ***Control Signal Selection****:* When a Trojan Data signal is not being used in a valid bus transaction, its value is unspecified. During idle bus cycles, either Trojan data is being transmitted, or the bus is truly idle, and no data (Trojan or valid) is sent. To distinguish between these two cases, existing bus signals are selected to be Trojan Control signals, which mark when Trojan data is on the bus.

The criteria for selecting these signals and their corresponding values is that when *leak_s/leak_r* is asserted, the normal behavior of the signal is predictable, but also unspecified. For most protocols, control signals are good candidates because they often are unused during idle cycles, yet their values remain static when idle for a given implementation.

IV. Protocol Specific Trojan Channel Definitions

Following the general Trojan channel procedure outlined in Section III-D, we present the Leakage Conditions Logic and selection of Trojan Control and Data signals in detail for two commonly used bus protocols from ARM: AMBA AXI4/AXI4-Lite and AMBA APB in order to insert a Trojan in unspecified functionality.

AXI4 is a protocol designed for connecting high speed components such as processors, memory, and network controllers, and contains complex features to increase channel throughput. In contrast, APB is a simple protocol designed to connect low speed peripherals such as UART, keyboard, and timer modules. In a typical SoC, components on the APB bus are connected to the high speed bus via a bridging component [3].

A. AMBA AXI4

AXI4 defines 5 independent transaction channels seen at the interface of every master and slave: read address channel,

0278-0070 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2638439, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

read data channel, write address channel, write data channel, and write response channel [24]. Each channel uses a VALID/READY handshake signal pair to indicate when the receiver is ready to process bus data, and to mark when valid data is on the bus.

Typically, buses using AXI4 choose MUX-based configurations such as those shown in Figure 1c, meaning that the red-colored circuitry in Figure 1a is required to create the Trojan channel.

*1) **Master Sender**:* Data can be leaked through any bus signals a master drives, mainly those on the read or write address channels, or the write data channel. The values of all master driven signals on these channels have no functional meaning when the channel VALID signal is low, hence:

leak_s = *troj_data_ready* & ~VALID

Control Signal Selection: WSTRB is used in both AXI4 and AXI4-Lite, and quoting the specification, "A master must ensure that the write strobes are HIGH only for byte lanes that contain valid data. **When WVALID is LOW, the write strobes can take any value**..."

If the application uses all byte lanes in every transfer, it is likely that all strobe bits would be kept HIGH, even when WVALID is LOW, so a good indicator of a valid Trojan transaction would be to set 1 or more bits LOW when *leak_s* is asserted. If the interconnect services peripherals with data widths of 1, 2, and 4 bytes, asserting exactly 3 out of 4 bits of WSTRB is a better option, since this set of values is unlikely to be assigned to WSTRB during normal operation. The following assignment of WSTRB (where WSTRB_ORIG is the Trojan-free value of WSTRB) would work in both cases:

WSTRB = *leak_s* ? 4'b1011 : WSTRB_ORIG

The signal WLAST is used to indicate the last transfer in a write burst transaction. When WVALID is low, WLAST is not used, however almost certainly will be de-asserted, meaning that asserting this signal can also mark a valid Trojan transaction:

WLAST = *leak_s* ? 1 : WLAST_ORIG

*2) **Slave Sender**:* Data can be leaked through any bus signals a slave can drive (those on the read data channel or write response channel). The logic for *leak_s* is identical to the logic presented in the previous section since both channels employ VALID signals. To mark when Trojan data is valid, RLAST can be used in a similar manner as WLAST.

RRESP and BRESP are 2-bit error reporting signals and are typically set to indicate "OKAY, normal access success" (all 0's) when not in use (channel VALID is LOW). Setting either RRESP or BRESP to a non-zero state when *leak_s* is asserted can indicate the presence of Trojan data on the bus, for example:

RRESP = *leak_s* ? 2'b10 : RRESP_ORIG

*3) **Trojan Receiver**:* A Trojan master/slave receives information on the same set of bus signals a Trojan slave/master sends. Because of this symmetry, the selection of Data and Control signals is identical to the previous sections. The only difference is that before leaking data to a receiver, the FIFO must not be empty, meaning:

leak_r = *fifo_not_empty* & ~VALID

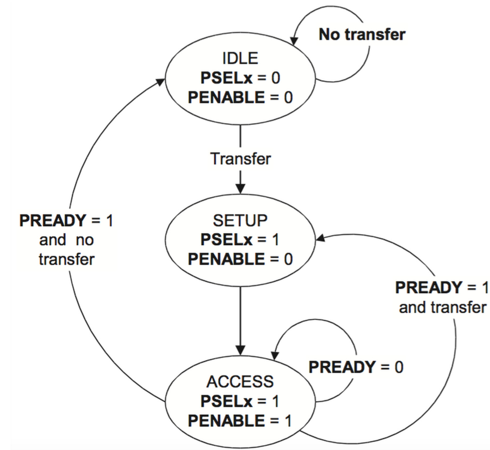


Fig. 2: AMBA APB Transaction State Diagram [25]

B. AMBA APB

The bridging component is the only bus master in APB. The slave components have their own slave select signal (PSELx), but typically share all read data (PRDATA) and control signals (PREADY and PSLVERR) in an AND-OR configuration like the one shown in Figure 1b.

*1) **Slave Sender**:* Since slaves can only drive PRDATA, PREADY, and PSLVERR, PRDATA is used for Trojan Data and PREADY and PSLVERR are selected as the Trojan Control signals. Since all 3 signals are visible to all bus components, the black-colored circuitry presented in Figure 1a is sufficient to implement the Trojan channel.

Figure 2 shows the state diagram for an APB transaction. PRDATA is only valid during the ACCESS state. The malicious slave leaks information by placing Trojan data on PRDATA as not to conflict with a valid transaction, but can only place data on PRDATA when PSELx is set, meaning information can only be leaked during the SETUP state:

leak_s = *troj_data_ready* & PSELx & ~PENABLE

Either PREADY or PSLVERR must be used to mark when valid data is on the Trojan channel. As seen in Figure 2, PREADY can take on any value during the SETUP phase without affecting the behavior of a valid transaction. Similarly, quoting the specification, "PSLVERR is only considered valid during the last cycle of an APB transfer, when PSEL, PENABLE, and PREADY are all HIGH" [25]. The combination of setting PSLVERR and de-asserting PREADY during the SETUP phase can be used to signal valid Trojan data.

*2) **Master Sender**:* The APB bridge is the only bus master, and a malicious APB bridge component can be used to connect a Trojan component from the high-speed bus with an APB bus slave. The APB bridge can leak data over PWRITE during the IDLE state, and use the combination of de-asserting all PSEL lines while asserting PENABLE to signal the occurrence of a Trojan transaction.

0278-0070 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2638439, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

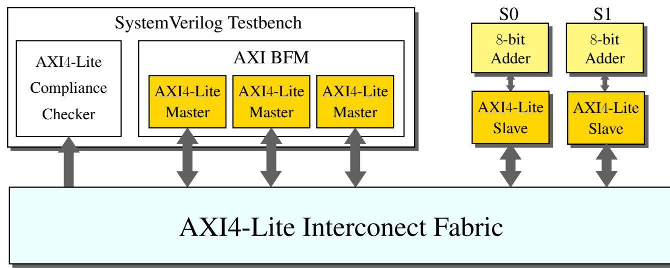


Fig. 3: AXI4-Lite Example System Verification Infrastructure

V. AXI4-Lite Interconnect Trojan Example

The system shown in Figure 3 is created to verify the AXI4-Lite Interconnect Fabric through RTL simulation. The two slaves are simple 8-bit adder coprocessors which receive 3 operands to add via the interconnect from 3 processors. Since the specifics of the main processors are irrelevant, in the example infrastructure, they are replaced by AXI4-Lite bus functional models (BFMs) from [28]. Additionally, AXI4-Lite assertions packaged by ARM for protocol compliance checking [13] are active during system simulation.

The AXI4-Lite Interconnect Fabric IP block used is the LogiCORE IP AXI Interconnect (v1.02.a) from Xilinx [27] configured in Shared-Address Multiple-Data (SAMD) mode (the topology shown in Figure 1c).

A. Trojan Operation

The AXI4-Lite Interconnect IP in Figure 3 is infected with two copies of the circuitry shown in red in Figure 4 to allow S1 to snoop on read requests for S0 and vice versa. Without the Trojan, the read data channel for S0 is not visible to S1 and vice versa.

The waveform in Figure 5 first demonstrates how 3 read data responses (values 42, 15, then 14) from S1 are snooped and routed to S0's write channel, then shows a single read data response (value 96) from S0 routed to S1's write channel, and finally another read data response from S1 (value 13) leaked to S0. All Trojan transactions are highlighted in red in Figure 5. The WSTRB signal is used to indicate when leaked data is on the bus. Normally WSTRB $== 1$, but when information is leaked, WSTRB $== 9$.

For AXI4-Lite, there are over 50 assertions monitoring bus signals during simulation, and none of them are violated even when information is flowing through the Trojan channel!

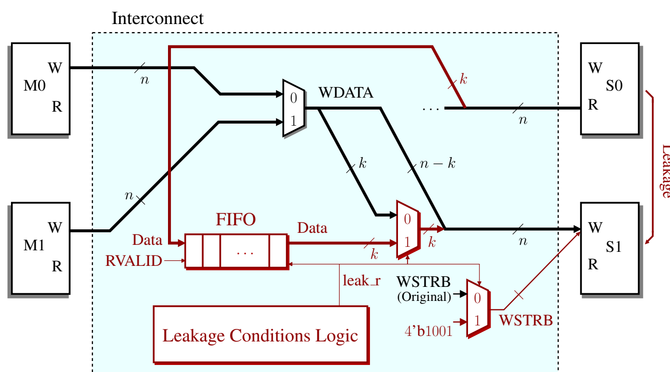


Fig. 4: Trojan Channel Logic for AXI4-Lite Interconnect

TABLE I: Trojan-Free Design Results (After Place and Route)

Configuration	# FF	# LUT	# BRAM	Frequency [MHz]
3 Masters 2 Slaves	1814	2474	2	250
4 Masters 6 Slaves	3071	4247	3	250

TABLE II: Area Overhead of 2-way HW-Trojan Channel

Data Width	FIFO Depth	% Increase in FF		% Increase in LUT	
		3M2S	4M6S	3M2S	4M6S
2	2	0.8	0.5	0.9	0.4
	4	1.1	0.7	1.5	0.6
	8	1.4	0.8	1.8	1.1
4	2	1.0	0.6	1.4	0.7
	4	1.3	0.8	2.0	0.8
	8	1.7	1.0	2.0	1.5
8	2	1.4	0.8	1.8	1.0
	4	1.8	1.0	2.4	1.2
	8	2.1	1.2	3.0	1.7

B. Overhead

To determine the area and timing overhead of implementing a 2-way Trojan channel between S0 and S1, the SystemVerilog Testbench in Figure 3 is replaced by several simple bus masters. Table I shows results for the Trojan-free design, after placement and route, assuming 3 masters and 2 slaves (labeled as 3M2S) as well as 4 masters and 6 slaves (labeled as 4M6S) for a Virtex-7 FPGA (7vx330t-3).

Table II illustrates how the selection of Trojan channel parameters Data Width (k) and FIFO Depth (d) affect the results. The Trojan channel does not affect the operating frequency of the design, and stays within 3% of the original FF and LUT utilization. As the number of masters and slaves increases, the interconnect and overall design area increases, but the size of the Trojan circuitry does not change.

The Trojan channel is easier to hide as the complexity of the interconnect and the number of components connected increases. The master and slave components used to generate the results in Tables I and II are far simpler than those in a typical SoC, so the results in Table II give a loose upper bound on the expected percentage of area increase caused by the Trojan channel in a modern design.

VI. Trojan Channel in SoC Implementation

To demonstrate how our proposed Trojan channel can give an attacker an extremely powerful foothold in a complex system, we infest a Xilinx Zynq ARM processor based SoC framework running a Linux OS with Trojan circuitry allowing an unprivileged user access to root-user memory transactions. In this section, we detail the Trojan channel operation, the interactions of users within the OS, and the area overhead of the Trojan.

A. Zynq-7000 Based SoC Platform Overview

We design and implement a Trojan infested SoC architecture based on the Zynq-7000 programmable SoC platform in order to demonstrate the operation of the proposed Trojan

0278-0070 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2638439, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

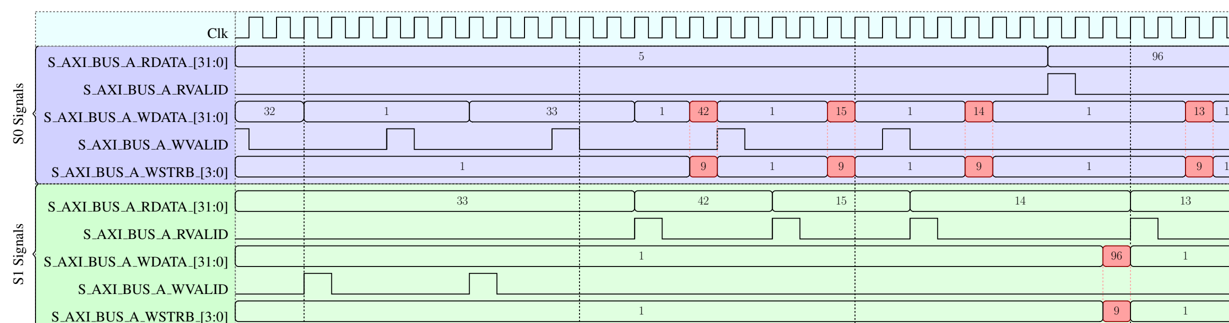
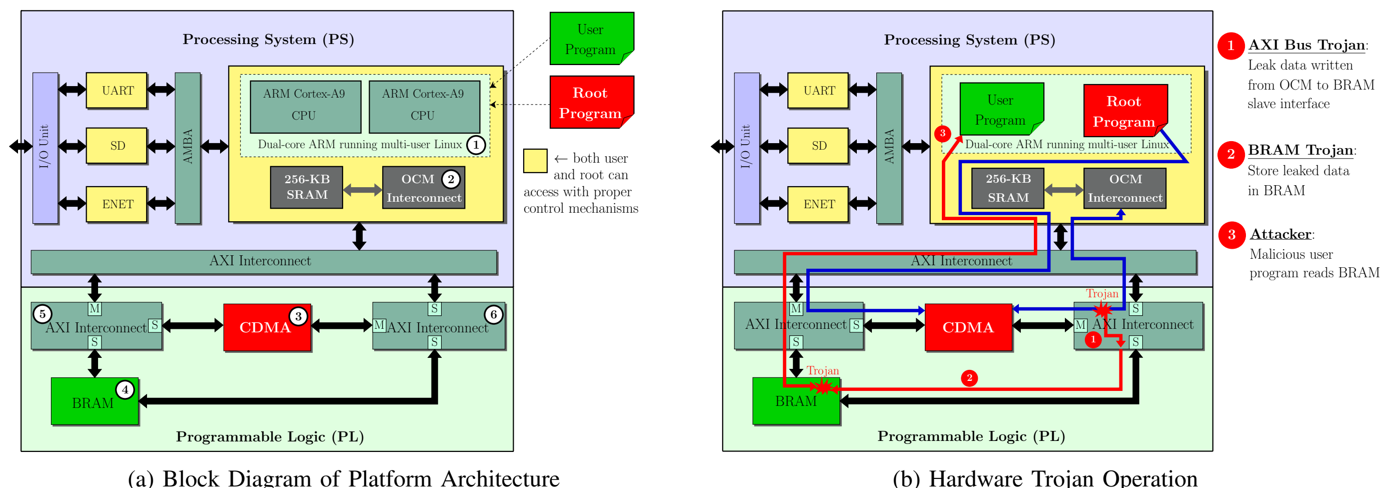


Fig. 5: 2-way Information Leakage Waveform



(a) Block Diagram of Platform Architecture

(b) Hardware Trojan Operation

Fig. 6: Zynq-7000 Based SoC Platform Used for Trojan Demonstration

channel in a real-world application. A full SoC environment running multi-user Linux is created containing Trojan infected Interconnect and Block RAM (BRAM) Controller IP allowing an unprivileged user to observe any data transferred via the Central Direct Memory Access (CDMA) Controller.

A block diagram of the SoC architecture is shown in Figure 6a. The SoC architecture includes (1) ARM processors running a multi-user Linux OS, (2) an on-chip memory (OCM) available to all users, but managed by the kernel to ensure memory isolation and privacy, (3) a central direct memory access (CDMA) controller only accessible by a user with root privileges which performs direct memory transfers from a source address to a destination address and (4) a BRAM component which can be accessed directly by any user.

Components communicate through several AXI Interconnect blocks, the most relevant labeled as (5) and (6) in Figure 6a. The ARM cores access the CDMA and BRAM peripherals through (5), and in (6) the CDMA initiates read/write transactions to the BRAM and on-chip memory.

The system is created using Vivado 2015.1 [29] targeting the Zynq-7000 All-Programmable SoC found in the Zedboard platform [30]. The Zynq-7000 architecture integrates two ARM Cortex-A9 cores, on-chip memory, and other peripherals, designated as the Processing System (PS) with Xilinx Programmable Logic (PL) [31]. The Processing System provides the necessary resources to run Xillinux [32], a multi-user Linux distribution, while the flexibility of the Programmable Logic allows for Trojan insertion.

B. Hardware Trojan Operation

Figure 6b illustrates how Trojan circuitry inserted in the BRAM Controller and AXI Interconnect enables an unprivileged user program to observe memory transfers made by root. Details of the inserted circuitry are given in the Appendix.

First, a root program must initiate a DMA transfer by writing to control registers in the CDMA. The most basic DMA transfer requires specifying the Source Address (SA), Destination Address (DA), and number of Bytes to Transfer (BTT) [33]. Once the BTT register is written, the DMA transfer is performed by issuing read and write transactions to the relevant peripheral (in Figure 6b the CDMA is transferring data between two locations in on-chip memory). This flow is illustrated by blue arrows in Figure 6b.

The following steps, shown using red arrows in Figure 6b, illustrate Trojan operation:

1) AXI Bus Trojan leaks transactions visible only at the OCM slave interface to the BRAM slave interface
2) BRAM Trojan captures leaked data at the AXI interface, stores at incrementing BRAM memory locations
3) Malicious unprivileged user program reads BRAM locations containing the leaked data

0278-0070 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2638439, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

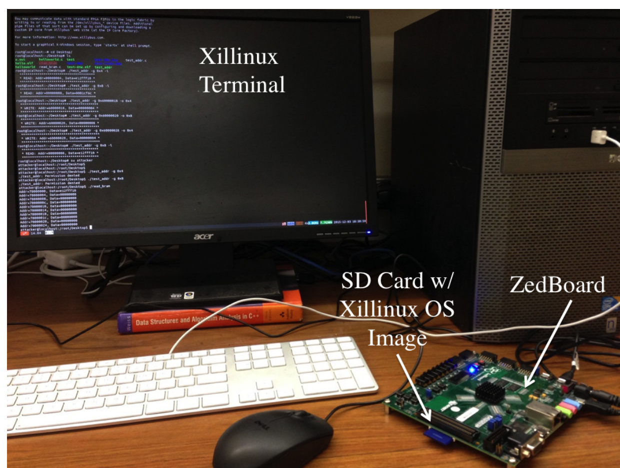


Fig. 7: Demonstration Environment

One should note that even if an attacker does not have the ability to run or infiltrate a software program running on the SoC, information from the Trojan channel can be captured and transmitted to the attacker using only hardware Trojans. For example, instead of leaking the DMA transfer data to BRAM, a Trojan infested Ethernet or UART Controller could be used to send data to an attacker.

C. OS-Level Extraction of Trojan Channel Information

Figure 7 shows the demonstration environment. Xillinux runs on an SD card located on the Zedboard, and a USB/UART cable connects a desktop workstation to a Xillinux root terminal.

The demo uses two Xillinux users: `root` and `attacker`. The privileged user `root` can read/write directly to physical addresses using a program called `access_addr` while `attacker` is unprivileged, and cannot use this program.

However, to allow non-privileged users access to the BRAM, the executable `read_bram` runs with root privileges, but can be executed by any user, and reads the first 10 locations in the BRAM. The `read_bram` program can be thought of as a very simple device driver since it provides an unprivileged user with controlled and limited access to a peripheral.

In the demo, `root` uses the DMA controller to transfer the contents at address 0x4 to address 0x8 (both on-chip memory locations). In the system memory map, on-chip memory addresses start at 0x0, the CDMA base address is 0x60000000, and the BRAM base addresses is 0x70000000.

Figure 8 shows Xillinux terminal output during the demonstration of Trojan functionality. Note that the commands at the beginning of the demo are executed as `root`.

(1) Data at addresses 0x4 and 0x8 are read using `access_addr`. (2-4) CDMA registers are written, instructing the CDMA to transfer 4 bytes of data from address 0x4 to address 0x8. (5) `access_addr` is used to confirm that the correct data from address 0x4 (0xe12fff10) is written to address 0x8. (6) The demo switches to the perspective of the `attacker` user. Notice that `attacker` tries to execute `access_addr` to learn the contents of addresses 0x4 and 0x8, but does not have sufficient privileges to do so. (7)

```
root@localhost:~/Desktop#
root@localhost:~/Desktop# ls
access_addr  helloworld    read_bram    test        test-dma.elf  test-linux-dma
hello.elf    helloworld.c  read_bram.c  test-dma    test-dma_bsp  test_addr.c
root@localhost:~/Desktop# ./access_addr -a 0x4 -i
  *************************************
  * READ: Addr=00000004, Data=e12fff10 *
  *************************************
root@localhost:~/Desktop# ./access_addr -a 0x8 -i
  *************************************
  * READ: Addr=00000008, Data=0001cf6c *
  *************************************
root@localhost:~/Desktop# ./access_addr -a 0x60000018 -o 0x4
  **************************************
  * WRITE: Addr=60000018, Data=00000004 *
  **************************************
root@localhost:~/Desktop# ./access_addr -a 0x60000020 -o 0x8
  **************************************
  * WRITE: Addr=60000020, Data=00000008 *
  **************************************
root@localhost:~/Desktop# ./access_addr -a 0x60000028 -o 0x4
  **************************************
  * WRITE: Addr=60000028, Data=00000004 *
  **************************************
root@localhost:~/Desktop# ./access_addr -a 0x8 -i
  *************************************
  * READ: Addr=00000008, Data=e12fff10 *
  *************************************
root@localhost:~/Desktop# su attacker
attacker@localhost:/root/Desktop$
attacker@localhost:/root/Desktop$ ./access_addr -a 0x4
./access_addr: Permission denied
attacker@localhost:/root/Desktop$ ./access_addr -a 0x8
./access_addr: Permission denied
attacker@localhost:/root/Desktop$ ./read_bram
Addr=70000000, Data=e12fff10
Addr=70000004, Data=00000000
Addr=70000008, Data=00000000
Addr=7000000c, Data=00000000
Addr=70000010, Data=00000000
Addr=70000014, Data=00000000
Addr=70000018, Data=00000000
Addr=7000001c, Data=00000000
Addr=70000020, Data=00000000
Addr=70000024, Data=00000000
attacker@localhost:/root/Desktop$
```

1) Read on-chip memory locations
2) Write DMA SA Register
3) Write DMA DA Register
4) Write DMA BTT Register
5) Check DMA Transfer
6) Switch from root to attacker user
7) Read leaked data from BRAM

Fig. 8: OS-Level Trojan Demonstration Shell Commands

Because of the hardware Trojan, the attacker is able to recover the data transferred by `root` using `read_bram`.

D. Overhead

Table III shows the overhead of inserting the Trojan circuitry in the AXI Interconnect and BRAM Controller IP. The Trojan channel data width is 32 bits, and the interconnect topology is such that no FIFO is necessary. Further Trojan circuitry details are given in the Appendix.

The utilization results given are for the Programmable Logic portion of the platform, since the Processing System exists on the FPGA board as hard silicon, and cannot be modified or further optimized by Vivado.

TABLE III: Overhead of Programmable Logic in SoC Platform (After Place-and-Route)

	# FF	# LUT	# Memory LUT	# Block RAMs	Freq. [MHz]
Trojan-Free	4766	4149	267	1	50
Trojan-Infested	4809	4201	267	1	50
% Increase	0.9	1.2	0	0	0

The presence of the Trojan circuitry did not affect the frequency of the design, and the FF and LUT utilization rose by approximately 1% making the Trojan circuitry unlikely to be detected due to anomalous area consumption.

VII. Detection Strategies

To *guarantee* that no Trojan channel exists in the interconnect circuitry, one must:

1) Fully specify the behavior of every bus signal
2) Modify the bus implementation to comply with the fully refined specification
3) Formally prove the bus implementation conforms to the behavior specified in 1)

0278-0070 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2638439, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Even if the requirement for formal verification is replaced by assertions monitoring the interconnect during simulation, for complex protocols, the task of complete behavior specification without causing unacceptable overhead is formidable.

For example, in AXI4, it is easy to require that if a channel VALID signal is LOW, all other channel signals must be driven LOW. However, given that data and address buses in AXI4 are typically 32 or 64 bits wide, an implementation adhering to this requirement must augment hundreds of bits with MUX circuitry to switch between LOW and the original signal value.

To overcome the large area and power overhead of zeroing circuitry for data and address signals, this circuitry can be implemented only for signals that have the potential to become Trojan channel Control signals (ex. WSTRB and WLAST). Preventing the ability to signal when Trojan transactions occur greatly decreases the usability of the Trojan channel.

If no zeroing circuitry can be afforded, the Trojan channel can be targeted by developing additional complex assertions, which define the behavior of bus signals during invalid cycles in a less straight forward, but more area efficient way. For example, instead of requiring WSTRB $== 0$ when VALID is LOW, a test bench monitor can record the value of WSTRB during the most recent valid write transaction and require that this value remain unchanged until the next valid transaction.

The detection strategy employed will ultimately be determined by the complexity of the protocol, the amount of overhead tolerated, the amount of effort budgeted for design verification, and the overall level of security desired. Detailed descriptions of detection methods are beyond the scope of this paper and are not included.

VIII. Conclusion

We present a new type of Hardware Trojan which creates a covert communication channel between components spread across an SoC using only existing on-chip bus signals without affecting normal bus functionality. We illustrate how our Trojan channel communication model is applicable to any bus topology and protocol, and give details for two widely used protocols. Our Trojan channel circuitry is shown to avoid detection by a protocol compliance checking suite from the IP vendor, and confirmed to have manageable area overhead. We also illustrate how Trojan channel information can be extracted by malicious unprivileged software by creating a complete SoC platform infected with a bus Trojan. Additionally, several detection strategies are outlined.

Appendix
Details of Trojan Insertion in Xilinx IP

Each block in the Programmable Logic portion of Figure 6a corresponds to a Verilog or VHDL module provided by Xilinx, with Vivado integrating the IP into a complete system. Trojans are inserted in the AXI4 Interconnect and AXI BRAM Controller IP blocks.

AXI4 Interconnect: The AXI Interconnect block labeled (6) in Figure 6a has a single bus master (the CDMA) and two slaves. The Verilog file, *axi_crossbar_v2_1_axi_crossbar.v*, from AXI Interconnect 2.1 (Rev. 5) [34] is modified to insert the Trojan into this block.

Because there is only a single bus master, the 32-bit write data is broadcast to both of the slaves. Even though the BRAM slave can observe write data destined for the processing system, WVALID signals are not broadcast, meaning only the processing system knows which cycle the data is valid. Trojan circuitry is needed to notify the BRAM slave when valid data is being sent to the processing system.

Similar to the example in Section V, the 4-bit WSTRB signal seen at the BRAM slave interface is used to mark when valid data is being written to the processing system. Since the BRAM data width is 32 bits, WSTRB is always 4'b1111. The Trojan circuitry sets WSTRB to 4'b1110 to mark when data is being written to the processing system.

Since there is only one bus master, valid write data can never be sent to BRAM and the processing system simultaneously, guaranteeing that valid write transactions to BRAM are not disrupted when the Trojan alters WSTRB. This eliminates the need for Trojan FIFO or buffering circuitry.

AXI BRAM Controller: The Trojan inserted in the AXI BRAM Controller, labeled (4) in Figure 6a, captures WDATA (32-bits) when WSTRB is 4'b1110, then writes the data to Port B of the BRAM. In our example framework, the address the leaked data is written to starts at 0x70000000, then increases by 4 with every data word written.

The VHDL file, *full_axi.vhd*, from AXI BRAM Controller v4.0 [35] is modified by adding a counter to increment the BRAM address for the leaked data and logic to monitor the AXI write data channel and write the leaked data to the BRAM.

Acknowledgements

This work was supported by NSF/SRC STARSS (1526695). The authors would also like to thank the Xilinx University Program for the generous donation of multiple ZedBoard development kits.

References

[1] S. Adee, “The hunt for the kill switch,” *IEEE Spectrum*, vol. 45, no. 5, pp. 34–39, May 2008.
[2] M. Tehranipoor and F. Koushanfar, “A survey of hardware trojan taxonomy and detection,” *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, Jan 2010.
[3] S. Pasricha and N. Dutt, *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann Publishers Inc., 2008.
[4] L.-W. Kim, J. D. Villasenor, and Ç. K. Koç, “A trojan-resistant system-on-chip bus architecture,” in *Proceedings of the 28th IEEE Conference on Military Communications*, ser. MILCOM'09, 2009, pp. 2452–2457.
[5] L.-W. Kim and J. D. Villasenor, “A system-on-chip bus architecture for thwarting integrated circuit trojan horses,” *VLSI Systems, IEEE Transactions on*, vol. 19, no. 10, pp. 1921–1926, 2011.
[6] A. Waksman and S. Sethumadhavan, “Silencing hardware backdoors,” in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, ser. SP'11, 2011, pp. 49–63.
[7] M. Henson and S. Taylor, “Memory encryption: A survey of existing techniques,” *ACM Computing Surveys*, vol. 46, no. 4, pp. 53:1–53:26, 2014.
[8] “Arm trustzone controllers.” [Online]. Available: http://www.arm.com/markets/trustzone-controllers.php
[9] D. Wang, “Formal verification of the PCI local bus: A step towards ip core based system-on-chip design verification,” Master's thesis, Carnegie Mellon University, May 1999.

0278-0070 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2016.2638439, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

[10] A. Roychoudhury, T. Mitra, and S. R. Karri, "Using formal techniques to debug the AMBA system-on-chip bus protocol," in *Design, Automation and Test in Europe Conference and Exhibition, DATE'03*, 2003, pp. 828–833.
[11] R. Luo and H. Tan, "Formal modeling and model checking analysis of the wishbone system-on-chip bus protocol," in *Proceedings of the Third International Conference on Information Computing and Applications, ICICA'12*. Springer-Verlag, 2012, pp. 211–220.
[12] "Synopsys vip for arm amba." [Online]. Available: http://www.synopsys.com/Tools/Verification/FunctionalVerification/VerificationIP/amba/Pages/default.aspx
[13] "Amba 4 axi4, axi4-lite and axi4-stream protocol assertions bp063 release note (r0p1-00rel0)," ARM. [Online]. Available: https://silver.arm.com/browse/BP063
[14] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS'13*. ACM, 2013, pp. 697–708.
[15] D. Sullivan, J. Biggers, G. Zhu, S. Zhang, and Y. Jin, "FIGHT-Metric: Functional identification of gate-level hardware trustworthiness," in *Proceedings of the 51st Annual Design Automation Conference, DAC'14*. ACM, 2014, pp. 173:1–173:4.
[16] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "VeriTrust: Verification for hardware trust," in *Proceedings of the 50th Annual Design Automation Conference, DAC'13*. ACM, 2013, pp. 61:1–61:8.
[17] M. Hicks *et al.*, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP'10*. IEEE Computer Society, 2010, pp. 159–172.
[18] D. Agrawal *et al.*, "Trojan detection using ic fingerprinting," in *IEEE Symposium on Security and Privacy*, 2007.
[19] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, June 2008, pp. 51–57.
[20] Y. Liu, K. Huang, and Y. Makris, "Hardware trojan detection through golden chip-free statistical side-channel fingerprinting," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14. New York, NY, USA: ACM, 2014, pp. 155:1–155:6.
[21] C. Dunbar and G. Qu, "Designing trusted embedded systems from finite state machines," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 5s, pp. 153:1–153:20, Oct. 2014.
[22] N. Fern, S. Kulkarni, and K.-T. Cheng, "Hardware Trojans hidden in RTL don't cares - Automated insertion and prevention methodologies," in *Proceedings of the 2015 IEEE Internation Test Conference (ITC)*, Oct 2015.
[23] N. Fern and K.-T. Cheng, "Detecting hardware trojans in unspecified functionality using mutation testing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD'15*. IEEE Press, 2015, pp. 560–566.
[24] *AMBA AXI and ACE Protocol Specification, Issue E*, ARM, 2013.
[25] *AMBA 3 APB Protocol v1.0 Specification, Issue B*, ARM, 2004.
[26] IBM, "Coreconnect bus architecture." [Online]. Available: http://crkit.orbit-lab.org/export/453/design/trunk/bfm/bfm_nt_12_1/third_party/doc/crcon_pb.pdf
[27] *DS768: LogiCORE IP AXI Interconnect (v1.02.a)*, Xilinx Inc., March 2011.
[28] "Axi4 bfm." [Online]. Available: https://github.com/sjaeckel/axi-bfm
[29] "Vivado design suite, 2015.1." [Online]. Available: http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2015-1.html
[30] *ZedBoard Hardware User's Guide (v2.2)*, Avnet Inc., 2014. [Online]. Available: http://zedboard.org/
[31] *UG585: Zynq-7000 All Programmable SoC Technical Reference Manual (v1.10)*, Xilinx Inc., February 2015.
[32] "Xillinux." [Online]. Available: http://xillybus.com/xillinux
[33] *PG034: LogiCORE IP AXI Central Direct Memory Access (v4.1)*, Xilinx Inc., November 2015.
[34] *PG059: LogiCORE IP AXI Interconnect (v2.1)*, Xilinx Inc., April 2016.
[35] *PG078: LogiCORE IP AXI BRAM Controller (v4.0)*, Xilinx Inc., April 2016.

Nicole Fern received her undergraduate degree in Electrical Engineering from The Cooper Union for the Advancement of Science and Art (2011) and her PhD degree in Electrical & Computer Engineering from University of California, Santa Barbara (2016) under the advisement of Professor Tim Cheng. She is currently a post-doc at UC Santa Barbara and a Visiting Scholar at Hong Kong University of Science and Technology. Her industry experience includes internships at Cisco and Apple. Her research interests include hardware verification and security, specifically identifying unspecified design functionality susceptible to malicious manipulation and exploring the role of hardware in both undermining and strengthening system security.

Ismail San is an Assistant Professor in the Electrical and Electronics Engineering Department at Anadolu University, Turkey. He received his B.Sc. degree from the same department at Anadolu University (2008); B.Sc. degree from Department of Avionics at Anadolu University (2008); PhD degree from the Electrical and Electronics Engineering Department at Anadolu University (2014). He was a student intern at IBM Zurich Research Laboratory as part of a Great Minds Student Internship program in 2013. He held a visiting research scholar position at UC Santa Barbara, from 2015 to 2016. His research interests include hardware verification and security, design space exploration of application specific processors, high performance computing, and fault tolerant computation.

Çetin Kaya Koç received his PhD in Electrical & Computer Engineering from University of California Santa Barbara. His research interests are in electronic voting, cyber-physical security, cryptographic hardware and embedded systems, elliptic curve cryptography and finite fields, and deterministic, hybrid and true random number generators. Koç is the co-founder of the Workshop on Cryptographic Hardware and Embedded Systems, and the founding Editor-in-Chief of the Journal of Cryptographic Engineering. He has also been in the editorial boards of IEEE Transactions on Computers (2003-2008, 2015-now) and IEEE Transactions on Mobile Computing (2003-2007). Furthermore, he was a guest co-editor of April 2003 & November 2008 issues of the IEEE Transactions on Computers. Koç is the co-author of the three books Cryptographic Algorithms on Reconfigurable Hardware, Cryptographic Engineering, and Open Problems in Mathematics and Computational Science, all published by Springer. In 2007, he was elected as IEEE Fellow for his contributions to cryptographic engineering.

Kwang-Ting (Tim) Cheng received his Ph.D. in EECS from the University of California, Berkeley in 1988. He has been serving as Dean of Engineering and Chair Professor of ECE and CSE at Hong Kong University of Science and Technology (HKUST) since May 2016. He worked at Bell Laboratories from 1988 to 1993 and joined the faculty at Univ. of California, Santa Barbara in 1993 where he was the Chair of the ECE Department (2005-2008) and Associate Vice Chancellor for Research (2013-2016). His current research interests include design automation for photonics IC and flexible hybrid circuits, memristive memories, mobile embedded systems, and mobile computer vision. He has published more than 400 technical papers, co-authored five books, advised 40+ PhD theses, and holds 12 U.S. Patents in these areas. Cheng, an IEEE fellow, received 10+ Best Paper Awards from various IEEE and ACM conferences and journals. He has also received UCSB College of Engineering Outstanding Teaching Faculty Award. He served as Editor-in-Chief of IEEE Design and Test of Computers and was a board member of IEEE Council of Electronic Design Automation's Board of Governors and IEEE Computer Society's Publication Board.

0278-0070 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.