Hardware Trojans in Incompletely Specified On-chip Bus Systems

Nicole Fern, Ismail San, Çetin Kaya Koç, and Kwang-Ting (Tim) Cheng
University of California, Santa Barbara
Email: {nicole, timcheng}@ece.ucsb.edu, {ismail.san, koc}@cs.ucsb.edu

***Abstract*—The security, functionality, and performance of the on-chip bus system is critical in an SoC design. We highlight the susceptibility of current bus implementations to Hardware Trojans hiding in unspecified functionality. Unlike existing Trojans which aim to disrupt normal bus behavior and are often designed for a specific protocol and topology, we present a general model for creating a covert Trojan communication channel between SoC components. From our channel model, which is applicable to any topology and protocol, one can create circuitry allowing information to flow covertly by altering existing bus signals *only* when they are *unspecified*. We give the specifics of this circuitry for AMBA AXI4, then create a 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.**

I. Introduction

Hardware Trojans are a concern for both semiconductor design houses and the U.S. government [4]. 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 [12].

This work focuses on Trojans targeting 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.

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 making Trojan behavior during these cycles difficult to detect.

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.

In Section II we will review the current solutions addressing bus architecture security issues. Section III outlines the threat model, Section IV introduces the Trojan Channel model and circuitry, and Section V further refines this circuitry for AMBA AXI4. A case study examining the overhead of creating a 2-way information leakage channel between slaves in an AXI4-Lite interconnect is presented in Section VI, and Section VII summarizes our results and contributions.

II. Related Work

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, observing bus transactions between other components, corrupting bus data, and allowing a master to access forbidden address ranges [8].

In [8], 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 [7, 13] 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, ARM TrustZone Controllers are commercial IP blocks which provide access control mechanisms to memory regions and bus peripherals by monitoring *valid* bus transactions for violations [1]. Since our proposed Trojans never modify existing or create new valid bus transactions, TrustZone will not detect communication between unauthorized masters and slaves on the Trojan channel or rouge communication between 2 slaves.

Extensive research on formal verification of bus protocols has been performed to ensure deadlock avoidance and fairness [9, 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. [3] for AMBA protocols) and pre-packaged

SystemVerilog assertions suites [5] 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.

III. 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. 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. A complex SoC requires hundreds of engineers to design and test, and relies on third party IP cores 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.

IV. Trojan Communication Channel

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. Figure 1 shows the components necessary to create a Trojan channel in the most complicated bus topology (MUX-based). A Trojan channel can easily be created in other bus topologies by simplifying the circuitry in Figure 1.

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

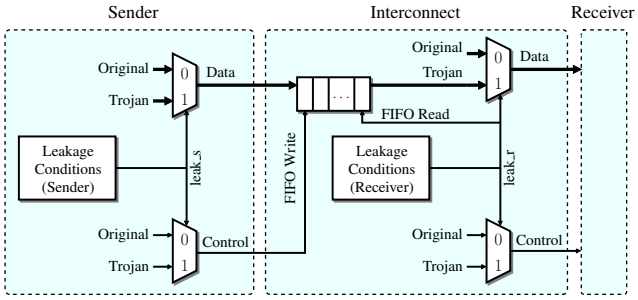

Fig. 1: Trojan Channel in a MUX-based Configuration

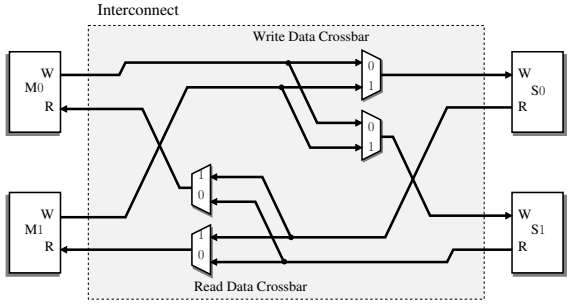

Fig. 2: MUX-based Interconnect Topology [14]

from the sender to the receiver. Since Trojan data is sent 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 1, and like the Trojan data, are mapped to pre-existing data/address/control signals, meaning **no additional interface ports are created by the Trojan**.

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. Further detail is presented in Section V.

The interconnect topology specifies the degree of parallelism between read, write, data, address, and control signals, and the connectivity between masters and slaves [10]. Figure 2 shows the read and write data channels for a MUX-based topology, which supports multiple simultaneous transactions at the expense of extra circuitry. A key feature to note is that since data cannot be snooped by a bus component uninvolved in the transaction due to the routing provided by the MUX circuitry, the Trojan circuitry must provide this extra connectivity.

Simpler bus topologies which remove the MUX circuitry on the data channels and broadcast read and write data to all components are more area efficient, but can only support a single transaction at a time. In a broadcast topology, a Trojan receiver can already observe the sender's data meaning only the circuitry inside the Sender block in Figure 1 is necessary for successful information leakage.

V. Protocol Specific Trojan Channel Definitions

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. We will present this logic in detail for the commonly used AMBA AXI4/AXI4-Lite protocol from ARM. Details are available for the Leakage

Conditions Logic in the AMBA APB protocol, but are omitted due to space limitations.

AXI4 defines 5 independent transaction channels seen at the interface of every master and slave: read address channel, read data channel, write address channel, write data channel, and write response channel [6]. 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. Such mechanisms exist in all bus protocols, and the Leakage Conditions Logic definitions based on AXI4 VALID signals can easily generalize to other bus protocols.

Typically, buses using AXI4 choose MUX-based configurations such as those shown in Figure 2, meaning that all the circuitry in Figure 1 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:

$$leak\_s = troj\_data\_ready \ \& \ \sim \text{VALID}$$

Control Signals: The general criteria for selecting bus signals and their corresponding values to act as Trojan Control signals is that when *leak_s* is asserted, the normal behavior of the signal is predictable, but also unspecified. For AXI4, master-driven signals WSTRB and WLAST both meet these criteria.

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:

$$\text{WSTRB} = leak\_s \ ? \ 4'b1011 : \text{WSTRB\_ORIG}$$

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:

$$\text{WLAST} = leak\_s \ ? \ 1 : \text{WLAST\_ORIG}$$

*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.

Control 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:

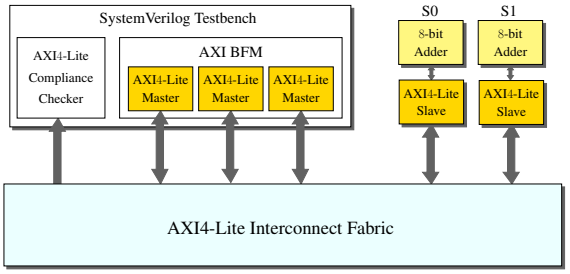

Fig. 3: AXI4-Lite Case Study Verification Infrastructure

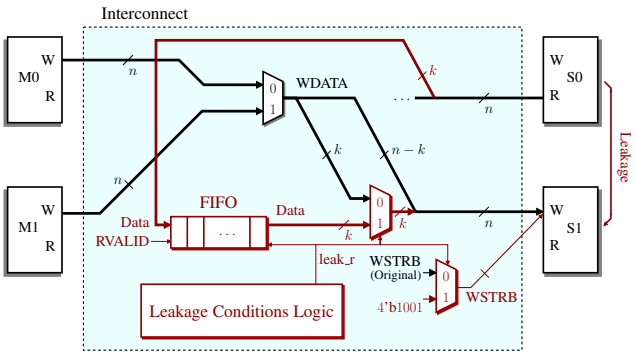

Fig. 4: Trojan Channel Logic for AXI4-Lite Interconnect

$$\text{RRESP} = leak\_s \ ? \ 2'b10 : \text{RRESP\_ORIG}$$

*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:

$$leak\_r = fifo\_not\_empty \ \& \ \sim \text{VALID}$$

VI. Case Study: AXI4-Lite Interconnect

The infrastructure shown in Figure 3 is created, then 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. The two slaves are simple 8-bit adder coprocessors which receive 3 operands to add via an AXI4-Lite bus 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 [2].

The AXI4-Lite Interconnect Fabric IP block used is the LogiCORE IP AXI Interconnect (v1.02.a) from Xilinx [14] configured in Shared-Address Multiple-Data (SAMD) mode (the topology shown in Figure 2). 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

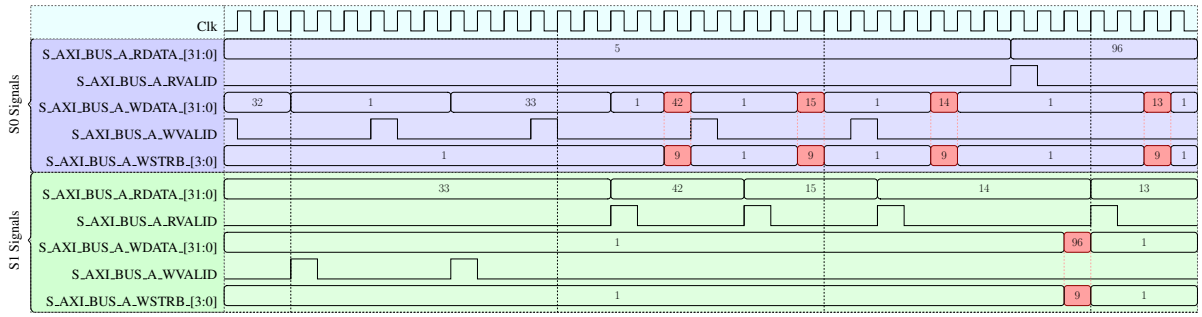

Fig. 5: 2-way Information Leakage Waveform

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

on the bus. Normally WSTRB $==1$, but when information is leaked, WSTRB $==9$.

AXI4-Lite assertions packaged by ARM for protocol compliance checking [5] are active during system simulation. **For AXI4-Lite, there are over 50 assertions, and none of them are violated even when information is flowing through the Trojan channel!**

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 size of interconnect increases as does the overall design area, but the size of the Trojan circuitry does not change meaning the Trojan channel is easier to hide as the complexity of the interconnect and the 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.

VII. 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 interface signals without affecting normal bus functionality. This Trojan can capture and send sensitive data to attacker-controlled modules during idle bus cycles. We illustrate how our general model of Trojan channel communication can be mapped to any bus topology and protocol, and give details for AMBA AXI4. 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. Due to space limitations, detection strategy details are not included, but preventing the insertion of bus Trojans requires refinement of the bus specification, meaning an increase in area/timing/power overhead and design verification effort.

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

Acknowledgements: This work was supported by NSF/SRC STARSS (1526695).

References

[1] ARM TrustZone Controllers: http://www.arm.com/markets/trustzone-controllers.php.
[2] AXI4 BFM: https://code.google.com/p/axi-bfm/.
[3] Synopsys VIP for ARM AMBA: http://www.synopsys.com/tools/verification/functionalverification/verificationip/amba/pages/default.aspx.
[4] S. Adee. The Hunt for the Kill Switch. *IEEE Spectr.*, May 2008.
[5] ARM. *AMBA 4 AXI4, AXI4-Lite and AXI4-Stream Protocol Assertions BP063 Release Note (r0p1-00rel0)*, 2012.
[6] ARM. *AMBA AXI and ACE Protocol Specification*, 2013.
[7] M. Henson and S. Taylor. Memory encryption: a survey of existing techniques. *ACM Computing Surveys*, 46(4):53, 2014.
[8] L.-W. Kim and J. D. Villasenor. A System-On-Chip Bus Architecture for Thwarting Integrated Circuit Trojan Horses. *VLSI Systems, IEEE Transactions on*, 19(10):1921–1926, 2011.
[9] R. Luo and H. Tan. Formal Modeling and Model Checking Analysis of the Wishbone System-On-Chip Bus Protocol. In *Information Computing and Applications*. Springer, 2012.
[10] S. Pasricha and N. Dutt. *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
[11] A. Roychoudhury et al. Using Formal Techniques to Debug the AMBA System-On-Chip Bus Protocol. In *DATE*, 2003.
[12] M. Tehranipoor and F. Koushanfar. A Survey of Hardware Trojan Taxonomy and Detection. *IEEEDT*, 2010.
[13] A. Waksman and S. Sethumadhavan. Silencing Hardware Backdoors. In *IEESP*, 2011.
[14] Xilinx. *LogiCORE IP AXI Interconnect (v1.02.a)*, 2011.