Paper ID# 901650

A TROJAN-RESISTANT SYSTEM-ON-CHIP BUS ARCHITECTURE

Lok-Won Kim [1], John D. Villasenor [1] and Çetin K. Koç [2]
[1] Electrical Engineering Department, University of California, Los Angeles
[2] Computer Science Department, University of California, Santa Babara
May 15, 2009

ABSTRACT

Communications systems are increasingly reliant on system-on-chip (SoC) solutions. As the complexity and size of SoCs continues to grow, so does the risk of "Trojan" attacks, in which an integrated circuit (IC) design is surreptitiously and maliciously altered at some point during the design or manufacturing process. Despite the risks that such an attack entail, relatively little attention has been given in the literature to methods enabling detection of and response to run-time Trojan attacks. In the present paper, we present a Trojan-resistant system bus architecture suitable across a wide range of SoC bus architectures. The system bus detects malicious bus behaviors associated Trojan hardware, protects the system and system bus from them and reports the malicious behaviors to the system CPU. We show that use of this bus and associated embedded software is highly effective in greatly reducing IC Trojan vulnerabilities without loss of bus performance.

INTRODUCTION

Silicon systems in general and communications system-on-chips (SoCs) in particular are getting exponentially more complex, harder to test, and interdependent. Such systems increasingly involve third party IP blocks and are increasingly reliant on outsourced and/or offshore aspects in the design and manufacturing process. With more and more of the system design steps occurring in environments where it is difficult to ensure the security of the design, there is a growing threat of "Trojan" attacks, in which an integrated circuit (IC) design is surreptitiously and maliciously altered at some point during the design or manufacturing process

While the threat of Trojan ICs has received increasing attention in recent years, most anti-Trojan efforts are directed at identifying Trojans during verification and testing, prior to silicon deployment. For example, there is a class of techniques based on comparing measured physical characteristics such as power consumption, timing variations, and layout structures with respect to a "golden model" deemed to be trustworthy [1]–[9]. There are also attempts to design "malicious hardware" in order to demonstrate how significant large-scale attacks can be mounted by the help of hardware [10]. Other methods rely on adding logic which is used to identify authentic chips or test original designs to identify functional defects that may have malicious origins [11], [12]. These approaches, while they are an important part of an overall mitigation strategy, are far from comprehensive in SoC(System on Chip) and SiP(System in Package) applications.

For example, when third party IP blocks are provided using register transfer level (RTL) descriptions, it is likely that there will be no trusted golden model to use for comparison. In addition, the use of increasingly complicated SoC (System on Chip), SiP (System in Package) and MCP (Multi-chip-package) designs provides a would-be attacker with multiple opportunities for the insertion of Trojans, including front-end logic design, place-and-route, floor planning, mask creation, large scale manufacturing, and packaging. Even if all the constituent IP blocks and chips are known to be trustworthy, an attacker could insert a malicious die during the manufacturing process. In this context, it is very difficult to reliably create a trustworthy system-level model against which production samples can be compared. In addition, traditional approaches would not be particularly effective at identifying a true Trojan – an attack designed to remain hidden and inactive until triggered either internally or externally.

While there are many aspects of an SoC that could be targeted by a would-be attacker, in the present paper we consider the specific issue of bus arbitration at the presence of Trojan attacks, which is obviously critical to the overall system operation. The bus arbitration process represents one of the most significant points of vulnerability to run-time Trojans because it is the step that allows master devices (such as processor, different DMA for different communication blocks, various I/O interface blocks) to have access to slave devices (memory controllers, UART, timers, etc.). Once a device is granted mastership of the bus, it can retain this mastership for as long as desired. In systems in which the master devices are behaving cooperatively this is not a problem. However, a Trojan attacker could cause a master device to maintain lock on the bus for arbitrarily long periods of time. The system could continue in the power-on state, but would be locked and unable to function normally. We present a bus architecture that can be used to detect the Trojan attacks and protect systems from the real-time Trojan attacks without affecting on bus performance.

To the best of our knowledge, these aspects of IC Trojan

protection have not been addressed before. Particularly, this is the first paper that deals with 1) real-time Trojan attacks and real-time protections against such attacks, 2) system protection via a Trojan-resistant SoC bus architecture. Our proposed architecture accomplishes protection without incurring high costs on the bus resources and performance. The remainder of this paper is organized as follows. We first provide a brief overview of a conventional SoC bus. Next, the proposed SoC system bus architecture and its design goals are described. Next, we present a detailed description of the key techniques that underlie the proposed SoC bus architecture in the context of the AMBA bus architecture. Finally, we present an interrupt handling method to receive information about Trojan behaviors to exclude the Trojan IP from system by cutting the connection between system bus and IP.

Proposed Bus Architecture

Normal SoC Bus Structure and Operation

While an SoC bus is not a real physical bus, it performs the functions associated with a physical bus of interconnecting a processing core to the surrounding interface logic. Examples of SoC buses include AMBA (Advanced Micro-controller Bus Architecture) from ARM, CoreConnect from IBM, Avalon Bus Specification from Altera, Wishbone from Opencores, and STBus from STMicroelectronics [13]–[17] While each bus is characterized by different architectures, performances, advantages and protocols, they address the similar function of mediating bus mastership in multiple bus master environments. This involves translating incoming addresses from a current master into selection signals to slave IPs, transferring data from a selected slave IP to a current master, bridging between multiple bus levels. Fig. 1 shows a conventional bus arbiter connected, in this example, to three master devices and three slave devices. A device desiring bus mastership can issue a request using its Mastership_REQ line. If mastership is granted, the arbiter will inform the requesting device using the appropriate GRANT line, and simultaneously, will set MASTER ID so that the multiplexors will allow signals from the selected master to be provided to the slave devices. Each master also has an LOCK signal that it can optionally assert when it needs to lock the bus. Such a need can arise, for example, when there is a time constraint that forces the master device to complete a task (such as transferring a received packet to main memory) immediately. When the bus is locked, the arbiter informs the other potential master devices using the MASTER LOCK signal. In cases where the master is performing a less time-critical task, such as when the CPU is fetching an instruction from memory, the CPU will typically not assert LOCK. This allows the arbiter to take mastership away from the CPU and grant it to a different device, such as a modem, should a time-critical task arise.

In a normally operating system, mastership passes among different devices in accordance with the arrival of data and the various demands and priorities of the master devices. In

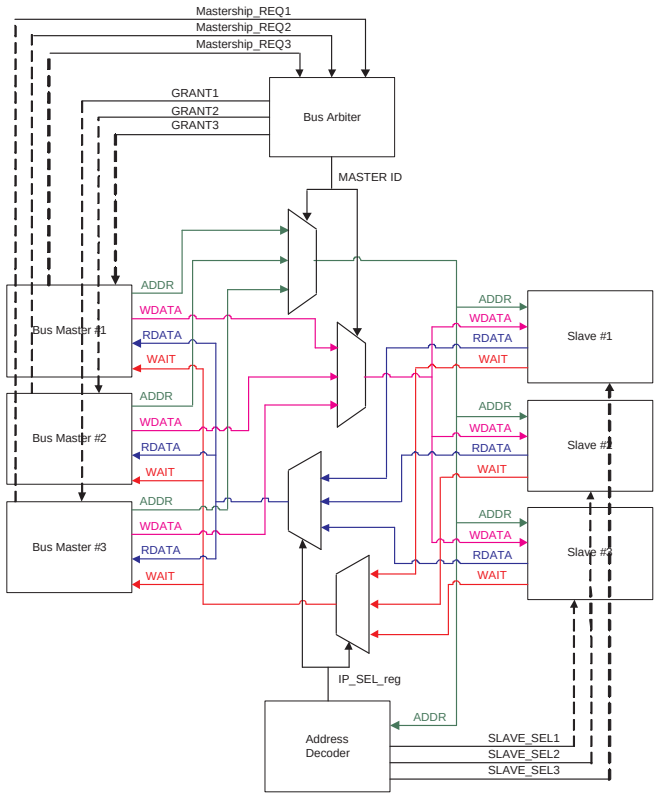

Fig. 1. A high-level view of conventional SoC bus interconnections, showing master and slave devices, an arbiter, an address decoder, and various multiplexers.

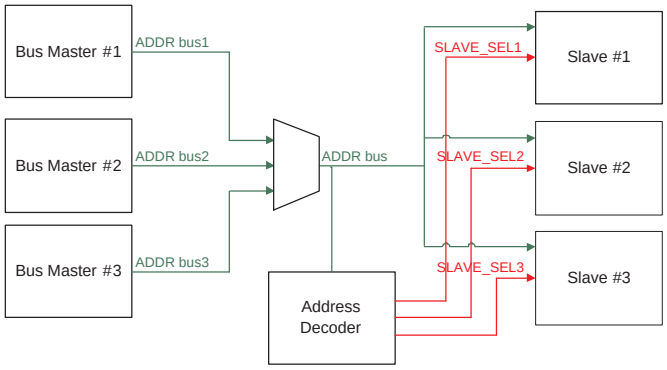

Fig. 2. A conventional address decoder and its connection. The decoder examines the MSBs of the address and selects the appropriate slave

a Trojan attack, however, a malicious master could request and receive bus mastership, and then proceed to lock the bus for an indefinite period of time. Once in control of the bus, it could halt the system by blocking the CPU from fetching code instructions and loading data. In addition, it could access normally restricted system addresses in order to gain access to confidential and control system operating modes.

Address decoder

The function of an address decoder is to receive address signals arriving from a master and to make a selection of an

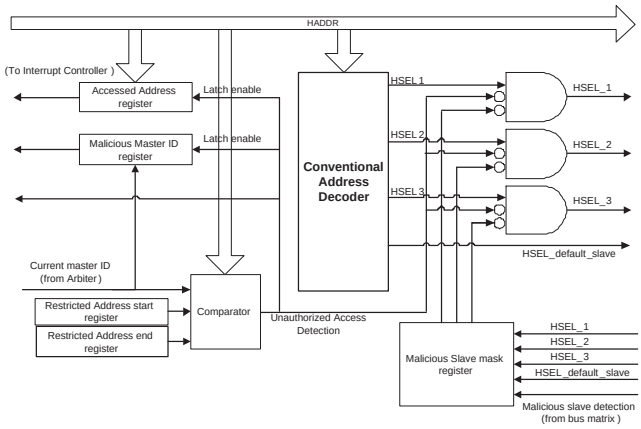

Fig. 3. The proposed secure address decoder

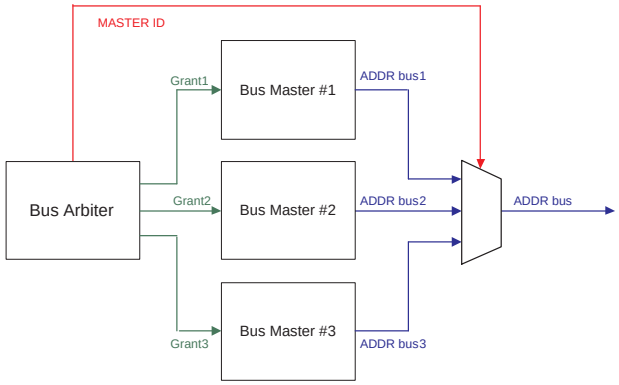

Fig. 4. Bus master grant signals and mux connections.

appropriate slave device. Fig. 2 shows an example of the conventional (in this case based on AMBA) address and address decoding connections. The decoder logic contains combinatorial logic to examine the most significant bits (MSBs) of the address and activate the corresponding slave block. For example, if the MSBs indicate an address corresponding to Slave #1, the address decoder will set the SLAVE_SEL1 signal appropriately so that the LSBs will be used to read data from Slave #1.

Fig. 3 shows a secure address decoder containing a conventional address decoder as well as additional logic to 1) detect an attempt by a malicious bus master to access a restricted address, and 2) block normal masters from inadvertently accessing malicious slaves. The "Restricted Address Start Register" and "Restricted Address End Register" contain address values to define restricted address ranges. The embedded software configures these values in an initialization step or dynamically re-configures them in run-time. The comparator receives address signals from the master, compares them with the restricted address register, and upon detection of an unauthorized access attempt generates an "Unauthorized Access Detection" signal that disables all the slave select signals except for a single default slave. The Unauthorized Access Detection signal is also connected to the interrupt controller as an interrupt source so that CPU can handle the malicious behavior accordingly. The CPU initiates an interrupt service routine to identify information about the malicious master and the unauthorized access address, and initiates an appropriate countermeasure. In addition, the identity of the malicious master is stored in the malicious master mask register so that future attempts to obtain bus mastership can be handled accordingly.

The secure address decoder also blocks access by normal masters to malicious slaves. A malicious slave could attempt to halt bus operation through continuously asserting a wait. This behavior can be detected using the bus matrix block, and information regarding the malicious slave information recorded into the malicious slave mask register. The Trojan slave mask value disables bus connection to the Trojan slave mask. When a master attempts to access a malicious slave, the address decoder instead diverts the access into a default slave containing empty address ranges, thus effectively excluding the malicious slave block from the bus system.

Arbiter

As noted earlier, and arbiter is used to ensure that only one master has access to the bus at a time. The arbiter performs this function by observing (possibly simultaneous) bus-mastership requests to use the bus, deciding which requester has the highest priority, and granting mastership to one master at at times. There are multiple approaches for bus arbitration, including round-robin, fixed priority, or a combination of round-robin and fixed priority. Fig. 4 shows the grant signals and address multiplexor connections of a conventional bus standard. Upon receiving a grant signal from the arbiter, the selected master can then provide an address via the multiplexor to the appropriate slave. The arbiter also provides and MASTER ID signals to the multiplexor to ensure the ADDR information from the selected master is provided to the slaves.

Fig. 5 shows the proposed arbiter. The arbiter contains a register, counter and combinatorial logic and performs the functions of 1) detecting and nullifying malicious bus locking by a Trojan master, and 2) avoiding grants of bus mastership to known Trojan masters.

In principle, the bus master lock is used to protect data integrity when a master needs to perform a time-constrained transfer through the bus. However, a Trojan master could obtain exclusive possession of the bus mastership through improper use of the LOCK bus signal. This would exclude other masters from gaining access to the bus, and would also prevent an interrupt from being used to switch bus mastership. To address this, several logic functions are included in the arbiter of Fig. 5. A counter counts the number of clocks for

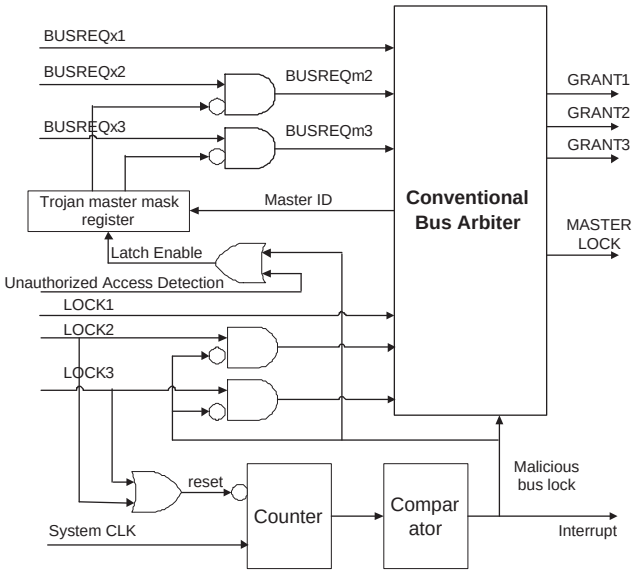

Fig. 5. The proposed arbiter.

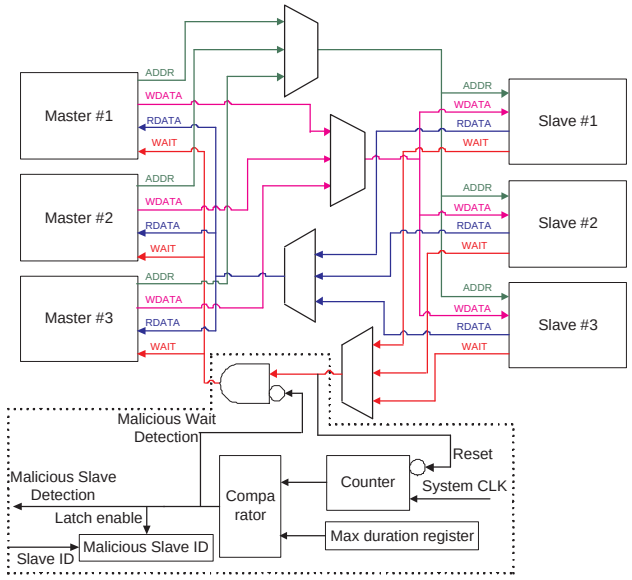

Fig. 6. The proposed bus matrix

which LOCK signals are active during each use of the bus by a master. When the counter exceeds a threshold, a malicious bus lock signal is activated. The threshold can be set on an application-specific basis, and can also be varied adaptively during operation as a function of specific SoC conditions, thus minimizing the probability of a false alarm. Upon activation of the malicious bus lock signal, the master mask register forcibly gates the lock and request signals and the arbiter returns to normal operation. The arbiter of Fig. 5 also receives the Unauthorized Access Detection signal from the address decoder as explained above. The arbiter saves the MASTER ID of the malicious Master into the master mask register, so that future attempts by this master to access the bus can be denied.

Bus matrix

Fig. 6 shows a secure bus matrix. In normally operating system, the bus matrix enables the connection between the appropriate master and slave signals in accordance with the signals from the arbiter and decoder, passing data, address, and transaction status. The secure bus matrix detects, blocks, and reports malicious wait signals from a Trojan slave. When a slave needs additional time to finish a data writing or reading operation instructed by the master, it can assert the wait signal so that the master knows to wait for completion. For example, if the bus is running at 200Mhz and memory access is clocked at 50Mhz, the memory controller can assert the wait signal for four clock cycles when a master needs to read from or write to memory. A Trojan IC could utilize the wait signal to halt the system, thus forcing the master to wait indefinitely and preventing the arbiter from switching mastership. In the bus matrix shown in Fig. 6, a counter is used to detect a malicious wait in a manner analogous to the counter described above for detecting malicious bus lock. When the wait exceeds a

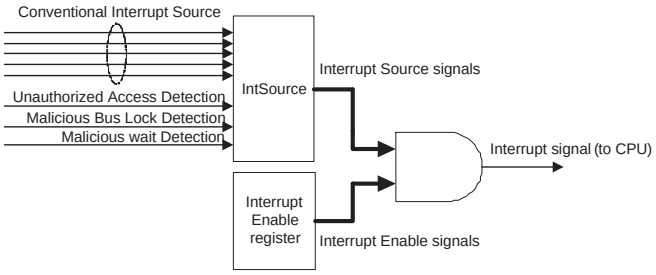

Fig. 7. Interrupt controller, modified to handle signals identifying the presence of a Trojan

threshold, a malicious wait signal is generated and used to nullify the wait signal, and also used as a latch enable signal for the slave mask register to identify the current slave as malicious.

POST-DETECTION SOC OPERATION

The approaches described above enable detection of Trojan behavior and temporary or permanent quarantining of master or slave devices known to be malicious. However, if the act of excluding a malicious master or slave leaves the SoC unable to function, the Trojan attack will still have succeeding in halting the system. Thus, it is important to not only quarantine malicious devices, but to maximize the ability of the SoC to continue operation despite the presence of a Trojan. The appropriate response depends strongly on the specific nature of the Trojan.

In addition to their use in blocking malicious blocks, the “Unauthorized Access Detection”, “Malicious Bus Lock Detection” and “Malicious wait Detection” signals can also be used in conjunction with the system interrupt. Fig. 7 shows

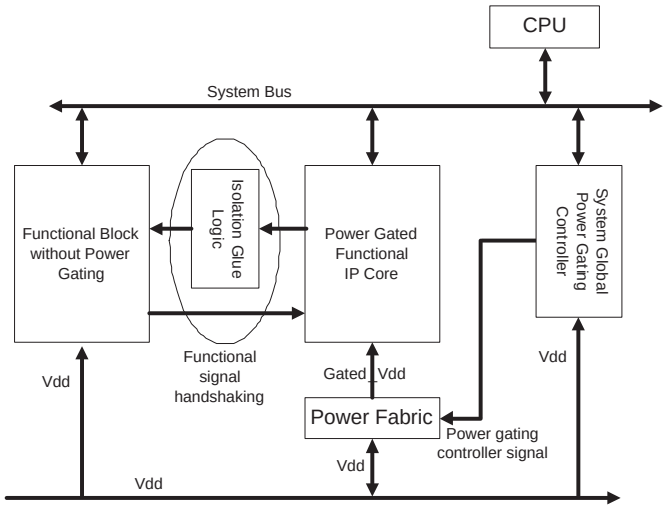

Fig. 8. A SoC architecture with power gating

TABLE I
AREA DISTRIBUTION OF THE EXAMPLE SOC.

Area	Soft/hard macros	Pads	Standard cells	Total Chip size
(mm^2)	14.028	2.736	6.631	29.89

a simplified interrupt controller which connects the detection signals as interrupt sources. When malicious behavior is detected in one of the proposed bus components, at first, the behavior is temporary blocked. The corresponding detection signal triggers a system interrupt, causing the CPU to jump to a vector address corresponding to an appropriate interrupt handler routine. In the interrupt handler routine, the CPU utilizes a specific interrupt service routine corresponding to the detection signal. Actions taken can include reporting malicious behaviors to users or host systems.

In addition, or alternatively, the CPU can assert the reset signal of the Trojan IC block to initialize all registers inside the block, turn on clock gating on the block to halt the Trojan block's operation, or turn on power gating on the block power down every element of the Trojan block. Fig. 8 shows a block diagram of an SoC with power gating. The Power Switching Fabric includes a unified VDD mesh for the Power Gated Functional Block. This isolates the functional block from power sources using the Power Gating Controller. When the block goes into power gated status, output signals of the power gated block must be tied to Vdd or GND. Otherwise, a block which receives the signals as input signals may experience problems because of the floating inputs. The Isol block performs the appropriate signal ties to handle this.

DESIGN EXPERIMENTS

To explore the hardware costs of some of the approaches described above, the AMBA-based SoC shown in Fig. 9 was used. We have chosen the AMBA AHB from ARM

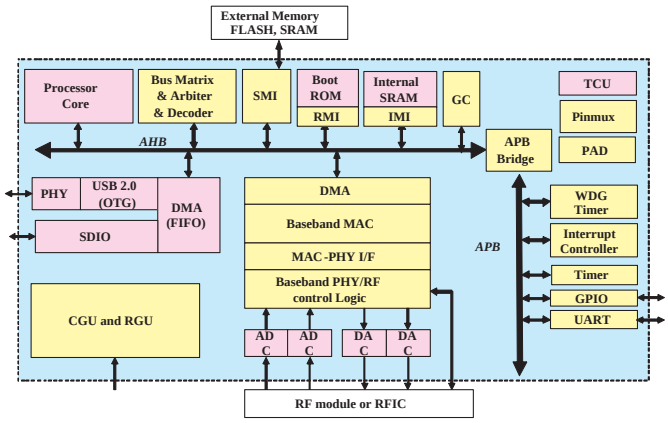

Fig. 9. The AMBA-based SoC used for experiments. This SoC contains approximately 4 million logic gates.

TABLE II
ADDITIONAL GATE COUNT OF THE PROPOSED ARCHITECTURES.

Function	Additional gate count
Arbiter	124
Address decoder	116
Bus matrix	186
Data integrity checker for memory	306
Total additional gates	732

because it is a very widely used SoC bus. The SoC in Fig. 9 contains approximately 4 million logic gates and includes a processor core, SoC bus components, various memory controllers, various interface blocks, a baseband processor, timers, interrupt controller, general purpose IO and UART.

The RTL description of the system was synthesized using a 90 nm technology library. The processing core and system bus operates at 132Mhz. The number of standard cells used in this SoC is 574472, and the total number of logic gates is 4010814. Table I shows the area distribution of the SoC. Table II shows the additional gate count costs associated with the various techniques described earlier. As the table shows, the total increase in gate count is less than 800 for the specific implementation used here.

It is also important to consider the potential for delays that may be introduced by this additional logic. For the modified address decoder, there is a delay due to the three-input AND gate in addition to the delay of conventional address decoding. The new arbiter and bus matrix each contain an additional two-input AND gate with its associated delay.

All of the above delays are negligible in the context of the overall SoC design. In the overwhelming majority of cases, the additional delay would not have any impact with regard to the ability to meet performance constraints. In the unlikely event

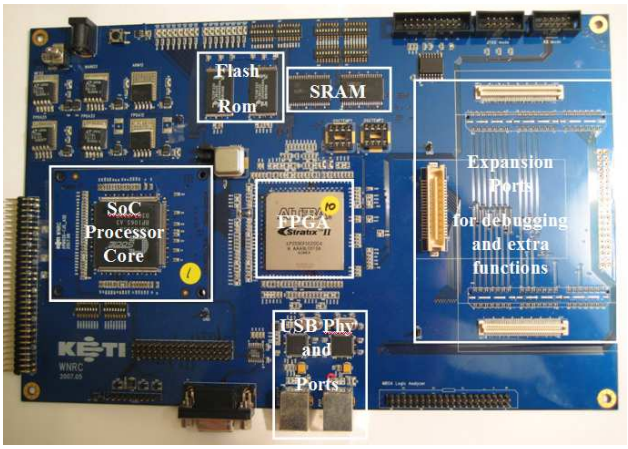

Fig. 10. An AMBA-based development card for emulation of the proposed architectures

that the ability to meet these constraints is impacted, this can be automatically handled in the synthesis stage through the use of faster logic gates.

Fig. 10 shows an AMBA-based development emulation card used for implementation. The anti-Trojan logic elements listed in Table II were designed in RTL and mapped to this board. Various virtual attacks were implemented to, for example, attempt to access restricted address ranges, assert continuous lock and wait signals during processing. On this emulation card and in RTL simulation, the detection, mitigation and system interrupt functions to detect and mitigate these attacks were successfully verified.

CONCLUSIONS

We have presented a bus architecture that is resilient to Trojan attacks. By constructing the new bus architecture around a core of traditional bus elements such as the arbiter, address decoder, bus matrix, etc., the design remains compatible with traditional systems. Mechanisms have been presented to identify malicious attempts at bus locking, unauthorized memory accesses, and malicious use of wait signals which, if left undetected, could freeze the operation of the entire SoC. Master and slave devices engaging in malicious behavior are identified and quarantined. Depending on the nature of the Trojan attack, the operation of the SoC can be modified on the fly, thereby enabling the SoC to continue to maintain full or partial functionality despite the attack.

REFERENCES

[1] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using IC fingerprinting," in *IEEE Symposium on Security and Privacy (SP'07)*, 2007, pp. 296 – 310.

[2] F. Wolff, C. Papachristou, S. Bhunia, and R. Chakraborty, "Towards Trojan-free trusted ICs: problem analysis and detection scheme," in *Proceedings, Design Automation and Test in Europe (DATE'09)*, Munich, Germany, March 10-14, 2008, pp. 1362–1365.

[3] J. Li and J. Lach, "At-speed delay characterization for IC authentication and Trojan horse detection," in *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'08)*, 2008, pp. 8 – 14.

[4] X. Wang, H. Salmani, M. Tehranipoor, and J. Plusquellic, "Hardware Trojan detection and isolation using current integration and localized current analysis," in *IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems (DFTVS'08)*, 2008, pp. 87 – 95.

[5] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'08)*, 2008, pp. 15 – 19.

[6] Y. Jin and Y. Makris, "Hardware Trojan detection using path delay fingerprint," in *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'08)*, 2008, pp. 51 – 57.

[7] R. Rad, J. Plusquellic, and M. Tehranipoor, "Sensitivity analysis to hardware Trojans using power supply transient signals," in *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'08)*, 2008, pp. 3 – 7.

[8] M. Banga and M. S. Hsiao, "A novel sustained vector technique for the detection of hardware Trojans," in *22nd International Conference on VLSI Design*, 2009, pp. 327 – 332.

[9] M. Abramovici and P. L. Levin, "Protecting integrated circuits from silicon Trojan horses," January/February 2009, http://www.mil-embedded.com/articles/id/?3748.

[10] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and implementing malicious hardware," in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET)*. San Francisco, California: USENIX Association, 2008, pp. 1–8.

[11] T. Kean, D. McLaren, and C. Marsh, "Verifying the authenticity of chip designs with the DesignTag system," in *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'08)*, 2008, pp. 59 – 64.

[12] R. S. Chakraborty, S. Paul, and S. Bhunia, "On-demand transparency for improving hardware Trojan detectability," in *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'08)*, 2008, pp. 48 – 50.

[13] D. Flynn, "Amba: enabling reusable on-chip designs," *Micro, IEEE*, vol. 17, no. 4, pp. 20 – 27, 1997.

[14] A. Goel and W. R. Lee, "Formal verification of an ibm coreconnect processor local bus arbiter care," in *Design Automation Conference, 2000. Proceedings 2000. 37th*, 2000, pp. 196 – 200.

[15] F. Lin, H. Wang, and J. Bian, "Hw/sw interface synthesis based on avalon bus specification for nios-oriented soc design," in *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on*, 2005, pp. 305 – 306.

[16] X. Xing, C. Zezong, J. Jing, and K. Hengyu, "Porting from wishbone bus to avalon bus in soc design," in *Electronic Measurement and Instruments, 2007. ICEMI '07. 8th International Conference on*, 2007, pp. 862 – 865.

[17] S. Murali and G. D. Micheli, "An application-specific design methodology for stbus crossbar generation," in *Design, Automation and Test in Europe, 2005. Proceedings*, 2005, pp. 1176 – 1181.