SAFER PATH: Security Architecture using Fragmented Execution and Replication for Protection Against Trojaned Hardware

Mark Beaumont, Bradley Hopkins and Tristan Newby
Defence Science and Technology Organisation
Adelaide, Australia
Email: Mark.Beaumont@dsto.defence.gov.au

***Abstract*—Ensuring electronic components are free from Hardware Trojans is a very difficult task. Research suggests that even the best pre- and post-deployment detection mechanisms will not discover all malicious inclusions, nor prevent them from being activated. For economic reasons electronic components are used regardless of the possible presence of such Trojans.**

We developed the SAFER PATH architecture, which uses instruction and data fragmentation, program replication, and voting to create a computational system that is able to operate safely in the presence of active Hardware Trojans. We protect the integrity of the computation, the confidentiality of data being processed and ensure system availability. By combining a small Trusted Computing Base with Commercial-Off-The-Shelf processing elements, we are able to protect computation from the effects of arbitrary Hardware Trojans.

I. Introduction

Hardware Trojans are undesired, malicious modifications to electronic circuits. They are designed to compromise the operation of systems containing the circuits, presenting a persistent threat to the security of the infected hardware, as well as any software executing on that hardware. Hardware Trojans can be inserted into an electronic circuit at any stage of development, manufacturing, or distribution [1].

Hardware Trojans may operate continuously, or may lie dormant, waiting to be activated before performing their function. This can include modifying the behaviour of the electronic circuit, degrading its performance, or compromising sensitive information that is processed or stored by the circuit.

Given the proliferation of electronic devices, and their underpinning of our financial, military, industrial and transportation sectors, the devastation that could be caused by a carefully designed Hardware Trojan is immense [2]. This threat has given rise to a flurry of research over the past five years, as detailed in [3] and [4]. Most research has focused on preventing the modification of an Integrated Circuit (IC), or methods for detecting a Hardware Trojan once it has been inserted into an IC. However, there can be no guarantee that an IC is free of Hardware Trojans before it is deployed [5], nor, given the large state-space of trigger mechanisms [6] is it possible to reliably prevent the activation of a Hardware Trojan in a device, despite current best efforts [7][8].

To maintain a capability edge, the Australian Military must procure and use Commercial-Off-The-Shelf (COTS) electronic components. Without being able to guarantee components free of Hardware Trojans, we must pursue methods for safely operating in their presence, assuming they will be active within our systems.

We have developed SAFER PATH, a new architecture that takes a generic, defence in depth approach to combating Hardware Trojans in COTS components. SAFER PATH allows the use of these *untrusted* components, including ICs and third-party Intellectual Property (IP) from different manufacturers, for general purpose computation. Computational integrity and availability is achieved by multiple processing elements (PEs) simultaneously voting on a computer program's execution. Further, data confidentiality is protected by fragmenting this execution over different sets of PEs.

The architecture is underpinned by a Trusted Computing Base (TCB) that controls the replication, voting, and fragmentation. Placing effort in the development and accreditation of this TCB provides the flexibility to incorporate many different COTS PEs into this architecture.

Section II introduces our threat model and assumptions. Section III discusses related work. Section IV details our solution. Section V and Section VI describe our experimental implementation and results. Section VII discusses some potential extensions while Section VIII summarises our work.

II. Threat Model

Many critical systems rely on embedded microprocessors to control their operation, including SCADA systems, telecommunications networks, and military cryptographic devices.

These processors are typically procured in a global market from a large array of vendors, leaving their design, implementation, and fabrication details untraceable. The pace of technological development has meant that even the most advanced military systems now incorporate these same COTS processors [9].

We assume the presence of arbitrary, undetected Hardware Trojans within these processors. These Hardware Trojans can be activated at any time and the repercussions of their activity is unknown a priori. Under normal circumstances these Trojans could undermine the security of systems through attacks against the integrity of computations being performed, the availability of processing resources, and the confidentiality of data being processed.

Rather than develop and accredit a single trusted processor, we want to augment these untrusted COTS processors with a small subset of trusted logic. This combination can then be used to do the job of a trusted processor, avoiding the unwanted effects of any resident Hardware Trojans. This approach saves considerable effort in the accreditation process and allows the use of the latest COTS components to track technological advances.

All PEs in our architecture have identical specifications. From an input, output and program execution perspective their normal (untrojaned) operation is indistinguishable. If all PEs were also physically identical then they would all be identically vulnerable to the inclusion of Hardware Trojans.

To counter this threat we assume that physical variability can be introduced into the pool of PEs. This can be achieved by utilising unique RTL descriptions of the same PE specification created by independent design vendors, using different sets of design tools. These descriptions can then be fabricated at independent facilities, utilising different processes, geometries and cell libraries. Large scale collusion would then be required to insert the same, or colluding, Hardware Trojans into the variant PEs.

We have focussed on countering Hardware Trojans within processing components. The architecture does not protect other system elements such as memory from Hardware Trojans. However, research exists into protection elements such as memory data guards [10], and a defence in depth approach utilising these elements would complement the architecture. Ultimately, SAFER PATH could be used as a drop-in replacement for the embedded processor in any one of a number of systems.

As our focus is on protecting against Hardware Trojans, this work ignores the threats posed by malicious software. Our intent is to enable hardware to correctly execute the software that it is given.

III. Related Work

Although the majority of research into Hardware Trojans has focused on prevention and detection, some recent research has looked at maintaining the secure operation of an electronic system in the presence of Hardware Trojans. This research can be broadly organised [6] into: Data Guards; Novel Architectures; Reconfigurable Logic; and Replication and Voting.

Data Guards attempt to prevent data from being used as a trigger for activating Hardware Trojans and prevent data from being leaked through a specific interface. Waksman and Sethumadhavan [7] propose bus scrambling, homomorphic cryptography, and time-guards. Bloom et al. [10] use a double memory guard to securely pass encrypted data to and from memory. Das et al. [11] detect real-time modifications to memory accesses. Beaumont et al. [12] scramble hard disk transactions as part of a Silicon Security Harness [2].

BlueChip [8], uses a defensive strategy that identifies and removes untrusted circuits from a design, replacing them with a hook to a software exception handler. Further research has since shown that circuits can be created that evade the untrusted circuit detection mechanisms [13]. Abramovici and Bradley detect Hardware Trojan tampering attacks at run-time through the addition of self checking logic to an IC's design [5].

Baumgarten, Tyagi and Zambreno use reconfigurable logic barriers within a design to prevent both the activation and operation of Hardware Trojans added during the manufacturing stage of an IC [14].

McIntyre et al. [15] spawn functionally equivalent but variant software processes on multiple identical processing elements, dynamically adjusting the trust in an individual processing element based on compared outputs. Similarly, we utilise variability and voting, however we use identical software operating on variant processors and combine this with data fragmentation to protect against Hardware Trojans.

Newgard and Hoffman [16] use a tightly coupled dual-processor lock-step configuration to check every instruction that is executed by both processors. SAFER PATH utilises a modified lock-step architecture, and can include any number of processors. Instructions are not checked by every processor, but rather a majority consensus is reached through a voting mechanism. Further, this voting mechanism has been extended to include peripherals of the processor.

We use replication and fragmentation to limit access to both program code and data. Trouessin et al. [17] investigated Fragmentation-Redundancy-Scattering (FRS) techniques to preserve data confidentiality in a distributed system, protecting against software threats. Data was split into small enough fragments that each piece on its own contained little information. Our work utilises similar ideas to ensure underlying untrusted hardware only has access to small windows of an executing program. However, instead of trusted software fragmenting data so that it can be operated on remotely, and recombining the data when the results are returned, we simply restrict access to a single data store. Processors are only allowed to access the subset of the data that they need in order to perform a given computation.

IV. Architecture

SAFER PATH consists of a small TCB and a pool of PEs over which a computer program is executed. The TCB facilitates replication and fragmentation of program execution, and resides between the PEs and program memory (Fig. 2).

PEs may typically be a microprocessor, or other processing component that loads executable code from an external memory. The architecture is not restricted by any particular configuration and the PEs may be bare die, packaged ICs, or implemented in reconfigurable logic. The PEs are operationally identical, but varied in their design and fabrication.

A. Replication and Voting

Program code is executed simultaneously (replicated) on a subset of the PEs, termed a *bank*, from a single program memory. All memory accesses, encompassing all instruction and data reads and writes, are voted on by the bank of PEs. This prevents a PE infected with a Hardware Trojan from accessing regions of memory independently of other bank PEs. Various schemes can be used to combine memory accesses from multiple PEs, and thereby generate the *voted* memory access. Figure 1 shows how multiple PEs in a bank connect to a single program memory through voting logic.

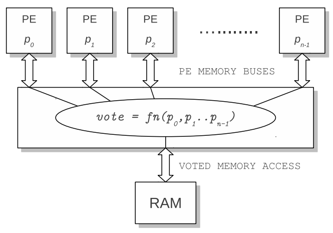

Fig. 1. Processing Element bank voting on memory accesses.

PE replication and voting on program execution provides integrity in the executed program, availability of the system, and prevents a Hardware Trojan from utilising memory accesses for direct data leakage.

B. Fragmented Execution

The PEs in the executing bank still have access to all instructions executed and data operated on during program execution. While voting prevents the PEs from leaking any information through memory accesses, trojaned hardware may still utilise other channels [18] to compromise sensitive data. To further protect data confidentiality, we switch a program's execution between different banks of PEs. Each bank runs a *fragment* of the program code before execution is switched to a different bank. A fragment may be as small as a single instruction or as large as an entire program. This switching is similar to software context switching, with the added characteristic that when a program is switched, it restarts execution on a different bank of PEs. When enough banks exist, any individual PE will only ever have access to small fragments of a program, limiting the amount and coherence of sensitive data a Hardware Trojan may leak.

A new bank is assembled from the pool of PEs at each occurrence of an execution switch. The make-up of the bank and the order of switching may be either fixed or random, trading the complexity of the TCB logic against the flexibility of the architecture. The generic SAFER PATH architecture is illustrated in Figure 2.

The manner and frequency of bank switching affects both the performance and security of the architecture. Best protection is achieved by switching the program execution to isolate sensitive data to particular banks of PEs, but more frequent switching also has a larger impact on performance.

Switching may be achieved through hardware, software or hybrid mechanisms, and may be enforced at regular, programmatic or random intervals. Hardware enforced switching logic would need to be a part of the TCB, increasing both the size and complexity of the trusted hardware. In Section V

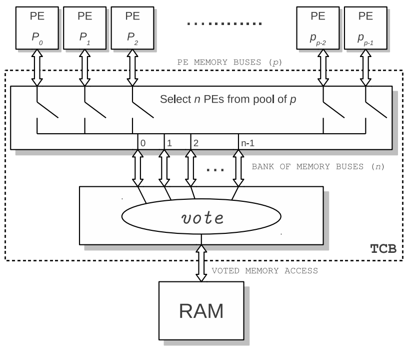

Fig. 2. SAFER PATH architecture: replication, voting and fragmentation.

we describe a hybrid mechanism that relies on the executing program to initiate the switching.

C. Trusted Computing Base

The TCB consists of the voting and switching logic. Our goals for architecting the TCB include amenability to accreditation, a generic design for use with different COTS ICs, and logic of smaller size than individual PEs. The TCB may be implemented in a variety of ways, including combined on a single die with PEs, in reconfigurable logic, or dedicated silicon.

Any TCB elements need to be designed, developed, and fabricated free from Hardware Trojans. The small size and simplicity of our TCB makes this a more tractable problem than developing a complete trusted processor.

V. Implementation

We have developed a prototype implementation of SAFER PATH consisting of a pool of 30 *Leon3* 32-bit SPARCv8 processors [19]. In this implementation the processors have no physical variability, but, as this variability is simply a defence against identical Hardware Trojans being introduced into the processors, the operation and analysis of the architecture remain valid. The processors, a TCB implementing replication, voting, and fragmentation, and a 128 kilobyte RAM were all instantiated in a single Xilinx XC6SLX150T Spartan 6 Field Programmable Gate Array (FPGA).

A. Leon3 Configuration

The Leon3 processor is a configurable IP core that uses the AMBA [20] AHB bus as its main memory bus and the AMBA APB bus as a peripheral bus. Each individual processing core is instantiated with the configuration shown in Figure 3. The Debug Support Unit (DSU) is attached to a single core and is used to load program code into the RAM. The RAM and APB bus UART are shared between all Leon3 cores. The RAM is the only memory in our configuration and contains both program code and data, as well as being used for stack and heap implementations.

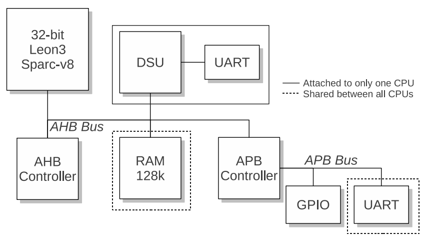

Fig. 3. Leon3 configuration used for prototyping.

B. Processor Banks

The pool of Leon3 processors is divided into 10 banks of three, although the number of banks, the number of processors per bank, and hence, the total number of processors are configurable at synthesis time. In our prototype, the individual processors allocated to each bank are fixed.

All Leon3 processors within the FPGA run synchronously. Given the same program code, they perform the same memory accesses at the same time, allowing a bank of processors to share access to, and execute code from, the single RAM. The TCB logic shares the RAM between processors in a bank by combining RAM accesses at the AHB bus level. The AHB Bus is bi-directional, with separate input and output channels to RAM. The output channel is separately connected to each processor in a bank, preventing direct links between banked processors. The input channel (an instruction fetch or a data read or write) is voted on across the input channels from each processor in a bank. Voting on bank $l_0..l_{n-1}$, is shown in Figure 4.

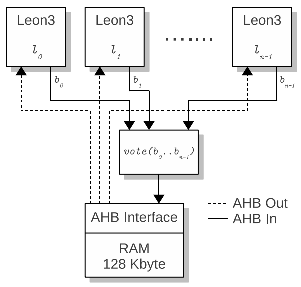

Fig. 4. Voting on the AHB Bus.

We used a comparison based voting scheme on the AHB input bus. For our implementation, with banks containing three processors with AHB input channel buses b_0, b_1 and b_2:

$$vote(b_0, b_1, b_2) = \begin{cases} b_0 & \text{if } b_0 = b_1 \vee b_0 = b_2 \\ b_1 & \text{if } b_1 = b_2 \\ b_{null} & \text{otherwise} \end{cases} \quad (1)$$

Voting happens in real time and has no effect on memory access times or overall program performance.

In our prototype, if two out of the three processors have the same AHB bus input then that memory access occurs. If all three are different, the memory access is voided and the processors are reset. The voting algorithm can be modified depending on the number of processors in a bank and security guarantees required.

A useful system requires input and output (IO). The APB UART is shared across a bank in a similar manner to the AHB RAM, allowing voted control and usage for serial communications. While each processor has its own GPIO unit, the outputs from the individual GPIO lines are voted on to create banked GPIO outputs.

Code executing on the processors is oblivious to the underlying banking and voting that is occurring for both the memory accesses and the IO peripherals.

C. Processor Execution Switching

Each processor bank appears to the wider system as a single processor executing the stored program code. A bank executes a fragment of program code before execution is switched to the next bank. To facilitate switching, the state of current bank processors is transferred to processors in the next bank. This is achieved in a similar manner to a context switch, as is typically used for multitasking on a modern CPU, combined with a hardware switch that moves program execution from one processor bank to the next. Program execution is switched by switching access to the RAM between banks.

In our implementation, execution switching is triggered by the program and is hence under the direct control of the programmer. We developed a C-code switching routine that begins execution on one processor bank and completes execution on another. The routine first saves the state of the current processor, i.e. its registers, to memory, along with the address of the first instruction in the state restoration code. A dedicated GPIO line is then toggled to signal the TCB hardware to switch RAM access to the next bank. The GPIO line is voted on by all bank processors, preventing a Hardware Trojan from interfering with the switching. When the TCB detects the GPIO line change, the bank is disconnected from the RAM.

After disconnecting the first bank, a new, pseudo-randomly chosen bank of processors is connected to RAM. Immediately after this, the TCB resets *all* processors. On being reset, processors attempt to boot from the AHB RAM. The banks that are not connected to RAM will not boot and will be in an error state until they are reset. The bank that is connected to the RAM executes program boot code from the base of RAM.

The boot code first reads in the saved address of the state restoration code from RAM, checking if this is the first boot (address=`0x00000000`), otherwise it jumps to the saved address (Listing 1).

```
start:  set     saved_address, %r5
        ld      [%r5], %r7
        cmp     %r7, 0
        be      first
        nop
        jmp     %r7
first:
```

Listing 1. Leon3 processor boot code.

Execution then continues from this address, loading the saved processor state from memory, to complete the execution switch. As all memory accesses are voted on, the state is saved and restored without concern for Hardware Trojan manipulation.

D. Software

The programmer has control over when and how the execution switching is implemented. A programmer can choose exactly which registers are saved and restored, only restoring enough registers for a bank to execute its code fragment. Minimising this context decreases switching overhead. It also restricts an individual processor's view into the executing program. Our implementation runs monolithic C-programs which call a generic switching routine that performs a full save and restore of *all* important processor registers. The frequency of the switching affects performance through the context switching overhead, and also affects data confidentiality through the granularity of the fragmentation.

E. TCB Analysis

In our prototype consisting of b banks of n fixed processors, the TCB consists of n *b:1* multiplexers on the AHB input channel, and n *1:b* demultiplexers on the AHB output channel. This switching connects the RAM, via the voting logic, to one bank at a time. For $n = 3$, the voting logic requires 3 comparators on the AHB input channels. Additional control logic resets the processors and updates the select signals for the multiplexers and demultiplexers. Table I compares synthesised resource usage within the Xilinx Spartan 6 FPGA for a minimal Leon3 core (as in Fig. 3, though without DSU and UARTs) against the TCB for $b = 1$ and $b = 10$.

TABLE I
TCB SIZE ANALYSIS[1]

	Slice LUT6	Slice Registers
Single Leon3 core	2516	1221
TCB, 3 PEs (b=1, n=3)	359	5
TCB, 30 PEs (b=10, n=3)	1096	13

[1] Results obtained using Xilinx ISE Release 13.1 - xst O.40d

For $b = 10$, the TCB logic is considerably smaller than a single Leon3 processor. As the number of processors increases, the TCB scales by increasing the size and number of multiplexers and demultiplexers. The small size, low complexity and mainly stateless, repetitive logic of the TCB lends the design to trusted fabrication when compared to the development of a full trusted processor.

VI. EXPERIMENTATION AND RESULTS

For SAFER PATH to be a viable architecture, it must protect against the effects of Hardware Trojans while maintaining an acceptable level of performance. We tested the implementation described in Section V for its ability to protect against data leakage, functional modification and Denial of Service (DoS) attacks.

A. Data Leakage

Hardware cannot leak data that it does not have access to. We use program execution switching to limit an individual processor's access to data. The key expansion phase of the AES-128 encryption algorithm was used to examine the data fragmentation effects of execution switching. This was implemented in C, with execution switches strategically placed to fragment access to the generated key material. During the process of expanding the 128-bit key into the required round keys, execution switches occurred after each 32 bits of the expanded key were generated.

By recording which processors had access to which parts of the generated key, we were able to show that no single processor was exposed to more than 32 bits of the key at any one time, nor was any one processor exposed to more than 32 consecutive bits of the key.

There is a risk that every bank contains a trojaned processor that continuously leaks *all* data that it has access to. Replication and voting ensure that all direct communication channels (e.g. memory, IO) cannot be used for leaking. This leaves the more challenging problem for an adversary of leaking a large amount of real-time processor data through indirect channels.

While our random bank selection is able to minimise key exposure in the manner described above, we could improve upon this by giving the programmer direct responsibility for bank selection. The programmer is then able to carefully analyse their program and data to ensure that over time no individual processor can piece together sensitive data. For example, with encryption keys it is important that the same parts of the key are always used on the same processors.

B. Functional Modification

Our processors are limited to computation, memory access, UART access, and GPIO manipulation. All effects of computation manifest through memory accesses (e.g. stack manipulation). We forced one processor (in a bank of three) to perform a different access to each of the memory, UART, and GPIO units. This simulated the effect of a trojaned processor attempting to modify the functionality of the system. On all occasions program execution continued correctly. Depending on the modification, the errant processor was able to either resynchronise on the next instruction, or the next time it was reset. Conversely, modifying two processors in an identical manner allowed the erroneous access to occur, demonstrating that our architecture makes no assumptions about "correct" behaviour, only majority behaviour.

When all three processors were tested with different memory accesses, the system was reset. Further work could allow software to recognise these situations and retry interrupted sections of code or remove recalcitrant processors from the pool. In general, SAFER PATH operates correctly as long as enough correctly executing processors exist in a bank at each memory access to outvote Trojan effects, further supporting the need for the use of variant processors.

C. Denial of Service

A DoS attack was simulated by forcing a processor to be reset during program execution. With a majority of PEs in a bank still performing correctly, the architecture was able to continue execution. Where a DoS attack may have been triggered by a rare event or a "ticking timebomb" Trojan [7], resetting the processors regularly may allow those processors to be used productively again.

D. Performance

During the key expansion tests, the 47-instruction execution switching routine is called 44 times. The fragmented performance is approximately 2.3 times slower than the unfragmented performance. For certain applications, a performance decrease of this magnitude would be an acceptable compromise in order to protect sensitive data. When operating without data confidentiality concerns, SAFER PATH provides increased assurances of integrity and availability solely through the use of the replication and voting logic. In terms of throughput, there is no performance loss in this configuration.

This improved security is achieved for the cost of extra processors, TCB hardware, and increased power draw.

VII. Further Work

Further work is required on handling IO and external interrupts. External interrupts can be shared and multiplexed by the TCB, with a small modification to the Interrupt Service Routine (ISR) to ensure the TCB can clear the interrupt line. Direct IO can be multiplexed and voted as with the UART example, but utilising memory mapped IO through the multiplexed memory provides the most convenient solution.

Execution switches could be made automatic and more fine-grained by instrumenting a compiler to insert them. Coupled with data flow analysis, this could provide an optimal solution to the provision of confidentiality through data fragmentation.

Further, resetting processors, reconfiguring banks from the pool of processors, and feeding processors dummy code while otherwise idle may provide added protection against Hardware Trojans.

VIII. Conclusion

SAFER PATH is a new processing architecture that is comprised mostly of COTS components and can operate correctly in the presence of active Hardware Trojans. Through a defence in depth approach, we utilise program replication and fragmented execution to provide system integrity and availability, and to maintain data confidentiality.

A prototype implementation with an FPGA has proven SAFER PATH to be viable, demonstrating integration of COTS processors with a minimal TCB to form a complete processing system capable of being used as an embedded processor replacement. The developed TCB is simple, providing a good starting point for accreditation of a single trusted component.

Ideal candidate systems for SAFER PATH are those where the cost of failure is significant, such as cryptographic ICs or on-board missile guidance systems. As a complement to prevention and detection techniques, it provides a last line of defence against Hardware Trojans.

References

[1] J. Rajendran, E. Gavas, J. Jimenez, V. Padman, and R. Karri, "Towards a comprehensive and systematic classification of hardware trojans," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 30 2010-june 2 2010, pp. 1871 –1874.

[2] M. Anderson, C. North, and K. Yiu, "Towards Countering the Rise of the Silicon Trojan," DSTO Information Sciences Laboratory, DSTO Technical Report DSTO-TR-2220, May 2008.

[3] R. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware trojan: Threats and emerging solutions," in *IEEE High Level Design Validation and Test Workshop*, 2009, pp. 166 –171.

[4] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design and Test of Computers*, vol. 27, pp. 10–25, 2010.

[5] M. Abramovici and P. Bradley, "Integrated circuit security: new threats and solutions," in *Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW'09. New York, NY, USA: ACM, 2009, pp. 55:1–55:3.

[6] M. Beaumont, B. Hopkins, and T. Newby, "Hardware Trojans - Prevention, Detection, Countermeasures (A Literature Review)," DSTO Information Sciences Laboratory, DSTO Technical Note DSTO-TN-1012, June 2011.

[7] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in *Proceedings of the 32nd IEEE Symposium on Security and Privacy, May 2011*, May 2011.

[8] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," *IEEE Symposium on Security and Privacy*, pp. 159–172, 2010.

[9] Lockheed Martin Corporation, "Lockheed martin delivers first advanced computer modules for F-35 JSF program," July 2003, http://www.lockheedmartin.com/news/press_releases/2003/Lockheed MartinDeliversFirstAdvanced.html.

[10] G. Bloom, B. Narahari, R. Simha, and J. Zambreno, "Providing secure execution environments with a last line of defense against trojan circuit attacks," *Computers & Security*, vol. 28, no. 7, pp. 660 – 669, 2009.

[11] A. Das, G. Memik, J. Zambreno, and A. Choudhary, "Detecting/preventing information leakage on the memory bus due to malicious hardware," in *Proceedings of Design, Automation and Test in Europe*, ser. DATE'10, 2010, pp. 861–866.

[12] M. Beaumont, C. North, B. Hopkins, and K. Yiu, "Hard disk guard based policy enforcement," in *Proceedings of POLICY'11*, June 2011.

[13] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating UCI: Building stealthy and malicious hardware," in *Proceedings of the 32nd IEEE Symposium on Security and Privacy, May 2011*, May 2011.

[14] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 66–75, 2010.

[15] D. R. McIntyre, F. G. Wolff, C. A. Papachristou, and S. Bhunia, "Dynamic evaluation of hardware trust." in *IEEE International Symposium on Hardware-Oriented Security and Trust*, 2009, pp. 108–111.

[16] B. Newgard and C. Hoffman, "Using multiple processors in a single reconfigurable fabric for high-assurance applications," in *HOST*, J. Plusquellic and K. Mai, Eds. IEEE Computer Society, 2010, pp. 25–29.

[17] G. Trouessin, Y. Deswarte, J.-C. Fabre, and B. Randell, "Improvement of data processing security by means of fault tolerance," in *Proceedings of the 14th National Computer Security Conference*, 1991, pp. 295–304.

[18] L. Lin, W. Burleson, and C. Paar, "MOLES: malicious off-chip leakage enabled by side-channels," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, ser. ICCAD '09. New York, NY, USA: ACM, 2009, pp. 117–122.

[19] Aeroflex Gaisler AB, "Leon3 multiprocessing cpu core product sheet," Feb 2010, http://www.gaisler.com/doc/leon3_product_sheet.pdf.

[20] ARM Limited, "AMBA specification (Rev 2.0)," 1999, accessed at http://www.arm.com/products/system-ip/amba/amba-open-specifications.php, 1 August 2011.