TMR: A Solution for Hardware Security Designs

Univ. of Virginia Dept. of Computer Science Tech Report CS-2015-02

Yubo Li
Department of Computer Science
University of Virginia
Chlarlottesville, Virginia
Email: liyubobuaa@gmail.com

Kevin Skadron
Department of Computer Science
University of Virginia
Chlarlottesville, Virginia
Email: skadron@virginia.edu

***Abstract*—Cyber-security has become an increasingly important issue in today's integrated circuit designs. The use of commercial off-the-shelf products and design outsourcing have introduced uncertainty in the supply chain and posed security threats such as an inside attacks. This paper describes the implementation of an FPGA design based on soft-core processor, and how triple modular redundancy (TMR) is applied to mitigate against the attacks in the supply chain. Four different applications are implemented and used as test cases for the TMR design. This paper also discusses the overhead of applying TMR, and the importance of adding diversity in redundancy.**

I. Introduction

Today, many mission-critical and safety-critical systems depend on integrated circuits (ICs). The supply chain for integrated circuits (ICs) is difficult to secure, involving multiple firms (processor design, fabrication, packaging, etc.), each with large teams of designers and technicians. A hardware Trojan can be very small and easy for a single individual to insert, especially at the design stage, where it might be as simple as one or two lines of innocuous-looking programming, or a few transistors and wires in a physical chip layout. Consequently, effective security techniques are required to prevent adversaries from interfering with the correct operations of the circuit.

Spatial redundancy in hardware provides the ability of behavioral checking and comparison. The three components vote on all actions, ensuring that a single corrupt unit will be outvoted. Redundancy also provides fault tolerance at no extra cost, and fault tolerance is becoming increasingly important as transistor miniaturization continues and transistors become more vulnerable to electrical upsets.

In the field of cyber security, designers try to increase as much as possible the difficulty of an adversary's deploying an attack while keeping the designing effort as little as possible. Heterogeneous TMR employs independently developed hardware (or IP blocks) from different sources, and greatly increases the difficulty of attacks. It forces an adversary to compromise at least two different pieces of hardware in order to deploy a meaningful attack, which is extremely challenging.

This paper discusses hardware design considerations and implementation of a hardware security design. We show the concepts of TMR using soft-core processors and then develop sample hardware Trojans and triggering mechanisms to demonstrate the effectiveness of TMR. Four different applications are developed and run on the TMR design: matrix multiplication, sum, protocol conversion, and SD card access. This paper also discusses the performance and area overhead of TMR designs, and the importance of adding heterogeneity to redundancy.

The rest of the paper is organized as follows: Section II describes the implementation details of a hardware security design and four benchmark programs. Section III discusses how TMR is applied to the design and the importance of heterogeneous redundancy. Then Section IV shows TMR's effectiveness in mitigating against hardware attacks. Section V discusses the area and performance overhead of TMR. Lastly, Section VI concludes the paper and proposes future work.

II. Soft-core Processor Implementation and Benchmark Applications

The re-configurability and flexibility of field programmable gate arrays (FPGAs) make them favorable for prototyping. In this paper, the soft-core processors and all hardware-based protections are implemented on FPGAs. In addition, FPGAs may also be suitable for the purpose of deployment, because of their short design-to-product time.

A. Implementation

The hardware security design is implemented in a custom board, SiCore SHIELD II board. Below is a summary of the board [1]: Xilinx Kintex-7 XC7K325T-1FFG676 FPGA, low-jitter 200 MHz oscillator, four 10M/100M/1G Ethernet PHYs with RGMII, 1Gb BPI Flash, one SD card slot, eight UART transceivers, four on-board LEDs and four on-board general-purpose buttons, and 512MB DDR3 memory (800 MHz).

Figure 1 shows a picture of the SiCore SHIELD II board.

We choose to build the hardware security design based on LEON3, an open-source soft-core processor [2]. LEON3 uses SPARC V8 instruction set and AMBA-2.0 AHB bus interface. It is released as synthesizable VHDL files, and is configurable through the use of VHDL generics. Figure 2 shows a simplified block diagram of the LEON3 architecture.

Due to the lightweight use of memory in this design, we use on-chip block RAMs (BRAMs) to implement the RAM and ROM. The BRAMs are memory resources available in the FPGA chip, and enable faster access time. The ROM is a module written in VHDL which contains hard-coded instructions, which are automatically loaded to the CPU when the FPGA

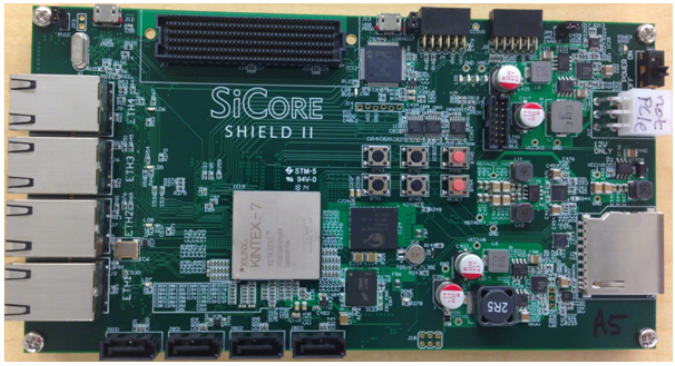

Fig. 1. SiCore SHIELD II FPGA Board

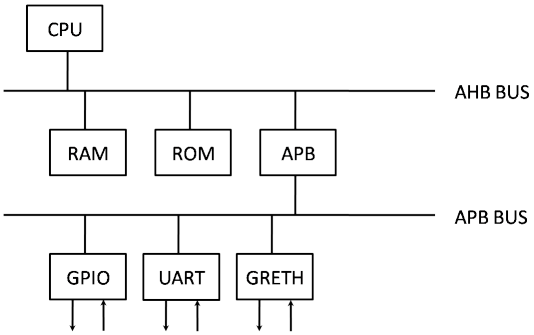

Fig. 2. A Block Diagram of the LEON3 Design

configuration is complete. The ROM can be generated from C program using a PROM generator. AHB bus is used for high speed operations, such as writing and reading data between CPU and memory. APB bus is used for low speed operations, such as communications to the peripheral components. The peripheral components include general purpose I/O (GPIO), UART, Ethernet controller (GRETH), and etc.

B. Benchmark Applications

Four different benchmark applications are developed to be the test cases of the TMR design.

The first benchmark is called MxM, which calculates matrix multiplication. Both matrices are composed of 100×100 random integers.

The second benchmark is Sum, which calculates the sum of intergers from 1 to 50,000.

The third and fourth benchmarks are from a mission-critical application, which is applied on an unmanned aerial vehicle (UAV). The target UAV has an autopilot system and a monitoring system, called Piccolo and Sentinel, respectively. The Piccolo controls the flight of the UAV; The Sentinel is an independently sourced and verified hardware unit to monitor behavior of various components in the UAV, including GPS and flight control. Figure 3 shows a block diagram of the UAV system.

The Piccolo communicates using RS-232 protocol, while the Sentinel communicates using TCP/IP protocol. Therefore a bi-directional protocol converter is required between the two subsystems. In addition, the Sentinel needs to writes/reads data

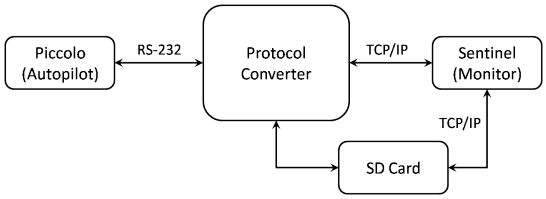

Fig. 3. A Block Diagram of the UAV system

to/from an SD card. The third benchmark in this paper is the protocol converter between RS-232 and TCP/IP, and the fourth is SD card access.

The protocol conversion and SD card access are implemented as a bare-metal applications, meaning that there is no operating system running on the processor. Less complexity in the implementation means less design effort and more importantly, fewer loopholes that might be taken advantage of by adversaries.

We use uIP [3], an open-source TCP stack, to establish TCP/IP connections and process TCP/IP packets. Figure 4 shows the implementation of the TCP stack.

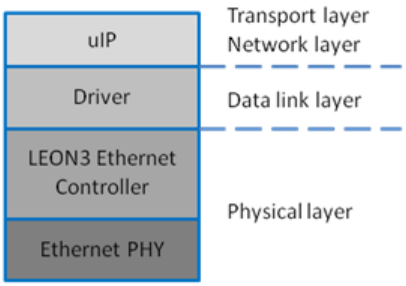

Fig. 4. Implementation of TCP Stack

On the physical layer, LEON3 Ethernet controller (GRETH) drives the pins of the Ethernet PHY. On the data link layer, a driver writes data to the transmitter of the Ethernet controller and reads data from the receiver. On the network layer and transport layer, uIP establishes and manages TCP/IP connections, e.g., sending and receiving packets, updating acknowledgement number and sequence number, etc.

The received TCP/IP packets are first analyzed, and then the data bytes are sent to the buffer of the UART transmitter. When the DATA_READY flag of the transmitter's buffer is high, the controller reads from the buffer and sends the data in RS-232 format. When there is incoming serial data, the DATA_READY flag of the receiver's buffer is set high, and the data will be processed when the CPU is idle.

FatFS, an open-source FAT file system [4], is used to implement the SD card access functionality. FatFS uses GPIO signals to drive the pins of the SD card slot in serial peripheral interface (SPI) mode. Write and read commands can be sent to the SiCore board through the user TCP/IP stream.

III. Applying TMR

Compared to software-based protections, which usually have a "big picture" of what the system is doing on the

function level, hardware-based protections focus on behavior on the instruction and word level, and require very high level of hardware sophistication and coordination to defeat, hence increase the difficulty to attack. Our TMR solution operates at the hardware layer and directly protect the most critical system functions. These solutions are embedded within the protected hardware and don't require modification to the software applications.

A. Homogeneous TMR

We first triplicate the soft-core processor design described in Section II with identical copies. The entire LEON3 implementation, including CPU, buses, RAM, ROM, and all peripheral components, is triplicated in the FPGA. The three implementations share the same clock, reset, and input signals. Additional comparators and MUXs are created to act as majority voters. The output signals of GPIO, UART, and GRETH are sent to their individual voters.

Figure 5 shows the voting mechanism for serial output. We call the three copies of LEON3 design copy 1, copy 2, and copy 3, respectively. The UART component in each copy is a slave of the APB bus. A comparing unit is implemented to detect disagreements among the data written to the UART components. If no disagreement detected, or if copy 2 or copy 3 disagrees with the other two, then the output of UART in copy 1 is used as the UART output of the TMR design. If copy 1 disagrees with the other two, then UART output of copy 2 is selected. The voting mechanism of GRETH output signals follows the same manner.

The most important function that TMR performs is to compare the output signals of the three implementations. In order to compare the UART output data, three FIFOs are used to store the data sent to the UART transmitters, one for each implementation. Every time data is sent from the bus to the UART transmitter, a copy of the data is also written to the corresponding FIFO. When all FIFOs are written, the three copies of data are read and compared. Similar approaches are applied to compare the output signals of GPIO and ETH components.

In a TMR design, synchronization between the three implementations is critical to ensure correct operation. A small difference in the progress of program execution can result in different outputs, which will lead to false positives reported by the TMR design. Since the three LEON3 implementations have identical configuration and share the input signals, ideally they should have exactly the same behavior in a cycle-by-cycle fashion.

In practice, however, we observed out-of-sync behavior among the three implementations. We believe this is due to asynchronous I/O behavior in terms of how cores check for TCP input. In order to make the TMR design work correctly, we added several synchronization points in the program to enforce synchronization. When a synchronization point is reached in program execution, a GPIO syn_cout signal is set to 1 and this signal is sent to the top level of the TMR design. When all three syn_out signals are 1, a controller sets a GPIO synchronize signal to 1 and this signal is sent back to each LEON3 implementation. After seeing the asserted synchronization signal, each LEON3 implementation sets their own sync_out signal to 0 and proceeds in program execution. This synchronization method has been proved to be reliable by running tests for tens of hours without errors.

B. Heterogeneous TMR

Homogeneous redundancy is insufficient in mitigating against hardware-based attacks, because identical hardware copies will be vulnerable to the same hardware Trojans. In contrast, heterogeneous redundancy can improve security by providing diversity in hardware.

The LEON3 processor is configurable; therefore we have implemented the benchmark designs with different cache configurations as a first step to mimic the heterogeneity. This will affect synchronous operation due to different hit/miss behavior. In our test, the three LEON3 CPUs have different cache configurations: copy 1 is fully associative, copy 2 is set-associative, and copy 3 is directly mapped. The different configurations are applied to both the instruction and the data cache. The results showed that the TMR design can work correctly with different cache configurations.

Ideally, heterogenous TMR should be composed of processors from different vendors or design teams, which may apply different instruction set architectures (ISAs). Processors with different ISAs may execute the same instruction flow in different orders, and therefore could be challenging to synchronize.

IV. TMR's Resistance against Cyber-attacks

In order to test benchmark 3 and 4, we inject a hardware Trojan into the Ethernet controller (GRETH) to mimic an attack in the supply chain. The hardware Trojan is idle when the circuit is powered on, and becomes active when a triggering pattern, "attack", in the incoming TCP packets is recognized. When triggered, the hardware Trojan will replace all received TCP data with garbage, lauching a denial-of-service attack to the protocol converter and SD card.

The implementation of the hardware Trojan takes about 50 extra lines in VHDL. Figure 6 illustrates the insertion of hardware Trojan.

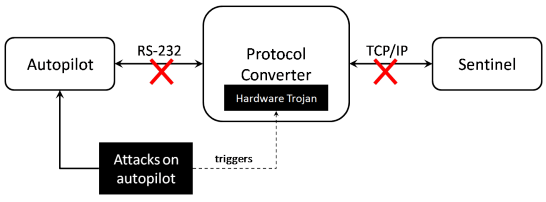

Fig. 6. Embedded Hardware Trojan

The non-TMR design has no defense mechanism against hardware Trojan and therefore is vulnarable to the attack. When the hardware Trojan is activated, the protocol converter and SD card stops responding to further conversion/access requests.

In the TMR design, the hardware Trojan is injected into copy 2. When the hardware Trojan is idle, the TMR design gives no false alarms. When the Trojan is triggered, the TMR

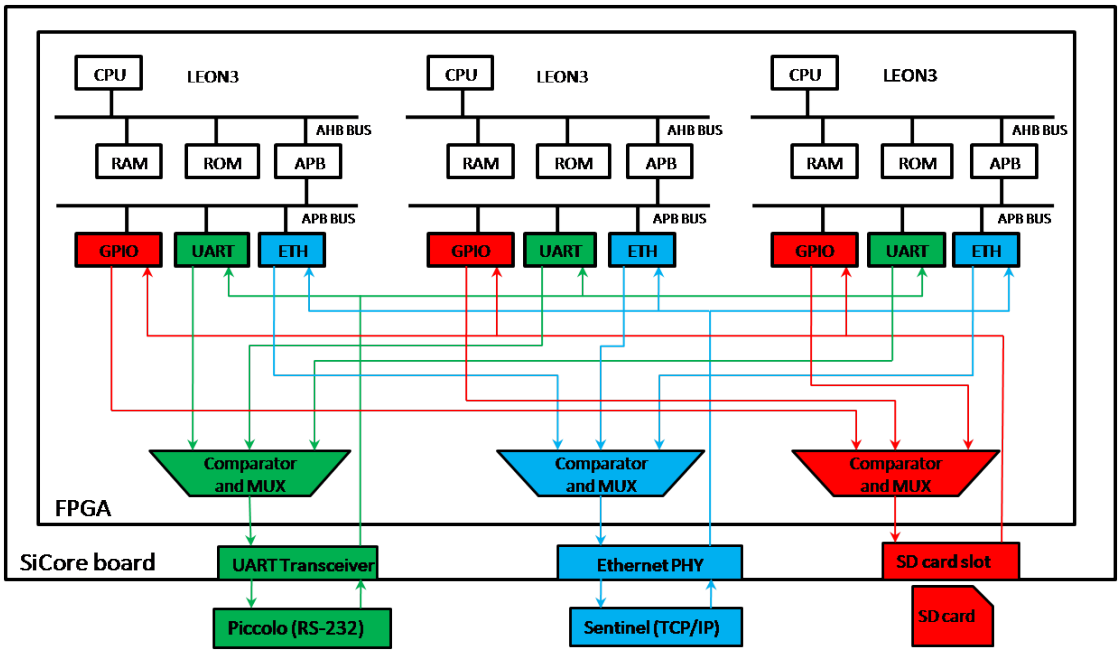

Fig. 5. A Block Diagram of the FPGA Design with TMR

design is able to continue the normal conversion/access and correctly raise an alarm flag for copy 2.

Benchmark 1 and 2 do not use input data, therefore we insert constant corrupted bits in copy 2 to mimic a compromised processor. Test results showed that the TMR worked correctly in masking the wrong output. In addition, the constant corrupted bits can also be seen as a simulation for radiation-induced faults caused by high-energy particles.

V. Performance and Area Overhead of TMR

Applying TMR will result in performance and area overhead to the design. In order to evaluate the overhead, we compare the non-TMR and TMR version of the four benchmark programs. Table I presents the resource utilization of the benchmark applications, in terms of the percentage of occupied slices in XC7K325T-1FFG676 FPGA.

Note that the overhead of TMR designs are less than 3X. This is because multiple look-up tables may be placed to the same slice during place and route.

TABLE I. Comparison of Resource Utilization

	Non-TMR	TMR	Overhead
MxM	6.04%	16.64%	2.72X
Sum	6.04%	16.44%	2.72X
Protocol Conversion	8.17%	23.36%	2.86X
SD Card Access	8.17%	23.36%	2.86X

Table II presents the performance comparison of the benchmark applications, in terms of maximum clock frequency.

TABLE II. Comparison of Maximum Clock Frequency

	Non-TMR	TMR	Overhead
Matrix Multiplication	110 MHz	100 MHz	1.1X
Sum	110 Mhz	100 MHz	1.1X
Protocol Conversion	110 MHz	100 MHz	1.1X
SD Card Access	110 MHz	100 MHz	1.1X

VI. Conclusions and Future Work

Security against cyber attacks has become an increasingly important issue in circuit designs, especially in mission-critical and safety-critical applications. Insider attacks in the supply chain, such as hardware Trojans, pose great threats to these applications.

This paper proposes to use TMR as a solution to improve hardware security. A soft-core processor design and four different benchmark applications are implemented in an FPGA, and TMR is applied to the design. Configuration changes have been made to the triplicated processors to mimic heterogeneity, and test results showed that TMR works correctly in both masking hardware Trojan and not issuing false alarms under normal operations.

As the next step, we propose to implement the TMR design with processors from different vendors or design teams. Heterogeneous processors may employ different ISAs, which makes it more difficult to tamper with the TMR design, but also rises new challenges in performing voting.

References

[1] Digilent, Inc. NetFPGA-1G-CML Board Reference Manual. 2013.

[2] http://www.gaisler.com/index.php/products/processors/leon3

[3] https://github.com/adamdunkels/uip

[4] FatFS - Generic FAT File System Module. http://elm-chan.org/fsw/ff/00index_e.html.