August 2021

FPGA Hardware Trojan Power Analysis

By

BRAJAN ILIEVSKI
KHANH HOA HUYNH

Department of Electrical and Computer Engineering
University of Windsor

Abstract

FPGA hardware trojans pose a security threat due to the rising adoption of FPGAs across many key industries. The lack of transparency into FPGA chipset design coupled with the many open nodes for trojan attack establishes significant risk of trojan infection and detection evasion. Trojans are triggered by rare events, and it is not possible to test all exhaustive input patterns. The current industry problem is the lack of mature strategies for detecting and protecting against trojan attacks in FPGAs, while the development of novel detection and evasion strategies for hardware trojans is a continual arms race. (Brajan)

Group 23 is responsible for demonstrating that available means of power analysis is unable to detect the changes in power consumption between a non-infected and infected ISCAS-85 benchmark circuit implemented on an FPGA. Our design is constrained to infecting a 16x16-bit multiplier (C6288 ISCAS-85 benchmark) with an existing half-adder trojan and measuring AC power consumption during active multiplication. The infected benchmark design was deployed on a Spartan-6 LX9 FPGA target board. Active power trace capture was achieved using a ChipWhisperer CW1200 scope and CW308 capture board. A custom Verilog model of the infected benchmark was developed to bypass the existing UART communication protocol used by ChipWhisperer platform, thereby enabling the infected circuit to successfully trigger a power capture. Group 23 determined that the power consumption of the infected circuit was negligibly impacted, and no significant differences could be measured. This work suggests that infected FPGAs can avoid existing power analysis detection strategies. (Brajan)

Table of Contents

2.0 Introduction

The present challenge facing the industry is defining what gate or transistor-level circuitry can be placed into FPGAs and where it can be inserted, which could lead to a trojan attack. Malicious modifications to FPGA hardware trojans can lead to unwanted operating points (trojan states) in safety-critical systems, the exposure of industry secrets/information, or the leakage of secret keys in cryptographic systems, constituting a serious threat to stakeholders in these industries. A never-ending arms race has ensued in the development of novel hardware trojan detection and evasion/masking strategies; nonetheless, solutions for detecting and protecting against trojan attacks in FPGAs are still in their infancy. Nowadays, modern industry relies on collaboration among business partners during manufacturing processes to minimize cost and enhance product performance. However, non-transparent manufacturing process can impose risks of hardware trojan insertion. Hardware trojan attacks inflict potential threats towards clients and stakeholders in the industries through information leak, denial of services, and undesired trojan-state results. Confronted with the increase of hardware trojans, many research regarding trojan detection and evasion technique has been published and implemented. However, common or permanent solutions for detecting any malicious hardware attacks are still not yet found.

This group seeks to investigate the malicious modification towards FPGA hardware trojans that generate unwanted results (trojan-state). The design goal of the capstone project will be to illustrate the severe effect of hardware trojan on FPGA by comparing side-channel information (power consumption, voltage, and timing diagram) between a clean and infected benchmark circuit. The side-channel results are anticipated to have similar output since trojan model is very minimal compared to the benchmark circuit. Particularly, in the trojan design, our group inserted additional 5 logic gates to implement a trojan-state within 16x16 bit multiplier circuit.

The scope of the capstone project is focused on demonstrating the subtle nature of hardware trojans on FPGA by comparing the insignificant difference between clean and infected circuit. Therefore, we used the ISCAS-85 benchmark circuits, particularly the C6288 16x16 Multiplier rather than creating a new hardware description language (HDL) behavioral script for a test circuit. The capstone group concentrated more on side-channel information capturing on both hardware and software approach, particularly Chipwhisperer 1200 and ModelSIM HDL Simulator. Moreover, the compatible half-adder hardware trojan HDL was researched and implemented in the ISCAS-85 benchmark circuits.

3.0 Benchmarking

The current state-of-the-art for hardware trojan detection in integrated circuits involves destructive and non-destructive methods. As the name implies, destructive methods mandate de-packaging the integrated circuit and comparing the circuit's functionality to a 'golden circuit' benchmark via an in-depth analysis using Scanning Electron Microscope (SEM) image re-construction [1]. Given the exhaustive nature of such testing, this approach to verification is not scalable and it is also possible that not all selected circuits of the population are infected with a Trojan.

Non-destructive methods can be classified as 1) invasive strategies where the original design is extended with embedded features to prevent or help detect a trojan, or 2) non-invasive strategies where the original design is left unaltered [2]. Invasive techniques aim to modify the original design and make it difficult for the bad actor to insert a trojan by obfuscating the circuit's true functionality and hiding the true rare events that may be exploited to trigger a trojan. This may completely invalidate the trojan or make it more easily detectable in the case that it is successfully inserted into the circuit [2].

Non-invasive strategies typically involve comparing the behaviour of the test circuit with that of the 'golden' benchmark circuit [3]. In this strategy, the run-time techniques leverage a parallel monitoring system that actively attempts to detect suspicious functionality during the field operation of the circuit. When a detection is triggered, the monitoring system then attempts to initiate countermeasures to invalidate the trojan. However, the efficacy of these methods presently requires additional testing and there is also significant resource overhead associated with running a monitoring system in parallel with the circuit [3].

Test-time techniques involve either logic-based testing or those involving side-channel measurements such as power, voltage, electromagnetics, etc. [4]. Although test-time techniques eliminate the overhead computational cost associated with run-time techniques, such strategies often require a golden circuit as a baseline comparison. Side-channel measurements are especially important in detecting the effect of smaller hardware Trojans given that these strategies rely on the physical parameters of the circuit rather than any sort of observable malfunctions in the circuit. One disadvantage of side-channel techniques is that process variations in circuit fabrication and measurement noise may obfuscate the effects of subtle trojans [4].

4.0 Design Criteria, Constraints, and Deliverables (Khanh)

The purpose of the hardware trojan design is to sabotage the ISCAS-85 benchmark circuits functionality while capturing the side-channel information of the operating circuits. Particularly, the targeted model was ISCAS-85 C6288 (16 by 16 multiplier); the implemented hardware trojan flipped the 16th bit of the multiplication process without causing a noticeable change in captured power traces. To capture the power traces, the team utilized Chipwhisperer 1200 scope, which measures real-time power consumption from the FPGA. From the scope, the capstone team can immediately retrieve side channel information such as current and power consumption for further analysis.

Throughout the design process, there were many constraints that limit the overall design of the final model. As mentioned in the scope, the capstone team will be using ISCAS-85 circuit models, which are all written in HDL. Therefore, the target board is limited to FPGA rather than microprocessor due to FPGA offers more flexibility in term of implementing structural HDL behaviour circuits. The team select Spartan 6XL9 FPGA target board since it is supported by the Chipwhisperer capture board and highly endorsed in the Chipwhisperer community. Moreover, the circuits of ISCAS benchmark are implemented in structural form to avoid any optimization process from the compiler, which could lead to different gate connections. With structural models, the team can easily insert the additional logic gates to sabotage the circuit functionality; however, the structural model is more challenging to interpret than behaviour model. For the communication prototype, our group decided to choose JTAG-HS2 programmable cable as a tool for uploading the HDL circuits to the target board. This communication is chosen due to its high flexibility and time efficiency.

The capstone project delivered a method to infect structural HDL circuits (16 by 16 multiplier circuit) and developed the side-channel information capturing scheme for ISCAS-85 benchmark. The deliverables of the capstone project consisted of three important components.

The first component is the half-adder trojan. The half-adder trojan structurally rewired to connect to the exist benchmark circuit (hard-wired), causing the single bit flipping during multiplication process. The second component is the extended Verilog code of the infected benchmark featuring a ROM for inputting multiplier and multiplicand for benchmark circuit. And the last component is a python script that receives and displays the captured output from the scope.

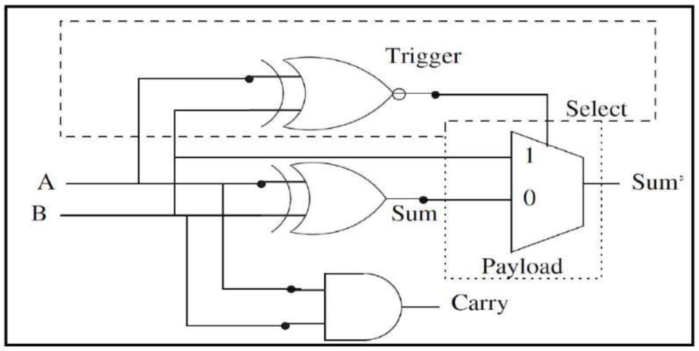
5.0 Design Methodology

Design Process

We set out to develop an infected ISCAS-85 benchmark circuit that could be successfully deployed on a FPGA platform and analyzed with the available ChipWhisperer power analysis tool chain. Work began with a focus on determining the required target board that could meet three requirements: 1) compatibility with the CW308 capture board, 2) accommodate an ISCAS-85 benchmark circuit, and 3) provide VHDL/Verilog source code for communication with the ChipWhisperer toolchain. Group 23 determined that the Spartan-6 FPGA platform could meet these requirements.

Next, Group 23 determined that the ChipWhisperer did not natively support programming the Xilinx-based FPGA over the required JTAG communication interface. This mandated that we deploy Xilinx's own ISE Impact suite to generate the necessary bitstream and program the FPGA. However, we continuously encountered challenges with USB communication errors when attempting to capture test traces with the ChipWhisperer platform. Analysis uncovered that the ChipWhisperer and Xilinx ISE Impact software suites required two different versions of VirtualBox to be successfully deployed without communication errors. With no way to deploy two simultaneous versions of VirtualBox on one physical machine, we experimented with installing a native Windows version of the ChipWhisperer suite while simultaneously running a VirtualBox version of Xilinx ISE Impact.

Having resolved the USB communication problems, we next set out to choose an appropriate ISCAS-85 benchmark and that would serve as our clean and infected circuit. Here, we chose to implement a C6288 16-bit multiplier due to its relatively simple structure and layout, comprised of an array of half adders and full adders (Figure 2). The C6288 was combined with a subtle half adder trojan placed on the last half adder circuit of the C6288 multiplier array that was responsible for outputting the 16th bit of the 32-bit multiplier output. The team could not rely on a simple behaviour Verilog model of the multiplier (P = A*B) given that we needed to specifically isolate the gates of the 16th bit half adder; as such, the team was required to leverage a structural model with gates and wires explicitly defined. This required us to iteratively work through the hundreds of NOR gates comprising the C6288 benchmark code to isolate the primitive gates of the 16th bit half adder and subsequently infected with the selected trojan.



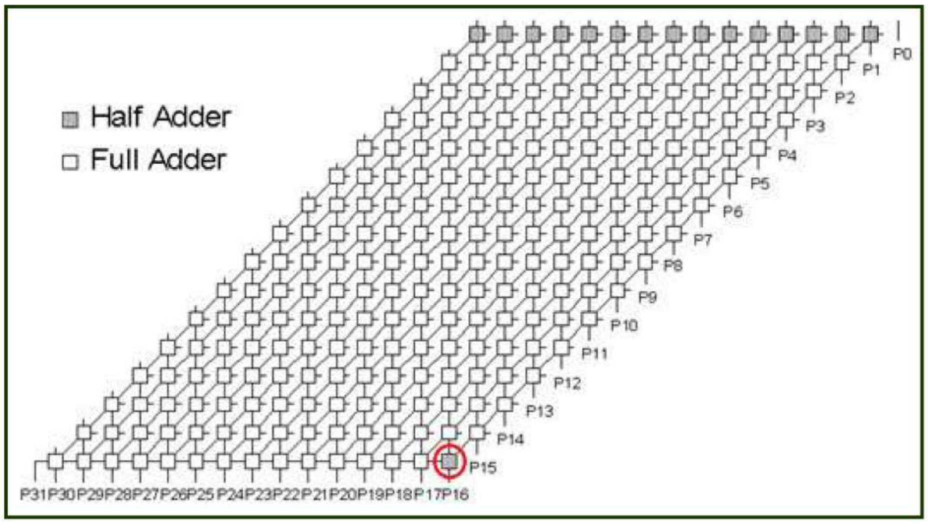

Half Adder
Full Adder
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15
P31 P30 P29 P28 P27 P26 P25 P24 P23 P22 P21 P20 P19 P18 P17 P16

Figure 2: The half-adder trojan logic gate schematic and point of insertion (red circle) [6, 7].

Having developed the Verilog model of the infected C6288 multiplier, our efforts then focused on the hardware setup and power trace capture. Knowing that the STM32 microcontroller target board that we had available with the ChipWhisperer kit could not accommodate a HDL model, we researched into the available FPGA target boards and concluded that a Xilinx Spartan-6 board could accommodate our design while minimizing budget. However, the ChipWhisperer toolchain provided no means to program the Spartan-6 with the existing tools, and the team had to purchase a Digilient HS2 cable and rely on the external Xilinx ISE Impact software to program the FPGA with the bitstream file.

Challenges arose with the lack of support, resources, and documentation for capturing power traces of custom circuit designs on the Spartan-6 target board. Unlike the STM32 target board which had a huge assortment of example projects, tutorials, and courses, the Spartan-6 only had one example project with capturing power traces on an AES core. The example project included Verilog source code for an AES core and SimpleSerial communication protocol (based on UART) that was used by ChipWhisperer to send key-text pairs for encryption. We initially ideated to modify the available SimpleSerial protocol to send the 16-bit multiplicand and multiplier values to the C6288; however, the team assessed that the SimpleSerial codebase was tightly coupled to the AES core and would require extensive modification to enable it to communicate with our custom C6288 circuit instead.

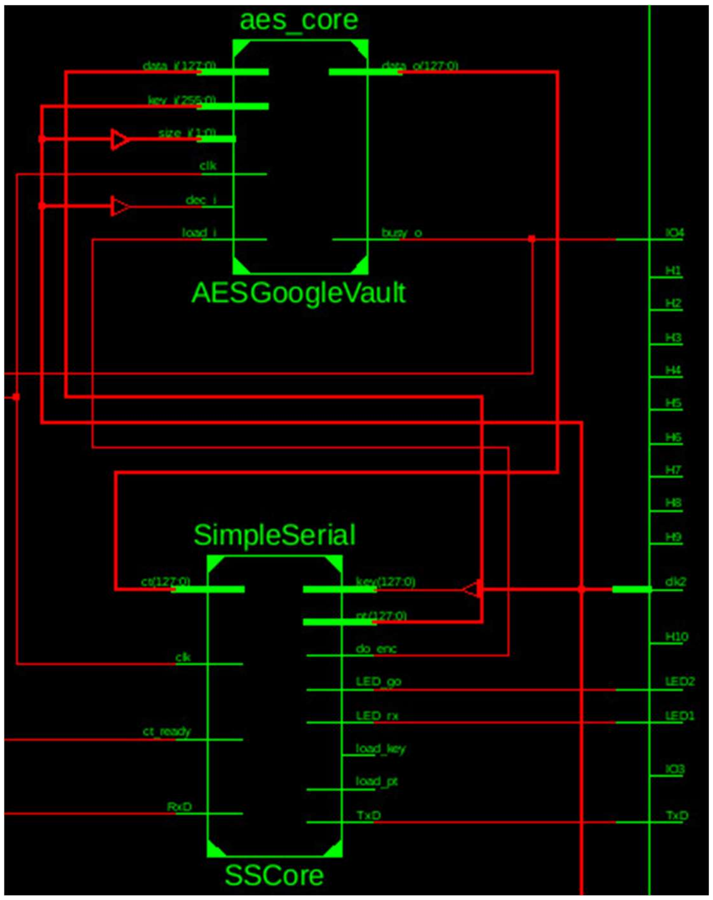


Figure 3: Tightly connection between AES core and Simple Serial in encryption circuit.

Realizing it would not be possible to modify the existing SimpleSerial Verilog code to send inputs to the C6288 multiplier, we abandoned the idea of sending external inputs to the multiplier with the CHipWhisperer toolchain. Instead, we developed a concept to extend the C6288 Verilog codebase with an 8x16 ROM to feed inputs into the multiplier. The ROM would be driven by the ChipWhisperer CW308's clock signal to increment an address variable to loop through the hardcoded multiplier/multipaned values.

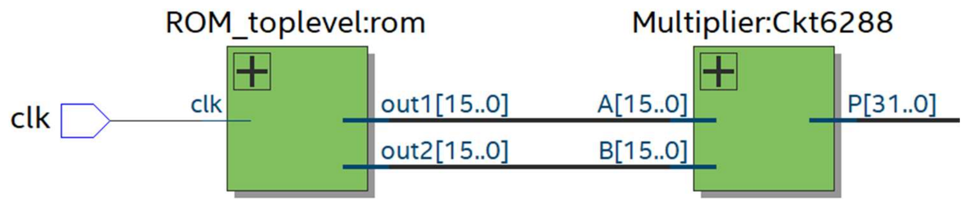


Figure 4: RTL connection between ROM and Multiplier circuit (16 by 16)

This allowed us to bypass the SimpleSerial protocol to send inputs to the multiplier; however, additional testing with this method revealed that the team could no longer control the exact timing of the power trace capture since this was previously controlled by the SimpleSerial protocol during certain trigger conditions (i.e., when an AES core would initiate encryption). Starting a power trace capture command had to be done manually, and as a result, every manual capture would result in graphs generated at random points of active multiplication and ROM values. To overcome this, the team decided to implement a high impedance output in 4 of the 8 values within the ROM. This would generate a power trace graph with an area in inactivity (high impedance sent into multiplier) and an area of power draw during active multiplication (binary inputs sent into multiplier). By leveraging these repeating patterns of multiplier activity/inactivity, the team was able to capture power traces at any point in time and align the power usage graphs at the start of the first value sent by the ROM. The figure below demonstrates what the unaligned power graph looks like versus one with a high impedance inactive state that allows us to align the power draw at the beginning of the multiplication process.

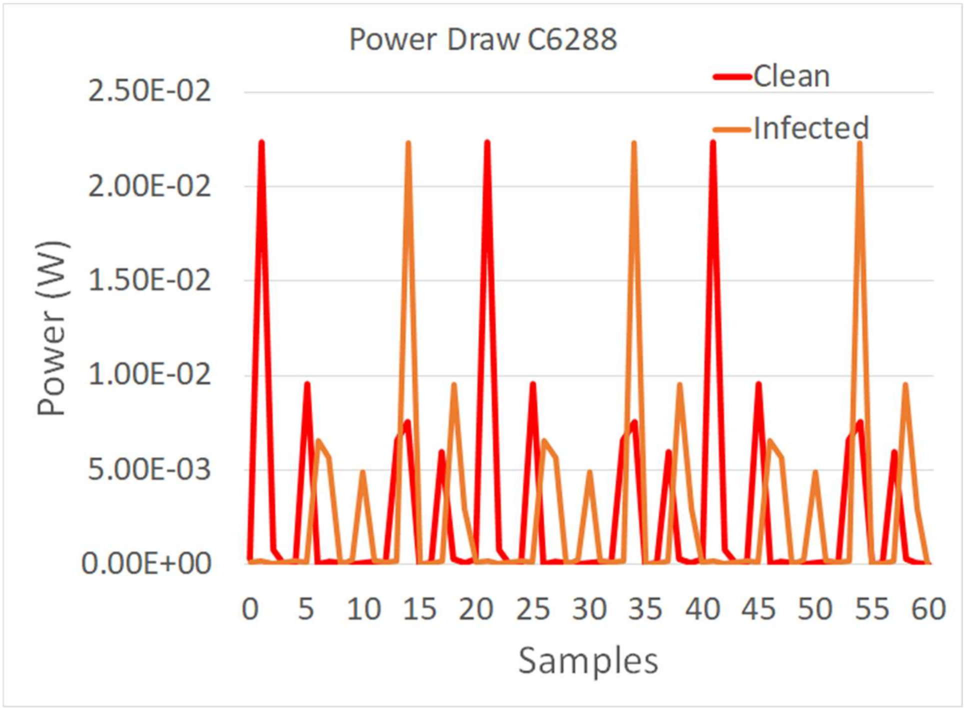


Figure 5: The power draw between clean and infected multiplier (without high-z period)

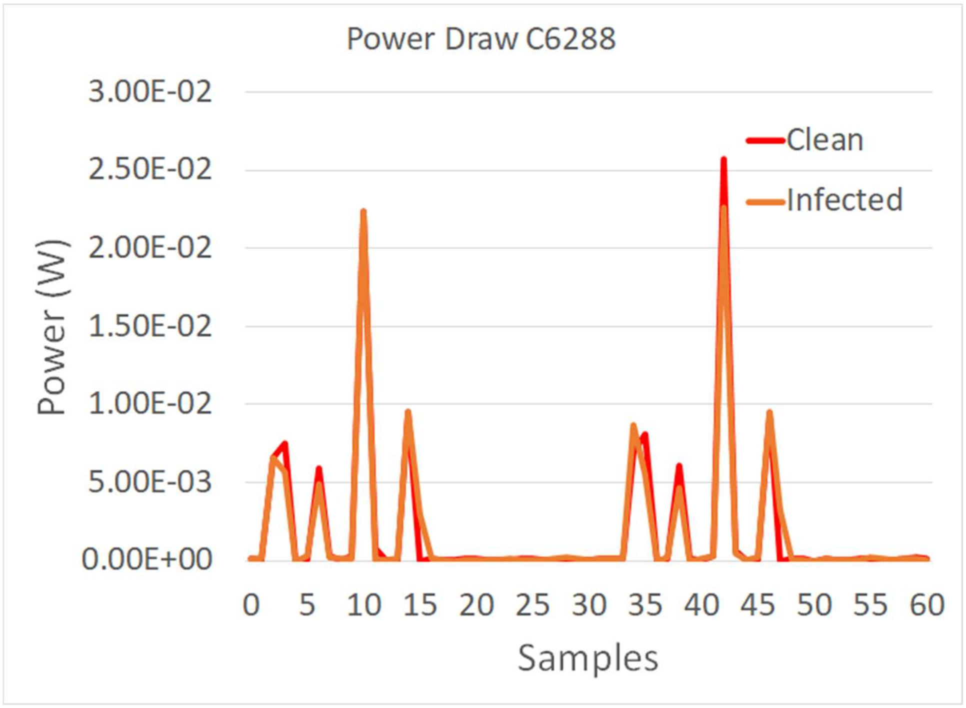


Figure 6: The power draw between clean and infected multiplier (with high-z period)

6.0 Physical Implementation/Simulation Model Development

Component

The major components of the toolchain include:

2.1 CW1200 scope

The ChipWhisperer Pro (CW1200) is a more advanced version of the ChipWhisperer Lite. The Pro has a significantly larger FPGA than the Lite, allowing for the addition of many more features such as a larger sample buffer, streaming mode captures, extra trigger mechanisms, and a touchscreen interface. It is a high-end tool ideal for laboratory application because of these qualities [8]. The ChipWhisperer Pro has many connectors ports which include using glitch port, measure port, and 20-pin connector [8]. In this project, the measure port was used to measure side-channel parameters such as the current in this case.

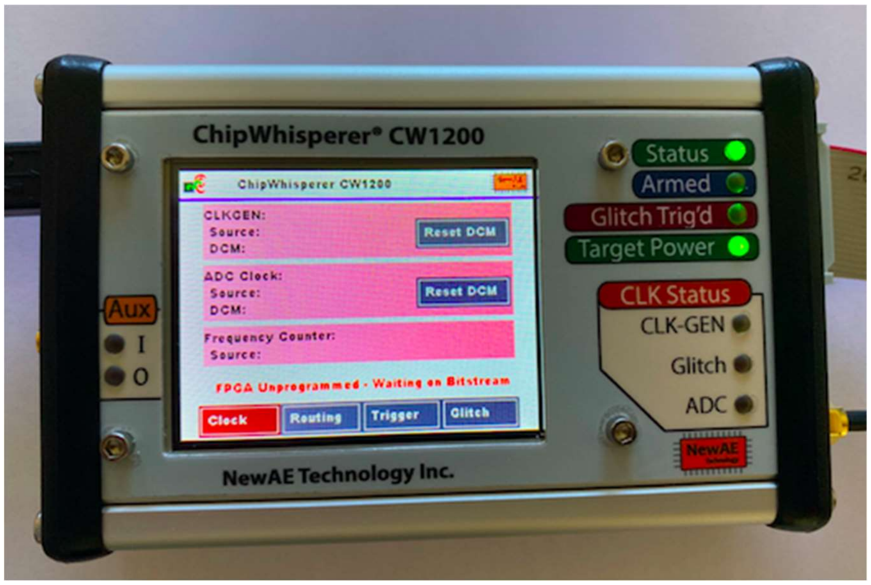


Figure 7: ChipWhisperer Pro (CW1200)

2.2 CW308 UFO Board

The CW308 UFO Board is a useful mainboard for targeting embedded targets of many forms. It was first released in 2016 and will serve as the foundation for most of the future target boards. The CW308 combines all the basic requirements (such as power supply, oscillators, and filters) into a single board, allowing to create easy target victim boards. It can be used either alone (with an oscilloscope) or in conjunction with ChipWhisperer-Capture devices. Scope boards are used to mount side channel assaults, and target boards are utilized as a device under test (DuT) in ChipWhisperer. The CW308 UFO board can be used with a wide range of targets such as CW308T-GENERIC (Protoboard), CW308T-AVR (Atmel AVR, 8-bit), CW308T-XMEGA (Atmel XMEGA, 8-bit), CW308T-87C51 (Intel 87C51RB, 8-bit), CW308T-ATSAMR21 (Atmel SAMR21 with 802.15.4 radio, 32-bit ARM Cortex M0), CW308T-MEGARF (Atmel MegaRF2564RFR2, 8-bit AVR), CW308T-MSP430FR5 (TI MSP430FR5xxx, 16-bit), CW308T-S6LX9 (Xilinx Spartan 6 LX9 in TQFP, FPGA) [9]. In this project, CW308T-S6LX9 (Xilinx Spartan 6 LX9 in TQFP, FPGA) is used which will be discussed in the next section.

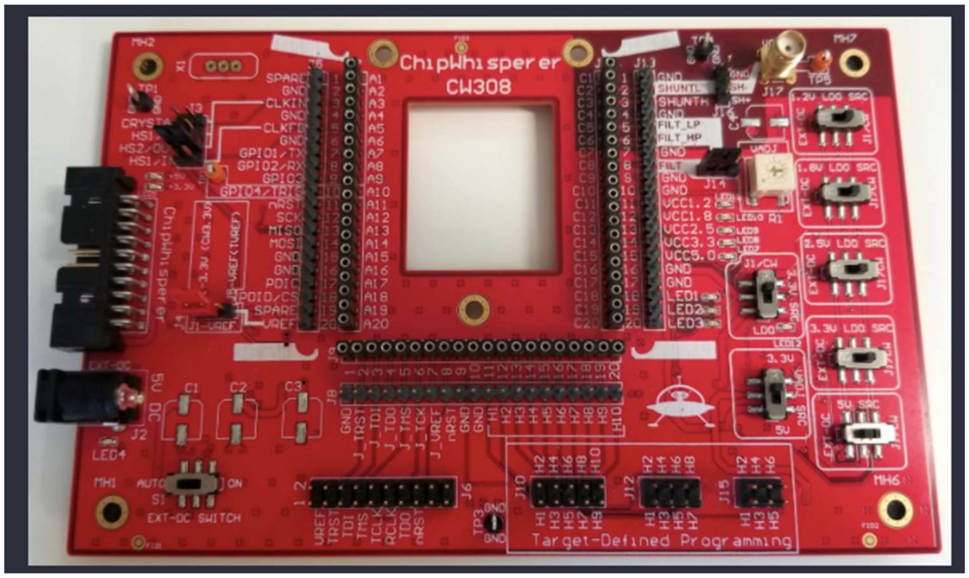


Figure 8: CW308 UFO Board [9]

2.3 CW308T-S6LX9 (Xilinx Spartan 6 LX9 in TQFP, FPGA) victim board

The Spartan 6 LX9 Target Board uses JTAG programming to communicate between the board and the computer. It uses 1.2 Vcc with 1 ohm shunt resistor. A single red LED (D1) is attached to the FPGA "DONE" pin on the board. When the FPGA is not programmed, this pin will be HIGH (and the LED will be on). That means, the LED will turn on as soon as the power is turned on. This LED will turn off once the FPGA has been correctly programmed [10].

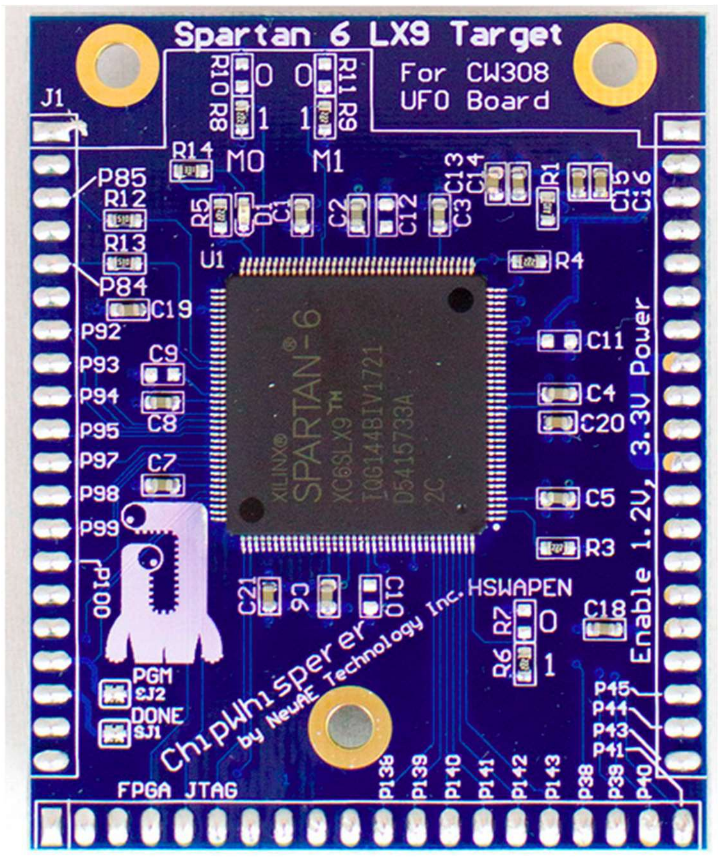


Figure 9: CW308T-S6LX9 (Xilinx Spartan 6 LX9 in TQFP, FPGA) victim board [10]

2.4 ISE design suite

The ISE Design Suite from Xilinx is used to create customized integrated circuits. The suite is most famous for designing the Field Programmable Array (FPGA), which allows a designer or user to customize a circuit after it has been manufactured. The ISE Design Suite was developed to help generate and analyze electronic circuit designs. The circuits are written in Hardware Description Language, a specialized computer language (HDL). The suite can be used to look at Register-Transfer Level diagrams, run timing analyses, and simulate various stimuli to see how the design reacts [11]. In the project, the Hardware Descriptive Language used is Verilog.

2.5 Intel Quartus Prime and ModelSim

Intel Quartus Prime is software for designing programmable logic devices. Quartus Prime allows developers to compile, execute timing analysis, analyze RTL diagrams, simulate a design's reaction to various stimuli, and configure the target device with the programmer using HDL designs. For hardware description, visual editing of logic circuits, and vector waveform simulation, Quartus Prime offers an implementation of VHDL and Verilog. ModelSim Simulator is used in this project to perform timing diagrams [12].

2.6 Xilinx iMPACT Tool

The iMPACT tool, when used in conjunction with a suitable JTAG interface, allows users to do boundary scans as well as direct programming and verification of Xilinx devices. The iMPACT tool also offers the ability to configure selected NOR flash devices for configuration indirectly linked to a Xilinx device [13].

2.7 VirtualBox (Linux)

VirtualBox is a general-purpose virtualization program for x86 and x86-64 hardware that allows users and administrators to run several guest operating systems on a single host. It is intended towards server, desktop, and embedded applications. VirtualBox supports Linux, Windows, and macOS [14].

2.8 ChipWhisperer python library

ChipWhisperer includes an open-source Python module for operating the capture hardware and interacting with the target [15].

2.9 Jupyter notebooks

The Jupyter Notebook is a web-based interactive environment that lets you combine code, photos, rich text, animations, videos, mathematical equations, maps, charts, interactive figures and widgets, and graphical user interfaces into one document. Jupyter's architecture is language independent. Because the client and kernel are decoupled, kernels can be written in any language [16].

As discussed in the Design Methodology section, much of the physical design/construction relied on the ChipWhisperer toolchain to provide a platform to capture and analyze the power from our benchmark circuit. As a result, much of the custom development focused on the Verilog structural models of the infected C6288 multiplier, the ROM extension, and placement of the trojan. To simulate and validate the functional design of the clean C6288 multiplier and the version infected with the half-adder trojan, the team relied on Intel Quartus and ModelSim. This combination enabled us to analyze and compare output and timing diagrams of the clean and infected C6288 in a controlled simulation. Figure X below demonstrates the ModelSim output of the clean circuit and verifies its functionality.

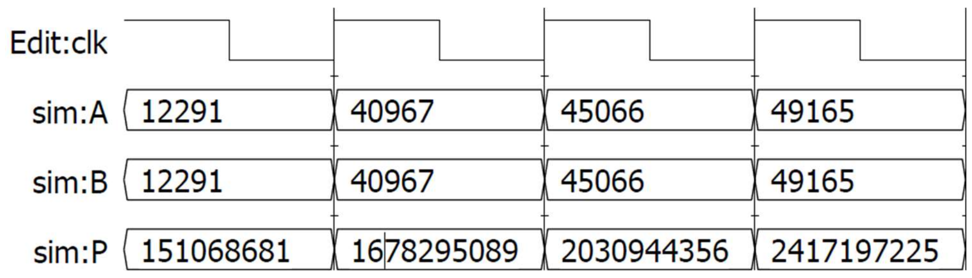


Figure 10: The multiplier timing diagram output through ModelSIM

Development of the Verilog model for the infected multiplier circuit first began by developing the Verilog code for a single half adder with the trojan payload/trigger components. This would first allow us to verify the bit flipping functionality of the trojan at a smaller scale, followed by implementing the trojan into the 16-bit position half adder of the full multiplier. Next, the team undertook an iterative analysis of the multiplier Verilog code to isolate the NOR and AND gates comprising the 16-bit position half adder. Successful infection of the multiplier was validated once again using ModelSim and analyzing the output for the 16-bit position bit flip. Figure X demonstrates the ModelSim timing diagram of the infected circuit where a subtle bit flip occurs at the 16-bit position of the output.

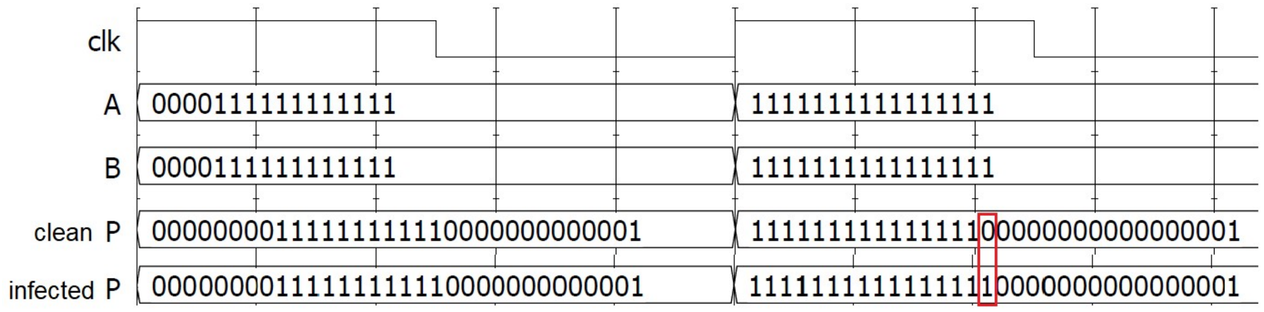


Figure 11: The single bit flipping due to trojan activation at the insertion point (bit 16)

Software interfacing required extensive setup of various development suites, including Intel Quartus, Xilinx ISE Project Navigator, ChipWhisperer toolchain, and a Jupyter server. The Xilinx ISE Project Navigator (deployed as a Linux Virtual Box image) was used to develop the Verilog code for the trojan and infected C6288 benchmark code. Further, this software suite provided the ISE Impact tool required to program the Spartan-6 target board via a JTAG-HS2 cable. After successful programming of the FPGA, the ChipWhisperer toolchain in combination with the Jupyter server was used to setup the default operating parameters of the CW1200 scope and capture the power traces of the benchmark circuit during active multiplication. This was done through standard Python3 libraries and ChipWhisperer-specific Python3 libraries executed on the local Jupyter server. In combination with ISE, we used Intel Quartus to view RTL schematics of our design and verify circuit functionality with ModelSim simulations.

7.0 Experimental Methods/Model Validation

Model testing of the infected and clean benchmark circuits was achieved using Intel Quartus ModelSim and physical testing was achieved with a simple process using the three LEDs on the Spartan-6 board to display the multiplier product. As previously mentioned, ModelSim was used to generate the timing diagrams and verify the 32-bit output of the infected and clean circuit in the figure above. To test the physical implementation of the multiplier on the board itself, a clock divider was used to slow down the incrementation of the address on the ROM and loop through the hard-coded ROM values in a more human-readable timing. This would feed the multiplier with the ROM values in a slower manner, and the output of the multiplier was tied to the 3 physical LEDs on the board that would represent bits 2 to 0 with the MSB on the left-most LED. Since the board only had three LEDs, the maximum product that could be achieved was decimal 7 (binary 111). As such, we were only able to physically test inputs into the multiplier that would maintain a product below the maximum.

Validation of the final design in terms of the power draw during active multiplication was achieved using the ChipWhisperer software toolchain. This was achieved with a Python script running on the Jupyter server to capture the current through a 1-Ohm shunt resistor on the capture board. After capturing the current, the raw data was dumped into an Excel file and converted to power values using the shunt resistor value. The subsequent power draws of the clean and infected C6288 Multiplier circuit are shown once again below.

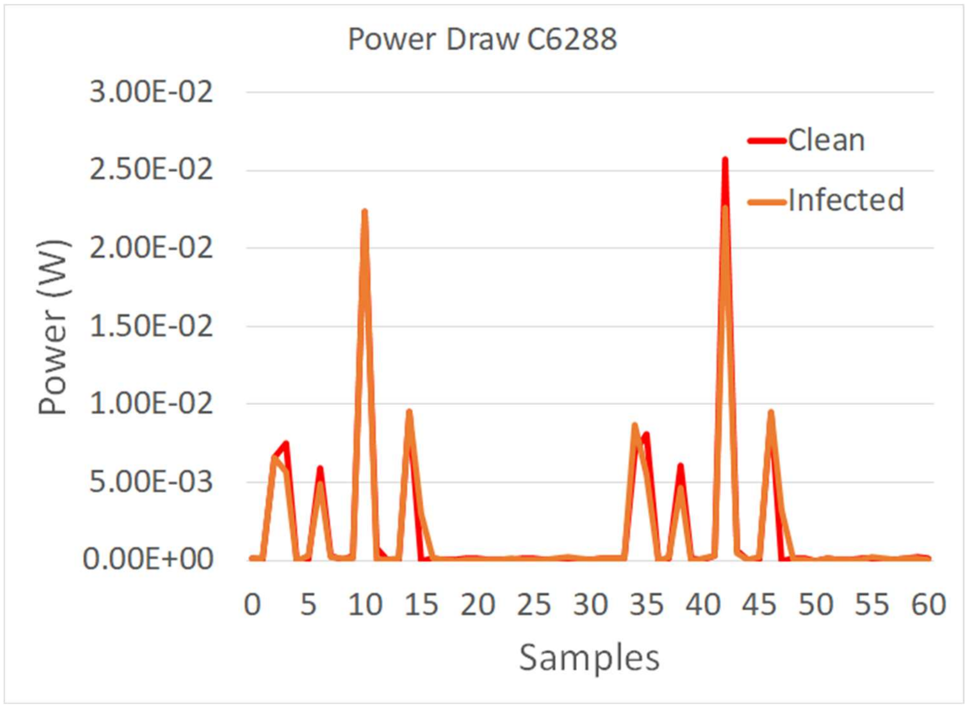


Figure 12: The power draw between clean and infected multiplier.

As can be observed above, no significant differences can be observed between the power draws of the infected versus clean C6288 Verilog models. In a real-world scenario, the minor differences between these trends could be attributed to either process variation effects or the electrical/EM noise that is present in each individual circuit. It is also important to highlight that in a real-world scenario, we have no means to leverage a clean (Golden) circuit as a standard to compare against a circuit that is potentially infected. As such, the best that one can do is look at the general trends of a power analysis and conclude that the trends seem similar to what would be expected.

10.0 Conclusion

In conclusion, the project was successful in implementing and simulating the 16 by 16-bit multiplier by displaying the clean and infected circuit. The measured and expected results were compared to be the same as the team expected to present a 32-bit value in binary as shown in figure 11. The goal was to multiply 00001111111111111111 by itself in binary which is 65535 in decimal. The clean circuit was expected to result in a correct value for this binary multiplication, which was achieved, and the result was 11111111111111100000000000000001 in binary which is 4294836225 in decimal. The infected circuit flipped bit position 16 from 0 to 1 which was expected after inserting a trojan to one of the half adders as shown in figure 2. The result obtained from the infected circuit for the multiplier showed that hardware trojan can result in a tremendous error, and the result of the infected multiplier was 4294901761 in decimal. However, the results produced from the power consumption graph were very similar as expected when comparing the two power trace graphs for the clean and infected circuit in figure 12. The minimal differences present in the power consumption graph might not contribute to a hardware trojan only but could be attributed to many other factors including but not limited to process variation effects, and electrical noise.

Trojan hardware problem affects most electronics industry and by displaying the minimal differences between a clean and infected circuit in this project, the threat of trojan hardware can be observed and its difficulty to detect it. The power consumption simulation and the multiplier results stood out as the innovative design aspect. The abilities of this design and simulation can contribute to alerting many researchers and industries in the electronics field to pay a closer attention to the danger of the hidden hardware trojan in the integrated circuits and how it cannot be detected by simply observing a graph using a side channel analysis or any other detection method because there is no such existence of a golden IC circuit in reality, therefore; the minimal differences can contribute to other factors that were mentioned in the previous paragraph and not only to a hardware trojan.

11.0 Back Matter

References

[1] D. Agrawal et al, “Trojan detection using IC fingerprinting”, IEEE Symp. on Security and Privacy, 2007.

[2] J. Li and J. Lach, “At-Speed Delay Characterization for IC Authentication and Trojan Horse Detection”, HOST, 2008.

[3] H. Salmani, M. Tehranipoor and J. Plusquellic, “New Design Strategy for Improving Hardware Trojan Detection and Reducing Trojan Activation Time”, HOST, 2009.

[4] L. Lin et al, “Trojan Side-Channels: Lightweight Hardware Trojans through Side-Channel Engineering”, CHES Workshop, 2009.

[5] B. Hou, C. He, L. Wang, Y. En, and S. Xie, “Hardware Trojan detection via current measurement: A method immune to process variation effects,” 2014 10th International Conference on Reliability, Maintainability and Safety (ICRMS), 2014.

[6] M. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," IEEE Design and Test, vol. 16, no. 3, pp. 72-80, July-Sept. 1999.

[7] N. Das, M. Saha and B. K. Sikdar, "Hard to Detect Combinational Hardware Trojans," 2018 8th International Symposium on Embedded Computing and System Design (ISED), 2018, pp. 194-198.

[8] N. A. E. T. Inc., “CW1200 ChipWhisperer-Pro¶,” *CW1200 ChipWhisperer-Pro - NewAE Hardware Product Documentation*. [Online]. Available: https://rtfm.newae.com/Capture/ChipWhisperer-Pro.html. [Accessed: 21-Jul-2021].

[9] N. A. E. T. Inc., “CW308 UFO,” *CW308 UFO - NewAE Hardware Product Documentation*. [Online]. Available: https://rtfm.newae.com/Targets/CW308 UFO.html. [Accessed: 21-Jul-2021].

[10] N. A. E. T. Inc., “CW308T-S6LX9¶,” *CW308T-S6LX9 - NewAE Hardware Product Documentation*. [Online]. Available: https://rtfm.newae.com/Targets/UFO Targets/CW308T-S6LX9.html#specifications. [Accessed: 21-Jul-2021].

[11] “Xilinx ISE Design Suite,” *Xilinx ISE Design Suite 14 Overview and Supported File Types*. [Online]. Available: https://fileinfo.com/software/xilinx/ise_design_suite. [Accessed: 21-Jul-2021].

[12] “Using the ModelSim-Intel FPGA Simulator by Drawing Waveforms,” no. January, pp. 1–25, 2020.

[13] “AR# 70681: ISE 14.7 iMPACT Flash Guidance,” *Xilinx*. [Online]. Available: https://www.xilinx.com/support/answers/70681.html. [Accessed: 22-Jul-2021].

[14] “What is the purpose of virtual box?,” *IT terms, answers and definitions*, 30-Mar-2021. [Online]. Available: https://defmean.com/what-is-the-purpose-of-virtual-box/. [Accessed: 22-Jul-2021].

[15] “ChipWhisperer,” *Overview - ChipWhisperer 5.5.2 documentation*. [Online]. Available: https://chipwhisperer.readthedocs.io/en/latest/getting-started.html. [Accessed: 22-Jul-2021].

[16] A. K. -, By, -, A. K. M. Editor, A. Kumaraswamy, and M. Editor, “10 reasons why data scientists love Jupyter notebooks,” *Packt Hub*, 23-Apr-2018. [Online]. Available: https://hub.packtpub.com/10-reasons-data-scientists-love-jupyter-notebooks/. [Accessed: 22-Jul-2021].

Appendices

Appendix A: Project Schedule

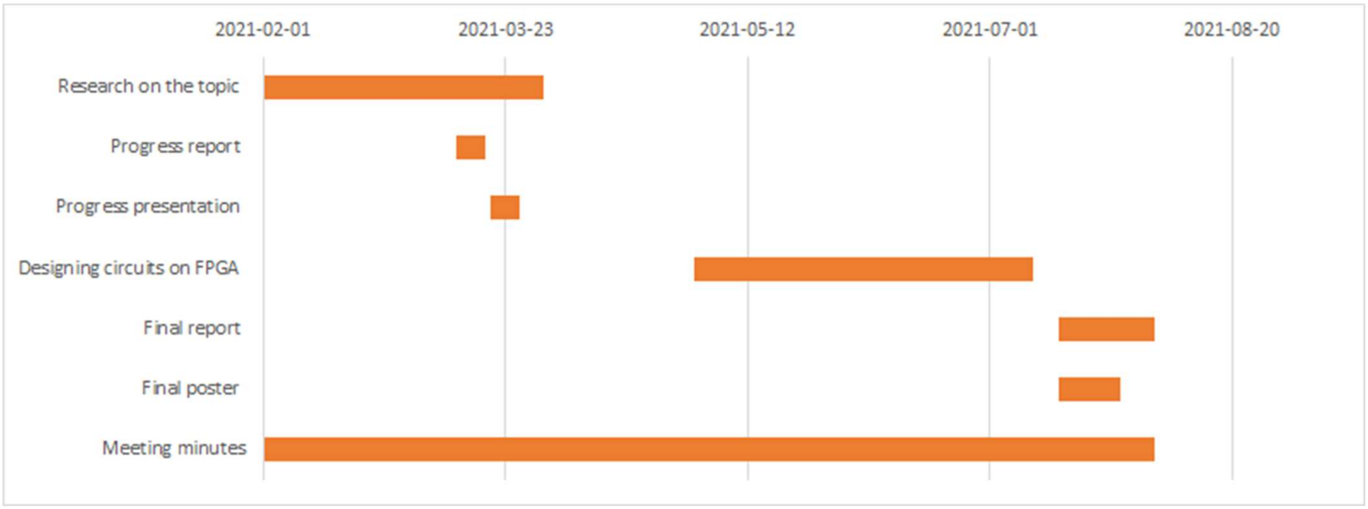


Figure 13: The Gantt chart displaying milestone and associated completed time range

Appendix B: Intermediate Results

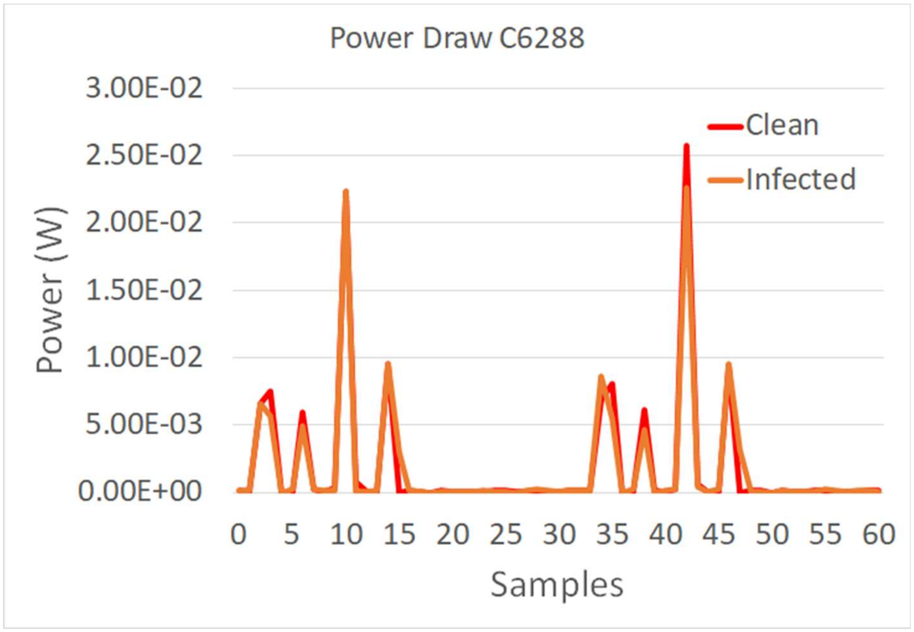


Figure 14: The power draw between clean and infected multiplier.

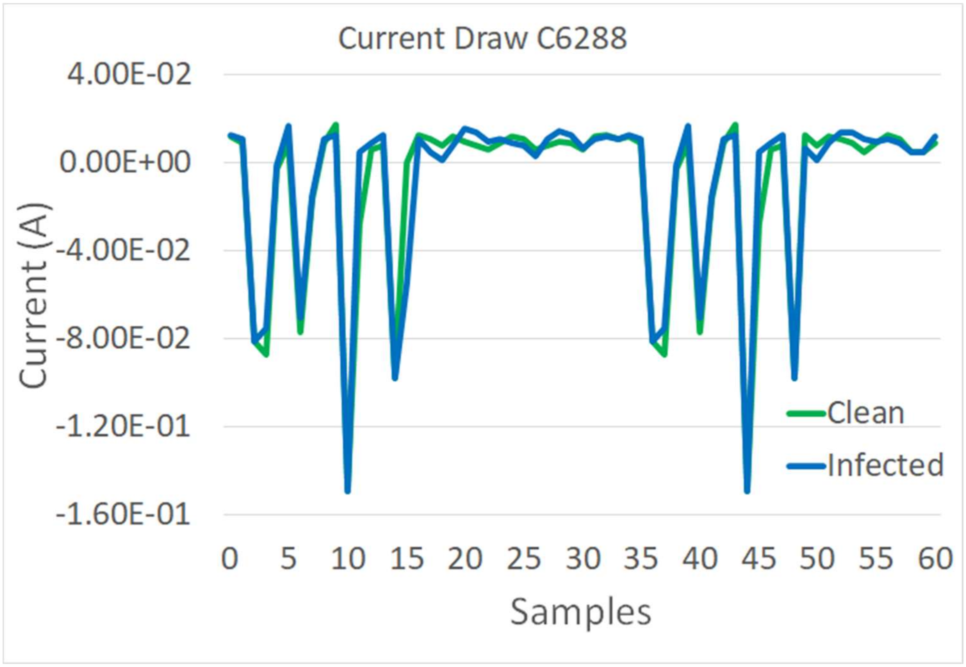


Figure 15: The current draw between clean and infected multiplier.

<u>Appendix C: Project Code</u>

C-1: Python script for capturing side-channel information

```
/* Khanh and Brajan, Team 23 W21*/

import chipwhisperer as cw
scope = cw.scope()
import numpy
target = cw.target(scope, cw.targets.SimpleSerial)
 scope.default_setup()
scope.adc.basic_mode = "high"
capture_clean_multi = cw.capture_trace(scope, target, plaintext=None, key=None,
ack=False)
numpy.savetxt("clean_multi.csv", capture_clean_multi.wave, delimiter=",")
%matplotlib notebook
import matplotlib.pylab as plt
plt.figure()
plt.plot(capture_clean_multi.wave)
plt.plot(capture_infected_multi.wave)
plt.xlim([0, 100])
plt.show()
plt.legend(['Clean Multiplier', 'Infected Multiplier'])
```

C-2: Verilog code for half-adder trojan

```
`timescale 1ns / 1ps
module Trojan(
output trojan_Sum,
input A,
input B,
input HA_Sum
);
xnor ht_trig(trigger, A, B);
m21 mux21(.Y(trojan_Sum), .D1(B), .D0(HA_Sum), .S(trigger));
endmodule
module m21(
output Y,
input D0,
input D1,
input S
);
wire T1, T2, Sbar;
and (T1, D1, S), (T2, D0, Sbar);
not (Sbar, S);
or (Y, T1, T2);
endmodule
```

C-3: Verilog code for ROM extension of the C6288 multiplier

```
module ROM_toplevel(clk, out1, out2);
output[15:0] out1;
output[15:0] out2;
input clk;
reg [15:0] out1;
reg [15:0] out2;
reg [2:0] addr = 3'b000;
wire [15:0] ROM[7:0];
assign ROM[0]=16'b0000000000001111;
assign ROM[1]=16'b0000000011111111;
assign ROM[2]=16'b0000111111111111;
assign ROM[3]=16'b1111111111111111;
assign ROM[4]=16'b0000000001000101;
assign ROM[5]=16'b0000000110100100;
assign ROM[6]=16'b0011010111111011;
assign ROM[7]=16'b1100111100100111;
assign ROM[7]=16'bz;
always @(posedge clk)
begin
addr <= addr + 1'b1;
out1 <= ROM[addr];
out2 <= ROM[addr];
end
endmodule
```

C-4: Verilog code for infected C6288

```
(* keep_hierarchy = "yes" *) module infected_circuit ( clk,
out6287, out6288, out6280, out6270,
out6260, out6250, out6240, out6230,
out6220, out6210, out6200, out6190,
out6180, out6170, out6160, out6150,
out6123, out5971, out5672, out5308,
out4946, out4591, out4241, out3895,
out3552, out3211, out2877, out2548,
out2223, out1901, out1581, out545,
IO4, clk2);
input clk;
output out6287, out6288, out6280, out6270,
out6260, out6250, out6240, out6230,
out6220, out6210, out6200, out6190,
out6180, out6170, out6160, out6150,
out6123, out5971, out5672, out5308,
out4946, out4591, out4241, out3895,
out3552, out3211, out2877, out2548,
out2223, out1901, out1581, out545,
IO4, clk2;
wire [15:0] A, B;
wire [31:0] P;
wire Z = 1'b1;
(* keep_hierarchy = "yes" *) Multiplier Ckt6288 (A, B, P);
(* keep_hierarchy = "yes" *) ROM_toplevel rom (.out1(A), .out2(B), .clk(clk));
assign
{out6287, out6288, out6280, out6270,
out6260, out6250, out6240, out6230,
out6220, out6210, out6200, out6190,
out6180, out6170, out6160, out6150,
out6123, out5971, out5672, out5308,
out4946, out4591, out4241, out3895,
out3552, out3211, out2877, out2548,
out2223, out1901, out1581, out545} = P[31:0];
assign IO4 = Z;
assign clk2 = 1'b0;
endmodule /* Circuit6288 */
/**************************************************************************/
module Multiplier (A, B, P);
input[15:0] A, B;
output[31:0] P;
wire C14_15, S14_15, S13_15, S12_15,
S11_15, S10_15, S9_15, S8_15,
S7_15, S6_15, S5_15, S4_15,
S3_15, S2_15, S1_15, trojan_S0_15,
S0_14, S0_13, S0_12, S0_11,
S0_10, S0_9, S0_8, S0_7,
S0_6, S0_5, S0_4, S0_3,
S0_2, S0_1, S0_0, A0B0;
assign
P[31:0] = {C14_15, S14_15, S13_15, S12_15,
S11_15, S10_15, S9_15, S8_15,
S7_15, S6_15, S5_15, S4_15,
S3_15, S2_15, S1_15, trojan_S0_15,
S0_14, S0_13, S0_12, S0_11,
S0_10, S0_9, S0_8, S0_7,
S0_6, S0_5, S0_4, S0_3,
S0_2, S0_1, S0_0, A0B0};
and GA0B0(A0B0, A[0], B[0]);
and GA0B1(A0B1, A[0], B[1]);
and GA0B2(A0B2, A[0], B[2]);
and GA0B3(A0B3, A[0], B[3]);



/********CODE TRUNCATED HERE FOR READABILITY**********/




/* HA MHA0_15(S0_15,C0_15,S1_14,C0_14); */
nor gn1_0_15(n1_0_15, S1_14, C0_14);
nor gn2_0_15(n2_0_15, n1_0_15, C0_14);
nor gn3_0_15(n3_0_15, S1_14, n1_0_15);
nor gn4_0_15(n4_0_15, n2_0_15, n3_0_15);
not gn5_0_15(n5_0_15, n4_0_15);
not gn6_0_15(n6_0_15, n5_0_15);
nor gn7_0_15(n7_0_15, n4_0_15, n5_0_15);
nor gn8_0_15(S0_15, n6_0_15, n7_0_15);
nor gn9_0_15(C0_15, n1_0_15, n5_0_15);

/*********************************************************************************/
Trojan trojan1(.trojan_Sum(trojan_S0_15), .A(S1_14), .B(C0_14), .HA_Sum(S0_15));/*****************/
/*********************************************************************************/

/* FA MFA1_15(S1_15,C1_15,S2_14,C1_14,C0_15); */
nor gn1_1_15(n1_1_15, S2_14, C1_14);
nor gn2_1_15(n2_1_15, n1_1_15, C1_14);
nor gn3_1_15(n3_1_15, S2_14, n1_1_15);
nor gn4_1_15(n4_1_15, n2_1_15, n3_1_15);
nor gn5_1_15(n5_1_15, C0_15, n4_1_15);
nor gn6_1_15(n6_1_15, C0_15, n5_1_15);
nor gn7_1_15(n7_1_15, n4_1_15, n5_1_15);
nor gn8_1_15(S1_15, n6_1_15, n7_1_15);
nor gn9_1_15(C1_15, n1_1_15, n5_1_15);




/********CODE TRUNCATED HERE FOR READABILITY**********/



/* FA MFA14_15(S14_15,C14_15,A15B15,C14_14,C13_15); */
```

```
nor gn1_14_15(n1_14_15, A15B15, C14_14);
nor gn2_14_15(n2_14_15, n1_14_15, C14_14);
nor gn3_14_15(n3_14_15, A15B15, n1_14_15);
nor gn4_14_15(n4_14_15, n2_14_15, n3_14_15);
nor gn5_14_15(n5_14_15, C13_15, n4_14_15);
nor gn6_14_15(n6_14_15, C13_15, n5_14_15);
nor gn7_14_15(n7_14_15, n4_14_15, n5_14_15);
nor gn8_14_15(S14_15, n6_14_15, n7_14_15);
nor gn9_14_15(C14_15, n1_14_15, n5_14_15);
endmodule /* TopLevel6288 */
/*************************************************************************/
```