Circumventing a Ring Oscillator Approach to FPGA-Based Hardware Trojan Detection

Justin Rilling, David Graziano, Jamin Hitchcock, Tim Meyer,
Xinying Wang, Phillip Jones, and Joseph Zambreno
Electrical and Computer Engineering
Iowa State University, Ames, IA, USA
{jrill, dgraz, jaminh, timmeyer, xinying, phjones, zambreno}@iastate.edu

***Abstract*—Ring oscillators are commonly used as a locking mechanism that binds a hardware design to a specific area of silicon within an integrated circuit (IC). This locking mechanism can be used to detect malicious modifications to the hardware design, also known as a hardware Trojan, in situations where such modifications result in a change to the physical placement of the design on the IC. However, careful consideration is needed when designing ring oscillators for such a scenario to guarantee the integrity of the locking mechanism. This paper presents a case study in which flaws discovered in a ring oscillator-based Trojan detection scheme allowed for the circumvention of the security mechanism and the implementation of a large and diverse set of hardware Trojans, limited only by hardware resources.**

I. Introduction

The 2010 Computer Security Awareness Week (CSAW) Embedded Systems Challenge hosted by the Polytechnic Institute of NYU presented student-led teams around the country with a hardware hacking challenge. Teams were given the RTL code for two different designs as well as a BASYS 2 evaluation platform with a Xilinx Spartan3E-100 Field Programmable Gate Array (FPGA). The two designs contained a specific hardening technique [1] to detect any modifications to their design. The challenge was to surreptitiously embed malicious circuitry, also known as hardware Trojans [2], into the design.

Our team concentrated the attack effort on the second design, codenamed "Beta", as we considered this to be the more vulnerable of the two designs. The Beta design consisted of an adder circuit with several embedded ring oscillators. A ring oscillator is a delay loop circuit, typically composed of wires and inverters, that oscillates at a particular frequency. This frequency is very sensitive to wire length, gate delay, and process variation [3]–[5]. In the Beta design, the embedded ring oscillators were inserted with the purpose of detecting any modifications to the hardware. Such modifications would change the wire lengths of the ring oscillator, resulting in a discrepancy in frequency values. The Beta design also included functionality that allowed the user to initiate a challenge for a particular ring oscillator value and receive the response via the 7-segment display on the BASYS 2 board. According to the challenge rules, all ring oscillator responses had to remain within 6.6% of their original value. This paper describes our successful circumvention of the Beta hardening scheme and describes the design methodology we used to create a large and diverse set of hardware Trojans.

II. Attack Methodology

We implemented two different approaches to circumventing the ring oscillator-based protection mechanism of Beta:

- A *design lockdown* approach that fixed the location of the ring oscillators.
- A *ring oscillator emulation* approach that reproduced the functionality of the ring oscillators with a look-up table.

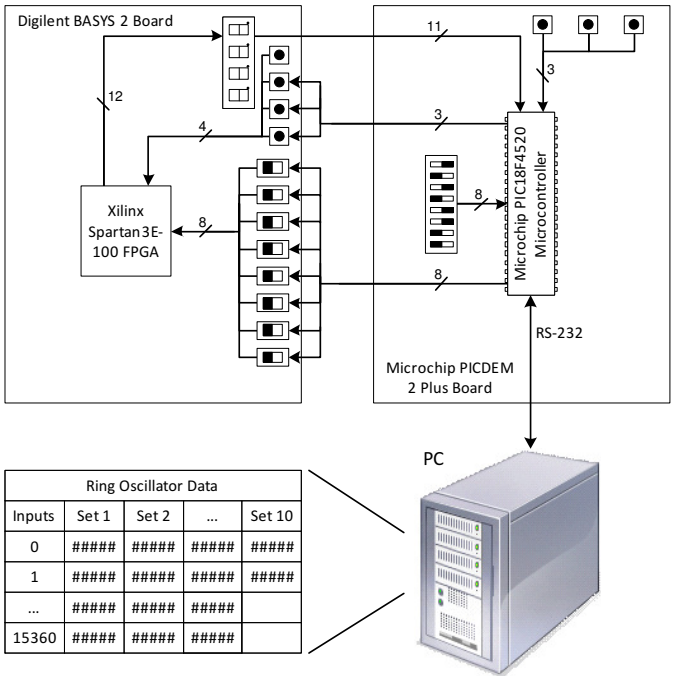
A. Design Lockdown

We first considered that changes made to the design would by default result in a modified physical placement and routing (P&R) of the ring oscillators. As noted previously, the queried ring oscillator value is by design very sensitive to wire length and other physical variations. By default, the Xilinx tools for P&R are completely automated to maintain high designer productivity but also offer options to give the designer more control. These options could be used to manipulate the location of the ring oscillators and fix the challenge responses while arbitrarily changing the rest of the design.

We used the Xilinx PlanAhead software (specifically the Floorplanner utility) to analyze the Beta design and to specify additional P&R constraints. With PlanAhead, we extracted the FPGA constraints file which we then applied to a compromised version of the Beta design. Using this method, the effect of the hardening scheme was reduced, but not completely removed. This allowed our team to implement a few small hardware Trojans that would have been detected without the use of this technique, but large Trojan designs still produced ring oscillator responses outside the acceptable variation. This was due to the fact that constraint violations were necessary to create valid placement and routing schemes for large designs which physically altered the ring oscillators.

B. Ring Oscillator Emulation

From a black box perspective, the Beta design takes in a sequence of two 8-bit values and outputs a corresponding 16-bit value. We reasoned that this functionality could be replaced by a module that would capture the input values and then fetch the corresponding output value from a lookup table, effectively emulating the ring oscillators. One issue with this approach was that inter-FPGA process variations would cause a slight difference between the correct response values on our FPGA and the correct response values on the FPGA



Ring Oscillator Data				
Inputs	Set 1	Set 2	...	Set 10
0	#####	#####	#####	#####
1	#####	#####	#####	#####
...	#####	#####	#####	
15360	#####	#####	#####	

Fig. 1. Automated testing environment

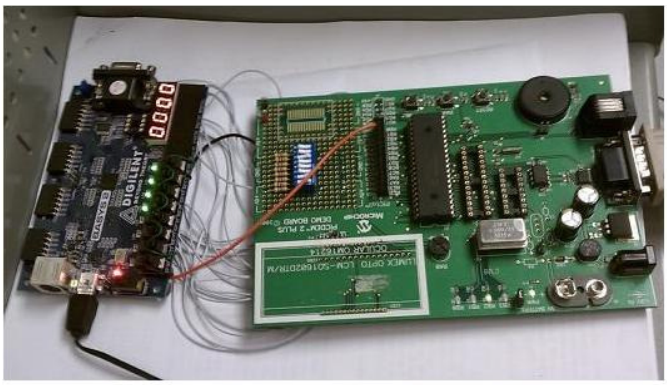


Fig. 2. Recording ring oscillator frequency responses

used to evaluate our design. However, given the wide range of acceptable responses, the outputs from our chip would most likely fall within the contest's allowable variation [6]. Another issue was storing the entirety of the challenge-response pairs, for as the Beta design required a 30KB (for 15,360 16-bit challenge-responses) lookup table, there was only 9KB of dedicated memory available on the Spartan3E-100 FPGA.

Further testing led to two key observations regarding the ring oscillator frequency values. First, for a given input, the lower 5-6 bits of the ring oscillator response varied in a seemingly random manner. Therefore, it wasn't necessary to store these bits in the lookup table as they could be reproduced with a pseudo-random number generator. Second, not all challenge vectors produced oscillations and there appeared to be long strings of 0's and 1's in the list of ring oscillator responses. We hypothesized it would be possible to compress the ring oscillator frequencies to a size that would fit on the FPGA, but needed to further characterize the ring oscillators to confirm our theory.

This ring oscillator characterization required multiple testing and recording passes of the output values, in order to analyze average responses and the amount of variation. To manually conduct this testing procedure would have taken several weeks to complete and would have been very tedious and error-prone. Consequently, our team attempted two different approaches to obtain these ring oscillator values.

1) Post-P&R Simulation: The Xilinx synthesis tools have the capability to generate timing models after each step of FPGA design generation. For example, after P&R, a Verilog or VHDL model of the design is created with components broken down into individual slices and LUTs, and timing information is added as delays for each individual component. The ModelSim HDL simulator was used to generate the timing results. Unfortunately, responses were off by a significant margin because the average-case timing information was not sufficiently accurate to model the ring oscillators. However, this simulation did correctly determine which challenge vectors produced oscillations and which vector sets produced long strings of 0's and 1's. This confirmed that the dataset was compressible.

2) Automated Testing Environment: We decided to take the direct approach of automating the testing procedure with additional hardware to obtain the ring oscillator values. Figure 1 shows our automated testing environment. The input and output pins of the FPGA were connected to a microcontroller on a PICDEM 2 Plus evaluation board. A PC sent input vectors to the microcontroller, which set the appropriate inputs on the FPGA. The microcontroller then read the 7-segment display and sent the value back to the PC where it was analyzed and recorded. Manual testing was still possible via the switches and pushbuttons on the PICDEM 2 Plus board. A picture of the testing setup is shown in Fig. 2. The automated testing hardware reduced the time to test the Beta design from a period of several weeks to a few hours.

Using the automated testing hardware, we recorded each ring oscillator response 10 times for all 15,360 inputs. The 10 datasets were analyzed to determine the average ring oscillator response for each input and amount of variation present within the data. Three distinct groups of data emerged: long strings of 0's and 1's with no variation, a few sporadic values greater than 1 but less than 2^8 which had no variation, and values greater than or equal to 2^{14} that typically had 5-6 bits of variation. Another interesting discovery was there were no values greater than $2^{15} - 1$ and, therefore, it wasn't necessary to store the most significant bit of the oscillator values in the lookup table as this was always zero.

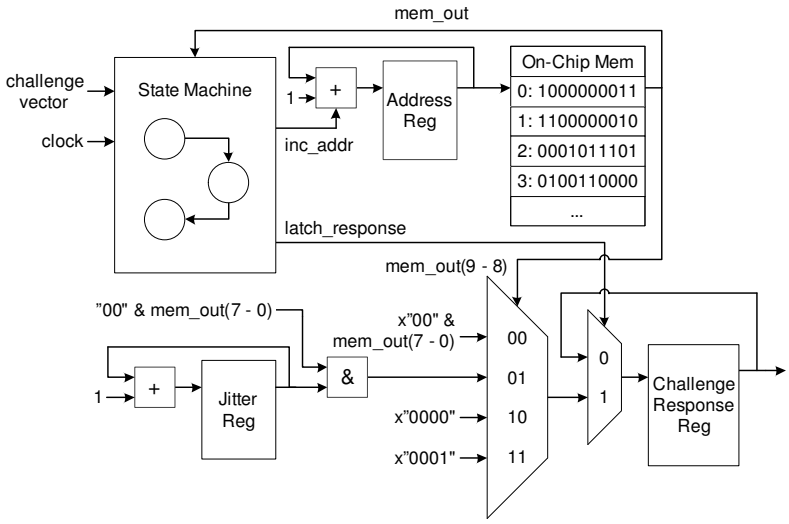
We created the compression scheme shown in Fig. 3(a) to compress the ring oscillator dataset. Each table entry is 10 bits wide. The most significant bit determines if the table entry represents a string of values (0's or 1's) or a single frequency value. When a table entry represents a string, the next most

Ring Oscillator Frequency Compression Scheme	
Bit #	Compression Encoding Rules
9	If string of 0's or 1's, set to 1 If single RO frequency, set to 0
8	If Bit 9 == 1, set to RO_Freq(0) If Bit 9 == 0, set to RO_Freq(14)
7 - 0	If Bit 9 == 1, set to string length If Bit 8 == 1, set to RO_Freq(13 downto 6) If Bit 8 == 0, set to RO_Freq(7 downto 0)

Challenge Vector	Response
0x0000	0
0x0001	0
0x0002	0
0x0003	1
0x0004	1
0x0005	3
0x0006	0x4C24
...	...

On-Chip Mem
0: 1000000011
1: 1100000010
2: 0000000011
3: 0100110000
...

(a) Compression scheme



(b) Response emulation architecture

Fig. 3. Table lookup-based ring oscillator emulator module

significant bit specifies if the string is composed of 0's or 1's and the lower 8 bits specifies the length of the string. If the string length is greater than 255, then the string is divided into several table entries. When a table entry represents a single frequency value, the next most significant bit represents bit 14 of the frequency value. The lower eight bits represent either bits 13-6 or 7-0 of the frequency value, depending if the frequency is large (greater than or equal to 2^{14}) or small, respectively. This distinction is necessary because small frequency values have no variation whereas large frequency values always have some variation. Using this compression scheme, we reduced the size of the lookup table from 30KB to less than 6KB.

A ring oscillator emulator module was created based on the compressed lookup table approach (see Fig. 3(b)). A state machine was implemented to latch the challenge vector, perform a sequential search of the compressed lookup table, and return the corresponding ring oscillator challenge response. To reproduce the slight variation present in the least six significant bits of values greater than or equal to 2^{14}, a counter incrementing every clock cycle was latched when the ring oscillator value was output. This appeared to be random variation to the user. Implementing this module in the Beta design allowed us to add any Trojan to the design without affecting the ring oscillator value displayed to the user.

Table 1 shows the overhead introduced by the ring oscillator emulator module. The impact on area and delay was minimal. The remaining flip-flops and LUTs allowed for the implementation of elaborate Trojans several times the size of the original circuit. Since the original Beta design didn't utilize the on-chip block RAMs, we were able to use all 4 BRAMs to store the compressed ring oscillator frequency dataset.

Beta Design	Flip-Flops	LUTs	BRAMs	Delay
Original	98 (5%)	152 (7%)	0 (0%)	3.409ns
w/ RO Emulator	128 (6%)	237 (12%)	4 (100%)	3.931ns

TABLE I
RING OSCILLATOR EMULATOR MODULE OVERHEAD

III. TROJAN DESIGN

We designed a diverse set of Trojans based on the taxonomy shown in Fig. 4 (derived from [2]). Each Trojan implemented a circumvention technique to mitigate the Beta hardening method as well as an effect and activation mechanism. While our Trojans specifically targeted the Beta design, we remained mindful of their real-world practicality.

A. Effects

We concentrated on two aspects of functional modification: the frequency of modification and the type of modification. These aspects are highly dependent on the target. A Trojan that constantly modifies functionality will have a large initial impact but is more likely to be detected. A Trojan that produces malfunctions very sporadically will have less impact in the short term, but may have a larger impact in the long term since its presence could go unnoticed. Similarly, the type of malfunction should be large enough to have an impact but small enough to minimize detection.

Keeping these two aspects in mind, we implemented Trojans with both types of malfunction frequency while trying to keep the type of malfunction moderate. One Trojan replaced an AND gate with a XOR gate in the adder to create a sporadic malfunction dependent on the input value. Another persistently inverted a single bit of the output.

Since the Beta design didn't contain "secret" information, we focused on different methods of leaking information from the FPGA. The BASYS 2 board contained minimal hardware and peripherals. This offered both an advantage and a disadvantage in terms of leaking information. The disadvantage was

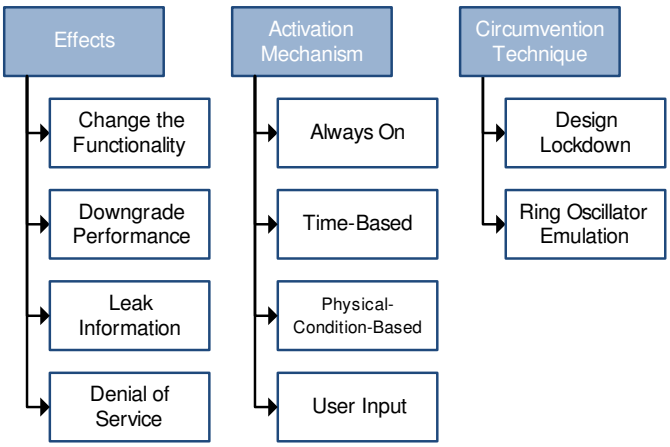


Fig. 4. Taxonomy of our implemented Trojans (derived from [2])

that there were simply less channels to transmit information. The advantage was that all output peripherals (except the USB) were directly controlled by the FPGA, creating a larger Trojan design space to leak information. For instance, the FPGA was connected directly to the VGA port. This allowed for the implementation of Trojans that exploited the VGA protocol, such as leaking information on the data lines during the horizontal and vertical syncs as described in [7].

One of our Trojans used the external oscillator IC socket on the BASYS 2 board as an antenna to leak information via an RF signal in the AM radio frequency range. The transmission could be heard as a beeping pattern on a standard AM radio. The radio had to be very close (a few inches) to the board as the antenna was not optimal, but extending the length of the antenna by placing a wire in the IC socket extended the reception distance to several feet.

Similar to functional modification, our performance degradation design methodology concentrated on the frequency and level of reduced performance. One way in which we degraded the performance of Beta was by implementing a Trojan that increased the timing delay of the addition operation. The Trojan held the inputs for a pre-set number of cycles before sending the data to the adder. Trojans on the opposite end of the spectrum produced a constant denial of service once activated.

B. Activation Mechanism

Activation mechanisms were chosen to correspond to the Trojan effect. For instance, the "Always On" activation mechanism would be inappropriate to pair with a "Denial of Service" effect as this Trojan would certainly be detected during chip verification given a real-world scenario. However, this activation mechanism may be appropriate for a Trojan that covertly leaks information. For many Trojans, we used a combination of triggers to complicate activation.

An example of an activation mechanism we implemented was one that used a combination of a specific user input with a physical-condition-based internal trigger. This trigger was designed to work with the ring oscillator emulator module circumvention technique. In this case, the ring oscillators embedded in the Beta design were no longer utilized and could be used to create a crude temperature sensor by exploiting the relationship between temperature and the frequency of ring oscillation [8]. Since an unexpected rise in temperature could activate the Trojan before it could affect its intended target, the activation mechanism was combined with a specific user trigger. As temperature increased, the ring oscillator interconnection resistance increased, reducing the frequency of oscillations. The room temperature ring oscillator value was set as the basis for comparison. When the particular input was entered, the ring oscillator value was compared with the base value. If the difference was greater than a set threshold, then a Trojan was activated to produce a slight error in the result.

IV. Conclusion

Our ring oscillator emulation approach was successful in circumventing the Beta hardening mechanism and allowing the implementation of any Trojan the hardware resources would support. This was achieved by exploiting the relatively small challenge-response set associated with Beta's ring oscillator-based Trojan detection method. We acknowledge this approach was specific to the Beta hardening scheme and is limited by the number of challenge-response vectors in general. We also acknowledge this scheme was based on the assumption that the design would be verified at room temperature using standard operating voltage; the same environment in which we measured the ring oscillator frequency values. Our design was evaluated at the CSAW event in October 2010 and resulted in a 2^{nd} place finish in the embedded systems challenge.

Acknowledgment

This work was supported in part by the National Science Foundation (NSF) grant CNS-0958510. We would also like to thank Xilinx for providing the BASYS 2 platform, and the Polytechnic Institute of NYU for organizing the CSAW event.

References

[1] J. Rajendran, V. Jyothi, O. Sinanoglu, and R. Karri, "Design and analysis of ring oscillator based design-for-trust technique," in *VLSI Test Symposium (VTS)*, May 2011, pp. 105–110.

[2] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, Oct. 2010.

[3] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proceedings of the Design Automation Conference (DAC)*, 2007, pp. 9–14.

[4] J. Guajardo, S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2007, pp. 63–80.

[5] J. Li and J. Lach, "At-speed delay characterization for IC authentication and trojan horse detection," in *Hardware-Oriented Security and Trust (HOST)*, June 2008, pp. 8–14.

[6] A. Maiti, J. Casarona, L. McHale, and P. Schaumont, "A large scale characterization of RO-PUF," in *Hardware-Oriented Security and Trust (HOST)*, June 2010, pp. 94–99.

[7] A. Baumgarten, M. Steffen, M. Clausman, and J. Zambreno, "A case study in hardware trojan design and implementation," *International Journal of Information Security*, vol. 10, pp. 1–14, 2011.

[8] K. M. Zick and J. P. Hayes, "On-line sensing for healthier FPGA systems," in *Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA)*, 2010, pp. 239–248.