R2D2: Runtime Reassurance and Detection of A2 Trojan

Yumin Hou*, Hu He*, Kaveh Shamsi†, Yier Jin†, Dong Wu*, Huaqiang Wu*
*Institute of Microelectronics, Tsinghua University, Beijing 100084, China
†Department of Electrical and Computer Engineering, University of Florida, USA
hou-ym12@mails.tsinghua.edu.cn, hehu@tsinghua.edu.cn, kshamsi@ufl.edu, yier.jin@ece.ufl.edu
dongwu@tsinghua.edu.cn, wuhq@tsinghua.edu.cn
Correspondent Author: Hu He

***Abstract*—With the globalization of semiconductor industry, hardware security issues have been gaining increasing attention. Among all hardware security threats, the insertion of hardware Trojans is one of the main concerns. Meanwhile, many current Trojan detection solutions follow the assumption that the hardware Trojan itself should be composed of digital logic. This assumption is invalidated by recently proposed analog Trojans which are extremely small and can detect rare events. This paper proposes a runtime hardware Trojan detection method which is geared towards detecting such advanced Trojans. The principle of this method is to guard a set of concerned signals, and initiate a hardware interrupt request when abnormal toggling events occur in these guarded signals. To prove the effectiveness of this method, we design a processor based on ARMv7-A&R ISA, and insert an analog Trojan into the processor. We fabricated the design in the SMIC 130 nm process and demonstrate the effectiveness of the proposed methodology.**

I. Introduction

Today's globalized semiconductor industry is facing critical security, integrity, and privacy concerns. Among all hardware security threats including reverse engineering, IP piracy, etc., the insertion of malicious logic (aka hardware Trojans) is still one of the main concerns. The globalization of the integrated circuit (IC) supply chain makes it difficult and costly for regulations only to maintain the integrity of the IC design through the design and fabrication process. This is especially true for the case of third party components.

Upon this challenge, researchers from the government, industry and academia have proposed various techniques to help identify malicious logic both pre-fabrication, on RTL, netlist, and layout levels, as well as post-fabrication on manufactured circuits, using a combination of special testing pattern generation techniques and side-channel analysis. A fundamental limitation of all these schemes is that the effect of a small enough Trojan on the logic and on the side-channel fingerprints of the circuit, can be masked by process variation and noise. This is exacerbated by the ever-increasing scale of integration and process-variation in advanced nodes.

Recently proposed analog and RF Trojans [1] fall into this category. An analog Trojan circuit can detect an extremely rare sequential event with just a handful of transistors added to the circuit. This hurdles even invasive detection techniques and poses a real threat to the IC supply chain despite a decade of research in the area.

In this paper, we present a novel on-chip hardware Trojan detection mechanism called R2D2 geared towards such analog Trojans. Since such analog Trojans are triggered by high frequency wire-flops in the processor, we propose an abnormal-toggling detection scheme that can easily be integrated into the processor with low overhead and discuss why it is difficult to remove it from the design. The main contributions of the paper are listed as follows:

- We develop an on-chip Trojan detection method. This method targets hardware Trojans triggered by a successive toggling events. We present an in-depth security analysis of the scheme;
- We design a processor based on the ARMv7-A&R ISA, and insert an analog Trojan (based on the A2 Trojan [1]), into the ARM processor. We explore the architecture of the processor and present various novel ways to integrate the Trojan and its payload;
- We provide simulation results, which demonstrate that the analog Trojan works on the ARM processor, and the R2D2 method is effective in detecting the analog Trojan. We also fabricate a proof-of-concept chip in the SMIC 130 nm Mixed-Signal 1P7M process with the hardware Trojan and the detection mechanism.

The remainder of the paper is organized as follows: Section II provides a background. Section III presents the R2D2 detection scheme. Section IV provides an overview of the implemented processor. Section V presents experimental results, and Section VI concludes the paper.

II. Background

A. Hardware Trojan Detection

Hardware Trojans are typically categorized according to a) their triggering method e.g. sequential or combinational, b) their payload e.g. active modification of logic values, or passive leakage of information through side-channels [2]. Post-fabrication detection mechanisms fall into testing-based [3], [4], or side-channel-based [5], [6]. Various statistical methods have been used to extract the side-channel traces of a Trojan in a sea of other components. In addition, design time techniques can also accompany post-fabrication detection such as reducing rare-events in the circuit [7], or inserting on-chip sensors that will be measured post-fabrication for Trojan detection [8].

B. Analog Hardware Trojans

Sequentially triggered Trojans can be made extremely difficult to detect through testing patterns. A digital sequential Trojan typically requires a large FSM to detect a rare sequence of events, which in turn can reduce the Trojans resiliency to side-channel detection techniques. Analog switch-capacitor circuits on the other hand can be used to do signal shaping and detection with a much lower transistor count. The analog Trojan presented in [1] known as A2 uses a simple analog circuit to detect high frequency toggling.

A2 is a small analog circuit, which can be inserted into an already placed and routed design. It reads a digital pulse signal (the trigger input), and triggers the payload when the pulse signal has toggled with a high frequently for a certain period of time. The trigger input is connected to a signal that can be toggled with high frequency through a special code snipper running on the processor. The attacker insures that the trigger signal has a much lower toggle rate during typical workloads. This makes detecting the hardware Trojan difficult through testing, not to mention that there exists many low toggling frequency bits in modern processors.

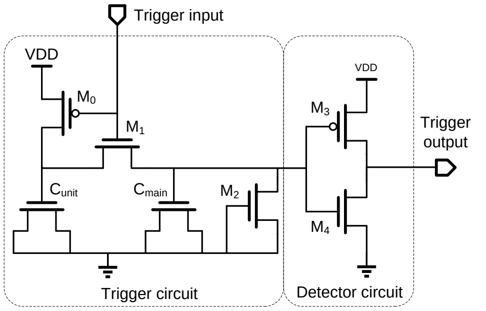

Figure 1: Transistor schematic of the analog Trojan circuit from [1]

The transistor schematic of the A2 analog Trojan [1] circuit is shown in Figure 1. When the trigger input is low, C_{unit} is charged to VDD. When the trigger input switches to high, C_{unit} shares its charge with C_{main}. This will raise the charge on C_{main} by an amount controllable by the size ratio of C_{main} and C_{unit}. When the trigger input is stationary at either high or low, the charge at C_{main} dissipates through M_2 and other stray-paths. Hence, only with sufficiently frequent toggling of the trigger input, one can raise C_{main}'s voltage. This voltage is fed to a detector circuit which is an imbalanced inverter with a controllable switching threshold. The attacker connects the trigger input to a software controllable bit which has a low toggling rate, and uses the trigger output to launch the payload.

III. The R2D2 Detection Scheme

In this paper, we propose an on-chip runtime hardware Trojan detection method. The detection scheme targets A2-alike analog hardware Trojans. The proposed detection method aims to detect high frequency toggling on several signals before they can activate the hardware Trojan.

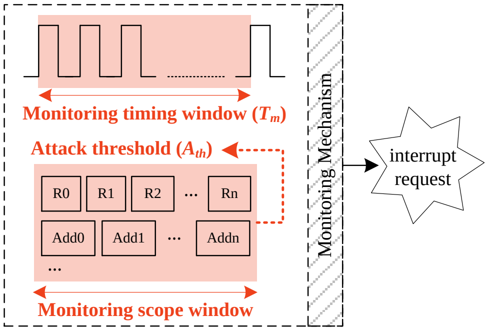

Figure 2: Mechanism of the hardware Trojan detection method

Due to the small size of the A2 Trojan, and the fact that it can be connected to any low toggling signal, we conclude that runtime detection is more feasible. Figure 2 shows the mechanism of the proposed runtime hardware Trojan detection scheme. The principle of this method is to guard a set of concerned software controllable registers or memory related signals. A hardware interrupt will be generated if abnormal toggling events occur in the guarded items. The mechanism cannot be disabled through unprivileged software.

As shown in Figure 2, R2D2 has several parameters that must be tuned in order to ensure the effectiveness of the scheme and eliminate false positives. The first parameter is the *monitoring timing window size* denoted by T_m. During a monitoring window the detection unit counts the toggling events on the concerned signal. Throughout this time window, if the toggling frequency increases beyond the *Attack threshold* parameter Ath the detection circuit will generate an interrupt request. The other important parameter is the *monitoring scope* which decides the signals to be selected for monitoring. These signals must be ones that have a low toggling frequency during normal processor workloads. Each guarded signal can have a different attack threshold value.

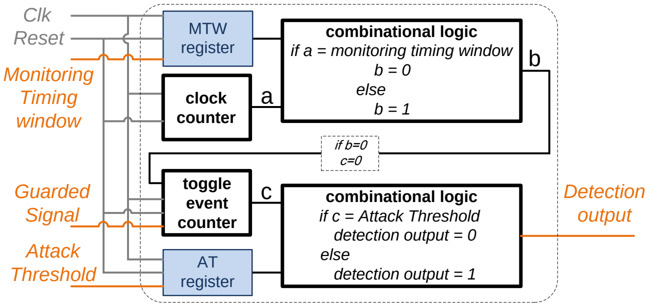

Figure 3: Runtime detection circuit design

The detection circuit is shown in Figure 3. The window size, T_m, and Ath are written into dedicated registers, Monitoring Timing Window register (MTW) and Attack Threshold register (AT) respectively. By keeping these values as software programmable registers, we can create flexibility and uncertainty to the defence mechanism and prevent the attacker from learning them through IC reverse engineering. We must ensure that only privileged software can configure these registers. The clock counter is used to count the clock cycle, and compare with the value in the MTW register. The clock counter is reset when its value reaches the value in the MTW register, and a new monitoring window starts. The toggle event counter is used to count the number of toggle events of the guarded signal, and compare them with the value in the AT register. This toggle event counter is reset when a new monitoring timing window starts. In one monitoring window, if the toggle event counter reaches the value in the AT register, the detection output (the alarm signal) will be activated.

IV. Demonstration Design

We demonstrate the effectiveness of the proposed detection scheme and the operation of the hardware Trojan itself on an in-house designed ARM-compatible processor which is described herein. ARM processors are the most popular processors in the mobile and embedded system domain. Hence, the security of systems based on ARM architectures is of critical concern.

A. The ARM-compatible Processor

We propose a fused microarchitecture based on the ARMv7-A&R ISA [9]. ARMv7-A&R was the up to date ARM ISA when we started this project. The fused microarchiecture [10] was evaluated based on the gem5 simulator [11], and proved to be feasible, before real hardware design. This ARM processor is named Merlin. Using microarchitectural techniques, Merlin [12] expands the DSP capabilities of the ARM processor. Merlin supports most traditional ARM instructions, but does not support some ISA extensions, such as Thumb, ThumbEE, Jazelle, Floating-point, and Advanced SIMD. We realize 181 ARM instructions in total, which is enough to run common benchmarks, such as DhryStone, CoreMark, DSPStone, and EEMBC telecom. There are 7 execution modes defined in ARMv7-A&R ISA, while Merlin works only under user mode.

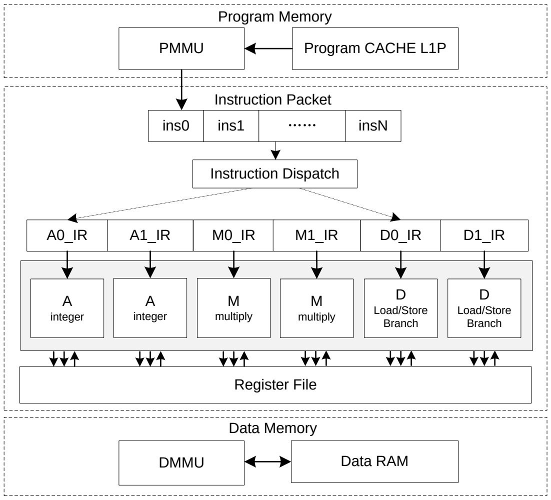

Figure 4: Architecture of Merlin processor

Merlin adopts a fused microarchitecture [13]–[15] integrating in-order superscalar and VLIW [16]. Normally, Merlin works under dual-issue in-order superscalar mode. It can be switched to 6-issue VLIW mode when the task is compute-intensive. Mode switch can be performed through software. The VLIW approach expands the ILP of the Merlin processor, without modifying the ARM ISA. So Merlin can be used as an MCU, or a DSP under different application scenarios. The architecture of Merlin is shown in Figure 4. Merlin has 16 KB of on-chip L1 instruction Cache, with a 256-bit wide port, and 32 KB dual-port data memory, with each port being 64-bits wide. Merlin has 6 functional units, consisting of 2 arithmetic units (A), 2 multiply units (M), and 2 load/store units (D).

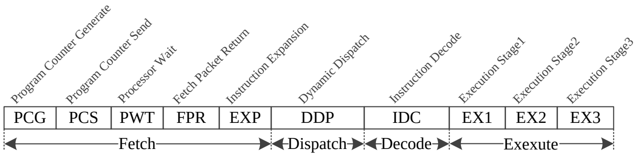

Figure 5: Pipeline of Merlin processor

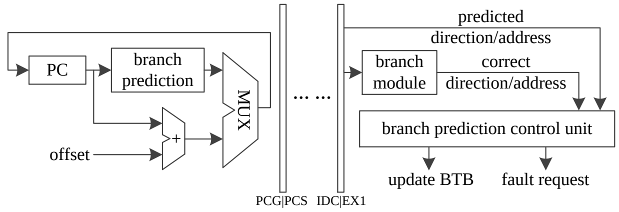

Figure 6: Branch prediction processing flow

As shown in Figure 5, Merlin has a 10-stage pipeline. At the dispatch stage (DDP), the instructions are dispatched using in-order superscalar or VLIW, which is decided by the user. Branch prediction plays a significant role in improving the processor performance, especially for processors with deep pipeline stages. In this design, we propose a combined Bimodal and PAp branch prediction method [17]–[20]. This method achieves 94% prediction accuracy, with limited hardware budget. Figure 6 shows the branch processing flow for Merlin. Branch prediction mainly solves two problems: predict the branch direction, and recover from mispredictions. For Merlin, branch prediction happens at the PCG stage, which is the first stage of the pipeline, as shown in Figure 5. At EX1 stage, the correct branch direction can be acquired, and the predicted direction can be verified to be correct or not. If not, the pipeline will be flushed, and restarted from the correct place. Afterwards, the corresponding branch information recorded in the BTB (Branch Target Buffer) should be corrected.

An SoC is designed [21], where Merlin is used as an MCU. The chip diagram is shown in Figure 7. On this chip, Merlin is integrated with DMA, ROM, SRAM, four 128KB embedded ReRAM, and a variety of peripheral I/O. The Dhrystone performance of Merlin is 1.9 DMIPS/MHz, which is comparable to ARM Cortex-A8 processors.

B. A2-like Analog Trojan in Merlin

When inserting the analog Trojan trigger circuit into the Merlin processor, we should first select a viable trigger input.

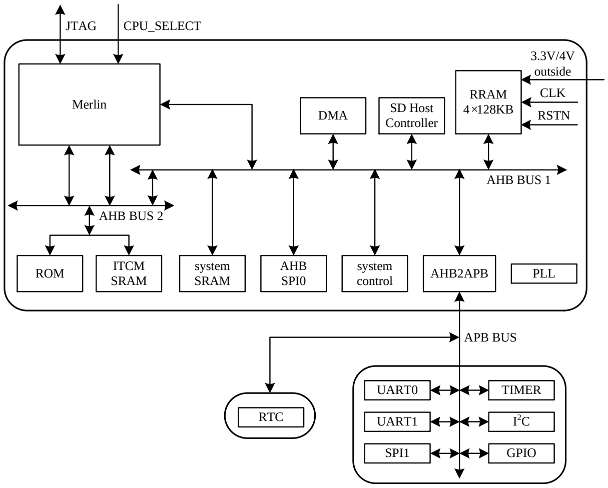

Figure 7: The SoC chip diagram

The trigger input should have low toggling rate in common cases. It should be controllable through software, so that the trigger code can make it toggle at high frequency to launch the attack. We also discuss what can be utilized as the payload of the trigger in the Merlin processor.

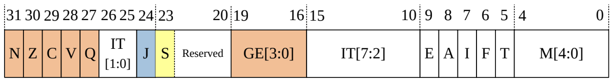

Figure 8: ARM CPSR register

1) Select the Trigger Input: In ARMv7-A&R ISA, there are 16 core registers including R_0-R_{12}, SP (Stack Pointer), LR (Link Register), and PC (Program Counter) under user mode. For each core register, it is possible that the register is frequently used during a time period. So it is not safe to utilize these registers to trigger the attack.

Another software reachable register is CPSR (Current Program Status Register). The definition of CPSR is shown in Figure 8. APSR (Application Program Status Register) is the same register as the CPSR in ARMv7-A&R ISA, but the APSR must be used only to access the N, Z, C, V, Q, and GE[3:0] bits. N, Z, C, and V are condition flags. Q is the overflow or saturation flag. GE[3:0] are the greater-than or equal flags. In the Merlin processor, we realize all these flag registers as part of APSR. All the flag registers can be modified directly using the MSR instruction, or be modified indirectly using arithmetic or logic instructions.

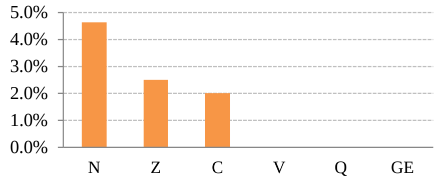

Figure 9: Toggling rate of the NZCV, Q, and GE registers when running the MFCC program

Figure 9 shows the toggling rate of the NZCV, Q, and GE registers when running MFCC, which is a speech recognition program. We can see that the toggling rate of N, Z, C registers are all below 5%. V, Q, GE registers does not toggle at all in this benchmark. So these registers can be utilized as the trigger input. We also utilize one reserved bit, CPSR[23], as the mode switch flag in Merlin. We name it CPSR_S. The toggle rate of CPSR_S is decided by the users. CPSR_S always has very low toggling rate, since it may degrade the performance if the processor is to switch between the two modes too frequently. So the CPSR_S bit can also be used as the trigger input.

Table I: Some special instructions in ARMv7-A&R ISA

Instruction	Introduction
PLD, PLDW, PLI	Preloading caches
CLREX	Clear local exclusive access record
DBG	Provide a hint to debug and related systems
DMB, DSB	Memory barriers that regulate memory accesses

The bits in CPSR that we do not implement in Merlin include IT[7:0], J, T, E, A, I, F, and M[4:0]. These bits cannot be modified by MSR instruction directly, but some of these bits can still be used as the trigger input. J and T compose the instruction set state register. It indicates whether the processor is working under ARM, Thumb, ThumbEE, or Jazelle mode. The BLX instruction calls a subroutine at a PC-relative address, and changes instruction set from ARM to Thumb, or from Thumb to ARM. Exchange of the instruction set between ARM and Thumb can make the T bit toggle frequently. While there is little chance that this happens in common cases. So the BLX can be utilized to trigger a Trojan in an ARM processor. IT[7:0] is the IT block state register. This field holds the If-Then execution state bits for the Thumb IT instruction. It is possible to make one bit in IT[7:0] toggle by exchanging between IT mode and normal mode. E is the endianness mapping register. Normally, endianness is not changed in one application, so the E bit almost does not toggle at all. But we can use the SETEND instruction to set and clear this bit, to make E bit toggle frequently. A (Asynchronous abort), I (IRQ), and F (FIQ) are mask bits. These bits are less software controllable, and it could be dangerous to use these bits to trigger a Trojan attack.

There are also many special instructions which are rarely used in common programs. Signals related to these instructions are predicted to have low toggling rates. So these instructions can also be used to trigger the attack. We list some of the special instructions in Table I. For example, the PLD instruction signals the memory system that data memory accesses from a specified address are likely to happen in the near future. The memory system can respond by preloading the cache line into the data cache (pre-fetching). In Merlin, PLD is decoded in the D unit, and then it signals the DMMU. Continuous execution of PLD can make the related signals toggle frequently.

Regarding Merlin, there is another method to trigger the attack. As we mentioned before, Merlin adopts a combined Bimodal and PAp branch prediction method. Merlin fetches 256bit instructions each time, including 8 to 16 instructions. This is called an instruction packet. Once branch instructions

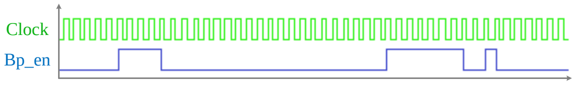

Figure 10: Branch prediction enable signal toggling rate when running MFCC program

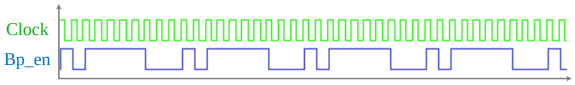

Figure 11: The branch prediction enable signal toggles more frequently if triggered by software

are found in the instruction packet, the branch prediction mechanism is enabled. As shown in Figure 10, Bp_en stands for branch prediction enable signal. We can see that the Bp_en signal rarely toggles when running MFCC program. Whereas, we can make the branch predictor work more frequently simply by adding branch instructions into the program. Figure 11 shows that, when running the designed program, the Bp_en signal toggles more frequently than running the MFCC program.

```
while $success==0$ do
  $i \leftarrow 0$
  $R0 \leftarrow 0$
  while $i<200$ do
    $CPSR\_J \leftarrow 0$
    $CPSR\_J \leftarrow 1$
    $i \leftarrow i+1$
  end while
  if $read(R0) \neq 0$ then
    $success \leftarrow 1$
  end if
end while
```

Figure 12: Trojan trigger code

2) Select the Payload: In the hardware implementation, we select the CPSR_J bit as the trigger input. Since Merlin does not support ISA extensions, this bit has no function. We use R_0 as the attack payload. Once the attack is triggered, the value store in R_0 will be modified. The trigger code is shown in Figure 12. We generate the trigger input by frequently writing 0 and 1 to CPSR_J alternatively. When the Trojan is triggered, it changes the value stored in R_0 from 0 to 1 so that we can observe the change through a register read.

V. Experimental Results

Considering that the MCU is digital logic, and the hardware Trojan is analog circuitry, we use Synopsys CustomSim to run simulation. By declaring the name of the analog top-level cell and the analog netlist, CustomSim will simulate the digital logic via VCS [22], and call the related analog simulator to simulate the analog logic. In this section, we will give the simulation result, and introduce the SoC fabrication.

A. Simulation Results

Figure 13 shows the simulation result of the A2 analog circuit. The frequency of the processor is 150MHz. We choose the CPSR_J as the trigger input. The toggling frequency of the trigger input signal is 20MHz. The trigger time is 9 µs. It means that the analog Trojan is activated after 180 toggling events of the trigger input. This result demonstrates that the analog hardware Trojan works in the ARM processor.

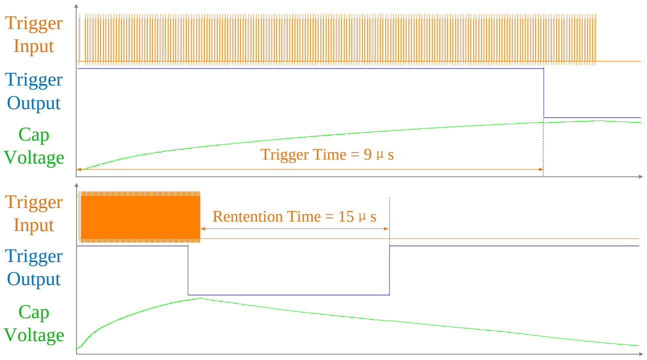

Figure 13: A2 Trojan simulation result

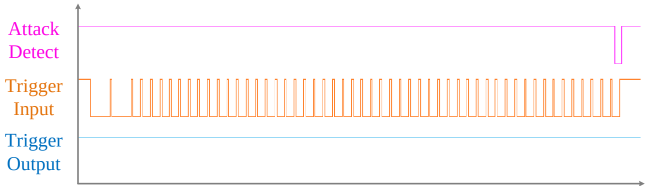

Figure 14: R2D2 detection circuit simulation result

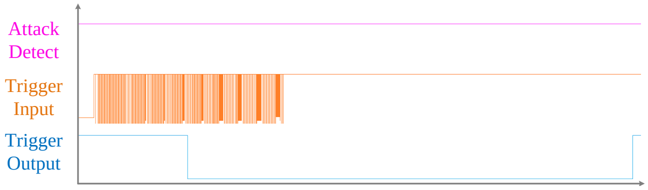

Figure 15: A2 attack simulation without detection

Figure 14 shows the simulation result of the R2D2 detection circuit. The detection circuit guards the CPSR_J register. We set the attack threshold as 64, and the monitoring timing window is 256 clock cycles. From Figure 14, we can see that the attack-detected signal generates a low level pulse, after several toggling events of the trigger input. No trigger output is generated. It means that the Trojan is detected and the attack is prevented. The simulation result when we turn the detection circuits off is shown in Figure 15. The attack-detected signal remains 1. Trigger output is generated after several number of toggling events of the trigger input. The result shows that the R2D2 detection method is effective in detecting the Trojan.

B. Fabrication

The demonstration SoC is fabricated using the SMIC 130 nm Mixed-Signal 1P7M process. The layout of the MCU and the analog hardware Trojan is shown in Figure 16. Figure 17 shows the photo of the SoC silicon die. In the SoC, the MCU is integrated with 16 KB on-chip L1 program Cache, and 32 KB dual-port data SRAM. ROM is used to store the boot

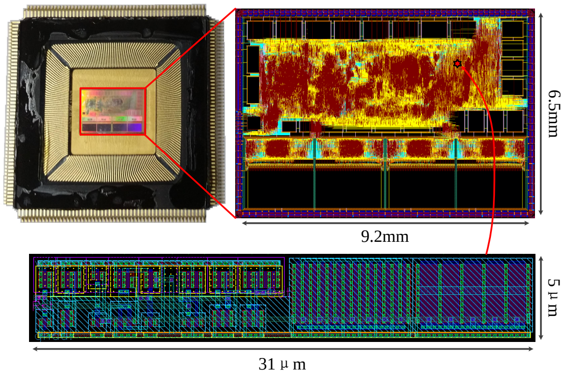

Figure 16: The SoC chip layout

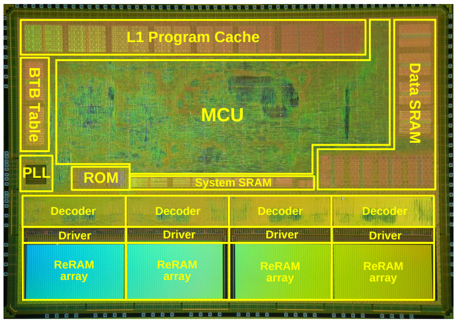

Figure 17: Photo of SoC silicon die

loader. It also embraces 4Mb embedded ReRAM. As shown in Figure 16, the chip area is 9.2 mm by 6.5 mm. The area of the analog hardware Trojan is 31 μm by 5 μm. Trojan-to-circuit ratio is 2.6×10^{-6}.

The detection circuitry is also included in the fabricated SoC, and it is guarding the CPSR_J signal in the MCU. The detection circuit is included in the MCU, by automatic place and route, the gates composing the detection circuits are scattered in the layout. The post-layout area of the detection circuit is about 2225 μm^2. Detection-to-circuit ratio is 3.7×10^{-5}. For the detection circuit, the main area consumption comes from the counters. The size of the counters is related to parameters setting. We set T_m=256, A_{th}=64, so the clock counter width is 8, and the toggling event counter width is 6. This includes about 70 gates. The AT register and MTW register include about 12 gates. With these parameters we were able to verify the operation of the Trojan and detection circuitry successfully.

VI. Conclusion and Future Work

In this paper, we implement an analog Trojan in a in-house designed ARM processor. We also propose a runtime Trojan detection method. The method targets Trojans triggered by toggling events, overcoming a significant limitation of existing Trojan detection schemes in detecting A2-alike Trojans. This method is proved to be effective in detecting an analog Trojan inserted in the ARM processor. The chip is also fabricated using SMIC 130 nm Mixed-Signal 1P7M process. We intend to continue this research direction by exploring topics such as optimal parameter tuning, post-fabrication configuration using ReRAMs, and split-manufacturing.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China under Grant No. 61502032, and by Tsinghua and Samsung Joint Laboratory.

References

[1] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog malicious hardware," in *Security Privacy*, 2016, pp. 18–37.

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

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

[4] S. Narasimhan, D. Du, R. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia, "Hardware Trojan detection by multiple-parameter side-channel analysis," *IEEE Transactions on Computers*, vol. 62, no. 11, pp. 2183–2195, 2013.

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

[6] S. Saha, R. S. Chakraborty, S. S. Nuthakki, D. Mukhopadhyay *et al.*, "Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 577–596.

[7] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware trojan detection and reducing trojan activation time," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 112–125, 2012.

[8] S. Kelly, X. Zhang, M. Tehranipoor, and A. Ferraiuolo, "Detecting hardware trojans using on-chip sensors in an asic design," *Journal of Electronic Testing*, vol. 31, no. 1, pp. 11–26, 2015.

[9] ARM, "ARM information center," 2017. [Online]. Available: http://infocenter.arm.com

[10] Y. Hou, H. He, X. Yang, D. Guo, X. Wang, J. Fu, and K. Qiu, "Fumicro: A fused microarchitecture design integrating in-order superscalar and vliw," *VLSI Design,2016,(2016-12-15)*, 2016.

[11] N. Binkert, B. Beckmann, G. Black, A. Saidi, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, and S. Sardashti, "The gem5 simulator," *Acm Sigarch Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[12] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2012.

[13] C. Villavieja, J. A. Joao, R. Miftakhutdinov, and Y. N. Patt, "Yoga: A hybrid dynamic VLIW/OoO processor," 2014.

[14] C. Fallin, C. Wilkerson, and O. Mutlu, "The heterogeneous block architecture," in *IEEE International Conference on Computer Design*, 2014, pp. 386–393.

[15] Khubaib, M. A. Suleman, M. Hashemi, W. Chris, and Y. N. Patt, "Morphcore: An energy-efficient microarchitecture for high performance ILP and high throughput TLP," in *Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 305–316.

[16] Z. Shen, H. He, X. Yang, D. Jia, and Y. Sun, "Architecture design of a variable length instruction set VLIW DSP," *Tsinghua Science & Technology*, vol. 14, no. 5, pp. 561–569, 2009.

[17] J. K. F. Lee, "Analysis of branch prediction strategies and branch target buffer design," *Computer*, vol. 17, no. 1, pp. 6–22, 1984.

[18] J. E. Smith, "A study of branch prediction strategies," *Proceedings of the 8th annual symposium on Computer Architecture*, vol. 29, no. 6, pp. 135–148, 1981.

[19] J. Hoogerbrugge, "Dynamic branch prediction for a vliw processor," in *International Conference on Parallel Architectures & Compilation Techniques*, 2000, pp. 207–214.

[20] G. Palermo, M. Sam, C. Silvan, V. Zaccari, and R. Zafalo, "Branch prediction techniques for low-power vliw processors," in *ACM Great Lakes Symposium on Vlsi 2003, Washington, Dc, Usa, April*, 2003, pp. 225–228.

[21] S. Furber, "ARM system-on-chip architecture," *Network IEEE*, vol. 14, no. 6, p. 4, 2000.

[22] G. Nunn, F. Delguste, A. Khan, A. Verma, and B. Geden, "White paper using digital verification techniques on mixed-signal socs with customsim and vcs," *Synopsys, Tech. Rep.*, 2011.