Communication Architecture Under Siege: An In-depth Analysis of Fault Attack Vulnerabilities and Countermeasures

Hongwei Zhao, Vianney Lapôtre, Guy Gogniat

Communication Architecture Under Siege: An In-depth Analysis of Fault Attack Vulnerabilities and Countermeasures

Hongwei Zhao
Université Bretagne Sud
UMR 6285, Lab-STICC
Lorient, France
Email: hongwei.zhao@univ-ubs.fr

Vianney Lapôtre
Université Bretagne Sud
UMR 6285, Lab-STICC
Lorient, France
Email: vianney.lapotre@univ-ubs.fr

Guy Gogniat
Université Bretagne Sud
UMR 6285, Lab-STICC
Lorient, France
Email: guy.gogniat@univ-ubs.fr

***Abstract*—Fault attacks aim to disturb integrated circuits using physical methods to break a security system or steal information. Nowadays, a particular attention has been paid to fault attacks on SoCs (System-on-Chip). A SoC is made up of numerous IPs which are connected to each other by a communication architecture. This work focuses on a RISC-V based SoC with a wishbone bus. It aims to study the security of the bus against fault attacks. Using a simulation environment, an automate-attack script, generated by Python, targets all the registers within the bus to identify all the vulnerabilities in the communication architecture. After integrating some countermeasures, the resistance to several fault models of the extended communication structure is measured and discussed.**

I. Introduction

Modern integrated circuits find applications in diverse fields, including security, health, and transportation. However, fault attacks—where internal signals of integrated circuits are physically manipulated to leak data or break cryptographic property—pose a significant threat across these domains [1][2].

Fault attacks can be implemented using various methods, including lasers [3], clock glitches [4], and electromagnetic pulses [5]. At the RTL (Register Transfer Level), these attacks manifest as bit-flips [6] or bit set/reset operations [7]. At the instruction level, they involve actions such as instruction skipping [8], instruction substitution [9], or combinations thereof.

Most of the existing papers discuss attacks on the processor, for example D. Karaklajić et al. [2] list fault attacks methods targeting a processor, B. Yuce et al. [10] talk about countermeasure FAME to defend embedded software against fault attacks as an extension in processor. On the other hand, the interconnect network has not been studied much in terms of logical and physical attacks, even though it is a prime target for an attack against a SoC. Therefore, this topic is the focus of this work.

Sergei Skorobogatov investigates fault attack on NVM memory modules integrating error correction codes, changing the security settings by aborting the NVM write operation through an optical or power attack [19]. In contrast, we focus on the bus and change the timing and sequence of memory data transfers to the CPU by corrupting the control signals, which also prevents the detection of vulnerabilities due to the integration of error correction codes on the memory modules. Our work highlights existing vulnerabilities due to control signals in a bus-based communication architecture and evaluates several countermeasures.

Our work focuses on the communication architecture of a `RISC-V` based system. All vulnerabilities have been highlighted by performing fault injections through simulations. To perform this study, a deep analysis of the wishbone communication architecture [11] obtained using the `LiteX` framework [12] has been carried out. Based on this architecture, possible attack vectors have been defined. After that, different countermeasures based on vulnerable registers have been integrated and tested under different fault models. The steps of our study are as follows:

1) Build the SoC system;
2) Analyze the SoC structure;
3) Automate fault injection and identify vulnerabilities;
4) Integrate countermeasures and explore faults.

The remainder of the paper is organized as follows. In Section II, we describe the configuration of the environment for an automated fault injection campaign. In Section III, we explore the vulnerabilities of two important attacks and how they propagate within the communication architecture. In Section IV, we integrate countermeasures in the bus and discuss their resistance to different fault models, and in Section V, we discuss the results of an attack-change memory that raises some interesting questions and we mathematically prove that a double-bit attack with a duplication is better than triplication. Finally, in Section VI, we conclude and discuss future research directions.

II. Environment configuration

We developed a SoC using the `LiteX` software framework. `LiteX` is an open-source tool that offers a streamlined and efficient infrastructure for creating SoCs for FPGA targets. It enables exploration of diverse digital design architectures and facilitates the development of complete FPGA-based systems.

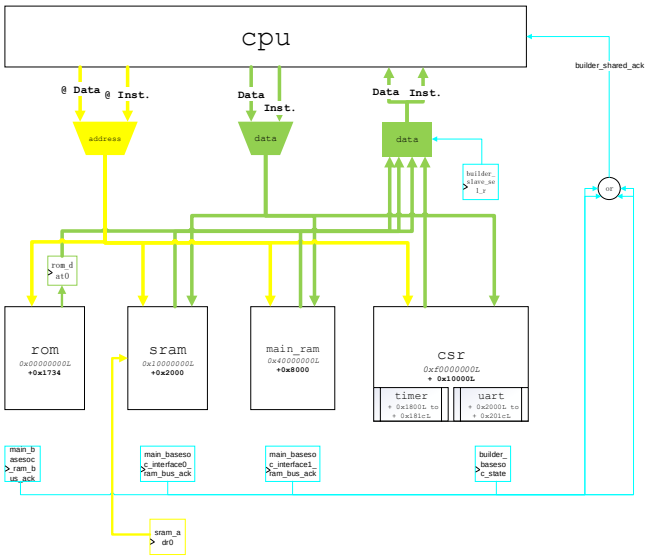

Fig. 1. Wishbone communication architecture

Specifically, we configured our architecture to align with the `Digilent Basys 3` board. For ease of analysis, we opted for the `VexRiscv` processor and the wishbone bus, which represents a straightforward choice.

Figure 1 shows a simplified representation of the wishbone communication architecture within a SoC composed of a processor and some peripherals. Through the bus, the CPU and the memory modules such as ROM, RAM and CSR exchange data signals (green lines), address signals (yellow lines), and control signals (blue lines). One of the control signals, `builder_slave_sel_r` governs memory reads by the CPU through a multiplexer formed by combinational logic gates. The other control signal, `builder_shared_ack` interacts with the CPU to manage the timing of memory reads.

After analyzing all the signals on the bus, we decided to investigate potential vulnerability. A combinatorial logic injection on the bus can be equated to a register injection associated with it. As a result, the attack targets every bit of all register signals on the bus. We employed a 1-bit-flip as the fault model because it covers both bit set/reset cases and allows for straightforward checks of bus vulnerabilities. Our approach involved using a `TCL` script generated by a program in Python[13]. This script was executed in `ModelSim` to automate the injection process. We logged relevant information, including memory state at the end of the simulation, injection time, injected registers, and the resulting outcomes.

To exploit the fault, we selected a software application developed by the FISSC project [14]. This application conducts a comparison between the user input (`g_userPin`) and a stored password (`g_cardPin`) shown in Listing 1. If a match is identified, the state of an authentication variable (`g_authenticated`) is updated to 1 (otherwise it rests in default as 0). We initialize the user input and password to be different, with the authentication variable changing to 1 as the criterion for the success of the attack. There are many versions of FISSC programs, and we chose the version without countermeasure to avoid fault on bus to be corrected. The application is stored in the Read-Only Memory (ROM) and subsequently executed within the Central Processing Unit (CPU). To optimize the experimental process and reduce the number of trials, the window for fault injection is confined to clock cycles during which the instructions associated with a portion of the comparison function (denoted as `verifyPIN` in the diagram) are processed by the CPU.

Listing 1. Unprotected verifyPIN function from FISSC

```
BOOL verifyPIN() {
    g_authenticated = 0;

    if(g_ptc > 0) {
        if(byteArrayCompare(g_userPin,
            g_cardPin, PIN_SIZE) == 1) {
            g_ptc = 3;
            g_authenticated = 1; //
                Authentication();
            return 1;
        } else {
            g_ptc--;
            return 0;
        }
    }

    return 0;
}
```

III. Identified Vulnerabilities

As shown in Figure 2, we find four ways to change the authentication variables: a. Attacking the address signal `sram_adr0` that stores the address of the memory module SRAM; b. Attacking the ROM data transfer register located in the middle of the memory ROM and the data bus `rom_dat0`; c. Attacking the strobe signal `builder_slave_sel_r` (4 bits) with the change of one memory block in SRAM (could be a legal write-in SRAM or an attack on SRAM); d. Attacking the 4 acknowledge registers (each one 1 bit).

Numerous strategies have been proposed to protect data and addresses, enabling the detection of potential attacks [15]. However, it's important to note that encryption or redundancy of data and addresses are not foolproof measures. They are incapable of preventing data substitution resulting from manipulated control signals. Consequently, our attention is primarily directed towards the control signals `ack` and `sel`. These signals are critical in maintaining the integrity of data and addresses, and thus, warrant our focus.

In the subsequent section, we delve into the specifics of how control signal attacks can successfully inject vulnerabilities. The `sel` signal, which is connected to four memory units, operates on combinational logic gates (as depicted in Equation 1). When the CPU accesses the SRAM memory, the `sel`

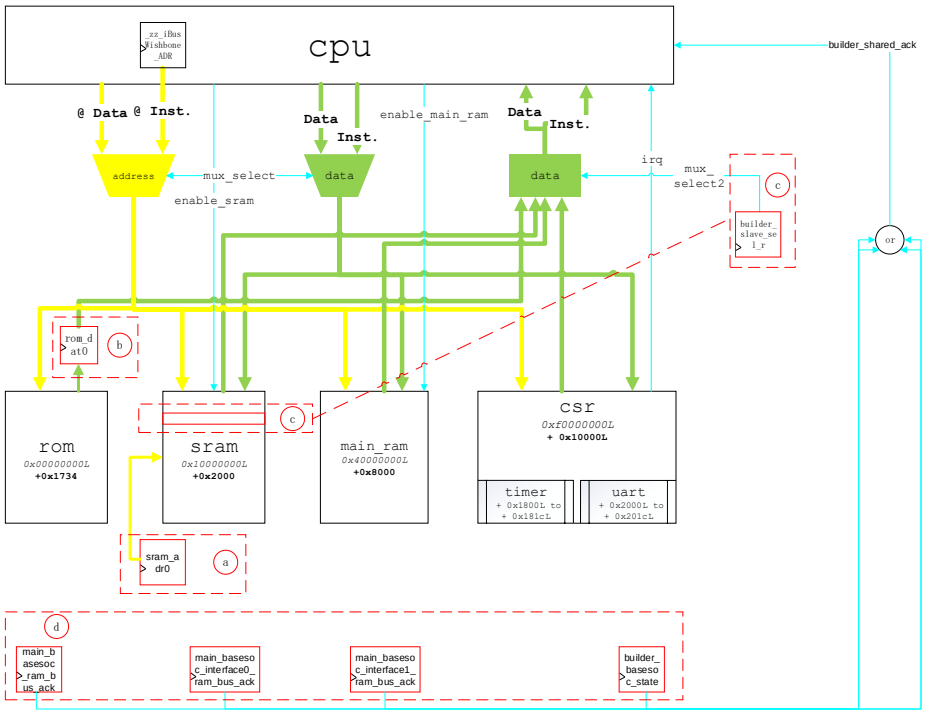

Fig. 2. Vulnerabilities in Wishbone architecture

signal is set to `0010`. If the CPU reads the password value `04030201`, we can alter the `sel` signal to `0000` through bit-flipping, causing the CPU to read the data as `00000000`. This vulnerability model is akin to a multi-bit reset.

$$\begin{aligned}\text{data\_bus} =&(\text{sel}[0] \wedge \text{rom}) \vee (\text{sel}[1] \wedge \text{sram}) \\ &\vee (\text{sel}[2] \wedge \text{main\_ram}) \vee (\text{sel}[3] \wedge \text{csr})\end{aligned} \tag{1}$$

We also observe that if we manipulate the `sel` signal to `0011`, the CPU will read the `or` value in both ROM and SRAM. This manipulation can lead to a successful attack by causing a phase shift in the ROM and SRAM values. If the attacker has the ability to write to certain memory locations or exploit their existing values, and subsequently attacks the `sel` signal to read the sum of the attacker's set values, he can achieve a specific objective. This type of attack is undetectable by methods such as parity checking. This highlights the importance of comprehensive security measures beyond traditional methods.

The `ack` signal is susceptible to a bitflip attack, which could potentially lead to successful authentication. As depicted in Figure 3, the CPU is currently reading the SRAM. Consequently, the `ack` signal of SRAM `main_basesoc_interface0_ram_bus_ack` fluctuates. Meanwhile, the other three `ack` signals remain at `0`. The total `ack` signal, `builder_shared_ack`, is the sum of these four `ack` signals. Therefore, it mirrors the behavior of `main_basesoc_interface0_ram_bus_ack`. When we alter one of the `ack` signals `builder_basesoc_state`, from `0` to `1` within a single cycle when the `main_basesoc_interface0_ram_bus_ack` is in a low level, the total `ack` signal `builder_shared_ack` also changes from `0` to `1` within the same cycle.

This change causes the total `ack` signal `builder_shared_ack` to transition (as shown in Figure 4) from alternating between high and low every cycle to remaining high for two consecutive cycles (indicated by the red and yellow sections) before returning to its normal state (blue section). A high `ack` level increases the address from which the CPU reads data. As a result, the address signal `builder_shared_adr` also increases for two consecutive cycles (red and yellow sections), before reverting to increasing every two cycles (blue section). Simultaneously, the SRAM transfers data to the bus every cycle, causing `builder_shared_dat_r` to change from one cycle (SRAM[3] in red and SRAM[4] in yellow) to two cycles (SRAM[5] in blue).

Due to a two-cycle delay between the acknowledgment signal affecting the data received by the CPU and the data sent to the data bus, the CPU only receives data from the bus that is two cycles before. This data is then written to the data cache under the label `ways_0_data_symbol`. On the data bus, the lifespan of SRAM[3] and SRAM[4] is only one cycle. As a result, the CPU stores SRAM[2] and SRAM[3] in their place. This action causes the code stored in SRAM[4] to be replaced with `00000000`, which is the value of SRAM[3] and coincidentally, the value of the user input. This sequence

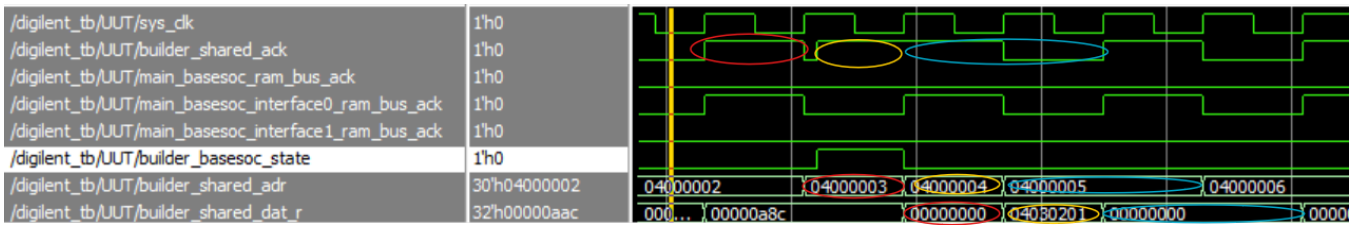

Fig. 3. Bit-flipping attack on the `ack` signal: waveform of data, bus address and `ack` signal

of events leads to a successful attack.

In summary, the effect of the attack can be understood in terms of control signals. The `sel` signal in the attack is equivalent to a multi-bit reset of data or the mixture of multiple data. The `ack` signal in the attack is associated with the repetition and replacement of instructions.

IV. Countermeasures

Upon identifying these vulnerabilities, we implemented countermeasures for the 4-bit `sel` register and the four 1-bit `ack` registers. These countermeasures include simple parity, duplication, complimentary duplication, triplication, Hamming code[18], and Single Error Correction Double Error Detection (SECDED). For the multiplexer composed of logic gates associated with the `sel` signal, we first altered the structure to a combination of 4 to 2 bits encoder and multiple selectors. This change allowed us to deploy countermeasures effectively, thereby preventing the data mixing and zero substitution that could occur when the `sel` signal is attacked. In the case of the `ack` signal, we aimed to minimize hardware resource consumption. To achieve this, we applied the countermeasures to the whole 4-bit `ack`. The following are, in order, the original structure (Figure 5), the `ack` signal with duplication deployed (Figure 6), and the `sel` signal with SECDED deployed (Figure 7).

To evaluate the effectiveness of the countermeasures, we automatically inject faults during the simulation phase as before. The target of these injections includes every bit of all registers on the bus, including the new registers introduced by the countermeasures. The timing of these injections coincides with the execution of the instructions corresponding to the compare user input and password function, `verifyPIN`, in the CPU. To simulate faulty injections during the experiment (for instance, 4 laser attacks are possible [16]), we opt for various scenarios. These include a single bit-flip (equivalent to 1 laser spot), a total of 2 bit-flips either on the same register or across two registers (equivalent to 2 laser spots), ”manipulate 1 register” as bit-flips at any number and position on a single register (up to 4 laser spots or 1 electromagnetic attack), and ”manipulate 2 registers” as bit-flips at any number and position across two registers (up to 8 laser spots or 2 electromagnetic attacks).

We categorize the results into six distinct groups. `Crash` occurs when the exception detection signals in the CPU are enabled, or when the emulation time surpasses our estimated maximum; `Detect` includes cases where at least one of the attacked signals is detected; `Correct` refers to instances where all attacked signals are corrected; `Success` includes cases where there are attacked signals that are neither detected nor corrected, and the authentication signal is changed to success; `Silence` refers to instances where an attacked signal is not detected or corrected, and no changes are made to the memory; `Change` includes cases where an attacked signal is not detected or corrected, and some changes are made to the memory.

We utilized automated fault injection to test the robustness of various countermeasures, whose structures were synthesized using `Vivado`. The statistical results, presented in Table I, led us to several conclusions. Firstly, an attack targeting multiple bits of the `ack` or `sel` signal, or both the signal and its replica (such as its complement or parity), can still result in successful authentication, indicating a successful attack. Secondly, even without impacting the authentication signal, an attack on the `ack` or `sel` signal can alter the memory content. Thirdly, the deployment of countermeasures, despite enlarging the attack surface due to the involvement of more logic, effectively reduces the success rate of an attack. Lastly, correction strategies, while more resource-intensive than detection, are more vulnerable to successful attacks. These findings underscore the complexity of securing digital systems and the trade-offs involved in implementing countermeasures.

V. Discussion

In this section we are interested in the case of `change`. By reproducing the injections corresponding to the `change` case, we found that the cause is still an attack on the `ack` and `sel` registers, which causes the CPU to read incorrect data and then write the incorrect data to SRAM, resulting in a small amount of memory changes; or the cause is an attack on the `ack` and `sel` registers, which directly results in a small amount of memory changes. There are other cases where the CPU reads the wrong instruction and the memory module is not written, resulting in a large amount of memory remaining at the initial value.

Another interesting item is that duplication resists 2-bit flip-flop attacks better than triplication, which we can prove mathematically. Assuming that the protected register is `n` bits, which can be attacked by 1 bit, and the other registers without weaknesses are `x` bits, the total number of bits after duplication is `2n+x`, the total number of bits after triplication is `3n+x`,

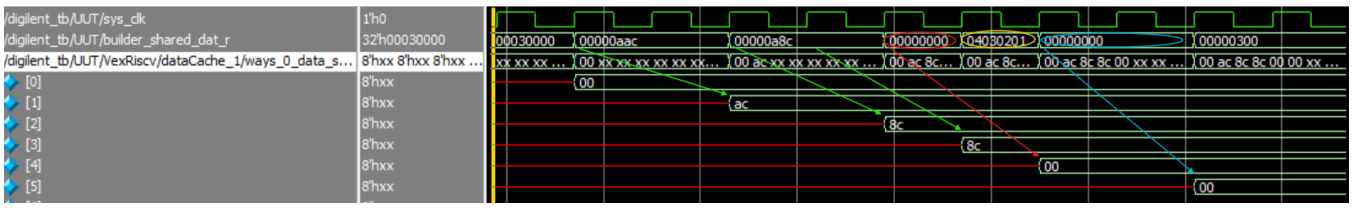

Fig. 4. Effect of the total `ack` signal `builder_shared_ack` to transition from alternating between high and low every cycle to remaining high for two consecutive cycles (indicated by the red and yellow sections) before returning to its normal state (blue section).

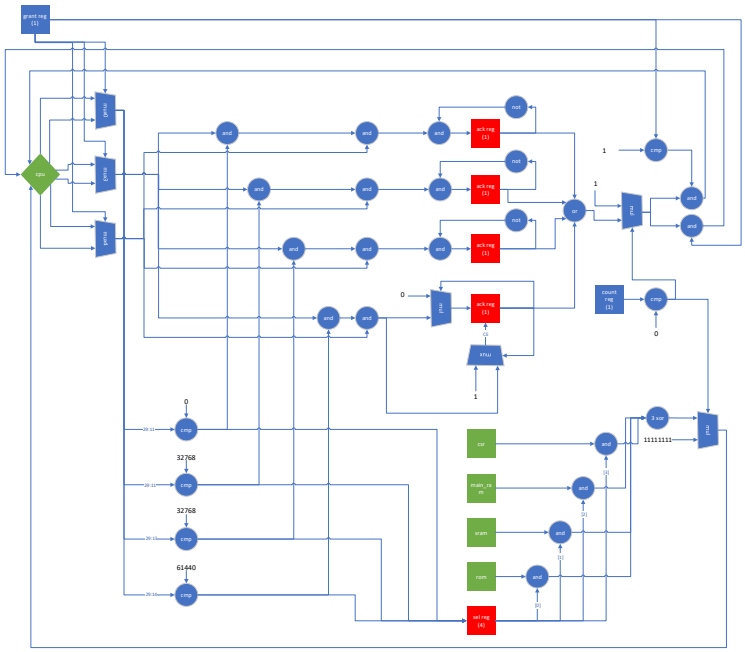

Fig. 5. Original structure of the wishbone bus

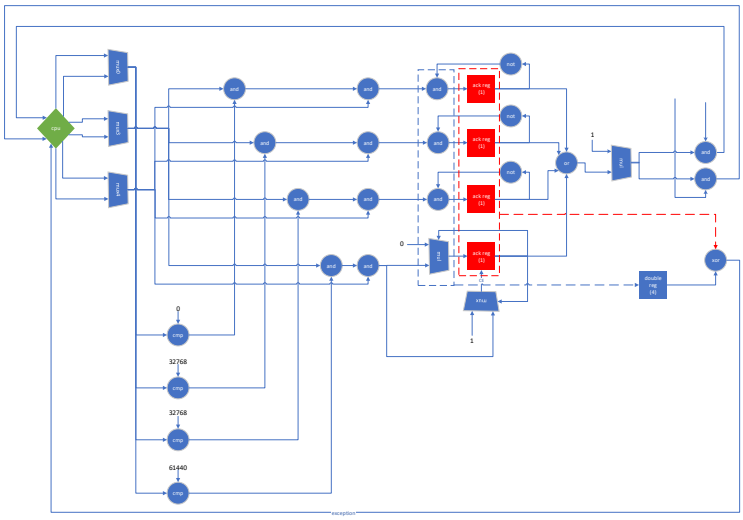

Fig. 6. Duplication of `ack` signal

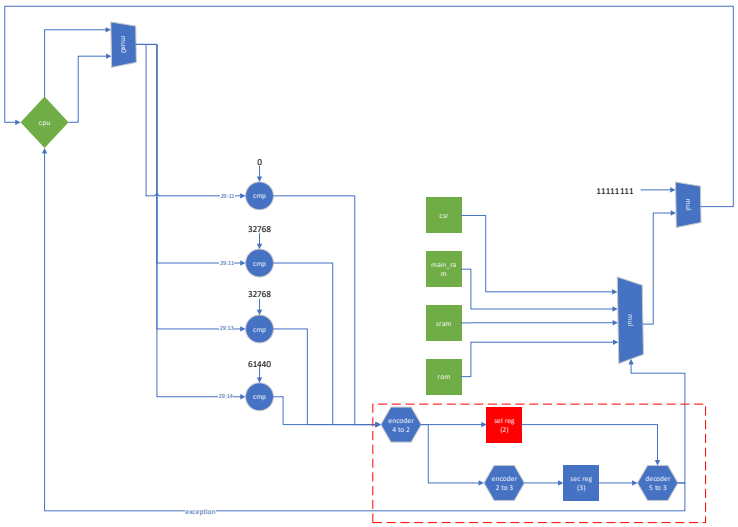

Fig. 7. SECDED of `sel` signal

the probability of success after randomly attacking the 2 bits respectively is:

$$P(duplication) = \frac{n}{\binom{2n+x}{2}} = \frac{2n}{(2n+x)(2n+x-1)}$$

$$P(triplication) = \frac{3n}{\binom{3n+x}{2}} = \frac{6n}{(3n+x)(3n+x-1)}$$

$$P(triplication) - P(duplication) = \frac{6n}{(3n+x)(3n+x-1)} - \frac{2n}{(2n+x)(2n+x-1)}$$

$$= \frac{(2n)(3n^2 - 3n + 2x^2 - 2x + 6nx)}{(3n+x)(3n+x-1)(2n+x)(2n+x-1)}$$

Since `n` and `x` are both numbers greater than 0, the calculation shows that P(triplication) is always greater than P(duplication).

VI. Conclusion

In this paper, we constructed a SoC based on the RISC-V processor and a wishbone bus. We analyzed all structures within the SoC bus. We implemented automated fault injection on all registers on the bus, identifying vulnerabilities and investigating their impact on program operation. Unlike the work of Skorobogatov[19], who implements the countermeasure on the memory block vulnerable to attack on control signals as we find, our countermeasures were then deployed on the vulnerable registers on the bus, and the resilience of different countermeasures was tested and analyzed using different fault models. We also analyzed changes in memory under special circumstances and performed theoretical calculations on the resistance of duplication and triplication to 2-bit attacks.

For future work, we plan to test programs that have already deployed countermeasures to compare the effectiveness of software and hardware countermeasures. We will also switch to other well-known programs to explore whether the discovered vulnerabilities differ. We will choose protocols that have deployed peer-to-peer self-verification, such as `AXI` [17], and attack them to obtain an overall vulnerability of the bus. Finally, we will provide a guide for designers on how to avoid vulnerabilities in the design of communication architecture at the RTL level. This guide will serve as a reference to prevent easy attacks on the circuits.

Acknowledgment

This work has been supported by a PhD grant from the DGA Creach Labs research program and the ARED Brittany Region program.

References

[1] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," in Proceedings of the IEEE, vol. 94, no. 2, pp. 370-382, Feb. 2006, doi: 10.1109/JPROC.2005.862424.

[2] D. Karaklajić, J. -M. Schmidt and I. Verbauwhede, "Hardware Designer's Guide to Fault Attacks," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 12, pp. 2295-2306, Dec. 2013, doi: 10.1109/TVLSI.2012.2231707.

[3] S. Tajik, H. Lohrke, F. Ganji, J. -P. Seifert and C. Boit, "Laser Fault Attack on Physically Unclonable Functions," 2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), Saint-Malo, France, 2015, pp. 85-96, doi: 10.1109/FDTC.2015.19.

[4] Agoyan M, Dutertre J M, Naccache D, et al. When clocks fail: On critical paths and clock faults[C]//International conference on smart card research and advanced applications. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010: 182-193.

[5] P. Maistri et al., "Electromagnetic analysis and fault injection onto secure circuits," 2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC), Playa del Carmen, Mexico, 2014, pp. 1-6, doi: 10.1109/VLSI-SoC.2014.7004182.

[6] A. Barenghi, L. Breveglieri, I. Koren and D. Naccache, "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures," in Proceedings of the IEEE, vol. 100, no. 11, pp. 3056-3076, Nov. 2012, doi: 10.1109/JPROC.2012.2188769.

[7] Joye, Marc, and Michael Tunstall, eds. Fault analysis in cryptography. Vol. 147. Heidelberg: Springer, 2012.

TABLE I
COUNTERMEASURE ANALYSIS

Countermeasure	Fault model	Crash	Detect	Correct	Success	Silence	Change	Success rate	Resources
Original	bit-flip	46	0	0	10	1394	118	0.0063776	LUT 2225
	manipulate reg	300	0	0	13	4208	183	0.0027636	FF 1793
	2 bit-flip	379	0	0	58	4415	636	0.0105685	WNS 0.001ns
	manipulate 2 reg	3138	0	0	171	24196	2287	0.0057398	Period 14.16ns
Simple parity	bit-flip	79	1489	0	0	0	0	0.0000000	LUT 2216
	manipulate reg	120	1489	0	1	1522	4	0.0003189	FF 1791
	2 bit-flip	668	2395	0	35	2057	333	0.0063776	WNS -0.029ns
	manipulate 2 reg	1633	12205	0	40	6937	353	0.0018896	Period 14.02ns
Duplication	bit-flip	83	2269	0	0	0	0	0.0000000	LUT 2228
	manipulate reg	128	2626	0	0	1950	0	0.0000000	FF 1791
	2 bit-flip	990	10847	0	10	970	119	0.0007730	WNS 0.012ns
	manipulate 2 reg	2613	37527	0	11	9902	123	0.0002192	Period 14.23ns
Complementary duplication	bit-flip	83	2269	0	0	0	0	0.0000000	LUT 2234
	manipulate reg	128	2626	0	0	1950	0	0.0000000	FF 1791
	2 bit-flip	990	10847	0	10	970	119	0.0007730	WNS -0.015ns
	manipulate 2 reg	2613	37527	0	11	9902	123	0.0002192	Period 14.22ns
Hamming code	bit-flip	12	0	2340	0	0	0	0.0000000	LUT 2238
	manipulate reg	132	0	2340	11	2882	123	0.0020044	FF 1794
	2 bit-flip	388	0	6825	74	4940	709	0.0057205	WNS 0.002ns
	manipulate 2 reg	3085	0	6825	275	48013	2943	0.0044978	Period 14.20ns
Triplication	bit-flip	18	0	3510	0	0	0	0.0000000	LUT 2220
	manipulate reg	60	0	10530	0	1170	0	0.0000000	FF 1791
	2 bit-flip	366	8547	18135	30	2910	0	0.0010004	WNS 0.001ns
	manipulate 2 reg	1737	149007	110565	183	11340	0	0.0006707	Period 14.27ns
SECDED code	bit-flip	14	0	2340	0	0	0	0.0000000	LUT 2203
	manipulate reg	115	3315	2340	10	2606	119	0.0011758	FF 1789
	2 bit-flip	91	10920	6825	0	0	0	0.0000000	WNS -0.005ns
	manipulate 2 reg	3415	78594	6825	239	42087	2628	0.0017864	Period 14.25ns

[8] Barbu, G., Thiebeauld, H., Guerin, V. (2010). Attacks on Java Card 3.0 Combining Fault and Logical Attacks. In: Gollmann, D., Lanet, JL., Iguchi-Cartigny, J. (eds) Smart Card Research and Advanced Application. CARDIS 2010. Lecture Notes in Computer Science, vol 6035. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-12510-2_11

[9] N. Timmers, A. Spruyt and M. Witteman, "Controlling PC on ARM Using Fault Injection," 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), Santa Barbara, CA, USA, 2016, pp. 25-35, doi: 10.1109/FDTC.2016.18.

[10] Yuce B, Ghalaty N F, Deshpande C, et al. FAME: Fault-attack aware microprocessor extensions for hardware fault detection and software fault response[M]//Proceedings of the Hardware and Architectural Support for Security and Privacy 2016. 2016: 1-8.

[11] M. Mitić and M. Stojčv, "A Survey of Three System-on-Chip Buses: AMBA, CoreConnect and Wishbone," ICEST 2006.

[12] "Litex." https://github.com/enjoy-digital/litex. cessed: 2018-14-01].

[13] William PENSEC, "FISSA – Fault Injection Simulation for Security Assessment", 2024. https://github.com/WilliamPsc/FISSA

[14] L. Dureuil, G. Petiot, M.-L. Potet, T.-H. Le, A. Crohen, Ph. de Choudens. FISSC: a Fault Injection and Simulation Secure Collection. In International Conference on Computer Safety, Reliability and Security (SAFECOMP), 2016.

[15] Patrick C, Yuce B, Ghalaty N F, et al. Lightweight fault attack resistance in software using intra-instruction redundancy[C]//Selected Areas in Cryptography–SAC 2016: 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers 23. Springer International Publishing, 2017: 231-244.

[16] Colombier B, Grandamme P, Vernay J, et al. Multi-spot laser fault injection setup: New possibilities for fault injection attacks[C]//International Conference on Smart Card Research and Advanced Applications. Cham: Springer International Publishing, 2021: 151-166.

[17] AMD. Performance AXI Traffic Generator LogiCORE IP Product Guide. 2023.

[18] William Pensec, Francesco Regazzoni, Vianney Lapotre, Gogniat Guy. Defending the Citadel: Fault Injection Attacks against Dynamic Information Flow Tracking and Related Countermeasures. 2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Jul 2024, Knoxville, United States.

[19] S. Skorobogatov, "Compromising device security via NVM controller vulnerability," 2020 IEEE Physical Assurance and Inspection of Electronics (PAINE), Washington, DC, USA, 2020, pp. 1-6, doi: 10.1109/PAINE49178.2020.9337736.