Analysis of vulnerabilities in communication architectures in systems-on-chip with regard to fault attacks and countermeasure propositions for trusted systems

PhD Defense

Hongwei ZHAO

Université Bretagne Sud, UMR 6285, Lab-STICC, Lorient, France

December 2, 2025

Composition of the Jury	
President of the jury:	Lionel TORRES
Reviewers:	Guillaume BOUFFARD
	Jean-Max DUTERTRE
Examiners:	Karine HEYDEMANN
PhD supervisor:	Guy GOGNIAT
PhD co-supervisor:	Vianney LAPÔTRE

Embedded Systems

- Wide range of applications
- Fast growing market
- Increasingly vulnerable to multiple threats

Figure 1: Embedded systems (from [1])

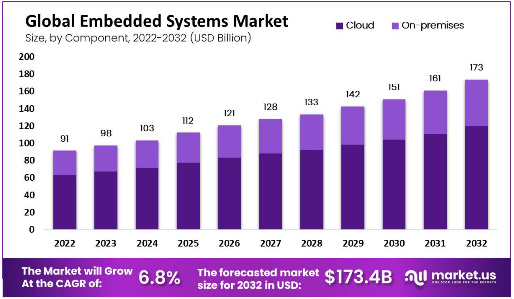


Figure 2: Embedded systems market trend (from [2])

Context: Embedded Systems Under Attack

Threats

- Software attacks: memory overflow [3], buffer overflow [4], control hijacking, etc.
- Hardware attacks: Reverse Engineering [5], Side-Channel Attacks [6], Fault Injection Attacks [7]

Figure 3: Possible methods of attacks on embedded systems [8]

SoC

- Major realization of embedded systems, also vulnerable to fault attacks [9]
- Build with different parts, bus as connection and control

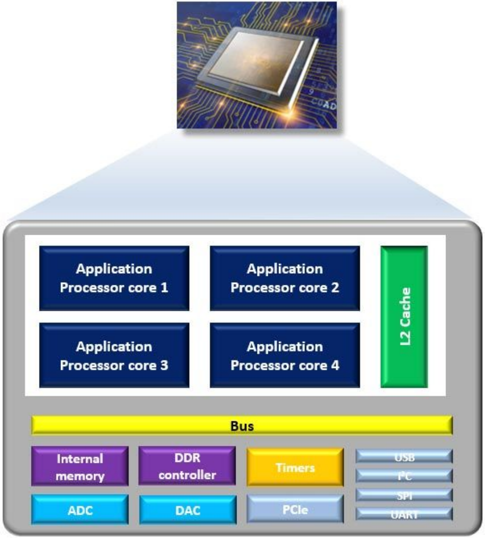


Figure 4: SoC architecture [10]

Communication architecture under fault injection

- Fault injection attacks: physical modification of data(laser, EM, tension ...)
- Attack on CPU and memory can be solved by integrity mechanisms [11]
- Bus can control the data transfer, vulnerable under fault injection attacks [12]

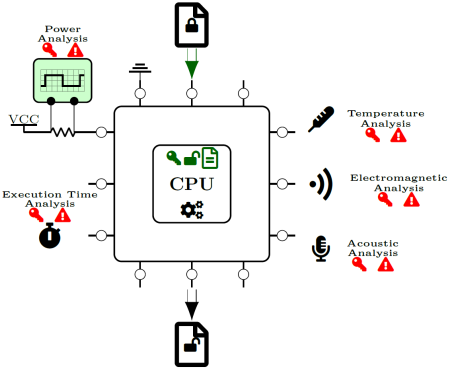


Figure 5: Different injection methods [13]

Vulnerabilities into critical bus

- Buses are fundamental to SoC functionality
- Their standardized and predictable behavior makes them attractive fault-injection targets.

Research challenge

- Identifying vulnerabilities across different protocols and handshake semantics
- Analyzing realistic fault effects at the RTL level within complex interconnects
- Designing countermeasures that remain effective without incurring prohibitive hardware overhead

Objectives of this PhD Thesis

- Analyze the behavior of Wishbone, AXI-Lite, and AXI under fault injection attacks
- Evaluate existing hardware and software countermeasures to determine their practical limitations
- Propose and validate a hardware-integrated countermeasure that can detect bus-level faults

I. Experimental Setup

LiteX Framework

- Open-source SoC builder for FPGA-based systems [14].
- Modular design with Python-based HDL (Migen).
- Supports several processor architectures (e.g. RISC-V).
- No built-in security → interesting to explore vulnerability analysis in this functional-oriented architecture.

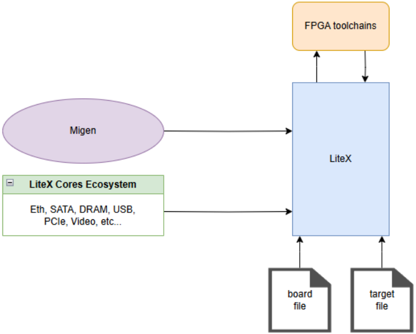


Figure 6: LiteX framework components

SoC Architecture Overview

- CPU: VexRiscv (RISC-V ISA).
- Interconnect protocols: Wishbone, AXI-Lite, AXI.
- Memory regions: ROM, SRAM, CSR, MAIN_RAM.
- Target FPGA: Digilent Basys3 (Artix-7).

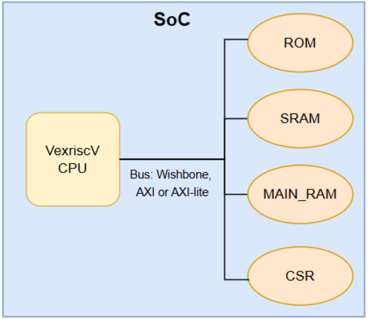


Figure 7: Our SoC architecture

VerifyPin Benchmark

- Written in C [15].
- Simulates a PIN verification process.
- Suite of 8 implementations: V0 (unprotected) + V1–V7 (protected).

VerifyPin V0 Example

- Compares user PIN ("0000") with card PIN ("4321").
- Fault success: g_authenticated set to 1 despite mismatch.

```c
BOOL byteArrayCompare(UBYTE* a1, UBYTE* a2, UBYTE
        size){
    int i;
    for(i = 0; i < size; i++)
        if(a1[i] != a2[i]) return 0;
    return 1;
}
BOOL verifyPIN() {
    g_authenticated = 0;
    if(g_ptc > 0) {
        if(byteArrayCompare(g_userPin, g_cardPin,
                PIN_SIZE) == 1)
            g_ptc = 3;
            g_authenticated = 1;
            return 1;
        else
            g_ptc--;
            return 0;
    return 0;
    }
}
```

Figure 8: C code of VerifyPin function in benchmark V0

Countermeasures implemented in V1 to V7

- HB: Hardened Boolean
- FTL: Fixed-Time Loop
- INL: Inlined Function
- DPTC/PTCBK: Token Counter decremented first/Back up
- LC: Loop Counter
- DC/DT: Double Call/Test
- SC: Step Counter

VerifyPin V1 Example

- Implement with Hardened Boolean.
- Replace 1 and 0 with BOOL_True(0xAA) and BOOL_False(0x55).
- Detect fault => execute countermeasure()

```c
BOOL byteArrayCompare(UBYTE* a1, UBYTE* a2, UBYTE size)
{
    int i;
    for(i = 0; i < size; i++) {
        if(a1[i] != a2[i]) {
            return BOOL_FALSE;
        }
    }
    return BOOL_TRUE;
}

BOOL verifyPIN() {
    int comp;
    g_authenticated = BOOL_FALSE;

    if(g_ptc > 0) {
        comp = byteArrayCompare(g_userPin, g_cardPin, PIN_SIZE);
        if(comp == BOOL_TRUE) {
            g_ptc = 3;
            g_authenticated = BOOL_TRUE; // Authentication();
            return BOOL_TRUE;
        } else if(comp == BOOL_FALSE) {
          g_ptc--;
          return BOOL_FALSE;
        } else {
          countermeasure();
        }
    }
    return BOOL_FALSE;
}
```

Figure 9: C code of VerifyPin function in benchmark V1

FISSA Overview

- Python-based tool for fault injection campaigns [13].
- Works with HDL simulators (Questasim, Vivado, Verilator).
- Automates TCL script generation and simulation logging.

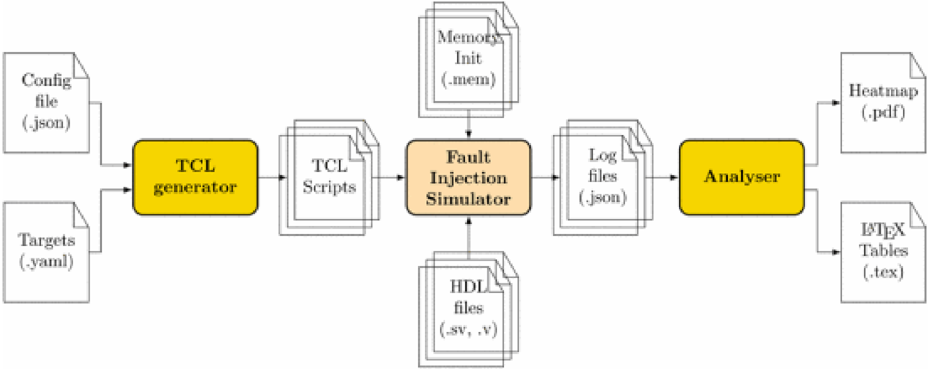


Figure 10: FISSA components

Register lists

- Non-control registers (I/O, LEDs, clock/reset, timers, program data) were excluded due to low injection effectiveness.
- Fault injection was applied to Wishbone, AXI-Lite, and AXI under multiple fault models.

Table 1: All the registers targeted by fault injection in the 3 buses

Bus	Registers
Wishbone	ACK, SEL, done, grant
AXI-Lite	state, selection driver, last_was_read, rr_read_grant, completion flags
AXI	same with AXI-Lite, ax_beat_first, ax_beat_last, last_ar_aw_n, pipe_valid_source

Considered Fault Models

1. Bit-Flip: single bit in register.
2. Manipulate Register: arbitrary bit-flips in one register.
3. 2 Bit-Flips: exactly two bits flipped.
4. Manipulate Two Registers: arbitrary bit-flips in two registers.

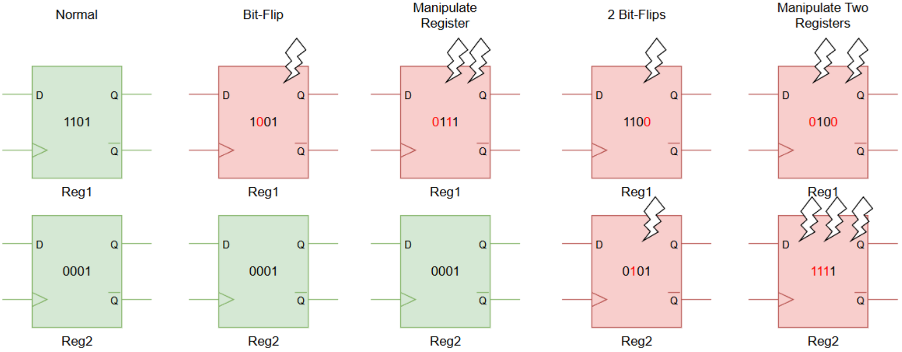


Figure 11: Illustration of our fault models

Outcome Categories

- Crash: simulation terminated.
- Software detection: software countermeasure to detect the fault.
- Hardware detection: hardware countermeasure to detect the fault.
- Hardware correction: hardware countermeasure to correct the fault.
- Success: unauthorized authentication achieved.
- Change: memory state altered.
- Silence: no visible impact.

Fault Injection Campaign

- Attacker has no PIN knowledge.
- Attacker has knowledge of the bus architecture.
- Faults injected at the RTL level via simulation.
- Target: control registers connected to the system bus.

II. Vulnerabilities Exploitation on Bus

Handshake in Wishbone

- Acknowledge generation: ack register controls acknowledge signal in CPU
- Selection mechanism: selection registers selects memory

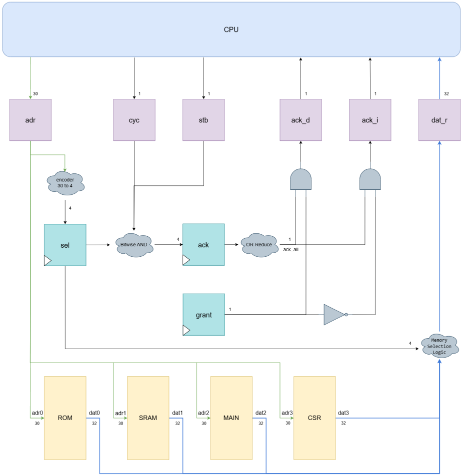


Figure 12: ACK and SEL register connections on Wishbone bus

- Acknowledge generation: state-machine controls acknowledge signal in CPU
- Selection mechanism: state-machine and selection register assign in selection signal to select memory

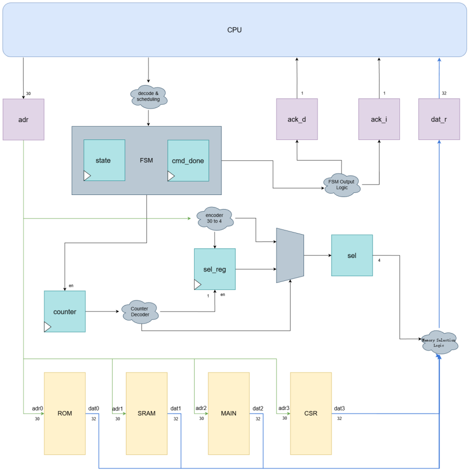


Figure 13: Connection of handshake and selection signals in AXI and AXI-Lite bus protocols

Fault Injection Result

Table 2: Fault injection results for each bus and fault model in V0

bus	fault model	crash	success	change	silence
Wishbone	Bit-Flip	47	37	124	2366
	Manipulate Register	320	43	161	6496
	2 Bit-Flips	547	272	914	11137
	Manipulate Two Registers	5178	778	2891	63225
AXI-Lite	Bit-Flip	282	4	1913	11577
	Manipulate Register	391	6	2589	29942
	2 Bit-Flips	10725	153	69811	194831
	Manipulate Two Registers	36215	549	224347	1236105
AXI	Bit-Flip	158	4	4833	34269
	Manipulate Register	435	6	6723	90996
	2 Bit-Flips	14760	333	428382	1421565
	Manipulate Two Registers	104655	1328	1514455	9762850

Analysis on Wishbone

- ACK register is the most frequently targeted in successful attacks across all fault models.

Table 3: Distribution of successful register combination attacks under different fault models on the Wishbone Bus

Fault model	ack	sel	ack & grant	ack & sel
Bit-flip	94.59%	5.41%	-	-
Manipulate Register	81.40%	18.60%	-	-
2 Bit-Flips	94.12%	3.68%	1.10%	1.10%
Manipulate Two Registers	85.73%	12.34%	0.38%	1.55%

- State register is the primary target for successful attacks across all fault models.

Table 4: Distribution of successful register combination attacks under different fault models on the AXI-Lite Bus

Fault model	state	selection driver & state	completion flag & state
Bit-flip	100.00%	-	-
Manipulate Register	100.00%	-	-
2 Bit-Flips	98.02%	1.32%	0.66%
Manipulate Two Registers	98.36%	1.46%	0.18%

Fault effect

	Wishbone	AXI-Lite	AXI
Bit-flip	instruction skip data reset data multiread	data reset	data reset data misread
Manipulate Register	instruction skip data reset data misread data multiread	data reset data misread	data reset data misread
2 Bit-Flips	instruction skip data reset data misread data multiread	instruction skip data reset data misread data multiread	data reset data misread
Manipulate Two Registers	instruction skip data reset data misread data multiread	data reset instruction skip data misread data multiread	data reset data misread

Wishbone: Success Distribution

- Instruction- vs. data-related fault ratios under "Manipulate Two Registers".
- AXI-Lite/AXI: Frequent data resets (to 0) can unintentionally match the user PIN, making data corruption the main attack pathway.

Table 5: Percentage of data or instruction vulnerabilities by manipulate 2 registers attacks across the three bus architectures

	Wishbone	AXI-Lite	AXI
Data related	17.22%	97.27%	100%
Instruction related	82.78%	2.73%	0%

III. Protection Implementation on Wishbone Bus

- `ack`: 4-bit register, controlled by sel register, cyc/stb signal, assign to ack_d and ack_i
- `sel`: 4-bit register, assigned by adr register
- `grant`: arbitration register for ack_d/ack_i, etc

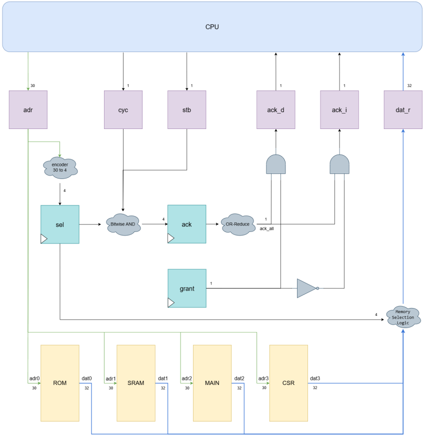


Table 6: Distribution of successful register combination attacks under different fault models on the Wishbone Bus

Fault model	ack	sel	ack & grant	ack & sel
Bit-flip	94.59%	5.41%	-	-
Manipulate Register	81.40%	18.60%	-	-
2 Bit-Flips	94.12%	3.68%	1.10%	1.10%
Manipulate Two Registers	85.73%	12.34%	0.38%	1.55%

Normal behavior:

- `ack1` toggles each cycle while other `ack` signals remain low.
- `ack_d` updates both address and data every two cycles.
- CPU sequentially reads values and commits them to cache.

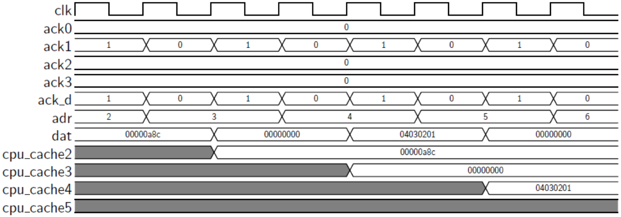


Figure 14: Impact of acknowledge signal on address, data signals and CPU cache without an attack

ACK Under Attack

Faulted behavior:

- A bit-flip on `ack0` disrupts the expected read timing.
- Two-cycle transfers collapse into a single cycle, causing premature cache allocation.
- Cache lines are filled with corrupted values (e.g., PIN replaced by zeros).
- The program compares identical PINs and incorrectly grants access.
- It corresponds to an Instruction skip

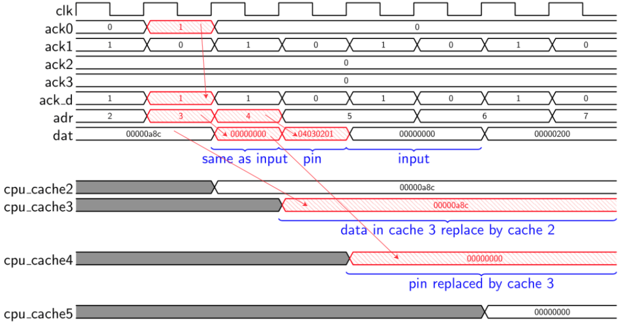


Figure 15: Impact of acknowledge signal on address, data signals and CPU cache with an attack

Countermeasure Deployment

Table 7: Software countermeasures deployed in each benchmark version

	HB	FTL	INL	DPTC	PTCBK	LC	DC	DT	SC
V0									
V1	✓								
V2	✓	✓							
V3	✓	✓	✓						
V4	✓	✓	✓	✓	✓	✓			
V5	✓	✓		✓			✓		
V6	✓	✓	✓	✓				✓	
V7	✓	✓	✓	✓				✓	✓

- 8 versions of the benchmark with/without software countermeasure
- Fault model: Bitflip
- Lizard: analysis NLOC, CCN, token

	crash	detect	success	change	silence	success rate	detect rate	sum	NLOC	CCN	token
V0	47	0	37	124	2366	1.44%	0	2574	32	6	107
V1	61	21	36	136	3266	1.02%	0.60%	3520	36	7	127
V2	98	20	30	164	5078	0.56%	0.37%	5390	44	8	149
V3	58	19	31	96	3789	0.78%	0.48%	3993	32	7	129
V4	92	101	33	104	5478	0.57%	1.74%	5808	47	11	191
V5	105	28	9	153	5238	0.16%	0.51%	5533	42	9	163
V6	61	49	14	108	4377	0.30%	1.06%	4609	38	9	153
V7	105	186	9	252	6708	0.12%	2.56%	7260	77	18	312

Instruction protection:

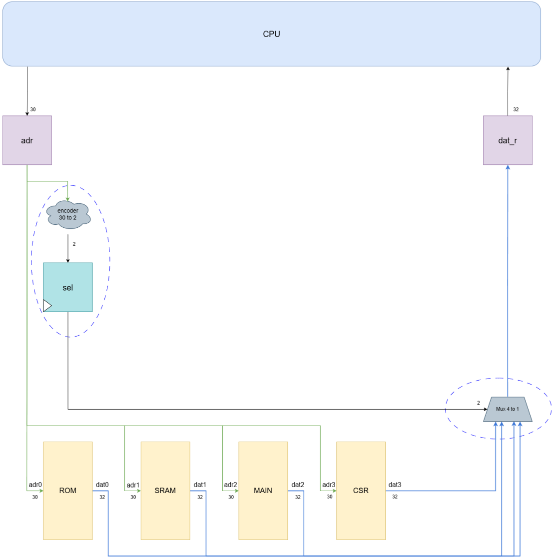
- HB: Prevents register values defaulting to 0/1, mitigating some branch instruction.
- FTL/LC: Fixes/Records loop iteration count, blocking loop manipulation.
- INL: Merges functions can't reduce fault.
- DPTC/PTCBK: State counters not targeted, no effective defense observed.
- DC/DT: Redundant execution neutralizes function-call attacks.
- SC: Detects skipped instructions, effective against instruction-skipping.

Data protection: Only INL (fewer reads) and DC (double read) provide defense.

Table 8: Evaluation of software countermeasures against fault attacks on instructions and data

	HB	FTL	INL	DPTC	PTCBK	LC	DC	DT	SC
Instruction fault	✓	✓	✗	✗	✗	✓	✓	✓	✓
Data fault	✗	✗	✓	✗	✗	✗	✓	✗	✗

- Software-only defenses are limited, Hardware countermeasures are needed.
- Architectural change: combi=> mux, reducing multiple-read attacks.
- Hardware protections on `ack` and `sel` registers.



Hardware Countermeasures Overview

- Simple Parity: Detects faults using a 1-bit parity code.
- Duplication: Creates a duplicate of the registers and compares it with the unprotected version.
- Complementary Duplication: Duplicates the inverse of the registers and compares it with the unprotected version.
- Hamming Code: Corrects the output signal of a register using a 3-bit checksum.
- SECDED Code: Corrects or detects errors using a 4-bit checksum.
- Triplication: Duplicates a register twice, correcting the signal if two registers have matching outputs and detecting errors if all three differ.

Results of Hardware Countermeasure with VerifyPin V0

Countermeasure	Fault model	Success rate	Detection rate	Correction rate
Unprotected	Bit-Flip	1.44%	-	-
	Manipulate Register	0.61%	-	-
	2 Bit-Flips	2.11%	-	-
	Manipulate Two Registers	1.08%	-	-
Simple parity	Bit-Flip	0%	69.54%	-
	Manipulate Register	0.04%	34.77%	-
	2 Bit-Flips	0.89%	64.38%	-
	Manipulate Two Registers	0.27%	50.34%	-
Duplication	Bit-Flip	0%	77.66%	-
	Manipulate Register	0%	44.94%	-
	2 Bit-Flips	0.15%	86.43%	-
	Manipulate Two Registers	0.04%	66.46%	-
Complementary Duplication	Bit-Flip	0%	77.66%	-
	Manipulate Register	0%	44.94%	-
	2 Bit-Flips	0.15%	86.43%	-
	Manipulate Two Registers	0.04%	66.46%	-
Hamming code	Bit-Flip	0%	-	80%
	Manipulate Register	0.49%	-	58.16%
	2 Bit-Flips	0.97%	-	91.30%
	Manipulate Two Registers	1.01%	-	75.95%
Triplication	Bit-Flip	0%	0%	85.71%
	Manipulate Register	0%	0%	81.82%
	2 Bit-Flips	0.23%	20%	76.94%
	Manipulate Two Registers	0.14%	36.74%	58.97%
Secded code	Bit-Flip	0%	11.77%	70.59%
	Manipulate Register	0.32%	34%	37.92%
	2 Bit-Flips	0%	45.56%	52.06%
	Manipulate Two Registers	0.45%	51.94%	36.19%

Resource Overhead Analysis

- Countermeasures increase LUT usage by max. 0.7%.
- Frequency reduced by up to 0.97%.
- Differences largely due to synthesizer auto-optimization.
- Overall: negligible additional hardware resource loss.

Table 9: Hardware resource overhead of each hardware countermeasure (LUT, Flip-Flop, frequency)

Countermeasure	LUT	Flip-Flops	frequency (MHz)
Unprotected	2198	1793	70.13
Simple parity	2214	1791	70.27
Duplication	2201	1791	70.18
Complimentary	2199	1791	70.37
Hamming code	2199	1794	70.32
Triplication	2199	1791	70.27
Secded code	2193	1789	69.44

Comparison of the Protection Effectiveness

- Hardware-only protections achieve consistently lower attack success rates.
- In some fault models, hardware countermeasures reduce success rate to zero.

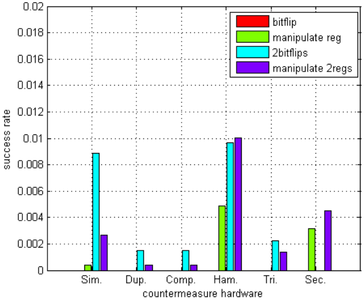


Figure 17: Success rates under the four fault models for the benchmark V0 with different hardware countermeasures

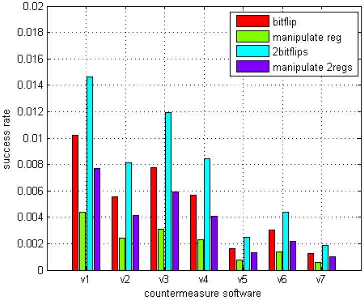


Figure 18: Success rates under the four fault models for the seven benchmark versions with software countermeasures

Comparison of the Protection Effectiveness

- Figure 19: clear improvement with duplication + software vs. duplication alone.
- No hardware/software combination fully neutralizes the Manipulate Two Registers fault model.
- Persistent vulnerability indicates need for more advanced or hybridized protection beyond duplication/redundancy.

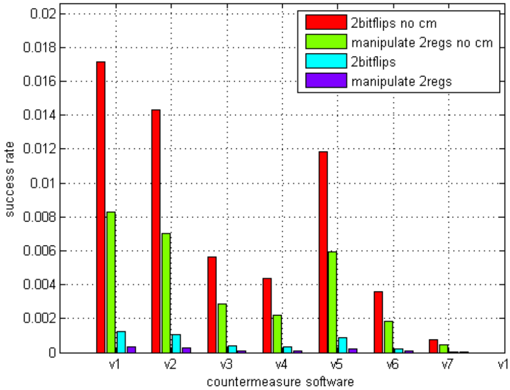


Figure 19: Success rates under the four fault models for the seven benchmark versions with software and/without duplication countermeasures

Proposed Architecture

- Purely hardware-based countermeasure for reliability and efficiency
- Detection-only strategy (higher accuracy, modest correction trade-off)
- Applied to `ack`, `sel`, and `grant` registers

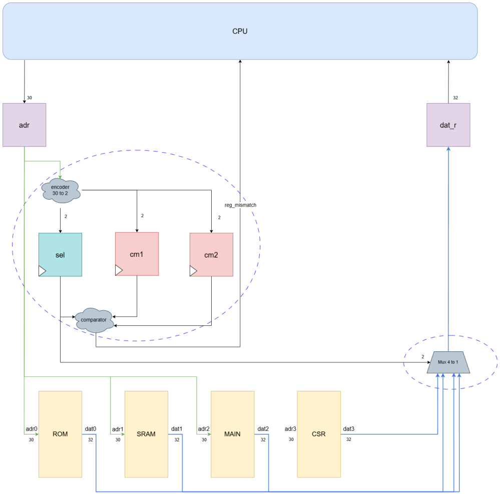


Figure 20: Triplication-redundant detection scheme

Proposed Countermeasure Results

Countermeasure	Fault model	crash	detect_hw	success	change	silence
triplication-redundant v0	Bit-Flip	86	4828	0	0	468
	Manipulate Register	125	5491	0	0	5148
	2 Bit-Flips	2007	56961	0	0	234
	Manipulate Two Registers	5498	174916	0	0	53586
triplication-redundant v1	Bit-Flip	109	6611	0	0	640
	Manipulate Register	161	7519	0	0	7040
	2 Bit-Flips	2572	78068	0	0	320
	Manipulate Two Registers	7106	239614	0	0	73280
triplication-redundant v2	Bit-Flip	170	10120	0	0	980
	Manipulate Register	242	11518	0	0	10780
	2 Bit-Flips	3879	119601	0	0	490
	Manipulate Two Registers	10544	367246	0	0	112210
triplication-redundant v3	Bit-Flip	102	7521	0	0	726
	Manipulate Register	150	8562	0	0	7986
	2 Bit-Flips	2366	89110	0	0	363
	Manipulate Two Registers	6574	273299	0	0	83127
triplication-redundant v4	Bit-Flip	174	10914	0	0	1056
	Manipulate Register	256	12416	0	0	11616
	2 Bit-Flips	4046	129010	0	0	528
	Manipulate Two Registers	11228	395860	0	0	120912
triplication-redundant v5	Bit-Flip	177	10386	0	0	1006
	Manipulate Register	249	11823	0	0	11066
	2 Bit-Flips	3999	122757	0	0	503
	Manipulate Two Registers	11822	375991	0	0	115187
triplication-redundant v6	Bit-Flip	115	8684	0	0	838
	Manipulate Register	169	9887	0	0	9218
	2 Bit-Flips	2684	102904	0	0	419
	Manipulate Two Registers	7424	315625	0	0	95951
triplication-redundant v7	Bit-Flip	200	13660	0	0	1320
	Manipulate Register	295	15545	0	0	14520
	2 Bit-Flips	4683	161637	0	0	660
	Manipulate Two Registers	16041	492819	0	0	151140

Resource Overhead of the Proposed Design

- LUT utilization increases by 0.36%.
- Maximum operating frequency increases by 0.48%.
- Changes attributed to Vivado synthesis/optimization heuristics.
- Overall practicality of the design remains unaffected.

Table 10: Triplication-redundant resource usage

Countermeasure	LUT	Flip-Flops	Frequency (MHz)
Unprotected	2198	1793	70.13
Triplication-redundant	2206	1791	70.47

IV. Conclusion

Key Contributions

- Identified vulnerabilities in SoC buses (Wishbone, AXI-Lite, AXI).
- Developed experiment framework using LiteX + VerifyPin + FISSA.
- Demonstrate limitation of software countermeasure
- Proposed hardware-based countermeasure for the Wishbone bus.
- Demonstrated full protection under evaluated fault models.

Experimental Highlights

- Over 40 million simulations executed on 3 Xeon Gold servers.
- Total campaign duration: more than 2 months.

Global Reflection

- Bus-level vulnerabilities exist across Wishbone, AXI-Lite, AXI.
- Real-world SoCs lack built-in protection for handshake signals.
- Multi-bit fault injection remains a realistic threat with advanced equipment.

Future Directions

- Apply countermeasures to AXI-Lite and AXI buses.
- Combine advanced protections (CFI, runtime checks) against both software/hardware attack.
- Integrate protection into LiteX to add the security dimension.
- Evaluate NoC-based architectures under fault injection attacks.

Publications

International peer-reviewed conferences with proceedings

1. **Hongwei Zhao**, Vianney Lapotre, and Guy Gogniat. "Communication Architecture Under Siege: An In-depth Analysis of Fault Attack Vulnerabilities and Countermeasures." [16] *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, IEEE, 2024. **Published**.
2. **Hongwei Zhao**, Vianney Lapotre, and Guy Gogniat. "Fault Injection in On-Chip Interconnects: A Comparative Study of Wishbone, AXI-Lite, and AXI." *Workshop on Design and Architectures for Signal and Image Processing*. **Minor revision**.
3. **Hongwei Zhao**, Vianney Lapotre, and Guy Gogniat. "Analyzing and Mitigating Wishbone Bus Vulnerabilities in RISC-V SoC Architecture: A Hardware and Software Countermeasures Approach." *IEEE Transactions on Dependable and Secure Computing*. **Rejected, resubmitted**.

Thank you for attention!

References

References

[1] EmmaAshely. *What is an Embedded System?* Accessed: 2025-11-26. Dec. 2022. URL: https://www.rs-online.com/designspark/what-is-an-embedded-system.

[2] *Embedded Systems Market*. Market.us. 2025. URL: https://market.us/report/embedded-systems-market/ (visited on 11/21/2025).

[3] S. Chen et al. "Defeating memory corruption attacks via pointer taintedness detection". In: *2005 International Conference on Dependable Systems and Networks (DSN'05)*. 2005, pp. 378–387. DOI: 10.1109/DSN.2005.36.

[4] C. Cowan et al. "Buffer overflows: attacks and defenses for the vulnerability of the decade". In: *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*. Vol. 2. 2000, 119–129 vol.2. DOI: 10.1109/DISCEX.2000.821514.

[5] Tamas Varady, Ralph R Martin, and Jordan Cox. "Reverse engineering of geometric models—an introduction". In: *Computer-aided design* 29.4 (1997), pp. 255–268.

[6] John Kelsey et al. "Side channel cryptanalysis of product ciphers". In: *Computer Security—ESORICS 98: 5th European Symposium on Research in Computer Security Louvain-la-Neuve, Belgium September 16–18, 1998 Proceedings 5*. Springer. 1998, pp. 97–110.

[7] Mei-Chen Hsueh, Timothy K Tsai, and Ravishankar K Iyer. "Fault injection techniques and tools". In: *Computer* 30.4 (1997), pp. 75–82.

[8] Przemysław Sowa. *Cybersecurity for Embedded Systems*. Somco Software. 2025. URL: https://somcosoftware.com/en/blog/cybersecurity-for-embedded-systems (visited on 11/21/2025).

[9] Thomas Trouchkine. "SoC physical security evaluation". PhD thesis. Université Grenoble Alpes [2020-....], 2021.

References

[10] MathWorks. *SoC Architecture*. MathWorks. 2025. URL: `https://fr.mathworks.com/discovery/soc-architecture.html` (visited on 11/21/2025).

[11] Nathan Burow et al. "Control-flow integrity: Precision, security, and performance". In: *ACM Computing Surveys (CSUR)* 50.1 (2017), pp. 1–33.

[12] Lin Sun and Ping Xu. "Design and implement of RS-485 bus fault injection". In: *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*. IEEE. 2011, pp. 975–980.

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

[14] EnjoyDigital. *LiteX: A Lightweight SoC Builder for FPGA-Based Systems*. Accessed: 2025-11-26. 2025. URL: `https://github.com/enjoy-digital/litex`.

[15] Louis Dureuil et al. "FISSC: A Fwault Injection and Simulation Secure Collection". In: *Computer Safety, Reliability, and Security - 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings*. 2016, pp. 3–11.

[16] Hongwei Zhao, Vianney Lapotre, and Guy Gogniat. "Communication Architecture Under Siege: An In-depth Analysis of Fault Attack Vulnerabilities and Countermeasures". In: *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE. 2024, pp. 890–896.

[17] Nimish Mishra, Anirban Chakraborty, and Debdeep Mukhopadhyay. "Faults in Our Bus: Novel Bus Fault Attack to Break ARM TrustZone". In: *Network and distributed system security symposium*. 2024.

Communication architecture under fault injection

- Exist article talk about EMP injection on the bus, point out the risk of bus [17]
- More global research is needed

Why a Custom SoC?

Motivation

- Vendor-locked SoCs lack transparency and configurability.
- Need precise control over interconnects, memory, and processor interfaces.
- Enable reproducible fault injection experiments.

FISSA Workflow

1. Parse configuration (JSON) and targets (YAML).
2. Generate TCL scripts for fault injection.
3. Run simulations and collect logs.

Configuration file

- path_...: Defines simulator and all required file paths.
- threat_model: Selects the fault model (here: spatial bit-flip)
- avoid_register & avoid_log_registers: Optional exclusion/log lists for registers (unused in our setup)
- target_window: Sets the fault-injection window based on VerifyPIN execution cycles
- cycle_ref: Specifies total cycles to observe the final authentication outcome
- cpu_period & batch_sim: Uses an 8 ns CPU period and groups 4000 simulations per batch

```json
{
    "name_simulator": "modelsim",
    "path_tcl_generation": "C:/Users/zhao/Desktop/FISSA-main/",
    "path_files_sim": "C:/Users/zhao/Desktop/FISSA-main/simu_files/",
    "path_generated_sim": "C:/Users/zhao/Desktop/FISSA-main/simu_files
      /generated_simulations/",
    "path_results_sim": "C:/Users/zhao/Desktop/FISSA-main/simu_files/
      results_simulations/",
    "path_simulation": ["C:/Users/zhao/Desktop/mo_project/", "__code",
       "/", "__code"],
    "threat_model": ["single_bitflip_spatial"],
    "multi_fault_injection": 2,
    "avoid_register": [],
    "avoid_log_registers": [],
    "log_registers": [],
    "target_window": {"total": [[20765, 24037]]},
    "cycle_ref": 3067,
    "cpu_period": 8,
    "batch_sim": {"total": 4000},
    "name_reg_file_ext_wo_protect": "/faulted-reg.yaml"
}
```

Figure 21: config.json file

Register file

- Each entry lists a full hierarchical register name.
- A bit-width value specifies how many bits are injected.
- Example: the signal `builder_axirequestcounter0_full` shown is configured as a 1-bit injection target.

```yaml
TOTAL:
  -
      name: sim:/digilent_tb/UUT/builder_axirequestcounter0_full
      width: 1
  -
      name: sim:/digilent_tb/UUT/builder_axirequestcounter0_empty
      width: 1
  -
      name: sim:/digilent_tb/UUT/builder_axirequestcounter1_full
      width: 1
  -
      name: sim:/digilent_tb/UUT/builder_axirequestcounter1_empty
      width: 1
  -
      name: sim:/digilent_tb/UUT/builder_basesoc_axi2axilite0_state
      width: 2
...
```

Figure 22: yaml file

- Table 2 compares outcomes for all fault models across Wishbone, AXI-Lite, and AXI.
- More simulations are required as bus complexity increases (Wishbone $\rightarrow$ AXI-Lite $\rightarrow$ AXI).
- Under simple models, Wishbone exhibits higher attack success, showing that simpler interconnects are easier to disrupt.
- Subsequent analysis examines each successful register-fault combination in detail.

Precise analysis on AXI

- State register is the main target for successful attacks across all fault models.
- completion flag register shows fewer successful attacks.

Table 11: Distribution of successful register combination attacks under different fault models on the AXI Bus

Fault model	state	completion flag	completion flag & state
Bit-flip	75.00%	25%	-
Manipulate Register	83.33%	16.67%	-
2 Bit-Flips	60.00%	20%	20.00%
Manipulate Two Registers	83.13%	16.79%	0.08%

Fault Effect Types

- Instruction skip: Faults (e.g., in ack) disrupt fetch timing, causing key instructions—such as the PIN comparison—to be skipped.
- Data reset: Faults (e.g., in state) trigger error-handling behavior, resetting bus outputs to zero and making g_cardPin appear equal to g_userPin.
- Data misread: Address-selection faults (e.g., in sel) cause reads from unintended modules, substituting variables like g_userPin with g_cardPin.
- Data multiread: Faulty sel may enable multiple memories at once; merged outputs (e.g., ORed values) corrupt data used in authentication.

SEL Under Attack

- **Data reset:** Fault forces `sel` = "0000", masking SRAM data with zeros.
- **Data misread:** Single-bit error in `sel` causes ROM to be read instead of SRAM.
- **Data multiread:** Multiple unintended bits set in `sel` (e.g., "1100"), CPU reads CSR + MAIN_RAM simultaneously.

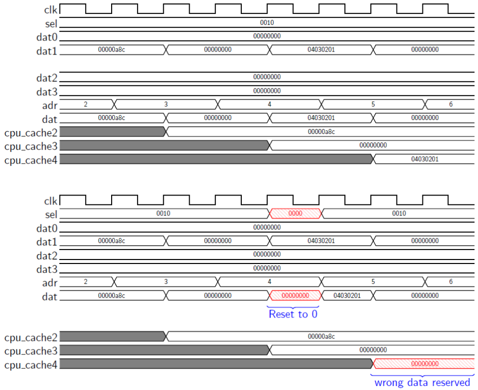


Figure 23: Impact of sel signal on address, data signals and CPU cache without an attack

- V7 showed the lowest fault injection success rate and highest detection capability.
- Countermeasure complexity (code length, CCN, token count) correlated with longer execution time and larger fault surface.
- Misaligned countermeasures (e.g., V4–V5) produced higher success rates than baseline, failing to intercept relevant fault paths.
- None of the countermeasures fully neutralized the fault model impact.
- Findings suggest need for granular analysis of countermeasure logic and deployment.

Grant and ACK Under Attack

- grant from 0 to 1, CPU begins to read from SRAM, ack1 becomes 1
- grant becomes 1 one period before, so as ack1
- Combine attack with ack0, cause ack_d =1 during 2 periods, influence 2 data only during 1 period on the bus, cause the same fault as before.

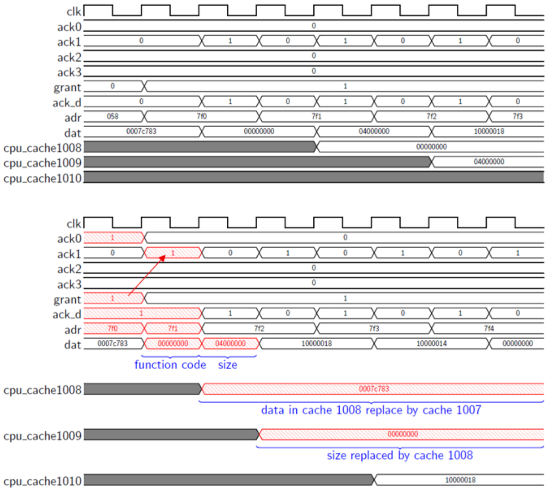


Figure 24: Impact of grant and acknowledge signal on address, data signals and CPU cache without an attack

- V0 / V1: HB reduce little instruction fault.
- V1 / V4: FTL and LC reduce less instruction fault than INL increases, LC reduce many data fault.
- V4 / V5: Double-call reduce instruction fault and data fault.

Benchmark version	CM implemented	Instruction success times	Data success times
V0	-	20	17
V1	HB	19	17
V4	HB+FTL+INL+DPTC+PTCBK+LC	26	5
V5	HB+FTL+DPTC+DC	4	5