

An Ibex RISC-V Dual-Core Lockstep SoC with Integrated Watchdog, Error Management and Reset Control

Madhubabu kandregula¹, Dr. S. Aruna mastani²

¹M. Tech, Department of ECE, JNTUA College of Engineering Anantapur, Andhra pradesh, India,

²Professor, Department of ECE, JNTUA College of Engineering Anantapur, Andhra pradesh, India

Abstract - Embedded processors in automotive electronic control units must continue to communicate correctly even when transient faults, caused by radiation, supply noise or electromagnetic interference, disturb their operation. This paper presents the design and verification of an open-source dual-core lockstep System-on-Chip (SoC) based on the Ibex RISC-V processor, whose objective is reliable transmission of data from the processor to a UART peripheral in the presence of runtime transient faults. Two identical Ibex RV32IMC cores execute the same program in delayed lockstep, so that a transient fault in either core is detected while the program is running. Fault recovery is performed in hardware through a staged reset strategy. When a fault is detected, the Reset Controller first resets the UART peripheral so that corrupted data is not transmitted. If the processor does not return to normal operation, the Watchdog Timer (WDT) times out and the Reset Controller then resets the RISC-V core, restoring correct program execution and data transmission. An Error Collection Unit (ECU) records each fault and classifies the system into OK, Warning, Critical and Safe states. The design was verified by RTL simulation in Xilinx Vivado with instruction-level fault injection. The proposed SoC provides an open, low-cost platform for reliable communication in safety-critical automotive systems.

Keywords - Functional Safety, Ibex RISC-V, Dual-Core Lockstep, Automotive Safety, Watchdog Timer, Error Collection Unit, Reset Controller, Fault Injection, FPGA Verification, Safety-Critical Systems.

I. INTRODUCTION

Electronic control units have taken over many safety-critical functions in modern vehicles, including braking, steering, battery management and driver assistance. Each of these units relies on embedded processors, and a hardware fault in a processor can propagate to an actuator if it is not detected in time. Such faults may be permanent, caused by manufacturing defects or ageing, or transient, caused by radiation, supply noise or electromagnetic interference. Transient faults are particularly difficult to handle because they may last only one clock cycle

and leave no lasting trace. Functional safety requirements for road vehicles [1] define how random hardware faults must be handled. Each safety goal is assigned an Automotive Safety Integrity Level (ASIL), and these requirements specify how reliably faults must be detected and how quickly the system must reach a safe state. For the highest level, ASIL D, faults must be detected and handled within the fault tolerant time interval (FTTI) of the affected function. A processor-level safety mechanism therefore has to cover the complete sequence of detection, classification, reaction and recovery.

Dual-core lockstep is the most widely used processor-level safety mechanism. Two identical cores execute the same program, and a comparator checks their outputs in every clock cycle. When one core runs a few cycles behind the other, a disturbance that affects both cores at the same moment hits different instructions in each. Even a common-cause fault then produces a detectable mismatch [2]. This technique is used in commercial automotive microcontrollers such as the Arm Cortex-M33 in its dual-core lockstep configuration [3]. The open RISC-V instruction set and open-source cores such as lowRISC Ibex [4] now allow lockstep systems to be designed, modified and evaluated openly. The effectiveness of a lockstep system depends on where faults occur and how the system responds after detection. Many evaluations inject faults at a single point, typically the instruction input, and report only whether the fault was detected. However, faults can also strike the register file, the program counter, the data path, or the safety logic itself, and their detectability and detection latency may differ considerably. In addition, the response after detection is critical for a complete safety mechanism. The system must enter a defined safety state, record the fault, catch software activities that stall without causing a mismatch, and restart the processor without losing diagnostic information. we presented an Ibex-based dual-core lockstep System-on-Chip (SoC) with a watchdog timer, an Error Collection Unit (ECU) and a reset controller. That evaluation, however, was limited to a single injection site and did not measure detection latency. Moreover, the ECU moved directly from its normal state to its critical state, so the Warning status was never produced and the fault was not recorded before reset.

II. RELATED WORK

A. Lockstep Processor Architectures

The use of redundant processor cores for error detection is well established in dependable computing. Troppmann and Fuessl introduced delayed lockstep, in which the checker core runs a fixed number of clock cycles behind the main core and their outputs are compared after alignment [2]. Because the two cores are never in the same state at the same time, a disturbance that affects both of them produces different errors that the comparator can detect. Arm applied this principle to the Cortex-M33, combining a redundant core with a dedicated comparator to support safety-critical automotive applications [3]. These commercial solutions confirm the value of lockstep for safety-critical systems. However, their internal safety logic is proprietary and cannot be examined or modified by designers.

B. Open-Source RISC-V Lockstep Design

The open RISC-V instruction set and freely available cores such as Ibex [4] have enabled several open lockstep implementations. CEVERO combines two Ibex cores in lockstep with a recovery module that monitors both cores and, when an error is detected, resets them and restores their state [6]. Its fault injection module is placed between the instruction memory and the core, so faults are applied to the fetched instructions. SafeLS implements a lockstep version of the NOEL-V RISC-V core, in which the inputs of the shadow core pass through staggering registers to desynchronise the two cores, and integrates it into a complete SoC that can be synthesised on an FPGA [7]. In SafeLS, management of the lockstep error signal is left to the system integrator. Rogenmoser et al. proposed Hybrid Modular Redundancy, in which the cores of a RISC-V cluster can be grouped at runtime into dual- or triple-core lockstep, with both software- and hardware-based recovery after an error [8].

C. Supervision and Safe-State Management

Detection is only the first step of a complete safety mechanism. The system must also react to the fault and reach a safe state within the fault tolerant time interval [1]. Abella et al. proposed a safety island, a dedicated hardware block built from open-source RISC-V components that supervises a main processing device, manages its error signals and enforces safe states [9]. This concept shows a clear trend: supervision, error classification and recovery are increasingly realised as dedicated hardware functions rather than left to application software. Our previous work followed the same direction by combining lockstep detection with a watchdog timer, an error collection unit and a reset controller [5].

D. Fault Injection for Safety Evaluation

Fault injection is the main experimental method for verifying that safety mechanisms work as intended. It can be performed physically on hardware, through software, or in simulation [10].

Simulation-based injection at register-transfer level gives full visibility of internal signals and precise control over where and when each fault is applied. This makes it well suited to measuring detection latency and comparing different fault locations. In our previous work [5], faults were injected only at the instruction input of the main core, and the evaluation reported detection without latency. A broader campaign covering several fault locations and fault models gives a more complete picture of the protection provided by the lockstep architecture.

III. SYSTEM ARCHITECTURE

A. Overview of the Proposed SoC

The proposed System-on-Chip (SoC) is intended for automotive applications in which a processor fault could lead to a hazardous situation, such as braking, steering or engine control. It is built on the open-source Ibex RISC-V processor, which provides a compact and well-documented 32-bit core that can be modified freely.

As shown in Fig. 1, the SoC is made up of six functional parts:

1. Dual-core lockstep processing unit: two identical Ibex RV32IMC cores that execute the application.
2. Watchdog Timer (WDT): supervises the processor after a fault has been detected.
3. Error Collection Unit (ECU): collects all fault information and decides on the safety state of the system.
4. Reset Controller: restarts the processor subsystem when the ECU requests it.
5. Memory and bus subsystem: a shared Instruction RAM (IRAM) and Data RAM (DRAM) connected through the Open Bus Interface (OBI).
6. UART: Universal Asynchronous Receiver/Transmitter is a serial communication interface used to transmit and receive data between electronic devices without requiring a shared clock signal.

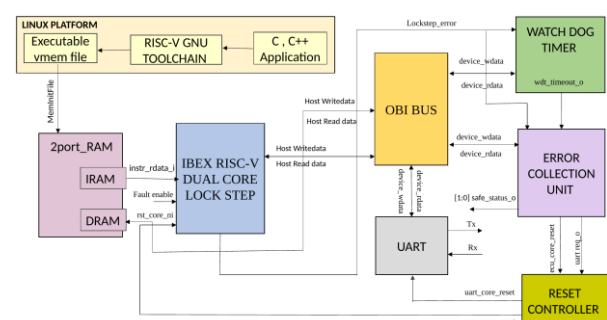


Fig. 1. Architecture of the proposed Ibex RISC-V dual-core lockstep SoC with Watchdog Timer, Error Collection Unit, Reset Controller and UART peripheral.

B. Dual-Core Lockstep Processing Unit

The processing unit contains two identical copies of the Ibex RV32IMC core. This configuration supports the base integer instruction set, integer multiplication and division, and compressed 16-bit instructions, which reduce program size. The two cores work in lockstep: they start from the same reset state, receive the same inputs and execute the same instruction stream. Because they are identical in design and state, a fault-free system always produces exactly the same results in both cores. The cores have different roles:

1. Main core: executes the program and performs all accesses to memory and peripherals through the OBI bus. Only this core is allowed to drive the bus.
2. Shadow core: executes the same instructions in parallel. Its outputs are used only for comparison and never reach the bus.

C. Lockstep Comparator

The comparator is the fault detector of the system. In every clock cycle, it compares the aligned outputs of the main core and the shadow core. These outputs include the program counter, the addresses and data of memory accesses, and the control signals that describe what the core is doing. The comparison uses exclusive-OR (XOR) logic. Each pair of corresponding output bits is XORed, giving 0 when the bits agree and 1 when they differ. Thus, a mismatch in any single bit causes the comparator to assert a fault indication signal immediately.

D. Watchdog Timer

The Watchdog Timer provides *time-based supervision* of the processor once a fault has been detected. The comparator can tell that something has gone wrong. The WDT checks whether the processor manages to recover within an acceptable time.

The behaviour of the WDT is described by the finite state machine (FSM) in Fig. 2. After reset, the WDT starts in the IDLE state and remains there as long as no lockstep error is reported. When the comparator raises the lockstep error signal, the FSM leaves IDLE and enters the counting state, in which the counter is updated in every clock cycle. The FSM stays in the counting state until the counter reaches the programmed timeout value of 10 clock cycles. At that point the WDT asserts `wdt_time_out` and reports the timeout to the ECU, after which the FSM always returns to IDLE and is ready to supervise the next fault.

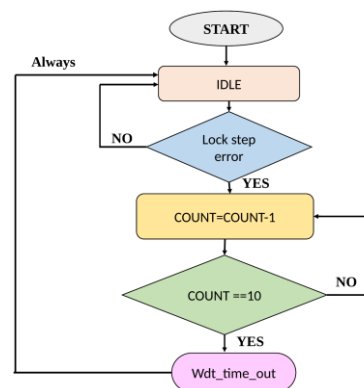


Fig. 2. Finite state machine of the Watchdog Timer.

E. Error Collection Unit

The ECU is the central safety management unit of the SoC. All fault information from the safety mechanisms is brought together in this one block, so the system reacts to every fault in a consistent and predictable way.

Fig. 3 shows the FSM of the ECU, which reports the current safety state on the two-bit output `safe_status_o`. After reset, the ECU is in the OK state (`safe_status_o = 00`). If a lockstep error is reported, the ECU moves to the WARNING state (`safe_status_o = 01`) and asserts `ecu_uart_reset`, so that the UART is reset and no corrupted data is transmitted. If there is no lockstep error, the ECU checks the watchdog: when `wdt_time_out` is asserted, it moves to the CRITICAL state (`safe_status_o = 10`) and asserts `ecu_core_reset` to request a reset of the RISC-V core; otherwise it remains in OK. From both WARNING and CRITICAL, the ECU enters the SAFE state (`safe_status_o = 11`), in which `ecu_uart_reset` and `ecu_core_reset` are released. The FSM then always returns to OK, so that every fault is recorded and handled in the same way.



Fig. 3. Finite state machine of the Error Collection Unit.

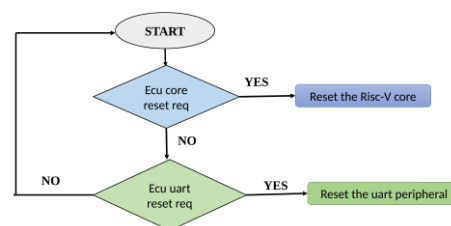


Fig. 4. Finite state machine of the Reset Controller.

F. Reset Controller

The Reset Controller carries out the staged recovery requested by the ECU. Its FSM, shown in Fig. 4, checks two request signals in priority order. If `ecu_core_reset` is asserted, it resets the RISC-V core through `rst_core_ni`, which restarts program execution. Otherwise, if `ecu_uart_reset` is asserted, it resets only the UART peripheral through `rst_uart_ni`, so that the processor keeps running while the transmission path is cleared. When neither request is active, the FSM returns to its start state and waits for the next request. Resetting only the affected block keeps recovery time short and avoids an unnecessary restart of the whole SoC.

G. Memory and Bus Subsystem

The SoC uses a dual-port RAM that is divided into two regions:

- 1) Instruction RAM: holds the program code. The application, written in C or C++, is compiled with the RISC-V GNU toolchain into an executable memory image, which is loaded into IRAM before execution.
- 2) Data RAM: holds the variables, stack and other data used by the program.

H. UART Peripheral

The SoC includes a UART peripheral that provides a serial communication interface to the outside world. It allows the program running on the lockstep processor to send data to an external terminal. This gives a simple and direct way to confirm that the processor fetches instructions, executes them and accesses peripherals correctly.

The Proposed lockstep processor acts as the master, and the WDT, ECU and UART are connected to the OBI bus as slave peripherals. The architecture has two goals. First, it executes an application program that transmits the "Hello World" string through the UART. Second, it continuously supervises the processor so that faults are detected, classified and recovered from in hardware.

IV. METHODOLOGY

The aim of this work is to design an Ibex RISC-V dual-core lockstep architecture and it can be used in safety-critical automotive systems. The design covers four tasks: it detects faults while the program is running, supervises the processor with a watchdog, assigns the system a safety state, and recovers by restarting only the affected subsystem. Two identical Ibex RV32IMC cores form the processing unit and run in delayed lockstep. They receive the same instruction stream, but the second core trails the first by a few clock cycles. This deliberate lag improves the detection of transient faults. At any clock cycle t , the behaviour of each core can be described by the values it produces, namely its program counter, its arithmetic result and its control signals:

$$Y_1(t) = \{PC_1, ALU_1, CTRL_1\} \quad (1)$$

$$Y_2(t) = \{PC_2, ALU_2, CTRL_2\} \quad (2)$$

When neither core is affected by a fault, the two sets of values are identical:

$$Y_1(t) = Y_2(t) \quad (3)$$

A fault in one of the cores breaks this equality:

$$Y_1(t) \neq Y_2(t) \quad (4)$$

The comparator turns this condition into a binary detection signal:

$$Z(t) = 0, \text{ when } Y_1(t) = Y_2(t) \quad (5)$$

$$Z(t) = 1, \text{ when } Y_1(t) \neq Y_2(t) \quad (6)$$

In hardware, the comparison is carried out with exclusive-OR gates applied to every pair of corresponding output bits:

$$N = Y_1 \oplus Y_2 \quad (7)$$

All bits of N are 0 while the cores agree. A single bit equal to 1 is enough to indicate a fault. As soon as $Z(t) = 1$, the comparator raises the lockstep error signal, which activates the Watchdog Timer and informs the ECU. Algorithm 1 outlines this detection procedure.

A Fault enable control is added on the instruction path between IRAM and the processor. When this control is active, the word delivered to the main core on `instr_rdata_i` is modified, while the shadow core still receives the original word. The modified instruction is described as:

$$I_f = I_o \oplus F_b \quad (8)$$

where:

- I_f = faulty (modified) instruction delivered to the main core,
- I_o = original (true) instruction read from IRAM,
- F_b = fault bit mask that selects the bits to be inverted.

Algorithm 1: Lockstep Fault Detection

- 1: Reset both cores to the same initial state
 - 2: Load the same program into both cores
 - 3: while the system is running do
 - 4: Execute the current instruction on Core-1 and Core-2
 - 5: $N = Y_1 \text{ XOR } Y_2$
 - 6: if N contains a 1 then
 - 7: Raise the lockstep error
 - 8: Start the Watchdog Timer
 - 9: end if
 - 10: end while
-

The Watchdog Timer adds a time-based check to the comparison-based detection. Its counter T_{WDT} is cleared whenever the processor services the timer. Otherwise, the counter increases by one in every clock cycle:

$$T_{WDT}(t+1) = T_{WDT}(t) + 1 \quad (9)$$

When the counter reaches the programmed limit $T_{WDT,max}$, the timer raises its timeout output:

$$\text{timeout_o} = 1, \text{ if } T_{WDT}(t) \geq T_{WDT,max} \quad (10)$$

The ECU uses watchdog signals to place the system in one of four safety states, encoded on two bits:

- 00 – OK
- 01 – Warning
- 10 – Critical
- 11 – Safe

V. RESULTS AND DISCUSSION

The proposed SoC was tested in Xilinx Vivado. The design was simulated at RTL level, checked with fault injection results. The results for each block are given below.

A. Lockstep Fault Detection

Fig. 5 shows the two Ibex cores running the same program in delayed lockstep. In normal operation, both cores fetch the same instructions and the comparator output stays low. To test fault detection, the `fault_en` signal was turned on. The main core then received a corrupted instruction, while the shadow core continued with the correct one. As a result, the outputs of the two cores no longer matched, and the comparator raised the `alert_major_internal_o` signal. After a short reset, both cores started again and continued in lockstep. The `instr_addr_o` signal of the instruction RAM confirms this recovery. While `rst_ni` is low, the fetch address is held, and as soon as the reset is released the processor fetches its first instruction from the reset vector at offset 0x80 from the base of the instruction memory (base + 0x80). In Fig. 5 this appears as address 0x00100080 on `instr_addr_o`, after which the address advances normally and the program continues from the start. This shows that the RISC-V core recovered successfully from the injected fault and resumed correct execution.



Fig. 5. Simulation waveform of delayed lockstep execution with a fault injected into the main core and the resulting comparator alert.

B. Fault Injection Results

Table I shows sample fault injection cases. For each case, Core-1 is the corrupted instruction delivered to the main core and Core-2 is the original instruction executed by the shadow core. In every case shown, the comparator detected the mismatch.

TABLE I
Sample Fault Injection Results

Core-1	Core-2	Status
4681A011	4681A012	Detected
25912961	25912962	Detected
05130000	05130001	Detected
85138000	85138001	Detected

C. Watchdog Timer

Fig. 6 shows the Watchdog Timer in operation. Its timeout was set to 10 clock cycles. When abnormal execution was signalled, the counter started counting. After 10 cycles, which is 200 ns at 50 MHz, the timer raised the `timeout_o` signal. This shows that the watchdog reacts quickly when the processor does not behave normally.

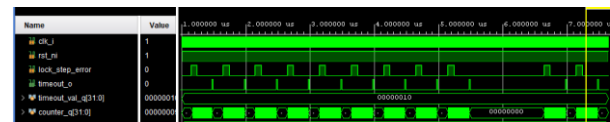
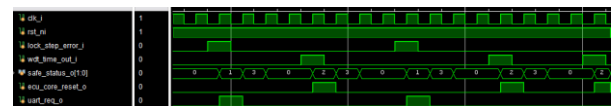


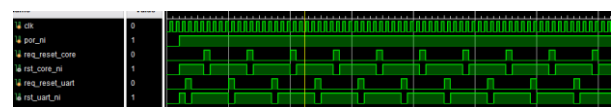
Fig. 6. Simulation waveform of the Watchdog Timer showing counter operation and timeout generation after 10 clock cycles.

D. ECU and Reset Controller

Fig. 7 shows how the ECU and the Reset Controller respond to faults. Fig. 7(a) shows the ECU. After reset, `safe_status_o` is 0 (OK). When the comparator reports a lockstep error on `lock_step_error_i`, the ECU moves to the Warning state (`safe_status_o` = 1) and asserts `uart_req_o`, which requests a reset of the UART so that no corrupted data is transmitted. It then passes through the Safe state (`safe_status_o` = 3) and returns to OK. When the watchdog later reports a timeout on `wdt_time_out_i`, the ECU moves to the Critical state (`safe_status_o` = 2) and asserts `ecu_core_reset_o` to request a reset of the RISC-V core, again passing through Safe before returning to OK. The waveform repeats this sequence for several faults, and every fault produces the same state sequence and the same reset request, which confirms that the ECU behaves as described by its FSM in Fig. 3.



(a) Error Collection Unit



(b) Reset Controller

Fig. 7. Simulation waveform of the ECU and Reset Controller showing safety-state transitions, reset request generation and processor recovery.

Fig. 7(b) shows the Reset Controller. After power-on, `por_ni` is released and both reset outputs, `rst_core_ni` and `rst_uart_ni`, go high. Each pulse on `req_reset_uart` drives `rst_uart_ni` low for a short time, so that only the UART is reset while the core keeps running. Each pulse on `req_reset_core` drives `rst_core_ni` low, so that the RISC-V core is restarted. Both resets are active-low and are released automatically after the reset pulse. A UART reset never disturbs the core reset line and a core reset never disturbs the UART reset line, which shows that the Reset Controller performs selective recovery of only the affected block.

E. UART Output

The processor sent the string "Hello World" to the UART. Fig. 8 shows the output on the Tx line. Each character is sent as one start bit, eight data bits and one stop bit. The received characters matched the expected string, which confirms that the processor and the UART communicate correctly over the OBI bus.

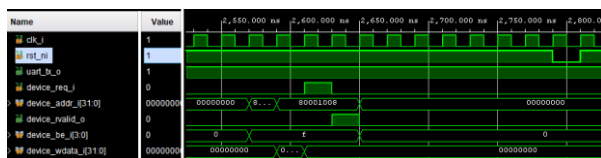


Fig. 8. Simulation waveform of the UART Tx line transmitting the "Hello World" string.

VI. CONCLUSIONS

This paper presented a dual-core lockstep SoC based on the open-source Ibex RISC-V processor for automotive safety-critical applications. Two identical Ibex RV32IMC cores operate in delayed lockstep as the bus master, while the WDT, ECU, Reset Controller and UART are connected as slave peripherals through the OBI bus. Instruction-level faults are detected by comparing the outputs of the two cores. The WDT detects abnormal processor behaviour within 10 clock cycles (200 ns at 50 MHz), the ECU classifies faults into defined safety states, and the Reset Controller performs selective recovery to improve availability and minimise downtime. Correct functional operation was confirmed by transmitting the "Hello World" message through the UART. Because the shadow core only mirrors the main core and never drives the bus, the lockstep mechanism adds fault detection without adding instructions to the application.

REFERENCES

- [1] ISO 26262-5:2018, *Road Vehicles – Functional Safety – Part 5: Product Development at the Hardware Level*. Geneva, Switzerland: International Organization for Standardization, 2018.
- [2] R. Troppmann and B. Fuessl, "Delayed lock-step CPU compare," U.S. Patent Application US 2008/0244305 A1, Texas Instruments Deutschland GmbH, Oct. 2, 2008.
- [3] ARM Ltd., "Application note: Cortex-M33 dual core lockstep," Version 1.0, 2017. [Online]. Available: <https://www.arm.com>
- [4] lowRISC C.I.C., "Ibex RISC-V core documentation." [Online]. Available: <https://ibex-core.readthedocs.io>

- [5] M. B. Kandregula and S. A. Mastani, "Open-source Ibex RISC-V based ISO 26262-compliant dual-core lockstep SoC with watchdog timer, error collection unit, and reset controller for automotive safety-critical applications," in *Proc. 1st Int. Conf. VLSI Circuits and Systems (ICVCS)*, 2026, doi: 10.1109/ICVCS69586.2026.11683163.
- [6] I. Silva, O. do Espírito Santo, D. do Nascimento and S. X. de Souza, "CEVERO: A soft-error hardened SoC for aerospace applications," in *Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais (SBESC)*, 2020.
- [7] M. Saraseca, S. Alcaide, F. Fuentes, J. C. Rodriguez, F. Chang, I. Lasfar, R. Canal, F. J. Cazorla and J. Abella, "SafeLS: An open source implementation of a lockstep NOEL-V RISC-V core," in *Proc. IEEE 29th Int. Symp. On-Line Testing and Robust System Design (IOLTS)*, Crete, Greece, 2023, pp. 1–7, doi: 10.1109/IOLTS59296.2023.10224867.
- [8] M. Rogenmoser, Y. Tortorella, D. Rossi, F. Conti and L. Benini, "Hybrid modular redundancy: Exploring modular redundancy approaches in RISC-V multi-core computing clusters for reliable processing in space," *ACM Trans. Cyber-Physical Systems*, vol. 9, no. 1, Art. 8, pp. 1–29, 2025, doi: 10.1145/3635161.
- [9] J. Abella, F. J. Cazorla, S. Alcaide, M. Paulitsch, Y. Peng and I. Pinto Gouveia, "Envisioning a safety island to enable HPC devices in safety-critical domains," arXiv preprint arXiv:2307.11940, 2023.
- [10] M.-C. Hsueh, T. K. Tsai and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, Apr. 1997, doi: 10.1109/2.585157.