A Dual Lockstep Processor System-on-a-Chip for Fast Error Recovery in Safety-Critical Applications

Mong Tee Sim
Dept. Computer Science
University of Colorado Colorado Springs
Colorado Springs, CO, USA
mong_sim@hotmail.com

Yanyan Zhuang
Dept. Computer Science
University of Colorado Colorado Springs
Colorado Springs, CO, USA
yzhuang@uccs.edu

***Abstract*—In a lockstep system, errors induced by single-point and common-mode failures could cause catastrophic fatality without proper error detection or write-protection circuitries. In this paper, we design and implement a dual lockstep processor using a two pipelined assembly that executes two virtual cores each, in an interleaved fashion. Such an approach could overcome common-mode failures (CMFs). Furthermore, by running the same code in a secondary pipeline, our system can detect a single-point-of-failure (SPOF). Our technique easily maintains the synchronization between the two virtual cores and omits other fabric that binds a typical dual-core lockstep processor. This reduces the die size and provides early error detection before an error becomes unrecoverable. We achieve our goal by incorporating the lockstep function in the micro-architecture and employing fine-grained multitasking to increase a system's fail-safe capabilities. Finally, we validate our lockstep processor using the RISC-V 32IM ISA test benches, Dhrystones and Coremark benchmarks, and ModelSim. Our results show a 100% self-checking coverage for stuck-at faults and complete error containment. Since our framework operates fine-grained multitasking, we achieve two lockstep processors instead of one, which saves hardware costs.**

Keywords—Lockstep, fault-tolerance, Redundancy, Reliability, fine-gained, dual-core

I. INTRODUCTION

The rise of autonomous vehicles requires that control systems have high reliability and fault tolerance. Particularly, safety-critical applications such as anti-lock braking, vehicle stability, and control by wire have high safety requirements. Despite the shrinking semiconductor geometry that makes chips prone to transient errors, ***lockstep processors*** provide opportunities to implement algorithms for error detection and error handling. These algorithms allow errors to fail silently and the system to recover quickly and gracefully. A conventional solution is to use a dual-core system running the same application in each core, which produces the same result. If the two core results are different, the system can take further actions, such as retry an instruction, or reset the system.

A dual-core system is generally known as a Master and Slave lockstep processor [7]. The redundant processor, called the Slave, starts with the same state as the Master processor upon power-on reset (POR), runs the same instructions, receives the same input, and generates the same output in each clock cycle if no error occurred. If the Master and Slave outputs are different, then an error is detected, and the system silently discards the error and re-executes the instruction for error correction. If the number of attempts to correct the error exceeds a permitted threshold, then a system reset is triggered. At a minimum, the lockstep system that implements a dual modular redundancy (DMR) can detect and prevent such errors from becoming unrecoverable, for example, by early mitigation.

Some lockstep systems implement a time shift between the Master and the Slave to increases the error detection probability induced by external influences, such as an electrical surge and ionizing radiation. Such a variant is known as a Delayed Lockstep [9] system. This approach introduces time diversity, which reduces the probability that a noise pulse hits the system that can result in the same failure at both the Master and Slave. With time diversity, the Master and Slave will never run the same instruction simultaneously. However, such a dual-core implementation also poses other challenges such as synchronization between the cores, the requirement of additional delay fabric, and data storage for delay comparison.

In our approach, we modify a two-pipelined assembly micro-architecture to support fine-grained multitasking that supports two virtual cores (VCs) in each pipeline. Through fine-grained multitasking, we eliminate the need for additional fabric for synchronization among the four virtual cores. Furthermore, as our micro-architecture feeds instructions into the two processor pipelines in an interleaved manner, by alternating between the Master and Slave VCs, we accomplish delayed lockstep [9]. Thus we eliminate all potential common-mode-failures (CMFs). Through careful analysis, we systematically augment the lockstep functional blocks to eliminate all potential single-point-of-failures (SPOFs) [8]. As a result, we achieve a high error detection rate and low recovery latency. In particular, we detect all errors within our VCs before they propagate to the output and pollute other functional units such as system registers, memory, or peripherals. This error containment is essential for fast recovery. Finally, unlike other dual-core designs, we achieve two lockstep processors instead of one. This implementation could reduce the high hardware costs.

Our research makes the following contributions:

- We propose a novel method that uses a fine-grained multitasking technique to design two lockstep processors with an interleaved time diversity, to improve error detection up to 100%.

- Our system guarantees fast error recovery by hardware protection to prevent an error from propagating into other system modules, which can cause an unrecoverable failure.
- Our approach uses two five-stage pipelined processors to create two lockstep processors. It reduces the additional fabric that binds the four VCs to operate in the lockstep mode. It also reduces both the die size and manufacturing costs.

The rest of this paper starts with introducing the proposed lockstep processors and the block diagram of our implementation in Section II. In Section III, we describe our lockstep processor's design, error detection, and error containment. Section IV presents the experimental results. Related work is reviewed in Section V, and finally, Section VI concludes.

II. Lockstep Processor Architecture

Our lockstep processor design differs from the traditional Master/Slave design [7]. To mitigate SPOFs [8], we compare the results from both pipeline stages, before allowing further operations. To achieve fast error recovery, we implement hardware protection to prevent errors from propagating into memory, registers, and peripherals. Finally, to achieve fast error recovery through a reset, we implement a system-wide hard reset to initialize the register files, interrupt controller, and exception storage memories [13].

Fig. 1 shows the block diagram of our lockstep processor that comprises two five-stage pipelined assemblies augmented with lockstep functional blocks. The functional blocks work as follows:

- The Program Counter Module (PCM) manages the application instruction address and the virtual core identification (VCID). The VCID is used to synchronize the two virtual cores in each pipeline.
- The Exception Hazard Control controls an application when it restarts from the point of exception.
- The M-Stage Checker verifies the Master and Slave's results before allowing memory and I/O operations to proceed.
- The WB-Stage Checker verifies the Master and Slave's results before granting a register operation to proceed.
- The Virtual Core Verification Block (VCVB) ensures the VCIDs operate in an interleaved fashion to ensure that the lockstep functional blocks are in synchronization.
- Pipeline A runs the applications in Master A and Slave B, and Pipeline B runs the applications in Master B and Slave A.

Next, we describe the circuit design of each functional block in Section III.

III. Micro-Architecture Circuit Design

This section first describes how we use a single five-stage pipelined assembly with fine-grained multitasking technique [14-20] to create a framework for our lockstep processor. To maintain synchronization between the Master and Slave, we implement a virtual core verification block (VCVB) to ensure the VCs operate in a synchronous and interleaved manner, with VC0 followed by VC1. We then explain the five stages of the lockstep processor and VCID correction, i.e., resetting the system when VCIDs are out of sync.

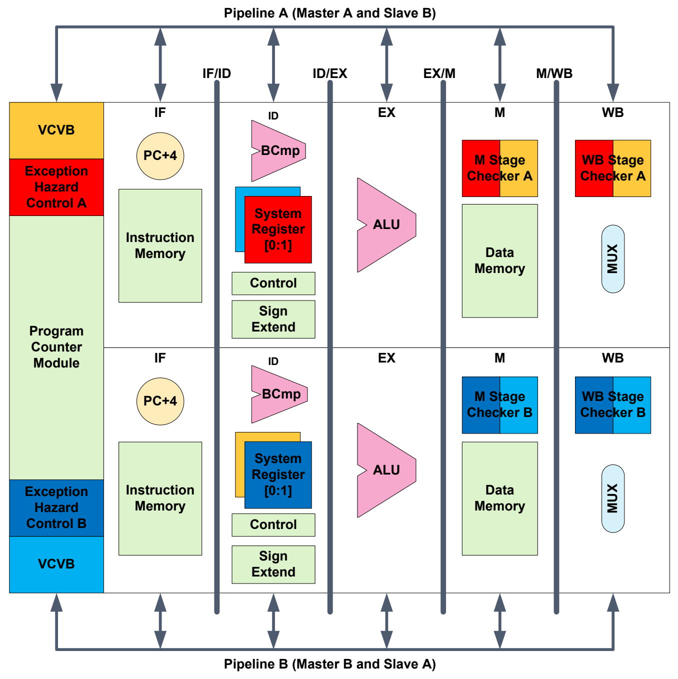


Fig. 1. This block diagram shows the Interleaved Delayed Lockstep Processor.

A. *Fine-grained Multitasking*

Our design framework supports two lockstep processors, each with VC0, followed by VC1 (shown in Fig. 2). To run two VCs in each pipeline, we augment each pipeline with two program counters and two register files. We designate each VC0 to run the instructions from the Masters, and the VC1 to run the instructions from the Slaves. To mitigate CMFs, we arrange the execution of the Masters and Slaves in such a way, as shown in Fig. 2, where Pipeline A and Pipeline B are the two pipelines. This process repeats until the processor is powered off. In the following, whenever Pipeline A executes instructions as a Master (Slave), it is shown in red (light blue). Similarly, when Pipeline B executes instructions as a Master (Slave), it is shown in dark blue (yellow). By running the VCs in an interleaved fashion, or running two copies of the same program with one stage delay from a different pipeline, we mitigate the CMFs and SPOFs in the lockstep processor [8] [9].

Our design can run two programs concurrently with each pipeline running as a Master for a program and as a Slave for the other program. The Checker compares the results of the Masters and Slaves to prevent errors from becoming unrecoverable. Fig.

3 shows how our design executes the two programs in the two pipelines.

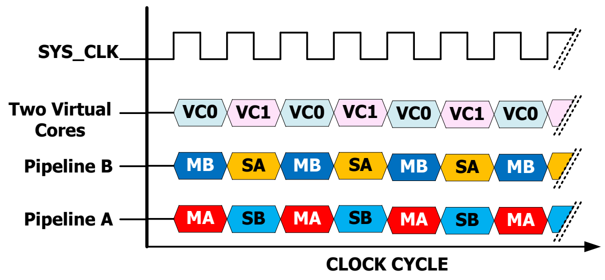


Fig. 2. Fine-grained multitasking operation with VC0, followed by VC1. Master A and Slave B run in pipeline A, and Master B and Slave A run in pipeline B (MA: Master A, MB: Master B, SA: Slave A, and SB: Slave B).

In our design, as shown in Fig 3, we designate Pipeline A (PLA) to run as the Master of the first program, or P1, and the Slave of the second program, or P2, using the same coloring as in Fig. 2. Similarly, Pipeline B (PLB) is run as the Master of P2 and the Slave of P1. As shown in Fig. 3, PLA and PLB alternate between one another, executing as the Master or Slave of different programs in adjacent clock cycles (CC). For example, in clock cycle 0 (when CC=0), PLA runs as the Master of P1 and PLB as the Master of P2, both at an instruction fetch (IF) stage. In the next clock cycle (CC=1), PLA starts the next instruction in the IF stage as P2's Slave and PLB as P1's Slave. This arrangement provides one clock cycle delay between the running programs and allows the pipeline stages to execute different instructions at any given time. By providing such time diversity, we increase our system's error detection caused by CMFs, and by having a two pipelined assembly, we can detect all SPOFs. We detail the design of our micro-architecture and how we achieve our goals in the following subsections.

PLA		PLB		Instructions in Pipelines Stages						
VC0	VC1	VC0	VC1	CC	IF	ID	EX	M	WB	PL
MA	SB	MB	SA	0	lw					A
lw	addi	addi	lw		addi					B
slli	sub	sub	slli	1	addi	lw				A
add	ble	ble	add		lw	addi				B
sltu	sub	sub	sltu	2	slli	addi	lw			A
add	sub	sub	add		sub	lw	addi			B
srli	srai	srai	srli	3	sub	slli	addi	lw		A
srli	srai	srai	srli		slli	sub	lw	addi		B
lui	add	add	lui	4	add	sub	slli	addi	lw	A
addi	srli	srli	addi		ble	slli	sub	lw	addi	B
🟥	P1 Master			5	ble	add	sub	slli	addi	A
🟧	P1 Slave				add	ble	slli	sub	lw	B
🟦	P2 Master			6	sltu	ble	add	sub	slli	A
🔷	P2 Slave				sub	add	ble	slli	sub	B

Fig. 3. The two applications (MA/SA and MB/SB) execute in both pipelines in both the Master and Slave in an interleaved fashion to prevent CMFs and to cross-check each other's results to detect SPOFs. (MA: Master A, MB: Master B, SA: Slave A, SB: Slave B, CC: Clock Cycle, PL: Pipeline, PLA: Pipeline A, PLB: Pipeline B)

B. Program Counter Module (PCM)

This section describes how the system updates the PCs (shown in Fig. 4) during normal operation (i.e., when no jump/branch or error occurs), a jump or branch condition, and an exception. Upon POR, Master and Slave A's reset vector addresses are the default 0x0000_0000. The user of the processor must define Master and Slave B's reset vector addresses. This feature is essential because not all programs require the same amount of memory space.

Since both pipelines' PCs work in the same way, we will only describe Pipeline A, as shown in the top half of Fig 4. Upon POR, the system sets PCA, as shown in Fig. 4, to the Master's reset vector address and PCB to the Slave's reset vector address. During normal operation, PCA in the top half of Fig. 4 gets its next address from PCB (Slave B's address) through MUX3, and PCB gets its following address from PCA + 4 (Master A's next address) through MUX1. This process repeats until it is interrupted by the system, e.g., with a jump or a branch condition, or an exception.

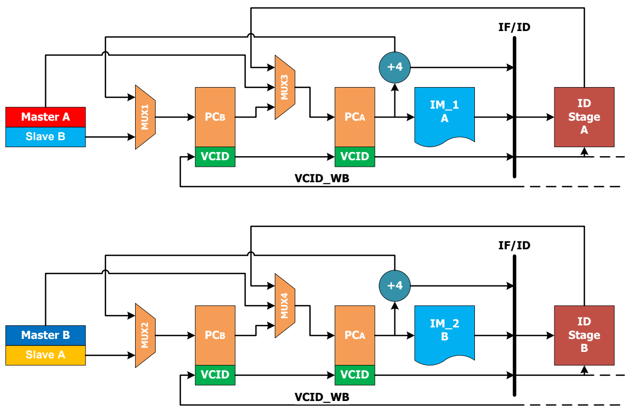


Fig. 4. This diagram shows how the system updates the program counters during a normal operation, branch target address, and exception.

When the Instruction-Decode (ID) stage decodes a jump or a branch-taken, the ID stage provides the jump/branch target address and a MUX3 select signal to MUX3. Assume that the Master's program executes a jump instruction, then PCA has Slave B's next address, and PCB has the Master's following address. In the next clock cycle, PCB has the Slave's address PCA + 4 through MUX1, and PCA has the jump address from MUX3.

During an exception, the system feeds the exception address in Master to PCA through MUX3 and the Slave's address to PCB through MUX2. We will explain the exception return in the following sections.

C. Virtual Core Verification Block (VCVB)

Our lockstep processor must maintain the synchronization between the interleaving VCs, even in a harsh operating environment, such as a power surge and ionizing radiation. To prevent the VCIDs from flipping out-of-order, we implement a Virtual Code Verification Block (VCVB).

The VCVB (shown in Fig. 5) design ensures that the VCIDs are operating in the correct order, and the fine-grained processor is running in the correct sequence with VC0, followed by VC1, and repeats in an infinite loop. The six VCID inputs in Fig. 5 come from PCB, PCA, the ID stage, EX (Execution) stage, M (Memory) stage, and WB (Write-Back) stage. All the input VCID values are XOR-ed with each other. The XOR-ed pairs are shown in Fig. 5, i.e., the VCID from PCA is XOR-ed with the VCID from PCB, and so on. The output of the six XOR-ed pairs is then fed to a six-input NAND gate. If any two of VCID values are the same, then the processor is out of sync. After being XOR-ed, the two identical VCID values produce a low input to the NAND gate. This causes the NAND gate to produce a high output, which raises a VCID exception.

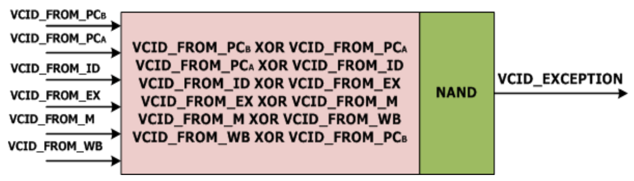

VCID_FROM_PCB
VCID_FROM_PCA
VCID_FROM_ID
VCID_FROM_EX
VCID_FROM_M
VCID_FROM_WB
VCID_FROM_PCB XOR VCID_FROM_PCA
VCID_FROM_PCA XOR VCID_FROM_ID
VCID_FROM_ID XOR VCID_FROM_EX
VCID_FROM_EX XOR VCID_FROM_M
VCID_FROM_M XOR VCID_FROM_WB
VCID_FROM_WB XOR VCID_FROM_PCB
NAND
VCID_EXCEPTION

Fig. 5. The Virtual Core Validation Block (VCVB) ensures the VCs are running in the correct sequence.

D. Memory Stage (M)

Both the M and WB stages are critical in our lockstep design. The M stage writes datum to the data memory and I/O, and the WB stage writes datum to the system registers. In these two stages, we implement checkers and delay buffers to allow the checkers to verify the Master and Slave's results without delaying the processors' operation.

Fig. 3 shows that Pipeline A is the Master of program1, and Pipeline B is the Slave of program1. During a memory write operation, Pipeline A processes an instruction of the program first, followed by Pipeline B in the next clock cycle. Since the Master and the Slave are operating from different pipelines with one clock cycle apart, we implement a delay buffer to store the Master's result for one clock cycle, in time for the arrival of the Slave's result for comparison.

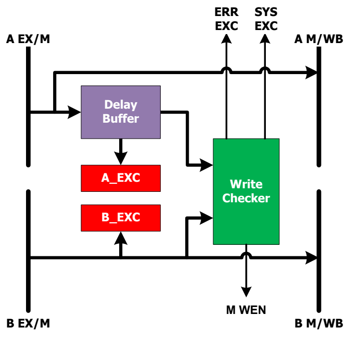


Fig. 6. Memory Stage Checker

When a memory operation is at the M stage, the Master's I/O and Memory information gets stored in the EX/M pipeline register. One clock cycle later, the system stores the Master's inverted information (control signals, write address, datum, PC, PC + 4, and VCID) in the delay buffer (shown in Fig. 6). In this cycle, the Slave's information from Pipeline B arrives. The Checker takes the inverted Master's information and XORs it with the Slave's information. If the PC addresses or the control signals are in disparity, the Checker throws a system exception, disables the write operation, and invokes the exception handler. The system stores both Master and Slave's exception information in A and B EXC buffers (information in the delay buffer for the Master, and information in the EX/M pipeline register for the Slave), for the exception handler to process further. The exception handler will perform the following:

- If Master and Slave's control signals mismatch, the system resets the lockstep processor that runs program1.
- If Master and Slave's PC addresses mismatch, the handler checks if the PC + 4 addresses also match. If so, the program resumes from the point of error. Otherwise, the system resets the lockstep processor.

If the Checker detects any error between the Master and Slave other than control signal or PC address mismatch, it throws an erroneous exception and the application restarts from the point of failure using the address stored in the PC exception buffer. This method allows for a fast recovery from errors. We include an erroneous counter to enable the user to set an error threshold before the system resets.

If the Checker detects no error, the system inverts the information in the delay buffer to improve CMFs detection during the checking stage. It stores the datum from Pipeline A in data memory A, and the datum from Pipeline B to data memory B.

E. Write-Back Stage (WB)

The WB stage writes datum to the system register. In this stage, we implement a checker block similar to the M-Stage Checker (shown in Fig. 6). However, due to the delay write-back to the system register, we need to forward the Master's data to the ID stage and write the datum to the Master's system registers in the same clock cycle. Meanwhile, the writing of the Slave's datum also happens in the same clock cycle.

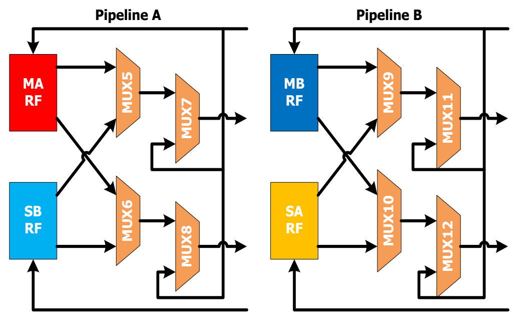


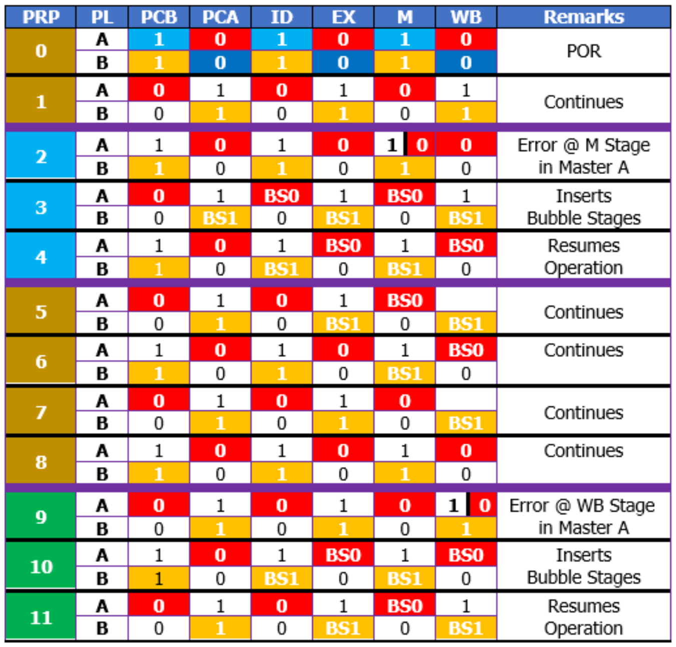
Fig. 7. The register file block allows the validated write-back to be stored in both register files, and it enables data forwarding to the ID stage if rs1 or rs2 register number is equal to the write-back register number (only Master register files, MA: Master A, MB: Master B, SA: Slave A, SB: Slave B, and RF: Register File).

Master A and B's register files provide write-back datapaths and data forwarding paths. This data forwarding path is necessary due to the delay write-back of the Master's write-back datum. The delayed write ensures no erroneous data can be propagated into the system registers. Furthermore, with data forwarding, the processor functions as usual with no stalling.

F. VCID Correction

Upon POR, the VCID values are set, as shown in the first row in Fig. 8**.** The first column in Fig. 8 is the Program Reference Point or PRP, and the second column indicates which pipeline (PL) is run. In the next clock cycle, the VCID values are shown in the next row**,** when PRP=1. The VCIDs repeat themselves between the two instances when PRP=0 and 1, until the processors are powered off.

When the lockstep M-Stage Checker throws an erroneous exception, Master A is fetching a new instruction in Pipeline A, using PCA address (Fig. 8**,** when PRP=2). At this instant, the Exception Hazard Control feeds the exception stage's PC address to Master A in Pipeline A, through PCB, and inserts bubble stages to both lockstep processors' Master A and Slave A stages, in both Pipelines A and B (Fig. 8**,** when PRP=3). A clock cycle later, it feeds the exception stage's PC to Pipeline B, into PCB. After that, the system resumes its operation.



PRP	PL	PCB	PCA	ID	EX	M	WB	Remarks
0	A	1	0	1	0	1	0	POR
	B	1	0	1	0	1	0	
1	A	0	1	0	1	0	1	Continues
	B	0	1	0	1	0	1	
2	A	1	0	1	0	1 | 0	0	Error @ M Stage in Master A
	B	1	0	1	0	1	0	
3	A	0	1	BS0	1	BS0	1	Inserts Bubble Stages
	B	0	BS1	0	BS1	0	BS1	
4	A	1	0	1	BS0	1	BS0	Resumes Operation
	B	1	0	BS1	0	BS1	0	
5	A	0	1	0	1	BS0		Continues
	B	0	1	0	BS1	0	BS1	
6	A	1	0	1	0	1	BS0	Continues
	B	1	0	1	0	BS1	0	
7	A	0	1	0	1	0		Continues
	B	0	1	0	1	0	BS1	
8	A	1	0	1	0	1	0	Continues
	B	1	0	1	0	1	0	
9	A	0	1	0	1	0	1 | 0	Error @ WB Stage in Master A
	B	0	1	0	1	0	1	
10	A	1	0	1	BS0	1	BS0	Inserts Bubble Stages
	B	1	0	BS1	0	BS1	0	
11	A	0	1	0	1	BS0	1	Resumes Operation
	B	0	1	0	BS1	0	BS1	

Fig. 8. VCID correction during M and WB Stages Exception. This diagram shows how the exception hazard control inserts bubble stages to ensure proper execution of the Lockstep Processor A (Our new design has two lockstep processors). For clarity, Lockstep Processor B is not highlighted and is operating without error. (PRP: Program Reference Point, PL: Pipeline).

When the lockstep WB-Stage Checker throws an erroneous exception, Master A's address is in Pipeline A's PCB. The Exception Hazard Control enables MUX3 (see Fig. 4). It bypasses PCB to store the exception stage's PC in PCA in Pipeline A. The Exception Hazard Control also saves the exception stage's PC to Slave A in Pipeline B, through PCB, and inserts bubble stages to both lockstep processors' Master A and Slave A stages, in both Pipelines A and B (Fig. 8, when PRP=10). After that, the system resumes its operation. This example shows fast recovery when dealing with erroneous WB exceptions.

IV. Validation

We validate that our lockstep processors are ISA compliant using the RISC-V ISA test benches and its runtime operation, including Dhrystones and Coremark benchmarks. The lockstep Checkers and VCVB functional operations are verified using ModelSim. Our design can detect CMFs through the time diversity between the Master and Slave processors. Additionally, we tested all SPOFs using ModelSim, in both Pipelines A and B, EX/M, and M/WB pipeline registers and the EX stages.

We induced 17,956 single flipped bits to simulate SPOFs, and the instructions are validated using either the RISC-V ISA test suite or are validated using ModelSim. Our lockstep processors detected all the flipped bits as errors. In the following, we provide the details of our tests. Fig. 9 shows the breakdown of the test results.

As shown in Fig 9, we injected errors in the EX/M and M/WB pipeline registers, the VCID registers, and the ALU's EX stages. For example, to test the control signal in the EX/M pipeline register, we first flipped one of the four bits to simulate a SPOF and then used the Read/Write Memory test in the RISC-V test suite for validation. We then repeated this error injection and validation for all the remaining three bits in the control signal, with one bit flip each time. The test result is reported in Fig. 9 as the number of errors detected divided by the total number of error injections. Therefore, a value of 4/4 for the control signals means that the error detection rate was 100%.

We then repeated such error injection tests for all the following: (1) the test for pipeline registers (from the third to 11th row in Fig 9) that consist of EX/M and M/WB data and address signal (the fourth column in Fig 9, 134 bits), Control Signals (the second column in Fig 9, 4 bits for EX/M and 2 bits for M/WB), and PC (the third column in Fig 9, 32 bits); (2) the VCID corruption detection test (from the 12th to 13th row in Fig 9) that ensures that the system always stays in sync (6 bits); and (3) the ALU stage registers test (from the 14th to the last row in Fig 9) that ensures all the ALU operational registers are 32 bits wide. Among all these tests, only (2) the VCID corruption detection test was manually validated with ModelSim; all others used the RISC-V ISA test suite with ModelSim. The tests in (1) used one of the tests in the test suite that fits the purpose, and the tests in (3) used all 45 tests from the test suite. Therefore, the value of the total number of error injections in (3) is 1440, or the number of bit flips (32) multiplied by the number of tests (45).

Based on our results in Fig. 9, the Checkers detected all errors, threw the exceptions as expected, and prevented the errors from manifesting in memory, I/O, and system registers. The M Stage Checker captures all the memory and I/O operation failures, and the WB Stage Checker captures all the register writeback operation failures.

Although we only injected faults in the pipeline between the EX and M stages (EX/M) and M and WB stages (M/WB), all data must be propagated through the pipeline registers and synchronized by the system clock in our pipelined architecture.

Therefore, we could have easily injected faults in all pipeline registers and ensured 100% self-checking coverage for stuck-at failures and complete error containment. In the rest of this section, we show the exception simulation of the VCVB, the M-Stage, and the WB-Stage Checkers.

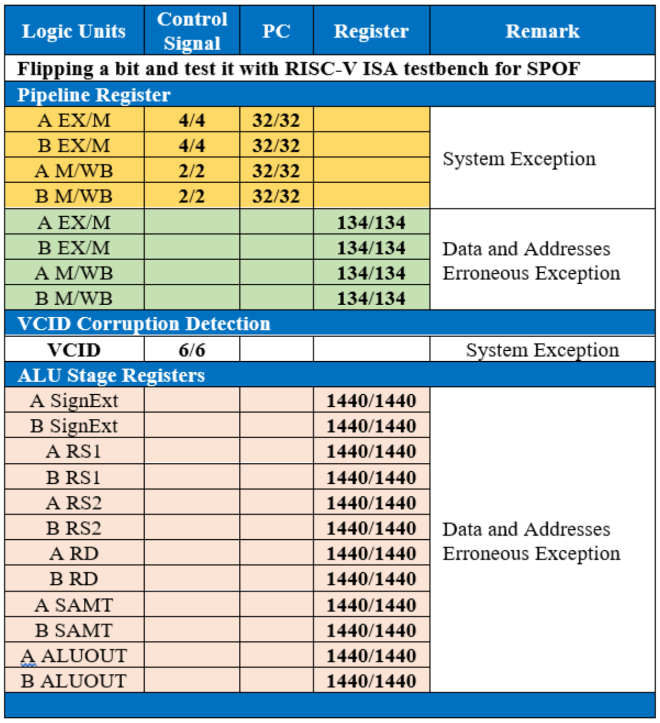
A. *VCVB Checker Validation*

We demonstrate that if VC0 and VC1 run out-of-sequence, the VCVB will detect this and reset the system. This type of problem is a non-recoverable error. Our validation shows the VCVB can detect consecutive 1s' or 0s' or out-of-sequence execution (shown in Fig. 10), where the VCVB throws a system reset (VCID_EXC=1) when consecutive 1s' or 0s' are detected.

B. *M Stage Checker Validation*

The M-Stage Checker checks if the information to be written to memory or I/O devices from the Master and the Slave is equal. If so, the datum is allowed to be written to its destination; otherwise, the Checker throws an exception.

Fig. 11 shows that if both cores' information is the same, they can write the datum to the destination address (VWD_EN=1). Otherwise, the Checker disables the write operation (VWD_EN=0) and throws a system exception (WD_SEXC=1). The Checker also stores both core's information in their respective exception storage area. The exception information allows the lockstep processor to support precise interrupts to resume from the point of exception.



Logic Units	Control Signal	PC	Register	Remark
Flipping a bit and test it with RISC-V ISA testbench for SPOF				
Pipeline Register				
A EX/M	4/4	32/32		System Exception
B EX/M	4/4	32/32		
A M/WB	2/2	32/32		
B M/WB	2/2	32/32		
A EX/M			134/134	Data and Addresses Erroneous Exception
B EX/M			134/134	
A M/WB			134/134	
B M/WB			134/134	
VCID Corruption Detection				
VCID	6/6			System Exception
ALU Stage Registers				
A SignExt			1440/1440	Data and Addresses Erroneous Exception
B SignExt			1440/1440	
A RS1			1440/1440	
B RS1			1440/1440	
A RS2			1440/1440	
B RS2			1440/1440	
A RD			1440/1440	
B RD			1440/1440	
A SAMT			1440/1440	
B SAMT			1440/1440	
A ALUOUT			1440/1440	
B ALUOUT			1440/1440	

Fig. 9. Test results when injecting errors in the EX/M and M/WB pipeline registers and EX stage logical units and the registers. Since our design uses fine-grained multitasking techniques, all data must be propagated through the pipeline registers and synchronized by the system clock. That is, we can easily inject faults on these pipeline registers to ensure a 100% self-checking coverage for stuck-at failures and complete error containment.

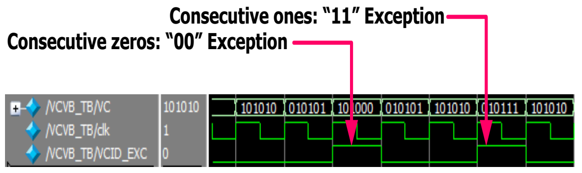


Fig. 10. VCVB detects consecutive 1s' and 0s' when VCIDs are out of sync.

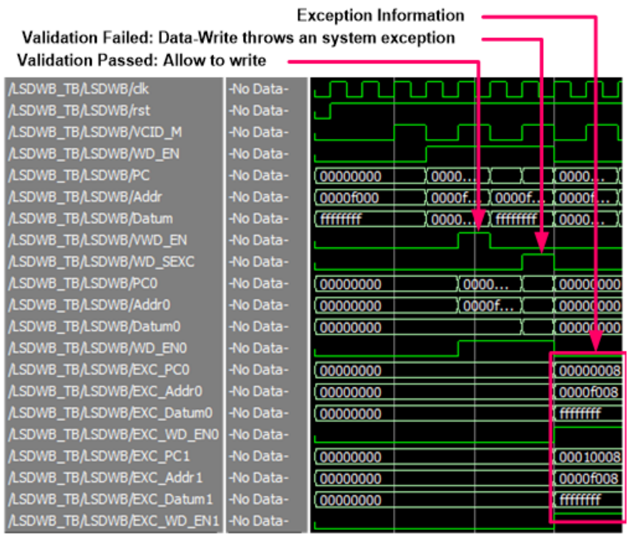


Fig. 11. This gram shows how the M Stage Checker handles matched and unmatched information from both cores.

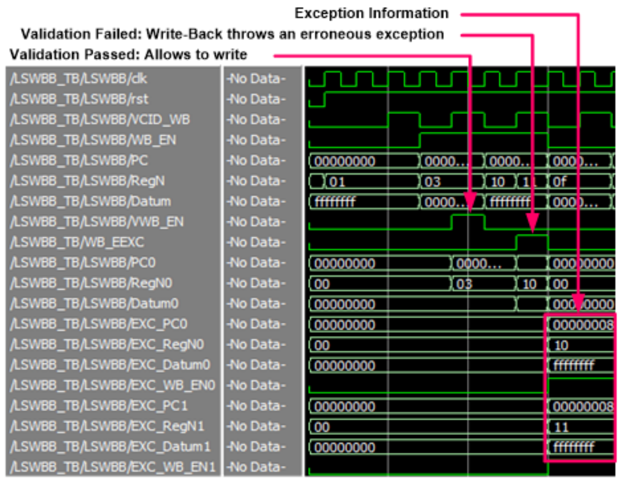


Fig. 12. This ram shows how the WB Stage Checker handles matched and unmatched information from both cores.

C. *WB Stage Checker Validation*

The WB-Stage Checker checks if the information to be written to the system register from the Master and the Slave is equal. If so, the datum is allowed to be written to its destination; otherwise, the Checker throws an exception. Fig. 12 shows if both cores' information is equal, they are allowed to write the datum to the destination register (VWB_EN=0). Otherwise, the

Checker disables the write operation (VWB_EN=0) and throws an erroneous exception (WD_EEXC=1). The Checker also stores both cores' information into their respective exception storage area. Similar to the M stage Checker validation, the exception information also allows the lockstep processor to support precise interrupts.

D. Lockstep Processor Runtime Validation

With all the safety-critical functional blocks integrated into the fine-grained multitasking processors, and creating delay pipelines at both the M and WB stages, we transform the fine-grained multitasking processors into two lockstep processors. To verify if our new lockstep processors can execute real code, we use the industry-standard benchmarks as our validation program. Both benchmarks' performance results show that the lockstep functional blocks work correctly, and ensure that the data is validated before being written to its destination, i.e., memory and register files.

Fig. 13 shows our two benchmarks results. The red bars show the Dhrystones results, and the blue bars show the Coremark results. Coremark benchmark doesn't support compiling optimization level O3. Our design mitigates the control and load-use data hazards, and since we use block RAM in this design validation, there is also no structural hazard. With this configuration, our Dhrystones benchmarks score a 1.961 DMIPS/MHz with O3, and 3.277 Coremark/MHz with O2. These results show that our system can execute one instruction per clock cycle, or has an IPC=1.

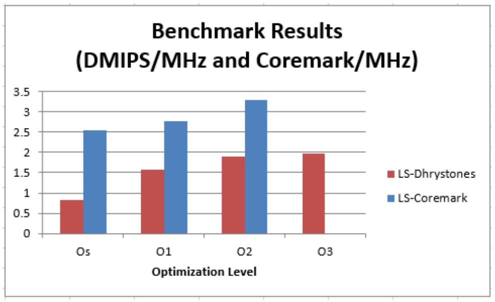


Fig. 13. Our lockstep processor runs both benchmarks, verifies the data before they are saved into the memory; the benchmarks results show that our Lockstep Processors can operate without errors.

V. Related Work

The main drawback of the dual-core lockstep processor, as Kottke and Steininger have pointed out in their research [10], is its vulnerability to CMFs and the existence of residual SPOFs. To cope with CMFs, they proposed a temporal redundancy method by introducing a specific delay element for outgoing data. They performed a systematic analysis and careful layout and introduced self-checking circuits and similar techniques to eliminate SPOFs. For stuck-at faults and complete error containment, their results showed a 100% self-checking coverage. According to the authors, all activated faults were detected within a latency of at most 1.5 clock cycles after they became effective. However, 15,619 of the activated faults could only be detected after a write access. This significant share of 14% late detections is problematic since a faulty Master may have already polluted external components, and as a result, recovery becomes difficult.

Yiu pointed out in his work that several vital technical requirements for high-reliability systems fell into the reduction of the possibility of errors, the support for error detection and correction, robustness—to prevent a SPOF from manifesting into a complete system failure. The author also highlighted other common sources of system failures and classified them into the following categories, such as memory, logic, and software [11].

The ARM Cortex-M33 Dual-Core Lockstep (DCLS) processors cannot detect CMFs since these failures can occur at the same point in both cores, and since there are no differences in the erroneous output, it will cause a false match in the DCLS system. There are several known proposed techniques to resolve CMFs. A common approach is to provide a temporal diversity between the two cores. This approach delays the inputs to the redundant core by several clock cycles. This technique reduces the possibility of having the same point of error in both cores. This method also requires the resynchronization of the output from two cores for comparison.

Another technique is to choose different types of arithmetic logic unit (ALU) block implementation or physical designs to implement the redundant core. This approach keeps the two cores with similar functionality but reduces the risk of failures caused by the same erroneous transient pulse from a power or signal interference [12].

A crucial requirement for the DCLS processors is that all registers in both cores must reset to the same state to guarantee the same initial state. The lack of a system-wide hardware reset prevents a read before write, which can cause a false failure detection. In the general SoC design practice, specific architectural registers are deliberately not reset at POR, to reduce the power consumption and die size. These non-reset registers in the lockstep system might potentially propagate to the output, causing a false mismatch. For the DCLS processors to function correctly, resetting all registers in the processors by either hardware or software is usually needed. The Cortex-M33 processor provides a Reset All Registers (RAR) configuration [13] that must be selected to ensure all registers are properly reset [12].

VI. Discussion and Conclusion

Lockstep processors are used in commercial and military products. They provide error-proof systems to save lives. Although it requires more than twice the amount of hardware to construct a typical dual-core lockstep processor, and inevitability, increasing the production cost, their existence is invaluable. Our research aims to reduce this constraint by using fine-grained multitasking techniques to reduce the die size and the manufacturing cost. Our lockstep design also provides a one-clock-cycle time diversity to enhance the Checker's error detection capability. Instead of producing one lockstep processor like in other existing dual-core system approaches, we achieved two lockstep processors using a two pipelines assembly, thus reducing the hardware cost.

References

[1] E. Bohl, T. Lindenkreuz and R. Stephan, "The fail-stop controller AE11," Proceedings International Test Conference 1997, Washington, DC, USA, 1997, pp. 567-577

DOI: 10.1109/TEST.1997.639665.

[2] I. D. Elliott and I. L. Sayers, "Implementation of 32-bit RISC processor incorporating hardware concurrent error detection and correction," in IEE Proceedings E - Computers and Digital Techniques, vol. 137, no. 1, pp. 88-102, Jan. 1990.

[3] G. Russell and I. D. Elliott, "Design of highly reliable VLSI processors incorporating concurrent error detection/correction," Euro ASIC '91, Paris, France, 1991, pp.316-321.

DOI: 10.1109/EUASIC.1991.212845

[4] M. Pflanz and H. T. Vierhaus, "Online check and recovery techniques for dependable embedded processors," in IEEE Micro, vol. 21, no. 5, pp. 24-40, Sept.-Oct. 2001.

DOI: 10.1109/40.958697.

[5] J. H. Patel and L. Y. Fung, "MULTIPLIER AND DIVIDER ARRAYS WITH CONCURRENT ERROR DETECTION," Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years.', Pasadena, CA, USA, 1995, pp. 268.

DOI: 10.1109/FTCSH.1995.532645

[6] J. Li and E. E. Swartzlander, "Concurrent error detection in ALUs by recomputing with rotated operands," Proceedings 1992 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, Dallas, TX, USA, 1992, pp. 109-116.

DOI: 10.1109/DFTVS.1992.224374.

[7] MC88100 32-Bit RISC Microprocessor Technical Summary, Document MC88100/D, Motorola Inc. 1990.

[8] D. E. Lenoski, "A highly integrated, fault-tolerant minicomputer: the NonStop CLX," *Digest of Papers. COMPCON Spring 88 Thirty-Third IEEE Computer Society International Conference*, San Francisco, CA, USA, 1988, pp. 514-519.

DOI: 10.1109/CMPCON.1988.4921

[9] Rainer Troppmann, Kranzberg (DE); Bernard Fuessl, Moosburg-Aich (DE), Delayed lock-step CPU compare, United States, Patent Application Publication No.: US2008/0244305 A1, October 2, 2008, Texas Instruments Deutschland, GMBH

[10] Kottke, Thomas, and Andreas Steininger. "A Generic Dual Core Architecture with Error Containment," Computers and Artificial Intelligence 23 (2004): 517-535.

[11] Yiu, Joseph, "Design of SoC for High Reliability Systems with Embedded Processors," Embedded World 2015.

[12] ARM, "Application Note Cortex-M33 Dual Core Lockstep," www.arm.com, Version 1.0, 2017.

[13] ARM, "ARM® Cortex®-M33 Processor Integration and Implementation Manual," www.arm.com, 2017.

[14] M. Sim, D. Perera, "A Fast and Secure Pipelined Barrel Processor for Safety-Critical Applications for Real-Time Operating Systems," 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York City, NY, USA 2019, pp 0894-0904.

DOI: 10.1109/UEMCON47517.2019.8992989

[15] M. T. Sim and Q. Yi, "An Adaptive Multitasking Superscalar Processor," *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, Chengdu, China, 2019, pp. 1293-1299.

DOI: 10.1109/ICCC47050.2019.9064185

[16] Gaitan, Nicoleta Cristina; Gaitan, Vasile Gheorghita; Ungurean, Ioan; Zagan, Ionel, "Methods to Improve the Performances of the Real-Time Operating Systems for Small Microcontrollers," in Control Systems and Computer Science (CSCS), 2015 20th International Conference on, vol., no., pp.261-266, 27-29 May 2015Apr. 1993.

[17] E. Dodiu and V. G. Gaitan, "Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers — Concept and theory of operation," *2012 IEEE International Conference on Electro/Information Technology*, Indianapolis, IN, 2012, pp. 1-5.

DOI: 10.1109/EIT.2012.6220705.

[18] I. Zagan and V. G. Gaitan, "Schedulability analysis of nMPRA processor based on multithreaded execution," 2016 International Conference on Development and Application Systems (DAS), Suceava, 2016, pp. 130-134

DOI: 10.1109/DAAS.2016.7492561.

[19] D. May, "XMOS architecture XS1 chips," *2011 IEEE Hot Chips 23 Symposium (HCS)*, Stanford, CA, 2011, pp. 1-30.

DOI: 10.1109/HOTCHIPS.2011.7477496.

[20] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "FlexPRET: A processor platform for mixed-criticality systems," *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Berlin, 2014, pp. 101-110.

DOI: 10.1109/RTAS.2014.6925994