IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 16 June 2026, accepted 13 July 2026, date of publication 22 July 2026, date of current version 29 July 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3716016

RESEARCH ARTICLE

An FPGA-Based Programmable ALPG Platform With Hardware Loop Profiling and a Fully Custom PHY for DRAM Fault Diagnosis

SUNYOUNG PARK[1,2], (Member, IEEE), SAEYEON KIM[1], (Graduate Student Member, IEEE), JIYOUNG LEE[1,3], (Graduate Student Member, IEEE), NAHYEON KIM[4,5], (Graduate Student Member, IEEE), AND JI-HOON KIM[4], (Senior Member, IEEE)

[1]Department of Electrical and Electronic Engineering, Ewha Womans University, Seoul 03760, Republic of Korea
[2]IMEC, 3001 Leuven, Belgium
[3]Synopsys, Seoul 13494, Republic of Korea
[4]Department of Electronic Engineering, Hanyang University, Seoul 04763, Republic of Korea
[5]Korea Electronics Technology Institute (KETI), Gyeonggi-do, Seongnam-si, Republic of Korea

Corresponding author: Ji-Hoon Kim (jhoonkim@hanyang.ac.kr)

This work was supported by SK Hynix Inc.

ABSTRACT The increasing fault rate in DRAM due to aging and advanced nanoscale fabrication exposes critical limitations of traditional testing methods, including Automatic Test Equipment (ATE) and Memory Built-In Self-Test (MBIST). These conventional approaches rely on fixed test patterns and static timing configurations, rendering them ill-equipped to detect timing-sensitive failures such as Write Recovery Errors (WREs) and Variable Retention Time (VRT) faults that emerge throughout a memory's operational lifetime. This paper presents a flexible, FPGA-based Algorithmic Pattern Generator (ALPG) platform designed for programmable and cost-effective DRAM fault diagnosis that *complements*, rather than replaces, ATE for post-manufacturing testing. Building on prior open-source FPGA-based DRAM testing infrastructures that already provide programmable command interfaces and runtime timing control, the platform integrates a custom 32-bit ALPG Instruction Set Architecture (ISA) supporting 17 distinct test patterns that cover 11 fault types, and a custom PHY built directly from FPGA I/O primitives—without modifying any vendor PHY IP—that enables runtime adjustment of JEDEC timing parameters. The principal novel contributions are: (i) a hardware Loop Profiler with three-iteration, basic-block-level loop detection that reduces instruction cycle overhead for loop-intensive test sequences; (ii) the formalization of a Pattern and Timing Space Coverage (PTSC) metric; and (iii) the integration of these into a compact, ASIC-synthesizable ALPG diagnosis engine. Synthesized on a Samsung 28 nm library, the ASIC-synthesizable ALPG core (excluding the FPGA-based custom PHY) maintains a compact gate count of 26.2K, and the PTSC metric—a *structural* measure of the testable pattern-and-timing space—shows a 3.4× improvement over existing ALPG frameworks and is shown to be robust to the normalization choice. The current implementation targets DDR4; extension to LPDDR5 and DDR5, together with empirical in-field, aging, and VRT characterization, is identified as future work.

INDEX TERMS Algorithmic pattern generator, DRAM testing, FPGA, memory fault diagnosis, custom PHY, timing-sensitive faults.

I. INTRODUCTION

The increasing complexity of DRAM technology driven by advanced nanoscale fabrication has introduced significant reliability challenges, including elevated fault rates and timing-sensitive failures that threaten data integrity in modern computing systems [1]. As DRAM cells scale to smaller geometries, reduced cell spacing intensifies cell-to-cell interference and exacerbates timing-sensitive faults such as Write Recovery Errors (WREs) [2], Variable Retention

The associate editor coordinating the review of this manuscript and approving it for publication was Ludovico Minati.

© 2026 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

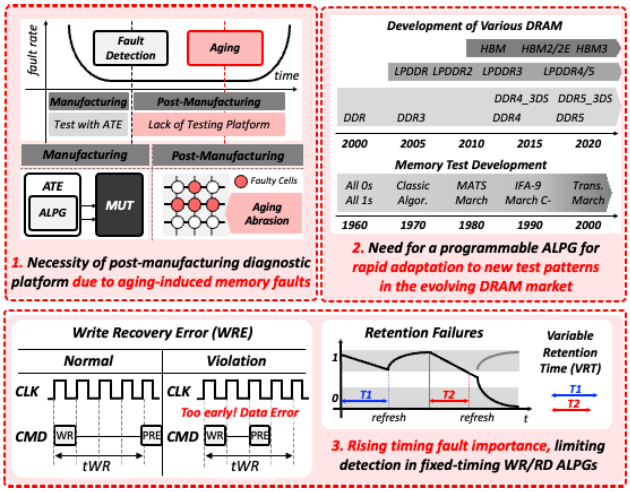


FIGURE 1. Three challenges motivating the proposed platform: (1) rising memory fault rate due to aging exceeds fixed-pattern ALPG coverage; (2) rapid DRAM market evolution demands programmable test generation beyond classical March and Checkerboard patterns; and (3) timing-sensitive faults (WRE, VRT) require fine-grained timing control that static-timing ALPGs cannot provide.

Time (VRT) [3], [4], and read-disturb errors [5]. These faults arise from complex interactions between scaling-induced degradation and retention-time variability, posing risks that conventional testing methods are ill-equipped to address [6], [7], [8]. In safety-critical domains such as automotive systems, where DRAM reliability is paramount [9], [10], [11], the inability to detect such timing-sensitive faults during operation represents a critical gap.

Traditional testing approaches, including Automatic Test Equipment (ATE) and Memory Built-In Self-Test (MBIST), are primarily deployed during post-manufacturing stages and fail to capture faults that emerge over a memory's operational lifetime [12], [13]. While ATE provides comprehensive coverage, it is widely recognized as prohibitively expensive and of limited availability for flexible characterization [14], [15], with high-end memory testers costing on the order of millions of dollars per system [16], [17]. This cost and rigidity motivate FPGA-based testing infrastructures that *complement* ATE by providing programmable, low-cost diagnosis of post-manufacturing and aging-induced faults rather than replacing ATE for manufacturing screening. Conventional Algorithmic Pattern Generators (ALPGs) [18], [19], [20], [21], [22] further compound this limitation by relying on fixed test patterns such as Checkerboard or March algorithms and static timing intervals. These implementations lack the adaptability required to detect timing-sensitive faults. Specifically, their rigid VLIW-based architectures cannot dynamically adjust JEDEC timing parameters such as *tWR* or *tREFI*, making them structurally incapable of timing-sensitive fault detection.

As illustrated in Fig. 1, three core challenges in DRAM testing remain unresolved. First, aging effects cause a progressively rising fault rate that fixed-pattern ALPGs cannot track. Second, the evolving DRAM market demands programmable test generation capable of targeting fault types beyond classical March and Checkerboard patterns. Third, static-timing ALPGs are architecturally incapable of detecting faults that require fine-grained, variable timing control during test execution.

To address these challenges, this paper proposes a programmable FPGA-based ALPG platform for cost-effective DRAM fault diagnosis that complements ATE. Building on open-source FPGA-based DRAM testing infrastructures such as SoftMC [14] and DRAM Bender [15], the platform integrates a custom 32-bit ISA, runtime JEDEC timing control, and a custom PHY, and adds three new elements: (1) a hardware Loop Profiler with three-iteration, basic-block (BB)-level loop detection that optimizes nested loop execution for large-scale testing; (2) the formalization of a Pattern and Timing Space Coverage (PTSC) metric; and (3) the integration of these into a compact, ASIC-synthesizable ALPG diagnosis engine whose custom PHY is built entirely from FPGA I/O primitives, without modifying any vendor PHY IP.

The main contributions of this work are as follows:

- A hardware Loop Profiler with three-iteration, BB-level loop detection that bounds the *processor-side loop-management and instruction-issue overhead* by the per-BB instruction count independently of the test range (e.g., reducing the software instruction-issue overhead from 48.33G to 92 cycles for MATS over a 256 MB address range), enabling large-scale DRAM testing with negligible processor-side overhead.
- A formalized Pattern and Timing Space Coverage (PTSC) metric—a structural measure of the testable pattern-and-timing space—together with a sensitivity analysis showing the reported 3.4× improvement over existing ALPG frameworks is robust to the normalization choice.
- The integration of the above into a compact ALPG core that is ASIC-synthesizable (26.2K gate count on a Samsung 28 nm process, lower than any prior ALPG of comparable or broader fault coverage), validated on an FPGA platform whose custom PHY is built at the FPGA I/O-primitive level without using or modifying the vendor's packaged DRAM PHY IP, packaged as a programmable diagnosis workflow that complements ATE. The current implementation targets DDR4.

A preliminary version of the Loop Profiler concept was presented in [23], where the basic idea of basic-block-level loop detection was introduced in isolation. The present work substantially extends that preliminary study by integrating the Loop Profiler within a complete custom ALPG ISA, a fully custom PHY for runtime JEDEC timing control, the PTSC evaluation framework, and a full FPGA-based platform implementation with fault coverage analysis; together, these elements form an end-to-end DRAM fault-diagnosis platform rather than an isolated profiling mechanism.

TABLE 1. Comparison of existing DRAM testing methods and the proposed ALPG platform.

	ECC	STL	MBIST	ATE	Proposed
Primary Purpose	Runtime Error Correction	Functional Validation	Built-in Memory Testing	Post-Manufacturing Testing	Comprehensive DRAM Testing
Deployment Context	In-field Runtime	Software Runtime	In-product	Manufacturing Equipment	Post-manufacturing
Target Error Type	Soft Errors	Functional Errors	Simple Faults (e.g., stuck-at)	General Faults	Simple Faults, Timing-Sensitive Faults
Flexibility	None	Low	Low	Low	High
Cost	Low	Low	Moderate	High	Low
Ease of Use	High (Automatic)	Require Expertise	Require Expertise	Require Expertise	User-friendly (Programmable)

TABLE 2. Order-of-magnitude platform-cost comparison: High-End ATE vs. proposed FPGA-Based platform.

Cost Factor	High-End ATE	This Work (FPGA)
Hardware acquisition	O($1M) class	O($1K) class (∼$1.7 K) [32]
Test-program NRE	High [17]	Lower; C-level pattern description
Vendor licensing	Required	No ATE-vendor test-program dependency
Field deployment	Not feasible	Feasible for lab/in-line diagnostic use
Runtime timing-parameter control	Limited [14], [15]	User-programmed JEDEC timing sweep

† This is an order-of-magnitude platform-cost comparison only; it excludes the host PC, socket/probing fixtures, handler/prober integration, validation cost, and engineering labor.

The remainder of this paper is organized as follows. Section II reviews the relevant background on DRAM fault mechanisms, existing testing methods, and prior FPGA-based DRAM testing infrastructures. Section III describes the proposed ALPG platform architecture, including the Loop Profiler, the Advanced Timing Manager, and the custom PHY. Section IV presents the implementation results and evaluation, covering loop profiling, fault coverage, the PTSC analysis, and hardware efficiency. Section V concludes the paper and outlines directions for future work.

II. BACKGROUND

A. MEMORY FAULT MECHANISMS AND DETECTION CHALLENGES

Memory faults in DRAM originate from various sources, including manufacturing defects, wear-out over time, and environmental factors such as temperature fluctuations and radiation. These faults are typically classified into three main categories: *permanent faults*, caused by irreversible physical damage to memory cells or peripheral circuitry; *intermittent faults*, which appear sporadically under specific environmental or operational conditions; and *transient faults*, induced by temporary external influences such as cosmic rays or electrical noise. Faults can also be characterized as *latent faults*, which remain undetected until triggered by specific access patterns or environmental conditions, or *active faults*, which immediately manifest as data errors.

From an error manifestation perspective, memory errors are broadly divided into soft errors and hard errors. Soft errors are transient in nature and do not indicate permanent damage, whereas hard errors stem from physical degradation or structural defects and persist until the affected component is replaced or repaired. While conventional Error-Correcting Codes (ECC) are highly effective at detecting and correcting soft errors during runtime, they are inherently limited in addressing timing-sensitive faults such as WREs and VRT, which arise from complex interactions between scaling-induced degradation, retention-time variability, and cell-to-cell interference in modern DRAM devices.

These timing-sensitive faults are particularly problematic because they often escape detection in standard testing flows, remaining latent until specific operational or environmental conditions activate them. Once triggered, they can lead to data corruption, performance degradation, or complete system failure. The growing prevalence of such faults in high-density, high-speed DRAM technologies underscores the urgent need for diagnostic approaches that operate at lower abstraction levels with precise timing control.

B. COMPARISON WITH EXISTING METHODS

Table 1 presents a comparative analysis of traditional DRAM testing methods, including ECC, STL, MBIST, and ATE. Because these methods operate at different abstraction levels and deployment contexts—ECC as runtime correction, STL as software validation, MBIST as in-product test, and ATE as manufacturing equipment—the table separates the *purpose* of each method from its *deployment context*, and positions the proposed platform as *complementary* to ECC, MBIST, STL, and ATE rather than as a direct replacement for any of them.

ECC is effective for detecting and correcting soft errors during runtime, improving memory reliability [24], [25]. However, it is limited to transient faults and cannot detect timing-sensitive failures due to its fixed, hardware-dependent error-correction capabilities.

Software Test Libraries (STL) ensure software-level integrity by verifying functional correctness during operation but lack hardware-specific fault localization, making them unsuitable for detecting low-level faults that require precise timing control [26], [27], [28], [29].

MBIST autonomously diagnoses simple memory faults, particularly in initial testing phases, without requiring external equipment. However, its reliance on fixed test patterns restricts adaptability, limiting its effectiveness for detecting runtime-induced faults [30], [31].

ATE offers accurate fault detection under controlled conditions but, with high-end memory testers costing on the order of millions of dollars per system [16], [17], is impractical for flexible or runtime characterization due to its high cost and dependence on specialized external hardware [14], [15].

In contrast, the proposed platform complements these methods by combining flexible test control with runtime

timing parameter adjustment at low cost. This approach enables efficient handling of loop-intensive and timing-sensitive fault scenarios during post-manufacturing diagnosis, providing programmable adaptability that conventional methods lack while leaving manufacturing-screening coverage to ATE.

Table 2 provides an order-of-magnitude platform-cost comparison. High-end memory ATE systems (e.g., Advantest T5503, Teradyne Magnum) are widely reported to cost on the order of millions of dollars per system and to require substantial NRE for test-program development and qualification; this high cost has itself motivated low-cost FPGA-based test approaches [16], [17] and limits the accessibility of ATE for flexible post-deployment characterization [14], [15]. The proposed FPGA-based platform, built on a Xilinx ZCU104 evaluation board (list price on the order of $1.7 - -1.9\,K$, *i.e.*, *an* $O(1\,\mathrm{K})$-class platform [32]), carries no ATE-vendor licensing or test-program NRE, and test patterns are expressed directly in C code compiled for the on-chip RISC-V core. The platform is therefore best understood as a low-cost diagnostic complement to ATE: it targets post-manufacturing aging and timing-sensitive fault characterization that ATE does not perform in the field, rather than replacing ATE for production screening.

C. FPGA-BASED DRAM TESTING INFRASTRUCTURES

The most directly comparable line of recent work is the body of open-source FPGA-based DRAM testing and characterization frameworks, which provide programmable, instruction-driven control of commodity DRAM.

SoftMC [14] introduced a flexible FPGA memory controller exposing DDR command sequences through a software API, but its hardware loop support is described as future work, leaving loop-intensive sequences to be issued from software.

DRAM Bender [15] extends this direction with a richer instruction set and runtime timing control; however, it builds its physical layer by *modifying the vendor (Xilinx) PHY IP*, which couples the design to a specific vendor IP version.

PiDRAM [33] targets processing-using-memory rather than fault diagnosis and is therefore complementary in scope, while EasyDRAM [34] improves usability through time-scaled evaluation of DRAM techniques.

DRAMscope [13] builds an FPGA-based testing platform—by extending SoftMC and DRAM Bender—that issues programmable DRAM command sequences to reverse-engineer DRAM microarchitecture and activate-induced bitflip (RowHammer/RowPress) characteristics. These platforms share several elements with our design, including a custom instruction set, programmable command sequences, and runtime JEDEC timing adjustment.

Our platform differs in two concrete respects: its custom PHY is built directly from FPGA I/O primitives without modifying any vendor PHY IP, removing the vendor-IP-version dependency present in DRAM Bender; and, in objective,

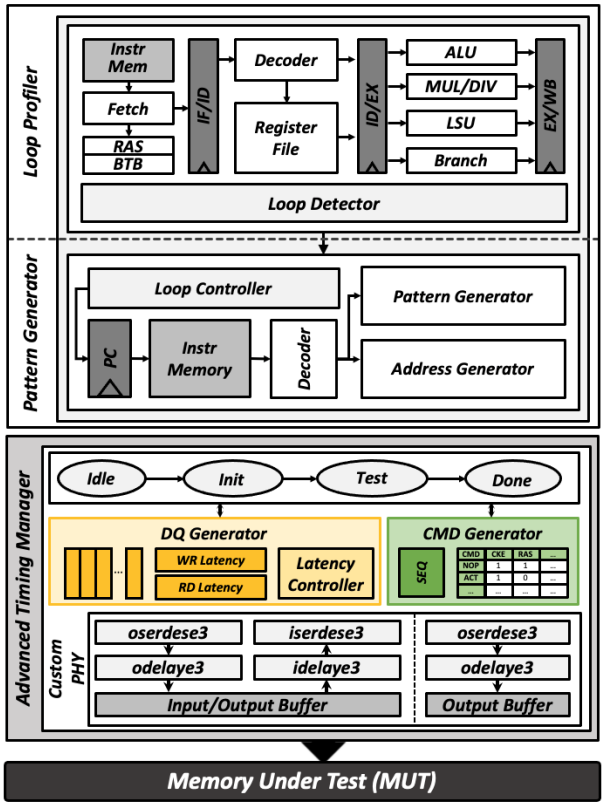


FIGURE 2. Overall architecture of the proposed ALPG platform.

DRAMscope targets microarchitecture reverse-engineering and security-oriented characterization, whereas the proposed platform targets post-manufacturing fault diagnosis with a programmable ALPG and runtime JEDEC timing sweep; the two are therefore complementary in purpose rather than competing. Building on this foundation, the contributions of this work are the Loop Profiler with three-iteration BB-level detection, the PTSC metric formalization, and their integration into a compact, ASIC-synthesizable diagnosis engine. A quantitative comparison against representative frameworks from this line of work is provided in Table 7.

D. LIMITATIONS OF CONVENTIONAL ALPGS

Conventional ALPG implementations have significant limitations in modern DRAM testing due to their reliance on VLIW-like architectures, which prioritize parallel instruction execution for high-speed operation [18], [19]. Although efficient for standard test patterns, this approach lacks flexibility, especially when dealing with timing-sensitive faults.

These VLIW-based ALPGs typically use fixed timing parameters, making it impossible to adjust test conditions dynamically [20]. Additionally, they rely on predefined test patterns, limiting their capability to address emerging fault types [21], [22]. The rigid architecture also struggles with loop-intensive test patterns, leading to instruction overhead and performance degradation [21]. The absence of real-time adaptability is particularly problematic in modern DRAM

TABLE 3. Proposed instruction set architecture (ISA).

Group	Mnemonic	Opcode	Notation	Description
JEDEC	DES	000000	DES Repetition	Instruction Memory ← Opcode, Repetition
	NOP	000001	NOP Repetition	Instruction Memory ← Opcode, Repetition
	PRE	000010	PRE Repetition	Instruction Memory ← Opcode, Repetition
	REF	000011	REF	Instruction Memory ← Opcode
	ACT	000100	ACT	Instruction Memory ← Opcode
Memory	RD	001000	RD Control	Instruction Memory ← Opcode, Pattern[Control]
	WR	001001	WR Control	Instruction Memory ← Opcode, Pattern[Control]
Data	MVI	010000	MVI Rd, Imm	Register[Rd] ← Immediate
	START	010001	START	Start ← 1
	END	010010	END	End ← 1

environments where fine-grained timing variation is essential for accurate fault detection; the quantitative impact of this limitation is captured by the Test Execution Diversity (TED) and PTSC metrics defined in Section IV-C, where conventional ALPGs yield $TED = 0$ owing to their static timing configurations.

In contrast, the proposed RISC-V-based DRAM testing framework addresses these challenges through a loop-aware hardware design that efficiently manages loop-intensive test patterns. By leveraging a custom ISA, the platform significantly reduces executed instruction counts by offloading loop management to dedicated hardware rather than the processor core. This minimizes RISC-V instruction cycles during loop execution while maintaining efficiency and accuracy in detecting timing-sensitive faults. Furthermore, the framework's dynamic JEDEC timing parameter control enables real-time adjustments during testing, overcoming the static nature of conventional ALPGs [2], [4], [35], [36].

III. PROPOSED DRAM TESTING PLATFORM

The proposed ALPG platform, illustrated in Fig. 2, comprises three primary components: the Loop Profiler, the Pattern Generator (PG), and the Advanced Timing Manager (ATM) with a custom PHY. This modular architecture is designed to enhance fault diagnosis and improve memory reliability assessment across a range of DRAM testing scenarios. The Loop Profiler dynamically identifies and optimizes loop patterns within test sequences generated from C code, improving pattern issuance efficiency in the PG. The ATM and custom PHY establish a direct interface between the PG and the Memory Under Test (MUT), facilitating precise timing adjustments and supporting reliable data transfer. These components collaboratively manage critical parameters such as refresh intervals and read/write latencies, enabling the detection of timing-sensitive faults.

The proposed ALPG platform supports test algorithms with diverse time complexities, including $O(n)$, $O(n^2)$, and $O(n\sqrt{n})$, where n represents the number of memory cells in the MUT [37], [38], [39]. The platform's modular architecture enables flexible adaptation to a wide range of test patterns, from simple linear scans to complex nested loops, effectively addressing a broad spectrum of memory faults. By leveraging the ATM, the platform extends beyond traditional test patterns to precisely configure JEDEC parameters such as auto precharge timing, refresh periods, and read latency.

A. ALPG INSTRUCTION SET ARCHITECTURE

The ISA of the proposed ALPG platform, as illustrated in Table 3, is structured into three main instruction types: the JEDEC Type, the Memory Type, and the Data Type. Each type is designed to perform distinct tasks critical for comprehensive DRAM testing and efficient operation control.

The JEDEC Type includes key commands defined by the JEDEC standard, such as Deselect (*DES*), No Operation (*NOP*), Precharge (*PRE*), Refresh (*REF*), and Activate (*ACT*). These instructions manage DRAM maintenance while allowing users to modify refresh periods and precharge timings within test patterns, enabling detailed evaluation of timing-sensitive faults.

The Memory Type provides instructions for direct interaction with the MUT, focusing on precise data operations. This type includes commands such as Read (*RD*) and Write (*WR*), which execute read/write operations and compare results against expected values, ensuring accurate fault detection.

The Data Type acts as a communication bridge between the RISC-V Core and the ALPG, supporting synchronized data transfer and control. Instructions include Move Immediate (*MVI*) for generating non-linear patterns and control signals such as START and END for marking test sequence boundaries, ensuring smooth execution of test patterns.

By integrating the Loop Profiler, the ALPG ISA dynamically optimizes complex testing loops, such as variable refresh intervals for VRT testing, while reducing redundant operations. Unlike conventional VLIW-based ISAs, the proposed platform employs a simplified 32-bit ISA that supports a wide range of test patterns with minimal hardware overhead.

The modular structure of the ALPG ISA allows flexible adaptation to evolving test requirements, reducing the need for hardware redesigns when adding new test patterns. Combined with precise timing control and the ATM, the ISA provides a scalable, programmable solution for addressing modern DRAM reliability challenges.

B. LOOP PROFILER AND PATTERN GENERATOR

Efficient memory testing requires executing write and read operations across large memory regions with varying levels of complexity. As the test area and algorithm complexity increase, loop structures become more intricate, resulting in frequent branch operations that degrade pattern issuance efficiency. To address this challenge, a dedicated hardware solution is essential for optimizing loop execution.

The platform incorporates a dedicated Loop Profiler integrated into a custom RISC-V Core to address the inefficiencies associated with complex loop structures in

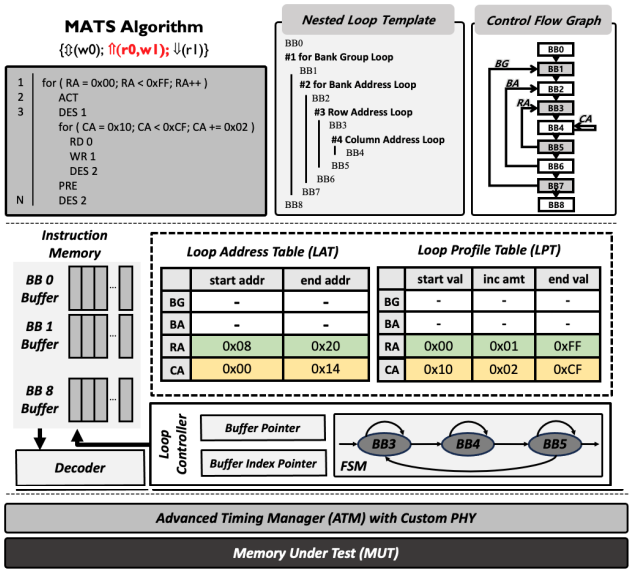


FIGURE 3. Instruction execution flow of loop profiler and pattern generator (PG) with MATS algorithm.

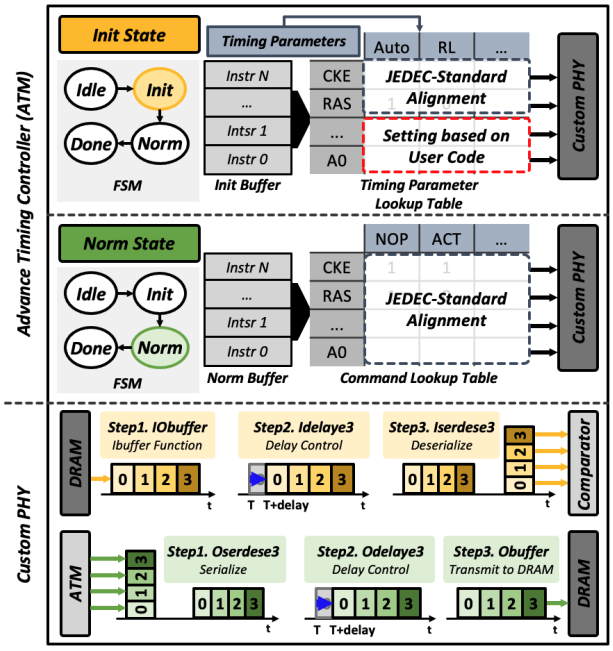


FIGURE 4. Advanced timing manager (ATM) and custom PHY operation flow.

memory testing. The Loop Profiler builds upon our preliminary RISC-V nested loop analyzer presented in [23], and is integrated here within the broader ALPG platform together with the custom ISA, ATM, and PHY presented in this work. Fig. 3 demonstrates the operation of the Loop Profiler and PG using the MATS algorithm. The profiler processes nested loop information comprising four levels: Bank Group (BG), Bank Address (BA), Row Address (RA), and Column Address (CA). The nested loop template is defined through nine Basic Blocks (BB), which collectively describe the structure of all patterns. A Control Flow Graph (CFG) is utilized to select the appropriate BB to execute next, ensuring accurate traversal of the defined loop structure.

The three-iteration detection mechanism operates as follows. In the first iteration, the Loop Detection Unit (LDU) checks for a backward branch to identify a potential loop region. In the second iteration, it verifies that the branch target address is consistent with the first, confirming the loop boundary. In the third iteration, the Loop Profiling Unit (LPU) extracts the iteration parameters (start, step, end values) and asserts a loop-exit signal that redirects the program counter, enabling an early exit from the loop regardless of its total iteration count. Three iterations are the minimum required: one iteration suffices only to detect a backward branch (necessary but not sufficient to distinguish a loop from arbitrary control flow); a second iteration is needed to confirm a consistent branch target (establishing that a true loop boundary exists); and a third iteration is required to extract the step size and end condition, since these can only be determined by observing two consecutive loop-body executions. Fewer than three iterations would leave the loop parameters unextracted; more than three would add overhead without providing additional information about a regular counted loop. This three-iteration process therefore bounds the profiling overhead to a fixed cost independent of the DRAM test range, and its result—the per-BB instruction list together with the loop address table—is passed directly to the PG and instruction memory for pattern generation.

When a user defines a test pattern in C code, the pattern information is transmitted to three primary components: the Loop Address Table (LAT), the Loop Profile Table (LPT), and the Instruction Memory. The LAT stores the start and end addresses for each of the four nested loops, while the LPT maintains parameters such as increment values for loop iterations. For example, in a test scenario targeting Column Address (CA), the LAT specifies a start address of 0×10, an increment value of 0×02, and an end address of 0xCF.

The instruction memory is segmented into nine addresses, each storing instructions corresponding to a specific BB. Execution is managed through a pointer that selects the instruction to execute within each BB. The pointer is controlled by a Finite State Machine (FSM), which monitors when the loop end address is reached and selects the next BB accordingly, ensuring correct loop traversal.

By leveraging this hardware-optimized Loop Profiler and PG, the proposed platform significantly reduces redundant operations and enhances pattern issuance efficiency. This optimization ensures efficient memory testing and scalability for complex loop-intensive algorithms. The end-to-end flow from user-defined C test pattern to fault detection output is summarized in Fig. 5.

C. ADVANCED TIMING MANAGER AND CUSTOM PHY

Fig. 4 illustrates the architecture of the ATM, which configures and manages DRAM timing parameters throughout the testing process. In the initialization state, the ATM configures

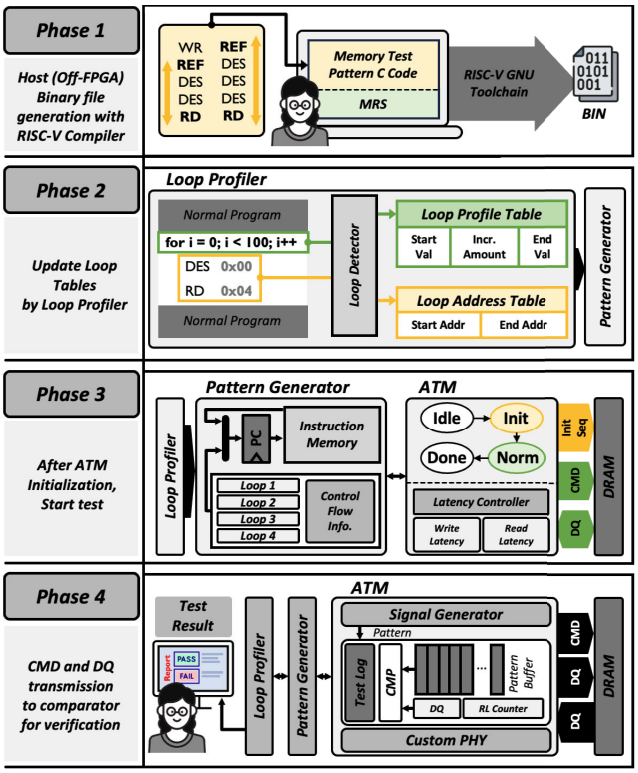


FIGURE 5. Proposed ALPG Platform Flow: From User Input to Fault Detection. Phase 1 (C-code authoring, compilation, and binary generation) executes off-FPGA on the host; Phases 2–4 (ISA decode, test pattern execution via the ATM/PHY, and fault reporting) execute entirely on the FPGA platform.

timing parameters, such as auto precharge and read latency, according to user-defined inputs via Mode Register Settings (MRS). These configurations are managed through a timing parameter lookup table aligned with JEDEC standards and transmitted to the Custom PHY to ensure precise control of DRAM operations.

In the normal state, the ATM operates based on opcodes received from the PG. Using a command lookup table, the ATM maps received opcodes to specific DRAM commands and signals, forwarding them to the Custom PHY. This structured design enhances the platform's flexibility and efficiency in handling diverse fault detection scenarios.

Unlike conventional Xilinx PHY implementations, which restrict read/write operations to specific predefined timing intervals, the Custom PHY enables flexible timing control for precise testing. In contrast to prior FPGA-based DRAM testing infrastructures such as DRAM Bender [15], which achieve runtime timing control by *modifying the vendor (Xilinx) PHY IP*, our Custom PHY is constructed entirely from low-level FPGA I/O primitives. This removes any dependency on a specific vendor PHY IP version and enables the variable read/write latency and retention-time control required for WRE and VRT exercising. This capability ensures that DRAM operations such as reads and writes can be executed exactly at the desired timing instant, which is critical for detecting timing-sensitive faults.

The Custom PHY, implemented using Xilinx primitives, integrates components such as SerDes (*ISERDESE3*, *OSERDESE3*), delay elements (*IDELAYE3*, *ODELAYE3*), and buffers (*OBUF*, *IOBUF*). These elements are carefully configured to ensure precise timing alignment and signal integrity during data transmission, compensating for variations in the DRAM interface.

Incoming data from the DRAM is processed through the *IBUF* of the *IOBUF*, where timing adjustments are applied via *IDELAYE3* primitives before deserialization by the *ISERDESE3* module. The deserialized data is forwarded to the comparator for fault analysis. Outgoing data is serialized through the *OSERDESE3* module, adjusted using *ODELAYE3* primitives, and transmitted back via the output buffer. This structured design ensures precise delay control and synchronization across the DRAM interface.

The integration of the ATM and Custom PHY facilitates efficient and adaptable testing of DRAM modules across different standards, enabling precise fault detection under variable timing conditions. By supporting fine-grained adjustments and removing the timing constraints present in conventional PHY implementations, the platform enhances its ability to detect subtle timing-sensitive faults near specification boundaries. These innovations position the ATM and Custom PHY as integral components of the proposed platform, delivering precise timing control. The architecture is designed to extend to multiple DRAM standards; the current implementation and validation target DDR4.

IV. IMPLEMENTATION RESULTS AND EVALUATION

A. LOOP PROFILING ANALYSIS

The proposed DRAM testing framework leverages loop profiling to bound processor-side instruction-issue overhead within each Basic Block (BB), independent of the test range. This approach eliminates unnecessary repetitions by offloading loop management to dedicated hardware, thereby substantially enhancing execution efficiency. By analyzing the CFG and identifying the optimal execution path, the framework minimizes processor-side loop-management overhead in loop-intensive test patterns.

For a 256 MB DRAM test range, loop profiling reduces the processor-side loop-management and instruction-issue overhead to 92 cycles for the *MATS* algorithm, 112 cycles for *Checkerboard*, and 122 cycles for *Butterfly*. These figures account only for instruction issue on the processor and exclude DRAM access latency and memory-traversal time. In contrast, without loop profiling, a RISC-V core issuing the same loops in software would incur 48.33G instruction-issue cycles for *MATS*, 47.25G cycles for *Checkerboard*, and 29.01G cycles for *Butterfly*—reductions of 5.25×10^8, 4.22×10^8, and 2.38×10^8 times, respectively.

A direct consequence of the BB-level decomposition is that the transmitted pattern buffer contains only BB instructions and loop metadata (size $= BB_{instr} + Loop_{info}$), which is independent of both pattern complexity and memory region size;

TABLE 4. Detectable faults across different test patterns†.

Test Pattern	Detectable Faults	Underlying Fault Mechanism
March C	Stuck-at, Transition, Coupling, Address Decoder Faults	Cell fixed at a logic value or failing to switch; corrupted addressing exposed by alternating up/down marches
March X	Stuck-at, Transition, Coupling Faults	State-transition failures and inter-cell coupling exposed by a reduced march sequence
March Y	Stuck-at, Transition, Coupling Faults	Transition and coupling faults verified with read-after-write in both directions
March LA	Address Decoder, Stuck-at Faults	Linked address-decoder faults exposed by location-aware marches
GALPAT (Galloping)	Pattern-Sensitive, Coupling, Address Decoder Faults	A base cell "gallops" against all others, exposing pairwise coupling and decode errors
Butterfly	Pattern-Sensitive, Coupling Faults	Neighbourhood pattern sensitivity around a victim cell in a butterfly access order
Checkerboard	Stuck-at, Coupling, Pattern-Sensitive Faults	Adjacent-cell interference under a maximally alternating data background
Ping-pong	Data Retention, Coupling Faults	Repeated alternation between two cells stresses retention and coupling
Row Stripe	Address Decoder, Pattern-Sensitive, Coupling Faults	Striped row data exposes row-decoder and bitline-coupling effects
Column Stripe	Address Decoder, Pattern-Sensitive, Coupling Faults	Striped column data exposes column-decoder and coupling effects
Read Disturb	Read Disturb, Coupling Faults	Frequent activation of an aggressor row leaks charge from adjacent rows
Variable Retention	Retention, Data Retention Faults	Cells exhibiting time-varying leakage fail to hold data across refresh intervals
Write Recovery	Write Recovery Faults	Insufficient write stabilization (tWR) before the next command corrupts the write
Read Latency	Read Latency Violation	Data sampled before the read path settles (tCAS/tRCD boundary)
Write Latency	Write Latency Violation	Data driven before the write path is ready (tCWD/tWR boundary)
Activation Timing	Activation Latency Violation	Access issued before the row is fully activated (tRCD/tRAS boundary)
Refresh Timing	Refresh Timing, Data Retention Faults	Extended refresh interval (tREFI) lets stored charge decay below the sense margin

† Rows 1–10 are structural fault-coverage patterns (March, GALPAT, Butterfly, Checkerboard, Ping-pong, Row/Column Stripe) following traditional algorithmic ALPG design. Rows 11–17 (Read Disturb, Variable Retention, Write Recovery, Read/Write Latency, Activation Timing, Refresh Timing) are disturbance/timing-parametric characterization routines implemented as executable ALPG instruction sequences; they exercise access-pattern intensity and/or JEDEC timing boundaries rather than only structural data-path faults, and are included in the 17-pattern total on that basis.

as detailed in [23], this yields up to a 98.6% reduction in pattern buffer size relative to a FIFO-based approach, since conventional FIFO-based PG designs must allocate buffer depth proportional to the consumption–generation mismatch rate, which grows with both pattern complexity and test range.

The software-RISC-V figures above bound the cost of issuing every command from software. Recent FPGA-based DRAM characterization platforms instead expose native loop or repeat instructions, so the relevant comparison against them is not per-command issue avoidance but how the processor-side instruction-issue overhead scales with the test range. Unlike native loop or repeat instructions, which require the loop structure to be explicitly encoded in the command stream, the proposed Loop Profiler automatically extracts

TABLE 5. Timing parameter requirements for advanced DRAM fault detection.

Fault Type	Applicable Timing Parameters
Write Recovery Errors (WREs)	tWR, tWTR, tCCDL
Variable Retention Time (VRT)	tREFI, tRFC, tRAS, tRP
Read Disturb Errors	tRCD, tCAS, tRTP, tRAS
Activation Latency Violation	tRCD, tRAS, tRC, tRRD
Retention Timing Violation	tREFI, tRFC, tRP
Read/Write Latency Errors	tCAS, tRCD, tWR, tRP, tCCD, tCWD, tWTR

BB-level loop metadata from the C-generated RISC-V execution and passes the resulting LAT/LPT information to the Pattern Generator. Its contribution is therefore not merely range-independent loop repetition, but automatic BB-level profiling and its integration with ALPG pattern generation. A direct experimental comparison against native-loop and other programmable FPGA-based DRAM testing frameworks is left as future work.

B. FAULT MODELS AND TEST COVERAGE ANALYSIS

1) FAULT MODELS DETECTABLE BY CONVENTIONAL ALPG

As depicted in Table 4, *stuck-at faults* occur when a memory cell remains fixed at logic high or low due to defects or degradation. They are detected using *March C* and *Checkerboard* patterns, which force alternating logic state transitions.

Coupling faults arise from interference between adjacent cells, causing correlated bit errors. These faults are detected using *GALPAT* and *Butterfly* patterns that evaluate the influence of state transitions in neighboring cells.

Address decoder faults occur when address circuitry errors lead to incorrect memory access. Detection uses *March LA* and *Row Stripe* patterns, which activate addresses sequentially to verify correctness.

Transition faults prevent a cell from switching states due to degradation or voltage issues. These are detected using *March C* and *March X* patterns, which enforce state changes and verify successful transitions.

Pattern-sensitive faults are caused by data patterns in adjacent cells that create mutual interference. Detection uses *Checkerboard* and *Column Stripe* patterns, which systematically analyze dependencies between stored data.

Retention faults occur when a cell fails to hold data for the expected duration, typically due to charge leakage. These faults are identified using retention and disturb patterns that periodically verify data integrity over time.

2) DETECTABLE TIMING-SENSITIVE FAULTS

As shown in Table 5, *write recovery errors* result from insufficient write stabilization time before the next DRAM command. These are detected by dynamically adjusting *tWR*

TABLE 6. Robustness of the PTSC advantage to N_{timing} and $N_{exec,base}$ ($N_{combined} = 17$, $TED = 14$; ratio = $(17/N_{exec,base}) \times (1 + 14/N_{timing})$).

N_{timing}	Advantage vs. ALPG (ratio) for $N_{exec,base}$ =			
	8	9	10	12
10	5.10×	4.53×	4.08×	3.40×
12	4.60×	4.09×	3.68×	3.07×
14	4.25×	3.78×	**3.40**×	2.83×
16	3.99×	3.54×	3.19×	2.66×
18	3.78×	3.36×	3.02×	2.52×

within Write Recovery patterns, exercising the boundary conditions that trigger the fault.

VRT faults manifest as irregular data retention behavior requiring dynamic *tREFI* adjustments via the Custom PHY, which conventional static-timing ALPGs cannot replicate.

Read disturb errors, including row hammer effects, are caused by excessive activations of one row affecting adjacent rows. These are detected using Row Hammer patterns with frequent row activation controlled through adjustable *tRCD* and *tRAS* parameters.

Activation latency violations occur when memory access proceeds without adequate activation time, violating *tRAS*. The Custom PHY and ATM allow real-time adjustment of *tRAS*, *tRCD*, and *tRC* for precise boundary detection.

Retention timing violations occur when prolonged refresh intervals allow data to decay. Testing with retention-aware patterns while varying *tREFI* and *tRFC* allows systematic identification of cells susceptible to this fault.

Read/write latency violations occur when timing specifications (e.g., *tCAS*, *tRCD*, *tWR*) are violated, causing data errors. Real-time ATM adjustment of these parameters improves boundary-condition detection accuracy.

By enabling dynamic timing control, the proposed Custom PHY and ATM extend fault coverage well beyond conventional ALPG methods, particularly for timing-sensitive faults that require exploration of the timing parameter space around JEDEC specification boundaries.

C. PATTERN AND TIMING SPACE COVERAGE ANALYSIS

Conventional DRAM testing evaluates effectiveness using *Pattern Coverage (PC)*, which reflects how well predefined test patterns (e.g., *March*, *GALPAT*) detect faults across DRAM regions. However, PC alone does not account for variations in JEDEC timing parameters and is therefore insufficient for characterizing coverage of timing-sensitive faults such as WREs or VRT.

To address this limitation, we introduce *Pattern and Timing Space Coverage (PTSC)*, a composite metric that integrates both test pattern diversity and timing adaptability. The rationale is that two testing frameworks may support the same set of patterns but differ fundamentally in their ability to explore the timing parameter space; PTSC captures this distinction. PTSC is formulated using the concept of *Test Execution Diversity (TED)*, defined as the total number of distinct JEDEC timing parameters actively used across all supported fault scenarios:

$$TED = \sum_{i=1}^{N} (\text{Number of Valid Timing Parameters for fault } i) \tag{1}$$

where N is the number of supported fault types. The PTSC metric is then defined as:

$$PTSC = \frac{N_{\text{combined}} \times \left(1 + \dfrac{TED}{N_{\text{timing}}}\right)}{N_{\text{exec}}} \tag{2}$$

where N_{combined} is the number of supported test patterns, N_{exec} is the number of patterns supported by the baseline framework, and N_{timing} is the total number of unique JEDEC timing parameters relevant to the fault space. The normalization by N_{exec} ensures comparability against existing works, and the timing factor $(1 + TED/N_{\text{timing}})$ rewards frameworks that actively leverage timing diversity.

In our evaluation, $N_{\text{combined}} = 17$, $N_{\text{exec}} = 10$, $TED = 14$, and $N_{\text{timing}} = 14$. Here, $N_{\text{exec}} = 10$ is set to the number of test patterns uniformly supported across all prior ALPG frameworks [18], [19], [20], [21], [22], ensuring a fair and consistent baseline for comparison. This yields:

$$PTSC_{\text{ALPG}} = 1.0, \quad PTSC_{\text{Proposed}} = 3.4 \tag{3}$$

The 3.4× improvement quantitatively demonstrates the broader structural coverage of the testable pattern-and-timing space enabled by dynamic timing control in the proposed platform. Conventional ALPGs score $PTSC = 1.0$ by construction, as their TED is zero due to static timing configurations. The proposed platform, by supporting 14 unique JEDEC timing parameters across 11 fault types, achieves a PTSC of 3.4—reflecting both a broader pattern set and a richer timing parameter space.

PTSC is a *structural* coverage metric: it quantifies the pattern-and-timing space that the platform *can* exercise, not the physical faults observed on a particular device. The values reported here therefore characterize testable coverage rather than empirical detection rates.

Setting N_{exec} to the ten standard structural patterns supported across prior ALPGs [18], [19], [20], [21], [22] makes those works evaluate to $PTSC = 1.0$. Expanding the ratio algebraically:

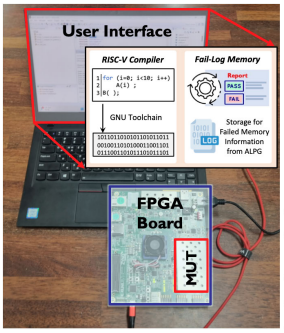
$$\frac{PTSC_{\text{Proposed}}}{PTSC_{\text{ALPG}}} = \frac{N_{\text{combined}} \times (1 + TED/N_{\text{timing}})/N_{\text{exec}}}{N_{\text{exec,base}} \times 1/N_{\text{exec}}} = \frac{N_{\text{combined}}}{N_{\text{exec,base}}} \times \left(1 + \frac{TED}{N_{\text{timing}}}\right) \tag{4}$$

The global normalization denominator N_{exec} cancels exactly, leaving the ratio dependent only on N_{combined} (our pattern count), $N_{\text{exec,base}}$ (the prior-ALPG structural pattern count), TED, and N_{timing}. Consequently, the ratio has two free parameters that warrant sensitivity analysis: N_{timing} and $N_{\text{exec,base}}$. Table 6 sweeps both over their plausible ranges; the advantage remains above 2.5× throughout

TABLE 7. Quantitative comparison against VLIW-Style ALPGs and Open-Source FPGA-Based DRAM testing frameworks.

Metric	[18] ('96)	[19] ('08)	[20] ('23)	[21] ('24)	[22] ('24)	SoftMC [14] (HPCA'17)	DRAM Bender [15] (TCAD'23)	This Work
Implementation	ASIC	ASIC	ASIC	ASIC	ASIC	FPGA	FPGA	ASIC/FPGA
Gate count	56.3K	84.5K	49.3K	21.7K	13.8K	N/A[a]	N/A[a]	26.2K
Supported DRAM Std.	N/A[b]	N/A[b]	N/A[b]	N/A[b]	N/A[b]	DDR3	DDR4	DDR4
Runtime JEDEC timing	X	X	X	X	X	O	O	O
Fully custom PHY	N/A[b]	N/A[b]	N/A[b]	N/A[b]	N/A[b]	X	X[e]	O
HW loop profiling (BB-level)	X	X	X	X	X	X[c]	X	O
Timing-sensitive support	X	X	X	X	X	O	O	O
# Supported faults	5[f]	5[f]	5[f]	5[f]	5[f]	N/A[d]	N/A[d]	11
# Supported patterns	10[f]	10[f]	10[f]	10[f]	10[f]	N/A[d]	N/A[d]	17
TED	0	0	0	0	0	N/A[d]	N/A[d]	14
PTSC	1.0	1.0	1.0	1.0	1.0	N/A[d]	N/A[d]	3.4

[a] FPGA-based infrastructures report LUT/FF utilization rather than ASIC-synthesized gate count and are therefore not directly comparable on this axis.
[b] Not reported (VLIW-style ALPGs are ASIC pattern generators without a configurable FPGA PHY or an explicitly reported DRAM standard).
[c] Hardware loop support is described as future work in SoftMC; loop-intensive sequences are issued from software.
[d] SoftMC and DRAM Bender are programmable substrates: their pattern/fault coverage is whatever is programmed (in principle unbounded), so a fixed-catalog count—and the TED/PTSC derived from it—applies only to the VLIW-style ALPGs [18]–[22], not as a coverage advantage over these frameworks. PiDRAM [33] is excluded (processing-using-memory; Section II-C).
[e] DRAM Bender obtains runtime timing control by *modifying* the vendor (Xilinx) PHY IP, coupling it to a specific IP version. Here "fully custom PHY" denotes a PHY built entirely from low-level FPGA I/O primitives without using or modifying any vendor-packaged PHY IP—the key distinction of this work from DRAM Bender.
[f] Lacking dynamic timing and disturbance/timing-parametric execution support, prior ALPG frameworks cannot exercise the six timing-sensitive fault classes in Table 5 (WRE, VRT, read disturb, activation latency, retention timing, R/W latency) or the routines in rows 11–17 of Table 4, leaving only the five structural fault categories and ten structural patterns (rows 1–10).



FPGA Specifications	
Platform	ZCU104
Clock	250MHz
MUT	DDR4 SODIMM
LUTs	11117
FFs	13567
DSPs	4
LUTRAMs	1591

	LUTs	FFs	DSPs	LUTRAMs
Loop Profiler	1687	1387	4	48
PG	513	240	0	40
ATM + PHY	8394	11317	0	1533

FIGURE 6. FPGA specifications for ALPG platform implementation.

(minimum 2.52×, at $N_{\text{timing}} = 18$ and $N_{\text{exec,base}} = 12$), confirming the result is robust to both normalization choices.

D. HARDWARE IMPLEMENTATION AND EFFICIENCY ANALYSIS

The FPGA implementation of the proposed platform demonstrates efficient resource utilization on the Xilinx Zynq UltraScale+ ZCU104 evaluation board, as shown in Fig. 6. All timing-sensitive fault measurements were performed on a single Micron MTA8ATF51264HZ-2G6B1 DDR4-2666 SODIMM (4 GB, PC4-2666, CL19-19-19) installed in the PL-side DDR4 SODIMM socket of the ZCU104 platform at room temperature. The Loop Profiler utilizes 1,687 LUTs, 1,387 flip-flops, 4 DSPs, and 48 LUTRAMs, efficiently managing loop-intensive test patterns. The PG module requires 513 LUTs, 240 flip-flops, and 40 LUTRAMs, resulting in a compact design for flexible test pattern generation. The Custom PHY, which demands the highest resource utilization due to its timing-control complexity, uses 8,394 LUTs, 11,317 flip-flops, and 1,533 LUTRAMs. The overall platform totals 11,117 LUTs, 13,567 flip-flops, and 4 DSP blocks, demonstrating that broad timing-sensitive fault coverage is achievable with modest FPGA resource overhead.

The proposed platform demonstrates a favorable trade-off between resource efficiency and test coverage, as detailed in Table 7. The ALPG core (excluding the FPGA-based custom PHY) synthesized with the Samsung 28 nm library reports a gate count of 26.2K, which is significantly lower than existing works such as [18] (56.3K), [19] (84.5K), and [20] (49.3K). While recent works like [21] and [22] achieve lower gate counts of 21.7K and 13.8K, respectively, they remain limited in terms of supported fault types and

timing control capability. We further note that these ASIC gate-count figures span different technology generations and implementation contexts; the numerical differences should therefore be read as indicative of relative compactness rather than as a controlled, iso-technology comparison.

Notably, the proposed work supports 11 fault types, including timing-sensitive failures such as WREs, VRT, and read disturb errors (including row hammer effects), compared to the 5 fault types supported across all prior implementations [18], [19], [20], [21], [22]. The 5-fault-type ceiling for prior works follows from their static timing: because conventional ALPGs cannot dynamically adjust JEDEC timing parameters, they are inherently incapable of exercising the six fault classes in Table 5 (WRE, VRT, read disturb, activation latency, retention timing, and R/W latency violations), each of which requires sweeping at least one timing parameter. Prior works are therefore bounded to the five fundamental structural fault categories (stuck-at, transition, coupling, address decoder, and pattern-sensitive) testable without timing control. By the same token, the disturbance/timing-parametric test routines (rows 11–17 of Table 4) are inaccessible to static ALPGs, leaving them with the 10 structural-pattern subset (rows 1–10) that constitutes the standard algorithmic DRAM test suite. It also expands the pattern set to 17, surpassing the structural-only count of 10 in prior studies.

In terms of timing diversity, our framework supports 14 unique JEDEC timing parameters, yielding a TED score of 14, while all previous works report TED scores of 0 due to their static timing configurations. Consequently, the proposed platform achieves a PTSC of 3.4, significantly outperforming the baseline PTSC of 1.0 reported for conventional ALPG-based frameworks [18], [19], [20], [21], [22]. This result highlights that the proposed platform's advantage lies not only in its broader pattern coverage but also in its capability to explore the timing parameter space—a capability absent in prior VLIW-style ALPG implementations—essential for detecting advanced DRAM reliability challenges.

To position the work against the most directly comparable literature, Table 7 additionally includes representative open-source FPGA-based DRAM testing frameworks (SoftMC [14] and DRAM Bender [15]) alongside the VLIW-style ALPGs, reporting LUT/gate usage, supported timing parameters, supported DRAM standards, and supported fault classes. Entries that are not reported or not applicable for a given framework are marked "N/A." PiDRAM [33], whose scope is processing-using-memory rather than fault diagnosis, is discussed in Section II-C but excluded from the quantitative comparison as it is not an ALPG-style testing platform.

V. CONCLUSION

This paper presented a programmable, FPGA-based Algorithmic Pattern Generator (ALPG) platform that addresses critical gaps in conventional DRAM testing, particularly for timing-sensitive failure mechanisms such as Write Recovery Errors (WREs) and Variable Retention Time (VRT) that evade detection by traditional ATE and MBIST approaches relying on fixed patterns and static timing configurations.

Three architectural innovations collectively enable the platform's diagnostic capabilities. First, a custom 32-bit ALPG ISA delivers programmable test execution across 17 pattern types spanning 11 fault categories, accommodating both established and emerging DRAM reliability challenges without hardware redesign. Second, a dedicated hardware Loop Profiler eliminates redundant loop execution at the processor level, bounding the processor-side loop-management overhead by the per-BB instruction count independently of the test range (e.g., reducing the software instruction-issue overhead from 48.33G to 92 cycles for MATS over 256 MB), making large-scale testing practically feasible. Third, an Advanced Timing Manager (ATM) paired with a fully custom PHY breaks the fixed-timing barrier of conventional Xilinx PHY implementations, enabling real-time sweep of 14 JEDEC timing parameters essential for detecting faults that manifest only near timing specification boundaries.

Validated on the Xilinx Zynq UltraScale+ ZCU104 FPGA and synthesized on a Samsung 28 nm library, the ALPG core achieves a gate count of 26.2K—substantially more compact than prior VLIW-style ALPG implementations with comparable or broader fault coverage (49.3K–84.5K in [18], [19], [20]), while additionally supporting timing-sensitive fault detection that those works do not provide. The proposed Pattern and Timing Space Coverage (PTSC) metric provides a *structural* measure of testable pattern-and-timing coverage, under which the platform achieves a 3.4× improvement over existing frameworks that is robust to the metric's normalization choice; PTSC therefore reflects the structural test capability of the platform rather than its measured fault-detection performance. Positioned within the line of open-source FPGA-based DRAM testing infrastructures, the genuinely novel contributions are the Loop Profiler with three-iteration BB-level detection, the PTSC formalization, and their integration into a compact, ASIC-synthesizable ALPG core (26.2K gate count in a Samsung 28 nm library), validated on an FPGA platform whose custom PHY is built at the FPGA I/O-primitive level without using or modifying the vendor's packaged DRAM PHY IP. These results establish the platform as a cost-effective and scalable programmable diagnosis workflow that *complements* ATE for post-manufacturing DRAM fault diagnosis rather than replacing it.

We emphasize that the present evaluation is a single-device proof of concept: it is restricted to DDR4 on a single SODIMM at room temperature and characterizes structural coverage rather than empirical detection rates. Because fault behavior, timing margins, and disturbance characteristics are vendor- and process-specific, the reported results should be read as a demonstration on a single device rather than as a cross-vendor characterization. Future work will extend

validation to additional DRAM standards including LPDDR5 and DDR5, characterize the platform across multiple DDR4 modules from different vendors to assess cross-vendor and process variation, conduct empirical in-field, aging-stressed, and VRT-affected-cell characterization, and augment the timing manager with automated timing sweep to enable statistical characterization of timing margin distributions across diverse operating conditions.

REFERENCES

[1] S. Shiratake, "Scaling and performance challenges of future DRAM," in *Proc. IEEE Int. Memory Workshop (IMW)*, May 2020, pp. 1–3.

[2] U. Kang, H.-S. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, and J. S. Choi, "Co-architecting controllers and DRAM to enhance DRAM process scaling," in *The Memory Forum*, vol. 14, 2014.

[3] M. K. Qureshi, D.-H. Kim, S. Khan, P. J. Nair, and O. Mutlu, "AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015, pp. 427–437.

[4] Y. Liu, D. Wang, P. Ren, J. Li, Z. Qiao, M. Wu, Y. Wen, L. Zhou, Z. Sun, and Z. Wang, "Understanding retention time distribution in buried-channel-array-transistors (BCAT) under sub-20-nm DRAM node—Part II: PBTI aging and optimization," *IEEE Trans. Electron Devices*, vol. 71, no. 8, pp. 4469–4475, Aug. 2024.

[5] J. S. Kim, M. Patel, A. G. Yaglikçi, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An experimental analysis of modern DRAM devices and mitigation techniques," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 638–651.

[6] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng shui of supercomputer memory: Positional effects in DRAM and SRAM faults," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2013, pp. 1–11.

[7] S. Mittal and J. S. Vetter, "A survey of techniques for modeling and improving reliability of computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 1226–1238, Apr. 2016.

[8] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-induced latency variation in modern DRAM chips: Characterization, analysis, and latency reduction mechanisms," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 1, pp. 1–36, 2017.

[9] S. Park, H. Kim, H. Kim, and J.-H. Kim, "PA-2SBF: Pattern-adaptive two-stage Bloom filter for run-time memory diagnostic data compression in automotive SoCs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2024, pp. 1–6.

[10] G. Boschi, E. Spano, H. Grigoryan, A. Kumar, and G. Harutyunyan, "Die-to-die testing and ECC error mitigation in automotive and industrial safety applications," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020, pp. 1–10.

[11] L. Steiner, K. Kraft, D. Uecker, M. Jung, M. Huonker, and N. Wehn, "An LPDDR4 safety model for automotive applications," in *Proc. Int. Symp. Memory Syst.*, Sep. 2021, pp. 1–10.

[12] N. Hajinazar, G. F. Oliveira, S. Gregorio, J. D. Ferreira, N. M. Ghiasi, M. Patel, M. Alser, S. Ghose, J. Gómez-Luna, and O. Mutlu, "SIMDRAM: A framework for bit-serial SIMD processing using DRAM," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2021, pp. 329–345.

[13] H. Nam, S. Baek, M. Wi, M. J. Kim, J. Park, C. Song, N. S. Kim, and J. H. Ahn, "DRAMScope: Uncovering DRAM microarchitecture and characteristics by issuing memory commands," in *Proc. ACM/IEEE 51st Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2024, pp. 1097–1111.

[14] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A flexible and practical open-source infrastructure for enabling experimental DRAM studies," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 241–252.

[15] A. Olgun, H. Hassan, A. G. Yağlikçi, Y. C. Tuğrul, L. Orosa, H. Luo, M. Patel, O. Ergin, and O. Mutlu, "DRAM bender: An extensible and versatile FPGA-based infrastructure to easily test state-of-the-art DRAM chips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 12, pp. 5098–5112, Dec. 2023.

[16] J. Zhang, Z. Liu, X. Hu, P. Liu, Z. Hu, and L. Kuang, "FATE: A flexible FPGA-based automatic test equipment for digital ICs," *Electronics*, vol. 13, no. 9, p. 1667, Apr. 2024.

[17] D. C. Keezer, T. H. Chen, T. Moon, D. T. Stonecypher, A. Chatterjee, H. W. Choi, S. Y. Kim, and H. Yoo, "An FPGA-based ATE extension module for low-cost multi-GHz memory test," in *Proc. 20th IEEE Eur. Test Symp. (ETS)*, May 2015, pp. 1–9.

[18] H. Imada, K. Fujisaki, T. Ohsawa, and M. Tsuto, "Generation technique of 500 MHz ultra-high speed algorithmic pattern," in *Proc. Int. Test Conf. Test Design Validity*, 1996, pp. 677–684.

[19] H. Yoon, M.-H. Yang, Y. Kim, Y. Park, J. Park, and S. Kang, "An effective parallel ALPG using instruction unrolling for high speed memory testing," in *Proc. Int. SoC Design Conf.*, Nov. 2008, pp. I-370–I-375.

[20] K. Deng, H. Wang, R. Zhu, S. Chen, and Z. Dai, "A dedicated memory testing processor design," in *Proc. IEEE 16th Int. Conf. Electron. Meas. Instrum. (ICEMI)*, Aug. 2023, pp. 260–266.

[21] J. Lee, H. Lee, S. Lee, and S. Kang, "A new ISA for high-speed and area-efficient ALPG," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 71, no. 7, pp. 3358–3362, Jul. 2024.

[22] J. Lee, H. Lee, S. Lee, and S. Kang, "A cost-effective per-pin ALPG for high-speed memory testing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 33, no. 3, pp. 867–871, Mar. 2025.

[23] S. Kim, S. Park, N. Kim, J. Lee, and J.-H. Kim, "RISC-V integrated nested loop analyzer for runtime DRAM test pattern generation," *IEEE Embedded Syst. Lett.*, vol. 17, no. 5, pp. 333–336, Oct. 2025.

[24] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 361–372, 2014.

[25] Y. Luo, S. Ghose, T. Li, S. Govindan, B. Sharma, B. Kelly, A. Boroumand, and O. Mutlu, "Using ECC DRAM to adaptively increase memory capacity," 2017, *arXiv:1706.08870*.

[26] N. Ferrante, F. Terrosi, L. Maruccio, F. Rossi, L. Fanucci, and A. Bondavalli, "HUSTLE: A hardware unit for self-test-libraries efficient execution," in *Proc. Int. Conf. Appl. Electron. Pervading Ind., Environ. Soc.* Cham, Switzerland: Springer, 2023, pp. 392–398.

[27] A. Floridia, E. Sanchez, and M. Sonza Reorda, "Fault grading techniques of software test libraries for safety-critical applications," *IEEE Access*, vol. 7, pp. 63578–63587, 2019.

[28] R. Cantoro, F. Garau, P. Girard, N. Kolahimahmoudi, S. Sartoni, M. S. Reorda, and A. Virazel, "Effective techniques for automatically improving the transition delay fault coverage of self-test libraries," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2022, pp. 1–2.

[29] E. Bartocci, R. Bloem, B. Maderbacher, N. Manjunath, and D. Ničković, "Adaptive testing for specification coverage in CPS models," *IFAC-PapersOnLine*, vol. 54, no. 5, pp. 229–234, 2021.

[30] V. Sridharan and D. Liberty, "A study of DRAM failures in the field," in *Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2012, pp. 1–11.

[31] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (SC)*, Jun. 2015, pp. 415–426.

[32] AMD. (2024). *Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit (EK-U1-ZCU104-G)*. [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/evaluation-boards/zcu104.html

[33] A. Olgun, J. G. Luna, K. Kanellopoulos, B. Salami, H. Hassan, O. Ergin, and O. Mutlu, "PiDRAM: A holistic end-to-end FPGA-based framework for processing-in-DRAM," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 1, pp. 1–31, Mar. 2023.

[34] O. Canpolat, A. Olgun, D. Novo, O. Ergin, and O. Mutlu, "EasyDRAM: An FPGA-based infrastructure for fast and accurate end-to-end evaluation of emerging DRAM techniques," in *Proc. 55th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2025, pp. 331–344.

[35] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-latency DRAM: Optimizing DRAM timing for the common-case," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 489–501.

[36] M. Patel, J. S. Kim, and O. Mutlu, "The reach profiler (REAPER) enabling the mitigation of dram retention failures via profiling at aggressive conditions," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 255–268, 2017.

[37] Y. Sfikas and Y. Tsiatouhas, "Testing neighbouring cell leakage and transition induced faults in DRAMs," *IEEE Trans. Comput.*, vol. 65, no. 7, pp. 2339–2345, Jul. 2016.
[38] A. Z. Jidin, R. Hussin, M. S. Mispan, and L. W. Fook, "Novel March test algorithm optimization strategy for improving unlinked faults detection," in *Proc. IEEE Asia–Pacific Conf. Circuit Syst. (APCCAS)*, Nov. 2021, pp. 117–120.
[39] M. G. Karpovsky and V. N. Yarmolik, "Transparent memory testing for pattern sensitive faults," in *Proc. Int. Test Conf.*, 1995, pp. 860–869.

SUNYOUNG PARK (Member, IEEE) received the B.S. degree in electronic and electrical engineering from Ewha Womans University, Seoul, South Korea, in 2019, the M.S. degree from the Digital System Architecture Laboratory, Ewha Womans University, in 2021, and the Ph.D. degree from Ewha Womans University, in 2025. She is currently a Researcher with IMEC, Belgium. Her research interests include safety and reliability for automotive chiplet systems, UCIe, error injection, fault tolerance, runtime test architecture, and functional safety.

SAEYEON KIM (Graduate Student Member, IEEE) received the B.S. degree in electronic and electrical engineering from Ewha Womans University, Seoul, South Korea, in 2022, and the M.S. degree from the Digital System Architecture Laboratory, Ewha Womans University, in 2024. Her current research interests include memory testing, microarchitecture design, and domain-specific accelerators.

JIYOUNG LEE (Graduate Student Member, IEEE) received the B.S. degree in electronic and electrical engineering from Ewha Womans University, Seoul, South Korea, in 2023, and the M.S. degree from the Digital System Architecture Laboratory, Ewha Womans University, in 2025. She is currently with Synopsys, South Korea. Her research interests include memory testing and microarchitecture design.

NAHYEON KIM (Graduate Student Member, IEEE) received the B.S. degree in electronic and electrical engineering from Ewha Womans University, Seoul, South Korea, in 2024, and the M.S. degree from the Digital System Architecture Laboratory, Hanyang University, in 2026. She is currently a Researcher with KETI. Her research interests include memory testing and microarchitecture design.

JI-HOON KIM (Senior Member, IEEE) received the B.S. degree (summa cum laude) and the Ph.D. degree in electrical engineering and computer science from KAIST, Daejeon, South Korea, in 2004 and 2009, respectively. In 2009, he joined Samsung Electronics, Suwon, South Korea, as a Senior Engineer, where he worked on next-generation architecture for 4G communication system-on-chip platforms. From 2018 to 2025, he was a Professor with the Department of Electrical and Electronic Engineering, Ewha Womans University, Seoul, South Korea. Since 2025, he has been with the Department of Electronic Engineering, Hanyang University, Seoul. His current research interests include CPU microarchitecture, domain-specific SoCs, and deep neural networks accelerators.