https://doi.org/10.31436/iiumej.v27i1.3749

FSM-Based MBIST Controller Implementation using the March AZ2 Test Algorithm

AIMAN ZAKWAN JIDIN[1,2*], MOHD SYAFIQ MISPAN[1,2], RAZAIDI HUSSIN[3], AHMAD RIDHWAN MD ZIN[1], TENGKU FARAH NURHIDAYU TUAN AB RAZAK[1]

[1]*Fakulti Teknologi dan Kejuruteraan Elektronik dan Komputer, Universiti Teknikal Malaysia Melaka, 76100 Durian Tunggal, Malaysia*
[2]*Center for Telecommunication Research and Innovation, Universiti Teknikal Malaysia Melaka, 76100 Durian Tunggal, Malaysia*
[3]*Faculty of Electronics Engineering and Technology, Universiti Malaysia Perlis, 02600 Arau, Malaysia*

**Corresponding author: aimanzakwan@utem.edu.my*

(Received: 23 May 2025; Accepted: 4 December 2025; Published online: 12 January 2026)

ABSTRACT: An efficient on-chip memory testing requires the use of a Memory Built-In Self-Test (MBIST) that applies a low-complexity test algorithm that offers excellent fault coverage, to ensure high test quality at a minimal cost. The March AZ2 algorithm, with 14N complexity, was previously established to balance the fault coverage and test complexity. It was previously implemented in an MBIST controller through an automatic generation process using an Electronic Design Automation (EDA) tool, which limits its customizability and optimization potential. This paper presents the implementation of an MBIST controller that employs the March AZ2 test algorithm and is based on a Finite-State Machine (FSM) architecture. The developed MBIST controller was implemented on a Field-Programmable Gate Array (FPGA) to validate its functionality and fault coverage. A comparison with an equivalent MBIST controller automatically generated using Siemens EDA Tessent MemoryBIST tool demonstrates that the proposed FSM-Based MBIST controller achieves 84% lower circuit area and 32% power consumption, while offering similar fault coverage.

ABSTRAK: Pengujian memori atas cip yang cekap memerlukan penggunaan Ujian-Kendiri Terbina-Dalam Memori (MBIST) menggunakan algoritma ujian berkompleksiti rendah dan menawarkan liputan kesalahan yang tinggi, bagi memastikan kualiti ujian baik pada kos minimum. Algoritma March AZ2, dengan kekompleksan 14N, telah diperkenalkan sebelum ini bagi mengimbangi antara liputan kesalahan dan kompleksiti ujian. Ia telah dilaksanakan dalam pengawal MBIST melalui proses penjanaan automatik menggunakan Automasi Reka Bentuk Elektronik (EDA), yang menghadkan tahap kebolehsuaian dan potensi pengoptimuman. Kajian ini membentangkan pelaksanaan pengawal MBIST yang menggunakan algoritma ujian March AZ2, berasaskan seni bina Mesin Keadaan-Terhingga (FSM). Pengawal MBIST yang dibangunkan telah dilaksanakan pada Litar Terpadu-Serba Guna (FPGA) bagi mengesahkan fungsi dan liputan kesalahan. Perbandingan dengan pengawal MBIST yang dijana secara automatik menggunakan Siemens EDA Tessent MemoryBIST membuktikan bahawa pengawal MBIST berasaskan FSM yang dicadangkan mengurangkan keluasan litar sebanyak 84% dan penggunaan kuasa 32% lebih rendah, sambil mengekalkan liputan kesalahan yang sama.

KEYWORDS: *MBIST, March Test Algorithm, FPGA, FSM, and Memory Testing*

https://doi.org/10.31436/iiumej.v27i1.3749

1. INTRODUCTION

Memory Built-In Self-Test (MBIST) has become an indispensable technique for on-chip memory testing, owing to its ability to independently generate the test inputs, apply them to the memory under test, and verify the test responses at the output [1]. Consequently, it is preferred in the industry to minimize both test and overall chip production costs, as reliance on high-performance and expensive external testers has been lessened [2], [3], [4]. Its utilization has become more crucial than ever, since modern chips are predominantly occupied by memories, which can take up as much as 94% of the chip's total area [5], [6], [7]. Thus, the overall chip quality depends significantly on the quality of the memories within that particular chip [6], [8].

The efficiency of an MBIST operation depends on the test algorithm employed, which specifies the sequence of test operations performed on the memory under test. Different test algorithms offer varying levels of complexity and fault coverage. Among them, March-series test algorithms are often preferred owing to their design simplicity and linear complexity [9], [10]. A March test algorithm defines a sequence of test operations, consisting of writing an x logic (wx) to memory cells or reading from memory cells (rx), where $x \in \{0, 1\}$. These test operations may be performed in either ascending address order (from the first to the last memory cell, denoted by ⇑), or descending address order (denoted by ⇓) [11]. In some cases, the direction of the test operations is not important (denoted by ⇕).

Table 1. The complexities and fault coverages of several March test algorithms.

Test Algorithm	Complexity	Fault Coverage						
		Stuck-At Fault (SAF)	Transition Fault (TF)	Deceptive Read Desctructive Fault (DRDF)	Write Destructive Fault (WDF)	TF Coupling Fault (CFtr)	DRDF Coupling Fault (CFdrd)	WDF Coupling Fault (CFwd)
March C- [12]	10N	100%	100%	0%	0%	100%	0%	0%
March CL [13]	12N	100%	100%	50%	0%	100%	25%	0%
March AZ1 [14]	13N	100%	100%	100%	100%	62.5%	75%	75%
March LR [15]	14N	100%	100%	0%	0%	100%	0%	0%
March SR [16]	14N	100%	100%	100%	0%	100%	50%	0%
March C+ [17]	14N	100%	100%	100%	0%	100%	100%	0%
March XR [18]	14N	100%	100%	100%	100%	62.5%	50%	50%
March AZ2 [14]	14N	100%	100%	100%	100%	75%	75%	75%
March-ee [19]	18N	100%	100%	100%	50%	100%	100%	25%
March MSS [20]	18N	100%	100%	100%	100%	100%	100%	100%
March CS [21]	20N	100%	100%	100%	100%	100%	100%	100%
March SS [22]	22N	100%	100%	100%	100%	100%	100%	100%

Table 1 lists several March test algorithms along with their complexities and fault coverages. The complexity is expressed as O (N), where N denotes the number of cells in the memory under test. Additionally, the coverage of Incorrect Read Fault (IRF) and Read Destructive Fault (RDF), which are not listed in Table 1, is similar to SAF coverage because

their detection requirements are identical [20]. SAF, TF, RDF, IRF, DRDF, and WDF are considered Single-Cell Faults (SCFs), in which fault sensitization and detection occur only within the affected victim cell. Meanwhile, CFtr, CFdrd, and CFwd are coupling faults (CFs), also known as Double-Cell Faults (DCFs), because the fault detected in a victim cell is caused by the state or operation of its aggressor cell. Each SCF comprises 2 Fault Primitives (FPs), as faults can occur when the affected cell is in either a low or a high state. Thus, an SCF coverage is determined by dividing the number of detectable FPs by 2. On the other hand, each DCF comprises 8 FPs: a fault in a victim cell may occur when its associated aggressor cell is in either a low or high state, and the aggressor cell can be located either before or after the victim cell within the memory. Therefore, each DCF coverage is calculated by dividing the number of detectable FPs by 8.

Based on Table 1, a minimum complexity of 18N, such as in the case of the March MSS algorithm, is required to achieve 100% coverage of all unlinked static faults in a Random Access Memory (RAM) [20]. To reduce test duration and cost, lower-complexity test algorithms can be used. However, as shown in Table 1, many of these algorithms provide limited or no coverage of several fault types, such as DRDF, WDF, and their associated CFs. These faults are particularly relevant to modern memory technologies, which are manufactured using miniaturized process technologies, and thus, their presence cannot be ignored [6]. The March AZ2 algorithm was introduced to balance test duration and quality. It provides excellent coverage of all targeted unlinked static faults in a RAM with only 14N complexity [14]. Compared with other test algorithms, it offers the best coverage of unlinked static faults while maintaining a complexity below 18N.

A test algorithm can be implemented in an MBIST controller in several ways. The work in [23] developed a microcode-based MBIST controller architecture. In this work, instructions representing the test sequence are stored in a dedicated register in binary format and deployed as the test patterns to be loaded into the controller. However, because it requires an instruction counter and a register to store instructions, it generally occupies a large chip area. Meanwhile, several works have successfully developed the Finite-State Machine (FSM)-based MBIST controllers, providing better test time and area overhead [3], [24]. This architecture comprises several states to represent each test operation, as well as the test initialization and termination.

Additionally, the MBIST controller generation process can be automated using an Electronic Design Automation (EDA) tool. The previous research in [14] demonstrated that the March AZ2 algorithm was successfully applied in an MBIST controller that was generated automatically using Siemens EDA Tessent MemoryBIST software as the EDA tool [25]. The generation process was carried out simply by developing a core description file that describes its test sequence in a format recognizable by the tool used [26]. Despite the process's simplicity and short design time, it imposes limitations on design customizability and optimization potential, as the MBIST controller was generated using the EDA tool's predefined resources and generation algorithm. Consequently, designers have limited options to optimize the circuit's area and speed.

Therefore, this paper presents a new MBIST controller designed using an FSM-based architecture to represent the March AZ2 algorithm's test sequence in hardware. The proposed FSM-based MBIST controller is then incorporated with a 1-kB RAM as the memory under test. The circuit was then implemented on an Intel MAX10 DE10-Lite FPGA Development Board, and a series of tests was conducted to validate its functionality and fault coverage. The main contribution of this research paper is the development of a user-defined FSM-based MBIST controller that operates correctly while offering reduced circuit area and greater

https://doi.org/10.31436/iiumej.v27i1.3749

flexibility and customizability for optimization, compared with an equivalent controller automatically generated by EDA tools.

2. METHODOLOGY

2.1. The March AZ2 algorithm description

⇓(w0);	⇓(w0, r0);	⇑(r0, w1, w1, r1);	⇑(r1, w0);	⇓(r0, w1, w1, r1);	⇓(r1)
ME_0	ME_1	ME_2	ME_3	ME_4	ME_5

Figure 1. The March AZ2 algorithm test sequence.

Figure 1 presents the test operation sequence defined by the March AZ2 algorithm, which is divided into 5 test elements (denoted as ME_0 to ME_5) [14]. In ME_0, all memory cells are initialized to logic 0 in descending order. Next, in ME_1, each memory cell is written to 0 and then read (expecting a 0 at the output), starting from the last to the first cell. After that, each cell is read (expecting a 0 at the output), written to 1 twice, and reread (expecting a 1 at the output) in ME_2, starting from the first to the last memory cell. The test sequence continues in ME_3, where each cell is read (expecting a 1 at the output) and then written to 0 in ascending address order. In ME_4, the same test operation sequence as in ME_2 is carried out, but in descending address order. Finally, all cells are read (expecting a 1 at the output) in ME_5 in descending address order. There are 14 read or write operations in this test algorithm that must be performed on N memory cells, thereby explaining its 14N complexity.

As shown in Table 1, the March AZ2 test algorithm achieves full coverage of SAF, TF, IRF, RDF, DRDF, and WDF fault models, while providing 75% coverage of CFtr, CFdrd, and CFwd. It fails to detect 2 out of 8 FPs for CFtr due to the absence of test operations that induce a high-to-low cell-state transition in descending address order, e.g., ⇳(…, w1); ⇓(r1, w0); ⇓(r0, …) or ⇳(…, w1, r1); ⇓(w0, r0, …). Similarly, it misses 2 out of 8 FPs for CFdrd, as its test sequence lacks two consecutive read operations from a low-state cell in ascending address order, e.g., ⇑(…, r0, r0) or ⇑(…, r0); ⇳(r0, …). For CFwd, the March AZ2 algorithm is unable to detect 2 out of 8 FPs because its test sequence does not include any non-transition write-to-0 operation in ascending address order, such as ⇳(…, w0); ⇑(w0, r0, …) or ⇑(…, w0, w0, r0).

2.2. MBIST Controller FSM development

Figure 2 depicts the 15-state FSM used to represent the March AZ2 algorithm in hardware. It starts in IDLE, the default state, which waits for a high-state en_mbist signal to initiate MBIST. The FSM then transitions to the LOAD state, where the address counter is initialized to the maximum address of the memory under test. It then commences the test in the M0_W0_D state, with the go flag asserted, indicating that the test is in progress. In this state, all memory cells are written to 0 in descending address order, as described in ME_0 of the March AZ2 algorithm's test sequence.

Once all memory cells are written to 0, indicated by a high-state *fin_cnt* input, the FSM proceeds to the **M1_W0_D** and **M1_R0_D** states to perform the w0 and r0 operations, respectively, on all cells in descending address order, as defined in ME_1. It remains in these two states until a high-state *fin_cnt* is received, indicating that the operations have been completed to all memory cells. Next, to perform the test operation sequence defined in ME_2, the FSM proceeds to the **M2_R0_U** state for one clock cycle to perform the r0 operation, then

https://doi.org/10.31436/iiumej.v27i1.3749

to the **M2_W1W1_U** state for two clock cycles to perform the w1w1 operation, and finally to the **M2_R1_U** state to carry out the r1 operation. It loops through these states until the last memory cell is reached, as indicated by a high-state *fin_cnt* input. Specifically, in the **M2_W1W1_U** state, the *en_cnt_twice* flag is asserted to count for two clock cycles (indicated by a high *twice_det* signal) to ensure that the w1 operation is performed twice in this state.

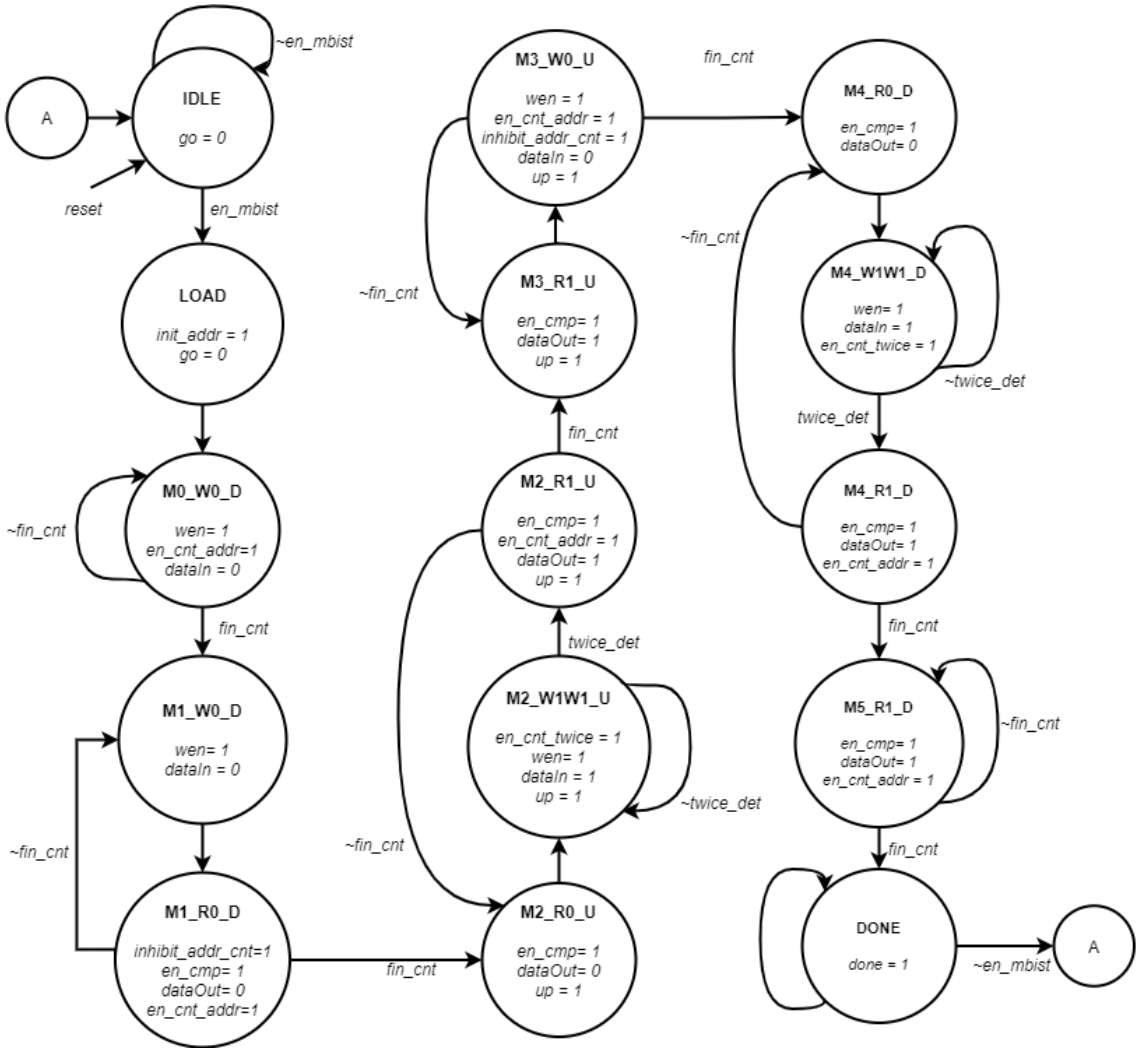

Figure 2. The state diagram for the FSM-based MBIST controller employs the March AZ2 as the test algorithm.

Next, the FSM proceeds to the **M3_R1_U** and **M3_W0_U** states to perform the r1 and w0 operations, respectively, as defined in ME_3. It remains in the same state loop until a high-state *fin-cnt* input is received, indicating that the last memory cell has been reached. After that, to perform the test operation sequence defined in ME_4, the FSM proceeds to the **M4_R0_D** state for one clock cycle to perform the r0 operation, the **M4_W1W1_D** state for two clock cycles to perform the w1w1 operation, and the **M4_R1_D** state to carry out the r1 operation. It remains in the same state loop until the first memory cell is reached, as indicated by a high *fin_cnt* input. Finally, the r1 operation, as defined in ME_5, is executed by proceeding to the **M5_R1_D** state. It remains in the same state until the first memory cell is reached. Once complete, the FSM transitions to the DONE state to terminate the MBIST operation, signified by the assertion of the *done_mbist* flag. It remains in this state until the *en_mbist* signal is low, to avoid unnecessary repetition of the MBIST operation.

https://doi.org/10.31436/iiumej.v27i1.3749

To enable the writing operation, the *wen* flag is asserted during the **M0_W0_D**, **M1_W0_D**, **M2_W1W1_U**, **M3_W0_U**, and **M4_W1W1_D** states, together with the appropriate *dataIn* value (0 for w0, 1 for w1). Meanwhile, in all states that require read operations, the en_cmp flag is asserted to enable comparison between the data read from the memory cell and the expected data, which is determined by the value of the *dataOut* signal. Additionally, in the final state for each test element (**M0_W0_D** for ME_0, **M1_R0_D** for ME_1, **M2_R1_U** for ME_2, **M3_W0_U** for ME_3, **M4_R1_D** for ME_4, and **M5_R1_D** for ME_5), the *en_cnt_addr* flag is asserted to enable counting up (signaled by a high-state *up* flag) or counting down of the address counter, so that all the test operations defined in each test element can be performed on all memory cells. A high-state *fin_cnt* is produced by the address counter when it finishes counting up (when the *up* flag is high) or down, allowing the MBIST operation to proceed to the next test element.

Furthermore, the *inhibit_addr_cnt* flag is set to high during the **M1_R0_D** and **M3_W0_U** states since their subsequent states (**M2_R0_U** and **M4_R0_D**, respectively) have the opposite address direction. By setting this flag high, the address counter maintains its final count value (the last memory cell's address when *up* is high, otherwise 0) upon completing the counting process. Hence, at the beginning of the next state, the address counter can directly start counting down from the address of the last memory cell (in descending address order) or counting up from 0 (in ascending address order).

2.3. Incorporating the proposed MBIST Controller with the memory under test

The functional block diagram presented in Figure 3 shows the integration of the proposed MBIST controller with the memory under test (MUT). It is connected to other submodules, including the data generator, the address counter, and the two-clock counter. The data generator produces 8-bit data to be written to the MUT (write_data) when the *wen* signal is high, or the expected data to be read from the MUT (*exp_data*) based on the received *dataOut* value.

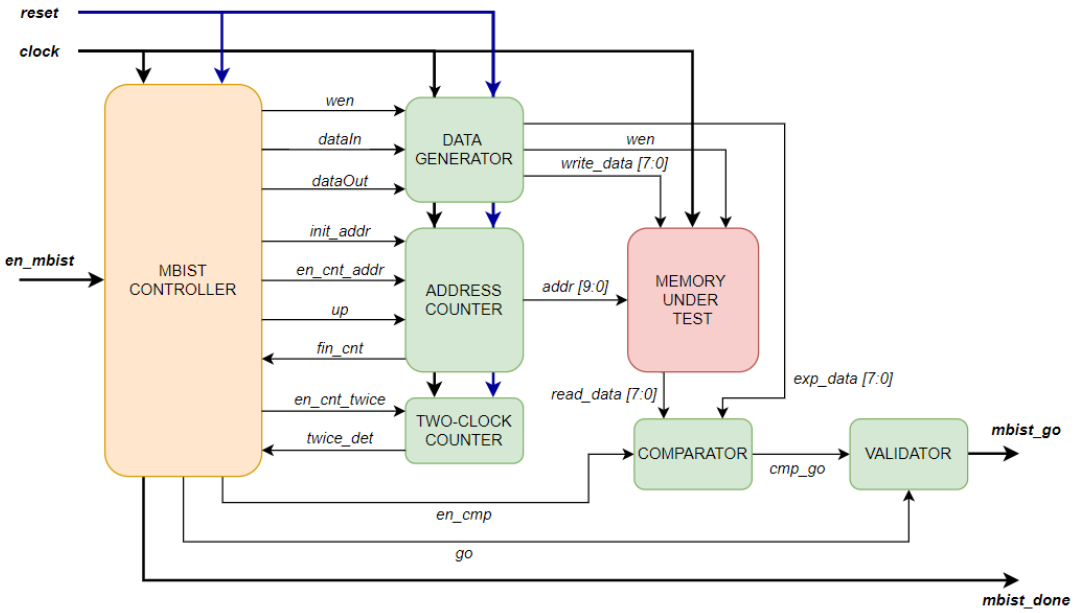

Figure 3. The block diagram shows the integration of the proposed MBIST Controller with the memory under test.

Meanwhile, the address counter initializes the address (*addr*) value to the MUT's maximum address (1023, since a 1-kB RAM is used as the MUT) upon receiving a high

https://doi.org/10.31436/iiumej.v27i1.3749

init_addr signal from the MBIST controller. When a high en_cnt_addr signal is received, the counter counts down (when the up signal is low) or up (when the up signal is high). Upon reaching the minimum memory address (when up equals 0) or the maximum address (when up equals 1), it asserts the fin_cnt signal, which is sent to the MBIST controller to indicate the end of address counting for a particular test element. The generated *addr* is then used to access the MUT's cells. Additionally, the two-clock counter is used to count the number of times the same test operation is executed over two consecutive clock cycles. It is enabled only during the **M2_W1W1_U** and **M4_W1W1_D** states, in which the w1 operation must be performed twice consecutively.

When *wen* is high, the generated *write_data* is written to the MUT cells, whose locations are accessed using *addr*. In contrast, during the read process (when *wen* is low), the data read from the MUT cells (*read_data*) are compared with the expected data (exp_data) generated by the data generator in a comparator when the *en_cmp* signal is high. The comparator produces a high-state *cmp_go* signal as long as there is no error or discrepancy between *read_data* and *exp_data*. Finally, the test validator performs a logical AND operation between the *cmp_go* and *go* signals received from the MBIST controller to produce the *mbist_go* signal at the output. When no errors are detected during MBIST operation, mbist_go should remain in the high state until the end of the test, indicated by a high *mbist_done* output.

2.4. The proposed MBIST Controller validation

The functionality and fault coverage of the proposed MBIST controller were evaluated using tests on the Intel Max10 DE-10 Lite FPGA Development Board, as shown in Figure 4. The proposed circuit, shown in Figure 3, was implemented on the target FPGA using Intel Quartus Prime Design software. The built-in 50 MHz clock oscillator served as the clock source, and an onboard pushbutton was used as the *reset* input. Additionally, a slide switch was used as the *en_mbist* input to start and stop the MBIST operation. Furthermore, a SignalTap logic analyzer core was integrated with the implemented circuit in the FPGA to observe the test outputs.

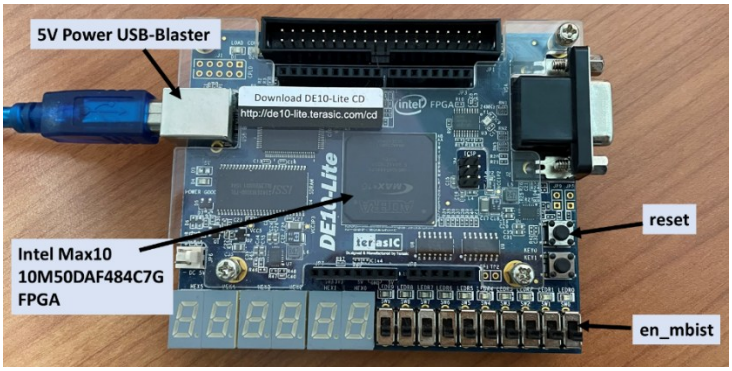

Figure 4. The configuration of the proposed MBIST controller in the Intel Max 10 DE10-Lite FPGA Development Board.

The first test was conducted by running the MBIST operation on a fault-free memory model as the MUT to evaluate its functionality. A high *mbist_go* is expected at the output when the *mbist_done* is asserted, indicating no mismatch between the expected data and the data read from the memory. Additionally, the overall test time was measured from the start (when mbist_go is asserted) to the end (when *mbist_done* is asserted). A test time of 286,720 ns is anticipated, as the proposed MBIST controller uses the March AZ2 test algorithm with 14N complexity (N = 1024 for a 1-kB RAM), with a clock signal period of 20 ns.

https://doi.org/10.31436/iiumej.v27i1.3749

Next, the second test was conducted by running the MBIST operation on a fault-inserted memory model as the MUT to evaluate fault coverage. At this stage, the memory model characteristic was purposely modified to replicate the occurrence of each targeted fault, with all affected victim cells and their aggressor cells randomly chosen [27]. In this case, errors are expected to be detected, and hence, *mbist_go* is likely to be in the low state when the *mbist_done* is asserted. Additionally, 36 fault-detection flags were incorporated to indicate the detectability of the corresponding faults during testing. Each flag is asserted when its corresponding fault is detected. Therefore, the fault coverage can be derived at the end of the test by computing the number of high-bit detection flags. The process of creating and setting the bit value of each detection flag during the MBIST operation is depicted in Figure 5.

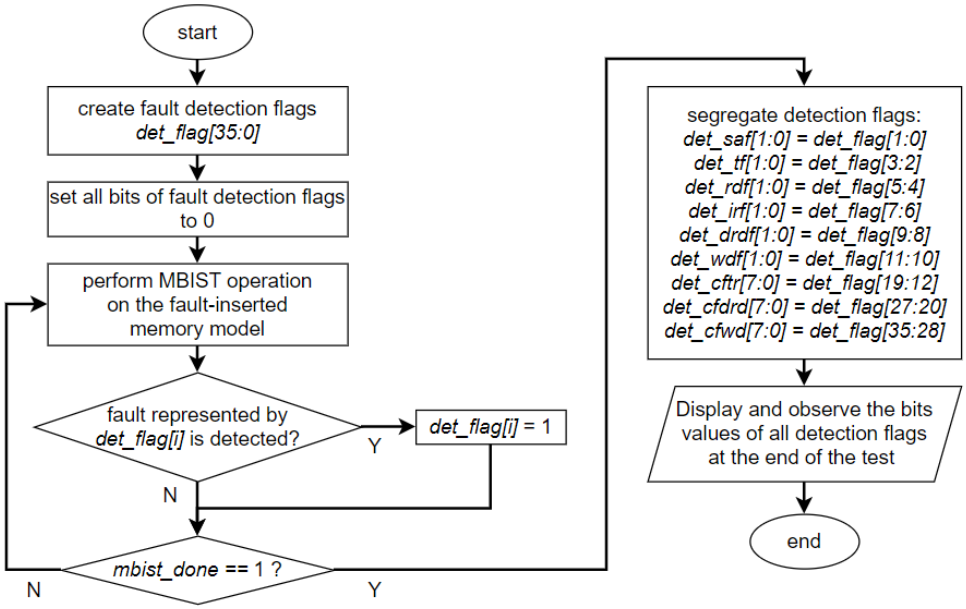

Figure 5. The process flow of fault detection analysis during the MBIST operation on the fault-inserted memory model.

3. RESULTS AND DISCUSSIONS

Figure 6 presents the output observed in the integrated SignalTap logic analyzer core from the MBIST operation performed on the fault-free MUT. No discrepancies were found between the data read from the MUT and the expected data, as the *mbist_go* output stays high until the end of the test, signaled by a high *mbist_done* signal. Additionally, the *cycle_count* signal indicates that the test lasted 14,336 *clock* cycles from start to finish. Given a clock period of 20 ns, the measured test time is 286720 ns, which matches the expected test duration for an MBIST operation applying the March AZ2 test algorithm. Hence, the functionality of the proposed MBIST controller is validated, as it successfully generates the correct test operation sequence.

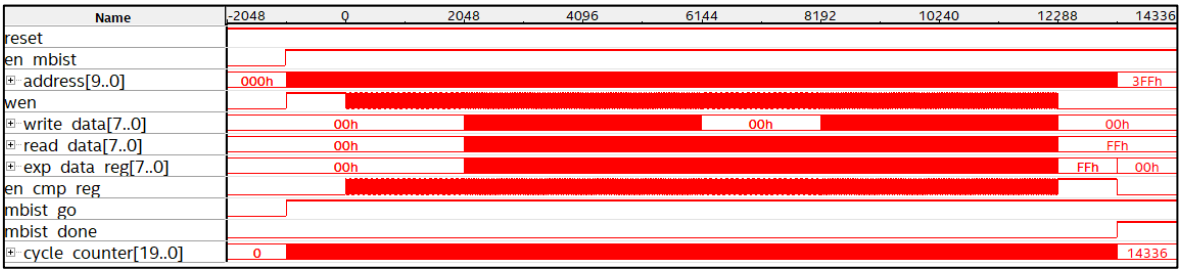

Figure 6. Observation in the SignalTap logic analyzer of the output produced from the experimental test on the fault-free MUT.

https://doi.org/10.31436/iiumej.v27i1.3749

Meanwhile, Figure 7 shows the MBIST output for the fault-inserted MUT, as observed using the integrated SignalTap logic analyzer. In this test, *mbist_go* is deasserted, as expected, when the first fault in the MUT is detected, and it remains at the low level until the end of the test. The values of all fault detection flags are recorded at the end of the test and are presented in Table 2. The fault coverage of the proposed FSM-based MBIST controller implemented on the FPGA device is derived and confirmed to match its expected coverage: SCF, TF, RDF, IRF, DRDF, and WDF are fully detected, whereas CFtr, CFdrd, and CFwd are covered at 75%. Therefore, the conducted test on the FPGA board successfully validates its fault coverage.

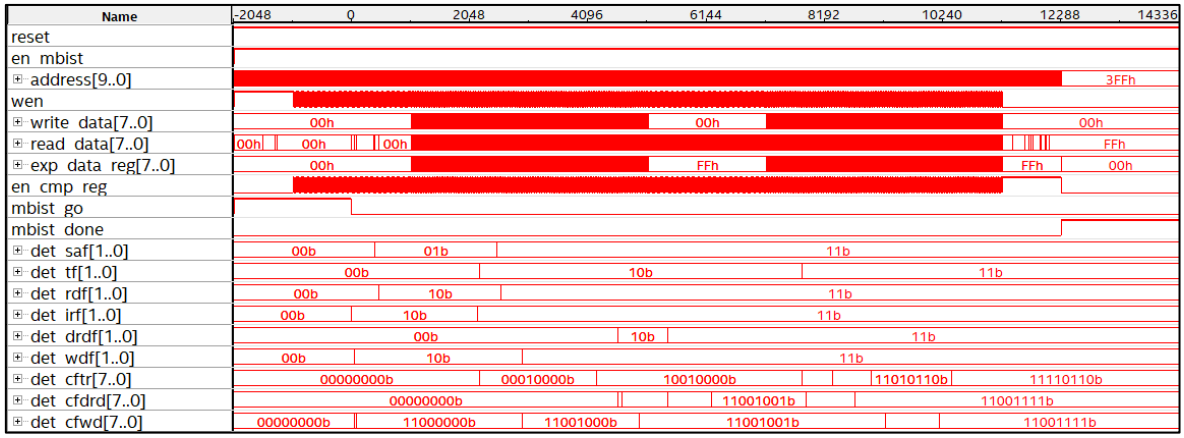

Figure 7. Observation in the SignalTap logic analyzer of the output produced from the experimental test on the fault-inserted MUT.

Table 2. Observed fault detection flag values in the test and derived fault coverages.

Detection Flag	Corresponding Fault	Observed Value	Derived Fault Coverage	Expected Fault Coverage [14]
det_saf	SAF	11_2	2/2 (100%)	2/2 (100%)
det_tf	TF	11_2	2/2 (100%)	2/2 (100%)
det_rdf	RDF	11_2	2/2 (100%)	2/2 (100%)
det_irf	IRF	11_2	2/2 (100%)	2/2 (100%)
det_drdf	DRDF	11_2	2/2 (100%)	2/2 (100%)
det_wdf	WDF	11_2	2/2 (100%)	2/2 (100%)
det_cftr	CFtr	11110110_2	6/8 (75%)	6/8 (75%)
det_cfdrd	CFdrd	11001111_2	6/8 (75%)	6/8 (75%)
det_cfwd	CFwd	11001111_2	6/8 (75%)	6/8 (75%)

Table 3. The performance of the implemented MBIST Controller circuit in FPGA.

MBIST Controller	FPGA Logic Element (LE)	Maximum Frequency (F_{max})	Power Consumption
Automatic EDA generated	580	112.2 MHz	153.05 mW
Proposed FSM-Based	92 (-84%)	89.3 MHz (-20%)	103.4 mW (-32%)

Table 3 presents the proposed MBIST controller's circuit area, speed, and power consumption. The circuit area is measured based on the utilization of logic elements (LEs) in the FPGA, as reported by the fitter. At the same time, speed is determined by its maximum clock frequency (F_{max}) calculated in the static timing analysis. Power consumption is assessed through the report generated during the power analysis. To evaluate its performance, the proposed FSM-based MBIST controller is compared to its equivalent version, which was

automatically generated using Siemens EDA Tessent MemoryBIST and implemented on the same FPGA development board. The results show that the proposed FSM-based MBIST controller utilizes fewer logic elements (84% less than those used by the automatically generated by the EDA), resulting in a smaller circuit area. Additionally, it consumes 32% less power than the automatically generated MBIST controller. However, based on the static timing analysis reports, the proposed FSM-based MBIST controller has a 20% lower F_{max} than its equivalent, suggesting that the latter may operate at a clock frequency above 89.3 MHz. Overall, this comparison highlights the superiority of the proposed FSM-based MBIST controller in terms of circuit area and power consumption. Despite having a lower F_{max}, it shall operate correctly using the 50 MHz onboard clock on the FPGA development board used.

4. CONCLUSION

This paper presents the design, implementation, and validation of a user-defined FSM-based MBIST controller that applies the March AZ2 test algorithm for efficient and comprehensive memory testing. The proposed controller addresses the limitations of automatically generated MBIST solutions from EDA tools by offering more flexibility to reduce circuit area and improve performance. The proposed MBIST controller, implemented using a 15-state FSM architecture, successfully replicates the March AZ2 test sequence in hardware while balancing fault coverage and test complexity. It was successfully implemented on the Intel MAX10 DE10-Lite FPGA Development Board. Experimental validations on both fault-free and fault-inserted memory models confirm the controller's functional correctness and its ability to fully detect memory faults such as SAF, TF, RDF, IRF, DRDF, and WDF, with 75% coverage of targeted coupling faults: CFtr, CFdrd, and CFwd. Moreover, a comparative analysis demonstrates that the proposed FSM-based design outperforms its automatically generated equivalent using the Siemens EDA Tessent MemoryBIST tool by utilizing 84% fewer logic elements and consuming 32% less power. Further analysis can be conducted to optimize the FSM by minimizing the number of states, thereby reducing the circuit area while preserving the same fault coverage.

ACKNOWLEDGEMENT

The authors extend their appreciation to the Ministry of Higher Education Malaysia for the financial assistance provided through the FRGS/1/2024/TK07/UTEM/02/17 research grant and Universiti Teknikal Malaysia Melaka for its support of this research.

REFERENCES

[1] K. Wojciechowski, Andrzej A. Marcinek and W. A. Pleskacz, “Configurable MBIST Processor for Embedded Memories Testing,” in 2019 MIXDES-26th International Conference" Mixed Design of Integrated Circuits and Systems", Rzeszow: IEEE, 2019, pp. 341–344. doi: https://doi.org/10.23919/MIXDES.2019.8787161.

[2] R. Wanga, Z. Huangb, G. Caib, and Z. Yua, “A Built-In Self-Test Circuit Based on March FRDD Algorithm for FinFET Memory,” in Industrial Engineering and Applications: Proceedings of the 10th International Conference on Industrial Engineering and Applications (ICIEA 2023), April 4-6, 2023, Phuket, Thailand, 2023, p. 437. doi: http://dx.doi.org/10.3233/ATDE230069.

[3] T. S. N. Kong et al., “An Efficient March (5n) FSM-Based Memory Built-In Self Test (MBIST) Architecture,” in 2021 IEEE Regional Symposium on Micro and Nanoelectronics (RSM), 2021, pp. 76–79. doi: 10.1109/RSM52397.2021.9511602.

[4] Q. Khasawneh, "Reducing the Production Cost of Semiconductor Chips Using (Parallel and Concurrent) Testing and Real-Time Monitoring," Southern Methodist University, 2019. [Online]. Available: https://scholar.smu.edu/engineering_electrical_etds/32

[5] O. S. Nisha and K. Sivasankar, "Architecture for an efficient MBIST using modified march-y algorithms to achieve optimized communication delay and computational speed," Int. J. Pervasive Comput. Commun., vol. 17, no. 1, pp. 135–147, Jan. 2021, doi: https://doi.org/10.1108/IJPCC-05-2020-0032.

[6] G. P. Acharya, M. A. Rani, G. G. Kumar, and L. Poluboyina, "Adaptation of March-SS algorithm to word-oriented memory built-in self-test and repair," Indones. J. Electr. Eng. Comput. Sci., vol. 26, no. 1, pp. 96–104, 2022, doi: 10.11591/ijeecs.v26.i1.pp96-104.

[7] D. Jamal and R. Veetil, "Efficient MBIST Area and Test Time Estimator Using Machine Learning Technique," in 2023 36th International Conference on VLSI Design and 2023 22nd International Conference on Embedded Systems (VLSID), 2023, pp. 223–228. doi: 10.1109/VLSID57277.2023.00054.

[8] P. Ramakrishna, T. Vamshika, and M. Swathi, "FPGA Implementation of Memory Bists using Single Interface," Int. J. Recent Technol. Eng., vol. 9, no. 3, pp. 55–58, 2020, doi: 10.35940/ijrte.b3975.099320.

[9] I. Mrozek, N. A. Shevchenko, and V. N. Yarmolik, "Universal Address Sequence Generator for Memory Built-in Self-test," Fundam. Informaticae, vol. 188, no. 1, pp. 41–61, 2022, doi: 10.3233/FI-222141.

[10] S. Martirosyan and G. Harutyunyan, "An Efficient Fault Detection and Diagnosis Methodology for Volatile and Non-Volatile Memories," in 2019 Computer Science and Information Technologies (CSIT), 2019, pp. 47–51. doi: 10.1109/CSITechnol.2019.8895189.

[11] L.-T. Wang, C.-W. Wu, and W. Xiaoqing, VLSI Test Principles and Architectures: Design for Testability. Elsevier, 2006.

[12] A. J. Van De Goor, "Using march tests to test SRAMs," IEEE Des. Test Comput., vol. 10, no. 1, pp. 8–14, 1993, doi: 10.1109/54.199799.

[13] V. A. Vardanian and Y. Zorian, "A March-based fault location algorithm for static random access memories," in Proceedings of the Eighth IEEE International On-Line Testing Workshop (IOLTW 2002), 2002, pp. 256–261. doi: 10.1109/OLT.2002.1030228.

[14] A. Z. Jidin et al., "Generation of New Low-Complexity March Algorithms for Optimum Faults Detection in SRAM," IEEE Trans. Comput. Des. Integr. Circuits Syst., vol. 42, no. 8, pp. 2738–2751, 2023, doi: 10.1109/TCAD.2022.3229281.

[15] A. J. Van De Goor, G. N. Gaydadjiev, V. G. Mikitjuk, and V. N. Yarmolik, "March LR: a test for realistic linked faults," in Proceedings of 14th VLSI Test Symposium, Apr. 1996, pp. 272–280. doi: 10.1109/VTEST.1996.510868.

[16] S. Hamdioui and A. J. Van De Goor, "An experimental analysis of spot defects in SRAMs: realistic fault models and tests," in Proceedings of the Ninth Asian Test Symposium, 2000, pp. 131–138. doi: 10.1109/ATS.2000.893615.

[17] Z. Zhi-chao, H. Li-gang, and W. Wu-chen, "SRAM BIST Design Based on March C+ Algorithm," Mod. Electron. Tech., vol. 34, no. 10, pp. 149–151, 2011.

[18] I. G. Matri, Aishwaraya, N. Shreya, S. V Siddamal, and S. V Budihal, "A Novel March XR Algorithm, Design, and Test Architecture for Memories BT - Advances in Electrical and Computer Technologies," T. Sengodan, M. Murugappan, and S. Misra, Eds., Singapore: Springer Nature Singapore, 2022, pp. 321–329.

[19] M. A. Ahmed and A. M. Abuagoub, "MBIST Controller Based on March-ee Algorithm," J. Circuits, Syst. Comput., 2020, doi: 10.1142/S0218126621501607.

[20] G. Harutunyan, V. A. Vardanian, and Y. Zorian, "Minimal march tests for unlinked static faults in random access memories," in Proceedings of the IEEE VLSI Test Symposium, 2005, pp. 53–59. doi: 10.1109/VTS.2005.56.

[21] Z. Jianping, W. Zhenyu, Y. Jia, P. Wei, and Z. Yun, "An Improved SRAM Fault Built-in-self-test Algorithm," J. Hunan Univ. Nat. Sci., vol. 46, no. 4, 2019.

[22] S. Hamdioui, A. J. Van De Goor, and M. Rodgers, "March SS: A test for all static simple RAM faults," in Records of the IEEE International Workshop on Memory Technology, Design and Testing, 2002, pp. 95–100. doi: 10.1109/MTDT.2002.1029769.

[23] A. Paul and P. Rony Antony, "Optimized Microcode BIST Architecture for Multiple Memory Cores in SoCs," in 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT), May 2018, pp. 910–914. doi: 10.1109/RTEICT42901.2018.9012276.

[24] X. Ning, H. Yang, M. Zhang, Y. Wang, Y. Zhao, and S. Qiao, "Multi-type SRAM Test Structure with an Improved March LR Algorithm," in 2022 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2022, pp. 578–582. doi: 10.1109/APCCAS55924.2022.10090328.

[25] Siemens, "Tessent MemoryBIST: Embedded memory self-test, repair, and debug," 2019. [Online]. Available: https://static.sw.cdn.siemens.com/siemens-disw-assets/public/81225/en-US/Siemens-SW-Embedded-memory-self-test-repair-and-debug-Tessent-MemoryBIST-FS-81225-C1

[26] A. Z. Jidin, R. Hussin, L. W. Fook, M. S. Mispan, and W. Y. Loh, "Automatic generation of user-defined test algorithm description file for memory BIST implementation," Int. J. Reconfigurable Embed. Syst., vol. 11, no. 2, p. 103, 2022.

[27] A. Z. Jidin, R. Hussin, M. S. Mispan, and L. W. Fook, "A new 13N-complexity memory built-in self-test algorithm to balance static random access memory static fault coverage and test time," Int. J. Electr. Comput. Eng., vol. 15, no. 1, pp. 163–173, 2025.