Journal of Engineering and Technology

ISSN 2180-3811 eISSN 2289-814X https://jet.utem.edu.my/jet/index

IMPLEMENTATION OF MINIMIZED MARCH SR ALGORITHM IN A MEMORY BIST CONTROLLER

A. Z. Jidin*[1,2], R. Hussin[1], M. S. Mispan[2], W. F. Lee[3], and N. A. Zakaria[4]

[1] Faculty of Electronics Engineering and Technology, Universiti Malaysia Perlis, 02600 Arau, Malaysia.

[2] Faculty of Electrical and Electronic Engineering Technology, Universiti Teknikal Malaysia Melaka, 76100 Durian Tunggal, Melaka, Malaysia.

[3] Emerald System Design Center, 11900 Bayan Lepas, Pulau Pinang, Malaysia.

[4] UST Global Sdn Bhd, 11900 Bayan Lepas, Pulau Pinang, Malaysia.

corresponding_aimanzakwan@utem.edu.my

Article history:
Received Date: 1 September 2022
Revised Date: 23 November 2022
Accepted Date: 12 December 2022

Keywords: Design For Testability, March Algorithm,

Abstract— Memory Built-In Self-Test (MBIST) is essential in testing memories on a chip. Its efficiency depends on its fault coverage and the complexity of the algorithm used, which defines the test sequence to be applied to every cell of the memory under the test. This paper presents the implementation of a minimized-complexity March SR algorithm in an MBIST controller for detecting unlinked static faults in an SRAM. It was implemented as a User-Defined Algorithm (UDA), which was hard-coded in the MBIST controller. The simulations validated its functionality and

Memory BIST, Memory Fault Models

fault detection ability, producing similar fault coverage as the initial March SR algorithm with a shorter test completion time.

I. Introduction

MBIST is a technique of the Design for Testability (DFT) that is very popular for testing embedded memories on a System-on-Chip (SoC), owing to its capability to carry out self-testing and self-checking the test responses [1-2]. Since the on-chip circuitry carries out the memory test, an expensive high-performance external tester is no longer necessary [3]. Moreover, the test length and cost are reduced since the test latency introduced by the external tester can be minimized. Hence, the overall test length and cost are reduced [3-4].

Besides, it is essential to have a good test quality since the embedded memories in the more recent chips may occupy up to 90% of the total chip area [5]. Furthermore, many defects can randomly occur during the manufacturing process due to the compact and dense nature of the memory [6-7].

MBIST generates the test sequences to be applied to the memory under test based on the selected test algorithm. The March-series test algorithm is among the popular choices in the industry, owing to its ability to detect many possible faults at a linear test complexity [8].

The efficiency of a March test algorithm depends on its test complexity and fault coverage. Based on studies, the March MSS algorithm, with 18N complexity, is required to detect all unlinked static faults in a RAM [9], where N is the size of the memory under test. However, it requires a larger chip area and longer testing time compared to those with lower complexities, such as March C- (10N) [10], March CL (12N) [11], March LR (14N) [12], and March SR (14N) [13].

Yet, many of them cannot detect newer faults introduced by the Very Deep Submicron (VDSM) transistor technologies [7], as shown in Table 1. These

include the Deceptive Read Destructive Fault (DRDF) and the Deceptive Read Destructive Coupling Fault (CFdrd), which are more relevant to nowadays memory technologies.

Table 1: Fault Coverage of Several March Algorithms (F: Full Coverage, H: Half Coverage, 0: No Coverage)

Fault Type	March C- (10N)	March CL (12N)	March LR (14N)	March SR (14N)
Stuck-At (SAF)	F	F	F	F
Transition (TF)	F	F	F	F
Read Destructive (RDF)	F	F	F	F
Incorrect Read (IRF)	F	F	F	F
Deceptive Read Destructive (DRDF)	0	H	0	F
Transition Coupling (CFtr)	F	F	F	F
Deceptive Read Destructive Coupling (CFdrd)	0	H	0	H

Several previous works were proposed to reduce the complexity of the existing March algorithms. Research in [10] removed a redundant read operation in the March C test sequence and reduced its complexity to 10N to become the March C-. Its complexity was further reduced to 8N in [14] by rearranging its test sequence into two concurrent subgroups that are executed in parallel. Somehow, they did not introduce any improvement to cover the undetectable DRDF and CFdrd. Therefore, a new minimized March SR algorithm, also known as the March mSR, was introduced [15]. It has 1N complexity less than the initial March SR algorithm, with the test sequence ⇳(w0); ⇑(r0, w1, r1, w0); ⇑(r0, r0); ⇑(w1); ⇓(r1, w0, r0, w1); ⇓(r1, r1), by removing a read operation identified as redundant for detecting the intended faults.

Hence, this paper presents the implementation of the March mSR algorithm in MBIST controller hardware. It was done using the Mentor Graphic Tessent MemoryBIST software to hard-code the algorithm test sequence inside the MBIST

controller. It was then simulated in the Questasim simulator to validate its functionality and fault detection capability.

Section II describes the test sequence and fault coverage of the March mSR algorithm. Next, Section III discusses the methods used to perform the MBIST insertion process by implementing the March mSR algorithm as the UDA. Finally, Section IV presents and discusses the results obtained from the simulations performed on the generated MBIST controller. This paper focuses on detecting 26 Fault Primitives (FP) of the following faults: SAF (2 FPs), TF (2 FPs), RDF (2 FPs), IRF (2 FPs), DRDF (2 FPs), CFtr (8 FPs), and CFdrd (8 FPs).

II. March mSR Algorithm Description

March mSR algorithm consists of the following test sequence: ⇳(w0); ⇑(w1, r1, w0); ⇑(r0, r0); ⇑(w1); ⇓(r1, w0, r0, w1); ⇓(r1, r1). It has in total of 13 test operations. Hence, its complexity equals 13N. It consists of 6 test elements, notated as Ei where i = {0, 1, 2, 3, 4, 5}. Table 2 describes the test sequence of each test element.

Table 2: March mSR Description

Test Element E	Test Operations	Description
0	⇳(w0)	Each cell is written to 0 in any address direction.
1	⇑(w1, r1, w0)	Each cell is written to 1, read (expecting 0), and rewritten to 0, starting from the cell with the minimum memory address (ascending address order).
2	⇑(r0, r0)	Each cell is consecutively read twice (expecting 0) in the ascending address order.
3	⇑(w1)	Each cell is written to 1 in the ascending address order.
4	⇓(r1, w0, r0, w1)	Each cell is read (expecting 1), written to 0, reread (expecting 0), and rewritten to 1, starting from the cell with the maximum memory address (descending address order).
5	⇓(r1, r1)	Each cell is consecutively read twice (expecting 1) in the descending address order.

While Table 3 summarizes its coverage of the intended faults obtained using a fault detection analyzer [15]. It has 50% coverage of CFdrd and 100% coverage of the remaining faults. Therefore, it has a total fault coverage of 84.6%, where it can detect 22 FPs out of a possible 26. It provides the same fault coverage as the March SR algorithm, even with lesser 1N complexity [15].

Table 3: March mSR Fault Coverage

Fault Type	Coverage
SAF	2/2 (100%)
TF	2/2 (100%)
RDF	2/2 (100%)
IRF	2/2 (100%)
DRDF	2/2 (100%)
CFtr	8/8 (100%)
CFdrd	4/8 (50%)
Total Fault Coverage	**22/36 (84.6%)**

III. Research Methodology

The flowchart shown in Figure 1 depicts the overall flow of the proposed implementation of the March mSR algorithm in an MBIST controller as the User-Defined Algorithm (UDA). Firstly, a Tessent Core Description (TCD) file was developed to define the test sequence of the UDA to be implemented into the MBIST controller. It was written in a format recognized by the Mentor Graphic Tessent MemoryBIST software, used for the MBIST insertion process.

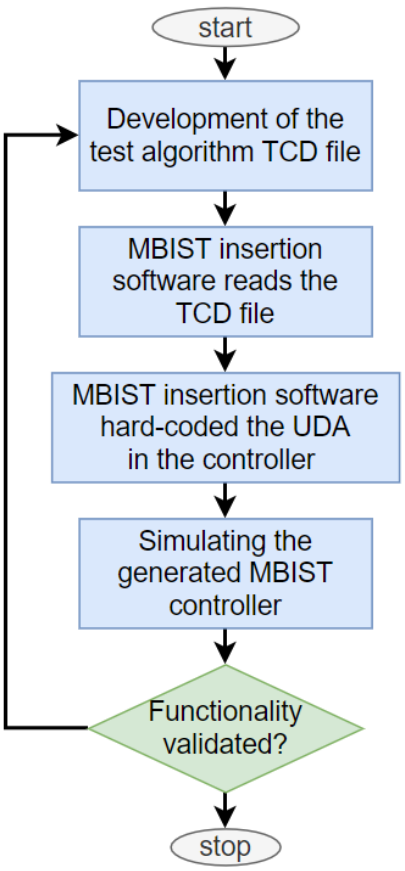


Figure 1: The Flow of the Proposed Research Methodology

The developed TCD file consists of the declaration of the UDA (in this case, it was named

march_mSR), the test setup, e.g., the selection of the test operation set and the initial memory address, and the description of the test sequence to be applied during each test element, as shown in Figure 2.

```
Algorithm (march_mSR) {
   TestRegisterSetup {
       OperationSetSelect : TessentSyncRamOps;
       AddressGenerator {
           AddressRegister (A) {
               LoadColumnAddress: MinColumn;
               LoadRowAddress: MinRow;
               X1CarryIn: None;
               Y1CarryIn: X1CarryOut;
           }
       }
       DataGenerator {
           LoadWriteData : 8'b00000000;
           LoadExpectData : 8'b00000000;
       }
   }
   MicroProgram {
       Instruction (M0_w0){
            OperationSelect : WriteWriteFastRow;
            X1AddressCmd : Increment;
            Y1AddressCmd : Increment;
            WriteDataCmd : DataReg;
            NextConditions {
                X1_EndCount : on;
                Y1_EndCount : on;
            }
       }
       Instruction (M1_w1r1w0){
            OperationSelect : WriteReadWriteInvert;
            X1AddressCmd : Increment;
            Y1AddressCmd : Increment;
            ExpectDataCmd : InverseDataReg;
            WriteDataCmd : InverseDataReg;
            NextConditions {
                X1_EndCount : on;
                Y1_EndCount : on;
            }
       }
       Instruction (M2_r0r0){
            OperationSelect : ReadRead;
            X1AddressCmd : Increment;
            Y1AddressCmd : Increment;
            ExpectDataCmd : DataReg;
            NextConditions {
                X1_EndCount : on;
```

```
                Y1_EndCount : on;
            }
        }
        Instruction (M3_w1){
            OperationSelect : WriteWriteFastRow;
            X1AddressCmd : Increment;
            Y1AddressCmd : Increment;
            WriteDataCmd : InverseDataReg;
            InhibitLastAddressCount : on;
            NextConditions {
                X1_EndCount : on;
                Y1_EndCount : on;
            }
        }
        Instruction (M4_r1w0){
            OperationSelect : ReadModifyWrite;
            ExpectDataCmd : InverseDataReg;
            WriteDataCmd : DataReg;
            NextConditions {
            }
         }
        Instruction (M4_r0w1){
            OperationSelect : ReadModifyWrite;
            X1AddressCmd : Decrement;
            Y1AddressCmd : Decrement;
            ExpectDataCmd : DataReg;
            WriteDataCmd : InverseDataReg;
            BranchToInstruction : M4_r1w0;
            NextConditions {
                X1_EndCount : on;
                Y1_EndCount : on;
            }
        }
        Instruction (M5_r1r1){
            OperationSelect : ReadRead;
            X1AddressCmd : Decrement;
            Y1AddressCmd : Decrement;
            ExpectDataCmd : InverseDataReg;
            NextConditions {
                X1_EndCount : on;
                Y1_EndCount : on;
            }
        }
    }
}
```

Figure 2: The TCD file for the March mSR algorithm

`WriteWriteFastRow` defines the sequential write operation at each clock cycle (w*x*w*x*), while `ReadRead` defines double read operations at consecutive clock cycles (r*x*r*x*).

`ReadModifyWrite` allows a read to be performed on the memory cell before overwriting it to the opposite value and rereading it (r*x*w*x'*r*x'*). Finally, `WriteReadWriteInvert` performs a write operation to the cell followed by a read before another write operation to the opposite value (w*x*r*x*w*x'*). The data to be read or written can be either `DataReg` (logic 0) or `InverseDataReg` (logic 1).

Specifically for E4, with test sequence ⇓(r1, w0, r0, w1), it was described using two instructions, M4_r1w0 and M4_r0w1, since the `TessentSyncRamOps` only includes operations sets with a maximum of three read or write operations. M4_r0w1 instruction was branched to M4r1w0 using the `BranchToInstruction` command.

The implementation of the UDA used the hard-coded method instead of the soft-coded one since the former offers design simplicity compared to the latter [16]. In addition, the ability to change the test algorithm during the program execution is not necessary for this research.

During the MBIST controller insertion process, the developed TCD file was read by the software mentioned earlier, which hard-coded the defined test sequences in the MBIST controller hardware in Verilog Hardware Description Language (HDL).

Once the intended MBIST controller was generated, it underwent two simulations to validate its functionality and fault detection ability. These simulations were carried out using the test benches and test patterns generated by the software upon completing the MBIST controller insertion process.

Should any errors be found during the simulations, the possible mistake(s) in the TCD file was fixed before repeating the MBIST insertion process and the simulations.

IV. Results and Discussion

A. March mSR Implementation in MBIST

Upon completion of the MBIST controller insertion, the generated MBIST controller was synthesized in Mentor Graphic Oasys-RTL software using the 130 nm CMOS process technology. Then, it was compared to the controller that implemented the initial March SR algorithm as the UDA regarding the chip area occupation and power consumption.

As shown in Table 4, the MBIST controller with the initial March SR as the UDA has a slightly lower area and power consumption, despite having a 1N complexity higher than the proposed March mSR. It has a more symmetrical test sequence structure than the latter, where its test elements E0 – E2 are symmetrical to E3 – E5, respectively. Hence, they can share the same hardware resources and only need to invert the test bits and test address orders.

Table 4: MBIST Controller Synthesis Report

MBIST UDA	Area (μm^2)	Power (mW)
March SR (14N)	10535	11.63
March mSR (13N)	10563	11.69

Oasys-RTL software using the 130 nm CMOS process technology. Then, it was compared to the controller that implemented the initial March SR algorithm as the UDA regarding the chip area occupation and power consumption.

As shown in Table 4, the MBIST controller with the initial March SR as the UDA has a slightly lower area and power consumption, despite having a 1N complexity higher than the proposed March mSR. It has a more symmetrical test sequence structure than the latter, where its test elements E0 – E2 are symmetrical to E3 – E5, respectively. Hence, they can share the same hardware resources and only need to invert the test bits and test address orders.

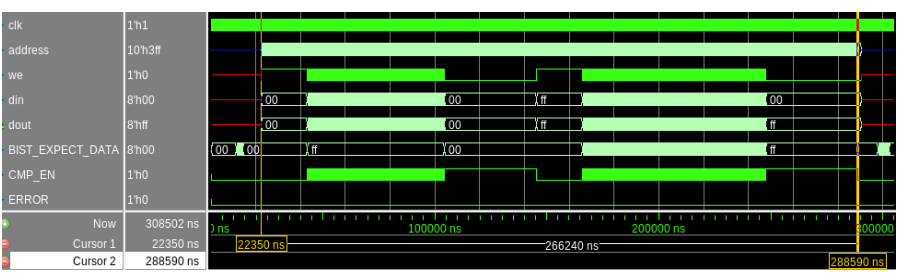
While in the case of the March mSR, E1 and E4 are not symmetry and, thus, are not sharing the hardware. Consequently, it consumes more power since it has more logic gates actively switching during the operation.

B. Validation via Functional Simulation

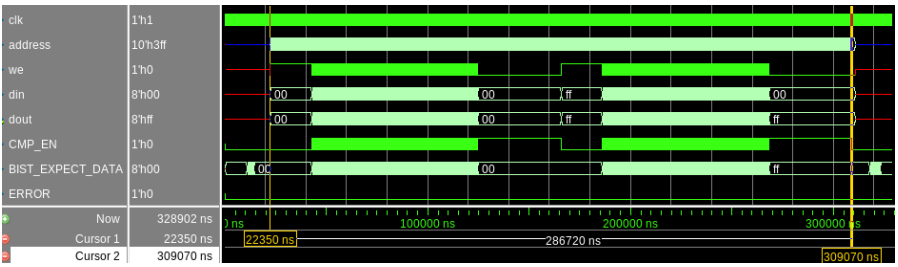
The MBIST controllers' functionality was validated via simulations in the QuestaSim simulator, using a 1KB Single-Port SRAM as the memory model (N = 1024) and a 20 ns clock as the system clock clk. By observing the resulting waveforms shown in Figure 3, the *ERROR* flag stayed at a low level throughout both simulations, which means that no mismatch has occurred between the observed read value (*dout*) and the expected value (*BIST_EXPECT_DATA*).

Regarding the simulation test completion time, 266.24 µs was required to complete the test using the March mSR algorithm. Based on Equation (1), a complexity *O* of 13 was derived, which equals its expected complexity. It also proves that it produces 20.48 µs or 1N clock cycles faster test than the March SR algorithm, which required 286.72 µs for completion. The difference in the completion time is more significant for a larger memory (e.g., 1 MB SRAM), which may have more impact on the overall chip testing time and production cost.

$$O = \frac{test\ time}{N * T_{clock}} \quad (1)$$



(a)



(b)

Figure 3: The waveforms of the simulation on the MBIST controllers that implemented: (a) March mSR algorithm, (b) March SR algorithm

C. Validation via Fault Detection Simulation

Next, a similar simulation was carried out on the MBIST controller with the March mSR as the UDA but using a faulty 1 KB Single-Port SRAM as the memory model. The purpose was to replicate the fault occurrences at the simulation level and validate the March mSR fault coverage. In this case, the addresses of all faulty or victim cells and aggressor cells were arbitrarily chosen. Its fault coverage was derived by observing the values of 26 bits from 7 fault detection flags at the end of the simulation: *saf_detect*, *tf_detect*, *irf_detect*, *rdf_detect*, *drdf_detect*, *cftr_detect*, and *cfdrd_detect*, as shown in Figure 4. A high bit indicates that the occurrence of the FP that it represents is detected during this simulation.

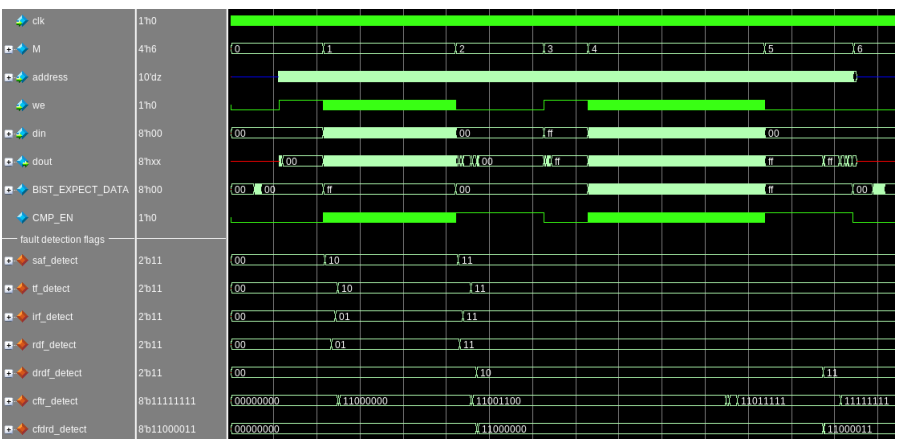


Figure 4: The March mSR fault detection simulation waveform

As shown in Table 5, which derived the fault detection flags' values from the simulation waveform in Figure 4, all FPs of SAF, TF, RDF, IRF, DRDF, and CFtr are detectable, and hence, it has 100% of these faults. While it only has 50% coverage of CFdrd since only 4 FPs of CFdrd are detectable. Therefore, these results validated the fault coverage of the implemented March mSR algorithm since its observed fault coverage in Table 5 is similar to its expected fault coverage shown in Table 3.

Table 5: Derived March mSR Fault Coverage from the Fault Detection Simulation

Fault	Detection Flag Value	Derived Fault Coverage
SAF	11	2/2 (100%)
TF	11	2/2 (100%)
RDF	11	2/2 (100%)
IRF	11	2/2 (100%)
DRDF	11	2/2 (100%)
CFtr	11111111	8/8 (100%)
CFdrd	11000011	4/8 (50%)

V. Conclusion

This paper has presented the implementation of the March mSR algorithm, a reduced-complexity March SR, as the UDA in an MBIST controller. Its test sequence was described in a TCD file, which was read and hard-coded into the MBIST controller by the Mentor Graphic Tessent MemoryBIST software during the MBIST insertion process. Simulations were performed on the generated MBIST controller to validate its functionality, test time, and fault coverage. Despite having slightly higher area and power consumption than the March SR algorithm, the proposed March mSR implementation produced an N-clock-cycle faster test while providing identical coverage of the intended faults. Subsequently, it reduces the overall test cost while preserving its quality.

VI. Acknowledgement

The authors would like to acknowledge the Faculty of Electronic Engineering and

Technology, Universiti Malaysia Perlis (UniMAP), Universiti Teknikal Malaysia Melaka (UTeM), and the Ministry of Higher Education Malaysia for their contribution and support to this research.

VII. References

[1] R. Manasa, R. Verma, and D. Koppad, "Implementation of BIST Technology using March-LR Algorithm," in *2019 4th International Conference on Recent Trends on Electronics, Information, Communication Technology (RTEICT)*, May 2019, pp. 1208–1212.

[2] A. K. S. Pundir and O. P. Sharma, "Fault tolerant reconfigurable hardware design using BIST on SRAM: A review," in *2017 International Conference on Intelligent Computing and Control (I2C2)*, 2017, pp. 1–16.

[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.

[4] A. Singh, G. M. Kumar, and A. Aasti, "Controller Architecture for Memory BIST Algorithms," in *2020 IEEE International Students' Conference on Electrical,Electronics and Computer Science (SCEECS)*, 2020, pp. 1–5.

[5] 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.

[6] Q. W. Chun, P. W. Leong, C. K. Yoong, L. I. Ee, and C. G. Chin, "VHDL Modelling of Low-Cost Memory Fault Detection Tester," *J. Eng. Technol. Appl. Phys.*, vol. 2, no. 2, pp. 17–23, 2020.

[7] N. A. Zakaria, "Multiple and solid data background scheme for testing static single cell faults on SRAM memories," Universiti Putra Malaysia, 2013.

[8] V. S. Chakravarthi, *A practical approach to VLSI System on Chip (SoC) design : a comprehensive guide*. Bangalore: Springer, 2019.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] M. Parvathi, N. Vasantha, and Ks. Parasad, "Modified March C- algorithm for embedded memory testing," *Int. J. Electr. Comput. Eng.*, vol. 2, no. 5, p. 571, 2012.

[15] A. Z. Jidin, R. Hussin, M. S. Mispan, L. W. Fook, and L. W. Ying, "Reduced March SR algorithm for deep-submicron SRAM testing," in *2022 IEEE International Conference on Semiconductor Electronics (ICSE)*, 2022, pp. 93–96.

[16] Siemens, "Tessent MemoryBIST User's Manual For Use with Tessent Shell," 2020.