Вычислительные технологии, 2024, том 29, № 4, с. 110–121. © ФИЦ ИВТ, 2024 ISSN 1560-7534
Computational Technologies, 2024, vol. 29, no. 4, pp. 110–121. © FRC ICT, 2024 eISSN 2313-691X

DOI:10.25743/ICT.2024.29.4.008

An investigation of augment March C− algorithm using IoT: heuristic approach

V. Mathur[1,*], K. S. Singh[1], K. A. Pundir[2]
[1]Amity University Rajasthan, 303002, Jaipur, India
[2]Arya College of Engineering & IT, 302028, Jaipur, India
*Corresponding author: Mathur Vinita, e-mail: vinita.mathurdec@gmail.com
Received August 24, 2023, revised February 06, 2024, accepted February 19, 2024.

In this paper an effective novel memory management strategy for IoT framework standard over MQTT with Raspberry Pi field protocol is used in RAID 6 (redundant array of independent drives) for data storage. RAID 6 offers a long data retention time for archiving as well as it provides a high fault and driven failure tolerance rate. The probability of rebuild failure rate is 0.397 % when using RAID 6. In this paper we have proposed a memory management strategy based on IoT framework for effective testing and repair. For this purpose a novel memory testing approach i. e. Improvised March C− (IM− March C−) Algorithm has been used with cases 32×8×1, 64×8×1, 256×8×1 for certain predefined injected-random fault at certain location in a RAID 6. The presented work has been tested on SPARTAN 2E, SPARTAN 3E, VIRTEX 2, VIRTEX 4, VIRTEX 5. From the results it is clear that the proposed algorithm has better fault coverage, requires less tracing time (0.14 s) and better device utilization.

Keywords: IoT framework, IoT gateway and smart device, master accelerator, MBIST and MBISR, RAID 6.

Citation: Mathur V., Singh K.S., Pundir K.A. An investigation of augment March C− algorithm using IoT: heuristic approach. Computational Technologies. 2024; 29(4):110–121. DOI:10.25743/ICT.2024.29.4.008.

Introduction

In the current scenario multiple sensors are connected to multiple smart devices to acquire high speed data through secure communication. For this purpose novel IoT (Internet of Things) technology is used for real time data tracking. IoT is a smart network that works on certain protocols for communicating information with sensor devices. It identifies, tracks, locates, monitors and controls objects in real time. There are two types of protocols: IoT network protocol and IoT data protocols [1]. IoT network protocols are set of rules for connecting multiple devices on the same network, these are LTE, CAT 1, LTE CATM 1, NB-IoT, Bluetooth, Zigbee, Lora wan [2]. IoT data protocols are standards for connecting and exchanging data with each other on a particular network. IoT Data protocols are AMQP (advance message queuing protocol) this is an open messaging standards, [3–5] MQTT (message queue telemetry transport) this is the sub messaging protocol used for connecting low power devices where memory requirement is less, HTTP (hypertext transfer protocol) is required for secure data communication for world wide web, CoAP (constrained application protocol) is used for saving the battery life and for increasing the storage capacity while

reducing the data requirement to operate, DDS (data distribution service) is used for real time data distribution, LW M2M (light weight machine to machine protocol) is designed for the machine to machine remote management with fast IoT solutions. In this paper, we have used an MQTT protocol with Raspberry Pi field protocol memory for testing and repair. RAID 6 memory array is used for memory testing. RAID 6 is a redundant array of independent disk. It provides high fault driven and failure tolerance with high data retention time period like archiving. This system is less prone to fault during the disk rebuilding process [6]. We have used 32×8×1, 64×8×1, 256×8×1 memory array as disk in RAID 6. For testing them a memory testing algorithms are used. These algorithms identifies the different faults in memory. These faults occur due to opens and short circuits in memory cells, errors in the address decoder or errors in logical read and write operations. These faults are classified as Stuck-at-fault (memory cell is stuck at logic one), stuck open fault (memory cell is stuck at logic zero), transition fault (once the logic is defined then it is not accessible back), neighbourhood pattern sensitive faults (memory cell can change its cell value with the influence of the neighbourhood cell), coupling fault [7]. BIST (build in self test) controllers works on the memory testing algorithms. These controllers provides fault-free memories [1]. The quality of testing algorithms is defined by the fault coverage concerning the FFM (function fault model) [8]. Various types of March testing algorithms have been designed [9], such as March DSS algorithm for simple static faults with certain conditions for fault coverage [3], March LSD algorithm for $75N$ complexity to detect all linked static and dynamic faults but are not suitable for unlinked faults [10], March LR algorithm to detect realistic linked faults [5], March PS algorithm for complexity of $23N$ to detect DRAM pattern-sensitive faults coverage, March SS with the test length of $22N$ which is able to detect all really simple static faults in RAMs but it can't detect linked faults [11], March NS algorithm to detect neighbourhood pattern-sensitive faults with SAF, TF, CF, and AFs. It deals with single-bit errors for 2D memories [12].

This paper is categorized into following sections. Section 1 describes the proposed IoT Framework in edge consists of Gateway and RAID 6. Section 2 covers the effective memory management unit for memory testing and repair. Section 3 presents the comparative result of the proposed algorithm. Concluding section presents conclusion and future scope.

1. Proposed IoT framework in edge computing using gateway and RAID 6

Fig. 1 shows the proposed IoT framework standard MQTT with Raspberry Pi field protocol for memory testing and repair. This architecture is sub-divided into three parts: master accelerator, gateways & smart devices, and memory management unit (MMU).

Master accelerator (MA). The master accelerator (Board Com's PCI 9000 series) main task is to monitor, collect, analyse and process the information for further stages. It provides information regarding the selection of memory testing algorithms. This block also performs the normalization and optimization of the selected algorithm [13]. In this paper, we have used the IM− March C−Algorithm,which is designed using its parents March C− algorithm. This algorithm is selected by MA in its default settings [14]. At the end, a fault dictionary is prepared which is connected to LCD display for real time monitoring.

Gateways and smart devices. Gateways and smart devices are the integral components of the proposed frame work, gateway serve as the central hub that connect the smart

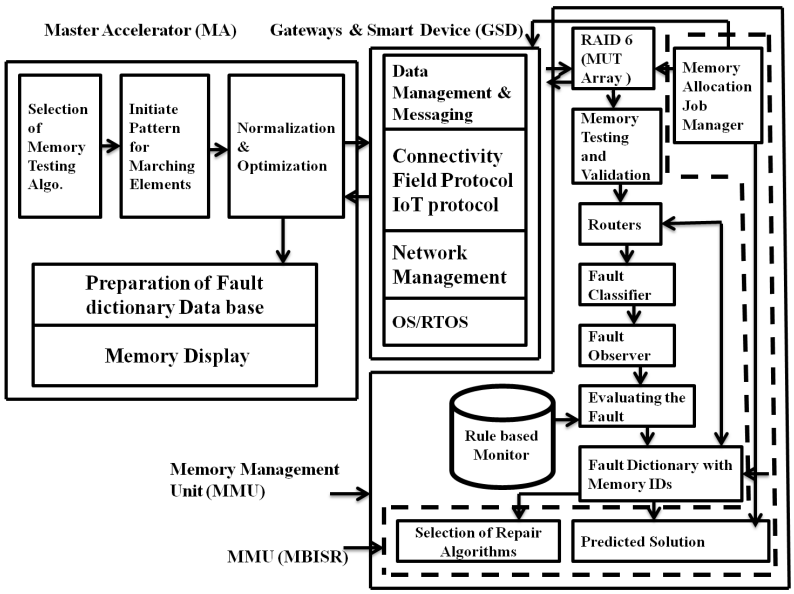

Fig. 1. Proposed IoT framework in edge computing using gateway and RAID 6

devices for real time monitoring system [15]. In this paper, MQTT protocol is used with network device (Raspberry Pi) [16] protocol for data managing and messaging.

Memory management unit (MMU). The MMU block provides assistance to MA block. It consists of sub blocks RAID 6 MUT (memory under test) array, memory testing and validation, routers, fault classifier, fault dictionary, MBISR module. Further classification of this unit is described in Sect. 2.

2. Memory management units (MMU)

MMU is used for effective Memory testing and repairing as shown in Fig. 2. For memory testing a novel memory testing approach is proposed Improvised March C− (IM March C−) Algorithm. We have used this algorithm over following cases 32×8×1, 64×8×1, 256×8×1 for fault detection in a RAID 6. This algorithm is derived from the parent algorithms. These faults are injected randomly without any connection to each other at certain predefined locations.

2.1. RAID 6

We have used three memory cases 32×8×1, 64×8×1 and 256×8×1 array as disk in RAID 6. These memory array are allocated through the memory allocation job manager with run time selection on MUT. RAID 6 is used to increase the performance and reliability of the data storage [13]. It is also known as double parity redundant array of independent drivers. It can continue the operation even if it faces disk failure twice. It provides enhanced fault tolerance capability.

2.2. Memory testing and validation

For memory testing and validation, we have proposed an IM− March C− algorithm for fault detection. The proposed algorithm adheres to the feature of its parent algorithm March C−. Difference between March C− and Proposed Improved March C−.

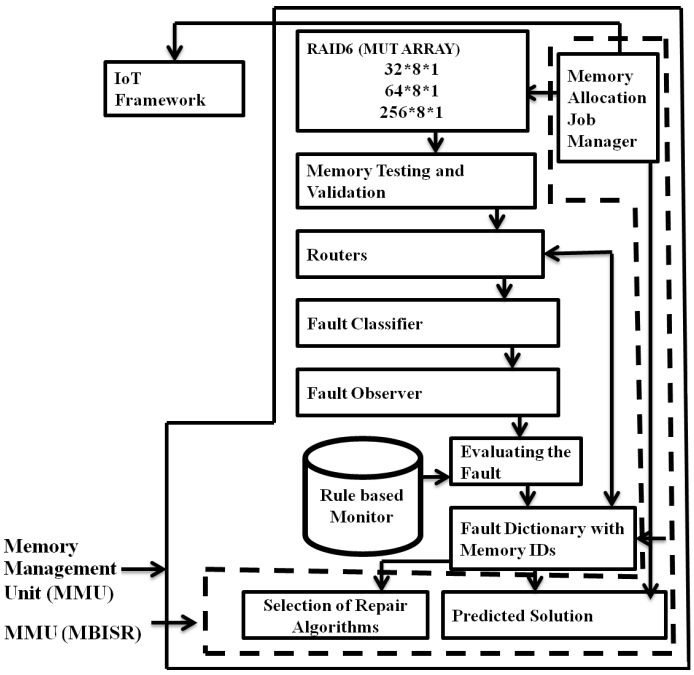

Fig. 2. Memory management units

T a b l e 1. Fault detection

Algorithm used	SAF0/1	TF	CFs	AF	Linked faults
March C−	Y	Y	Y	Y	–
IM March C−	Y	Y	Y	Y	Y

March C−
M1 M2 M3 M4 M5 M6
↕w0 ↑r0, w1 ↑r1, w0 ↓r0, w1 ↓r1, w0 ↕r0

Proposed Improved March C−
M1′ M2′ M3′ M4′ ↕wr0 ↑rd1, wr0, rd0, wr1 ↓rd0, wr1, rd1, wr0 ↕rd1 M5, M2 M4, M3

- Proposed Algorithm is designed with Complementary pairs of March C− elements.
- M2′ and M3′ contains all sensitive operators (rd0, wr1, rd1, wr0) which are cable of detecting all coupling CFs [2].
- All linked faults will also detect in the ↑ address order.
- A comparison between March C− and proposed is shown in Table 1.
- M1′, M2′, M3′ will also detect the SAF 0/1, TF, address decoder fault.
- As number of March element are reduced in IM March C− (M4′) by complementary the March C− element (M5′, M2′) and (M4′, M3′). This reduces its testing power as well as its testing time power consumption [2].

We have injected one Stuck-at-faults at memory location (1, 2), two transition faults at memory location (0, 0), (0, 1), (2, 2) and (2, 3) and one coupling fault at memory location (3, 2) and (3, 3) as shown in Fig. 3. These faults are randomly selected faults with non-resemblance. Fig. 3 shows the 4×4×1 memory size in which faults are injected and memory testing is performed using the proposed Improvised March C− algorithm.

Table 2, shows the proposed IM− March C− algorithm. Algorithm 1 has four March elements. In step 1, write Zero operation is performed in the first March element in any direction. In step 2 the second March element will read the expected one and write with zero at the last address location with expected read zero and write one in the upwards direction. In step 3, the third March element read the expected zero and write one at the last address location with read expected one and write zero in a downwards direction. At

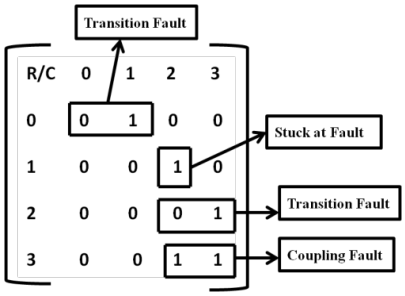

Fig. 3. 4×4×1 memory size

T a b l e 2. IM− March C− algorithms

Algorithm 1 IM− March C−
Step 1: ↕wr0
Step 2: ↑rd1, wr0, rd0, wr1
Step 3: ↓rd0, wr1, rd1, wr0
Step 4: ↕rd1

last, in step 4, the March fourth element will read the expected one in any direction. The proposed algorithms are compared with March C− for fault coverage, tracing time and device utilization.

2.3. Methodologies used in the proposed Improvised March C− test algorithm

Fig. 4, shows the state diagram of the proposed algorithm which is adhered to on March C−. The notations used in proposed algorithm are as follows.

Qn: March Elements
T: Test signal (1 = Test Mode and 0 = Normal Mode)
CLR: Clear address to the memory location 0
LA: Last address
DO: Data Out
QnB: Beginning of March elements
Qnr: Read data
Qnw: Write data

States/March elements Q0 and Q1 detects the Stuck-at-1 faults and transition faults. Q1, Q2, and Q3 detect Stuck-at-0 faults, transition faults, and coupling faults. This State diagram is working in two-modes. First, Normal mode (at $T = 0$) with the last address which is rolled over to the last memory location ($LA = 0$) this is when memory is in the idle phase. Second, when memory testing starts then it is Test mode (at $T = 1$) with last address which is rolled over to the last memory location ($LA = 1$).

The flow chart of the proposed algorithm is shown in Fig. 5. Memories under test are initially working at normal mode and once the testing starts then it switches to test mode. In test mode four March elements perform the read and write operation based on defined directions (upward/downward/irrelevant). When memory testing is completed an error free memory is achieved with the Fault dictionary which is generated at every read operation.

2.4. Simulation profile of the proposed Improvised March C− test algorithm

Fig. 6 shows the RTL top view of the MUT for 32×8×1 memory size. In this MUT an address array is 5 bit (0 to 4), data_in and data_out are 8 bit (0 to 7), and fault bit out is 8 bit (0 to 7). A proposed algorithm is applied to MUT whose RTL top view is shown in Fig. 7 using Xilinx ISE suite. In this clock, reset, test, last address, and data out are input signals, and count, clear, data in, write_CMD, up address and first address are output signals.

Fig. 8 and 9 show the test bench waveform of MUT 32×8×1 and proposed IM March C− algorithms, in this state change in input-output signals are shown as explained in the state diagram.

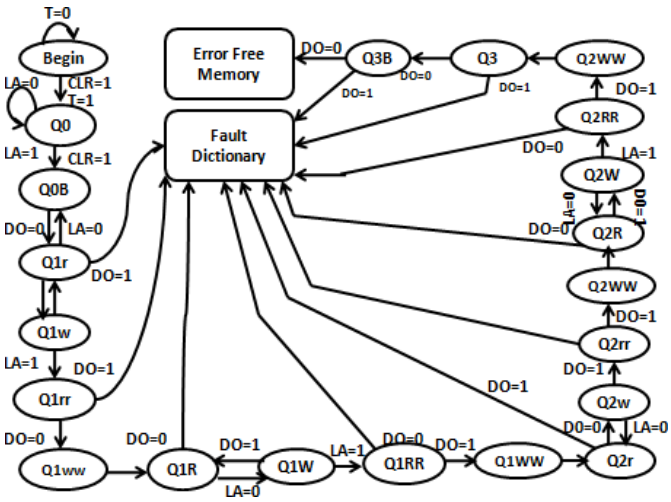

Fig. 4. IM March C− algorithms state diagram

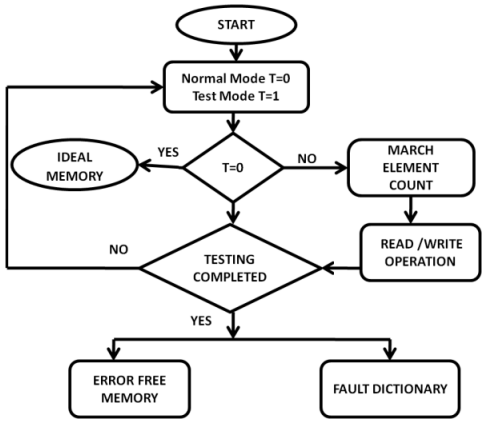

Fig. 5. Flow chart of IM− March C− algorithms state diagram

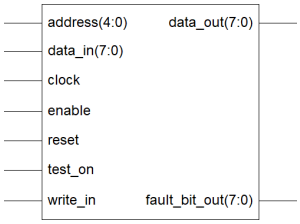

Fig. 6. MUT of size 32×8×1

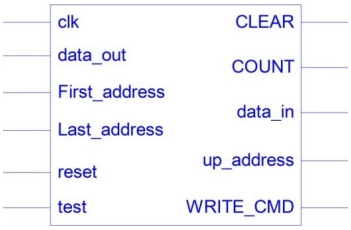

Fig. 7. Top view of proposed IM March C− algorithms

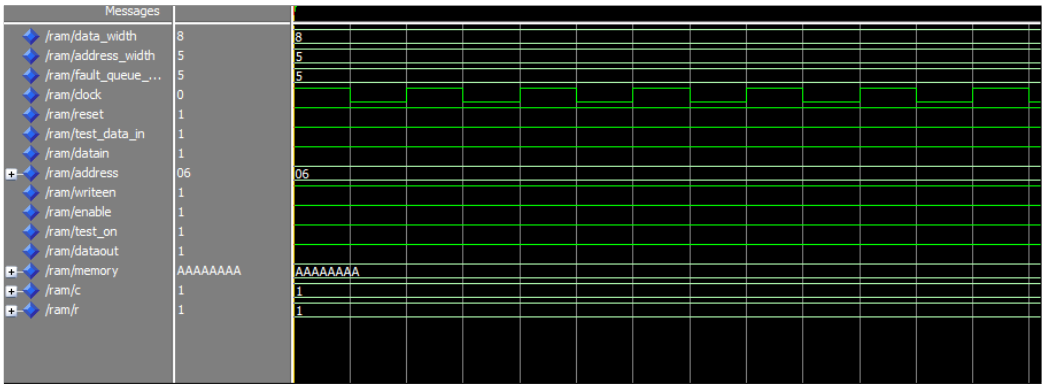

Fig. 8. Waveform of MUT 38×8×1

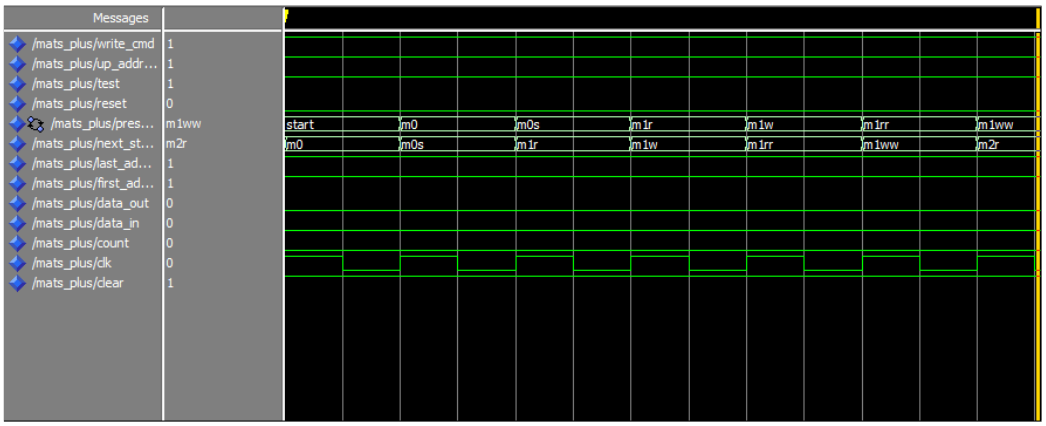

Fig. 9. Waveform of MUT 38×8×1

2.5. Process to repair the faulty memories (MBISR)

In the memory repairing process, MBISR (memory build-in self-repair) block is used to repair the faults. In this block a solution of Faulty memories is predicted on the basis of selected memory repair algorithm.

3. Comparative analysis

In this Section, the trade off performance of March C− and proposed IM March C− is analyzed. As shown in Table 3 testing time required by the proposed algorithm for fault coverage for address length $10N$ is less as compared to March C−. This simulation profile for fault coverage is conducted on MATLAB. Timing details and device utilization performance of MATS+, March C− and proposed IM is shown in Tables 4 and 5 implemented using the Xilinx ISE suite.

Trade-off performance of proposed and March C−. Table 3 shows the trade off performance of March C− with proposed IM− March C− based on tracing time and fault coverage. Address length is defined as $10N$. This trade-off performance is also simulated with MATLAB. Fig. 10 shows the tracing time comparisons of March C− and IM− March C− algorithm.

Timing details. Table 4 shows the minimum input arrival time before (Ta) the clock and maximum output required time after (Ta) the clock for MATS+, March C− and proposed IM− March C−. These testing algorithms are synthesized and implemented on Spartan 2E, Spartan 3E, Virtex 2, Virtex 4, and Virtex 5. Here, the minimum input arrival time of the proposed IM March C− algorithm required comparatively less time than MATS+ and March C−. Fig. 11 shows the comparison of these algorithms over different selected Devices. Here Ta defines the arrival time and Tr defines the required time.

Device utilization performance. Table 5, shows the device utilization performance for MATS+, March C− and the proposed IM March C− based on the number of slices,

T a b l e 3. Trade-off performance based on testing time, fault detected, and address length (complexity)

Algorithm	Tracing time (MATLAB), s	Fault detected	Address length (complexity)
March C−	0.25	SAF, ADFs, TF, CFs	6×6 ($10N$)
Proposed Improvised March C−	0.14	SAF (2), TF (2), and, CFs (2)	6×6 ($10N$)

T a b l e 4. Time required for input and output signals

Device selected	MAT+		March C−		Proposed IM March C−	
	Min. i/p Tb before clk, ns	Max. o/p required Ta clk, ns	Min. i/p arrival Tb clk, ns	Max. o/p required Ta clk, ns	Min. i/p arrival Tb clk, ns	Max. o/p required Ta clk, ns
Spartan 2E	3.509	6.613	4.139	6.613	4.139	6.609
Spartan 3E	2.6878	7.078	2.894	7.078	2.892	7.078
Virtex 2	2.011	4.659	2.096	4.659	2.096	4.599
Virtex 4	1.442	3.879	1.422	3.879	1.422	3.869
Virtex 5	1.211	2.699	1.211	2.699	1.211	2.670

T a b l e 5. Device utilization performance of MAT+, March C− and the proposed algorithm

Device selected	Parameter	MAT+			March C−			Proposed IM March C−		
		Used	Available	Utilization, %	Used	Available	Utilization, %	Used	Available	Utilization, %
Spartan 2E	Number of slices	14	768	1	25	768	3	25	768	3
	Number of slice flip flops	22	1536	1	42	1536	2	42	1536	2
	Number of 4 input LUTs	25	1536	1	41	1536	2	39	1536	2
	Number of bonded IOBs	11	124	8	11	124	8	11	124	8
	Number of GCLKs	1	8	12	2	8	25	2	8	25
Spartan 3E	Number of slices	14	768	1	25	768	3	25	768	3
	Number of slice flip flops	22	1536	1	42	1536	2	42	1536	2
	Number of 4 input LUTs	25	1536	1	41	1536	2	41	1536	2
	Number of bonded IOBs	11	98	11	11	178	6	11	178	6
	Number of GCLKs	1	4	25	2	4	50	2	4	50
Virtex 2	Number of slices	14	256	5	25	256	9	25	256	9
	Number of slice flip flops	22	512	4	42	512	8	42	512	8
	Number of 4 input LUTs	25	512	4	41	512	8	41	512	8
	Number of bonded IOBs	11	88	12	11	88	12	11	88	12
	Number of GCLKs	1	16	6	2	16	12	2	16	12
Virtex 4	Number of slices	14	6144	0	25	6144	0	25	6144	0
	Number of slice flip flops	22	12 288	0	42	12 288	0	42	12 288	0
	Number of 4 input LUTs	25	12 288	0	41	12 288	0	41	12 288	0
	Number of bonded IOBs	11	240	4	11	240	4	11	240	4
	Number of GCLKs	1	32	3	2	32	6	2	32	6
Virtex 5	Number of slices	22	19 200	0	42	19 200	0	42	19 200	0
	Number of slice flip flops	5	27	18	7	49	14	7	49	14
	Number of 4 input LUTs	7	27	25	36	19 200	0	36	19 200	0
	Number of bonded IOBs	11	220	5	11	220	5	11	220	5
	Number of GCLKs	1	32	3	2	32	6	2	32	6

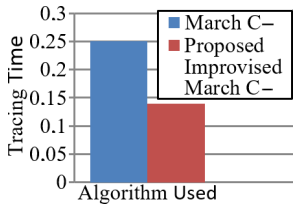

Fig. 10. Tracing time comparisons of March C– and improvised March C– algorithm

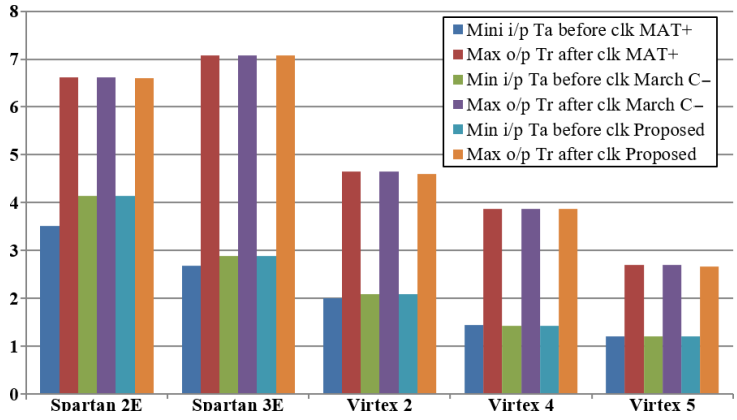

Fig. 11. Time required for input and output signals for the different selected devices

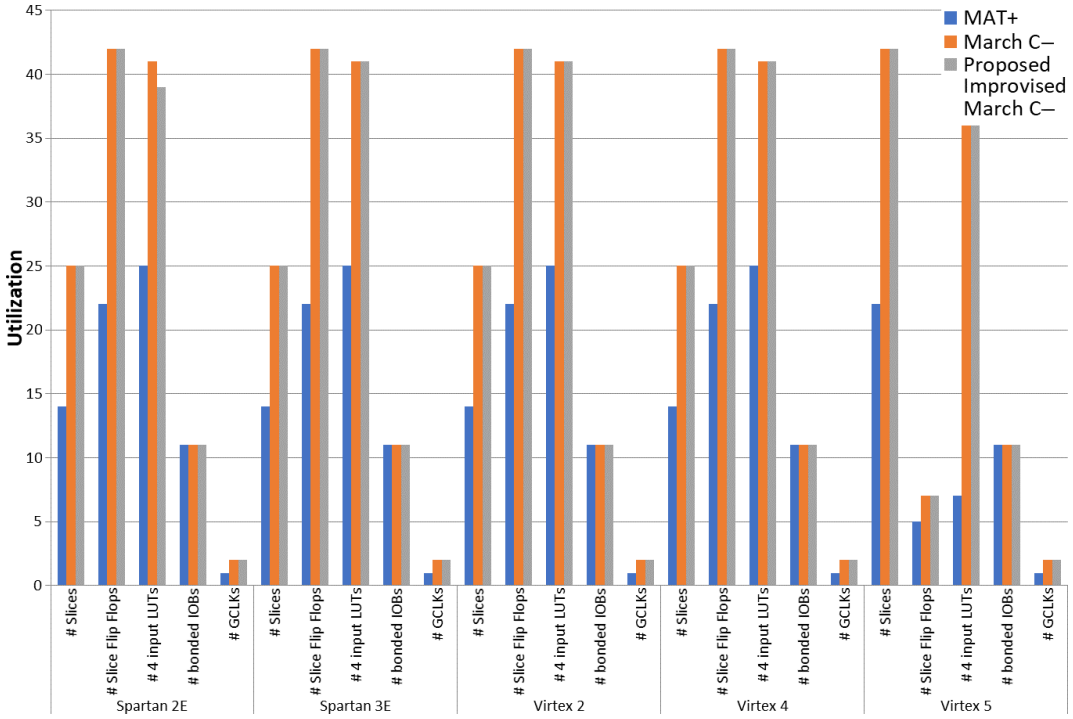

Fig. 12. Device utilization performances

Number of slice flip flops, Number of 4 input LUTs, Number of bonded IOBs, and number of GCLKs used and their utilization.Implemented on Spartan 2E, Spartan 3E, Virtex 2, Virtex 4, and Virtex 5. Fig. 12 shows the Device Utilization Comparisons Performance as mentioned in Table 5.

There is no difference in March C– and proposed Improvised March C–. The implementation results outcomes are shown in Table 5. However, the LUTs and bounded IOBs provide advantages over MAT+. The proposed algorithm adheres to the feature of its parent algorithm March C–. The difference between March C– and the proposed Improved March C– is discussed. As can be seen from Table 4 the Proposed IM March C– time required for input and output signals arrival time before and after the clock is less as compared to March C–.

Conclusion and future scope

With the advancement in IoT and supportive edge computing technologies the role of linked memory devices becomes crucial over time. The memory intensive applications demand the suitable memory management strategies for real time and fast switching operation along with fault tolerance. The presented work focuses on this role of memory used as a disk in RAID 6 in conjunction with IoT framework of edge computing. The presented architecture is composed of master accelerator (MA), gateway and smart devices (GSD) and a memory management Unit (MMU). As the work is focused on MMU, cases were taken where disks in RAID 6 were sized 32×8×1, 64×8×1 and 256×8×1 with injected random faults. The MMU provides testing, repair solutions to the MA with the help of IoT framework. For testing purpose an IM March C− algorithm was proposed which is based on the parent algorithm March C−. The result shows better fault coverage, less tracing time (0.14 s) for IoT framework module. The device utilization tested on Spartan 2E, Spartan 3E, Virtex 2, Virtex 4, and Virtex 5.

References

[1] **Iqbal A., Lee T.J.** GWINs: group-based medium access for large-scale wireless powered IoT networks. IEEE Access. 2019; (7):172913–172927. DOI:10.1109/ACCESS.2019.2956029.

[2] **Bosio A., Carlo S.Di., Natale G.Di., Prinetto P.** March test generation revealed. IEEE Transactions on Computers. 2008; 57(12):1704–1713. DOI:10.1109/TC.2008.105.

[3] **Dekker R., Beenker F., Thijssen L.** A realistic fault model and test algorithms for static random access memories. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 1990; 9(6):567–572. DOI:10.1109/43.55188.

[4] **Dekker R., Beenker F., Thijssen L.** Minimal march-based fault location algorithm with partial diagnosis for all static faults in random access memories. IEEE Design and Diagnostics of Electronic Circuits and Systems. 2006; 260–265. DOI:10.1109/DDECS.2006.1649632.

[5] **Mathur V., Pundir A.K., Singh S., Singh S.K.** An insight into algorithms and self repair mechanism for embedded memories testing. Lecture Notes in Electrical Engineering. Springer, Singapore: 2024; (1065).

[6] **Park K., Lee Jo., Kang S.** An area efficient programmable built-in self-test for embedded memories using an extended address counter. International SoC Design Conference. 2010; 59–62. DOI:10.1109/SOCDC.2010.5682974.

[7] **Singh B., Narang S.B., Khosla A.** Modeling and simulation of efficient March algorithm for memory testing. Contemporary computing. Communications in Computer and Information Science. 2010; (95):96–107. Available at: `https://link.springer.com/chapter/10.1007/978-3-642-14825-5_9`.

[8] **Zarrineh K., Upadhyaya S.J., Chakravarty S.** Automatic generation and compaction of March tests for memory arrays. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2002; 9(6):845–857. DOI:10.1109/92.974898. Available at: `https://www.researchgate.net/publication/3337228_Automatic_generation_and_compaction_of_March_tests_for_memory_arrays`.

[9] **Hamdioui S., Ars Z.Al., Van de Goor A.J., Rodgers M.** Linked faults in random access memories: concept, fault models, test algorithms, and industrial results. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2004; (23):737–757. DOI:10.1109/TCAD.2004.826578.

[10] **Lakshmi G.S., Neelima K., Subhas D.** A March Ns algorithm for detecting all types of single bit errors in memories. International Journal of Emerging Technologies and Innovative Research. 2019; 6(4):289–296.

[11] **Lee K.H.** Using OFDMA for MU-MIMO user selection in 802.11ax-based Wi-Fi networks. IEEE Access. 2019; (7):1860410–186055. DOI:10.1109/ACCESS.2019.2960555.

[12] **Pan Y., Li Y., Xu Y., Shen B.** DCS: diagonal coding scheme for enhancing the endurance of SSD-based RAID arrays. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2016; (35):1372–1385. DOI:10.1109/TCAD.2015.2504333.

[13] **Fu Y., Shu J., Luo X., Shen Z., Hu Q.** Short code: an efficient RAID-6 MDS code for optimizing degraded reads and partial stripe writes. IEEE Transactions on Computers. 2017; 66(1):127–137. DOI:10.1109/TC.2016.2576461.

[14] **Hamdioui S., van de Goor A.J., Rodgers M.** March SS: a test for all static simple RAM faults. Proceedings of the 2002 IEEE International Workshop on Memory Technology, Design and Testing (MTDT2002). 2002: 95–100. DOI:10.1109/MTDT.2002.1029769.

[15] **Mathur V., Pundir A.K., Gupta R.K., Singh S.K.** Recrudesce: IoT-based embedded memories algorithms and self-healing mechanism. Proceedings of Congress on Control, Robotics, and Mechatronics. CRM 2023. Smart Innovation, Systems and Technologies. 2024; (364).

[16] **Manasa R., Verma R., Koppad D.** Implementation of BIST technology using March-LR algorithm. 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT). 2019: 1208–1212. DOI:10.1109/RTEICT46194.2019.9016784.

Вычислительные технологии, 2024, том 29, № 4, с. 110–121. © ФИЦ ИВТ, 2024 ISSN 1560-7534
Computational Technologies, 2024, vol. 29, no. 4, pp. 110–121. © FRC ICT, 2024 eISSN 2313-691X

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

DOI:10.25743/ICT.2024.29.4.008

Исследование алгоритма Augment March C– с использованием Интернета вещей: эвристический подход

В. Матур[1,*], К. С. Сингх[1], К. А. Пундир[2]

[1]Университет Амити Раджастан, 303002, Джайпур, Индия
[2]Колледж инженерии и информационных технологий Арья, 302028, Джайпур, Индия
*Контактный автор: Винита Матур, e-mail: vinita.mathurdec@gmail.com

Поступила 24 августа 2023 г., доработана 06 февраля 2024 г., принята в печать 19 февраля 2024 г.

Аннотация

В этой статье представлена новая эффективная стратегия управления памятью для стандарта инфраструктуры Интернета вещей через MQTT с протоколом Raspberry Pi, которая используется в RAID 6 (резервный массив независимых дисков) для хранения данных. RAID 6 обеспечивает длительное время хранения данных для архивирования, а также высокую устойчивость к сбоям и управляемым отказам. Вероятность сбоя восстановления составляет 0.397 % при использовании RAID 6. В этой статье мы предложили стратегию управления памятью, основанную на платформе IoT, для эффективного тестирования и восстановления. Для этой цели был использован новый подход к тестированию памяти, а именно импровизированный

алгоритм March C− (IM-March C−) для реализаций 32×8×1, 64×8×1, 256×8×1 для заранее определенных и заранее введенных случайных ошибках в определенных местоположениях в RAID 6. Представленная работа была протестирована на SPARTAN 2E, SPARTAN 3E, VIRTEX 2, VIRTEX 4, VIRTEX 5. Из результатов видно, что предложенный алгоритм лучше обрабатывает сбои, требует меньшего времени трассировки (0.14 с) и более эффективно использует устройство.

Ключевые слова: IoT framework, IoT gateway и smart device, master accelerator, MBIST и MBISR, RAID 6.

Цитирование: Матур В., Сингх К.С., Пундир К.А. Исследование алгоритма Augment March C− с использованием Интернета вещей: эвристический подход. Вычислительные технологии. 2024; 29(4):110–121. DOI:10.25743/ICT.2024.29.4.008. (на английском)