Memory test and repair technique for SoC based devices

Mohammed Altaf Ahmed^{1, a), b)}, Abubaker E.M. Eljialy², and Sultan Ahmad³

Abstract System on Chip (SoC) architecture mainly consists of the memories in a larger area. Due to the availability of memories in a larger-size, it is difficult to test these memories for faults. Therefore, a smooth test solution to test these memories against fault and repair the faulty cells has introduced. In this research, we proposed a Memory Test Controller (MTC) to test the memories and Built-in Self-repair (BISR) mechanism to repair the faulty cells for any recent SoC based devices. The MTC not only identifies the fault, but it finds the type of fault available, and BISR block repairs the detected faulty cells. The paper provides empirical insights about how change is brought in features of the SoC based device after integrating both the proposed controller block. It is noticed that from the obtained results, the proposed methods are stands better in terms of the area overhead, power and timing when compared with the existing approaches.

Keywords: MTC, faults in memory, fault types detection, SoC, BISR

Classification: Integrated circuits (memory, logic, analog, RF, sensor)

1. Introduction

The current SoC based device needs many tasks to perform to sustain a reputation in the market. Among the various functions, the user wants one of the essential functions that the device must contain large memory space. The increase in the ample storage of memory in SoC design impacts its performance. More storage will expose prone to a fault [1, 2]. It results in difficulties in testing memories in terms of failures [3, 4]. As the survey reference to report of ITRS (International Technological Roadmap for Semiconductor) conducted by Semiconductor Industry Association (SIA), the portion of memory in SoC is above 95% of the total area [5]. Thus, testing of such a dense memory in SoC has become critical [6]. Therefore, to provide the test solution for memories in any SoC-based device (for example, the latest mobile phone) is the research objective of this paper.

Due to the large size and massive amount of data storage, the memories in SoC considerably impact on the yield of the production of the devices [7]. Therefore, a useful test solu-

tion with a fault repair mechanism will support improving the recent SoC-based systems' yield.

This research consists of two methods, first to test the memories and second one to repair the memories for fault. There are existing approaches available to test the memories for failures but can be further enhanced to get better results [8, 9, 10, 11, 12, 13, 14]. Among the various test techniques of memories, the algorithmic test approach retained its positions for a long time [15]. The algorithmic approach is fast and provides better fault coverage [9]. The stuck-at fault, transition fault, address decoder faults, neighbourhood pattern sensitive faults (NPSF), and coupling faults computes by memory built-in self-test (MBIST) approach [9, 12, 16, 17, 18, 19, 20]. The research approaches [21, 22, 23] are not able to detect all fault types in semiconductor memories. They identify only stack-at fault, address decoder faults, transition faults, and only some coupling faults. The coupling faults are detected using a March-ee algorithm in the research [12].

Memory repair is a standard method for yield improvement for today's modern SoC design. This process usually consists of memory test, redundancy analysis (RA) and repair mechanism. The repair signature is produced by the redundancy analysis depending on the failure information received from the MTC, also known as MBIST. The BISR block uses this repair signature to repair the memory under test (MUT) [24, 25, 26, 27, 28]. Memories are more proven to faults; thus, memory testing involves various test techniques on different test algorithms and various faults modules. We cannot merely ignore the defects if they exist in memory. Therefore, the spare memory method is used. In the spare memory scheme, some predefine spare rows and spare column are reserved, and when faulty cells are going to operate, spare rows and spare columns will use in place of the defected cells by the BISR block. Therefore the MBIST, together with BISR blocks, plays a significant role in SoC development. Recent SoC based devices (for example, mobile phone) are designed with large memory size. Therefore the significant memory test method improves the yield of the products.

With the advancements in the technology of the mobile industry, the device has to have smooth performance and functioning at the hardware level. Therefore, it is required to improve the architecture of the SoC for the current generation devices. This improvement in the SoC design will ensure the smooth operation of devices towards the value-added components in it. We targeted in this research, one of the value-added elements of ample memory storage in the SoC. A smooth memory test controller with a fault repair

¹ Department of Computer Engineering, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Saudi Arabia

² Department of Information System, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Saudi Arabia

³ Department of Computer Science, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Saudi Arabia

^{a)} m.altaf@psau.edu.sa

^{b)} altaface1@gmail.com

mechanism is proposed to test and repair the memories under SoC. This memory test engine examines the memories for faults and computes all the possibly generated defects and their types. The BISR block gets failure information of the memory from MTC and repairs the faulty cells by the proposed method. The proposed MTC and BISR are developed to integrate with the SoC design, and which results in a small hardware overhead. It became significant when it compares with the memory controllers available in SoC [3, 25, 28] and the BISR block [24, 26, 27, 29] of the existing approaches. The proposed MTC and BISR blocks stand better in terms of area overhead and power consumption to test and repair the memories under SoC when compared with existing methods. For example, it takes only 2.5% additional hardware overhead in case of MTC and 3.1% in case of BISR with 8kx64 memory size.

The contribution of this research work listed as,

- Proposing a memory test controller (MTC) based on modified March C algorithm to test memories for faults and their types in the modern SoC.
- Presenting the BISR method to repair the faults for MUT depending on the fault signature of BIRA.
- Comparison of the finding of the proposed method with other existing approaches in terms of area, power and timing enhancement.

The rest of the paper is structured as section 2 consists of the architecture of the proposed MTC and BISR controller. Section 3 consists of the experimental results and comparison with the existing methods. Section 4 outlines the conclusion and future scope.

2. Proposed architecture of MTC and BISR

The proposed architecture of an SoC-based device is consists of two central part first one is MTC, and the second one is BISR, as shown in Fig. 1. The MTC and BISR controllers are used to integrate into the SoC design to enhance the performance and features of a device.

2.1 Memory test controller (MTC)

The MTC have six main blocks, two register unit to start the state machine and to record the fail information, a controller state machine on modified March C algorithm, a test pattern generator, a comparator to compare the output with

Fig. 1 The architecture of the memory test controller and repair logic for an SoC based embedded device.

the desired pattern and a memory under test (MUT) block. The start register consists of clock, reset, start, resume, stop, memory id (two bits), and halt-on-error. The state machine begins once the start signal is programmed in start register. It triggers the state machine to perform the operations mentioned in the algorithm.

The test pattern generator will work on the proposed modified March C algorithm and generates the patterns as specified in the algorithm. These generated patterns are applied to the memory under test, during the read operations, while testing, read data compared with the desired patterns if the comparison fails, the state machine jumps to the status-of-fail-record state and records the comparison results. The memory ID, fail address, faulty cells, and defective cell count of the MUT records in the output response recorder.

These fail information is transfer to the BISR block to use spare memory instead of the faulty cells.

2.1.1 Test algorithm

The memory test controller works on the modified form of March C algorithm to improve fault coverage. We modified March C element because it is not able to detect coupling faults. The predefine patterns applied to the memory under test (MUT), and the responses recorded into the output response recorder. The writing patterns into the memory and simultaneously reading them from memory which described in the algorithm, the state machine is designed. The state machine consists of the state equal to the operations specified in the algorithm, and also two more states, idle and status-of-fail-record. The purpose of an idle state is to start operations, and the status-of-fail-record state is to store the fail information. In the present research, we proposed the modified March C algorithm for conducting the test, which has given below.

Proposed Modified March-C algorithm	
E^1 : $\uparrow \downarrow (w0)$; E^2 : $\uparrow (r0, w1)$; E^3 : $\uparrow (r1, w0, r0)$;	
E^4 : $\downarrow (w0, r0, w1)$; E^5 : $\downarrow (r1, w0)$; E^6 : $\downarrow (r0)$;	

Steps in the algorithm specified in Table I, further the operation sequence in each step explained below. There are two modified march elements E^3 and E^4 in the algorithm as compared with March C [30], and MTC works as per these steps. At each step, the operations under the parenthesis (read/write) are performed sequentially at each memory location (in ascending order or descending order as mentioned in the algorithm).

2.1.2 State machine

The state machine designed for the memory controller that consists of eight states, and it works on modified March C

Table I Illustrations used in the algorithm

Operation	Description
$\uparrow \downarrow$	This notation of modified March C indicates that the address ordering of the memory is not essential, ascending (from zero to maximum).or descending (from maximum to zero).
$\uparrow$	The address order of the memory has indicated in ascending order.
$\downarrow$	The address order of the memory has indicated in descending.
w0	Write 0s at the memory location.
w1	Write 1s at the memory location.
r0	The memory location, whose value is 0, read the data.
r1	The memory location, whose value is 1, read the data.

Fig. 2 Memory controller state machine

principles shown in Fig. 2.

These states are listed as follows:

- (1) Idle-state (2) w0 (3) r0w1 (4) r1w0r0 (5) w0r0w1 (6) r1w0 (7) r0 (8) status-of-fail-record

The state machine in idle state wait for the start signal once received it jumps to the w0 state and starts memory-test operation as described in the algorithm respectively. The memory under test is full of zero patterns when it jumps to the next r0w1 state. In this state, the state machine will perform two operations first read and then write as described in the algorithm. Similarly, the state machine performs all these operations sequentially at each state. While testing, read data compared with the desired patterns during the read operations. If the comparison fails, the state machine jumps to the status-of-fail-record state. It records the comparison results and resumes back to the next address location from where it arrived. The memory ID, fail address, faulty cells, and defective cell count of the MUT records in the output response recorder. In this manner, it completes all operations and displays the result in the status-of-fail-record state. The state machine can detect the fail memory ID, location of faults, faulty cell and faulty cell count. Moreover, the simulation process computes various fault types. This research methodology determines the stuck-at (SA) fault, address decoder fault, transition fault, and coupling faults.

2.2 Built in self repair

The memory fails information (memory ID, faulty location, faulty cell and faulty cell count) from the status of fail record register is transfer to the BISR block to repair of the defective cells of the fail memory. The BISR block consists of redundancy logic (RL) and the spare memory (row-column) block and is shown in Fig. 3. The RL block is used to store faulty addresses found during the memory test process. During read and write operation of memory, the address is compared with the addresses available in the redundancy logic. If it matches, the BISR will work, and data will access through the spare memory. The BIRA block accesses the failure information, calculates the repair signature, and stores it in the BIRA register. It also compares the defected addresses with the previously saved addresses in the fault table in case of multiple faults. The address is stored only when if it is not available in the fault table. BIRA will decide the spare

Fig. 3 BISR block diagram

Fig. 4 BISR algorithm flow

row/column allocation based on the faulty cell count information in a particular address. The BISR block works on the principle shown in the flow chart of Fig. 4.

The flow chart steps are as follows:

1. The repair principle works on the simple rule, if a row has multiple faults, it will repair, and if the column has numerous faults at different row, the column will fix.
2. The faulty cell count is received by MTC and is taken as a reference by BIRA to calculate the repair signature.
3. A predefined threshold value to use spare rows and spare columns will decide depending on the count of defected cells in defected rows or defected columns. The predefined threshold value is equal to or greater than '2'.
4. If row defect count is greater or equal to '2', the spare row will allocate first than the column; otherwise, the spare column will allocate first.
5. Once the spare memory (row/column) allocation finished, the remaining defected cells' checking process will carry on. If it is non zero, the spare row will allocate, the checking process continuing until it reaches null.
6. The spare memory can be increase depending on the faulty cells count. However, we considered in this research two spare rows and two spare columns.

Memory Controller Area Overhead Vs Memory Size **Fig. 6** Memory controller area overhead vs memory size plot**Table V** Memory size versus area overhead of MTC

Memory Size	Memory Area (nm ²)	Memory Controller Area (nm ²)	Test Area (%)	Overhead
2Kx64	142255	8910	6.26	
4Kx64	227608	8956	3.93	
8Kx64	396719	10071	2.5	
16Kx64	582678	9087	1.55	
32Kx64	932284	9113	0.977	
64Kx64	1491654	9233	0.618	

Table VI Area overhead comparison of proposed and existing approaches

Memory Size	Area Overhead (%)			
	Chin-Lung et al. [28]	Chang, DM et al. [3]	Xiao-feng et al. [25]	Proposed MTC
2Kx64	10.6	9.1	7.8	6.26
4Kx64	6.71	5.7	4.2	3.93
8Kx64	4.1	3.5	3.2	2.5
16Kx64	2.64	2.26	2.8	1.55
32Kx64	1.66	1.42	1.1	0.977
64Kx64	1.05	0.9	0.8	0.618

8Kx64 memory. The total area received for the 8Kx64 memory is 396719 nm², and for the MTC obtained area is 10071 nm², which is approximately 2.5% of the 8k memory area.

Therefore, the area overhead approximately 2.5% is received. The area of the memory controller always fixed with the different sizes of the memory. Consequently, it can conclude as memory size increases, the area overhead of the memory controller decreases in any SoC-based device. Different memory sizes with the same memory controller affect area overhead as given in Table V and illustrated in the graph (Fig. 6). The area overhead comparison of the proposed MTC with different existing methods is given in the Table VI.

3.2.2 ASIC results for BISR

Similarly, the BISR block is also implemented on ASIC, and the results have obtained for area, power and timing. The obtained results are compared with the existing methods of BISR. The existing methods of the research [24, 27, 31] are not enough to test the memory for fault, they either use fault injection method for getting faulty cells information instead of MTC or just comparing neighbouring cells to assume defects in the memory. The comparison with different existing approaches of BISR is given in Table VII.

The area report of the proposed BISR consists of 482 as a total cells count, and the entire area required for the BISR block is received as 8636 nm² and is compared with the area report of the said existing methods. The comparison is

Table VII Features wise comparison BISR approaches

	Cresta [27]	Counting Threshold [31]	paralogous gene [24]	Proposed
Fault Detection support (MTC)	No	No	No	Yes
Spare rows	3	3	16 Cell	2
Spare columns	3	1	16 Cell	2
Fault Detection Method	Fault insertion	Fault insertion	Neighboring cell-based decision	MTC

Table VIII Area report of the proposed BISR block

Design	Cell	Cell Area (nm ²)	Net Area (nm ²)	Total Area (nm ²)
Proposed BISR	482	5876	2968	8636
Counting Threshold [31]	596	6036	3978	10023
Paralogous gene [24]	894	10437	5870	15392
Cresta [27]	908	11036	6013	16972

Table IX Area overhead, timing, and power comparison of BISR

	Area	Power	Area overhead	Timing
Cresta [27]	High	Average	High	Moderate
paralogous gene [24]	High	High	High	Fast
Counting Threshold [31]	Average	Average	Moderate	Moderate
Proposed BISR	Low	Low	Low	Fast

Table X Area overhead calculation for both the blocks

MTC block obtained area	BISR block obtained area
Memory Controller area- 10071 nm ² Memory area- 396719 nm ² Area overhead = 2.5%	BISR area=8636 nm ² Memory area= 396719 nm ² Spare Memory area=4028 nm ² Area overhead = 3.19%

tabulated, in Table VIII.

The power consumption and the timing have also computed for the presented BISR block. They are relatively compared with the obtained power, area, and timing results of the existing approaches, given in Table IX. The total memory (8Kx64) area received is 396719 nm². It is the same as in the case of MTC. The obtained BISR area and spare memory area is 8636 nm², and 4028 nm² respectively, which is approximately 3.19% of the 8k memory area, and it is given in the Table X.

It has noticed that the proposed methods of the MTC and BISR are more efficient in terms of area overhead, power, and timing when compared with the existing techniques of memory testing and self-repair. The area overhead is only a limitation of this proposal, but it cannot ignore.

4. Conclusion

Memory test and repair technique has implemented for SoC based devices. The functional and logical test has carried out. The functional analysis computes the availability of faults and their types in memory, which is under examination. The logic synthesis has carried out to calculate the area overhead, timing, and power consumption. In this research, we considered an 8Kx64 memory to conduct the test. It obtained from the results, and the area overhead is approximately 2.5% and 3.1% for MTC and BISR respectively when they integrate with 8Kx64 memory. It has noticed from the results that the overhead area is decreases by increasing the memory size. The only limitation of this approach is a little area penalty with it integrate to any SoC. The proposed controllers can easily modify for any future SoC design and blend at the time of manufacturing.

Acknowledgments

This project was supported by the Deanship of Scientific Research at Prince Sattam bin Abdulaziz University under the research project # 2019/01/11244.

References

- [1] A. Bosio, *et al.*: *Advanced Test Methods for SRAMs: Effective Solutions for Dynamic Fault Detection in Nano Scaled Technologies* (Springer, 2010) 1 (DOI: [10.1007/978-1-4419-0938-1](https://doi.org/10.1007/978-1-4419-0938-1)).
- [2] A. Bosio, *et al.*: *Advanced Test Methods for SRAMs: Effective Solutions for Dynamic Fault Detection in Nanoscaled Technologies* (Springer-Verlag US, 2009) (DOI: [10.1007/978-1-4419-0938-1](https://doi.org/10.1007/978-1-4419-0938-1)).
- [3] D.-M. Chang, *et al.*: “A built-in redundancy-analysis scheme for random access memories with two-level redundancy,” *J. Electron. Test* **24** (2008) 181 (DOI: [10.1007/s10836-007-5032-4](https://doi.org/10.1007/s10836-007-5032-4)).
- [4] M.A. Ahmed, *et al.*: “FPGA based high speed memory BIST controller for embedded applications,” *IJST* **8** (2015) 1 (DOI: [10.17485/ijst/2015/v8i33/76080](https://doi.org/10.17485/ijst/2015/v8i33/76080)).
- [5] O. Hasan, *et al.*: “Formal probabilistic analysis of stuck-at fault in reconfigurable memory array,” *7th Int. Conf. IFM* (2009) 277 (DOI: [10.1007/978-3-642-00255-7_19](https://doi.org/10.1007/978-3-642-00255-7_19)).
- [6] M.A. Ahmed, *et al.*: “Embedded memory test strategies and repair,” *Int. J. Engg. (IJE) Trans. C: Aspects* **30** (2017) 839 (DOI: [10.5829/ije.2017.30.06c.03](https://doi.org/10.5829/ije.2017.30.06c.03)).
- [7] X. Peng, *et al.*: “Optimizing weight mapping and data flow for convolutional neural networks on processing-in-mem architectures,” *IEEE Trans. Circuits Syst. I, Reg. Papers* **67** (2020) 1333 (DOI: [10.1109/tcsi.2019.2958568](https://doi.org/10.1109/tcsi.2019.2958568)).
- [8] B. Chong and S. Zhang: “Algorithm-based fault tolerance for discrete wavelet transform implemented on GPUs,” *J. Sys. Arch.* **108** (2020) 101823 (DOI: [10.1016/j.sysarc.2020.101823](https://doi.org/10.1016/j.sysarc.2020.101823)).
- [9] Y. Wang, *et al.*: “The improvement of march C+ algorithm for embedded memory test,” *NCCET 2015: Computer Engineering and Technology* (2016) 31 (DOI: [10.1007/978-3-662-49283-3_4](https://doi.org/10.1007/978-3-662-49283-3_4)).
- [10] J. Kim, *et al.*: “Hardware-efficient built-in redundancy analysis for memory with various spares,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **25** (2017) 844 (DOI: [10.1109/tvlsi.2016.2606499](https://doi.org/10.1109/tvlsi.2016.2606499)).
- [11] I. Mrozek and V. Yarmolik: “Pseudo-exhaustive random access memory testing based on march tests with random background variation,” *2018 IEEE EWDTS* (2018) (DOI: [10.1109/ewdts.2018.8524824](https://doi.org/10.1109/ewdts.2018.8524824)).
- [12] M.A. Ahmed and A.M. Abuagoub: “MBIST controller based on March-ee algorithm,” *Journal of Circuits, Systems and Computers* (2020) (DOI: [10.1142/S0218126621501607](https://doi.org/10.1142/S0218126621501607)).
- [13] M. Koyanagi: “Recent progress in 3D integration technology,” *IEICE Electron. Express* **12** (2015) 20152001 (DOI: [10.1587/elex.12.20152001](https://doi.org/10.1587/elex.12.20152001)).
- [14] E. Buslowska and V. Yarmolik: “Multi-run march tests for pattern sensitive faults in RAM,” *2018 IEEE EWDTS* (2018). (DOI: [10.1109/ewdts.2018.8524643](https://doi.org/10.1109/ewdts.2018.8524643)).
- [15] P.K. Veenstra: *Random Access Memory Testing: Theory and Practice, The Gains of Fault Modelling* (Trefw.: elektronische schakelingen, Nov-2020).
- [16] J.-F. Li, *et al.*: “Diagnostic data compression techniques for embedded memories with built-in-self-test,” *J. Electron. Test. Theory Appl.* **18** (2002) 515 (DOI: [10.1023/A:1016557927479](https://doi.org/10.1023/A:1016557927479)).
- [17] C.-W. Wang, *et al.*: “Fault pattern oriented defect diagnosis for memories,” *Proc. ITC* (2003) 29 (DOI: [10.1109/TEST.2003.1270822](https://doi.org/10.1109/TEST.2003.1270822)).
- [18] L.-T. Wang, *et al.*: in *VLSI Test Principles and Architectures*, ed. L.-T. Wang, *et al.* (Elsevier, 2006) 37 (DOI: [10.1016/B978-012370597-6/50006-8](https://doi.org/10.1016/B978-012370597-6/50006-8)).
- [19] L.-T. Wang, *et al.*: *VLSI Test Principles and Architectures: Design for Testability* (Elsevier, 2006) (DOI: [10.1016/b978-0-12-370597-6.x5000-8](https://doi.org/10.1016/b978-0-12-370597-6.x5000-8)).
- [20] C.-W. Wang, *et al.*: “A built-in self-test and self-diagnosis scheme for hetero SRAM clusters,” *Proc. 10th ATS* (2001) 103 (DOI: [10.1109/ATS.2001.990267](https://doi.org/10.1109/ATS.2001.990267)).
- [21] G.P. Acharya and M.A. Rani: “Survey of test strategies for system-on chip and it’s embedded memories,” *2013 IEEE Recent Advances in Intelligent Computational Systems (RAICS)* (2013) (DOI: [10.1109/raics.2013.6745473](https://doi.org/10.1109/raics.2013.6745473)).
- [22] S. Hamdioui, *et al.*: “A new test paradigm for semiconductor memories in the nano-era,” *Proc. Asian Test Symposium* (2011) 347 (DOI: [10.1109/ats.2011.87](https://doi.org/10.1109/ats.2011.87)).
- [23] N. Li, *et al.*: “Logic BIST: state-of-the-art and open problems,” (2015) arXiv:1503.04628v1 [cs.AR].
- [24] K. Cho, *et al.*: “A survey of repair analysis algorithms for memories,” *ACM Computing Surveys* **49** (2016) 47 (DOI: [10.1145/2971481](https://doi.org/10.1145/2971481)).
- [25] K. Cho, *et al.*: “An efficient built-in self-repair scheme for area reduction,” *ISOC* (2017) (DOI: [10.1109/isocc.2017.8368791](https://doi.org/10.1109/isocc.2017.8368791)).
- [26] K. Jamal, *et al.*: “Test pattern generation using thermometer code counter in TPC technique for BIST implementation,” *Microprocess. Microsyst.* **71** (2019) 102890 (DOI: [10.1016/j.micpro.2019.102890](https://doi.org/10.1016/j.micpro.2019.102890)).
- [27] T. Kawagoe, *et al.*: “A built-in self-repair analyzer (CRESTA) for embedded DRAMs,” *Proc. of the Int. Test Conference* (2000) 567 (DOI: [10.1109/TEST.2000.894250](https://doi.org/10.1109/TEST.2000.894250)).
- [28] C.-L. Su, *et al.*: “A built-in self-diagnosis and repair design with fail pattern identification for memories,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **19** (2011) 2184 (DOI: [10.1109/TVLSI.2010.2073489](https://doi.org/10.1109/TVLSI.2010.2073489)).
- [29] K.H. Choi, *et al.*: “A decoupled bit shifting technique using data encoding/decoding for DRAM redundancy repair,” *IEICE Electron. Express* **14** (2017) 20170385 (DOI: [10.1587/elex.14.20170385](https://doi.org/10.1587/elex.14.20170385)).
- [30] M.I. Masnita, *et al.*: “March-based SRAM diagnostic algorithm for distinguishing stuck-at and transition faults,” *IEICE Electron. Express* **6** (2009) 1091 (DOI: [10.1587/elex.6.1091](https://doi.org/10.1587/elex.6.1091)).
- [31] K. Gopalan and S. Pothiraj: “A saboteur and mutant based built-in self-test and counting threshold-based built-in self repairing mechanism for memories,” *Journal of Ambient Intelligence and Humanized Computing* (2020) (DOI: [10.1007/s12652-020-02284-5](https://doi.org/10.1007/s12652-020-02284-5)).