2024 20th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD) | 979-8-3503-5192-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/SMACD61181.2024.10745441

FPGA Implementation of an Elementary ReRAM Memory Control Unit

Patricio Carrasco
Dept. of Electronic Engineering
Universidad Tecnica Federico Santa Maria (UTFSM)
Valparaiso, Chile
patricio.carrascoo@sansano.usm.cl

Ioannis Vourkas
Dept. of Electronic Engineering
Universidad Tecnica Federico Santa Maria (UTFSM)
Valparaiso, Chile
ioannis.vourkas@usm.cl

***Abstract*— The resistive RAM (ReRAM) technology is continuously attracting attention from the relevant industry who wish to exploit the potential of such technological advances in emerging applications. To incorporate such memory blocks in any application, a dedicated ReRAM memory control unit (ReMCU) is required, to carry out the READ and WRITE operations correctly and efficiently. Such ReMCU design depends on the target ReRAM organization and its peripheral circuitry. As a first approach to the hardware (HW) development of a ReMCU IP block, we developed a dedicated ReMCU module in digital HW to control the READ and WRITE operations from/to different ReRAM locations. The proposed design assumes a target ReRAM topology which allows independent access to the 1T1R cells in any bitline, thus facilitating parallel READ and WRITE operations with the simultaneous application of SET and RESET voltages. Our first prototype was implemented and tested in a field programmable gate array (FPGA) platform. Using the *Vivado Design Suite*, we demonstrate experimentally the correct processing and the parallel execution of the READ and WRITE instructions, received from a host PC via a serial communication protocol.**

Keywords—memristive device; resistive switching; ReRAM; memory control unit; FPGA; Verilog HDL; Vivado Design Suite

I. Introduction

Among different technologies for information storage, currently under research and development [1], the resistive RAM (ReRAM) alternative, which employs resistive switching (also called “memristive”) cells to store binary data in form of resistance (i.e., a high/low resistive state to represent logic “0”/“1”, or vice versa) [2], is continuously attracting attention from the relevant industry [3], [4] who wish to exploit the potential of such technological advances in emerging applications [5].

In voltage-based driving schemes for ReRAM, normally a small voltage is used for READ operations, whereas higher voltages, above the switching thresholds of the cells, are used for the WRITE operations, which are commonly associated to the resistive switching events (SET and RESET). The development of a memory control unit for ReRAM (ReMCU) should consider all the details of the target ReRAM module. These include the precise structure of the memory cells (e.g., 1R, 1T1R, etc.) [6], their switching performance (bipolar or unipolar), and of course the internal organization of the cells (number of tiles, word-/bit-lines per tile, parallel or independent access, etc.), which altogether define the requirements for the peripheral circuits that perform the bit-/word-line activation. In any computing system which employs a ReRAM module, an elementary ReMCU should carry out successfully the READ/WRITE operations, whereas more sophisticated ReMCU implementations could perform in a more comprehensive way, supporting progressive WRITE operations [7], multi-level storage, and anticipating potential errors produced by the nonidealities of cell performance (variability, incomplete switching, stuck-at errors [8], to name a few).

Several crossbar topologies for ReRAM modules exist in the literature. Particularly, the topology shown in [9], [10], was recently explored for in-memory computations in [11], and the authors of [11] proposed the design of a peripheral circuit that allows independent access to the 1T1R cells which are in different bit-lines. This property facilitates the parallel access to all cells of a memory word, which is needed for READ operations, but it is particularly interesting for WRITE operations too, since SET and RESET processes on bipolar cells could take place simultaneously in the accessed memory word. Aiming to take advantage of the functionality of such ReRAM design, we build upon the previous simulation results in [12] and developed a dedicated ReMCU module in digital hardware (HW), which processes instructions to READ and WRITE digital data from/to different ReRAM addresses.

More specifically, in this work we briefly describe the properties of the target memory circuitry and the assumptions we made for the ReRAM cell operation to carry out the design of the internal state diagram of our elementary ReMCU intellectual property (IP) HW. The first prototype was implemented in a field programmable gate array (FPGA) platform. Using the *Vivado Design Suite*, we demonstrate experimentally the correct processing and parallel execution of the READ and WRITE instructions, received from a host PC via serial communication. Such design constitutes a first approach to the hardware development of ReMCU IPs oriented to the industrial sector, and will serve as a benchmark design to evaluate tradeoffs between resource requirements, delay, and design complexity while considering different memory organization options and memory access types.

II. Features of the Target ReRAM Topology

In Fig. 1 we present the design concept of a ReMCU, for a ReRAM structure as proposed in [11]. The design has independent access drivers for every bitline (only one bitline is shown). The 1T1R memory cells that are vertically aligned, are simultaneously selected via the wordline (WL) signals that connect to the gate terminals of the select transistors. More details about operation of the such topology, can be found in [11]. Note that the sensing block compares the value read with two thresholds to distinguish logic “1” from logic “0” (binary stored data), considering an undefined region in-between. The read information is eventually stored in a local register and can be handled by the ReMCU. With this target memory topology in mind for one bit/cell storage, we designed the ReMCU module whose top-level diagram is shown in Fig. 2, for a memory organization considering one tile, even though our design is parameterizable and can generally handle bitlines of every word that can be distributed along *i* tiles, each having *N* bitlines accessed through *k* wordlines, as described in [12].

Supported by the Chilean research grants ANID-Basal FB0008 and FONDECYT Regular No. 1221747.

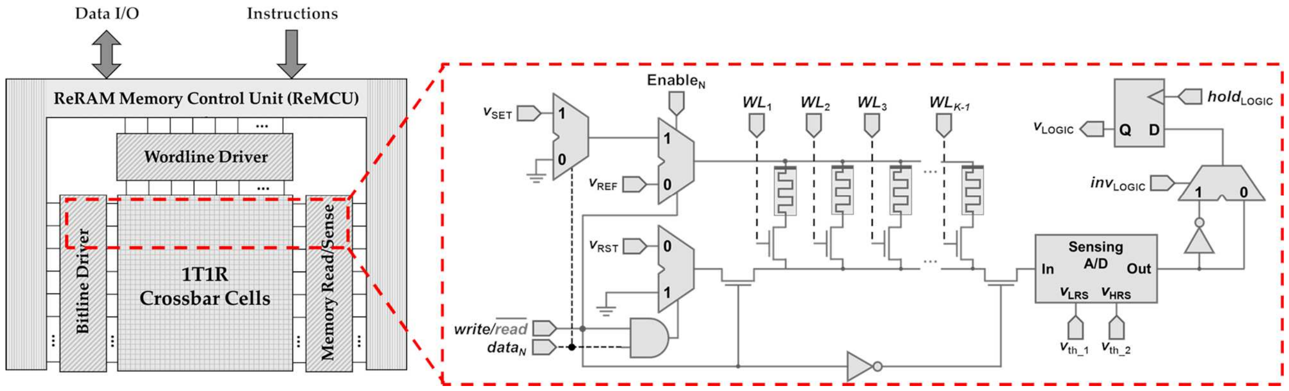


Fig. 1 General block diagram describing the assumed topology of a resistive memory (ReRAM) system, with the dedicated MCU, alongside the schematic of a single row (a single bitline - BL) of the memory array with the BL driver on the left side and the sensing circuitry on the right side. Adapted from [11].

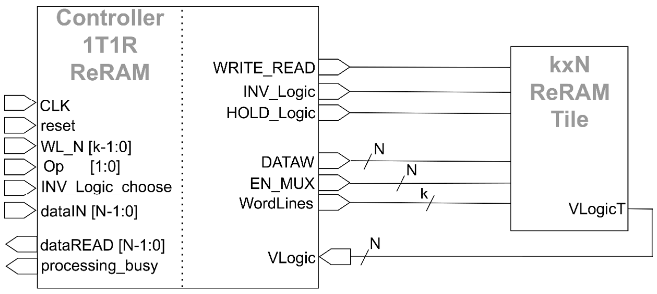


Fig. 2 Top-level block diagram of the developed ReMCU and the target memory module to be controlled. Only one tile is shown. However, the memory can be organized in several more tiles, each of *k*×*N* dimensions.

Such elementary ReMCU module allows parallel READ and WRITE operations, simultaneously accessing all the *N* bitlines of any memory address. To achieve parallel access, the *EN_MUX* signal is used to activate each bitline independently. The *ENABLE* signal in Fig. 1 for each bitline is effectively the same as the *EN_MUX* signal in Fig. 2. To store the result of a READ operation, the *VlogicT* signal is used, which must be *N* bits long. The controller stores the accessed information via *Vlogic*, which is also *N* bits long. Subsequently, for the WRITE process which involves storing any combination of digital values in memory simultaneously, it is necessary to have an independent input data signal for each bitline, which here is *DATAW*, also *N* bits long, and indicates what digital data are to be stored in memory. The rest of the input signals of the ReMCU module will be described next in the following sections. The ReMCU module was designed in Verilog HDL, and the state diagram that describes the functionality of the module is presented in Fig. 3.

The system starts in the IDLE state, where all internal signals take their default value. From the IDLE state, only two transitions are possible, corresponding to READ and WRITE operations. When the *Op* signal takes value 1, a read operation is initiated with the READ state which has five internal phases, identified with the value of the *CRead* counter. The *processing_busy* signal is set to logic "1" to avoid interruptions to the current process. Additionally, *Write_Read* is set to logic "0" and *WordLines* takes the value of *WL_N* to define the target memory address. All *EN_MUX* bits are set to access all bitlines simultaneously. After one cycle, *CRead* increases to 1 and the *INV_Logic* is set to the value of *INV_logic_choose*, to indicate whether the values read from the memory cells should be inverted before local storage. An additional counter is used to monitor the number of cycles that the reading phase must last, until the correct digital value can be obtained at the output of the reading circuits. This parameter depends on the analog circuitry in the periphery, so here is set to an arbitrary value for testing. Once this time has elapsed, *CRead* increases to 2, and the *HOLD_Logic* signal is activated to capture the read information to the local register, represented by *VlogicT*. After one cycle, *CRead* increases to 3, where each one of the *VlogicT* values are concatenated at the input of the ReMCU, forming *Vlogic*. This value is eventually assigned to the *dataREAD* output. Finally, *CRead* increases to 4, and the READ process is complete, so the system returns to the IDLE state.

One the other hand, when *Op* takes value 3, a WRITE operation is requested and the system goes to the WRITE

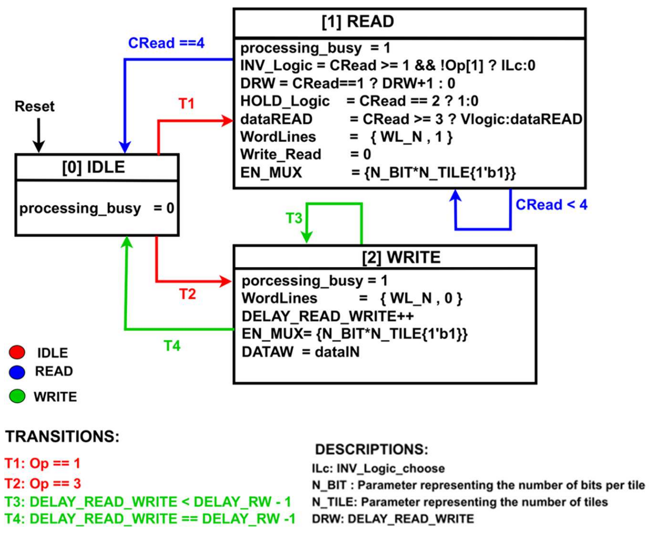

[1] READ
processing_busy = 1
INV_Logic = CRead >= 1 && !Op[1] ? ILc:0
DRW = CRead==1 ? DRW+1 : 0
HOLD_Logic = CRead == 2 ? 1:0
dataREAD = CRead >= 3 ? Vlogic:dataREAD
WordLines = { WL_N , 1 }
Write_Read = 0
EN_MUX = {N_BIT*N_TILE{1'b1}}
CRead ==4
Reset
T1
[0] IDLE
processing_busy = 0
CRead < 4
T3
T2
[2] WRITE
porcessing_busy = 1
WordLines = { WL_N , 0 }
DELAY_READ_WRITE++
EN_MUX= {N_BIT*N_TILE{1'b1}}
DATAW = dataIN
T4
● IDLE
● READ
● WRITE
TRANSITIONS:
T1: Op == 1
T2: Op == 3
T3: DELAY_READ_WRITE < DELAY_RW - 1
T4: DELAY_READ_WRITE == DELAY_RW -1
DESCRIPTIONS:
ILc: INV_Logic_choose
N_BIT : Parameter representing the number of bits per tile
N_TILE: Parameter representing the number of tiles
DRW: DELAY_READ_WRITE

Fig. 3 State diagram with all possible transitions during the READ and WRITE operations, supported by the designed ReMCU IP HW module.

state, where all the bitlines are also activated, for a parallel operation. Moreover, the *DATAW* signal adopts the value of *dataIN* to act accordingly on the WRITE drivers in the peripheral circuitry of the target ReRAM memory. In this state, like in the READ process, a delay is purposely introduced whose duration is defined by *DELAY_READ_WRITE* (DRW). The latter reflects the number of clock cycles that any WRITE pulse should last. This depends on the performance of the driving circuitry and on the response time of the memristive devices, but here is set arbitrarily for testing purposes. Once this time has elapsed, the WRITE pulse ends and the system returns to the IDLE state.

III. Description of the Experimental Setup

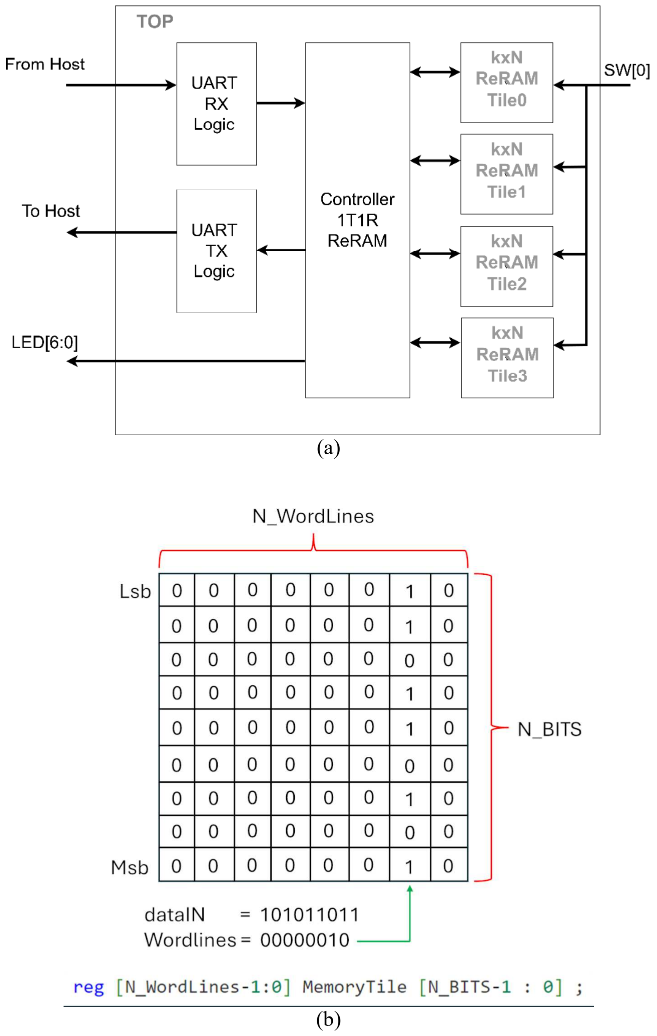
Testing our design on FPGA allows evaluating whether there are integrity or compatibility problems between the ReMCU and the assumed ReRAM architecture, facilitating subsequent adjustments. Likewise, it allows carrying out a performance evaluation and knowing the specific resource requirements, which reflect the complexity of the design. In our experimental setup we used the Nexys A7-100T development board which carries an Artix-7 FPGA. The tests were carried out with the *Vivado Suite* 2023.2 software on a PC with 16 GB of memory, and the HTerm 0.8.8 software for the serial communication between the host PC and the development board, for the delivery of instructions and the reception of data read from the memory. The latter is emulated in digital HW, not considering real ReRAM cell performance.

The high-level diagram of the complete design is presented in Fig. 4(a). The host PC communicates the required instructions through a UART interface. The instructions in their general format include (i) a flag that indicates the operation to be performed, (ii) the data to be stored (in case of a WRITE process), (iii) the target memory address, and (iv) the option to invert the value of the read data. In addition to the ReMCU module, the two modules corresponding to the UART protocol were also designed and implemented in the same FPGA. Each instruction received by the host PC is processed by the UART RX Logic block and the corresponding signals are activated in the ReMCU module to carry out the operation. Once the operation is successful, the data read from memory is sent to the host PC through the UART TX Logic module. The memory on which the ReMCU module operates is emulated in Verilog HDL with register arrangements in the form of a matrix of *N_WordLines* × *N_BITS*. The emulation strategy is summarized in Fig. 4(b).

For the tests that were carried out, the emulated memory was configured with 4 tiles, each having 8 bitlines, resulting in a complete 32-bit word. For simplicity, we assumed only 7 memory words (wordlines). The ReMCU module was configured appropriately to drive a target memory with these characteristics. The *DELAY_READ_WRITE* for reading and writing pulses was set equal to 3 clock cycles, and the system clock frequency for this design was 100MHz. After completing the implementation, the resource utilization report in *Vivado Suite* showed 449 Slice LUTs (0.7082% of the total) and 88 Slice Registers (0.0694% of the total) for the ReMCU module, while each of the four emulated memory tiles used 64 Slice Registers. These numbers indicate a compact ReMCU design, which leaves plenty of room for improvements to perform more comprehensive memory operations, as suggested elsewhere [12], [13]. Moreover, the timing report indicates a Worse Negative Slack (WNS) of 6.225ns.

IV. Experimental Results

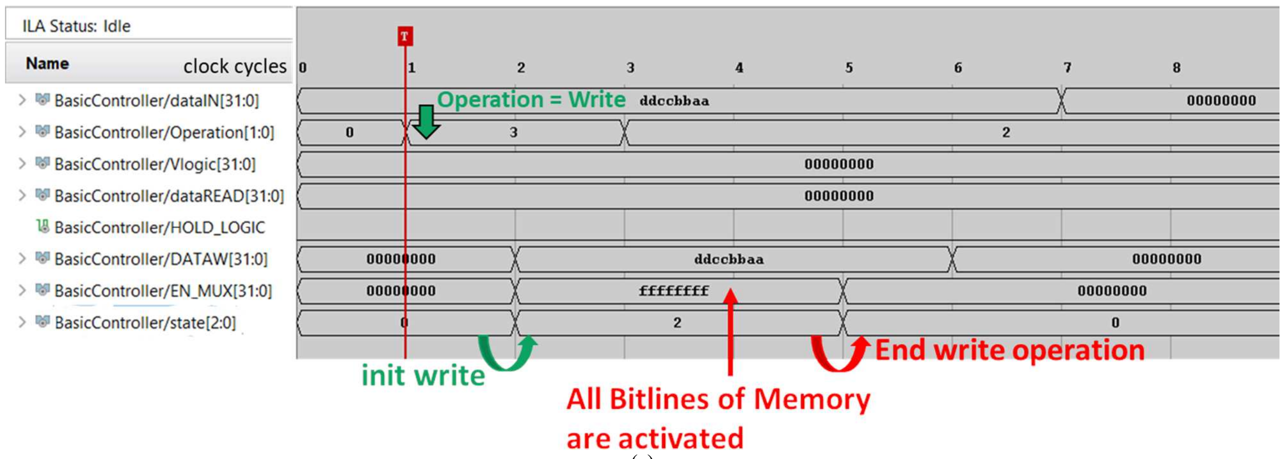
Some experimental results of the operation of the designed system are presented in Fig. 5. The debugging tool "Integrated Logic Analyzer (ILA)" of the *Vivado Suite* was used to analyze the signals of interest. The first instruction sent by the host PC had the hexadecimal value "0x03AABBCCDD02", where "0x03" defines that the operation to be executed is a WRITE process, "0xAABBCCDD" represents the value to be stored (*dataIN* = 0xDDCCBBAA), and "0x02" represents the memory address to be accessed, which here corresponds to the first valid accessible wordline of the emulated memory.



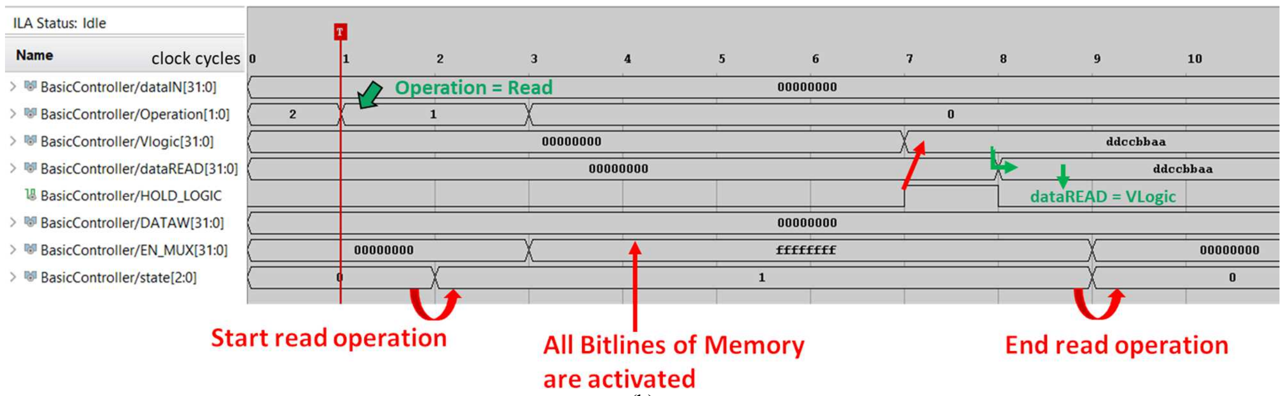
```
reg [N_WordLines-1:0] MemoryTile [N_BITS-1 : 0] ;
```

Fig. 4 (a) High-level block diagram of the complete design implemented in FPGA. (b) Schematic of the emulation of a tile on the FPGA, illustrating the division of the array based on parameters *N_WordLines* and *N_BITS*. The *dataIN* and *Wordlines* values are shown to exemplify how data is stored.

As we can observe in Fig. 5(a), all the bitlines are activated to simultaneously write in memory all the values contained in *dataIN*. The WRITE wait time was 3 cycles, as expected, according to what was mentioned above for the ReMCU system configuration. On the other hand, Fig. 5(b) shows the experimental results for a READ operation. More specifically, the host PC sent the hexadecimal value "0x010000000002", where the value "0x01" indicates that a READ operation should be performed, whereas "0x02" indicates the memory address to be accessed, which is the same as before. Note that, this operation is performed just after having performed the writing shown previously in Fig. 5(a). We observe that the READ operation goes through all the five different stages, in accordance with what was specified in the state diagram. After the predefined delay time, equal to 3 clock cycles, the *HOLD_LOGIC* signal is activated to save the data of all accessed bitlines in *Vlogic*. Then, in the next cycle, the value of *Vlogic* is saved in *dataREAD*, which is subsequently sent via the UART_TX_Logic module to the host PC for verification purposes. We note that the operation lasted 7 clock cycles, as expected. Note also that, the system transitions successfully to the IDLE state, after concluding each memory operation, waiting for upcoming instructions.


ILA Status: Idle
Name
clock cycles
0 1 2 3 4 5 6 7 8
BasicController/dataIN[31:0]
BasicController/Operation[1:0]
BasicController/Vlogic[31:0]
BasicController/dataREAD[31:0]
BasicController/HOLD_LOGIC
BasicController/DATAW[31:0]
BasicController/EN_MUX[31:0]
BasicController/state[2:0]
Operation = Write
ddccbbaa
00000000
0
3
2
00000000
00000000
00000000
ddccbbaa
00000000
00000000
ffffffff
00000000
0
2
0
init write
All Bitlines of Memory are activated
End write operation

(a)


ILA Status: Idle
Name
clock cycles
0 1 2 3 4 5 6 7 8 9 10
BasicController/dataIN[31:0]
BasicController/Operation[1:0]
BasicController/Vlogic[31:0]
BasicController/dataREAD[31:0]
BasicController/HOLD_LOGIC
BasicController/DATAW[31:0]
BasicController/EN_MUX[31:0]
BasicController/state[2:0]
Operation = Read
00000000
2
1
0
00000000
ddccbbaa
00000000
ddccbbaa
dataREAD = VLogic
00000000
00000000
ffffffff
00000000
0
1
0
Start read operation
All Bitlines of Memory are activated
End read operation

(b)

Fig. 5 Experimental results concerning (a) the WRITE operation of data "0xDDCCBBAA" to the first word of the emulated memory, and (b) the READ operation of the contents of the same memory word. Annotations are included in each plot to facilitate comprehension of the sequence of signal activation.

V. Conclusions

The design of a ReMCU module was experimentally validated in a target FPGA platform. The design constitutes an elementary version of a ReRAM memory controller, to be later used as benchmark for comparisons with other more comprehensive versions of the developed HW, which aim to improve reliability of WRITE operations through progressive feedback-assisted iterations. Although a specific memory organization was assumed, the HDL description of the ReMCU module is adaptable and supports both sequential and parallel operations, as well as per-tile operations, which are under research and were not shown here due to space limitations. The tests validate the success of such first approach to the development of a dedicated ReMCU IP; which is a key element to facilitate incorporation of ReRAM products in future applications. On going work concerns evaluating the delay Vs. reliability trade-offs arising in different memory accessing schemes, and their area overhead.

References

[1] G. Molas, and N. Etienne, "Advances in Emerging Memory Technologies: From Data Storage to Artificial Intelligence," *Applied Sciences*, vol. 11, no. 23, pp. 11254, 2021

[2] D. Ielmini, "Resistive switching memories based on metal oxides: mechanisms, reliability and scaling," *Semicond. Sci. Technol*., vol. 31, no. 063002, 2016

[3] WeebitNano, "A Quantum Leap in Emerging Memory Technology," [Online]. Available: https://www.weebit-nano.com/technology/

[4] Crossbar Inc, "ReThink IoT with ReRAM," [Online]. Available: https://www.crossbar-inc.com/

[5] I. Vourkas, *et al*., "Ubiquitous memristors in multi-level memory, in-memory computing, data converters, clock generation and signal transmission," in: P. Dimitrakis, I. Valov, and S. Tappertzhofen (Eds.) "*Metal oxides for non-volatile memory, materials, technology and applications*," Elsevier, 2022, pp. 445-463

[6] J. B. Roldán, *et al*, "Variability in Resistive Memories," *Adv. Intell. Syst*. No. 2200338, 2023

[7] L. Gao, P.-Y. Chen, S. Yu, "Programming Protocol Optimization for Analog Weight Tuning in Resistive Memories," *IEEE Electron Device Letters*, vol. 36, no. 11, pp. 1157-1159, 2015

[8] S. Kannan, *et al*., "Modeling, Detection, and Diagnosis of Faults in Multilevel Memristor Memories," *IEEE Trans. on Computer-Aided Design of Integr. Circuits and Systems*, vol. 34, no. 5, pp. 822-834, 2015

[9] C. Yakopcic, *et al*., "Hybrid crossbar architecture for a memristor based memory," 2014 *IEEE National Aerospace and Electronics Conference* (*NAECON*), Dayton, USA, June 24-27

[10] C. Zhao, *et al*., "Intra-array Non-Idealities Modeling and Algorithm Optimization for RRAM-based Computing-in-Memory Applications," 2021 *IEEE Int. Conference on ASIC* (*ASICON*), Kunming, China, Oct. 26-29

[11] C. Fernandez, A. Cirera, and I. Vourkas, "Design Exploration of Threshold Logic in Memory and Experimental Implementation using Known Memristors," *Int. J. of Unconventional Comp.*, vol. 18, no. 2-3, pp. 249-267, Jun. 2023

[12] P. Carrasco, and I. Vourkas, "On the HW Design of a Memory Control Unit Oriented to the Resistive Memory Technology," 2024 *Panhellenic Conf. on Electronics and Telecommunications* (*PACET*), Thessaloniki, Greece, March 28-29

[13] J. Cayo, and I. Vourkas, "A Comprehensive Simulation Framework to Validate Progressive Read-Monitored Write Schemes for ReRAM," 2023 *Spanish Conference on Electron Devices* (*CDE*), Valencia, Spain, June 06-08