uMemristorToolbox: Open source framework to control memristors in Unity for ternary applications

Steven Bos
Dept. of Science and Industry systems
University of South-Eastern Norway
Kongsberg, Norway
steven.bos@usn.no

Henning Gundersen
Dept. of Science and Industry systems
University of South-Eastern Norway
Kongsberg, Norway
henning.gundersen@usn.no

Filippo Sanfilippo
Dept. of Engineering Sciences
University of Agder
Grimstad, Norway
filippo.sanfilippo@uia.no

***Abstract*—This paper presents uMemristorToolbox, a novel open source framework that reads and writes non-volatile ternary states to memristors. The Unity (C#) framework is a port of the open source Java project Memristor-Discovery and adds a closed-loop ternary memory controller to enable both PC and real-time embedded ternary applications. We validate the closed-loop ternary memory controller in an embedded system case study with 16 M+SDC Tungsten dopant memristors. We measure an average switching speed of 3 Hz, worst case energy usage of 1 µW per switch, 0.03% random write error and no decay in (non-volatile) state retention after 15 minutes. We conclude with observations and open questions when working with memristors for ternary applications.**

***Keywords*-multi-valued logic, ternary computing, ternary storage, memristor SDK**

I. Introduction

Ternary computers are the logical next step to create a post-binary world. The ternary number system is the closest discrete number system to the mathematically proven optimal *e*, the radix economy proof [1]. This proof looks at the cost of representing a digit, which is assumed to be proportional to the base. A higher base requires more hardware or more complexity to represent it stably. More importantly, the proof assumes that there is a proper measure of hardware complexity (the cost) expressed as *possible values * number of digits in that base* to represent number N. Memristor are shown to stably represent binary digits, but the same hardware can also process multiple-values, currently up to 92 [2]. With new technology such as CNTFET for ternary logic gates [3] and memristors, a post-binary world is the most economical choice and the next frontier in computing.

Ternary computers are not a new phenomenon. The Setun showed in 1958 that electrical ternary computers can be constructed [4]. However to represent 1 trit, three out of possible four states were used [5] making binary (or base 4) a more economical choice. That and other reasons [4] led to a decreased investment in ternary and increased investment in binary. In this paper we want to verify and measure a memristive circuit implementation. According to [6] theoretical and simulation papers vastly outnumber implementation and measure papers (85% vs 15%) in a multi-database survey of 142 top articles. The core focus of this paper is the introduction of uMemristorToolbox, an open source software framework that allow researchers to experiment with memristive circuit implementations for ternary or multiple-valued applications. The framework contains visualisation tools and hardware/software debug methods to use memristors as memory cells by reading and writing to them.

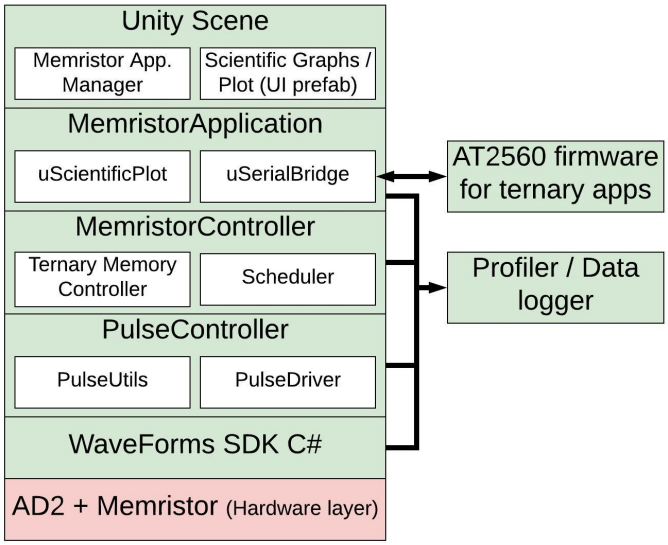


Fig. 1: uMemristorToolbox architecture

Using memristors as memory cells has various benefits over capacitor (eg. DRAM) or floating gate (eg. FLASH) based memory cells, although not all memristors have the same properties. The tested SDC (self-directed channel) memristor are non-volatile and can be used for both RAM and ROM operations. Compared to capacitor and floating gate memory cells considerable power consumption is saved as no periodic refresh rate for integrity is required and reading operations on the cells does not require discharging and recharging the cell. The current downside is that its switching speed is slow (in our case study) compared to traditional DRAM and FLASH. Research by [7] has shown that memristors can have sub-nanosecond switching speeds, but this has only been tested with binary thresholds.

Ternary applications have several benefits over their binary counterparts. Less space is required to store data. For example

1 byte of 8 bits can be stored in 6 trits, saving of 25%. This however has large overhead, since 9 full bits fit the same 6 trits saving 33% without any compression. The maximum theoretical gain of ternary over binary is log2(3) = 1.58 or 58 %. Processing and storing less data has other benefits too, reducing wiring complexity, energy consumption, and communication latency since fewer symbols have to be processed. The impact on society is therefore large, as ternary applications can reduce battery life, cost and size of electrical devices while improving performance.

This paper is structured in four parts. In the first part, the related work, we review literature that propose methods to control memristors. In the second part we present uMemristorToolbox and the results of the validation experiments. In the third part a case study that uses uMemristorToolbox is discussed, a real-time embedded ternary application. In the fourth part we discuss some measurements and observations and close with conclusions and future directions.

II. Related work

The memristor (or memory resistor) is, in circuit theory, the fourth fundamental element which uses the relationship between flux and electric charge for its operation. Chua postulated this component in 1971 [8] on the basis of the principle of symmetry with other components of the circuit, such as the resistor, inductor and capacitor. A memristor works as a variable resistor, the value of which depends on the current or voltage across the component, i.e. if there is a positive voltage across the memristor, its memristance decreases to a small value; if there is a negative voltage across the memristor, its memristance increases up to a high value. Theoretically, the memristor could potentially supplant state-of-the-art integrated electronic devices for advanced computing and digital and analog circuit applications. However, the memristor remained only of philosophical interest for more than 30 years [9] till HP Labs provided its physical implementation [10] based on a nanometer scale, TiO_2, thin film containing doped and undoped regions between two metal contacts, for its fabrication. Significant attention has been paid to the physical memristor model made by HP researchers, and many new studies have subsequently taken place in literature. Nevertheless, there are some significant shortfalls related to the HP's implementation, such as just having been only prototypically produced in the laboratory and retaining a low working speed. In 2012, Crupi, Pradhan and Tozer identified a new design to build circuits of neural synaptic memory using organic ion-based memristors [11]. However, the proposed model remains only as a proof of concept so far. In 2014, a flexible memristive system comprising a MoOx/MoS_2 heterostructure sandwiched between silver electrodes on a plastic foil was published by Bessonov et al in [12]. The manufacturing method of the proposed memristor is based exclusively on the use of two-dimensional layered transition metal dichalcogenides (TMDs) printing and solution processing technologies. This fabrication approach makes it possible to build memristors that are mechanically flexible, optically transparent and produced at low cost. However, the proposed system is still not available commercially to the best of authors' knowledge. The memristor, termed a self-directed channel (SDC) device was introduced by Dr Kris Campbell, in 2017 [13]. The SDC devices offer considerable advantages over other types of ion-conducting chalcogenide device, such as significantly higher processing and operating temperatures, and no photodoping required during manufacturing. This former feature makes the device manufacturing process achievable by simply using a generic commercially available sputter tool for all device layers, including the top electrode, for one in-situ film deposition stage. The SDC device is the first memristive component available commercially to researchers, students and electronics enthusiast worldwide [14]. This devise is distributed by Knowm. A simple Application programming interface (API) for automated memristor experiments and data collection is available together with a series of open source projects and libraries [15]. The former are usually very simple interfaces that are adopted by users such as researchers and industrial developers to add memristor capabilities to their applications by partially hiding the complexity of programming memristor devices. Nevertheless, these tools still need considerable programming skills to be integrated with user applications. This process can therefore often be a repetitive and time-consuming task, keeping developers from concentrating entirely on the software logic needed to accomplish the core task. Moreover, most of the currently available APIs do not offer flexible tools for executing extensive experiments with memristors for ternary storage. Developers are therefore forced to develop custom solutions that are often lacking in generality. To the best of authors' knowledge, an open-source real-time embedded framework to test and control memristors for ternary applications has not been released yet. This is the main contribution of this work.

III. uMemristorToolbox

This paper presents uMemristorToolbox [16], an open source GPL v3 framework to experiment with memristors and create multi-valued logic application in Unity C#. The framework architecture shown in Fig. 1 offers a software stack to hook into various levels of abstraction, allowing full control over the memristors. It supports hardware and software debugging and features a file data logger and scientific graph library to plot results. The repository contains a demo scene with currently 5 memristor applications ("experiments") discussed throughout this section. The repository also includes an AT2560 firmware for real-time embedded systems that interface with uMemristorToolbox applications.

A. Background

uMemristorToolbox initially started as a direct port of Knowm's Java based Memristor-Discovery [17] out of a desire to use a faster and more flexible prototyping environment. We implemented the Analog Discovery 2 C# SDK and developed various graphing tools mimicking the real-time diagrams found in the original project. To validate our port, we ran the Direct

Current (DC) and Checkerboard experiments on both projects and achieved identical results.

Several classes like the pulse controller, containing the PulseUtils and PulseDriver are direct ports of the original project, with minor modification to fit the C# best practices. Some original experiments are not (yet) ported. The Pulse experiment for example, requires a port of the JSpice analog circuit simulator first.

A key differentiation compared to the original project is the focus on multiple-valued logic applications. uMemristorToolbox features a ternary memory controller (described below). Three new experiments were created to analyse the memristors by controlling it with the ternary memory controller. The Retention experiment, The RandomWrite experiment and 1 Trit AnalogToDigital (ADC) experiment. They are described in more detail below.

B. Ternary memory controller

A key feature of this framework is the included closed-loop feedback ternary memory controller. The memory controller programs three resistance levels at 0-8k (logical 2) @ +1V, 8-100k (logical 1) @ +0.4V, >100k (logical 0) @ -2V using one or more 10Hz half-sine pulses. These values have empirically been found to be the most stable across the used memristors and is subject to future fine-tuning. The ternary memory controller has absolute control over the scheduler, allowing immediate read/write/erase operations to be scheduled if needed. The memory controller operation is shown in Fig. 1. The 25 ms wait periods (and 5ms wait periods for memristor selection) are reported in the original Java SDK, but further experimentation is required to see the influence on stability of these (costly) waits, potentially speeding up the ternary memory controller manifold. It should be noted that the algorithm does not guarantee that a desired state is written to the memristor, it merely attempts a fixed number of times. We only experimented with amplitude and number of pulse and did not experiment with pulse width or shape of the pulse. We noticed that by applying consecutive 1V pulses we can damage or "burn" the memristor, losing its typical pinched hysteresis loop behavior. The algorithm shows a single erase before writing the first write pulse try in the Write case, incurring another 25ms + Erase write delay penalty.

C. Validation

To validate the uMemristorToolbox framework and the ternary memory controller in particular, several experiments are performed. The testing platform was an Alienware m15 laptop with i9-8950HK CPU, GeForce RTX 2080 Max-Q and 32GB RAM. The software ran on Windows 10 with Unity 2019.2.2f1. Experiment 2 and 3 use the ternary memory controller. The memristor data in all three validation experiments is provided as open data.

D. Experiment 1: Pinched Hysteresis

The hallmark property of memristors are the zero-crossing current-conductance behavior, so-called "pinched hysteresis"

Algorithm 1: Ternary Memory Controller

```
   /* Init. Timer,Scheduler,UI,Logger  */
 1 Initialize();
 2 FileLogger.Log(time, ExperimentName);
 3 while Scheduler.IsActive do
 4     if Scheduler.Queue.Count > 0 then
 5         var task = Scheduler.Dequeue();
 6         ToggleMemristor(task.memristorId, ON);
 7         switch task.instruction do
 8             case READ do
 9                 StartOscilloscope(task);
10                 StartWaveformGenerator (task);
11                 var r = OscilloscopeToResistance();
12                 var trit = ResistanceToTrit(r);
13                 FileLogger.Log(time, task);
14                 Wait(25ms)
15             end
16             case WRITE do
17                 var tries = 0;
                   /* ERASE PASS                  */
18                 StartOscilloscope(task);
19                 StartWaveformGenerator (task);
20                 Wait(25ms);
21                 while DONE == FALSE do
                       /* WRITE PASS              */
22                     StartOscilloscope(task);
23                     StartWaveformGenerator (task);
24                     Wait(25ms) /* READ PASS    */
25                     StartOscilloscope(task);
26                     StartWaveformGenerator (task);
27                     var r = OscilloscopeToResistance();
28                     var trit = ResistanceToTrit(r);
29                     FileLogger.Log(time, task);
30                     if task.desiredState == trit then
31                         DONE = TRUE;
32                     else
33                         tries++;
34                         if tries >= 3 then
35                             DONE = TRUE;
36                             FileLogger.Log(time, "too
                                 many tries:", task);
37                         else
38                     end
39                     Wait(25ms)
40                 end
41             end
42         end
43         ToggleMemristor(task.memristorId, OFF);
44         UpdateUI/Console.Log(task.result);
45     end
46 end
47 FileLogger.Log(time, new ExperimentStatistics());
```

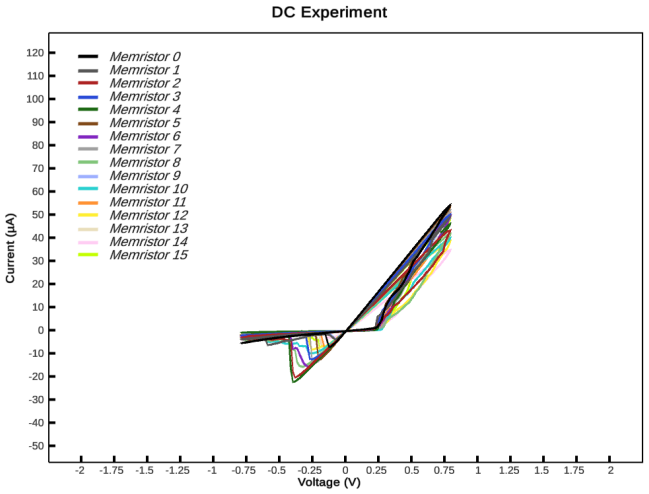


Fig. 2: Zero-crossing pinched hysteresis of 16 memristors

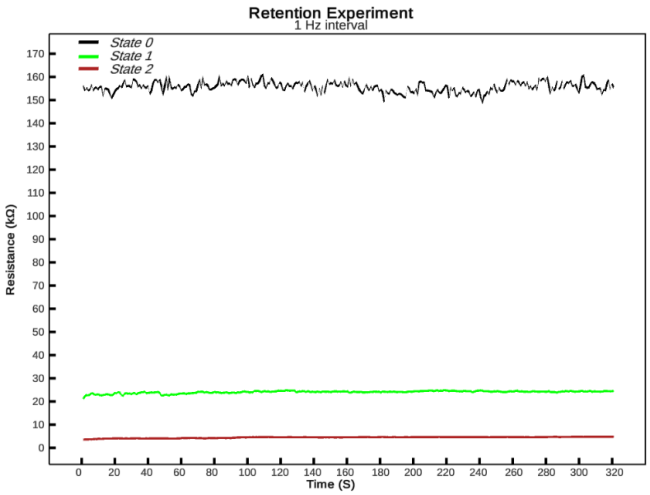


Fig. 3: Single Erase, Single Write, Multiple Read test with 1 Hz sample rate

loop. We select the "DC experiment" and test all 16 memristors for this behavior to determine the quality of the memristors. We use identical settings as published in the manual of the original Java version for a good comparison (+0,8V Triangle-UpDown wave signal at 2Hz), shown overlaid in Fig. 2

E. Experiment 2: Random Write

For this experiment we select the "Random Write" experiment and set N to be 100, so 100 0's, 100 1's, 100 2's are generated and scheduled in random order. We measure how often the desired (and written) state is the actual state and plot the result in a confusion matrix. The time to complete the task was 83 seconds.

F. Experiment 3: Data Retention

For this experiment we select the "Data Retention" experiment and set the read interval I to 1 Hz and T=320 seconds and another experiment with I = 1/60 Hz and T= 15. We write a single state to the memristor analyse how the resistance (and thus states) are affected by time and frequency of periodic read signals. After a complete trace, we erase the memristor and write another state until all states are measured.

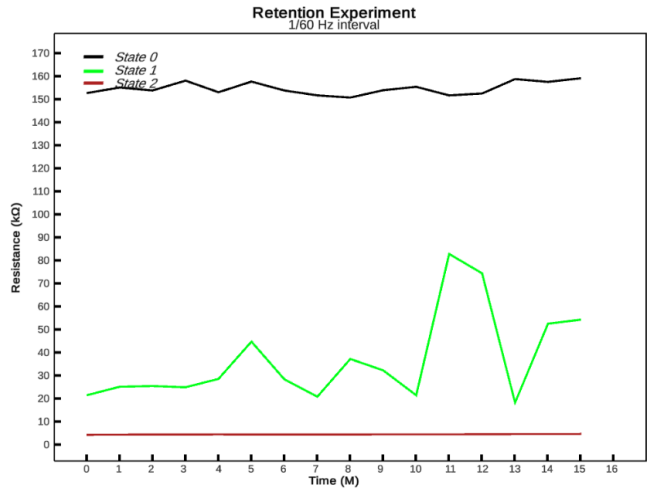
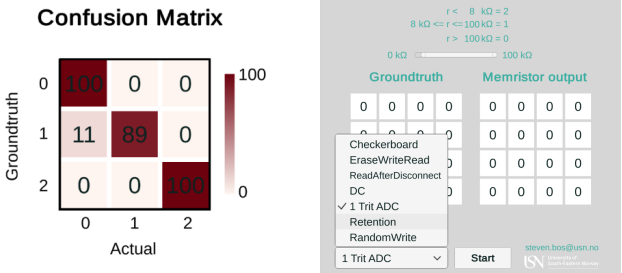


Fig. 4: Single Erase, Single Write, Multiple Read test with 1/60 Hz sample rate

IV. Case: Real-time embedded ternary application

To test uMemristorToolbox and the ternary memory controller, we developed a real-time embedded system with 16 button inputs and 16 x 4 LED outputs. The wiring schematic is presented in 8. The firmware on the embedded system is written in embedded C, directly programming the AT2560 but using the rich and affordable Arduino board to prototype various circuits. The ternary application, named one trit Analog To Digital Conversion (ADC) experiment was written in C#, interfacing over a serial bridge with asynchronous input and output. A system overview is shown in Fig 6

A file logger and UI event handler was connected to the memristor showing how the programmed logic values relate to the groundtruth. We have not done a quantitative analysis, but



(a) Confusion matrix

(b) User interface

Fig. 5: (a): Random write test of 100 0's, 100 1's and 100 2's in random order. (b): Unity user interface for 1 trit ADC experiment showing groundtruth and memristor output

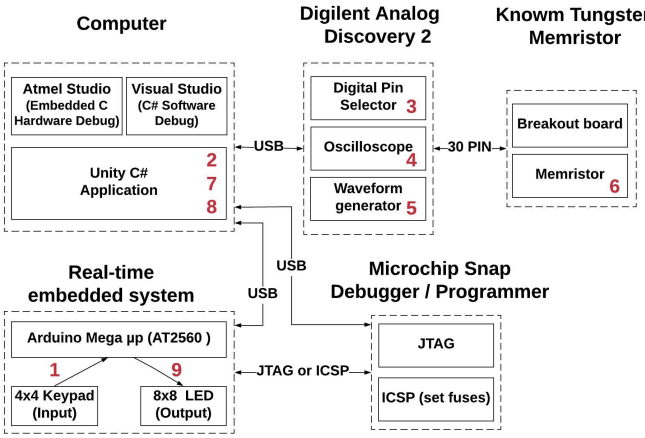


Fig. 6: System overview Ternary storage controller (1) Button interrupt, send button id. (2) Update trit state ground truth, send write and read signal to scheduler. (3) Select memristor. (4) Start oscilloscope for reading. (5) Start waveform generator for writing. (6) Update resistance in memristor. (7) Check if desired state, otherwise repeat step 4,5,6. (8) Deselect memristor, Write to log. (9) Update LED with new trit state.

our demonstration over the span of several weeks repeatedly show great correlation between keypad input and LED output in real-time. A data example from the 1 trit ADC experiment is shown in Fig 7

V. Discussion

Much is still unknown about using memristors in real-world applications. We could not find an extensive technical comparison of memristors as ternary memory cells against traditional DRAM (volatile) and FLASH (non-volatile) memory cells. What are the cost/MB if they are mass produced, MTBF, retention limits, silicon footprint and optimal switching speed? And, can the memristors discussed here be integrated or stacked with a hardware memory controller in existing silicon fabrication processes?

We have performed three experiments to validate the performance of the memristors. In the DC experiment in Fig. 2 we observe that all 16 memristors show similar hysteresis loops. These are similar to published in the Knowm Memristor manual. Note that the values are not deterministic and vary a little every experiment. After we burned the first memristor, the pinched hysteresis plot for that memristor changed, clearly showing it to be defective. The result was that we could not program the memristor in a high resistance state.

In the random write experiment in Fig. 5a we observe that the memristor controller fails in 11/300 times. We limited the amount of correction signals to 3 because the time penalty in this version is still high and will impact the use case in real-time applications. All reported errors were found in a confusion between writing a logical 1 while after 3 tries the logical value was a 0 (high resistance state, >100k). This could be improved by improving the memory controller to vary more parameters such as waveform, pulse width and a higher number of tries. Another approach might be to program the memristor in low resistance and slowly nudge the resistance using small (dynamically computed) negative pulses.

In the retention experiment in Fig. 3 we observe the read retention at 1 Hz interval over about 5 minutes. A read operation requires a memristor selection switch, setting a transistor, that could influence the memristance as reported in the manual. We do not observe any inconsistencies at 1 Hz interval, but have seen switching artifacts in earlier versions of our controller like in 9 . The behavior in Fig. 4 at 1 minute reading intervals over 15 minutes was more inconsistent. The memristance values were still within boundaries, hence the large chosen bandwidths for state 1 compared to state 0 and 2. Further testing to see how the memristance decays after hours or days or possibly longer is still needed.

Some final notes on working with memristors and ternary applications. We were careful by not exposing the memristor with large voltage amplitudes and placing two 5k series resistors like in [13]. Never the less, we exposed the memristor to +2V and did a few 1000 operations without showing any visible degradation, until we repeatedly and in quick succession exposed it to +1V pulses. We changed the memristor controller to protect the memristor for this pattern. Although temperature can influence the memristor as reported in [13] we used the memristor in various locations outside the lab. Future experiments will include temperature measurements and see its effects in Nordic environments.

VI. Conclusion

This work presents an open source framework to program memristors in Unity C# including various (real-time) interaction and visualisation tools and a firmware for interfacing with real-time embedded systems. The framework can be used to program multi-valued logic applications. We added a closed-loop ternary memory controller to program the memristor for 3-valued logic.

In a case study, we show that non-volatile ternary applications on real-time embedded devices are feasible with off-the-shelf memristors, however, various open research questions about the limits of using memristors in such applications remain open. The case study also reconfirms earlier studies reporting consistent current-conductance properties in commercially available, mass-produced memristors over several months. We encourage the research community to experiment with the Knowm Memristor-Discovery (Java) and this (Unity C#) SDK, replicate and share results as open data to start a dialogue in this new field.

VII. Future Work

Our research group is interested in the use of memristors for safety-critical applications embedded in autonomous collaborative robots. Ternary logic is perfectly suited for safety critical applications as the third state can be used to flag the status of a cell (eg. corrupt, not initialized, unknown, etc) on the hardware level. We are also exploring how the ternary memory