# Development System for Memristor Circuits

Jhon Faghih-Nassiri, Nicholas Maddy, Lucas Buckland, and Dmitri Strukov, *Member, IEEE*

***Abstract*— This paper reports on an experimental system comprising of an embedded 32-bit ARM microprocessor with ADC/DAC peripherals and supporting software designed for memristor circuit development. In particular, the system is tested with Pt/$TiO_{2-x}$/Pt memristors, and read and write operations with single memristive devices are demonstrated.**

**Keywords: ReRAM, memristor, development system**

## I. Introduction

Researchers interested in exploring memristor [1-4] formation, characterization, and applications often use very intricate equipment. This equipment is meant to operate on a variety of devices, and is not specific to memristors. As a result, it is also generally bulky, expensive, and can sometimes be slow due to software limitations. The cost of the equipment necessary for these tasks could exceed a hundred thousand US dollars. In this paper we describe a low cost and small form-factor development system comprising of commodity components and a simple custom printed board circuit, allowing for fast prototyping of medium-scale memristive circuits.

## II. System Overview

The development system consists of three hardware layers, and two software layers, as shown in Figure 1a. The core components of the custom printed circuit board (Fig. 1c) are five transistors which determine access to the memristors, digital in/out signals to gate them, and an op-amp. The circuit also includes two multiplexors in order to allow for addressing of various individual memristor elements with ease. The custom circuit is interfaced via the TS-7800 [5] and TS-ADC16 [6] boards (Fig. 1b) which includes a digital-to-analog converter, and an analog-to-digital converter. These components are used together to apply read and write pulses to the memristor circuit, as shown in the circuit design in Figure 1c. Four of the transistors are used to create an H-bridge architecture, allowing for the application of pulses in both directions using a single positive voltage source. If a negative power supply is available, the H-bridge is not required. All of the three hardware layers are abstracted by the Linux operating system which runs on the TS-7800 board. Software running on the Linux operating system is accessed over the network by a graphical user interface (GUI) designed to run on Windows machines.

This work is supported via NSF grant CCF-1017579.

J. Faghih-Nassiri, N. Maddy, L. Buckland, and D. Strukov are with University of California at Santa Barbara, Santa Barbara, CA, 93106 USA (phone: 619-549-9991; fax: 805-893-3262; e-mail: jfaghihnassiri@gmail.com).

J. Faghih-Nassiri and N. Maddy contributed equally to this work.

## III. Hardware Layers

In general, the two boards used allow a user to simultaneously source four and meter two (which can be extended to 16 channels via multiplexing) different analog voltages. Additionally, the embedded microprocessor has 30 digital input/outputs which can be used for gating transistors and addressing muxes. In our case, we have used only three digital-to-analog converter (DAC) outputs, one analog-to-digital converter (ADC) input, and 4/5 digital inputs for mux selection/gating to perform the specified operation. Various pieces of information about the two boards can be found in Tables I and II.

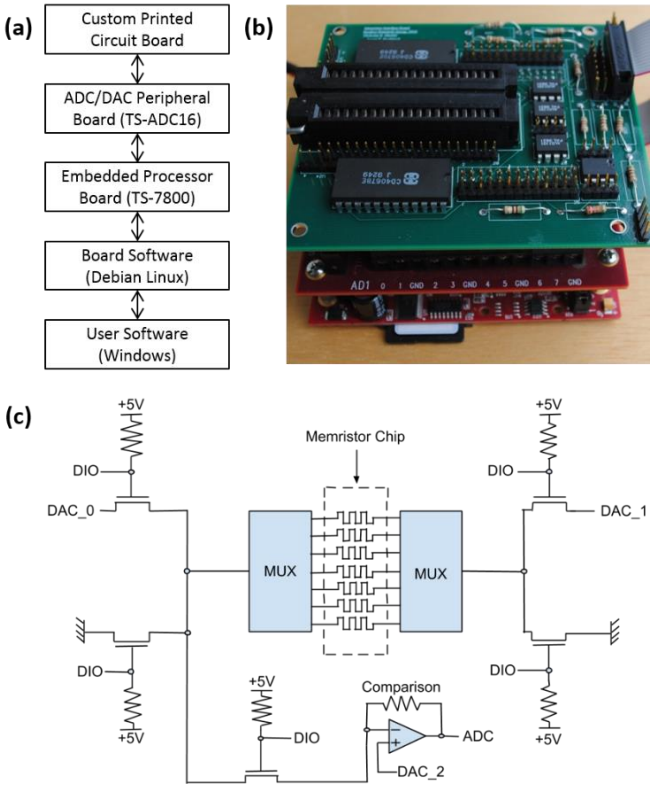


Fig. 1. Development setup: (a) Hardware and software stack, (b) photo of hardware setup, and (c) custom circuit diagram.

The op-amp shown in Figure 1c is used to create a voltage divider where the middle voltage is set by a DAC, and the other terminal of the memristor is connected to ground. The voltage across the negative feedback (comparison) resistor is measured by the ADC. In this configuration, the memristor is part of a voltage divider, and the resistance of the memristor can be determined through a simple calculation:

$$R_m = (V_{dac} \times R_{comp}) / (V_{adc} - V_{read}), \qquad \text{(Eq. 1)}$$

where $R_m$ is the memristors resistance, $V_{dac}$ is the voltage set by the DAC, $R_{comp}$ is the comparison resistance, and $V_{adc}$ is the voltage read by the ADC. In our case, a $V_{read}$ = 0.2 V read voltage was selected because it is high enough to provide a high signal to noise ratio during the ADC read, but still low enough to not modify the state of the memristors used [7]. A comparison resistor $R_{comp}$ = 22KΩ was selected to provide maximum ADC resolution when reading memristors with a resistance between 2KΩ and 50KΩ. Given memristors of different characteristics, the read voltage and comparison resistor can be chosen accordingly in order to preserve circuit functionality.

The printed circuit board (Fig. 2a) which implements the circuit shown in Figure 1c was designed such that it can fit directly on top of the TS-ADC16 (Fig. 2b), and plug into the ADC/DAC directly through header pins. Power is provided to the PCB through two wires running from the TS-ADC16, and the digital input/output signals are provided by the TS-7800 (Fig. 2c). The three boards sit stacked on top of each other and create an easy to transport bundle, which measures 10cm wide, 12cm long, and 6cm tall.

TABLE I. TS-7800 SPECIFICATIONS

| Embedded Processor Board | TS-7800 |
|---|---|
| Processor | 500MHz ARM9 |
| Bus | Internal PCI bus PC/104 connector |
| RAM | 128MB DDR |
| NAND Flash | 512MB (17MB/s) |
| FPGA | 12,000 LUT programmable |
| DIO | 30 |
| Power | 4W @ 5V |

TABLE II. TS-ADC16 SPECIFICATIONS

| ADC/DAC Peripheral Board | TS-ADC16 |
|---|---|
| ADC Chips | 2 x LTC1859 |
| ADC Channels | 16 x 16bit |
| ADC Sample Rate | 2 x 100ksps |
| ADC FIFO | 512 x 16bit |
| DAC Chips | 1x AD5327 |
| DAC Channels | 4 x 12bit |

## IV. SOFTWARE LAYERS

The software package used to interface with the proposed circuit consists of two layers separated by network connectivity. On the Linux end, running on the TS-7800 board, the software consists of a C program, a standalone executable, and a library that is provided with the TS-ADC16 board. On the Windows end, running on a machine on the same network as the TS-7800, the software consists of a GUI. The C program and GUI contain networking code which allows them to communicate.

The primary functions within the C program are called ReadMem and WriteMem. The core of the ReadMem and WriteMem algorithms is shown in Figure 3 in pseudo code. Below these functions are a layer of helper functions in charge of accessing the DIO, DAC, and ADC.

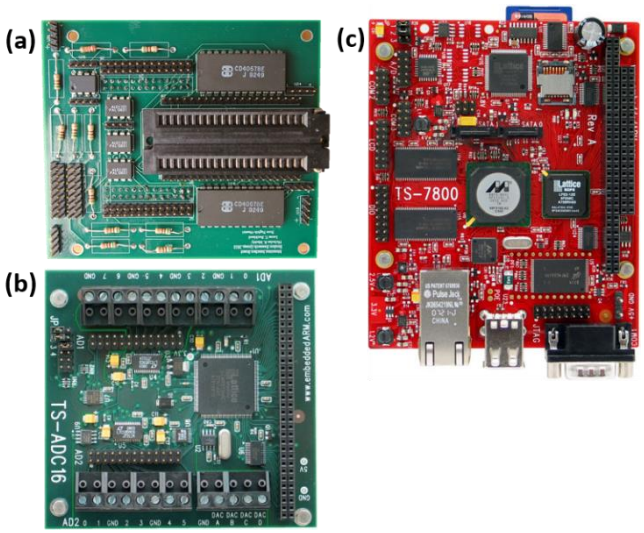


Fig. 2. Photographs of boards used: (a) Custom designed printed circuit board, (b) TS-ADC16 board, and (c) TS-7800 board.

```
/*  pseudo code for read operation */
writeDacReg(2,read_v) //Set DAC_2 (non-inverting of op-amp) to the read voltage
writeDioReg( mux_adr ) // Address the mux
writeDioReg(mux_adr | READ | GND_A) // Also turn on the two needed transistors
readv_out = sampleADC() // Take an ADC reading
writeDioReg(mux_adr | GND_A | GND_B) // Ground the memristor

/*  pseudo code for write operation */
if ( increase resistance )
  writeDacReg(1, write_v) // Set DAC_1 to this iteration's voltage
  writeDioReg( mux_adr | GND_B ) // Turn on the left ground
  writeDioReg( mux_adr | GND_B | POW_B ) // Also turn on the right DAC
if ( decrease resistance )
  writeDacReg(2, write_v) // Set DAC_2 to this iteration's voltage
  writeDioReg( mux_adr | GND_A ) // Turn on the right ground
  writeDioReg( mux_adr | GND_A | POW_A ) // Also turn on the left DAC
uwait( 2 ) // Wait 2 microseconds
writeDioReg( mux_adr | GND_A | GND_B ) // Ground the memristor
```

Fig. 3. Examples of pseudo code.

Through the network, Linux C functions can be called by the Windows GUI (Fig. 4). This process is preceded by opening a socket using the board IP address and a port number which matches the one contained in the C code. Once a connection is established, a user may specify specific multiplexer outputs which they wish to operate on using the "Mux Address" text field. Once that value is set, a user can choose between four different functions. The first, commit sweep, applies a series of staircase pulses of a default width, and allows the user to specify the start, end, and increment voltages. The second, commit single, applies a single pulse of a desired voltage and a desired width. The third function

calls WriteMem, and sets the current state of the memristor. The final function calls ReadMem, and returns the current state of the memristor in terms of resistance.

## V. Experimental Results

In principle, the developed circuit allows users to interface with a variety of memristive circuits. However, at the initial stage we limit testing to single $Pt/TiO_{2-x}/Pt$ memristive devices on a 40-pin wire bonded memristor package fabricated at UC Santa Barbara. In this case, our initial goal is to tune the state of memristive devices using an algorithm similar to the one described in reference 7.

In particular, the algorithm involves the application of many write pulses, with read pulses between them, which is implemented with a single WriteMem function call. The voltage of each successive write pulse will continue to grow until the read resistance is within a desired percentage of the target, as shown in Figure 3. In the case that a write pulse results in an overshoot of the resistance beyond the goal by more than a desired percentage, the algorithm begins applying write pulses in the opposite direction in finer increments in order to achieve a more accurate write.

In Figure 5, the data shows that the memristor state has changed after the first series of pulses, but that the standard maximum voltage threshold is not high enough to change it to the desired resistance. In response, the algorithm implements a series of pulses in the same direction, and increases the maximum voltage threshold. This is done in a separate sweep rather than as a continuation of the previous sweep in order to prevent overdriving the device in the other direction, and in order to prevent burning the device. The algorithm next switches direction three times in order to fine tune the state which is being set. Finally, the memristor is read multiple times in order to verify that the resistive state is nonvolatile and no drift occurs as a result of other secondary effects (i.e. electronic trapping/detrapping [1]).

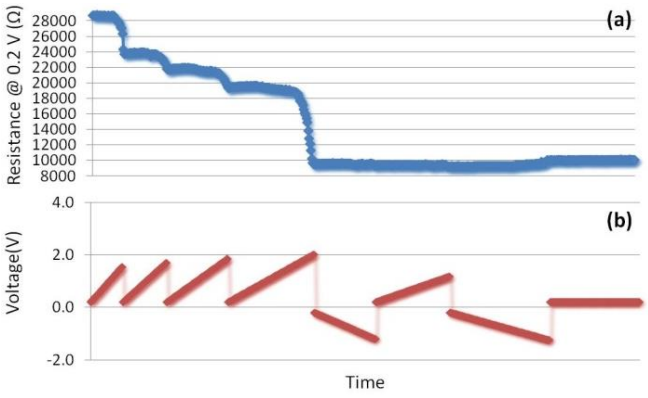


Fig. 5. Exceptional experimental results: (a) Measured resistance, and (b) voltage of write pulses applied.

Because the read pulses do not modify the state of the memristor, they are wider (approximately 75 milliseconds) to allow the ADC time to take multiple samples, and average them. The write pulse width can be controlled manually by modifying the duration of the wait code. The width of the write pulse is limited by the digital IO on the TS-7800 to approximately 1 microsecond, though the pulse width used is set to approximately 2 microseconds to account for slew rate.

Due to the additional resistance of the multiplexers, as well as other systematic error, the output of ReadMem is corrected to the following: $R_{corr} = (R_{orig}*1.06) - 800$, where $R_{corr}$ is the corrected memristor resistance, $R_{orig}$ is the original resistance, and 800 is resistance of the multiplexers in ohms. After applying the corrective formula, the error in resulting reads is reduced to below +/- 2.5 percent of the actual resistance.

## VI. Discussion and Summary

In comparison to the hardware setups often used in memristor and nanodevice research labs, the setup described in this paper is much smaller, lighter, and cheaper. While many pieces of hardware designed for accessing transistor-like elements can weigh over 50 kg and requires many slots in a server rack, the development system described here weighs under 2 kg and can be placed on a desk. While the software interface for the existing hardware can be very complex and sometimes inefficient, the user interface in the system described allows for both abstracted functions such as ReadMem and WriteMem, and general-purpose functions such as Commit Single and Commit Sweep.

In the future, improvements to the circuit might include a virtual crossbar method of addressing, which would allow for a greater number of input/outputs to be accessed by a single piece of equipment. Even though it was not tested, the hardware should also work with a memristor crossbar circuit. A simple expansion of the software would allow the automated tuning of memristors to a high precision. According to previous work and preliminary results with the current software and circuit design, a single memristor with a working range between 2 and 50 KΩ would be able to hold 54 unique states, or about 6 bits. Further optimizations could increase this capacity through reducing the read error, and applying a very small acceptable range for the writes of resistances.

## Acknowledgment

The authors thank the entire Strukov Research Group at UCSB for their technical support and the fabrication and characterization of the memristor devices used.

## References

[1] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnology*, vol. 8, pp. 13-24, Jan 2013.
[2] R. Waser, Nanoelectronics & Information Technology, 3rd Ed, Wiley, 2012.
[3] Y. V. Pershin and M. Di Ventra, "Memory effects in complex materials and nanoscale systems," *Advances in Physics*, vol. 60, pp. 145 – 227, 2011.
[4] S. D. Ha and S. Ramanathan, "Adaptive oxide electronics: A review", *J. Appl. Phys.*, vol. 110, art. 071101, 2011.
[5] TS-7800 Manual http://wiki.embeddedarm.com/wiki/TS-7800

[6] TS-adc16 Manual
http://www.embeddedarm.com/about/resource.php?item=344

[7] F. Alibart, L. Gao, B. Hoskins, and D. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm", *Nanotechnology*, vol. 23, art. 075201, 2012.

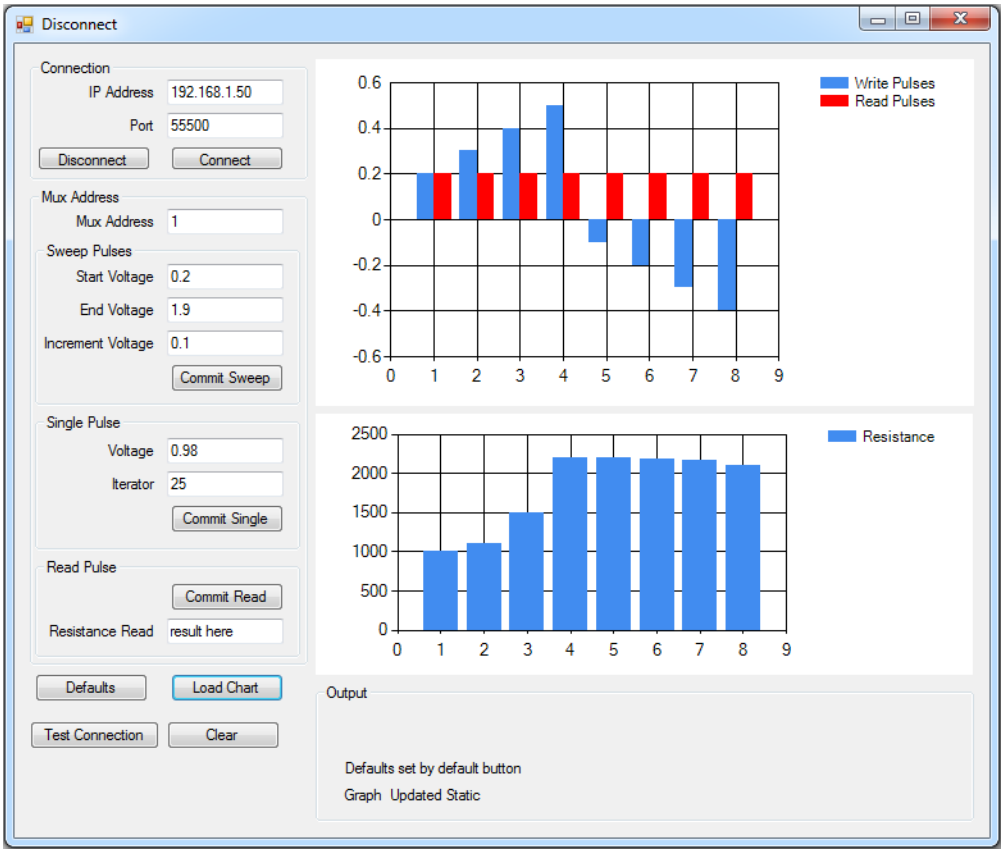


Fig. 4. The graphic user interface, which runs on Windows machines and communicates with the TS-7800 through the network.