Microelectronics Journal ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

A cellular computing architecture for parallel memristive stateful logic

Eero Lehtonen [a,*], Jari Tissari [a], Jussi Poikonen [b], Mika Laiho [a], Lauri Koskinen [a]

[a] Technology Research Center, University of Turku, Joukahaisenkatu 1C, 20520 Turku, Finland
[b] Department of Communications and Networking, Aalto University, Espoo, Finland

ARTICLE INFO

Article history:
Received 9 March 2014
Received in revised form
3 September 2014
Accepted 12 September 2014

Keywords:
Memristor
Memristive crossbar
CMOL
Stateful logic
Implication logic

ABSTRACT

We present a cellular memristive stateful logic computing architecture and demonstrate its operation with computational examples such as vectorized XOR, circular shift, and content-addressable memory. The considered architecture can perform parallel elementary memristor programming and stateful logic operations, namely implication and converse nonimplication. The topology of the crossbar structure used for computing can be dynamically reconfigured, enabling combinations of local and global operations with varying granularity. In the CMOS cells used for controlling the memristors, we apply a new type of capacitive keeper circuit, which allows for energy efficient implementation of logic operations. The correct operation of this architecture is verified by detailed HSPICE simulations for a structure containing eight memristive crossbars. This work presents a hardware platform which enables future work on parallel stateful computing.

1. Introduction

Memristive implication logic was originally proposed by Kuekes in [1] as a way to perform logic on memristors. In this form of logic, Boolean variables are represented by the low and high resistance states R_{ON} and R_{OFF} of binary memristors. This operation was first demonstrated empirically in [2]; since then, various memristive stateful logic operations and corresponding synthesis of Boolean functions have been considered for example in [3–9].

Memristive stateful logic is inherently sequential, and as noted already in [2], it is most efficiently used in parallel form in memristive crossbar architectures. However, as demonstrated for example in [6], a monolithic memristive crossbar circuit allows only limited parallelism. Solutions to partitioning crossbar circuits to allow increased parallelism have been proposed for example in [6,9,8,10]. Specifically Kim et al. [8] presents how memristive stateful logic can be performed in a CMOL-type [11] FPNI architecture [12], which allows pipelining stateful operations. A similar approach is discussed in [9], where for example a stateful eight-bit adder benefiting from crossbar partitioning is demonstrated. However, in the previously presented parallel stateful computing architectures the fan-out and fan-in of elementary stateful logic operations is limited. In this work we show how the keeper circuits presented in [13,6] can be used to facilitate large fan-in and fan-out in parallel stateful logic operations.

In the following we present a circuit architecture designed for efficient parallel stateful logic computing. This architecture consists of an array of small memristive crossbar circuits which can connected to form larger crossbars. We show how such an architecture allows us to perform in parallel complex vector operations using a relatively small number of sequential stateful logic operations. We present a CMOS cell design with the objective of minimizing the number of transistors per memristor required, and show that this design allows the implementation of unconditional write operations and two stateful logic operations, enabling the computation of arbitrary Boolean logic functions. Correct operation of the circuit architecture is verified by detailed HSPICE circuit simulations using 0.13 μm CMOS technology, and a memristor model whose characteristics are selected based on empirical results presented in [14].

To demonstrate parallel stateful computing algorithms in large-scale simulations, we developed a Matlab script language and its compiler which generates from a list of commands the control signals required in HSPICE simulations of the considered circuit. These commands are used also in the following text to define example computations. The main objective of this work is to propose and simulate a hardware platform which enables future work on parallel stateful computing.

The paper is organized as follows. In Section 2 we present the memristor model and define the stateful logic operations used in this work. In Section 3 we describe the cellular stateful logic architecture and define its control signals. Implementation of elementary write and logic operations and related compiler commands are described in Section 4. Examples of parallel vector

* Corresponding author. Tel.: +358 23336963.
E-mail address: eero.lennart.lehtonen@utu.fi (E. Lehtonen).

http://dx.doi.org/10.1016/j.mejo.2014.09.005
0026-2692/

operations and computations implemented using the considered architecture are presented in Section 5. Section 6 concludes.

2. Background

2.1. Memristors

In this paper we consider the use of rectifying linear bistable memristors in massively parallel logic computing. By rectifying memristors we mean devices which pass significant current only to the forward direction, while the switching behavior corresponds to that of a nonrectifying bipolar memristor, that is, positive voltages program the device into a more conductive state, whereas negative voltages program the device towards a more resistive state. In the following we define the mathematical model of the memristors assumed in this work. This model is inspired by the device demonstrated empirically in [14].

A memristor has a state variable $w \in [0,1]$ which corresponds to the value of its memristance R_m in the forward direction as follows. When $w=0$, the memristor is said to be in the OFF-state corresponding to $R_m = R_{\mathrm{OFF}}$. When $w=1$, the memristor is said to be in the ON-state, and $R_m = R_{\mathrm{ON}}$. Formally,

$$R_m = \begin{cases} R_{\mathrm{OFF}}(R_{\mathrm{ON}}/R_{\mathrm{OFF}})^w, & v \geq 0 \\ R_{\mathrm{OFF}}, & v < 0, \end{cases} \quad (1)$$

where v is the voltage across the memristor in the forward direction. In accordance with [14], we assume that $R_{\mathrm{OFF}} = 500\ \mathrm{M\Omega}$ and $R_{\mathrm{ON}} = 500\ \mathrm{k\Omega}$.

The memristor is programmed towards the ON-state by applying across it a voltage larger than a *threshold voltage* V_{TH}. Correspondingly applying a voltage more negative than $-V_{\mathrm{TH}}$ programs the memristor towards the OFF-state. When the voltage across the memristor is between $-V_{\mathrm{TH}}$ and V_{TH}, the state of the device is assumed to remain unchanged. Note that for simplicity, we assume symmetric threshold voltages for programming to OFF- and ON-states. In practice, these voltages may differ, as for example in the device of [14]. Such asymmetry must be taken into account when defining control voltages for programming and logic operations. The following dynamics are assumed for this bipolar memristor:

$$\frac{\mathrm{d}w}{\mathrm{d}t} = \begin{cases} \alpha(v - V_{\mathrm{TH}}), & v \geq V_{\mathrm{TH}} \\ \alpha(v + V_{\mathrm{TH}}), & v \leq -V_{\mathrm{TH}} \\ 0 & \text{otherwise}, \end{cases} \quad (2)$$

where α is a positive constant related to the programming rate, and $V_{\mathrm{TH}} = 1\ \mathrm{V}$. In this work we assume $\alpha = 125 \times 10^7\ (\mathrm{Vs})^{-1}$; with this value of α, a memristor initially in state $w=0$ is programmed to state $w=1$ in 4 ns by applying $+1.2\ \mathrm{V}$ across it. Comparable programming rates have been predicted for memristive devices in [15], and reported empirically for example in [16,17]. It should be noted that this memristor model is a very simplified one, for example its programming rate depends piecewise linearly on the applied voltage, and its threshold voltages are fixed, in contrast to what is observed in many physical devices [18]. The main motivation of this work is to investigate a computing architecture with a multitude of memristors, and the considered simple model allows for efficient simulation of the presented circuitry. However, due to this simplification, simulated values of operation durations and energies should be considered only as suggestive, while physical realizations of the considered circuits may have considerably different characteristics.

In the simulations presented in this work we use a HSPICE model of the above described memristor. A pinched hysteresis curve of this memristor is depicted in Fig. 1. The SPICE netlist for this simulation model is

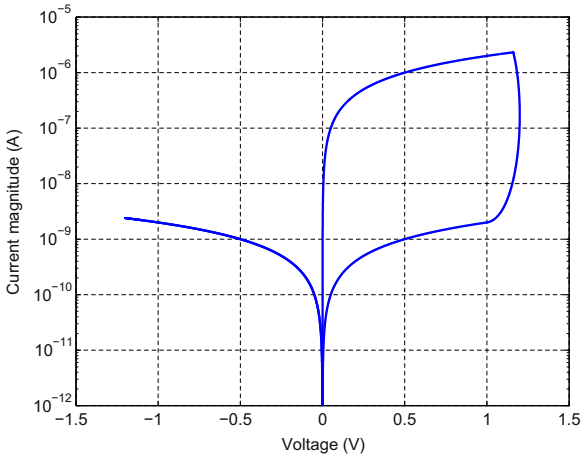

Fig. 1. I–V curve of the rectifying memristor model used in this paper. The model was driven with a sinusoidal input voltage with frequency 25 MHz and amplitude 1.2 V.

```
.SUBCKT memristor P M w
+Ron=500k Roff=500Meg vth=1 vprog=1.2
dwdtprog=250e6 winit=0 wmin=0 wmax=1
*State variable
Gsv 0 w value='sgn(V(P,M))*dwdtprog*absgeq(V(P,
  M),vth)
   *(abs(V(P,M))-vth)/(vprog-vth)*trunc(V(w),V
  (P,M))'
Csv w 0 1
.IC V(w)=winit
*I-V relation
Gmem P M value='V(P,M)/calcrecresistance(V(P,M),
  V(w))'
*Auxiliary functions
.PARAM sign2(var)='(sgn(var)+1)/2'
.PARAM trunc(var1,var2)='(sign2(var1-wmin)+
  sign2(var2))*
   (sign2(wmax-var1)+sign2(-var2))/2'
.PARAM absgeq(var1,var2)='sign2(var1-var2)+
  sign2(-var2-var1)'
.PARAM calcresistance(w)='Roff*((Ron/Roff)**w)'
.PARAM calcrecresistance(var1,var2)='sign2
  (var1)*
   calcresistance(var2)+sign2(-var1)*Roff'
.ENDS memristor
```

2.2. Stateful logic

Stateful logic is a form of computational logic in which devices both store and perform operations on logical values. When implemented with memristors, Boolean variables are represented by their memristances. Stateful logic operations are realized by programming the states of a set of *output memristors* conditionally to the states of a set of *input memristors*. An example of a circuit performing memristive stateful logic is depicted in Fig. 2. Details of memristive stateful logic have been discussed extensively for example in [5,6]. In Section 3 we present a CMOS–memristor circuit which implements two stateful logic operations: *material implication* and *converse nonimplication*. These operations can be used to sequentially compute the value of any Boolean function,

Please cite this article as: E. Lehtonen, et al., A cellular computing architecture for parallel memristive stateful logic, Microelectron. J (2014), http://dx.doi.org/10.1016/j.mejo.2014.09.005