**Modeling Discrete-Event System Using Cellular Automata (FALL 2016)**

**ASSIGNMENT 2**

MEMRISTOR BASED PARALLEL SORTING APPROACH USING ONE-DIMENSIONAL CELLULAR AUTOMATA

**NAGA PRASANTHI BOBBILI**

**STUDENT ID 101057215**

[**nagabobbili@cmail.carleton.ca**](mailto:nagabobbili@cmail.carleton.ca)

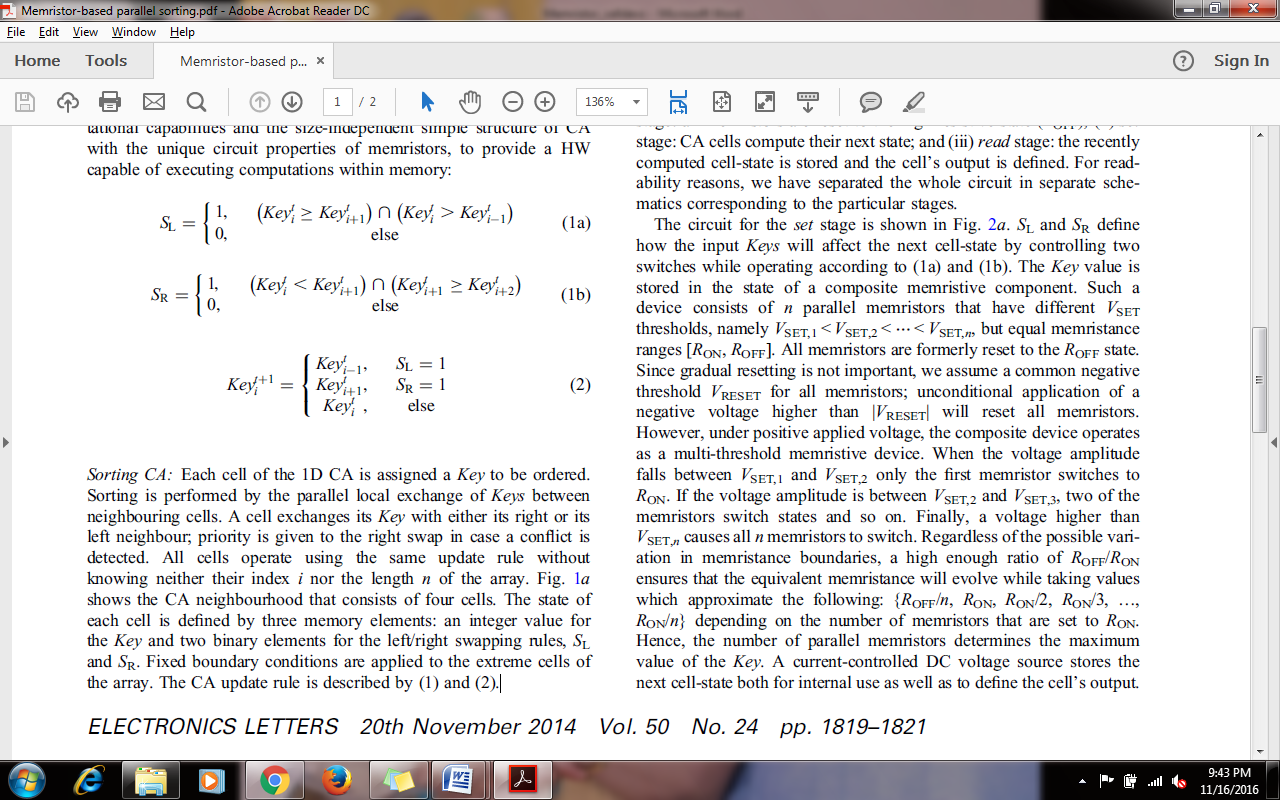
**Carleton University**

**Part 1:**

Memristor is a circuit-level cellular automata (CA) inspired approach to the solution of the classic sorting problem of n Keys in a linear array. This approach uses the structural simplicity of CA combined with the threshold-type switching behavior of memristors and composite memristive components. This idea is to implement the Memristor CA in Eclipse in CD++ Builder.

**Part 2:**

To implement the above idea in part 1, each cell of the 1D CA is assigned a Key to be ordered. Sorting is performed by the parallel local exchange of Keys between neighbouring cells. A cell exchanges its Key with either its right or its left neighbour; priority is given to the right swap in case a conflict is detected. All cells operate using the same update rule without knowing neither their index i nor the length n of the array. Fig. 1a shows the CA neighbourhood that consists of four cells. The state of each cell is defined by three memory elements: an integer value for the Key and two binary elements for the left/right swapping rules, SL and SR. Fixed boundary conditions are applied to the extreme cells of the array. The CA update rule is below



Each computation step comprises two stages: first, the swapping rules are computed; then Key exchanges are locally performed. At a certain moment, the array is sorted and there are no more valid exchanges to take place; the swapping rules remain the same.

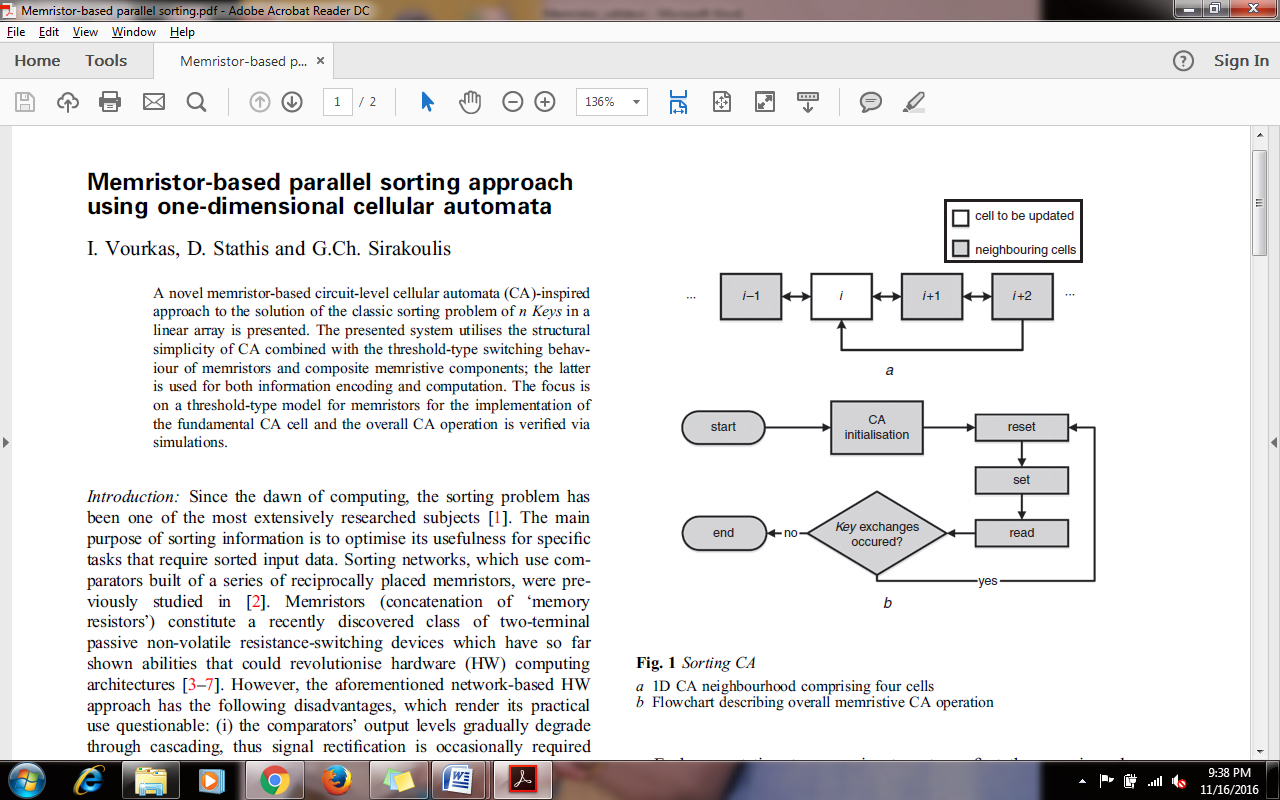


Fig: Sorting CA

The sorting is done in decreasing order.

**Part 3:**

I have implemented this cellular automaton using certain rules mentioned in the research paper. Initial conditions are given to start the process. As per the rule, if left value is greater than the right value, there is no change in the SL/SR values. But if the left value is smaller than the right value, then the

The sorting algorithm in excel sheet is as shown below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| t | SL1 | Key1 | SR1 | SL2 | Key2 | SR2 | SL3 | KEY3 | SR3 | SL4 | KEY4 | SR4 |
| 0 | 0 | 4 | 0 | 0 | 2 | 1 | 1 | 5 | 0 | 0 | 6 | 0 |
| 1 | 0 | 4 | 0 | 0 | 5 | 0 | 0 | 2 | 1 | 1 | 6 | 0 |
| 2 | 0 | 4 | 0 | 0 | 5 | 1 | 1 | 6 | 0 | 0 | 2 | 0 |
| 3 | 0 | 4 | 1 | 1 | 6 | 0 | 0 | 5 | 1 | 1 | 2 | 0 |
| 4 | 0 | 6 | 0 | 0 | 4 | 1 | 1 | 5 | 0 | 0 | 2 | 0 |
| 5 | 0 | 6 | 0 | 0 | 5 | 0 | 0 | 4 | 0 | 0 | 2 | 0 |

When the shift left and shift right keys are ‘1’s then the sorting happens.

The neighborhood and rules are defined below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (0,-3) | (0,-2) | (0,-1) | (0,0) | (0,1) | (0,2) | (0,3) | (0,4) | (0,5) | (0,6) |

Neighborhood definition:

M = <Xlist, Ylist, I, X, Y, ῃ, N, {r,c}, C, B, Z, select}

Xlist = {Ø}

Ylist = {Ø}

I = <PX, PY> Px = { Ø} Py={ Ø}

X = Ø

Y = Ø

ῃ = 10

N = {(0,-3), (0,-2), (0,-1), (0,0), (0,1), (0,2), (0,3), (0,4), (0,5),(0,6)}

R = 1

C = 24

S = {0, 1, 2, 3 , 4, 5 ,6}

B = Ф (wrapped)

Pij Y1 → Pi,j-1 X1 Pi,j+1 Y1 → Pij X1

Pij Y2 → Pi+1,j X2 Pi-1,j Y2 → Pij X2

Pij Y3 → Pi,j+1 X3 Pi,j-1 Y3 → Pij X3

Pij Y4 → Pi-1,j X4 Pi+1,j Y4 → Pij X4

Pij Y5 → Pij X5 Pij Y5 → Pij X5

SELECT = {(-3,0), (-2,0),(-1, 0), (0, 0), (0, 1), (0,2), (0,3),(0,4),(0,5),(0,6)}

**Cell-DEVS Coupled Model Specification**

[top]

components : memresis

[memresis]

type : cell

width : 24

height : 1

delay : transport

defaultDelayTime : 10

border : wrapped

neighbors : memresis(0,-3) memresis(0,-2) memresis(0,-1) memresis(0,0) memresis(0,1) memresis(0,2) memresis(0,3) memresis(0,4) memresis(0,5)

initialvalue : 0

initialrow : 0 ? 2 0 0 3 1 1 9 0 0 8 0 0 7 0 0 6 0 0 5 0 0 4 ?

localtransition : memresis-rule

[memresis-rule]

rule : ? 10 {(0,0) = ? and (0,-1) = ? }

rule : ? 10 {(0,0) = ? and (0,1) = 0 }

rule : 1 10 {(0,0) = 0 and (0,1) = 0 and (0,-1) < (0,2) and (0,-1) != ? and (0,2) != ? }

rule : 1 10 {(0,0) = 0 and (0,-1) = 0 and (0,1) > (0,-2)and (0,1) != ? and (0,-1) != ? }

rule : 0 10 {(0,0) = 1 and (0,1) = 1 and (0,2) > (0,-1) and (0,2) != ? and (0,-1) != ? }

rule : 0 10 {(0,0) = 1 and (0,-1) = 1 and (0,-2) < (0,1) and (0,-2) != ? and (0,1) != ? }

rule : {(0,3)} 10 {(0,0) != ? and (0,0) != 0 and (0,0) != 1 and (0,1) = 1 and (0,2) = 1 and (0,0) < (0,3) and (0,3) != ?}

rule : {(0,-3)} 10 {(0,0) != ? and (0,0) != 0 and (0,0) != 1 and (0,-1) = 1 and (0,-2) = 1 and (0,0) > (0,-3) and (0,-3) != ?}

rule : {(0,0)} 10 {t}

**Implementation and Testing**

The model definitions given above were implemented in the following test case

? 2 0 0 3 1 1 9 0 0 8 0 0 7 0 0 6 0 0 5 0 0 4 ?

Simulation result :



**Conclusion**:

The model is working as expected. The memristor CA is very useful in electronics in sorting methods. Hence I hereby confirm that this model is working as expected. The video file is attached to the folder.