Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1

Article

GERARD: GEneral RApid Resolution of Digital mazes using a memristor emulator

Pablo Dopazo [1], Carola de Benito [1,3], Oscar Camps [1], Stavros G. Stavrinides [2] and Rodrigo Picos [1,3]*

[1] Industrial Engineering and Construction Department, Balearic Islands University, Palma Mallorca, Spain; pablo.dopcor@gmail.com, carol.debenito@uib.es, oscar.camps@uib.es, rodrigo.picos@uib.es
[2] School of Science and Technology, International Hellenic University, Thessaloniki, Greece; s.stavrinides@ihu.edu.gr
[3] Health Institute of the Balearic Islands (IDISBA), Palma Mallorca, Spain
* Correspondence: rodrigo.picos@uib.es

Abstract: In this paper, a system of searching for optimal paths is developed and concreted on a FPGA. It is based on a memristive emulator, used as a delay element, by configuring the test graph as a memristor network. A parallel algorithm is applied to reduce computing time and increase efficiency. The operation of the algorithm in Matlab is checked beforehand and then exported to two different Intel FPGAs: a DE0-Nano board and an Arria 10 GX 220 FPGA. In both cases reliable results are obtained quickly and conveniently, even for the case of a 300x300 nodes maze.

Keywords: Memristor; memristive grid; maze solving; shortest path; programmable devices.

1. Introduction

Since ancient times humankind has tried to solve labyrinths or mazes. The paradigm of maze solving is found in the Greek myth of Ariadne who used a thread to help Theseus getting out of Minotaur's labyrinth. Today, maze resolution can have multiple applications, as in robotics, topology and many areas of science and technology [1–3]. Graph theory is used as an element to define the maze problem, where optimized path solving algorithms could then be applied. Some algorithms simply obtain an exit path, while others optimize it by finding the shortest one. One of the latter is the Dijkstra algorithm [4] that calculates all possible paths to reach a final node beginning from an initial one, and then compares the total cost of all of them, eventually keeping with the shortest. This algorithm, as well as all of its alternatives, quantum computing excluded [5], requires a long computation time when dealing with complex graphs. To overcome this, parallel computing becomes a very good alternative in reducing computing time and further improve efficiency [6–8].

One of the trends in high performance computing is the use of arrays of memristors performing parallel computing. Memristors are resistive devices whose resistance depends on their dynamical history [9]. In fact, they can be thought of as variable resistances capable to remember their past; this is, memristors can be used as a memories [10], as well as computation elements. One of the applications they have been used in is modelling the distance between nodes in a maze. In this approach, the shortest path between two points is corresponded to the current path with the minimum resistance [8], converting the problem to one determining the maximum current path.

Implementation and design of circuits with memristors requires extensive simulations when the number of devices involved is large like in memories or bio-inspired circuits[11]. In order to speed up simulations some researchers use digital, analog, or mixed-signal emulators, [12–18] to reproduce the behaviour of memristors, eliminating some undesired effects like the cycle to cycle variability appearing in ReRAMs [19,20].

In this paper, we implement a fully digital system (under the acronym *GERARD*: *GE*neral *RA*pid *R*esolution of *D*igital mazes) that solves mazes in a digital environment by implementing the topology of those mazes as a grid of nodes in a FPGA, which allows parallel computing. In this proposal, the interconnections between the nodes of the grid are implemented using memristor emulators that are purely digital, as in [21]. Determining

check for updates

Citation: Dopazo, Pablo; de Benito, Carola; Camps, Oscar; Stavrinides, Stavros G.; Picos, Rodrigo; GERARD: GEneral RApid Resolution of Digital mazes using a memristor emulator. *Preprints* **2022**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2021 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1

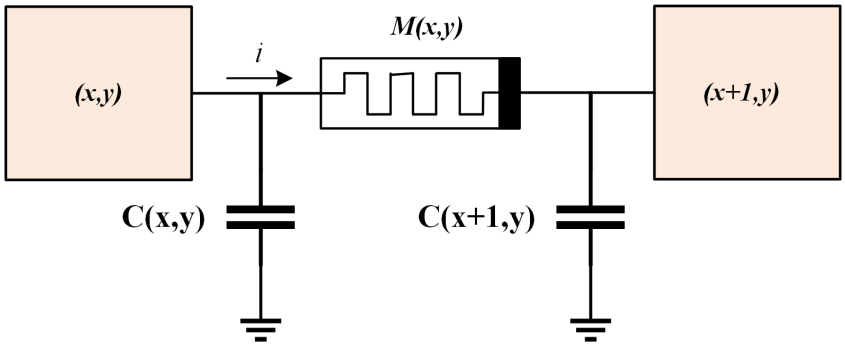

Figure 1. Scheme of the general interconnection pattern between two arbitrary adjacent nodes (x,y) and (x+1) of a NxM grid. Notice that for the sake of clarity only one of the four connections of each node is shown.

the minimum current path is achieved by mapping the distance between nodes to a memristance, which charges a fixed capacitor with a fixed voltage. This way, the time needed for charging the capacitor up to a given voltage provides with an estimation of the value of the memristance, thus the length of the path.

2. General method

The proposed method for the maze solver is based on representing the maze as a matrix of nodes connected through memristors (Fig. 1) to its four nearest neighbours, as in the original work of Pershin et al. [8]. The main idea in that paper was measuring the current through the interconnecting memristors, getting this way the minimum current path. That method was based on the capability of memristors to be programmed to a given resistance value using a (relatively) high voltage, while during its normal operation it could be considered to hold its programmed resistance value.

The above method had the problem of determining which was the minimum current path, which is a rather complex experimental problem. In this paper, we propose another novel approach, based on measuring the time demanded for a grounded capacitor to be charged through a memristor, as in Fig. 1. This time obviously depends on the memristance value (in fact, this is a dynamically changing resistance). Since the input is considered to be a constant current I_C fed through the memristor, its voltage V_C will be:

$$V_C = \frac{I_C}{C} \cdot t \tag{1}$$

where C is the value of the capacitor, and t is the time since the capacitor has started to charge, assuming an initial value for $V_C(t = 0) = 0$. The voltage drop through the memristor is $V_M = I_C M$, with M being the programmed resistance value of the memristor. The time t_C needed for these two voltages to equate is just:

$$t_C = MC \tag{2}$$

Thus, for a constant value of C, measuring t_C allows for directly estimating the value of M. This way the minimum time is used to calculate the minimum resistance current-path, instead of the maximum current. As a side comment, it is worth noticing that a similar effect could be achieved by using a complementary configuration with fixed resistors and memcapacitances.

The flow diagram of the algorithm is shown in Fig. 2. The time $t_{(x,y;x+1,y)}$ needed for a signal to propagate from an initial node (x, y) to a destination node $(x + 1, y)$ (Fig. 1), is calculated by Eq. 2. Once the signal propagates, the destination node is activated and performs a series of actions:

1. It stops listening to any other input, so no other signal can trigger it.
2. It identifies and stores the triggering input port.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1

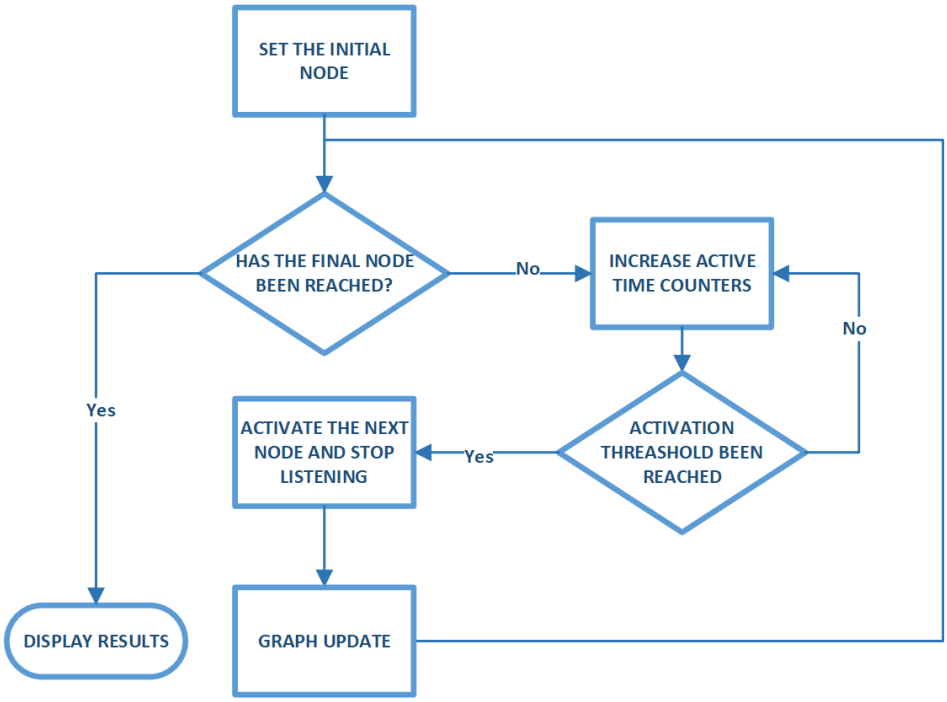

Figure 2. The flow diagram of the proposed shortest path algorithm.

3. It propagates the signal to all its non-activated ports.

Once the final target node is reached, it sends a signal to the controller, which in turn, recovers the path. This is simply achieved by recovering the activated input from each node, and working the way back from the final node to reconstruct the actual path.

Consequently, the time the algorithm needs to reach the solution is determined by the length of the path and the cost between the nodes in the path, as described in Eq. 3:

$$t_{total} = \sum t_{(x,y;x',y')} = C \sum M_{(x,y;x',y')} \tag{3}$$

where the summation is performed over the nodes (x,y) and (x′,y′) in the shortest path and $M_{(x,y;x',y')}$ is the resistance between them.

3. Algorithm Implementation

3.1. Memristor Model Implementation

We have implemented a memristor model based on a purely digital emulator [21]. This emulator implements a simple relation between charge Q and flux ϕ, i.e., $Q = M(\phi)\phi$. Memresistance $M(\phi)$ is also calculated with the simplest relation $M(\phi) = M_0\phi$. Full details of the implementation are provided in [21]. Note that only a minor modification was needed to fulfill the requirements for this application. This was adding a switch keeping constant the value of the memresistance, or allowing it to be programmed, as discussed in the section above. With this modification, once the memristor is programmed, it behaves as an element with a constant resistance.

3.2. Matlab Implementation

The operation of the system developed in Matlab implements the flow diagram appearing in Fig. 2, and performs the following operations:

1. Program all memristors with a memristance value M corresponding to the distance between nodes;
2. Set the starting point, taking into account that the bottom right element is the end by default (without any loss of generality);

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1

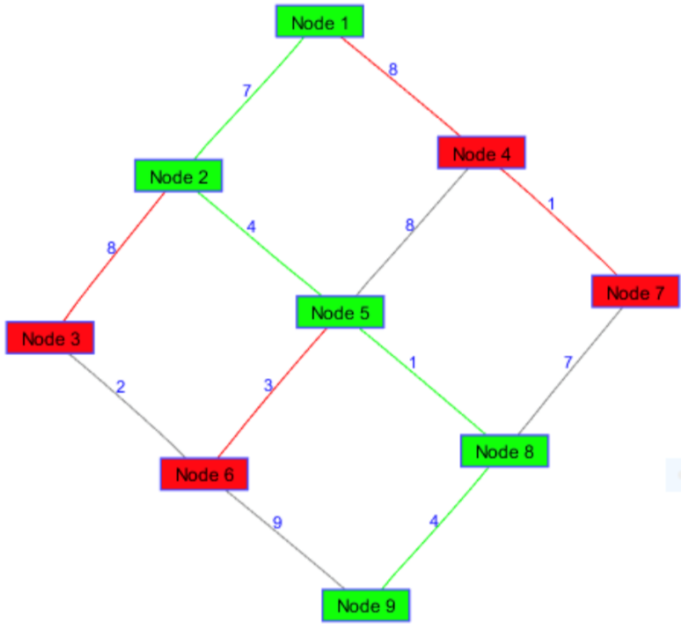

Figure 3. The solution in the case of a 3x3 example. Notice that the numbers between the nodes represent the resistance. The shortest path is the one passing through the green nodes. The initial node is node 1, and the final node is 9.

3. Start counting with the first node and propagate the signal to its neighbours with a delay given by Eq. 2;
4. When a node receives an input signal, it is marked as active and treated as a new starting point;
5. Repeat from step no. 3 until the final node is reached;
6. If the end node is reached, a signal that the process is finished is sent to the control unit, and the shortest path is then retrieved.

It is noted that the Matlab implementation of the designed algorithm has been validated against the Dijkstra algorithm [4] up to an 8x8 matrix, providing exactly the same results. An example is shown in Fig. 3 for a simple 3x3 matrix, where the numbers between the nodes correspond to the distance between them. The green color defines the calculated shortest path (which is the same both by Dijkstra and the proposed algorithm).

3.3. FPGA Implementation

The FPGA implementation of this novel maze solver consisted of three different parts, namely the control system, an interconnection element including the memristor emulator, and the nodes themselves, as illustrated in Fig. 4. Note that the number of cells and memristor-blocks depends on the grid size, since there is an one-to-one correspondence between the physical modules and the maze net.

3.3.1. Communications Block

This block is responsible for the communication between the FPGA and the external systems, in this case a PC. The communications part between the computer (Matlab) and the FPGA was implemented using a JTAG interface, as in [22], where a full description of the procedure is provided. This part consisted of two standard blocks: the vJTAG component and the vJTAG-interface, as shown in Fig. 5. The vJTAG component is responsible for receiving the information and injecting it into the vJTAG-interface. The later saves the data received in a register and then sends its contents to the other components of the system through a dedicated bus. In addition, these components were responsible for sending the result back to the user. The interface between the user and the maze solver in the FPGA was implemented in Matlab. This uses TCP/IP with a dedicated socket [22], and was

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1

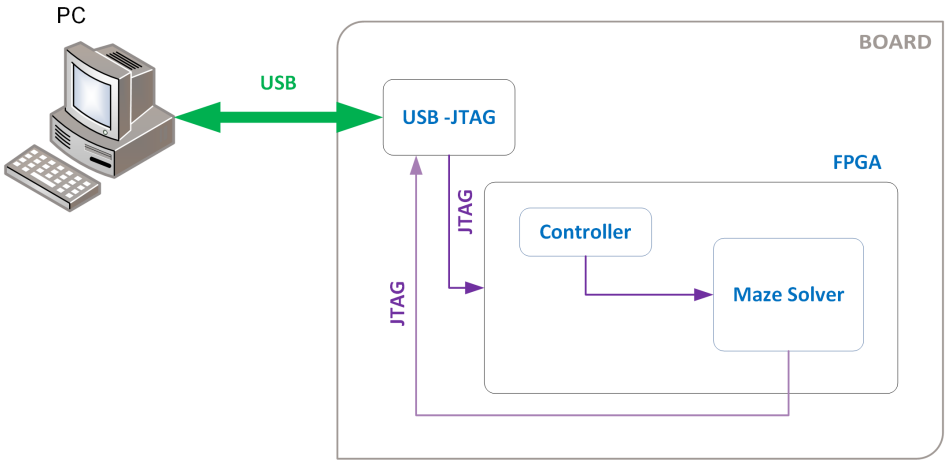

Figure 4. Illustration of the system, including the PC running the external software, the USB-to-JTAG interface on the electronic board, and the FPGA, where the general controller and the maze solver are located.

Table 1: Set of instructions for the control system. Notice that instructions 010 and 011 need additional arguments (target row and column) to function.

Code	Description	Parameters
001	Reset all the internal registers	–
010	Program the value of a memristor	Row, column
011	Set the starting point	Row, column
100	Start the process	–
101	Get the calculated path	–

responsible for converting user-commands to binary code. It was also receiving back data from the FPGA, displaying it accordingly.

3.3.2. Interconnection Block

This block implements a delay equivalent to the cost needed to traverse it. As mentioned above, this delay module (shown in Fig. 6) implements a memristor-capacitor emulator, as in Fig. 1. The system can be programmed to a given delay by setting the memristor to an equivalent value provided by Eq. 1 and the actual cost of the maze.

Once the memristor is programmed, and one of the input ports of the memristor has been activated, the capacitors are charged using a constant voltage input V_s until a threshold value is reached. For a known value of the capacitor and the memristor, this would be equivalent to determining the value of the current used to reach the threshold value.

We have used the memristor emulator implemented in [21] as described above, and considered it to be connected to a capacitor at each end, which was initially connected to ground. The capacitor has been implemented as an accumulator with input i_C and output v_{Co}. Moreover, the equations have been simplified by setting all the constants of the system to 1, without any loss of generality. At each clock cycle, $v_{Co}(t)$ would be updated as:

$$v_{Co}(t+1) = v_{Co}(t) + i_C C \Delta t \tag{4}$$

This block will then propagate the signal to the other end by activating the out terminal, when $t = M$, as determined by Eq. 2. Notice that the block must only accept the first input that reaches it (either in1 or in2), and any subsequent input signal has to be disregarded. In this module, the numbered inputs and the output are connected to the nodes, while

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1

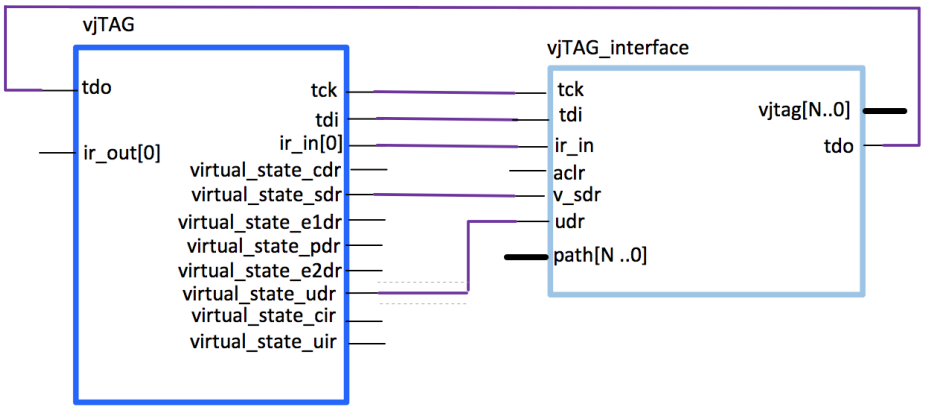

Figure 5. The connection between the vJTAG blocks in the communication module, as discussed in [22] and [23]

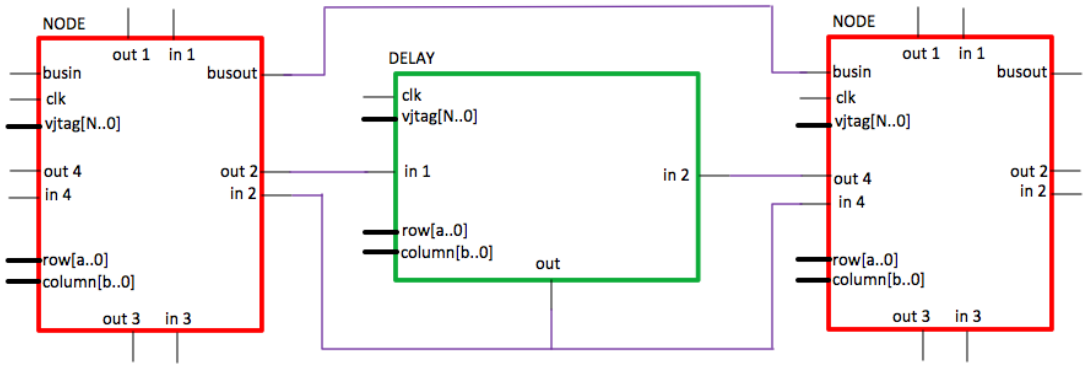

Figure 6. The equivalent FPGA implementation of Fig. 1, with the nodes and their interconnection. The block labelled *DELAY* is the equivalent of the memristor-capacitor elements, while the *NODE* block corresponds to the (x,y) node elements. All these blocks also show the additional inputs that allow external programming and data recovering, as discussed in the text.

all other entries follow the same logic as the Node. This module uses the clk fall edge to create a small delay between the components, allowing the Node to update its signals before the Delay starts to work. In addition, inside each node and memristor there was also a control unit connected to the vjtag[N..0] bus input, implemented as a state machine, with the corresponding part of the set of instructions shown in Table 1 that allows it to be programmed to the initial value. In order to use a single template component, we have also added two inputs setting the block row and column that identifies the block for programming purposes.

3.3.3. Node Element

The Node components represent each of the network-nodes and were connected according to the pattern established previously in Matlab. These interconnections were made using the intermediate components that generate the signal delay, i.e. the memristor and capacitor emulator described above. Notice that each block has four *in* and four *out* terminals, numbered clockwise from the top, that connect to the delay block as shown in Fig. 6.

The numbered inputs and outputs of the Node block are used to form an (NxM) graph, corresponding to the actual maze, and are connected to the delay modules. We repeat that the maze path is defined by the interconnections between these nodes. The vJTAG entry is the input for the data sent by the communications module through a shared, read-only bus, whereas the row and column entries mark the position of the component within the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1

system for programming purposes. The modules accepts the corresponding programming instructions in Table 1. These instructions allow the user to use the communications block to set the starting point in a specific node, defined by its position, and, once it is set, to start the process of finding the shortest path between this node and the (N,M) node. Finally, the *busin* input and *busout* outputs are connected between each pair of nodes to return the path through the JTAG interface.

4. Results

Having in mind the global operation that has been described, this was implemented as follows: when the initial Node gets activated by the user, it activates its four outputs in order to start the counter of all four Delays connected to it; then, when the assigned weight value has been reached, the Delay activates its output, resulting in turning on the Node on the opposite side. An activated Node saves in its internal register the one out of the four entries that launched the activation, stopping at the same time to listen to the other entries, which are now transformed into outputs. Then, the node sends a pulse through these new outputs, that propagates in the same fashion until the end node is reached.By this approach, there are several counters running in parallel, achieving the goal of reducing calculation time, thus, improving efficiency. As has been explained above, each node stores the information of the direction from where the first pulse reached it. This information is sent through a bus connecting each and every node up to the communications module. The later concatenates each of the bits it receives, forming a N-bit vector that is sent to the user via the vJTAG interface (*path[N..0]* input).

Initially, we checked the proof of concept of the proposed algorithm using the implementation of a 4x4 maze-example, as shown in Fig. 7. In this case the system needed 16 Node and 24 Delay modules, to implement the desired grid into the DE0-Nano FPGA board, using a 49-bit vJTAG vector and 3-bit row and columns vectors.

In this example, the grid defining the maze was initially described in Matlab, and then programmed into GERARD through the dedicated interface. As described above, once the solver was programmed, the signal propagated to the end node and the system returned a signal, which in our case was a 49-bit vector containing the direction of the incoming signal for each node according to the Table 2 code. Notice that each node used three bits and these

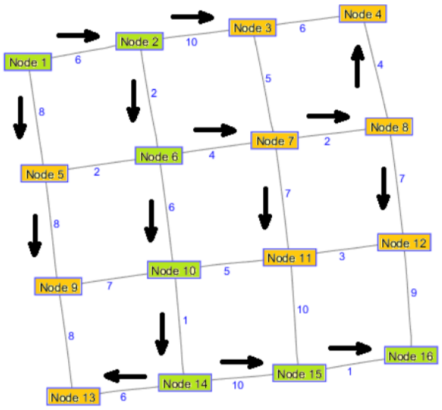

Figure 7. The FPGA returned result in the case of a 4x4 maze. The corresponding recovered shortest path (nodes in green) appears, showing the directions in Fig. 8, according to Table 2 .

"000"	"100"	"100"	"011"
"001"	"001"	"100"	"100"
"001"	"001"	"001"	"001"
"010"	"001"	"100"	"100"

Figure 8. Array obtained from FPGA for the example graph, indicating the first activating input for each node.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1

Table 2: Incoming signal direction codes.

Code	Description	Code	Description
000	Initial node	111	Final node
001	Above	011	Below
010	Right	100	Left

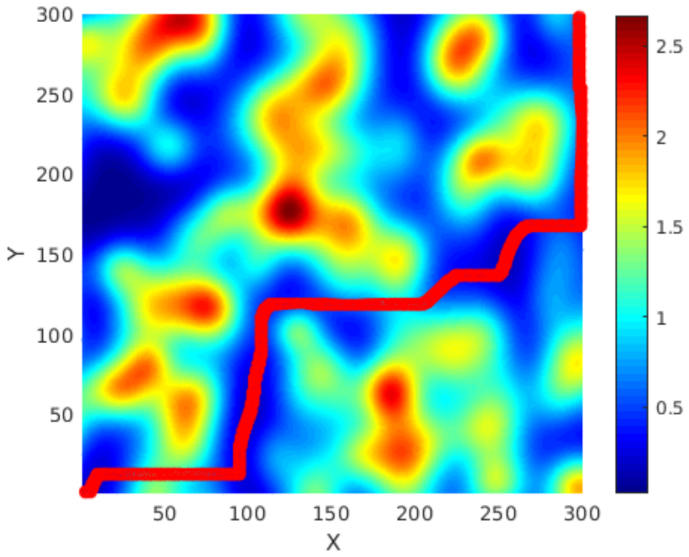

Figure 9. An illustration of the results returned in the case of a 300x300 maze, used for testing the implementation of the solver, appears. The colors represent the cost, which is the resistance between paths, as indicated in the right bar in arbitrary units, while the red line shows the returned shortest path.

were reorganized in a 4x4 array as shown in Fig. 8. Once the result vector was obtained, the user interface decoded it by working its way backwards, from the end node back to the initial node, obtaining the result shown in Fig. 7. In this specific case (a 4x4 matrix), the design required a total of 6142 elements (28 % of the total), with 6033 combinational functions (27 %) and 3274 dedicated logic registers (15 %), using a clock frequency of 50 MHz.

Finally, another example was implemented into an Arria 10 GX 220 FPGA card at 200 MHz using the Matlab FPGA-in-the-loop (FIL) methodology. In this example the FPGA has been used to speed-up the parallel calculations, and a 300x300 maze was generated, using a total of 196840 logic elements (89 %), 68654 ALM (85 %), 301368 registers (93 %), 10442 Kb of M20K memory (88 %) and 1612 Kb (95 %) of the MLAB memory. The resistance between the nodes was generated using 150 2D-Gaussian distributions with random position, dispersion and height, as shown in Fig. 9, where the colors represent the cost. In this same Figure, the red line depicts the shortest path, as returned by the algorithm between the start (left, bottom) and the end points (top, right). The time needed for a full Matlab implementation (with no FPGA) to solve the circuit was around 1900 s, while the FIL version needed only 82ms to solve the maze, for a total cost of 81630 (the total resistance, in arbitrary units) for a path length of 608 cells. Retrieving of the shortest path required thus 1800 bits.

5. Conclusion

In this paper an inherently parallel computation algorithm is demonstrated. It was initially designed in Matlab, then implemented in a FPGA using a memristor emulator and returned reliable results, equivalent to those obtained using Dijkstra's algorithm. It took

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1

profit of the study of multiple paths in parallel, showing how GERARD helps Ariadne to determine the way out of a maze. Two different examples were demonstrated, one with a 4x4 matrix, and another using a 300x300 matrix, both working in a very straightforward way.

The proposed design is simple and easy to scale up for implementing different graph configurations and has been checked with many other examples and using Dijkstra's algorithm [4]. Scalability of the system is limited only by the size of the FPGA. Overcoming this, a proper partitioning scheme could be also utilized. Finally, once actual memristor devices are finally out as a mainstream technology, they could be actually used to implement the proposed maze solver, paving the way for their use in autonomous robotics, among other possible fields.

Author Contributions: Conceptualization, Carola de Benito and Rodrigo Picos; Formal analysis, Carola de Benito, Oscar Camps, Stavros Stavrinides and Rodrigo Picos; Investigation, Pablo Dopazo, Carola de Benito and Oscar Camps; Methodology, Stavros Stavrinides; Supervision, Rodrigo Picos; Validation, Pablo Dopazo and Stavros Stavrinides; Visualization, Pablo Dopazo and Stavros Stavrinides; Writing – original draft, Pablo Dopazo, Oscar Camps, Stavros Stavrinides and Rodrigo Picos; Writing – review editing, Oscar Camps, Stavros Stavrinides and Rodrigo Picos.

Funding: Some of the authors wish to acknowledge support from DPI2017-86610-P, TEC2017-84877-R projects, awarded by the MICINN and also with partial support by the FEDER program.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments:

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Sample Availability: Sample code is available from the authors.

References

1. Mishra, S.; Bande, P. Maze Solving Algorithms for Micro Mouse. 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems, 2008, pp. 86–93.
2. Fattah, M.; Airola, A.; Ausavarungnirun, R.; Mirzaei, N.; Liljeberg, P.; Plosila, J.; Mohammadi, S.; Pahikkala, T.; Mutlu, O.; Tenhunen, H. A low-overhead, fully-distributed, guaranteed-delivery routing algorithm for faulty network-on-chips. Proceedings of the 9th International Symposium on Networks-on-Chip, 2015, pp. 1–8.
3. Kathe, O.; Turkar, V.; Jagtap, A.; Gidaye, G. Maze solving robot using image processing. 2015 IEEE Bombay Section Symposium (IBSS). IEEE, 2015, pp. 1–5.
4. Suriyanath. Dijkstra Algorithm, 2014.
5. Caruso, F.; Crespi, A.; Ciriolo, A.G.; Sciarrino, F.; Osellame, R. Fast escape of a quantum walker from an integrated photonic maze. *Nature communications* **2016**, *7*, 1–7.
6. Vourkas, I.; Stathis, D.; Sirakoulis, G.C. Massively parallel analog computing: Ariadne's thread was made of memristors. *IEEE Trans. on Emerging Topics in Computing* **2015**, *6*, 145–155.
7. Papandroulidakis, G.; Vourkas, I.; Sirakoulis, G.C.; Stavrinides, S.G.; Nikolaidis, S. Multi-state memristive nanocrossbar for high-radix computer arithmetic systems. 2015 IEEE 15th International Conference on Nanotechnology (IEEE-NANO). IEEE, 2015, pp. 625–628.
8. Pershin, Y.V.; Di Ventra, M. Solving mazes with memristors: A massively parallel approach. *Physical Review E* **2011**, *84*, 046703.
9. Chua, L. Memristor-The missing circuit element. *IEEE Transactions on Circuit Theory* **1971**, *18*, 507–519.
10. Chua, L. Resistance switching memories are memristors. *Applied Physics A* **2011**, *102*, 765–783.
11. Biolek, D.; Kolka, Z.; Biolková, V.; Biolek, Z.; Potrebić, M.; Tošić, D. Modeling and simulation of large memristive networks. *Intl. Journal of Circuit Theory and Applications* **2018**, *46*, 50–65.
12. Svetoslavov, G.; Camps, O.; Stavrinides, S.G.; Picos, R. A Switched Capacitor Memristive Emulator. *IEEE Transactions on Circuits and Systems II: Express Briefs* **2020, (early access)**. doi: 10.1109/TCSII.2020.3032632.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1

13. Camps, O.; Stavrinides, S.G.; Picos, R. Efficient Implementation of Memristor Cellular Nonlinear Networks using Stochastic Computing. 2020 European Conference on Circuit Theory and Design (ECCTD). IEEE, 2020, pp. 1–4.
14. Saxena, V. A Compact CMOS Memristor Emulator Circuit and its Applications. 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS). IEEE, 2018, pp. 190–193.
15. Yu, D.S.; Sun, T.T.; Zheng, C.Y.; Iu, H.; Fernando, T. A simpler memristor emulator based on varactor diode. *Chinese Physics Letters* **2018**, *35*, 058401.
16. Pershin, Y.V.; Di Ventra, M. Emulation of floating memcapacitors and meminductors using current conveyors. *Electronics Letters* **2011**, *47*, 243–244.
17. Vourkas, I.; Abusleme, A.; Ntinas, V.; Sirakoulis, G.C.; Rubio, A. A Digital Memristor Emulator for FPGA-Based Artificial Neural Networks. Verification and Security Workshop (IVSW), IEEE Intl, 2016, pp. 1–4.
18. Stavrinides, S.; Picos, R.; Corinto, F.; Al Chawa, M.M.; de Benito, C., Mem-elements for Neuromorphic Circuits with Artificial Intelligence Applications; Elsevier, 2021; Vol. in press, chapter Implementing memristor emulators in hardware.
19. Picos, R.; Roldan, J.; Al Chawa, M.; Jimenez-Molinos, F.; Garcia-Moreno, E. A physically based circuit model to account for variability in memristors with resistive switching operation. 2016 Conference on Design of Circuits and Integrated Systems (DCIS), 2016, pp. 1–6.
20. Naous, R.; Al-Shedivat, M.; Salama, K.N. Stochasticity modeling in memristors. *IEEE Transactions on Nanotechnology* **2015**, *15*, 15–28.
21. Camps, O.; Al Chawa, M.M.; de Benito, C.; Roca, M.; Stavrinides, S.G.; Picos, R.; Chua, L.O. A Purely Digital Memristor Emulator based on a Flux-Charge Model. 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2018, pp. 565–568.
22. Intel. *Virtual JTAG Intel FPGA IP Core User Guide*. Intel.
23. Zou, H.; Huang, J.; Gao, M. The application of virtual JTAG technology in FPGA design and debugging. 2011 International Conference on Electrical and Control Engineering. IEEE, 2011, pp. 2637–2640.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 3 December 2021 doi:10.20944/preprints202112.0047.v1