FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Memristor based logic circuits

Luís Diogo de Almeida Outeiro

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Vítor Manuel Grade Tavares

Second Supervisor: Guilherme Luís Leitão Teixeira Guia de Carvalho

April 9, 2021

Resumo

A lei de Moore está a chegar ao fim. Por causa disto, novas arquiteturas computacionais estão a ser desenvolvidas. Um crossbar de memristors é uma memória no formato de matriz composta por memristors, sendo que cada memristor armazena pelo menos um bit. Um memristor pode ser vistor como uma resistência programável. No modo normal de funcionamento, o memristor mantém o seu valor de resistência e quando está a ser programado este permite que a sua resistência seja alterada, dentro de certos limites. O objetivo desta dissertação é a implementação de um crossbar 4 x 4 com a lógica de controlo integrada no mesmo substrato que o crossbar. É apresentada uma breve revisão de modelos de memristors e lógica com memristors, levando à escolha de uma arquitetura IMPLY. O circuito de controlo permite o endereçamento das linhas e colunas do crossbar de modo a que possam ser escolhidos os memristors que vão fazer parte da computação. Como este circuito de controlo é digital, este foi descrito como um modelo verilog e posteriormente sintetizado para uma tecnologia customizada composta pelas portas lógicas inversor, NAND e NOR. O circuito resultante foi simulado com uma operação nand, de modo a demonstrar o funcionamento do crossbar.

Abstract

Moore's law is reaching its end. Because of this, new computational architectures are being developed. A memristor crossbar is a matrix-arranged memory device made of memristors, each storing at least one bit. A memristor can be seen as a programmable resistor. In the normal operation it hold its resistance value and when being programmed its value can be changed between certain limits. The goal of this dissertation is the implementation of a 4 x 4 memristor crossbar with the control logic being integrated in the same substrate as the crossbar. A brief review of memristor models and logic with memristors is presented, leading to the choice of the IMPLY architecture. The control circuit allows the addressing of the crossbar rows and columns so that we can choose which memristors will be part of the computation. Because the control circuit is a digital circuit (combinational), it was described as a verilog module and then synthesized to a custom target technology made of inverters, NAND and NOR gates. The resulting circuit was simulated and a nand operation was performed to demonstrate the functionality of the crossbar.

Acknowledgments

I would like to express my sincere gratitude to both my supervisor and co-supervisor for their patience and motivation that was needed when writing the dissertation.

Also, I would like to thank Bilal Hussain for making the layout of the chip and the team at CENIMAT for the support.

I would like to thank the author of SciencePlots [1], which was used to plot the graphs of this dissertation and the creator of LTspice [2], which was used to perform some of the circuit simulations.

Finally, I would like to thank my family and friends.

Luís

"An idiot admires complexity. A genius admires simplicity."

Terry Davis

Contents

List of Figures

List of Tables

Abbreviations and Symbols

ASIC Application Specific Integrated Circuit. 18

CMOS Complementary Metal Oxide Semiconductor. 8, 9

CPU Central Processing Unit. 8

FET Field Effect Transistor. 9

MAGIC Memristor-Aided Logic. 13

MOSFET Metal Oxide Field Effect Transistor. 8

MSE Mean Square Error. 5

TEAM ThrEshold Adaptive Memristor. 4

TFT Thin-Film-Transistor. ix, xi, 2, 3, 8, 9, 15, 17–19, 25

TN Twisted Neumatic. 8

VTEAM Voltage ThrEshold Adaptive Memristor. 4, 5

Chapter 1

Introduction

1.1 Context and Motivation

The memristor (memory resistor) is a passive two-terminal electronic component that behaves like a programmable resistor. Applying a certain voltage to its terminals allows the resistance value to change. In 1971, Leon Chua predicted the existence of the memristor by assuming that there is a passive electronic component per each pairwise relation of voltage (V), current (I), charge (Q) and flux linkage (Φ) [3]. Later in 2008, with the production of the first integrated circuit with memristors by HP, many researchers were encouraged to continue the developments in this field. The study of the physical conduction mechanisms, modeling, integration with CMOS, and novel architectures are some of the research topics that need further investigation. This electronic component has different applications in electronics such as memories, logic elements, oscillators, trimming [4] and neuromorphic circuits. With the Moore's law approaching its limit, there is a rising need of new computation architectures so that the computational performance keeps growing at the Moore's law growth rate. Also, the integration of transistors and memristors in the same integrated circuit opens the door to non Von-Neumann architectures such as dataflow and reduction machines, that are a natural solution to problems that can be parallelized. As Stanley Williams said,

> "Putting non-volatile memory on top of the logic chip will buy us twenty years of Moore's Law."

1.2 Problem Statement

This thesis is dedicated to the study and implementation of memristor circuits that are able to compute logic functions. The control sub-circuit is implemented with thin-film transistors. This circuit is responsible for driving both bit and word lines of a matrix arrangement of memristors, know as crossbar.

1.3 Solution

An 4 x 4 memristor crossbar was implemented and integrated with a thin-film-transistor based control circuit. The crossbar can work as a typical memory and also allows within-memory computations using a sequence of implication logic operations.

1.4 Contributions

This dissertations presents the different challenges faced when developing logic circuits with memristors. The contributions of the dissertation are:

- Implementation in Verilog-A of a memristor model based on a voltage threshold.
- Design and test of an integrated crossbar architecture and control circuit using a 5 µm TFT technology that works as both memory and computation unit, marking a new step in the state of the art of memristor and TFT integration using similar materials.

1.5 Document Structure

This dissertation starts with a brief introduction 1 that exposes the motivation behind the need of new computational architectures and the presented solution to the given problem. Then follows a background chapter 2 that sets the base concepts required for the comprehension of the following chapters. In chapter 3 we dive into the state of the art of logic circuits with memristors that has been developed over the years up to the date of writing of this dissertation. In chapter 4 the used models, libraries and tools required for the simulation and design of the crossbar components are explained, following the discussion of the architecture of the proposed solution and finally, the schematic and layout implementation.

Chapter 2

Background

In this chapter one can learn more about what a memristor is, its physical implementation, circuit models, crossbars, memristor logic and TFTs.

2.1 Memristor

The memristor, as the name suggests, is a resistor that has memory. This means that the resistance value of the device at the time t depends on the values of voltage/current at its terminals at times before t. The memory property is often modeled as a differential equation. What is this differential equation like and how can one arrive to its mathematical form? This is a question that can only be answered rigorously after the development of a physical model of the memristor. This model can then be used to create simplified models that are adequate for simulation. Let's first discuss how the memristor first appeared. Leon Chua, in 1971, postulated that a passive, fundamental device that relates the flux linkage Φ and charge q should exist. In figure 2.1 we can see the 4 fundamental passive electronic components interconnected by the respective equations. However, and according to different authors, the memristor is not considered a fundamental passive device [5]. If we consider that the memristance M is constant, one can say that the memristor is exactly similar to a resistor, as a result of equations (2.1), (2.2) and (2.3).

$$\int \Phi \, dt = V \tag{2.1}$$

$$\int q \, dt = I \tag{2.2}$$

$$d\Phi = M \cdot dq \tag{2.3}$$

In this dissertation we will consider the memristor to be a device with hysteresis that can alternate between different values of memristance. Being a fundamental component or not, the physical memristor exists and its importance in electronics is not reduced by this categorization. The symbol to be used for the memristor is pictured in figure 2.2.

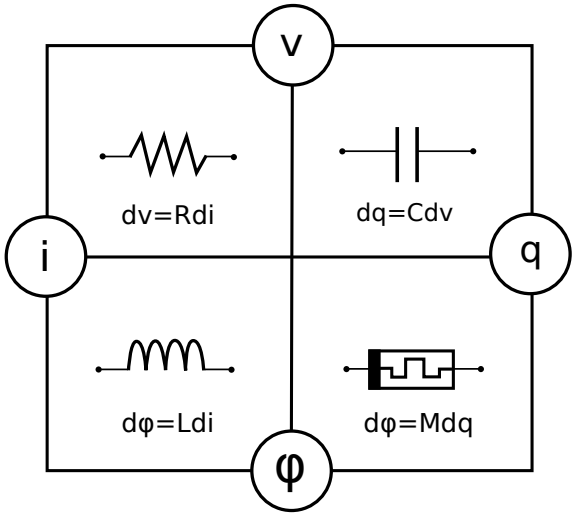


Figure 2.1: Basic circuit elements and physical units

2.2 Memristor models

2.2.1 HP memristor

In 2008, HP Labs produced the first crossbar array with memristors. The simplified model of the memristor 2.3 translates the ionic motion into a variable resistance that is dependent on an internal state variable w. Equation 2.4 is similar to Ohm's law and 2.5 describes how the internal state variable changes depending on the current that passes through the device. This simplified model considers two different regions in space sandwiched in between two metal contacts, show in figure2.3. The width of the TiO_{2-x} represents the spatial distribution of the oxygen vacancies so that an increment of this width increases the conductance. The resistance varies between R_{min} and R_{max}. as w varies between L and 0.

$$V(t) = \left[R_{off}\frac{w(t)}{L} + R_{on}\left(1 - \frac{w(t)}{L}\right)\right] \cdot I(t) \tag{2.4}$$

$$\frac{dw}{dt} = \frac{\mu_v R_{on}}{L} \cdot I(t) \tag{2.5}$$

2.2.2 VTEAM

Voltage ThrEshold Adaptive Memristor (VTEAM) [6] is a memristor model that takes into consideration a voltage threshold. This model is based on the ThrEshold Adaptive Memristor (TEAM) [7] model, that describes a current-threshold memristor. The internal state of the memristor is

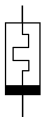


Figure 2.2: Memristor schematic symbol

described by equation 2.6.

$$\frac{dw(t)}{dt} = \begin{cases} k_{off} \cdot (\frac{v(t)}{v_{off}} - 1)^{\alpha_{off}} & 0 < v_{off} < v \\ 0, & v_{on} < v < v_{off} \\ k_{on} \cdot (\frac{v(t)}{v_{on}} - 1)^{\alpha_{on}} & v < v_{on} < 0 \end{cases} \tag{2.6}$$

All of the unknown variables of equation 2.6 can be determined using experimental data. Gradient descent is one of the methods that can make the model fit the given data, by minimizing a metric of error, for instance, the Mean Square Error (MSE).

The current-voltage relationship in VTEAM can be defined in many different ways; equation 2.7 is one example, as suggested in [6].

$$v(t) = \left[R_{ON} + \frac{R_{OFF} - R_{ON}}{w_{off} - won} \cdot (w - w_{on}) \right] \cdot i(t) \tag{2.7}$$

2.2.3 Window functions

Window functions are mathematical functions that ensure that the internal state variable that models the memristor stays within certain bounds. The working principle of these functions is to limit the derivative of the internal state variable to small values when the internal state variable is close to one of the boundaries. Depending on the implementation, the window function might introduce some problems to the model, for instance, slow changing near the boundaries.

A. Biolek

The Biolek window function depends on x (the normalized internal state variable) and I (current passing through the memristor) and is defined in equation (2.8).

$$f_B(x, I) = 1 - (x - step(-I))^{2p} \tag{2.8}$$

B. Joglekar

The Joglekar window function depends on x (the internal state variable) and is defined in equation 2.9.

$$f_J(x) = 1 - (2x - 1)^{2p} \tag{2.9}$$

As a conclusion, the Biolek window function has the advantage of not limiting the derivative of the internal state variable at the boundaries if the current applied to the memristor terminals

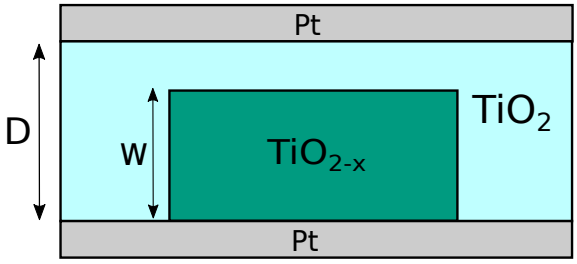


Figure 2.3: Doped and undoped regions of HP memristor

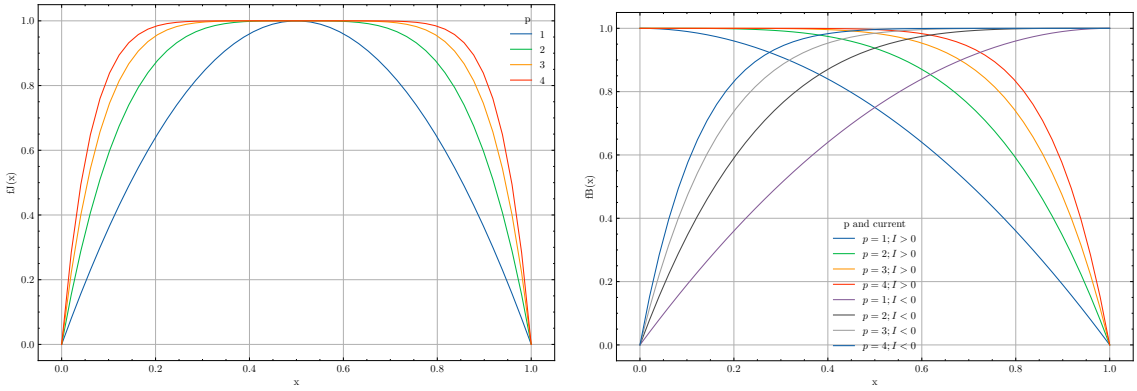
makes the internal state variable move towards the other boundary. This prevents the internal state variable to get 'stuck' near its boundaries.

2.3 Logic with memristors

Logic gates can be seen as abstract entities that perform logic operations. There are many ways to implement devices that perform these operations. A well known case is the transistor implementation, where the logic values 0/1 are mapped to voltages (0 V/5 V, for example). However, this is not the only way of performing these operations. Even though they are not very practical, it is possible to build an hydraulic computer [8] or even build the same circuit using dominoes [9]! The requirement is that one can consistently map a logic 0/1 to a physical quantity and ensure that the mapped output of the physical implementation of the gate is according to the truth table of the gate we want to build. With this, we can start to think about a way of implementing logic gates with memristors. We know that the memristance can have any value between R_{min} and R_{max}. Each bit of information can be mapped to a range of high/low conductances, rather than being mapped to voltages. We will consider 1 as high conductance (R_{min}) and 0 as low conductance (R_{max}). Depending on the behavior of the memristor, the implementation of the logic gates is different. This is why a good model of the memristor is required for circuit design. Let's consider the case of a memristor with the following behavior:

- The internal state of the memristor changes if the applied voltage to its terminals is greater than V_{clear_min}
- The internal state of the memristor changes in the other direction if the applied voltage to it's terminals is less than V_{set_max}
- If none of the above conditions are met, the internal state of the memristor remains unchanged.

This is a generic threshold model of a memristor.



(a) Joglekar window function

(b) Biolek window function

Figure 2.4: Two commonly used window functions. These ensure that the memristor internal state variable is bounded

2.3.1 IMPLY

The behavior described in section 2.3 allows us to implement an IMPLY logic operation having the truth table represented in table 2.1.

The inputs of the logic operation are the conductance of memristors P and Q, (1 for high conductance and 0 for low conductance) and the output of the operation is stored as the conductance of memristor Q. To perform the logic operation we must apply voltages V_{COND} and V_{SET} to the respective memristor, as pictured in 2.7. To understand how this operation works, lets consider a particular case where the memristor state does not change if the voltages applied to its terminals is between $-1\,\mathrm{V}$ and $1\,\mathrm{V}$. Also, $R_{min} = 10\,\mathrm{k\Omega}$, $R_{max} = 100\,\mathrm{k\Omega}$ and $R_G = 9\,\mathrm{k\Omega}$.

In this case, we can apply a voltage of $-0.85\,\mathrm{V}$ to the memristor P and $-1.3\,\mathrm{V}$ to the memristor Q. The result of this operation is stored as the memristor state Q. This should only change in the case where both P and Q are in a high resistive state. For the state to change, the voltage across memristor Q should be below -1V, so it drives this same memristor to a low resistive state. The allowed V_{SET} and V_{COND} voltages for $R_G = 1\,\mathrm{k\Omega}$ and $R_G = 9\,\mathrm{k\Omega}$ are displayed in figure 2.6.

2.3.2 Memristor crossbar

A crossbar is a matrix arrangement of memristors forming a 2D memory. Each memristor can be addressed as a 1 bit memory element by selecting the respective row and column of the crossbar, pictured in figure 2.8.

Table 2.1: Truth table of the material conditional/implication

p	q	$(p \rightarrow q)$
F	F	T
F	T	T
T	F	F
T	T	T

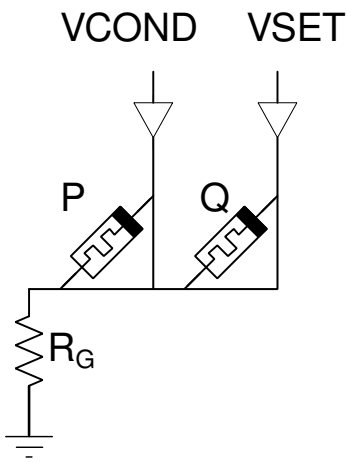


Figure 2.5: IMPLY logic gate with 2 memristors and 1 resistor

However, the use of the memristor crossbar is not limited to read and write operations of memory elements. It is also possible to make in-place logic operations with the information present in the crossbar memory elements. This means that the memristor crossbar extends the operation of a typical memory. This means that in a typical Central Processing Unit (CPU) architecture, the CPU can still copy memory content to its registers, perform logic/arithmetic operations on the data and copy the results back into memory. The operations can also be performed in-place, exploiting the benefits of the memristor crossbar architecture: By applying adequate voltage stimuli to the row/column lines of certain memory elements it is possible to perform any logic operation. This is done by decomposing the operation we want to perform in simpler ones, in this case, reset and imply operations.

2.4 TFTs

TFT is a type of field effect transistor whose substrate is an insulator instead of the typical Metal Oxide Field Effect Transistor (MOSFET) that have a semiconductor bulk. They are built by depositing thin layers of different materials to give shape to the metal contacts, active semiconductor areas and dielectric layers. The main application of these in the production of liquid crystal displays (LCDs). In the case of Twisted Neumatic (TN) panels, each transistor controls the orientation of liquid crystals through the application of an electric field. The orientation of these crystals defines the polarization of the light that reaches the topmost polarizing filter, allowing light to pass or blocking it. Over the last 10 years there has been a growth of the number of electronic circuits implemented entirely using thin-film technology, including 13.56 MHz RFID smart labels [10] used in many applications, as in transport systems. The performance of these type of transistors is much lower than typical Complementary Metal Oxide Semiconductor (CMOS) transistors. The low charge carrier mobility is one of the limiting factors that are translated to a low unity gain cut-off frequency and low intrinsic gain. Despite being slow, these transistor circuits are very cheap

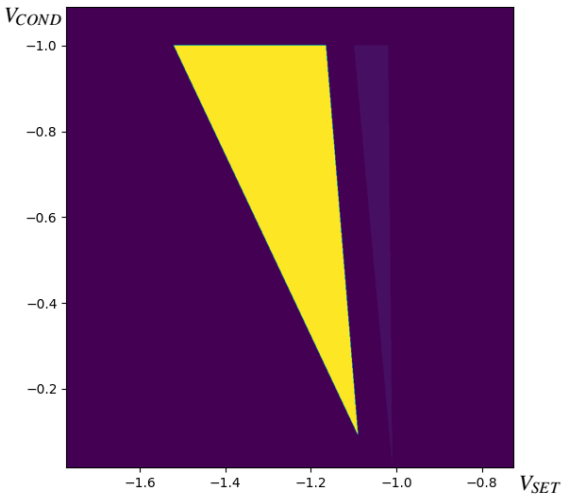


Figure 2.6: Triangular areas representing allowed V_{SET} and V_{COND} for $R_G = 1\,\text{k}\Omega$ and $R_G = 9\,\text{k}\Omega$ (yellow)

to manufacture compared to standard CMOS implementations [10], allowing the mass production of low cost, disposable technology.

TFT transistors can be manufactured in bottom-gate or top-gate, staggered or coplanar topologies. The topology to be used in the implementation of the crossbar circuit is bottom-gate, as shown in figure 2.9.

This type of transistor, like a typical Field Effect Transistor (FET), has three operating regions, cut-off, triode and saturation. In cut-off, the transistor does not conduct (there is only a small leakage current). In triode it behaves like a small-value resistor and in saturation it behaves as a voltage-controlled current source.

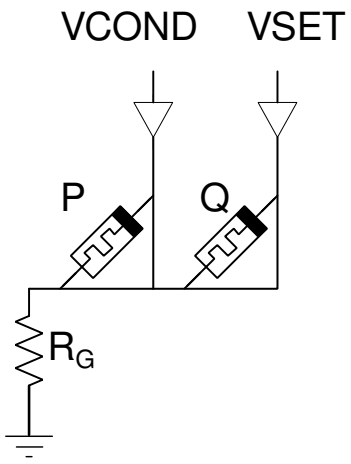


Figure 2.7: IMPLY logic gate uses 2 memristors and 1 resistor

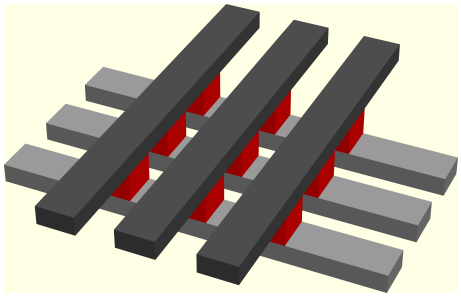


Figure 2.8: 3D representation of a memristor crossbar

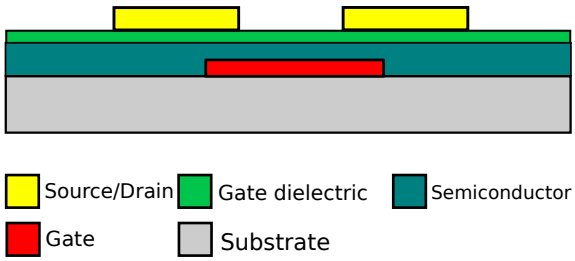


Figure 2.9: Bottom gate Thin-Film-Transistor

Chapter 3

Related work

This chapter serves as a base reference of the work that has been developed over the years about implication logic with memristors.

3.1 IMPLY Logic

Memristor-based implication logic circuits enable in-memory computation of any logic function. There are several important topics that need to be discussed in order to define circuit design constraints. The success of the proposed task depends not only on the choice of an appropriate circuit architecture but also on the characteristics of the chosen memristor, and intra-chip variability of the memristor/transistor parameters. Speed, power consumption, size and durability of the circuit is also an important topic, however, we will not focus on optimizing these to a great extent as the main goal is the logic correctness of the fabricated chip. As stated in [11] and [12], a linear ion drift memristor is not suitable for the implementation of IMPLY logic gates due to the drift of the internal state variable when parasitic voltages are applied. This is explained by the fact that this memristor does not have a voltage threshold and, as a consequence, small voltages applied to the memristor can change its state over time, destroying the stored data. Nevertheless, it is possible to mitigate this by increasing the non-linearity of the memristor. This can be done by including a pair of antiparallel diodes in series with the memristor, for instance. If the crossbar is properly designed, sneak-path currents through memristors will be very low because the voltage through the respective antiparallel diode pair will be bellow the diode built-in potential (0.7 V for silicon diodes). The addition of this non-linearity is not needed if the memristor is a memdiode, as it already presents this desired non-linearity. For memristors that are connected in a crossbar arrangement, the shortest sneak paths include 3 memristors in series. If we add one or more set of antiparallel diodes in series with each memristor, it is possible to increase the undesired switching time of the memristors that are in that sneak path without increasing the switching time of the memristor we want to change state too much. Figure 3.1 shows the switching time for 4 different scenarios:

- Memristor we want to change the state, without addition of antiparallel diodes in series

- Memristors belonging to the sneak path, without addition of antiparallel diodes in series
- Memristor we want to change the state, with antiparallel diodes added in series with each memristor
- Memristors belonging to the sneak path, with antiparallel diodes added in series with each memristor

From this we can conclude that the added non-linearity increases the time it takes to flip a memristor state that belongs to a sneak path, compared to the corresponding time of a memristor that does not have these diodes in series.

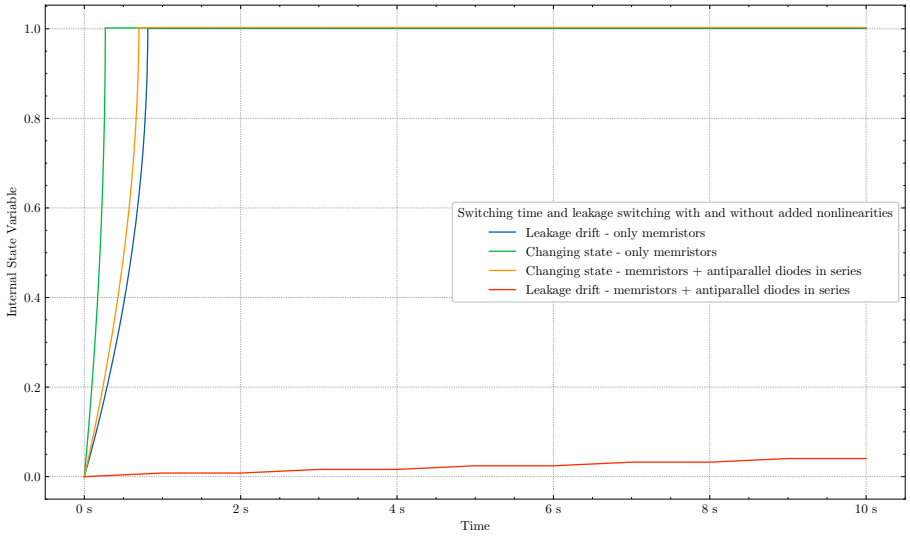


Figure 3.1: Switching times of a memristor vs and the ones in the sneak path with and without adding the antiparallel diodes in series with each memristor

Let's consider a memristor that is well modeled by linear ion drift. The equations that describe this model are (3.1) and (3.2).

$$v(t) = \left(R_{ON} \frac{w(t)}{D} + R_{OFF} \left(1 - \frac{w(t)}{D} \right) \right) i(t) \tag{3.1}$$

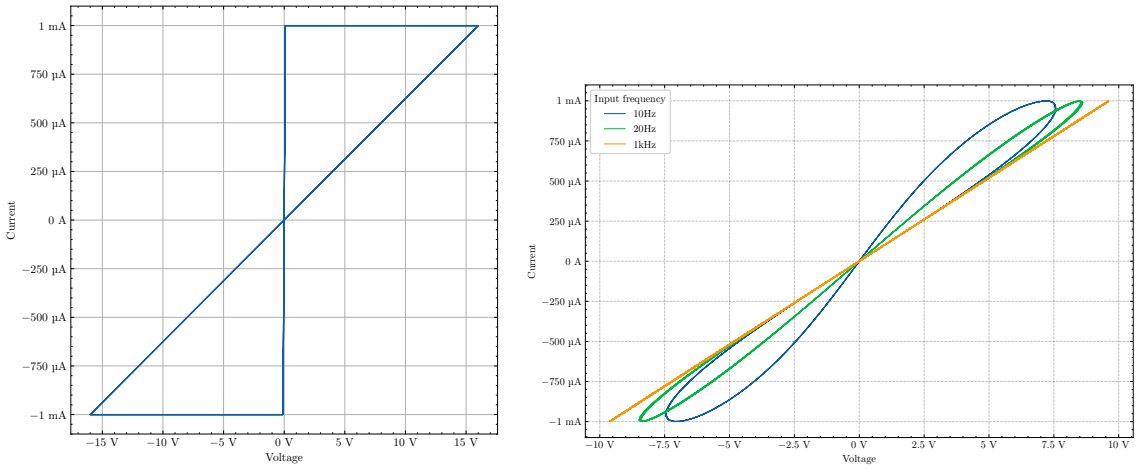
$$\frac{dw(t)}{dt} = \mu_v \frac{R_{ON}}{D} i(t) \tag{3.2}$$

This model can be implemented in SPICE. The differential equation (3.2) can be solved by considering an auxiliary circuit node L. We will choose the voltage in this node to be equal to the normalized internal state variable $x(t) = \frac{w(t)}{D}$ of the memristor. The state variable is bound between 0 and 1, implementing a window function with the same behaviour as a Biolek window function with very large p. The SPICE code is in appendix A.2. The I-V curve for a square-wave current source input is presented in figure 3.2a. It is composed of two almost straight lines, the steepest corresponding to R_{min} and the other to R_{max}. The transition between memristive states is

Table 3.1: Sequence of imply operations to perform a NAND operation

Operation	Voltage stimuli	
Step 1: $S = 0$	$V_S = V_{CLEAR}$	
Step 2: $p \implies s$	$V_P = V_{COND}$	$V_S = V_{SET}$
Step 3: $q \implies q$	$V_Q = V_{COND}$	$V_S = V_{SET}$

much faster than the input signal. For sinusoidal inputs, the I-V curve is a pinched hysteresis loop. For high frequencies (in this case, 1 kHz) the memristor behaves like a resistor.



(a) Linear ion drift memristor I-V curve $R_{ON} = 100\,\Omega$, $R_{OFF} = 16\,\text{k}\Omega$, $\mu_v = 100\,\Omega$, $D = 10\,\text{nm}$, $I_{SOURCE} = \pm 1\,\mu\text{A}$, $T = 2\,\text{s}$.

(b) Linear ion drift memristor IV curve with sinusoidal current input (1 mA peak current) at 10 Hz, 20 Hz and 1 kHz.

Figure 3.2: Linear ion drift IV plots

3.1.1 Making a NAND with IMPLY

Lets consider the NAND logic operation. It is important to note that the operation P $\implies$ Q is logically equivalent to ¬P *OR* Q, as well as *P* NAND ¬Q. Also, P $\implies$ 0 is equivalent to ¬P. Similarly, it is possible to describe a NAND operation as a composition of IMPLY operations:

$$\text{p NAND q} = p \implies \neg q \tag{3.3}$$

A NAND operation between memristor P and Q requires an auxiliary memristor S. The sequence of operations is the one in table 3.1. The result of the operation is stored as the memristance of memristor S.

3.2 MAGIC

Memristor-Aided Logic (MAGIC) [13] is an alternative to IMPLY logic. This logic family does not use resistors, instead, logic operations are performed only using memristors. Both inputs and

outputs of the performed logic operations are stored as memristor states.

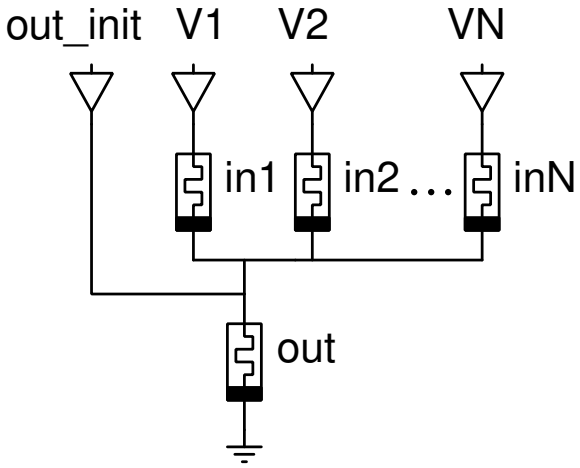


Figure 3.3: N input MAGIC NOR gate

The initial step consists in writing a 1 (low resistance) to the output memristor, by applying V_{CLEAR}. This is the same V_{CLEAR} that is used in IMPLY gates. Memristors $in1$ and $in2$ are also initialized, if needed. The second step consists in applying V_0 to both V1 and V2. If at least one of the memristors $in1$ or $in2$ is in a low memristive state, then the voltage at the middle node will be approximately V_0, changing the state of the output memristor. The choice of V_0, the write 1 (V_{SET}) voltage and the write 0 (V_{CLEAR}) voltage must follow certain rules, so the logic operation is correctly implemented and also that the input memristors does not change state.

Chapter 4

Fully integrated IMPLY logic

In this chapter we present the setup that was used to design the memristor crossbar and the required control logic, from the memristors and TFTs models to the layout of the chip.

4.1 Memristor modeling

After testing several different memristor model implementations, one concluded that most of the implementations have convergence problems even when used in very simple circuits [14]. Initially, a verilogA model for the memristor was implemented and can be found in appendix A.3.

With this, a simple model was implemented in SPICE that takes into account two threshold voltages, one for the *on* and another for *off* states. The internal state is represented as a circuit voltage of the node *L*. The voltage of this node can vary from 1 V to 10 V. The memristor resistance is $10\,\mathrm{k}\Omega \times \mathrm{V(L)}$ which means that $R_{min} = 10\,\mathrm{k}\omega$ and $R_{max} = 100k\Omega$. Note that the internal state is dimensionless, even though it is represented as a voltage for the simulation program. If the voltage across the memristor is over 1 V (positive threshold voltage) then the internal state will start to change at a constant rate until node L reaches 10 V. If the voltage across the memristor is less than -1 V then the internal state will start to decrease at a constant rate until node L reaches 1 V. The state node voltage increases/decreases by charging a capacitor connected to that node by one of the two current sources connected to it. The equivalent circuit is presented in figure 4.1 and the respective SPICE code can be found in appendix A.1. This model does not take into account that the change of rate of the internal state depends on the applied voltage to the terminals of the memristor as well as the state itself. Also, it does not model the drift of the internal variable when small voltages are applied to its terminals (below positive and above negative thresholds). However, the model is good enough so that it allows the design of memristor circuits including crossbar circuits. The model also behaves satisfactorily when it comes to represent the sneak path currents. Most importantly, the current that goes through it is 0 A when the potential difference at its terminals is 0 V. It also models the low-pass filter behaviour of a memristor, this is, when very small pulses of voltage are applied to its terminals so that it changes its state (above positive threshold or below negative threshold) the state will only vary a small quantity that is proportional

to the pulse duration. Because of the chosen capacitor size (10 µF) and the current sources that charge/discharge node L (−1 A / 1 A), the time that takes the internal state to vary from the minimal value to its maximum value is given by the charge time of a capacitor by a constant current source $\Delta t = C \cdot \Delta V / I$, which gives $t = 10\mu \cdot (10-1)/1 \approx 1\,\mu s$. This transition is much faster than the speed of the thin-film-transistors that make the control circuit, which allows us to say that the memristor speed is not the bottleneck of the circuit. This time of 1 µs is rather conservative, as physical implementations of memristors with switching times of 50 ns have already been achieved.

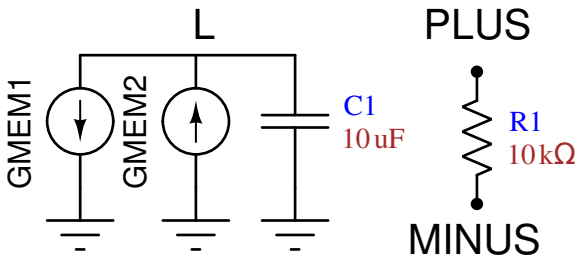


Figure 4.1: Memristor simple model

4.2 Memristor Simulation

A square wave input voltage from −2 V to 2 V was used to test the memristor. The results are according to the simulation model, as show in figure 4.2. The I-V curve is shown in figure 4.3, where both R_{min} and R_{max} can be extracted by taking the inverse of the slopes of the 2 straight lines. A large slope corresponds to a large conductance.

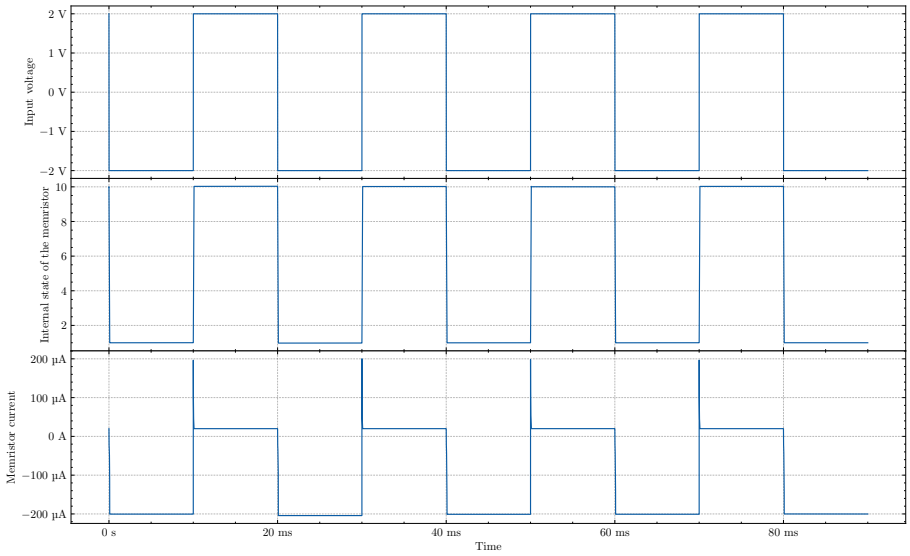


Figure 4.2: Memristor response to square input voltage

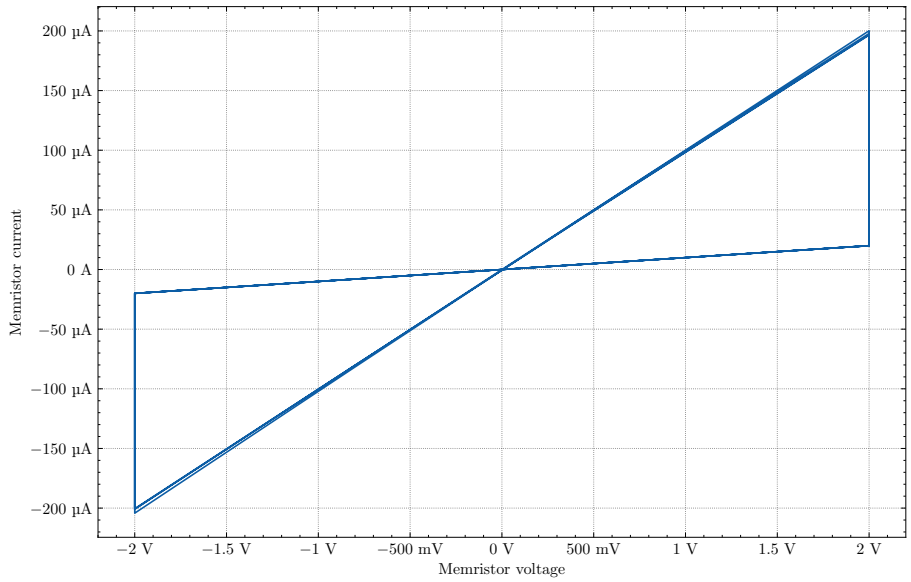


Figure 4.3: Current-Voltage curve of the memristor

4.3 TFTs models

The TFT model is based in a neural network whose weights are fitted with measurements taken from a real transistor. It is composed of an input layer with 3 inputs (V_G V_D V_S), 1 hidden layer with 105 hidden weights and 35 hidden biases, and one output (I_{DS}). The capacitor is approximated by a Meyers model. Due to convergence problems, an ideal switch transistor model was used instead of the TFT model, when simulating the crossbar.

4.4 TFTs simulation

A TFT with size $W/L = 20\mu/20\mu$ was used to trace the characteristic curves. The result can be seen in figure 4.4. Other sizes were also tested to confirm that the current scales linearly with W. One of the problems of this model is that it does not work well when small currents are involved. This can be seen in figure 4.4. This is due to the fact that the transistor is modeled by a neural network that does not enforce a zero current when all of the transistor terminals are at zero potential. For a transistor of size $W/L = 20\mu/20\mu$, when all of its terminals are at 0 V, the drain-source current is 5.5 µA, which is not physically possible. This could be solved by implementing a square-law transistor model.

A non-bootstrap TFT NAND gate was simulated in Cadence Virtuoso with a supply voltage of 8 V peak-to-peak. The result of the simulation is in figure 4.5.

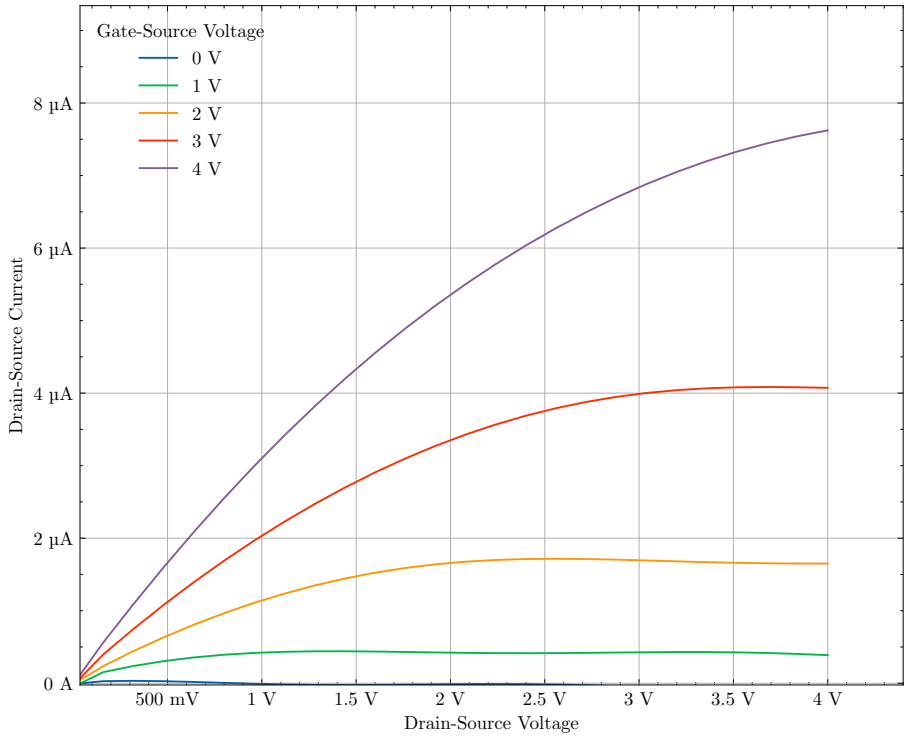


Figure 4.4: TFT characteristic curve

4.5 Logic synthesis and schematic generation

The first step in implementing the control circuit logic was to design a structural Verilog block that allows the addressing of one row and one column of the memristor crossbar. This Verilog code is available in appendix B.2. According to the control circuit and figure 4.8 the Voltage V_{SET} is applied to the column **i** if the $[3:0]set\_addr$ is set to the correct address and signal **set** is 1. The same applies for voltages V_{CLEAR} and V_{COND}. A row also has to be selected, by setting the signal **grnd** to 1 and $[3:0]grnd\_addr$ to the address we want to connect to ground through R_G.

Next, this code had to be synthesized to a target technology, and the open source tool Yosys was chosen. The software allows the target technology to be an FPGA or an Application Specific Integrated Circuit (ASIC). In this case we want an ASIC so we must first specify the .lib (Liberty) files that describe the standard cells technology. In this case we only provide the synthesis tool with 3 basic logic gates (INVERTER, NAND and NOR), presented as a Verilog logic function. The .lib file is presented in appendix B.

4.6 Crossbar design

The design of the crossbar started with the simulation and development of memristor models. Using the Cadence Virtuoso simulation environment it was shown that the transistor can only operate up to a few kHz before it started to lose gain. Taking into consideration that this circuit is a proof of concept and that the memristors do not impose an upper bound on the switching time, we

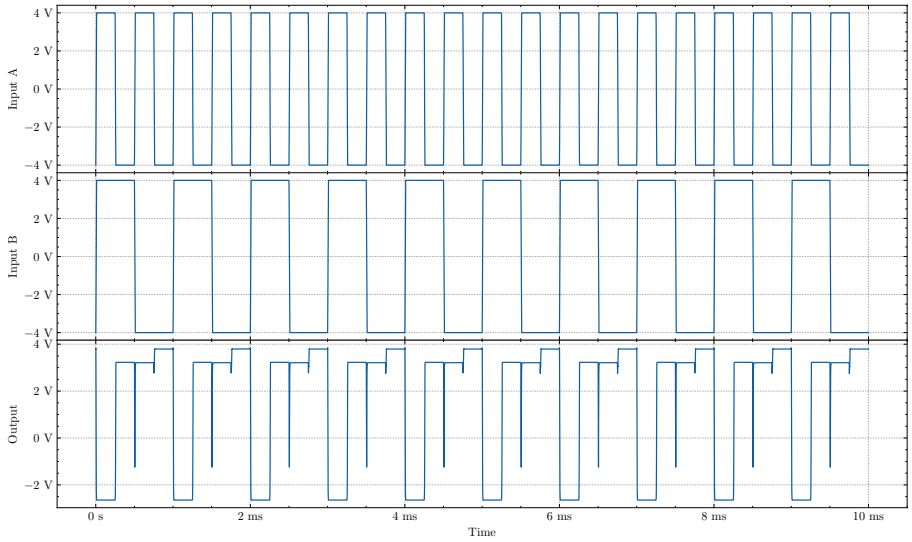


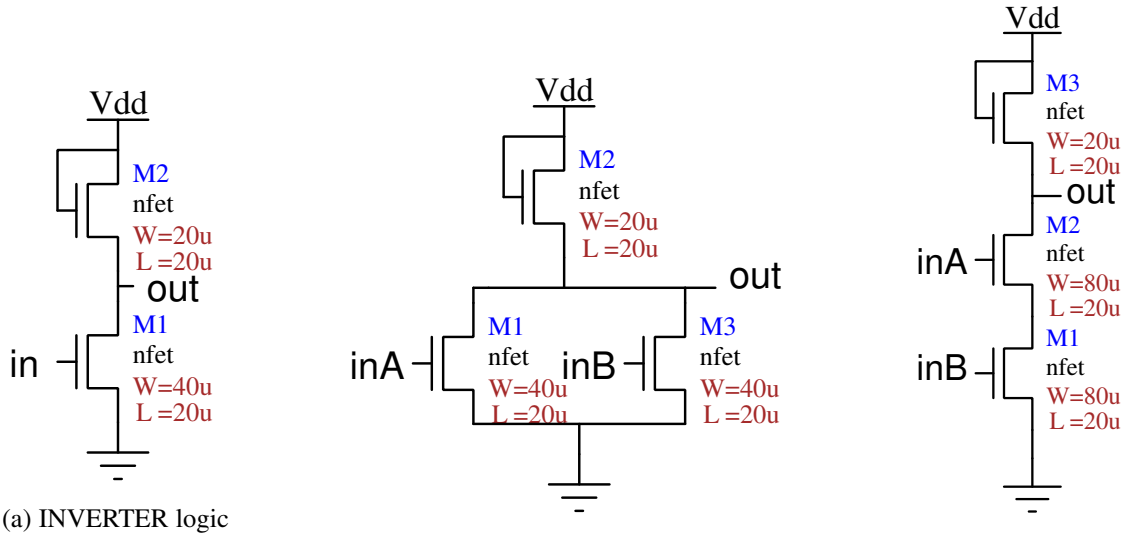
Figure 4.5: Non-bootstrap TFT NAND gate simulated voltage output response to 2 kHz and 1 kHz voltage inputs

can work at lower clock frequencies without impact on the functionality of the circuit. A voltage threshold memristor model was chosen for simulations, being this model described in section 4.1. Three logic gates were implemented using NMOS TFTs, as pictured in figures 4.6a, 4.6c and 4.6b. These NMOS-only logic gates have the problem that the output voltage can not go above $V_{DD} - V_{TH}$. To fix this issue, a new set of logic gates was implemented where a bootstrap load is used in place of a diode connected transistor. The bootstrap load is a dynamic circuit that allows its diode connected transistor gate voltage to go above V_{DD}, so that the output can reach V_{DD}. The schematic of these gates is pictured in figures 4.7a, 4.7c and 4.7b. The DC operating point of node GL is $V_{DD} - V_{TH}$. When the output of the logic gate is at 0 V and starts rising, the node GD also starts to rise through capacitor C. The $\Delta V_{GL}/\Delta V_o \approx 1$ if we do not consider the voltage divider with capacitor Cd. This positive feedback circuit allows almost rail to rail operation. A logic 0 would ideally be represented by a voltage equal to VSS (we chose to work with VDD and VSS instead of VDD and GND). However, because of the voltage divider consisting of the transistor resistances, the output node cannot go to V_{SS} but close to it. How close we get to V_{SS} is a matter of making the bottom transistor wider, at the expense of consuming more static power.

4.7 Layout

The layout was done hierarchically, being the logic gates implemented first and later more complex parts of the circuit.

The layout of the logic gates are presented in figure 4.9. The size and area of each of the building blocks is presented in table 4.1. As seen in figure 4.9, the major contributor to the logic gates area is the 10 pF capacitor used in the bootstrap load.

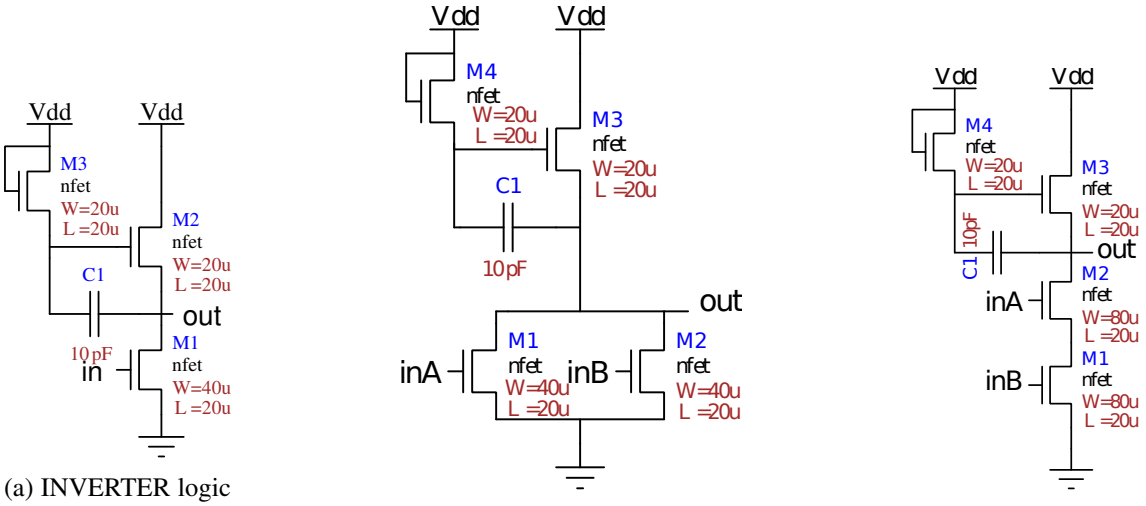


(a) INVERTER logic gate

(b) NOR logic gate

(c) NAND logic gate

Figure 4.6: Diode load logic gates



(a) INVERTER logic gate

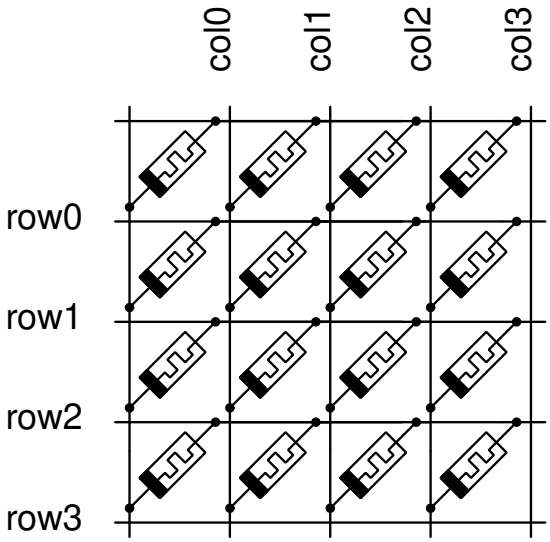
(b) NOR logic gate

(c) NAND logic gate

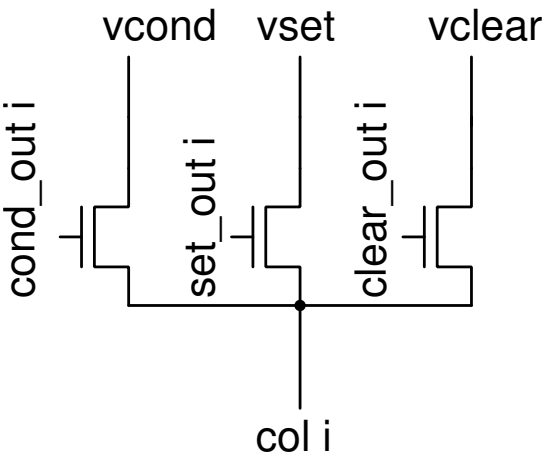
Figure 4.7: Bootstrap logic gates that make up the target technology of the digital synthesis

Table 4.1: Size area and transistor count of the control circuit

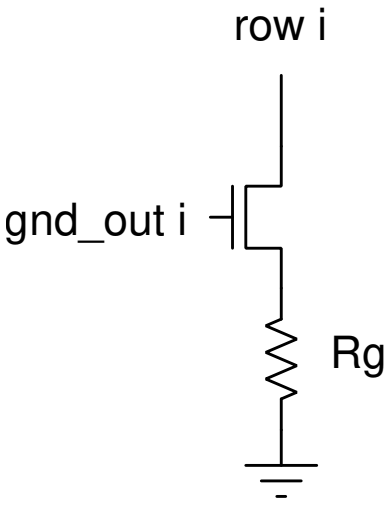
	Size (µm×µm)	Area (mm^2)	Transistor Count
Inverter	383 × 219	0.084	3
Nand	222 × 392	0.087	4
Nor	402 × 219	0.088	4
Control Circuit	5000 × 2500	12.5	120



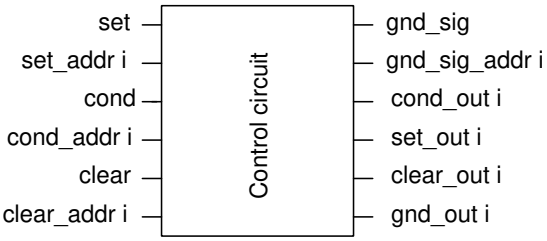
(a) Schematic of the implemented memristor crossbar



(b) Each column is addressed by $cond_{out}$, set_{out} and $clear_{out}$ signals that are generated by the synthesized digital block

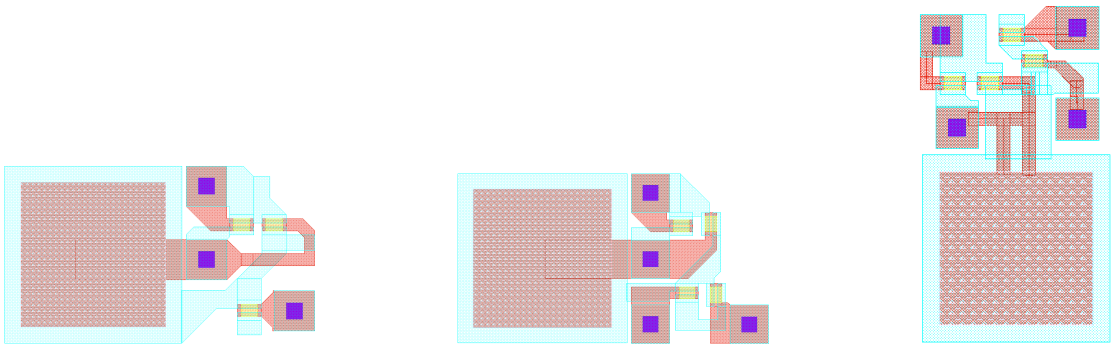


(c) Each row is addressed by gnd_{out} that is generated by the synthesized digital block



(d) Input Output block diagram of the synthesized digital control circuit

Figure 4.8: The main components of the crossbar



(a) INVERTER logic gate (b) NOR logic gate (c) NAND logic gate

Figure 4.9: Layout of the logic gates with bootstrap load

4.8 Crossbar simulation

An in-memory NAND operation was performed between 2 crossbar elements. The addressing of the elements is performed with the the addr.x and the respective enable signal. The results of the simulation is in figure 4.11. All of the memristors start with a state of 0. The NAND operation starts with a clear of the third memristor, which is equivalent to writing a 0 to that same memristor. Because its initial state was zero, its state remains the same. Then an implication from the first to the third memristor is performed, being the result stored in the third memristor. The result of the operation is 1, resulting in the third changing its state. Finally an implication from the second to the third memristor is performed, resulting in a final value of the computation of 0 NAND 0 = 1.

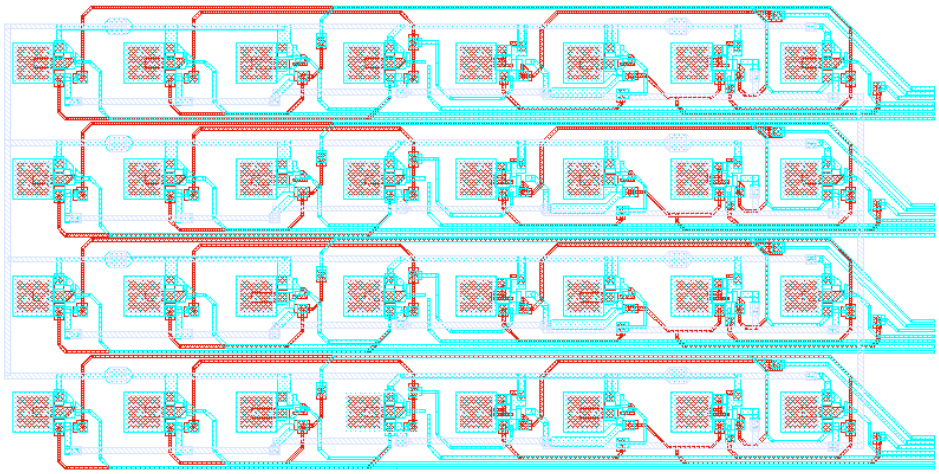


Figure 4.10: Layout of the control circuit

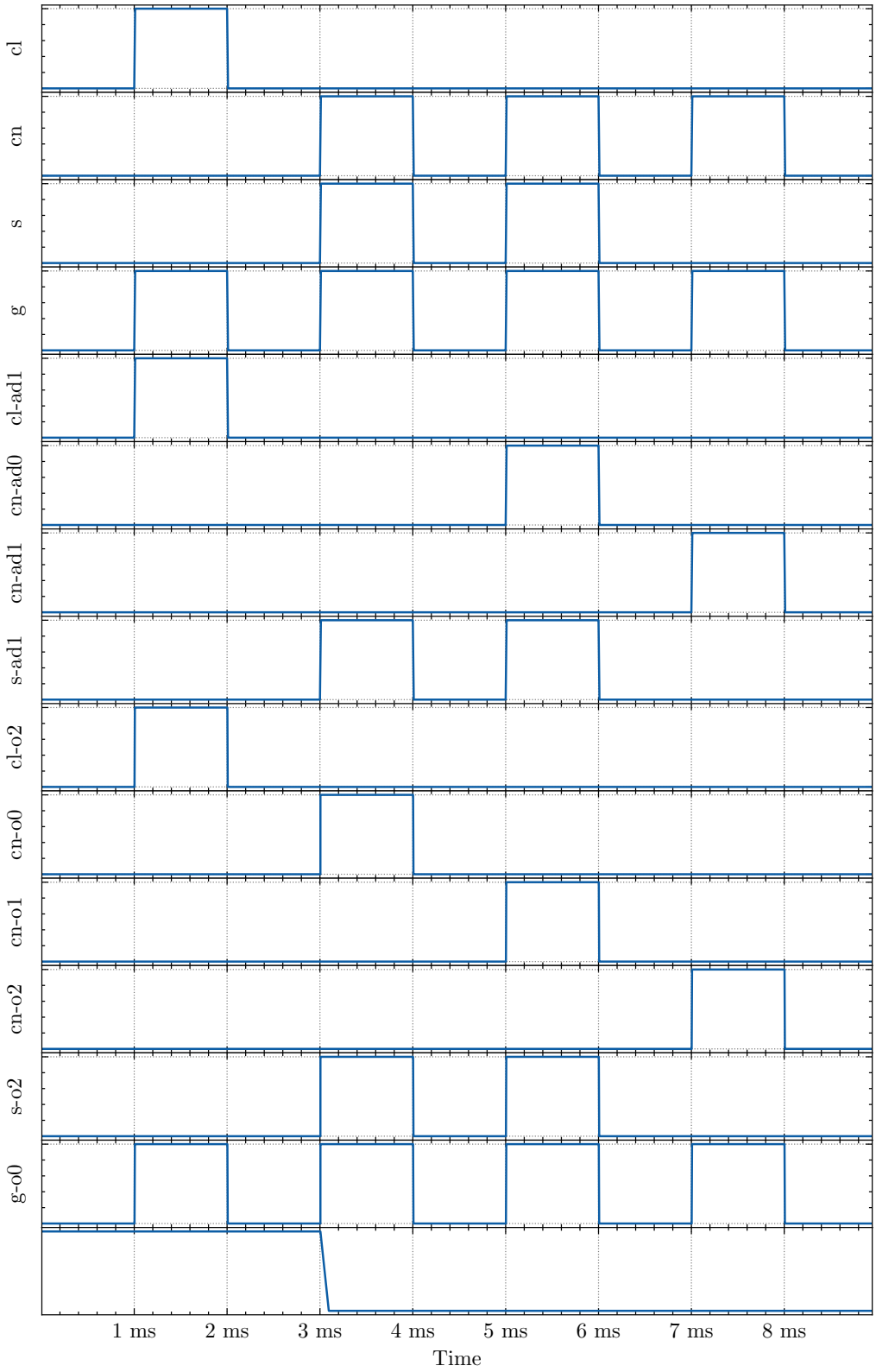


Figure 4.11: NAND operation within crossbar. The input memristors are memristor 0 and 1 from the first row and the output memristor is memristor 2 from the same row. s means set, cl means clear, cn means cond, g means grnd, o means out and ad means addr. The memristor state is the last signal and it is inverted, which means that a low signal represents a 1 and vice versa. All omitted signals are zero. Initial state values of all memristors are 0.

Chapter 5

Conclusion

Creating new architectures that enable in-memory computation is an important step to take if we want to extend Moore's law validity for a few more years. This dissertation presented one of these architectures, integrating TFTs and memristors in a single chip.

5.1 Future work

As future work it is important to perform adequate post-fabrication tests to make sure the chip is working according to the design specifications. A bigger crossbar also might be implemented, with an upgraded architecture that allows the operation between any two arbitrary memristors as the current architecture only allows performing logic operations between memristors belonging to the same row.

Appendix A

Simulation Models

Note: The character \ is used as a new line.

A.1 Simple Memristor Model - Spice

SIMPLE MEMRISTOR MODEL **LUIS OUTEIRO**

```
.subckt memristor PLUS MINUS L
R1 PLUS MINUS R={V(L) * 10000}
GMEM1 L 0 value = {(V(PLUS,MINUS) < -1) * (V(L) > 1)}
GMEM2 0 L  value = {(V(PLUS,MINUS) > 1 ) * (V(L) < 10)}
C1 L 0 10u
.ic V(L) = 10
.ends memristor
```

A.2 Linear Ion Drift Memristor Model - Spice

LINEAR ION DRIFT MEMRISTOR MODEL **LUIS OUTEIRO**

```
.subckt memristor_linear_ion_drift PLUS MINUS X
.param RON = {100} ROFF = {16k} L = {10n} mu = {10f}
R1 PLUS MINUS R= {RON} * V(X) + (1 - V(X))*{ROFF}
GMEM1 0 X value = {(V(X) > 0) * (V(X) < 1)*I(R1)*(MU * RON / (L^2))}
GMEM2 0 X value = {(V(X) >= 1) * (I(R1) <= 0)*I(R1)*(MU * RON / (L^2))}
GMEM3 0 X value = {(V(X) <= 0) * (I(R1) >= 0)*I(R1)*(MU * RON / (L^2))}
C X 0 1
.ic V(X) = 0.5
.ends memristor_linear_ion_drift
```

A.3 Memmristor with antiparallel diodes in series - Memristor Model

```
// VerilogA for memristor_lib, memristor, veriloga

`include "constants.vams"
`include "disciplines.vams"

module memristor (a, b, l);
  inout a,b;
  output l;
  electrical a, b, l, foff, fon, c1, c2, is1, is2, r1, r2;
  parameter koff=61.49, kon=-3.4*10**6, voff=1, von=-1.1, \
  aon=0.76, aoff=0.28, alfaon=0.36, alfaoff=0.21, is1on=0.68, \
  is1off=0.39, c1on=26*10**-3*5.02*10**-4, c1off=26*10**-3*53.82, \
  r1on=13.41, r1off=1.03*10**-2, is2on=3.56*10**-2, is2off=0.33, \
  c2on=26*10**-3*1.63, c2off=26*10**-3*11.73, r2on=13.35, r2off=17.2, \
  wc=0.29, L0=-0.086952;

  analog begin
    I(l) <+ 1 * ddt(V(l)); // C = 1
    if(V(a,b) > voff)
      I(l) <+ - pow( koff*(V(a,b)/voff-1) , alfaoff * exp(-exp((V(l)-aoff)/wc)));
    else if(V(a,b) < von)
      I(l) <+ - pow( kon*(V(a,b)/von-1) , alfaon * exp(-exp(-(V(l)-aon)/wc)));
    else
      I(l) <+ 0;
    I(a,b) <+ -V(is1) + V(c1)/V(r1) * log(1+(V(is1) * V(r1)/V(c1) *\
    exp(  (V(a,b)+ V(r1) * V(is1) )/ V(c1)  )))*(1-(log(1+log(1+(V(is1) * \
    V(r1)/V(c1) *exp(  (V(a,b)+ V(r1) * V(is1) )/ V(c1)  ))))/(2+log(1+(V(is1) \
    * V(r1)/V(c1) *exp(  (V(a,b)+ V(r1) * V(is1) )/ V(c1)  ))));

    I(a,b) <+ V(is2) - V(c2)/V(r2) * log(1+(V(is2) * V(r2)/V(c2) *exp(  (V(r2) \
    * V(is2) - V(a,b))/ V(c2)  )))*(1-(log(1+log(1+(V(is2) * V(r2)/V(c2) \
    *exp(  (V(r2) * V(is2) - V(a,b))/ V(c2)  ))))/(2+log(1+(V(is2) \
    * V(r2)/V(c2) *exp(  (V(r2) * V(is2) - V(a,b))/ V(c2)  ))));

    V(foff) <+ exp(-exp((V(l)-aoff)/wc));
    V(fon) <+ exp(-exp(-(V(l)-aon)/wc));
    V(c1) <+ c1on + (c1off-c1on)*V(l);
    V(c2) <+ c2on + (c2off-c2on)*V(l);
```

```
    V(is1) <+ is1on + (is1off-is1on)*V(l);
    V(is2) <+ is2on + (is2off-is2on)*V(l);
    V(r1) <+ r1on + (r1off-r1on)*V(l);
    V(r2) <+ r2on + (r2off-r2on)*V(l);

  end
endmodule
```

Appendix B

Design Files and scripts

B.1 Yosys synthesis commands

```
read_verilog driver_crossbar_4x4.v
read_verilog -lib black.v
proc; opt; memory; opt; fsm; opt
techmap; opt
dfflibmap -liberty mycells.lib
abc -liberty mycells.lib
clean
write_spice driver_crossbar_4x4_netlist.sp
show
```

B.2 4x4 crossbar control logic - Verilog

```
module driver(set, set_addr, cond, cond_addr, clr, \
clr_addr, grnd, grnd_addr, set_out, cond_out, clr_out, grnd_out);
input set;
input [1:0] set_addr;
input cond;
input [1:0] cond_addr;
input clr;
input [1:0] clr_addr;
input grnd;
input [1:0] grnd_addr;

output reg [3:0] set_out;
output reg [3:0] cond_out;
output reg [3:0] clr_out;
```

```
output reg [3:0] grnd_out;

always @(*)
begin
set_out = 0;
case (set_addr)
  2'b00: set_out[0] = set;
  2'b01: set_out[1] = set;
  2'b10: set_out[2] = set;
  2'b11: set_out[3] = set;

endcase
end

always @(*)
begin
cond_out = 0;
case (cond_addr)
  2'b00: cond_out[0] = cond;
  2'b01: cond_out[1] = cond;
  2'b10: cond_out[2] = cond;
  2'b11: cond_out[3] = cond;

endcase
end

always @(*)
begin
clr_out = 0;
case (clr_addr)
  2'b00: clr_out[0] = clr;
  2'b01: clr_out[1] = clr;
  2'b10: clr_out[2] = clr;
  2'b11: clr_out[3] = clr;

endcase
end

always @(*)
begin
```

```
grnd_out = 0;
case (grnd_addr)
  2'b00: grnd_out[0] = grnd;
  2'b01: grnd_out[1] = grnd;
  2'b10: grnd_out[2] = grnd;
  2'b11: grnd_out[3] = grnd;

endcase
end

endmodule
```

B.3 8x8 crossbar control logic - Verilog

```
module driver(set, set_addr, cond, cond_addr, clr, \
clr_addr, grnd, grnd_addr, set_out, cond_out, clr_out, grnd_out);
input set;
input [3:0] set_addr;
input cond;
input [3:0] cond_addr;
input clr;
input [3:0] clr_addr;
input grnd;
input [3:0] grnd_addr;

output reg [7:0] set_out;
output reg [7:0] cond_out;
output reg [7:0] clr_out;
output reg [7:0] grnd_out;

always @(*)
begin
set_out = 0;
case (set_addr)
  3'b000: set_out[0] = set;
  3'b001: set_out[1] = set;
  3'b010: set_out[2] = set;
  3'b011: set_out[3] = set;
  3'b100: set_out[4] = set;
  3'b101: set_out[5] = set;
```

```
  3'b110: set_out[6] = set;
  3'b111: set_out[7] = set;
endcase
end

always @(*)
begin
cond_out = 0;
case (cond_addr)
  3'b000: cond_out[0] = cond;
  3'b001: cond_out[1] = cond;
  3'b010: cond_out[2] = cond;
  3'b011: cond_out[3] = cond;
  3'b100: cond_out[4] = cond;
  3'b101: cond_out[5] = cond;
  3'b110: cond_out[6] = cond;
  3'b111: cond_out[7] = cond;
endcase
end

always @(*)
begin
clr_out = 0;
case (clr_addr)
  3'b000: clr_out[0] = clr;
  3'b001: clr_out[1] = clr;
  3'b010: clr_out[2] = clr;
  3'b011: clr_out[3] = clr;
  3'b100: clr_out[4] = clr;
  3'b101: clr_out[5] = clr;
  3'b110: clr_out[6] = clr;
  3'b111: clr_out[7] = clr;
endcase
end

always @(*)
begin
grnd_out = 0;
case (grnd_addr)
  3'b000: grnd_out[0] = grnd;
```

```
  3'b001: grnd_out[1] = grnd;
  3'b010: grnd_out[2] = grnd;
  3'b011: grnd_out[3] = grnd;
  3'b100: grnd_out[4] = grnd;
  3'b101: grnd_out[5] = grnd;
  3'b110: grnd_out[6] = grnd;
  3'b111: grnd_out[7] = grnd;
endcase
end

endmodule
```

B.4 .lib technology file

```
library(memristortft){
  comment:"Ttf memristor crossbar - Luis Outeiro - FEUP";
  cell(BUF) {
    pin(A) {direction: input;}
    pin(Y) {direction: output;
              function: "A"; }
  }
  cell(NOT) {
    pin(A) {direction: input;}
    pin(Y) {direction: output;
              function: "A'"; }
  }
  cell(NAND){
    pin(A) {direction: input;}
    pin(B) {direction: input;}
    pin(Y) {direction: output;
              function: "(A*B)'";}
  }
  cell(NOR){
    pin(A) {direction: input;}
    pin(B) {direction: input;}
    pin(Y) {direction: output;
              function: "(A+B)'";}
  }
  cell(DFF){
    ff(IQ, IQN) {clocked_on: C;
```

```
                next_state: D;}
    pin(C) {direction: input;
            clock: true;}
    pin(D) {direction: input;}
    pin(Q) {direction: output;
             function: "IQ";}
  }
}
```

B.5 Output of the digital synthesis of control block for 4x4 crossbar

```
* SPICE netlist generated by Yosys 0.9 (git sha1 UNKNOWN,\
 gcc 10.2.0 -march=x86-64 -mtune=generic -O2 -fno-plt -fPIC -Os)

.SUBCKT driver set set_addr.0 set_addr.1 cond cond_addr.0 \
cond_addr.1 clr clr_addr.0 clr_addr.1 grnd grnd_addr.0 \
grnd_addr.1 set_out.0 set_out.1 set_out.2 set_out.3 \
cond_out.0 cond_out.1 cond_out.2 cond_out.3 clr_out.0 \
clr_out.1 clr_out.2 clr_out.3 grnd_out.0 grnd_out.1 grnd_out.2 grnd_out.3
X0 grnd_addr.0 1 NOT
X1 grnd_addr.1 2 NOT
X2 clr_addr.0 3 NOT
X3 clr_addr.1 4 NOT
X4 cond_addr.0 5 NOT
X5 cond_addr.1 6 NOT
X6 set_addr.0 7 NOT
X7 set_addr.1 8 NOT
X8 grnd_addr.1 grnd 9 NAND
X9 grnd_addr.0 9 grnd_out.2 NOR
X10 2 grnd 10 NAND
X11 1 10 grnd_out.1 NOR
X12 1 9 grnd_out.3 NOR
X13 grnd_addr.0 10 grnd_out.0 NOR
X14 clr_addr.1 clr 11 NAND
X15 clr_addr.0 11 clr_out.2 NOR
X16 4 clr 12 NAND
X17 3 12 clr_out.1 NOR
X18 3 11 clr_out.3 NOR
X19 clr_addr.0 12 clr_out.0 NOR
X20 cond_addr.1 cond 13 NAND
```

```
X21 cond_addr.0 13 cond_out.2 NOR
X22 6 cond 14 NAND
X23 5 14 cond_out.1 NOR
X24 5 13 cond_out.3 NOR
X25 cond_addr.0 14 cond_out.0 NOR
X26 set_addr.1 set 15 NAND
X27 set_addr.0 15 set_out.2 NOR
X28 8 set 16 NAND
X29 7 16 set_out.1 NOR
X30 7 15 set_out.3 NOR
X31 set_addr.0 16 set_out.0 NOR
.ENDS driver

************************
* end of SPICE netlist *
************************
```

B.6 Output of the digital synthesis of control block for 8x8 crossbar

```
* SPICE netlist generated by Yosys 0.9 (git sha1 UNKNOWN, \
gcc 10.1.0 -march=x86-64 -mtune=generic -O2 -fno-plt -fPIC -Os)

.SUBCKT driver set set_addr.0 set_addr.1 set_addr.2 set_addr.3 \
cond cond_addr.0 cond_addr.1 cond_addr.2 cond_addr.3 clr clr_addr.0 \
clr_addr.1 clr_addr.2 clr_addr.3 grnd grnd_addr.0 grnd_addr.1 \
grnd_addr.2 grnd_addr.3 set_out.0 set_out.1 set_out.2 set_out.3 \
set_out.4 set_out.5 set_out.6 set_out.7 cond_out.0 cond_out.1 cond_out.2 \
cond_out.3 cond_out.4 cond_out.5 cond_out.6 cond_out.7 clr_out.0 clr_out.1 \
clr_out.2 clr_out.3 clr_out.4 clr_out.5 clr_out.6 clr_out.7 grnd_out.0 \
grnd_out.1 grnd_out.2 grnd_out.3 grnd_out.4 grnd_out.5 grnd_out.6 grnd_out.7
X0 set_addr.0 1 NOT
X1 set_addr.2 2 NOT
X2 set_addr.3 3 NOT
X3 set_addr.1 4 NOT
X4 cond_addr.0 5 NOT
X5 cond_addr.3 6 NOT
X6 cond_addr.2 7 NOT
X7 cond_addr.1 8 NOT
X8 clr_addr.3 9 NOT
```

```
X9 clr_addr.0 10 NOT
X10 clr_addr.1 11 NOT
X11 clr_addr.2 12 NOT
X12 grnd_addr.0 13 NOT
X13 grnd_addr.1 14 NOT
X14 grnd_addr.3 15 NOT
X15 grnd_addr.2 16 NOT
X16 2 set_addr.3 17 NOR
X17 set_addr.2 3 18 NAND
X18 set_addr.0 set_addr.1 19 NAND
X19 19 20 NOT
X20 set 20 21 NAND
X21 18 21 set_out.7 NOR
X22 1 4 22 NAND
X23 set 17 23 NAND
X24 22 23 set_out.4 NOR
X25 1 set_addr.1 24 NAND
X26 set_addr.2 set_addr.3 25 NOR
X27 25 26 NOT
X28 set 25 27 NAND
X29 24 27 set_out.2 NOR
X30 22 27 set_out.0 NOR
X31 23 24 set_out.6 NOR
X32 set_addr.0 4 28 NAND
X33 27 28 set_out.1 NOR
X34 21 26 set_out.3 NOR
X35 23 28 set_out.5 NOR
X36 cond_addr.3 7 29 NOR
X37 6 cond_addr.2 30 NAND
X38 cond_addr.0 cond_addr.1 31 NAND
X39 31 32 NOT
X40 cond 32 33 NAND
X41 30 33 cond_out.7 NOR
X42 5 8 34 NAND
X43 cond 29 35 NAND
X44 34 35 cond_out.4 NOR
X45 5 cond_addr.1 36 NAND
X46 cond_addr.3 cond_addr.2 37 NOR
X47 37 38 NOT
X48 cond 37 39 NAND
```

```
X49 36 39 cond_out.2 NOR
X50 34 39 cond_out.0 NOR
X51 35 36 cond_out.6 NOR
X52 cond_addr.0 8 40 NAND
X53 39 40 cond_out.1 NOR
X54 33 38 cond_out.3 NOR
X55 35 40 cond_out.5 NOR
X56 clr_addr.3 12 41 NOR
X57 9 clr_addr.2 42 NAND
X58 clr_addr.0 clr_addr.1 43 NAND
X59 43 44 NOT
X60 clr 44 45 NAND
X61 42 45 clr_out.7 NOR
X62 10 11 46 NAND
X63 clr 41 47 NAND
X64 46 47 clr_out.4 NOR
X65 10 clr_addr.1 48 NAND
X66 clr_addr.3 clr_addr.2 49 NOR
X67 49 50 NOT
X68 clr 49 51 NAND
X69 48 51 clr_out.2 NOR
X70 46 51 clr_out.0 NOR
X71 47 48 clr_out.6 NOR
X72 clr_addr.0 11 52 NAND
X73 51 52 clr_out.1 NOR
X74 45 50 clr_out.3 NOR
X75 47 52 clr_out.5 NOR
X76 grnd_addr.3 16 53 NOR
X77 15 grnd_addr.2 54 NAND
X78 grnd_addr.0 grnd_addr.1 55 NAND
X79 55 56 NOT
X80 grnd 56 57 NAND
X81 54 57 grnd_out.7 NOR
X82 13 14 58 NAND
X83 grnd 53 59 NAND
X84 58 59 grnd_out.4 NOR
X85 13 grnd_addr.1 60 NAND
X86 grnd_addr.3 grnd_addr.2 61 NOR
X87 61 62 NOT
X88 grnd 61 63 NAND
```

```
X89 60 63 grnd_out.2 NOR
X90 58 63 grnd_out.0 NOR
X91 59 60 grnd_out.6 NOR
X92 grnd_addr.0 14 64 NAND
X93 63 64 grnd_out.1 NOR
X94 57 62 grnd_out.3 NOR
X95 59 64 grnd_out.5 NOR
.ENDS driver

************************
* end of SPICE netlist *
************************
```

References

[1] J. D. Garrett. SciencePlots (v1.0.6). October 2020. URL: http://doi.org/10.5281/zenodo.4106650, doi:10.5281/zenodo.4106650.

[2] Mike Engelhardt, Linear Technology, Analog Devices. Ltspice. URL: https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html.

[3] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, 1971. doi:10.1109/TCT.1971.1083337.

[4] O. A. Olumodeji and M. Gottardi. Behavioural modelling of memristive devices targeted to sensor interfaces. In *2015 XVIII AISEM Annual Conference*, pages 1–4, 2015. doi:10.1109/AISEM.2015.7066780.

[5] Isaac Abraham. The case for rejecting the memristor as a fundamental circuit element. *Scientific Reports*, 8(1):10972, Jul 2018. URL: https://doi.org/10.1038/s41598-018-29394-7, doi:10.1038/s41598-018-29394-7.

[6] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny. Vteam: A general model for voltage-controlled memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(8):786–790, 2015. doi:10.1109/TCSII.2015.2433536.

[7] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser. Team: Threshold adaptive memristor model. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):211–221, 2013. doi:10.1109/TCSI.2012.2215714.

[8] Nicolas Taberlet, Quentin MARSAL, Ferrand Jérémy, and Nicolas Plihon. Hydraulic logic gates: Building a digital water computer. *European Journal of Physics*, 39, 11 2017. doi:10.1088/1361-6404/aa97fc.

[9] Domino computer. URL: https://www.youtube.com/watch?v=OpLU__bhu2w.

[10] Tsung-Ching Huang, Kenjiro Fukuda, Chun-Ming Lo, Yung-Hui Yeh, Tsuyoshi Sekitani, Takao Someya, and K.-T. Tim Cheng. Pseudo-cmos: A novel design style for flexible electronics. pages 154–159, 03 2010. doi:10.1109/DATE.2010.5457220.

[11] Shahar Kvatinsky, Avinoam Kolodny, Uri Weiser, and E.G. Friedman. Memristor-based imply logic design procedure. pages 142–147, 10 2011. doi:10.1109/ICCD.2011.6081389.

[12] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser. Memristor-based material implication (imply) logic: Design principles and methodologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(10):2054–2066, 2014. doi:10.1109/TVLSI.2013.2282132.

[13] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser. Magic—memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014. doi:10.1109/TCSII.2014.2357292.

[14] Tianshi Wang and Jaijeet Roychowdhury. Well-posed models of memristive devices. *arXiv preprint arXiv:1605.04897*, 2016.