Universidad de Valladolid

Facultad de Ciencias

TRABAJO FIN DE GRADO

Grado en Física

ESTUDIO DE FAMILIAS LÓGICAS MULTINIVEL BASADAS EN MATERIALES Y DISPOSITIVOS MEMRISTIVOS

Autor: Javier Aparicio Merino
Tutor: Salvador Dueñas Carazo
Año 2023/24

Índice general

Resumen y palabras clave

Resumen: durante los últimos 20 años se ha observado un crecimiento emergente de las tecnologías basadas en memristores para reemplazar o complementar a las memorias flash y RAM que se utilizan extensivamente hoy en día. En un memristor, o memoria resistiva, la resistencia es una variable que puede ser modificada mediante pulsos eléctricos adecuados. Estos dispositivos ofrecen también la posibilidad de diseñar funciones lógicas multinivel con propiedades, integración, consumo y velocidad muy superiores a los circuitos lógicos binarios convencionales. En este trabajo de fin de grado se persiguen dos objetivos principales: realizar una actualización documental del estado del arte de la lógica multinivel basada en memristores; diseñar y simular celdas lógicas combinacionales multi-nivel a partir de dispositivos reales fabricados para y medidos por el Grupo de Caracterización de Materiales y Dispositivos Electrónicos (GCME). Los datos experimentales ya disponibles serán utilizados para la simulación de las celdas lógicas.

Palabras clave: memristor, RRAM, lógica multinivel, memorias no volátiles.

Abstract: over the past 20 years, there has been an emerging growth in memristor-based technologies to replace or complement the extensively used flash and RAM memories. In a memristor or resistive memory, the resistance is a variable that can be modified by appropriate electrical pulses. These devices also offer the possibility of designing multilevel logic functions with properties, integration, consumption and speed far superior to conventional binary logic circuits. This project aims to achieve two main objectives: to perform a documentary update of the state of the art of memristor-based multilevel logic; and to design and simulate multilevel combinational logic cells based on real devices fabricated and measured by the Materials and Electronic Devices Characterization Group. The already available experimental data will be used for the simulation of the logic cells.

Keywords: memristor, RRAM, multilevel logic, non-volatile memories.

Introducción

El memristor, o resistencia con memoria, es un dispositivo capaz de variar su resistencia en función de la corriente que lo ha atravesado. Fue predicho en 1971 por L. O. Chua para completar los cuatro elementos básicos pasivos eléctricos: resistencia, condensador, inductor y memristor. Sin embargo, no fue hasta 2008 cuando el equipo de R. S. Williams construyó el primer memristor. Desde entonces las investigaciones se han centrado en la mejora y desarrollo de los materiales y estructuras empleadas.

El primer capítulo presenta la creación del memristor junto con sus características típicas. Se hace también un repaso de los materiales y aplicaciones actuales, destacando la computación neuromórfica gracias a la no volatilidad de estos dispositivos.

El segundo capítulo se centra en el estudio de las familias lógicas, empezando por la lógica binaria y ampliándola a familias multinivel. Actualmente la tecnología CMOS, basada en transistores mosfet de tipo p y n, es la más utilizada en los sistemas lógicos. Como alternativa, se presentan dos familias lógicas basadas en memristores. La primera de ellas, la familia binaria MAGIC, emplea dos memristores distintos para la entrada y la salida de las puertas lógicas. La segunda familia combina la tecnología CMOS junto con los memristores para conseguir una lógica ternaria de alta densidad. En ambas familias se estudia y analiza el funcionamiento de cada una de las puertas lógicas básicas: OR, AND, NOT, NOR y NAND.

En el tercer capítulo se emplean datos experimentales del GCME de la Universidad de Valladolid para simular las puertas ternarias TOR y TAND de la segunda familia estudiada en el capítulo 2. Para ello, se han desarrollado programas en MATLAB que reproducen el funcionamiento de las puertas lógicas OR y AND basadas en el memristor experimental. Los resultados obtenidos con el memristor real se comparan con el caso ideal en el que las transiciones entre los estados de SET y RESET son abruptas.

Capítulo 1

Fundamentos y descripción del memristor

1.1. Descubrimiento y creación del primer memristor

En el año 1971 el ingeniero eléctrico Leon Ong Chua publicó su famoso artículo *"Memristor-The Missing Circuit Element"* (1). En este paper Chua predice la existencia de un nuevo elemento electrónico pasivo y detalla sus características. Las cuatro variables fundamentales de un circuito eléctrico son la corriente eléctrica I, el voltaje V, la carga eléctrica Q y el flujo magnético Φ. Existen por tanto seis combinaciones posibles que relacionen las magnitudes anteriores. Sin embargo, únicamente eran conocidas cinco de ellas:

$$\begin{aligned} dQ &= Idt \ , \\ d\Phi &= Vdt \ , \\ dV &= RdI \ , \\ dQ &= CdV \ , \\ d\Phi &= LdI \ . \end{aligned}$$

Las tres últimas ecuaciones permiten definir los tres elementos básicos pasivos de un circuito eléctrico, es decir, la resistencia, el condensador y el inductor. Esta ausencia de la sexta ecuación que relacionara el flujo magnético Φ y la carga Q fue la que llevó a Chua a postular la existencia del cuarto elemento básico: el memristor. El nombre es una abreviatura de la expresión *memory resistor*. El motivo de esto es que el memristor se comporta como una especie de resistencia no lineal con memoria. En consecuencia, un memristor está caracterizado por la relación $d\Phi = MdQ$ de forma que la diferencia de potencial entre sus extremos viene dada por

$$V(t) = M(Q(t))I(t) \ ,$$

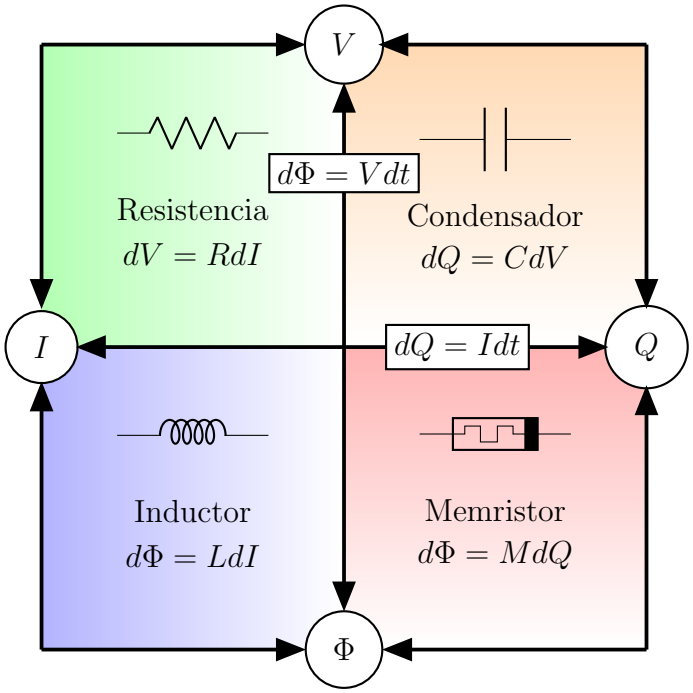


Figura 1.1: Resumen de los cuatro elementos pasivos básicos de un circuito eléctrico y sus relaciones.

donde $M(Q) = \frac{d\Phi(Q)}{dQ}$ se denomina memristancia incremental (pues tiene las mismas unidades que la resistencia). El valor de M en un instante de tiempo t_0 depende de la carga en ese instante, y ésta a su vez de la integral de la corriente desde $t = -\infty$ hasta $t = t_0$. Por tanto, el memristor se comporta como una resistencia ordinaria en un instante t_0 pero su resistencia depende de lo que ha ocurrido en el pasado, tanto de la magnitud como del sentido de la corriente. Esta característica de memoria es lo que hace interesante al memristor de cara a sus aplicaciones.

Tuvieron que pasar 37 años desde la publicación de Chua para que el primer memristor viera la luz en 2008. Fue creado en los *HP Labs* por R. Stanley Williams y su equipo (2). Sin embargo, el objetivo de Stanley inicialmente no era la creación del memristor, de hecho, ni tan siquiera conocía el artículo de Chua. Williams estaba centrado en el diseño de conmutadores de barras cruzadas de escala nanométrica. Un conmutador de barras cruzadas es un conjunto de cables perpendiculares entre sí. Los cables se encuentran conectados en las intersecciones por medio de interruptores. De esta forma un conmutador de barras cruzadas constituye una memoria, donde los interruptores abiertos representan un 0 y los cerrados un 1. Lo que buscaban Williams y su equipo era conseguir conectar y desconectar estos interruptores mediante la aplicación de voltajes en los extremos de los cables.

Para ello el dispositivo que fabricaron tenía un estructura tipo sándwich. Estaba formado por dos electrodos de platino que constituían las capas externas. Sobre el electrodo inferior se había crecido dióxido de platino, un excelente conductor. Encima del dióxido de platino se depositó una fina película de una molécula de espesor (hecha *de moléculas especialmente diseñadas para la conmutación*). Entre esta película y el electrodo superior se montó una capa de unos 2-3 nm de titanio. Después de numerosas pruebas aplicando voltajes al dispositivo anterior observaron algo sorprendente. El dióxido de platino se había convertido en platino puro y el titanio en dióxido de titanio. Además, la capa de dióxido de titanio estaba dividida en dos subcapas: la capa inferior era dióxido de titanio en una proporción estequiométrica exacta de 2 a 1, es decir, TiO_2; mientras que la capa superior se trataba de dióxido de platino con una deficiencia muy pequeña de oxígeno del 2 % al 3 %, TiO_{2-x} con x próximo a 0,05.

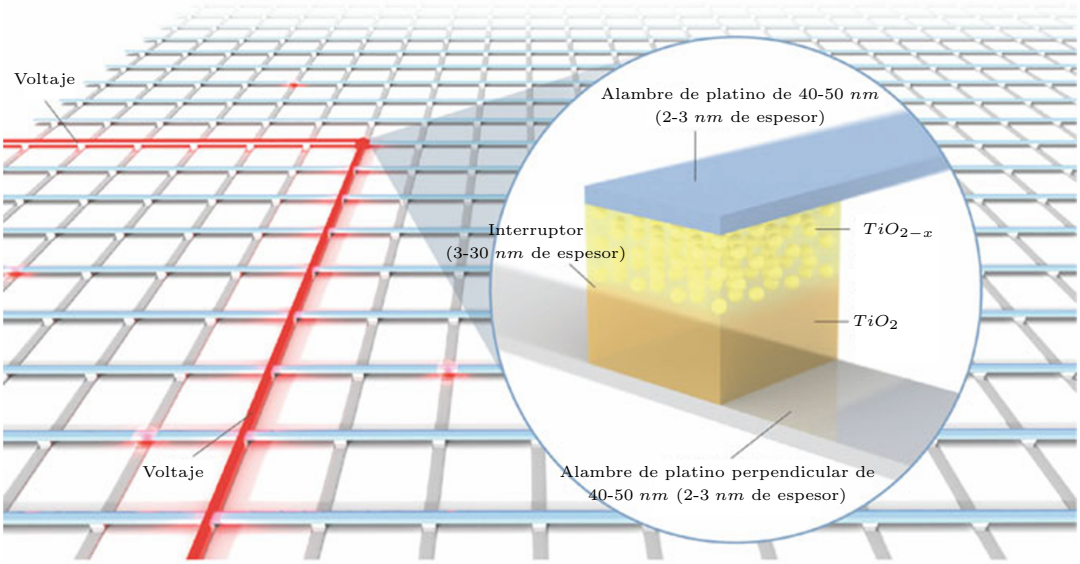


Figura 1.2: Conmutador de barras cruzadas con memristores (3).

El dispositivo anterior se comportaba como un interruptor con una relación de resistencias *off-on* mayor de 1000. Este comportamiento como interruptor se debía a la bicapa de los dióxidos de titanio. El TiO_2 es aislante eléctrico (en realidad es un semiconductor pero con un *gap* grande), pero el TiO_{2-x} es un conductor debido a que las vacantes de oxígeno actúan como donores de electrones y por tanto como cargas positivas. Estas vacantes se pueden visualizar como burbujas. Cuando al electrodo superior se le somete a un voltaje positivo las vacantes de oxígeno (burbujas cargadas positivamente) se ven desplazadas hacia la parte inferior. Esto hace que la capa de TiO_2 se convierta en TiO_{2-x} y por tanto la conductividad aumenta. Si, en cambio, se aplica una tensión negativa al electrodo superior en-

tonces la capa de TiO_{2-x} disminuye y aumenta la resistencia. La película de una molécula de espesor no intervenía en el comportamiento como interruptor pero se encargaba de controlar el flujo de oxígeno del dióxido de platino al titanio para crear las capas uniformes de TiO_2 y TiO_{2-x}.

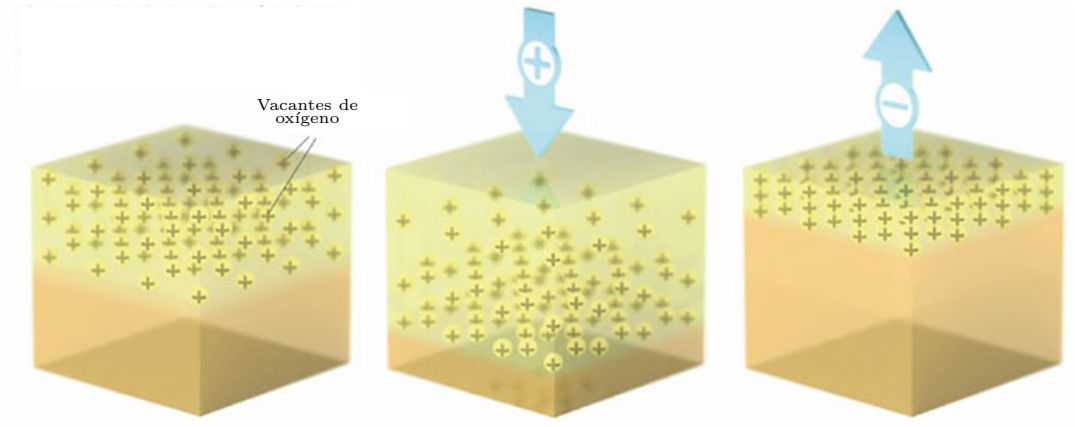


Figura 1.3: Funcionamiento de la memristancia (3).

El interés del memristor reside en que su resistencia no cambia cuando se deja de aplicar tensión en los electrodos. Esto se debe a que las vacantes de oxígeno no vuelven a sus posiciones iniciales y se quedan inmóviles hasta que se vuelve a aplicar un voltaje positivo o negativo que debe superar un valor crítico. En definitiva, los memristores se pueden emplear como memorias no volátiles almacenando valores de resistencias. Esto los hace ideales para el desarrollo de redes neuronales y la computación neuromórfica.

1.2. Materiales, estructura y aplicaciones del memristor

Los datos y programas en la arquitectura de Von Neumann se almacenan en la memoria. Cuando se requiere realizar cálculos y operaciones se envía esta información por medio de los *buses* o canales al procesador. El procesador está formado principalmente por la unidad de control que se encarga de procesar y ejecutar las instrucciones; y la unidad aritmético lógica que se ocupa de realizar los cálculos y operaciones. En su conjunto, el ordenador se comunica con el exterior mediante los dispositivos de entrada y salida (figura 1.4). Sin embargo, este diseño de los

ordenadores tiene un problema: para ejecutar cualquier programa primero se necesita acceder a la memoria y luego enviarlo al procesador. Pese a que la tecnología ha permitido crear procesadores y memorias más rápidas de acuerdo a la *ley de Moore*, la llegada del análisis de datos masivo y la inteligencia artificial ha hecho que el tiempo de acceso a la memoria sea mucho mayor que el tiempo de cálculo de la CPU, provocando que el procesador se encuentre inactivo hasta que le llegan los datos. Esto es lo que se conoce como el cuello de botella de Von Neumann. Otro gran problema que tiene la arquitectura de Von Neumann es el enorme consumo energético empleado para la transferencia de información entre memoria y procesador. De hecho, se estima que si el almacenamiento de datos continúa aumentando al ritmo actual, la energía consumida para hacer operaciones binarias empleando tecnología CMOS superará los 10^{27} J en 2040, mayor que la energía producida globalmente.

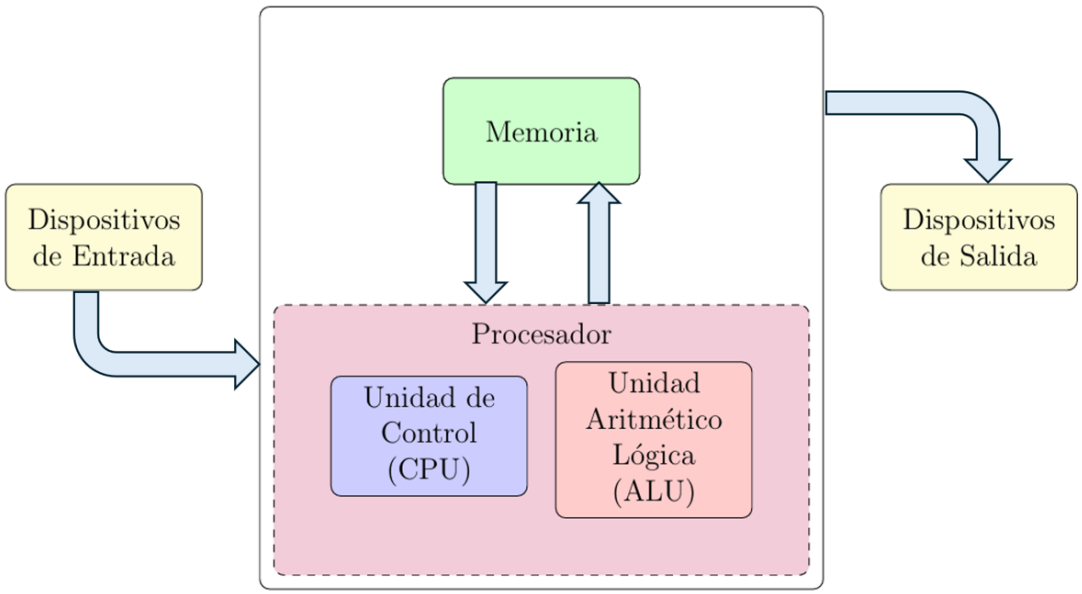


Figura 1.4: Arquitectura de Von Neumann

Una alternativa a la arquitectura de Von Neumann es la computación neuromórfica. Este diseño se basa en cómo funciona nuestro cerebro: el procesamiento de datos y la memoria se encuentran en el mismo lugar, las neuronas. Esto no solo elimina el problema de la transferencia de datos y la inactividad de la CPU sino también la eficiencia energética. Las neuronas se comunican unas con otras mediante neurotransmisores en la sinapsis. El estado de una neurona se modifica controlando el flujo de iones que entran y salen de ella. Por tanto, las neuronas se comportan en última instancia como dispositivos eléctricos. Sin embargo, hasta la llegada del memristor no existía ningún dispositivo capaz de comportarse como

una sinapsis artificial. La no volatilidad del memristor junto con la fabricación masiva y miniaturización hacen del memristor el componente idóneo para este tipo de arquitecturas.

Un memristor posee dos estados bien diferenciados que vienen dados por su resistencia: estado *on* o de baja resistencia (LRS *low resistance state*) y estado *off* o de alta resistencia (HRS *high resistance state*). La característica I-V de un memristor general se representa en la figura 1.5. Variando la tensión aplicada al memristor podemos cambiar de un estado a otro. El proceso $HRS \rightarrow LRS$ se denomina *SET process* (proceso de formación) y a la transición $LRS \rightarrow HRS$ se denomina *RESET process* (proceso de destrucción). Es importante observar que esta curva siempre pasa por el origen y se estrecha al aumentar la frecuencia.

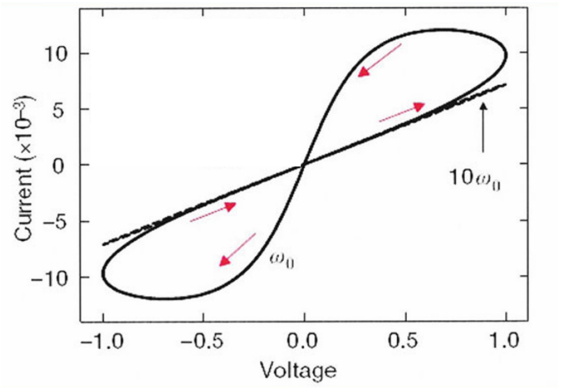


Figura 1.5: Dependencia del ciclo con la frecuencia.

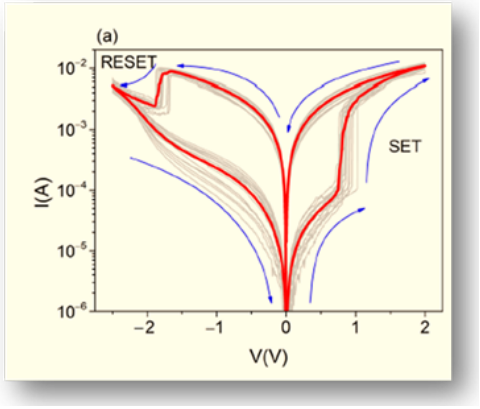


Figura 1.6: Curva $|I|$-V del Grupo de Caracterización de Materiales y Dispositivos Electrónicos (GCME) de la UVa.

En los circuitos electrónicos el memristor se representa con el símbolo de la figura 1.7. Se toma el convenio de que si la corriente entra de izquierda a derecha la resistencia disminuye ($R \downarrow$) y si entra de derecha a izquierda la resistencia aumenta ($R \uparrow$).

Figura 1.7: Símbolo del memristor.

1.2.1. Materiales

Un memristor típico tiene una estructura tipo sándwich metal-aislante-metal (estructura MIM, metal-insulant-metal). Las características del memristor como los voltajes de SET y $RESET$, la proporción off/on de resistencias, la velocidad de conmutación, consumo energético ..., dependen de los materiales que lo componen.

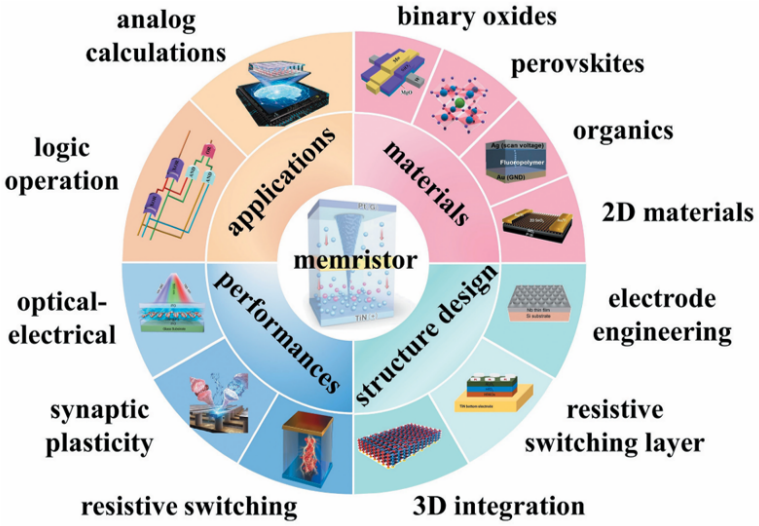


Figura 1.8: Memristores: materiales, aplicaciones, estructura y funcionamiento (4).

Electrodos

La función del electrodo no solo es la de transportar la corriente, sino también intervenir en el proceso de cambio de la resistencia del memristor. Normalmente se emplean metales convencionales, metales nobles, aleaciones, nitruros, materiales basados en carbono como el grafeno y nanotubos de carbono, etc. Los materiales empleados como electrodos se clasifican de acuerdo a su importancia en el proceso de conmutación resistiva. Los metales inertes como el Au o Pt se encargan únicamente del transporte de la corriente eléctrica. Otro de tipo de electrodos son aquellos que intervienen en la formación de los filamentos conductores en los memristores basados en la migración de cationes como el Cu y la Ag. Recientemente se ha empezado a emplear materiales como ITO, FTO (óxido con estaño dopado) o grafeno para memristores transparentes y flexibles. Otro grupo importante de electrodos es el formado por aleaciones lo que permite obtener memristores más estables.

Materiales memristivos

Los materiales memristivos se clasifican en orgánicos e inorgánicos. Los materiales

orgánicos tienen mayor flexibilidad, menor costo y son más fáciles de fabricar. En cambio los materiales inorgánicos (óxidos, perovskitas, materiales 2D) son más estables, rápidos y duraderos.

- Óxidos binarios: poseen gran estabilidad, son baratos y fáciles de fabricar y además son compatibles con la tecnología CMOS. Los más empleados son TiO_x, SiO_x, AlO_x, TaO_x y HfO_x. De hecho, se han fabricado memrsitores de HfO_x y TaO_x que consiguen velocidades de conmutación inferiores a los nanosegundos, tiempos de vida mayores de 10^{10} ciclos y una relación de resistencias *off/on*$> 10^{10}$. Sin embargo, tienen el inconveniente de que consumen una gran cantidad de potencia comparada con memristores de otros materiales.

- Perovskitas: son materiales con una estructura del tipo ABX_3 como $MAPbI_3$, $LaFeO_3$ o $CH_3NH_3PbClXI_3$. Este tipo de materiales son idóneos para optoelectrónica (células solares, fotodetectores, diodos emisores de luz, etc) debido a sus altos coeficientes de absorción óptica, altos rendimientos de fotoluminiscencia y grandes longitudes de difusión. Su principal punto débil es que son incompatibles con la tecnología CMOS. Además, son materiales frágiles y fácilmente dañables por las inclemencias del tiempo, lo que limita su uso en aplicaciones exteriores.

- Materiales 2D: son materiales con buenas propiedades eléctricas y que poseen características novedosas como la transparencia y la flexibilidad. Por otro lado, debido a su espesor atómico son ideales para dispositivos de alta densidad electrónica. Por contra, los memristores basados en materiales 2D tienen una vida útil pequeña del orden de las decenas de ciclos, un ratio *off/on*< 10 y baja estabilidad de operación. Algunos de estos problemas se pueden solventar superponiendo capas de materiales 2D distintos por medio de fuerzas de Van der Waals. Las propiedades eléctricas, magnéticas y ópticas se pueden mejorar mediante el dopado. No obstante, las técnicas de dopado habituales como la difusión térmica y la implantación iónica han de ser modificadas pues dañan a los finas capas de materiales 2D.

- Materiales orgánicos: sus principales características son el bajo coste, la flexibilidad y la fácil fabricación a gran escala. Los más empleados son los polímeros orgánicos, especialmente en la producción de memristores flexibles. Por contra, los memristores orgánicos son muy delicados y tienen vidas de unas pocas decenas de ciclos. Existen memristores fabricados con materiales exóticos como moléculas de ADN e incluso de proteínas del huevo.

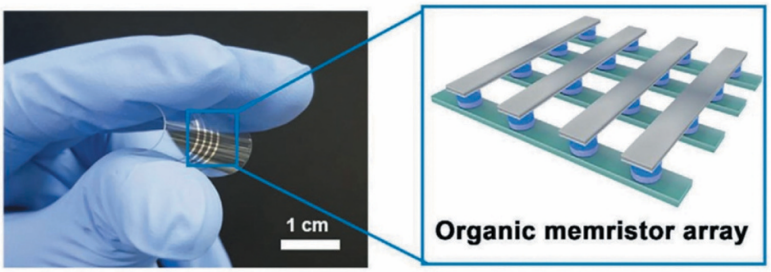


Figura 1.9: Memristor fabricado con materiales orgánicos (4).

1.2.2. Diseño estructural

Para mejorar el funcionamiento y el rendimiento de los memristores existen diferentes técnicas. Una de ellas se centra en la modificación de los electrodos para potenciar determinadas cualidades. Esto es lo que se conoce como ingeniería de electrodos. Como ejemplo de los métodos empleados, en la figura 1.10 se muestra un memristor con electrodos de niobio que han sido modificados para formar una malla de puntas metálicas que se encajan en el material memristivo. De esta forma, el campo eléctrico es mayor debido al efecto punta y se consigue disminuir los voltajes de SET y RESET.

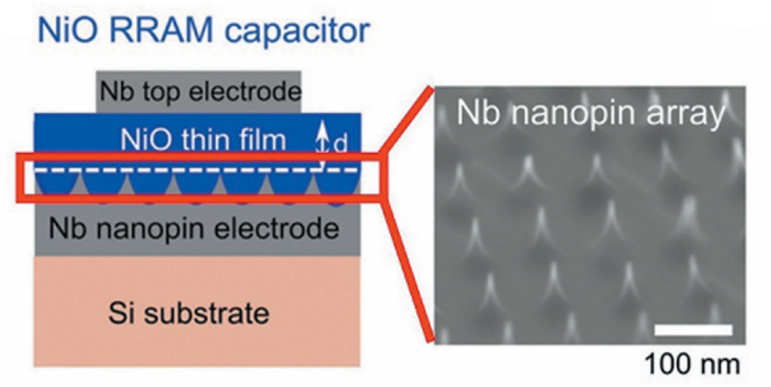


Figura 1.10: Memristor con ingeniería de electrodos (4).

Otros memrsitores cuentan con una bicapa de material memrsitor que ha sido agujereado permitiendo así la formación de canales permanentes con los que se han llegado a conseguir velocidades de operación inferiores a los nanosegundos y vidas superiores a 10^{10} ciclos.
Una de las ventajas principales de los memristores de cara a la fabricación de memorias no volátiles es su gran escalabilidad. En la integración 3D de memristores, estos se sitúan en las intersecciones de un array de cables. Puesto que la corriente

puede llegar por ambos extremos del memristor, puede haber problemas a la hora de leer la información almacenada. Una forma de solucionar este problema es integrar el memristor en una célula conectada con un elemento rectificador como un diodo o un transistor.

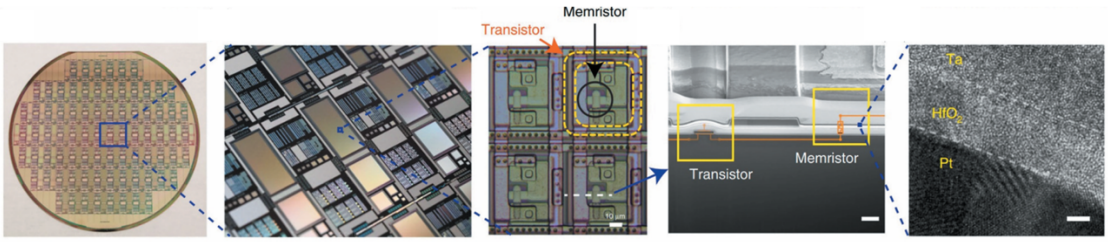


Figura 1.11: Integración 3D de una célula memristor-transistor en un array (4).

1.2.3. Aplicaciones

La computación en memoria basada en memristores se divide en dos grandes grupos: memristores abruptos para realizar operaciones lógicas; y memristores graduales para realizar cálculos analógicos. La tecnología CMOS convencional permite realizar operaciones lógicas. Sin embargo, no es capaz de mantener el estado lógico una vez desconectada la fuente de potencia. Los arrays de memristores pueden solucionar este problema a la vez que aumentan la eficiencia computacional gracias a que pueden desarrollar computación paralela.

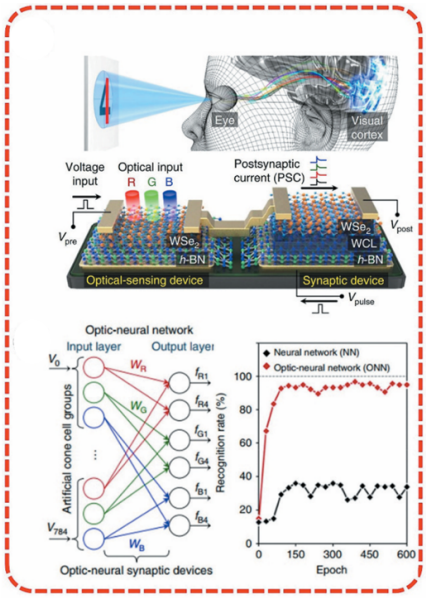


Figura 1.12: Dispositivo neuro-óptico fabricado con memristores de BN/WSe_2 (5).

Los memristores se han convertido en una candidato idóneo para la computación neuromórfica. Para simular la sinapsis, el memristor debe ser capaz de regular de forma gradual su resistencia en lugar de presentar un cambio abrupto. Esto se puede conseguir mediante técnicas como el dopado de la capa de material memristivo, diseño de estructuras bicapa, ingeniería de electrodos, etc. Se han conseguido fabricar memristores basados en SiC cuya conductancia se puede modular gradualmente. También se han obtenido resultados satisfactorios en funciones sinápticas como el aprendizaje y la pérdida de memoria. Se han empleado del mismo modo memristores graduales para simular la visión humana en procesos como reconocimiento de colores y patrones, obteniendo resultados satisfactorios en más del 90 % de los casos (5). Además, algunos investigadores han conseguido mejorar el rendimientos de memristores basándose en efectos piezoeléctricos y termoeléctricos.

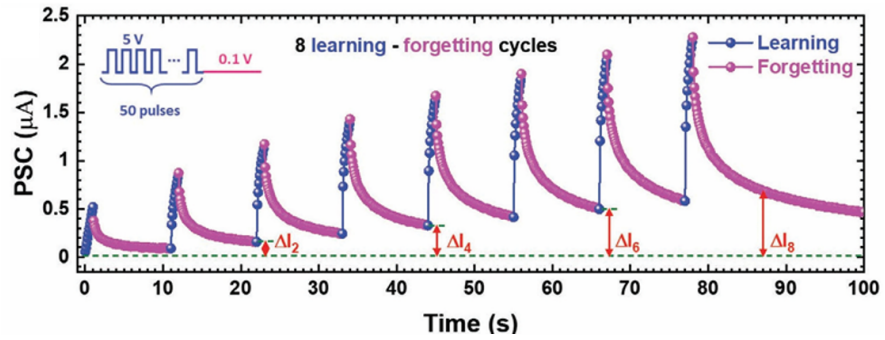


Figura 1.13: Comportamiento del memristor $Cu/SiC/W$ emulando procesos de aprendizaje/olvido (5).

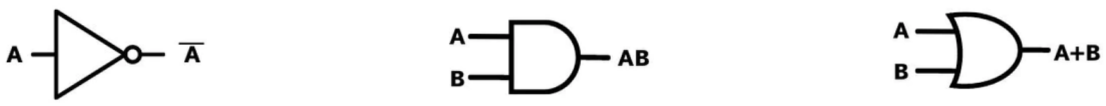
Por último, es importante mencionar que todavía hay mucho desconocimiento sobre los memristores. De hecho, los investigadores no se ponen de acuerdo en cómo es el mecanismo que describe la memristancia en algunos dispositivos. Además, exceptuando el HfO_x y TaO_x, la gran mayoría de materiales memristivos no son aún compatibles con la tecnología CMOS, limitando en gran medida sus aplicaciones futuras.

Capítulo 2

Familias lógicas multinivel basadas en memristores

2.1. Introducción

Los sistemas digitales convencionales emplean la lógica binaria, donde las variables y funciones únicamente toman dos valores: verdadero (valor '1' lógico) y falso (valor '0' lógico). Entre ambos valores debe existir un margen que permita la tolerancia de un cierto nivel de ruido. El valor que toma cualquier función lógica binaria a partir de sus dos entradas se puede determinar mediante un cálculo simbólico (Álgebra de Boole). De forma general, la definición de una función lógica viene dada por una tabla de verdad en la que se representa la salida de la función para cada pareja de valores a la entrada. En el ámbito de las ciencias de la computación y electrónica, a las funciones lógicas se las denomina a menudo puertas lógicas ya que modifican el camino que sigue la electricidad en el circuito. Las tres puertas lógicas elementales son la puerta NOT, la puerta AND y la puerta OR. Sus tablas de verdad se pueden ver en el cuadro 2.1. La puerta NOT se encarga de invertir la entrada, mientras que las puertas AND y OR toman el mínimo y el máximo de sus valores de entrada respectivamente. Combinando estas funciones básicas se puede construir cualquier puerta lógica por complicada que sea. Existen diferentes tipos de familias lógicas binarias en función de los dispositivos empleados: lógica DL (diodos y resistencias), DTL (diodos, transistores y resistencias)...



Puerta NOT Puerta AND Puerta OR

Figura 2.1: Simbología de las puertas lógicas.

Entrada	NOT
0	1
1	0

Entrada A	Entrada B	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Cuadro 2.1: Tablas de verdad de las puertas lógicas NOT, AND y OR.

Las puertas lógicas anteriores se pueden implementar para trabajar con variables discretas que tomen más de 2 valores. De esta forma surgen las distintas familias lógicas multinivel. Exceptuando aplicaciones como la codificación de datos y el almacenamiento de datos en memorias de alta densidad, la lógica multinivel apenas ha sido empleada. Esto se debe a que la lógica binaria es más simple, estable, fácil de implementar y se requiere una menor complejidad en el diseño del hardware. Además, la mayoría de dispositivos actuales emplean lógica binaria, por lo que cambiar a lógica multinivel supondría realizar una integración adecuada de ambos sistemas. Sin embargo, la investigación y el desarrollo de memorias no volátiles multinivel, como los memristores, han despertado un interés en la lógica de alta densidad, tanto digital como multinivel. Existen principalmente dos tipos de lógica binaria basada en memristores: la lógica de estados, cuya salida se almacena en el valor de la resistencia del memristor; y la lógica convencional, que guarda los estados lógicos en forma de voltajes. Familias lógicas como IMPLY (6) (*material implication logic*) y MAGIC (7) (*Memristor Aided loGIC*) son ejemplos de familias que emplean la resistencia para representar los estados lógicos. En cambio, familias como MRL (8) (*Memristor Ratioed Logic*) son similares a la lógica CMOS convencional, representando los estados por medio de voltajes.
La lógica ternaria convencional (es decir, la lógica cuyas variables y funciones pueden tomar tres valores distintos de voltajes) se divide en dos grandes grupos: la lógica equilibrada, en la que los estados están representados por $(-1, 0, +1)$; y la lógica no equilibrada. A su vez la lógica no equilibrada puede ser de dos tipos: positiva, que toma los valores $(0, 1, 2)$; y negativa, que toma los valores $(-2, -1, 0)$. En lógica ternaria no equilibrada existen tres puertas NOT distintas dependiendo del valor asignado al estado lógico intermedio. El cuadro 2.2 muestra la tabla de verdad de los 3 tipos de inversores: STI (*Simple Ternary Inverter*), PTI (*Positive Ternary Inverter*) y NTI (*Negative Ternary Inverter*).

Entrada	STI	PTI	NTI
0	2	2	2
1	1	2	0
2	0	0	0

Cuadro 2.2: Tabla de verdad de los inversores ternarios.

V_{in1}	V_{in2}	**TAND**	**TOR**	**TNAND**	**TNOR**	**TXOR**	**TXNOR**
0	0	0	0	2	2	0	2
0	1	0	1	2	1	1	1
0	2	0	2	2	0	2	0
1	0	0	1	2	1	1	1
1	1	1	1	1	1	1	1
1	2	1	2	1	0	1	1
2	0	0	2	2	0	2	0
2	1	1	2	1	0	1	1
2	2	2	2	0	0	0	2

Cuadro 2.3: Lógica ternaria no equilibrada positiva: tablas de verdad.

Las puertas TAND y TOR devuelven el valor mínimo y máximo respectivamente de su entrada. Por otro lado, las funciones TNAND y TNOR son la combinación de las puertas TAND y TOR seguidas de un inversor ternario, en particular se ha tomado un STI. La puerta TXOR se encarga de realizar la función lógica A **TXOR** $B = \overline{A}B + A\overline{B}$. Finalmente, la puerta TXNOR se trata de la inversión (simple, en nuestro caso) de la salida de la puerta TXOR.

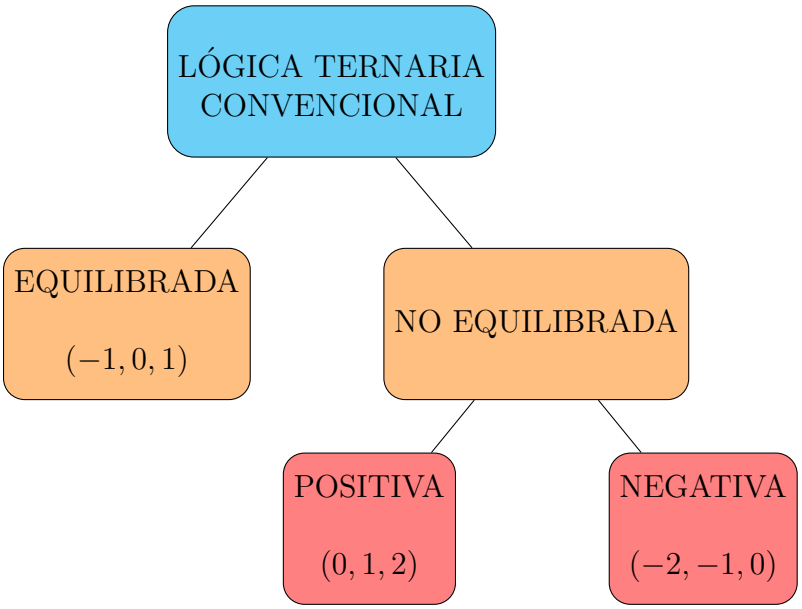


Figura 2.2: Resumen de los tipos de los tipos de familias lógicas ternarias convencionales.

Function	Design Constraints	Design Constraints – Multiple Inputs
NOR	$2V_{T,OFF} < V_0 < \min\left[\frac{R_{OFF}}{2R_{ON}} \cdot V_{T,OFF}, \lvert V_{T,ON}\rvert\right]$	$\frac{V_{T,OFF}}{R_{ON}} \cdot \left[R_{ON} + \left(\frac{R_{OFF}}{\chi - 1}\right) \parallel R_{ON}\right] < V_0 < \min\left[V_{T,OFF} \cdot \left(1 + \frac{R_{OFF}}{\chi R_{ON}}\right), \left(1 + \frac{\chi R_{ON}}{R_{OFF}}\right) \cdot \lvert V_{T,ON}\rvert\right]$
NAND	$3V_{T,OFF} < V_0 < \min\left[\lvert V_{T,ON}\rvert, \left(2 + \frac{R_{OFF}}{R_{ON}}\right) \cdot V_{T,OFF}\right]$	$(\chi + 1) \cdot V_{T,OFF} < V_0 < \min\left[\lvert V_{T,ON}\rvert \cdot \left(1 + \frac{\chi R_{ON}}{R_{OFF}}\right), \left(\chi + \frac{R_{OFF}}{R_{ON}}\right) \cdot V_{T,OFF}\right]$
OR	$V_{T,ON} < V_0 < 1.5V_{T,ON}$	$V_{T,ON} < V_0 < \left(1 + \frac{1}{\chi}\right) V_{T,ON}$
AND	$V_{T,ON} < V_0 < 2V_{T,ON}$	$\left(1 + \frac{R_{ON}}{R_{OFF}} \chi\right) V_{T,ON} < V_0 < \left(2 + \frac{R_{ON}}{R_{OFF}} (\chi - 1)\right) V_{T,ON}$
NOT	$2V_{T,OFF} < V_0 < \frac{R_{OFF}}{R_{ON}} \cdot \min\left(V_{T,OFF}, \lvert V_{T,ON}\rvert\right)$	-

Figura 2.3: Restricciones para V_0 en las puertas lógicas de la familia MAGIC (7).

2.2. Familia lógica binaria basada en memristores (MAGIC)

La familia MAGIC (Memristor-Aided Logic) emplea puertas lógicas en las que la entrada y salida están formadas por memristores distintos. De esta forma el input de la puerta son los estados iniciales de los memristores de entrada y el output de la puerta es el estado final del memristor de salida. El estado lógico en una puerta MAGIC se representa como una resistencia, de forma que R_{ON} (baja resistencia) se considera el '1' lógico y R_{OFF} (alta resistencia) se considera el '0' lógico.
El funcionamiento de una puerta MAGIC está formado por 2 etapas secuenciales:

1. En primer lugar se inicializa el memristor de salida a un estado lógico conocido.

2. Se aplica una tensión V_0 a través de la puerta lógica. Mientras se está aplicando esta tensión, la tensión en el memristor de salida depende de los estados lógicos de los memristores de entrada (y el de salida).

De esta forma, para combinaciones concretas a la entrada se consigue que el voltaje sea lo suficientemente alto como para cambiar el estado lógico del memristor de salida mientras que para otras combinaciones el estado del memristor de salida permanece en el estado inicializado en la etapa 1. Para la discusión posterior del funcionamiento de las puertas lógicas supondremos que todos los memristores empleados son iguales y tienen tensiones umbrales $V_{T,ON}$ y $V_{T,OFF}$. Además en cada puerta se tendrá unas condiciones para la tensión aplicada a la puerta V_0. Estas restricciones se reflejan en el cuadro 2.3.
La inicialización del memristor de salida se puede conseguir de varias formas. La más común es mediante la operación habitual de escritura como en una memoria.

2.2.1. Puertas AND y OR

Las puertas más básicas y sencillas son las puertas AND y OR. El montaje de ambas se muestra en la figura 2.6. El memristor de salida se conecta con la misma polaridad que los memristores de entrada y se inicializa su estado lógico a '0'.

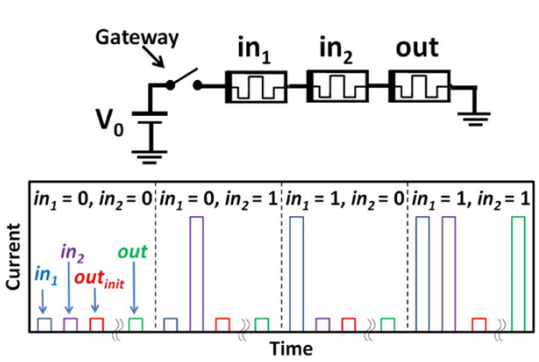


Figura 2.4: Puerta AND.

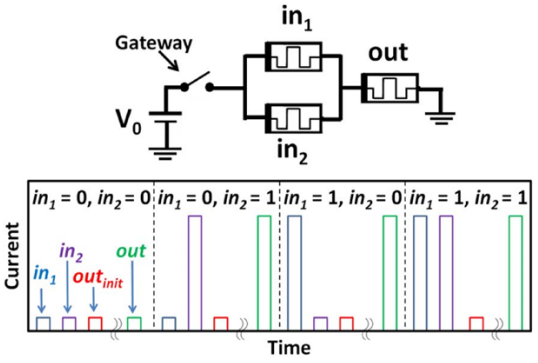


Figura 2.5: Puerta OR.

Figura 2.6: Funcionamiento de las puertas AND y OR de la familia lógica MAGIC (7).

Para la puerta AND:

- Si $in_1 = in_2 = 0$, al aplicar V_0 ocurre que $R_{M1} \rightarrow R_{ON}$ y los memristores M_2 y M_{OUT} mantienen el estado lógico inicial pues $V_{T,ON} < V_0 < 2V_{T,ON}$ y los memristores están en serie (pudiera ocurrir que M_2 conmutara y cambiara a R_{ON} si su inicialización no ha sido del todo exacta, pero en ningún caso el estado de M_{OUT} cambiaría gracias a los márgenes que hemos impuesto a V_0).
- Si $in_1 = 0$ y $in_2 = 1$, entonces $R_{M_1} \rightarrow R_{ON}$ pero $R_{M_{OUT}}$ permanece igual.
- Si $in_1 = 1$ y $in_2 = 0$, entonces se razona como en el caso anterior.
- Si $in_1 = in_2 = 1$, entonces ahora $R_{M_{OUT}} \rightarrow R_{ON}$.

Para la puerta OR:

- Si $in_1 = in_2 = 0$, al aplicar V_0 ocurre que $R_{M1}, R_{M2} \rightarrow R_{ON}$ pero $R_{M_{OUT}}$ permanecen igual que al inicio pues $V_{T,ON} < V_0 < 1{,}5V_{T,ON}$.
- Si $in_1 = 0$ y $in_2 = 1$, entonces la corriente circula por M_2 sin apenas caída de potencial lo que permite que $R_{M_{OUT}} \rightarrow R_{ON}$.
- Si $in_1 = 1$ y $in_2 = 0$, entonces se razona como en el caso anterior.
- Si $in_1 = in_2 = 1$, se tiene la misma situación que los dos casos previos.

2.2.2. Puerta NOT

La siguiente función básica en cualquier familia lógica es un inversor, es decir, la puerta NOT. En la imagen 2.7 se puede ver su estructura. Está formada por dos memristores conectados en serie y con polaridades opuestas. El memristor de salida se inicializa en ambas situaciones al estado '1'.

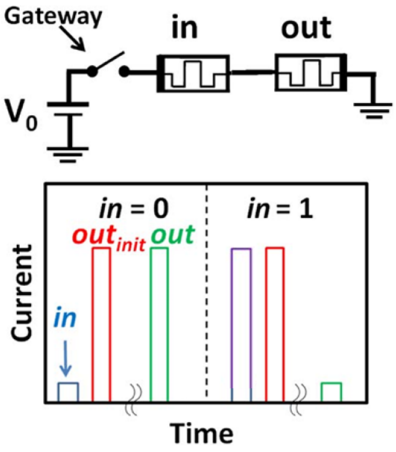


Figura 2.7: Funcionamiento de la puerta NOT de la familia lógica MAGIC (7).

Al aplicar la tensión V_0, el principio del divisor de tensión determina si cambia o no $R_{M_{OUT}}$.

- Si $in = 0$, entonces la diferencia de potencial aplicado a M_{OUT} es prácticamente 0 pues $R_{OFF} \gg R_{ON}$. En consecuencia, $R_{M_{OUT}}$ no cambia. Es importante observar que el memristor M_1 está sometido a una tensión relativamente alta, por lo que podría conmutar y cambiar al estado lógico '1'. Para prevenir esto y también su destrucción, es necesario establecer una cota superior para V_0, en concreto, $V_0 < \frac{R_{OFF}}{R_{ON}} min\{V_{T,OFF}, |V_{T,ON}|\}$.

- Si $in = 1$, entonces M_2 está sometido a una diferencia de potencial $\frac{V_0}{2} > V_{OFF}$ (por las restricciones impuestas a V_0), suficiente como para que M_{OUT} cambie al estado lógico '0'.

2.2.3. Puertas NAND y NOR

Por último las dos puertas lógicas que nos quedan por definir de la familia son la NAND y la NOR. En ambas puertas el memristor de salida se inicializa con el valor lógico '1'. En las figuras 2.8 y 2.9 se muestra el esquema de la puerta NAND para 2 y N entradas.

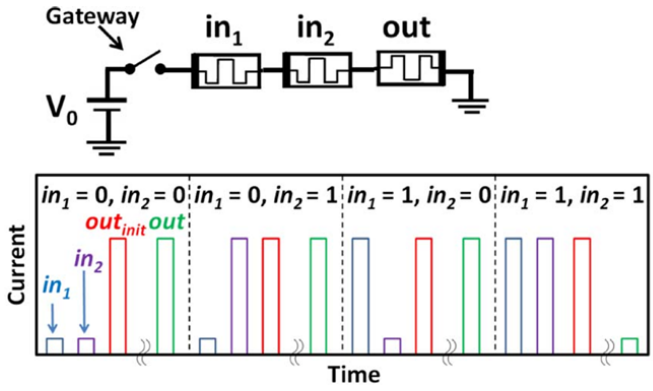


Figura 2.8: Puerta NAND con dos entradas (7).

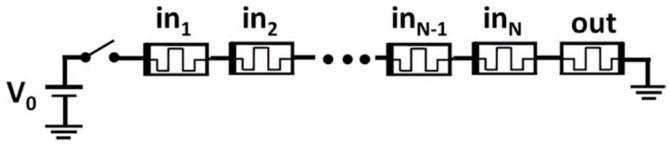


Figura 2.9: Puerta NAND con N entradas (7).

Para la puerta NAND:

- Si $in_1 = in_2 = 0$, al aplicar V_0 ocurre que M_1 y M_2 no cambian sus estados (pues $V_0 < |V_{T,ON}|$, evitando así la destrucción de la puerta) y como $V_0 < (2 + \frac{R_{OFF}}{R_{ON}})V_{T,OFF}$ se sigue cumpliendo que $R_{M_{OUT}} = R_{ON}$.

- Si $in_1 = 0$ y $in_2 = 1$, las mismas condiciones de antes nos aseguran que $R_{M_{OUT}}$ no cambia su estado inicial.

- Si $in_1 = 1$ y $in_2 = 0$, estamos en la situación anterior.

- Si $in_1 = in_2 = 1$, entonces ahora $R_{M_{OUT}} \to R_{OFF}$ ya que $V_0 > 3V_{T,OFF}$.

Una situación similar se tiene para la puerta NOR. El montaje es idéntico al de la puerta OR pero cambiando la polaridad del memristor de salida.

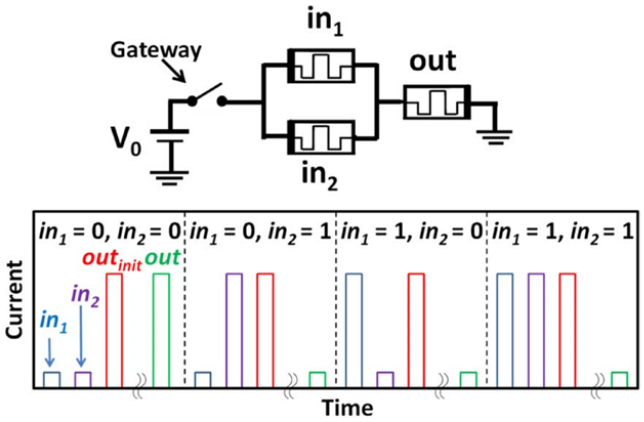


Figura 2.10: Puerta NOR con dos entradas.

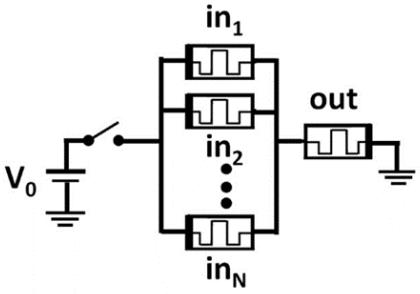


Figura 2.11: Puerta NOR con N entradas.

Figura 2.12: Funcionamiento de la puerta NOR de la familia lógica MAGIC (7).

Para la puerta NOR:

- Si $in_1 = in_2 = 0$, al aplicar V_0 los memristores M_1 y M_2 no cambian sus estados (ya que $V_0 < |V_{T,ON}|$, evitando así la destrucción de la puerta) y como $V_0 < \frac{R_{OFF}}{2R_{ON}} V_{T,OFF}$ se sigue cumpliendo que $R_{M_{OUT}} = R_{ON}$.
- Si $in_1 = 0$ y $in_2 = 1$, las mismas condiciones de antes junto con que $V_0 > 2V_{T,OFF}$ nos aseguran que $R_{M_{OUT}} \to R_{OFF}$.
- Si $in_1 = 1$ y $in_2 = 0$, estamos en la situación anterior.
- Si $in_1 = in_2 = 1$, entonces ahora $R_{M_{OUT}} \to R_{OFF}$ ya que $V_0 > 2V_{T,OFF}$.

En el caso de que el número de entradas de la puerta lógica sea χ, las restricciones para V_0 son similares a las empleadas para el caso de 2 inputs y se recogen también en el cuadro 2.3.

2.2.4. Conclusión

La familia lógica MAGIC presenta ventajas respecto de otras familias lógicas binarias basadas en memristores. Si se compara con la familia IMPLY, la familia MAGIC mejora la eficiencia energética y reduce el número de componentes electrónicos. Además, las puertas NOR y NOT MAGIC se pueden integrar fácilmente en conmutadores de barras cruzadas, permitiendo la computación en memoria.
Sin embargo, se trata de una familia que no se puede extender a la lógica multinivel ya que está basada en los 2 valores de resistencias R_{ON} y R_{OFF} del memristor. Requiere además la inicialización del estado de los memristores provocando un

aumento del tiempo de computación respecto de otras familias. Este retraso de tiempo se puede reducir aumentando la tensión de puerta V_0 como se ve en la gráfica 2.13. No obstante, hemos visto que es necesario imponer unas restricciones a V_0 por lo que tampoco se puede aumentar de forma indefinida.

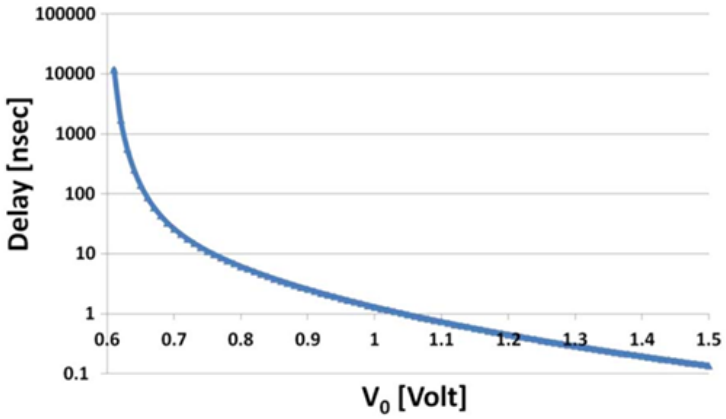


Figura 2.13: Tiempo de conmutación de la puerta NOR MAGIC en función de la tensión de puerta V_0 (7).

También es importante observar que el número de memristores empleados en cada puerta es igual al número de entradas más uno (el memristor dedicado a la salida). Este número es reducido por otras familias lógicas, como se verá en la próxima sección con la familia híbrida memristor-CMOS.

2.3. Familia lógica ternaria de alta densidad con tecnología memristor-CMOS

Se trata de una familia que emplea memristores para la construcción de las puertas lógicas TAND y TOR; y que combina la tecnología CMOS para el diseño de los inversores ternarios. En particular, es una familia ternaria convencional, no equilibrada y positiva. Los valores lógicos vienen dados por $(0, 1, 2) = (GND, V_{DD}/2, V_{DD})$, siendo GND la tensión de tierra y V_{DD} la tensión de alimentación.

2.3.1. Puertas TAND y TOR

Ambas puertas lógicas están compuestas por dos memristores conectados con polarizaciones opuestas. La entrada de las puertas son las tensiones V_{in1} y V_{in2} y la salida es la tensión del nodo común V_{out}. La representación de ambas puertas se muestra en la figura 2.14.

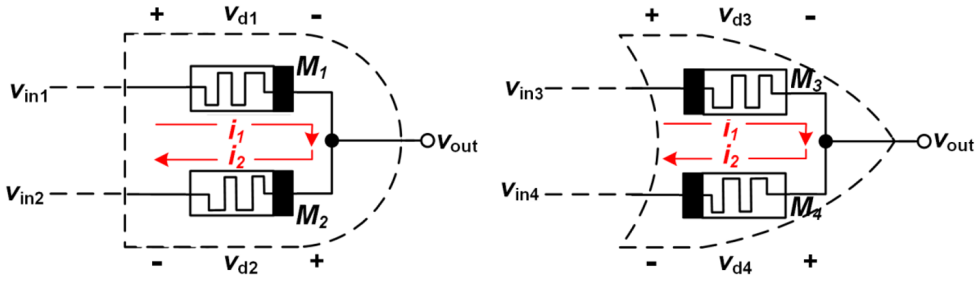


Figura 2.14: Puertas lógicas TAND y TOR con memristores (9).

Cuando V_{in1} y V_{in2} son distintos, la tensión de salida V_{out} viene determinada por el principio del divisor de tensión (o si se prefiere, por la ley de Ohm). Supongamos por ejemplo que $V_{in1} > V_{in2}$ en la puerta TAND. Entonces la corriente fluye desde V_{in1} hasta V_{in2} y la caída de tensión en cada uno de los memristores viene dada por

$$V_{d1} = R_{M1} i_1 \qquad V_{d2} = R_{M2} i_2 \ ,$$

siendo i_1, i_2, R_{M1}, R_{M2} las corrientes y resistencias de los memristores M_1 y M_2 respectivamente. Asumiendo que no hay corriente de fuga por V_{out}, es decir, $i_1 = i_2 = i$, se tiene que

$$i = \frac{V_{in1} - V_{in2}}{R_{M1} + R_{M2}} \ ,$$

$$V_{out} = V_{in1} - iR_{M1} = V_{in1} - \frac{V_{in1} - V_{in2}}{R_{M1} + R_{M2}} R_{M1} = \frac{V_{in1} R_{M2} + V_{in2} R_{M1}}{R_{M1} + R_{M2}} \ . \qquad (2.1)$$

Asumiendo que V_{d1} y V_{d2} son mayores que las tensiones umbrales para la conmutación de los memristores, el memristor M_1 cambia su estado a *off* ($R_{M1} \to R_{OFF}$) y el memristor M_2 pasa a estado *on* ($R_{M2} \to R_{ON}$). Además, si $R_{OFF} \gg R_{ON}$, de la ecuación (2.1) se sigue que

$$V_{out} = \frac{V_{in1} R_{ON} + V_{in2} R_{OFF}}{R_{OFF} + R_{ON}} \approx V_{in2} \ .$$

Observemos que la salida es aproximada pues existe una pequeña caída de potencial en el memristor M_2. Esta diferencia con el valor ideal será más pequeña cuanto mayor sea el cociente R_{OFF}/R_{ON}, y será mayor cuanto más entradas tengamos en la puerta. Esta disminución de la tensión también hay que tenerla en cuenta a la hora de combinar distintas puertas lógicas y puede ser necesario incluir nuevas fuentes de tensión entre ellas.
Cuando $V_{in1} = V_{in2}$ entonces $i = 0$ y $V_{out} = V_{in1} = V_{in2}$.

De forma similar para la puerta TOR se tiene que

$$V_{out} = \frac{V_{in4}R_{M3} + V_{in3}R_{M4}}{R_{M3} + R_{M4}} = \frac{V_{in4}R_{ON} + V_{in3}R_{OFF}}{R_{ON} + R_{OFF}} \approx V_{in3} \ ,$$

si $V_{in3} > V_{in4}$. Finalmente si $V_{in3} = V_{in4}$ entonces $i = 0$ y $V_{out} = V_{in3} = V_{in4}$. En el cuadro 2.4 se resumen las entradas y salidas de las puertas TAND y TOR construidas. Observemos que es consistente con la tabla de verdad del cuadro 2.3.

Entrada	**TAND**	**TOR**
$V_{in1} = V_{in2}$	$V_{out} = V_{in1,in2}$	$V_{out} = V_{in1,in2}$
$V_{in1} > V_{in2}$	$V_{out} \approx V_{in2}$	$V_{out} \approx V_{in1}$
$V_{in1} < V_{in2}$	$V_{out} \approx V_{in1}$	$V_{out} \approx V_{in2}$

Cuadro 2.4: Resumen puertas TAND y TOR.

Antes de pasar a las siguientes puertas lógicas es importante señalar que se pueden construir las funciones TMAX y TMIN de forma inmediata a partir de las puertas TAND y TOR anteriores. Puesto que las puertas TAND y TOR dan como valor el mínimo y máximo de sus dos entradas, se puede aumentar el número de entradas para obtener las puertas TMAX y TMIN. El montaje de estas puertas se muestra en la figura 2.15.

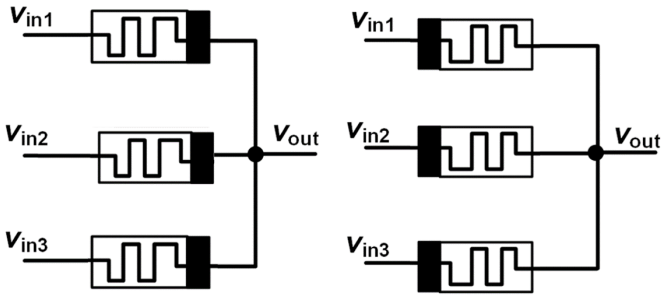


Figura 2.15: Puertas TMAX y TMIN (9).

En la figura 2.18 se muestra las gráficas resultantes de la simulación de las puertas anteriores, comprobando su correspondencia con la tabla 2.3.

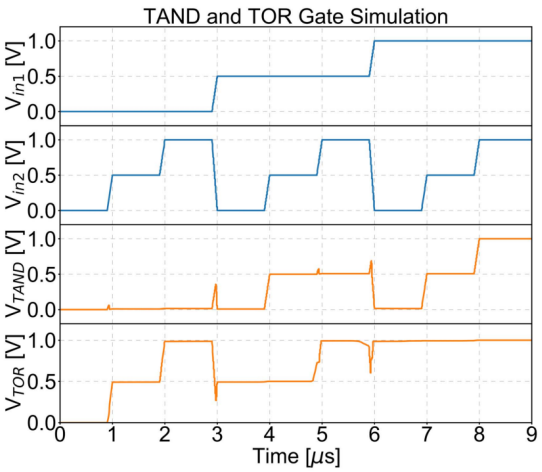


Figura 2.16: Puertas TAND y TOR.

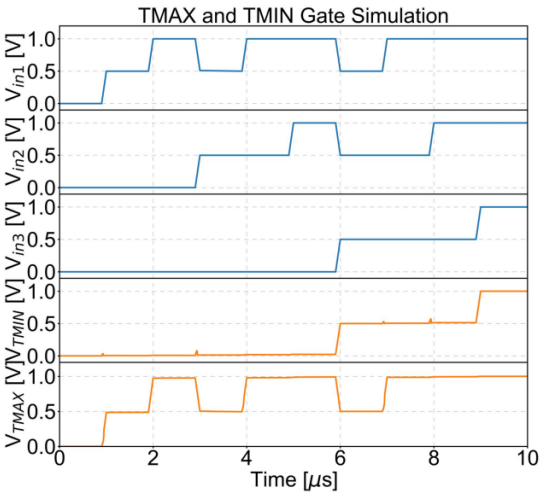


Figura 2.17: Puertas TMAX y TMIN.

Figura 2.18: Simulación con SPICE de las puertas TAND, TOR, TMAX y TMIN empleando el modelo de Knowm del memristor (9).

2.3.2. Puerta TNOT

Dado que existen tres inversores en lógica ternaria no equilibrada vamos a construir todos ellos.

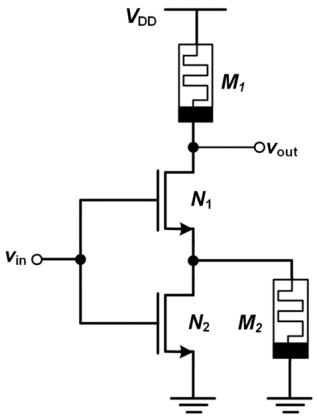
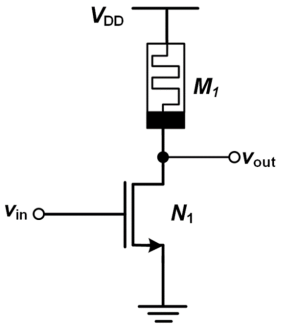


Figura 2.19: Puerta STI.



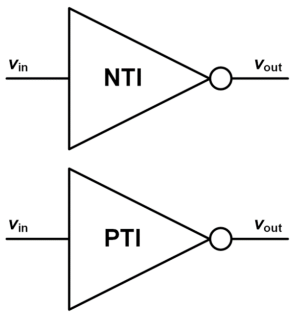


Figura 2.20: Puertas NTI y PTI.

Figura 2.21: Los tres tipos de inversores en lógica ternaria no equilibrada (9).

Puerta lógica STI

La configuración de esta puerta se muestra en la figura 2.19. Está formada por dos

transistores mosfet de acumulación de canal n y distintos. Sus tensiones umbrales deben cumplir que $V_{TH1} < V_{DD}/2 < V_{TH2}$. Los memristores M_1 y M_2 son iguales.

- Input lógico '0': si la entrada $V_{in} = GND$ entonces ambos transistores N_1 y N_2 están en corte. En consecuencia, $V_{out} = V_{DD}$. Si a la salida se coloca una resistencia de carga entonces $R_{M1} \to R_{OFF}$, consiguiendo una baja disipación estática de potencia pues la corriente que circula es muy pequeña.

- Input lógico '1': si $V_{in} = V_{DD}/2$ entonces N_2 está en corte y se tiene un divisor de tensión con las resistencias de ambos memristores iguales a R_{OFF}. Por tanto, $V_{out} = V_{DD}/2$.

- Input lógico '2': poniendo $V_{in} = V_{DD}$ ambos transistores N_1 y N_2 están en conducción. Por lo tanto, la salida está cortocircuitada a tierra a través de N_1 y N_2 haciendo que $V_{out} = GND$.

Puertas lógicas PTI y NTI

Ambas puertas tienen la misma estructura y únicamente se diferencian en la tensión umbral del transistor mosfet de acumulación de canal n. En el caso de la puerta NTI la tensión umbral debe cumplir que $V_{TH} < V_{DD}/2$, mientras que en la PTI debe ser $V_{TH} > V_{DD}/2$. El montaje de las puertas se muestra en la figura 2.20.

- Input lógico '0': para $V_{in} = GND$ el transistor se encuentra en corte en ambos casos (PTI y NTI) por lo que $V_{out} = V_{DD}$

- Input lógico '1': si $V_{in} = V_{DD}/2$ entonces N_1 estará en conducción para NTI y en corte para PTI. En consecuencia, $V_{out,NTI} = GND$ y $V_{out,PTI} = V_{DD}$. Para conseguir márgenes de ruidos óptimos se toma $V_{TH,NTI} = V_{DD}/4$ y $V_{TH,PTI} = 3V_{DD}/4$.

- Input lógico '2': poniendo $V_{in} = V_{DD}$ el transistor N_1 está en conducción en ambos inversores, haciendo que $V_{out} = GND$.

En la figura 2.22 se muestran los resultados de las simulaciones para los tres tipos de inversores lógicos ternarios. Observemos que los resultados están en concordancia con las tablas de verdad.

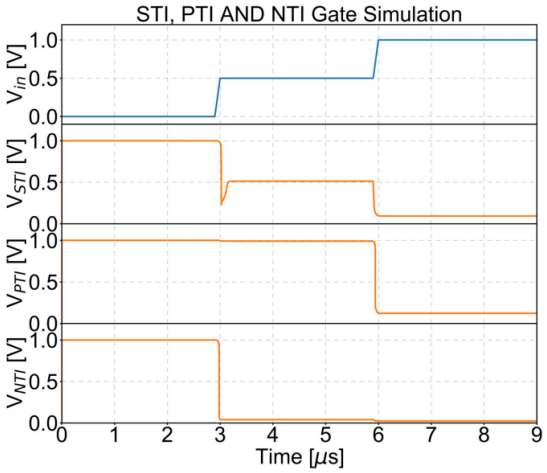


Figura 2.22: Simulación SPICE de las puertas STI, PTI y NTI (9).

2.3.3. Puertas TNAND y TNOR

Para construir las puertas TNAND y TNOR basta con combinar las puertas TAND y TOR con las TI. En la figura 2.23 se muestra el esquema de ambas donde a la salida de las puertas TAND y TOR se ha colocado una puerta STI. Es importante notar que la impedancia de salida de las puertas TAND y TOR no presenta ningún problema en la práctica ya que la señal de entrada al STI no se ve afectada debido a las grandes impedancias de entrada en los transistores mosfet.

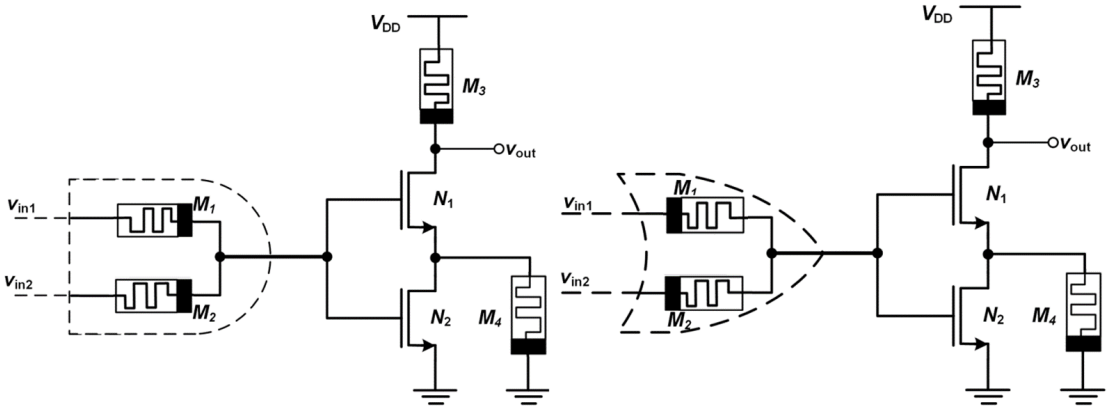


Figura 2.23: Montaje de las puertas TNAND y TNOR con la tecnología memristor-CMOS (9).

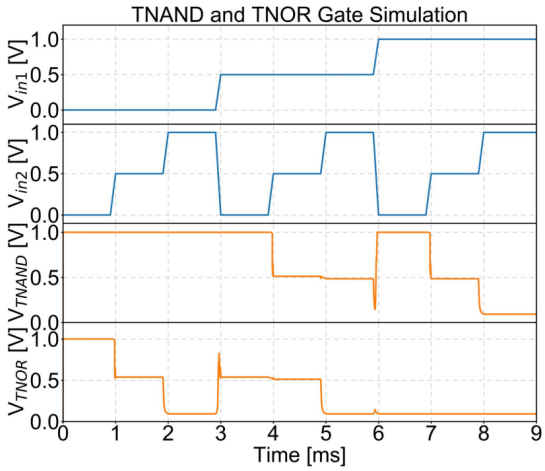


Figura 2.24: Simulación SPICE de las puertas TNAND y TNOR (9).

2.3.4. Puertas TXOR y TXNOR

En la figura 2.25 se muestran los mapas de Karnaugh de las puertas TXOR y TXNOR. Los mapas de Karnaugh son una generalización de las tablas de verdad, muy útiles para hacer cálculos complejos de funciones lógicas de forma sencilla y rápida. Observemos que

$$A \,\mathbf{TXOR}\, B = \overline{A}B + A\overline{B} = (A+B)\cdot(\overline{AB})\ ,$$

$$A \,\mathbf{TXNOR}\, B = AB + \overline{A}\,\overline{B} = \overline{(A+B)\cdot(\overline{AB})}\ .$$

A \ B	0	1	2
0	0	1	2
1	1	1	1
2	2	1	0

A \ B	0	1	2
0	2	1	0
1	1	1	1
2	0	1	2

Figura 2.25: Mapas de Karnaugh de las funciones TXOR y TXNOR (9)

De esta forma se ha reducido el número de puertas necesarias para construir las funciones TXOR y TXNOR. Por tanto, encadenando las puertas TAND, TOR, TNAND y STI como en la figura 2.26 se generan las puertas TXOR y TXNOR.

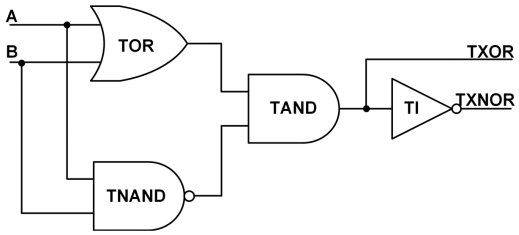


Figura 2.26: Puertas TXOR y TXNOR.

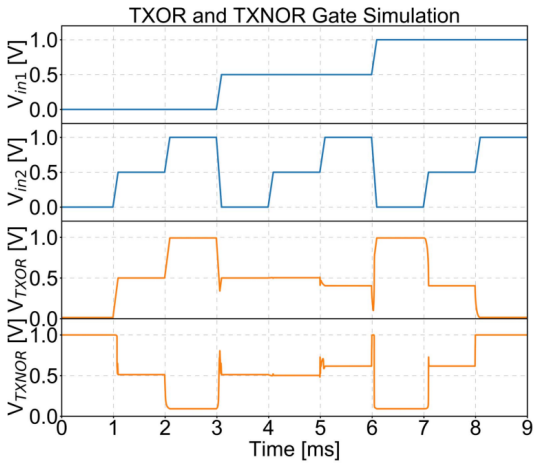


Figura 2.27: Simulación con SPICE de las puertas TXOR y TXNOR.

2.3.5. Conclusión

En la familia lógica multinivel memristor-CMOS los estados vienen determinados por valores de tensión. De esta forma se consigue un reducción notable del área empleado al aumentar el número de niveles por celda. De hecho, el número de funciones de n variables con p posibles valores es igual a p^{p^n}. Tomando $n = 2$ se deduce que existen 16 funciones binarias ($p = 2$) y 19683 funciones ternarias ($p = 3$). Esto se traduce en un aumento de la densidad de cálculo del dispositivo. En consecuencia, también se disminuye la cantidad de material empleado, permitiendo reducir los costes. Si se compara además con la tecnología CMOS actual, se ha conseguido reducir el área empleado hasta en 25 % y aumentar la densidad de datos en 5 órdenes de magnitud para las puertas TAND y TOR (9).
Uno de los inconvenientes que tiene la lógica multinivel es los márgenes de ruido entre niveles. Una solución para reducir estos márgenes es aumentar la transconductancia del memristor mediante ingeniería de materiales. Esta familia no requiere la inicialización de los memristores, permitiendo realizar operaciones en cascada sin dificultad. De hecho, se han obtenido tiempos de conmutación de nanosegundos en situaciones desfavorables (9), mejorando a otras familias lógicas memristivas pero lejos aún de la tecnología CMOS (que es capaz de conmutar en picosegundos). Por último, el consumo de energía es uno de los grandes problemas a mejorar. Si lo comparamos con la tecnología CMOS, la familia híbrida memristor-CMOS consume 100 veces más de media por cada puerta.

Capítulo 3

Simulación con resultados experimentales

En este capítulo se va a simular el comportamiento de las puertas lógicas TOR y TAND de la figura 2.14 empleando para ello la característica I-V de un memristor real.

3.1. Descripción de los dispositivos MIM utizados para esta simulación

En el contexto de los dieléctricos que conforman el aislante de una estructura Metal-Aislante-Metal (MIM) que presenta conmutación resistiva, el HfO_2 es uno de los óxidos más estudiados para la fabricación de celdas RRAM (ver (10), (11), (12), (13), (14)) debido a sus propiedades eléctricas. Los dispositivos basados en HfO_2 presentan buena escalabilidad, son compatibles con la tecnología CMOS y son capaces de operaciones rápidas y de bajo consumo.

Las muestras que se estudian y caracterizan en esta sección han sido fabricadas en el Instituto de Microelectrónica de Barcelona (IMB-CNM). Las muestras MIM estudiadas como dispositivos RRAM poseen una área de $1{,}44 \cdot 10^4\, m^2$, y presentan un electrodo bottom de 50 nm de tungsteno (W), un dieléctrico de $10\, nm$ de HfO_2 crecido por la técnica de deposición de capas atómicas (ALD) y un electrodo top compuesto por una capa de Ti de $10\, nm$ y la superior de $200\, nm$ de TiN (figura 3.1). Las capas metálicas fueron fabricadas mediante crecimiento por sputtering reactivo, y la definición de los electrodos superiores se realizó mediante la técnica de lift-off. En los diferentes pasos del proceso se utilizó un juego de máscaras específicamente diseñado para esta tecnología.

Para realizar medidas eléctricas, se aplicó voltaje sobre el metal top mientras el

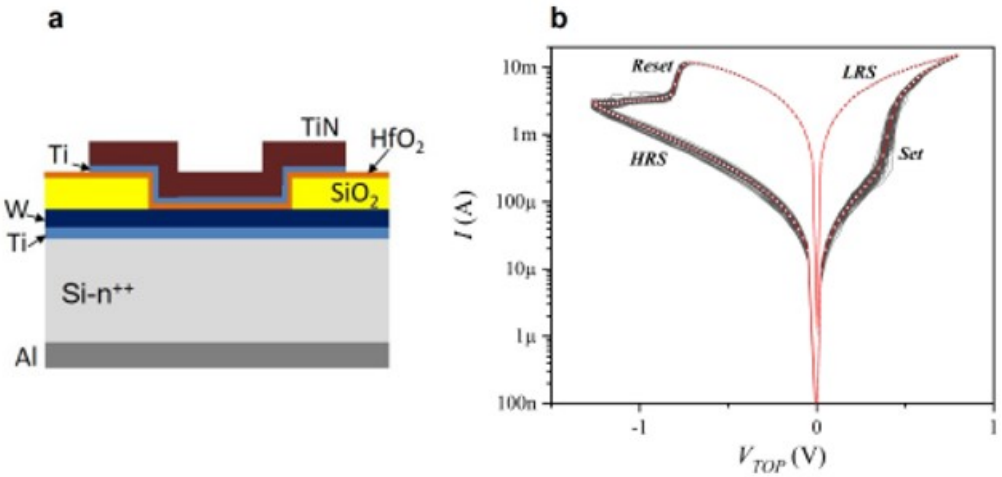


Figura 3.1: a) Sección transversal de una estructura MIM $W/HfO_2/Ti/TiN$.
b) Curvas I-V correspondientes a la misma estructura a 300 K.

bottom permanecía conectado a tierra. Tras realizar el electroforming, se obtuvieron las curvas I-V de estas estructuras a temperatura ambiente (300 K), que se pueden observar en la figura 3.1, claramente correspondientes a un mecanismo Valence Change Mechanism (VCM) debido a la formación de filamentos conductores constituidos por vacantes de oxígeno en el seno del aislante. Se obtuvo un comportamiento bipolar con una ventana (relación de corriente entre el estado de SET y RESET) de prácticamente dos órdenes de magnitud.

3.2. Resultados experimentales

Antes de comenzar la simulación, se ajusta la característica I-V del memristor mediante ecuaciones empíricas con ayuda del software *OriginLab*. Las ecuaciones resultantes se pueden ver en los programas del apéndice A. Si se deseara utilizar otro memristor bastaría con modificar las ecuaciones de los programas funcionOR.m y funcionAND.m. Observemos que en la figura 3.3 se ha prolongado la gráfica a la izquierda y a la derecha. Este comportamiento se observa experimentalmente cuando el memristor se satura por encima de un determinado valor y se ha tenido en cuenta a la hora de hacer la simulación.

En ambas puertas de la figura 2.14 se toma el convenio de que $V_{in1} > V_{in2}$. Si no fuera así, bastaría con intercambiar el papel que juegan los memristores M1 y M2. También se han desarrollado programas considerando el caso ideal en el que el memristor tiene dos estados con transiciones abruptas entre ellos.

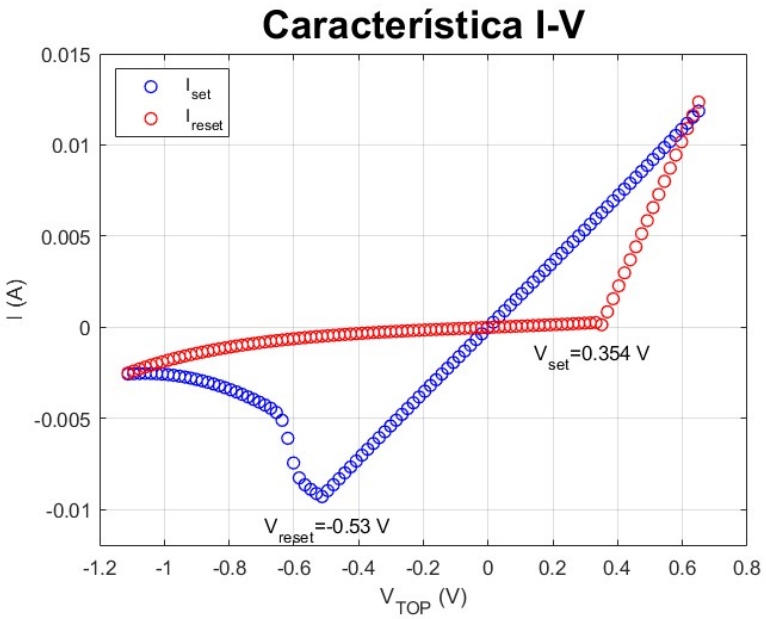


Figura 3.2: Datos experimentales del memristor empleado.

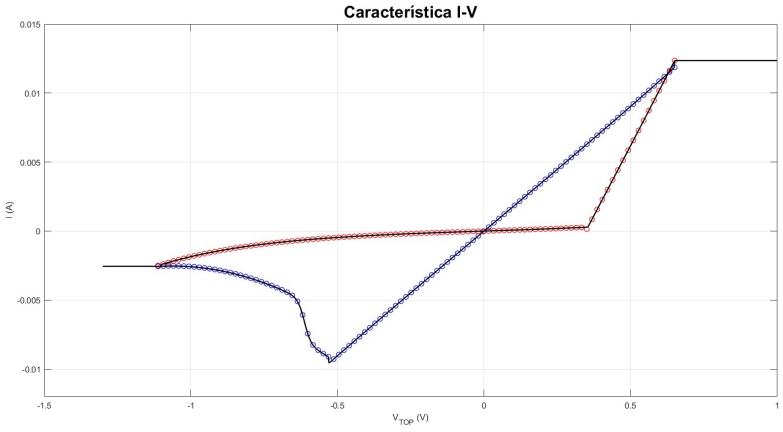


Figura 3.3: Ajuste de los datos experimentales con OriginLab.

3.2.1. Puerta lógica TOR

Como se observa en la imagen 3.4 el comportamiento de la puerta TOR con el memristor descrito en la sección anterior es excelente. Se ha simulado diferentes combinaciones de V_{in1} y V_{in2} (con $V_{in1} > V_{in2}$) y en todas ellas V_{out} es próximo a V_{in1}. Se muestran diferentes situaciones dependiendo del estado inicial en el que se encuentran los memristores. Se comprueba, por tanto, la independencia del valor de salida V_{out} con respecto a la situación inicial.

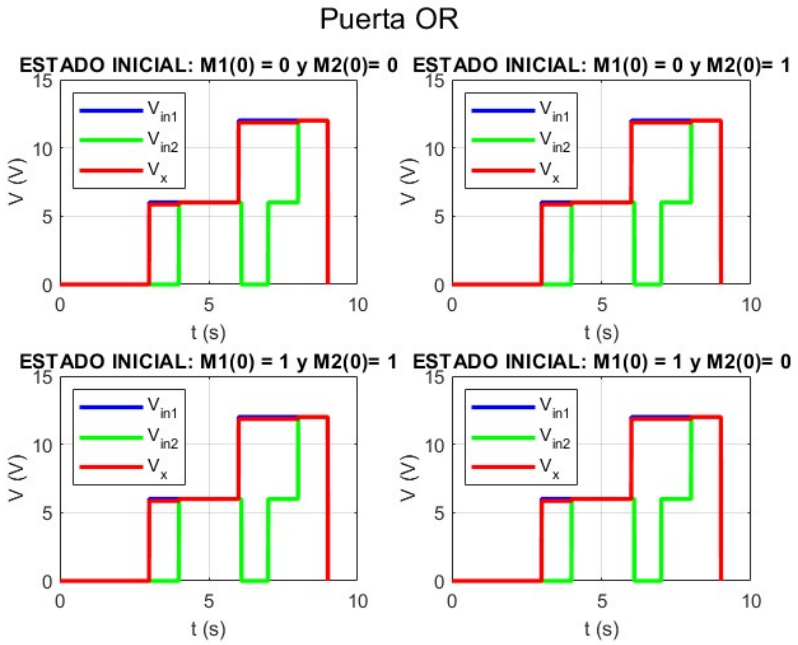


Figura 3.4: Evolución temporal de V_{in1}, V_{in2} y V_{out}.

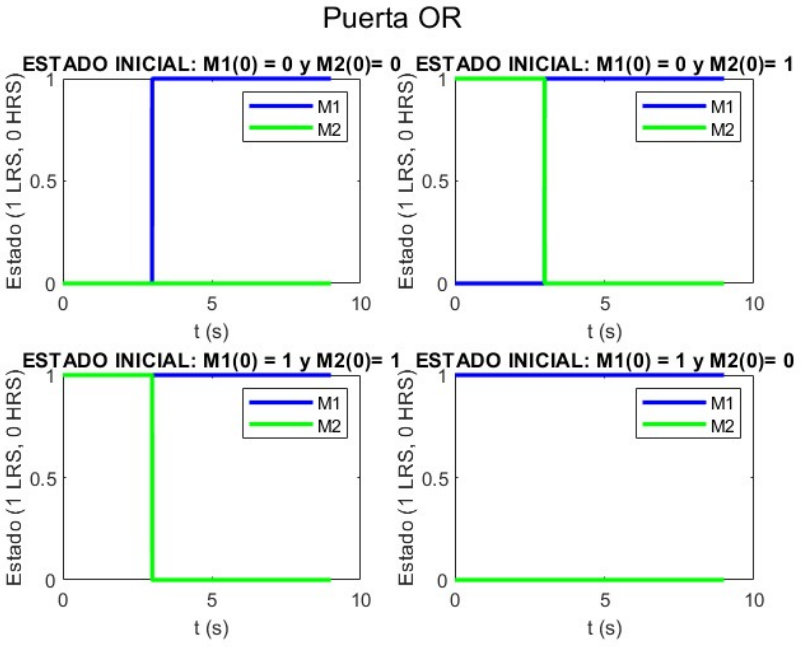


Figura 3.5: Evolución temporal de los estados de M1 y M2.

En la figura 3.5 se puede ver la evolución temporal de los estados de los memristores: si M1 pasa a estado SET entonces permanece en ese estado, y si M2 pasa a

RESET entonces permanece en ese estado. Esto es lógico pues $V_{in1} > V_{out} > V_{in2}$.

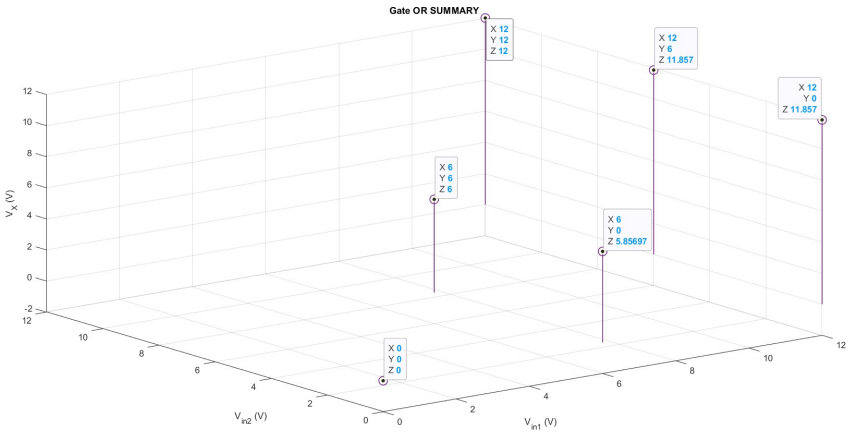


Figura 3.6: Resumen de la puerta TOR.

Se incluye también las mismas gráficas anteriores pero para el caso ideal. Se comprueba que los resultados son casi idénticos al caso real, confirmando los excelentes resultados obtenidos con el memristor real.

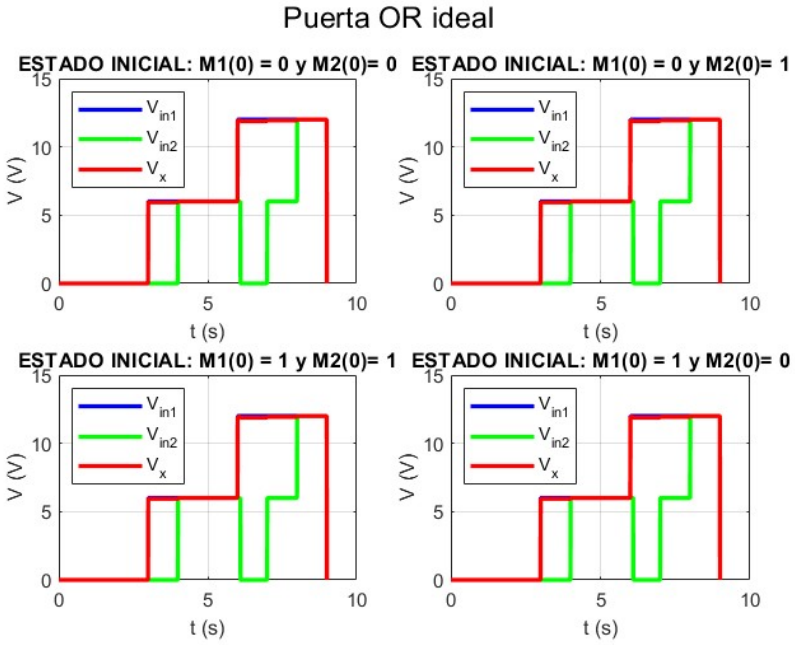


Figura 3.7: Evolución temporal de V_{in1}, V_{in2} y V_{out} para el caso ideal.

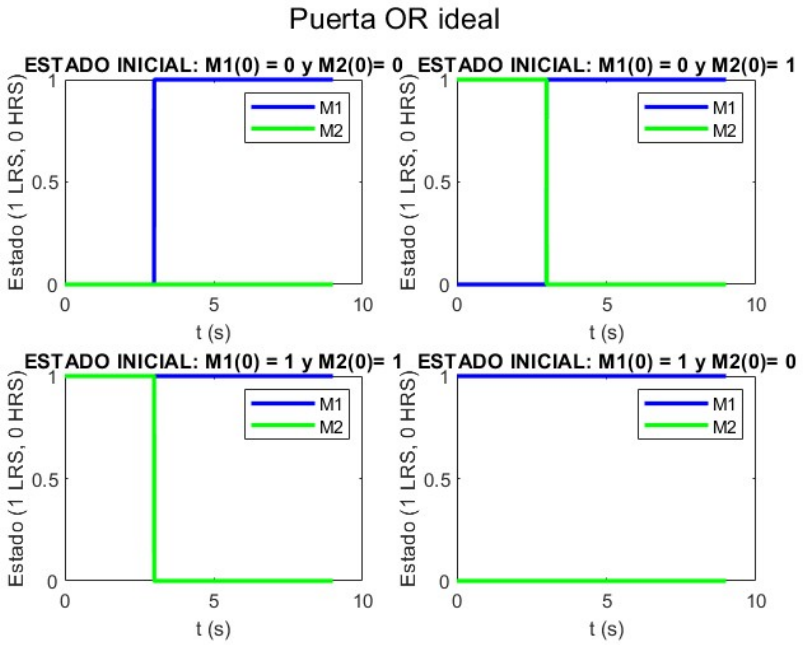


Figura 3.8: Evolución temporal de los estados de M1 y M2 para el caso ideal.

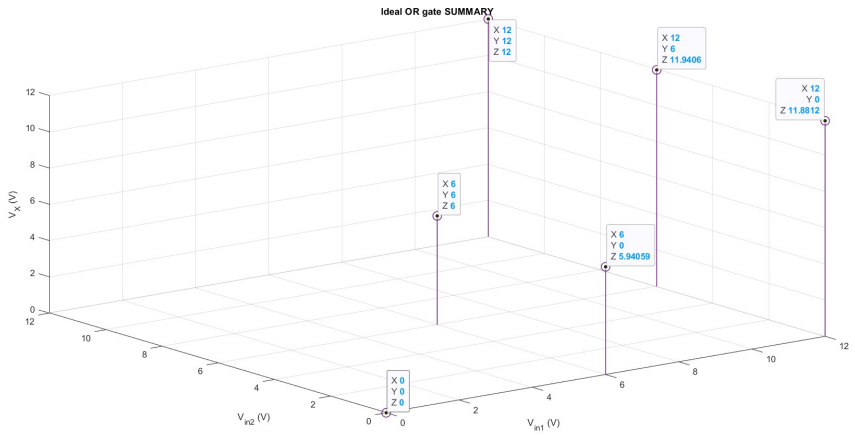


Figura 3.9: Resumen de la puerta TOR ideal.

3.2.2. Puerta lógica TAND

La puerta lógica TAND es idéntica a la puerta TOR pero cambiando la polaridad de los memristores. De nuevo se obtienen resultados muy buenos para diferentes parejas de tensiones (V_{in1}, V_{in2}).

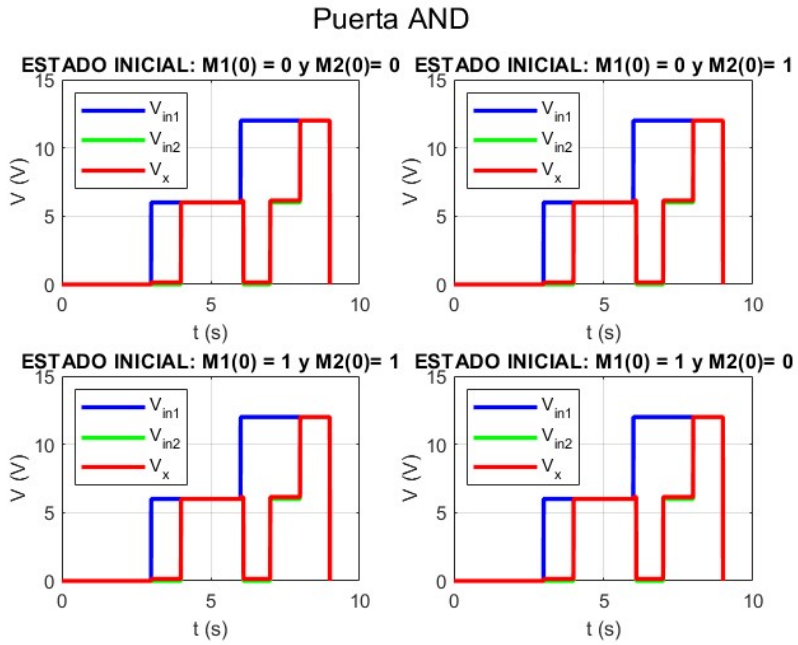


Figura 3.10: Evolución temporal de V_{in1}, V_{in2} y V_{out}.

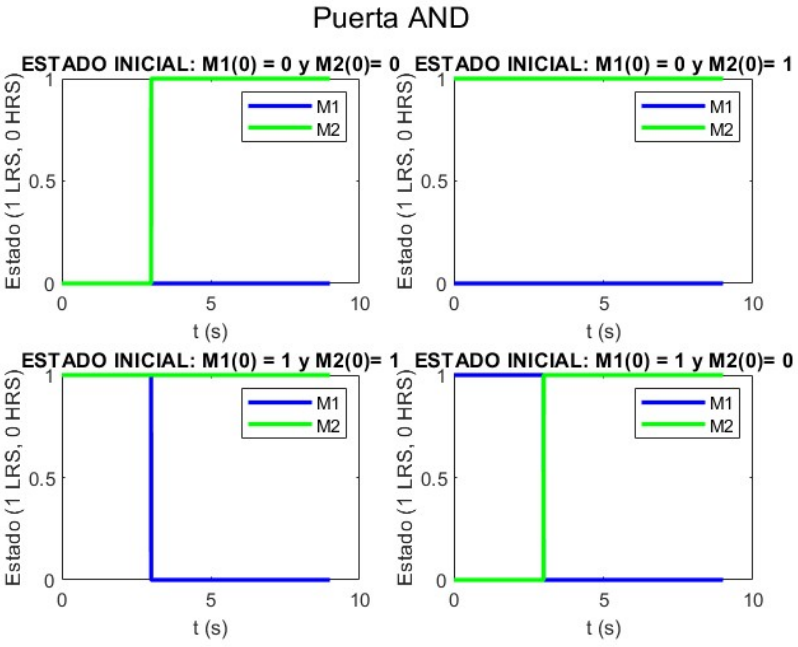


Figura 3.11: Evolución temporal de los estados de M1 y M2.

En la figura 3.11 se puede ver la diferencia respecto a la puerta TOR: en este caso si M1 está en RESET entonces permanece en este estado indefinidamente, y si M2

pasa a SET entonces no cambia a RESET al menos que V_{in2} pase a ser mayor que V_{in1}.

A continuación se muestran los resultados de la simulación para el caso ideal de la puerta TAND, comprobando el magnífico acuerdo con el caso real.

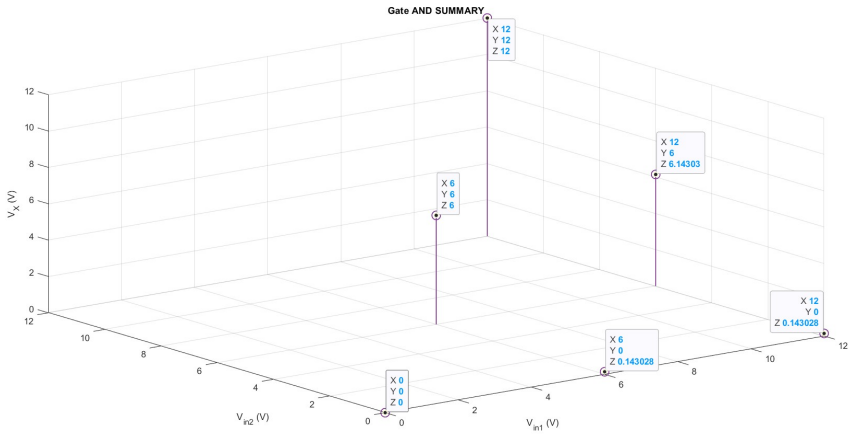


Figura 3.12: Resumen de la puerta TAND.

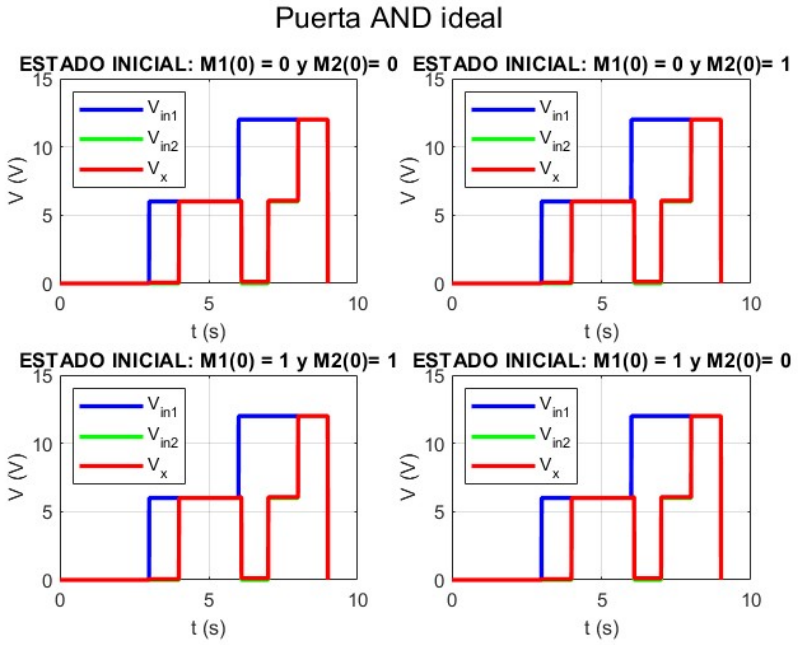


Figura 3.13: Evolución temporal de V_{in1}, V_{in2} y V_{out} para el caso ideal.

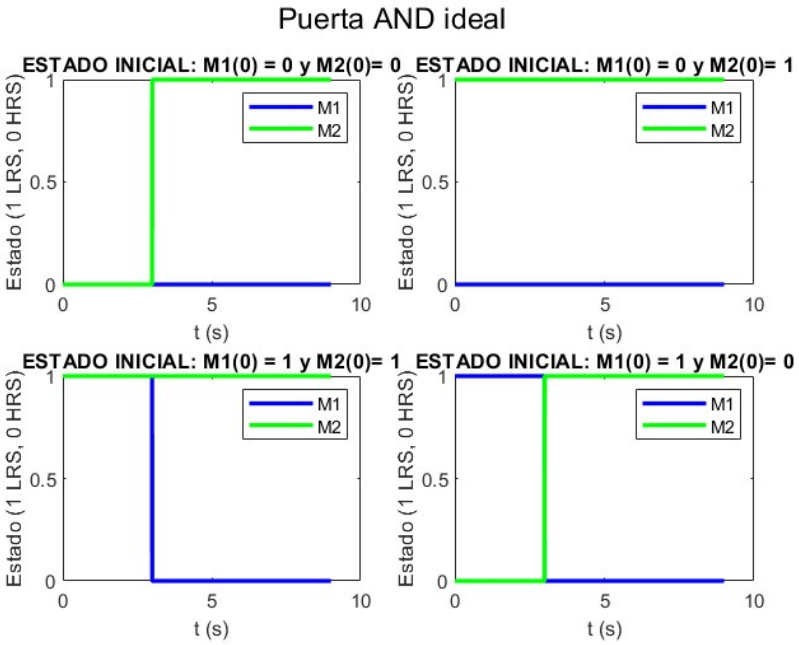


Figura 3.14: Evolución temporal de los estados de M1 y M2 para el caso ideal.

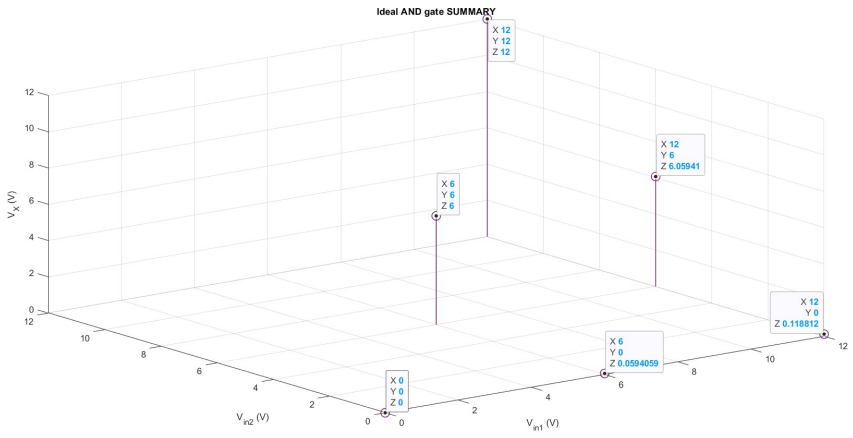


Figura 3.15: Resumen de la puerta TAND ideal.

3.3. Conclusiones

En la presente memoria se ha destacado el papel que juega el memristor como el cuarto elemento pasivo. Esto permite obtener una relación entre el flujo magnético y la carga que atraviesa el memristor, conocida como memristancia. Esta propiedad dota al memristor de un comportamiento no volátil, es decir, que el memristor

mantiene su estado lógico a pesar de que no exista una diferencia de tensión entre sus extremos. Este funcionamiento hace que el memristor sea un candidato ideal para la construcción de memorias RRAM (*Resistive* Random Access Memory). Este tipo de memorias son una alternativa prometedora a las memorias RAM o Flash actuales. En las memorias RRAM la conmutación resistiva es la responsable de almacenar los estados lógicos variando su resistencia.

Por otro lado, el desarrollo del memristor abre la puerta a nuevas familias lógicas multinivel. Se han desarrollado dos familias con comportamientos distintos, pero donde ambas aprovechan la conmutación resistiva para realizar operaciones lógicas. Además, la familia con la que se han realizado las simulaciones se combina con la tecnología CMOS, permitiendo así su integración en los próximos años.

Es importante tener en cuenta que las simulaciones que se han realizado emplean resultados experimentales sobre dispositivos fabricados para el grupo de investigación. De esta forma se ha podido comprobar la idoneidad de esta tecnología para construir puertas triestado. Tanto en la puerta TOR como en la puerta TAND se han simulado las salidas a partir de diferentes combinaciones para la entrada. Para su análisis se han simulado también los resultados para el caso ideal del memristor en el que la transición entre estados es abrupta. Como se ha podido comprobar a partir de las gráficas los resultados en ambas puertas son excelentes. Además el estado inicial en el que se encuentran los memristores apenas afecta a la salida de la puerta. Estos resultados ponen de manifiesto la posibilidad real de que los memristores sean empleados para la construcción de puertas lógicas. Por último es importante mencionar que se pueden repetir las simulaciones con otros memristores. Para ello bastaría con cambiar las ecuaciones que ajustan los datos experimentales en los programas.

3.4. Trabajo futuro

En este trabajo se han simulado las puertas TOR y TAND. Sin embargo, para completar la familia lógica se necesitaría simular también la puerta NOT ternaria: STI, PTI y NTI. De esta forma se podrían construir las puertas TNOR y TNAND. Una generalización del estudio llevado a cabo en esta memoria involucraría simular las puertas TMAX y TMIN y comprobar la dependencia de la salida con el número de entradas.

Una cuestión fundamental en las familias lógicas son los tiempos de conmutación. Por ello, sería interesante investigar la respuesta de este tipo de puertas lógicas basadas en memristores al tener señales dependientes del tiempo como entradas. A partir de los datos experimentales acerca de la respuesta temporal de un memristor individual se podrían construir modelos que permitieran predecir los tiempos

de respuesta y los retrasos de cada una de las puertas lógicas. En este sentido los materiales memristivos y los procesos físicos involucrados juegan un papel fundamental.

Apéndice A

Código MATLAB

En este apéndice se incluyen los scripts de MATLAB que se han desarrollado para llevar a cabo las simulaciones de las puertas lógicas TOR y TAND, explicadas en el capítulo 3. Para ambas puertas se han programado simulaciones de los modelos ideal y real del memristor.

A.1. Puerta lógica TOR

funcionOR.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FUNCION PARA HALLAR DE FORMA ANALITICA EL VALOR DE V_x EN LA PUERTA OR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% La salida de la funcion es la tension Vx
% La entrada de la funcion son las tensiones de puerta, Vin1 y Vin2, y los estados de los memristores M1 y M2
% Tambien toma como entrada un valor aproximado para Vx, aprox
% Si M1=0 entonces M1 esta en estado RESET
% Si M1=1 entonces M1 esta en estado SET
% De forma analoga para M2
function [Vx] = funcionOR(Vin1, Vin2, M1, M2, aprox)

% Parametros de la caracteristica I-V del memristor empleado
v1=-1.1;
vreset=-0.53;
vset=0.354;
v2=0.65;
% Parametro que se repite en las expresiones
A=2.66*10^(-4);

% En funcion del estado de M1 la expresion de la corriente cambia
if M1==1
```

```
    i1=@(x) ((Vin1-x)<=v1).*(-0.03172+...
        0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+v1)))+...
        0.90906./(1+10.^(1.05642.*(0.08094+v1))))-...
        A*sinh(-2.64134.*v1))+...
        ((((Vin1-x)>v1)&(Vin1-x)<=vreset)).*(-0.03172+...
        0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+(Vin1-x))))+...
        0.90906./(1+10.^(1.05642.*(0.08094+(Vin1-x))))))-...
        A*sinh(-2.64134.*(Vin1-x)))+...
        ((vreset<(Vin1-x))&((Vin1-x)<=v2)).*(0.01715.*(Vin1-x)-...
        A*sinh(-2.64134.*(Vin1-x)))+((Vin1-x)>v2).*(-0.01392+...
        0.03932*(v2)-A*sinh(-2.64134*(v2)));
else
    i1=@(x) ((Vin1-x)<=v1).*(-0.03172+...
        0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+v1)))+...
        0.90906./(1+10.^(1.05642.*(0.08094+v1))))-...
        A*sinh(-2.64134.*v1))+...
        (((Vin1-x)>v1)&((Vin1-x)<=vset)).*(...
        -A*sinh(-2.64134.*(Vin1-x)))+...
        ((vset<(Vin1-x))&((Vin1-x)<=v2)).*(-0.01392+...
        0.03932.*(Vin1-x)-A*sinh(-2.64134.*(Vin1-x)))+...
        ((Vin1-x)>v2).*(-0.01392+0.03932*(v2)-A*sinh(-2.64134*(v2)))
            ;
end

% Hacemos lo mismo con M2
if M2==1
    i2=@(x) ((Vin2-x)<=v1).*(-0.03172+...
        0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+v1)))+...
        0.90906./(1+10.^(1.05642.*(0.08094+v1))))-...
        A*sinh(-2.64134.*v1))+...
        ((((Vin2-x)>v1)&(Vin2-x)<=vreset)).*(-0.03172+...
        0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+(Vin2-x))))+...
        0.90906./(1+10.^(1.05642.*(0.08094+(Vin2-x))))))-...
        A*sinh(-2.64134.*(Vin2-x)))+...
        ((vreset<(Vin2-x))&((Vin2-x)<=v2)).*(0.01715.*(Vin2-x)-...
        A*sinh(-2.64134.*(Vin2-x)))+((Vin2-x)>v2).*(-0.01392+...
        0.03932*(v2)-A*sinh(-2.64134*(v2)));
else
    i2=@(x) ((Vin2-x)<=v1).*(-0.03172+...
        0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+v1)))+...
        0.90906./(1+10.^(1.05642.*(0.08094+v1))))-...
        A*sinh(-2.64134.*v1))+...
        (((Vin2-x)>v1)&((Vin2-x)<=vset)).*(-A*sinh(-2.64134.*(Vin2-x
            )))+...
        ((vset<(Vin2-x))&((Vin2-x)<=v2)).*(-0.01392+...
        0.03932.*(Vin2-x)-A*sinh(-2.64134.*(Vin2-x)))+...
        ((Vin2-x)>v2).*(-0.01392+0.03932*(v2)-A*sinh(-2.64134*(v2)))
            ;
end

```

```
% Ahora empleamos la funcion fzero para hallar el valor de tension
    que hace i1=-i2. La funcion de la que queremos hallar su raiz
    es f=i1+i2, f(Vx)=0
f=@(x) i1(x)+i2(x);
% El segundo argumento de fzero es una estimacion inicial de Vx
Vx=fzero(f, aprox);

end
```

evolucionOR.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PROGRAMA PARA MOSTRAR LA EVOLUCION TEMPORAL DE LA PUERTA OR %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Definimos el numero de puntos en cada eje
N=1500;
% Hacemos la malla del eje horizontal
t=linspace(0,9,N);
% Escogemos la forma de la onda Vin1
onda1 = @(t) ((t<3)).*0 +...
             ((t>=3) & (t<=6)).*1+...
             ((t>6) & (t<9)).*2 ;
% Escogemos la forma de la onda Vin2 con la condicion de que Vin2
   <=Vin1 (en otro caso bastaria intercambiar los papeles de M1 y
   M2)
onda2 =  @(t) ((t<3)).*0+...
              ((t>=3) & (t<4)).*0 + ...
              ((t>=4) & (t<5)).*1+...
              ((t>=5) & (t<6.1)).*1+...
              ((t>=6.5) & (t<7)).*0 + ...
              ((t>=7) & (t<8)).*1+...
              ((t>=8) & (t<9)).*2;

% Parametros del memristor
vreset=-0.53;
vset=0.354;
% Definimos un parametro que nos permite modificar la amplitud de
   las ondas onda1 y onda2 sin cambiar su forma y sus proporciones
escala=6;

% Definimos las tensiones de entrada de la puerta OR
Vin1=onda1(t).*escala;
Vin2=onda2(t).*escala;

% El vector raices contiene los valores de Vx para cada pareja de
   tensiones (Vin1,Vin2)
raices=zeros(1,N);

% El parametro "inicio" especifica la situacion en la que se
   encuentran inicialmente los memristores M1 y M2:
```

```
% inicio=1 ---> M1=RESET M2=RESET
% inicio=2 ---> M1=RESET M2=SET
% inicio=3 ---> M1=SET M2=SET
% inicio=4 ---> M1=SET M2=RESET

% Hacemos un bucle para representar todas las situaciones de M1 y M2
for inicio=1:4

% Dependiendo del valor de "inicio" tenemos comportamientos diferentes de M1 y M2 (en cuanto a la posibilidad de que conmuten entre estados SET y RESET)
switch inicio

    case 1
    % Si ambos estan en RESET entonces M2 va a permanecer en RESET pues Vx>Vin2 pero M1 puede cambiar a SET
        % Inicializamos los estados de los memristores
        M1=zeros(1,N);
        M2=zeros(1,N);
        % Para cada pareja (Vin1,Vin2) calculamos el correspondiente valor de Vx
        for i = 2:N
            % En cada iteracion se emplea el valor de Vx anterior como aproximante para la siguiente iteracion en funcionOR
            raices(i) = funcionOR(Vin1(i), Vin2(i), M1(i), M2(i), raices(i-1));
            % Comprobamos si M1 cambia a SET para actualizar el estado y volver a calcular el Vx correcto
            if (Vin1(i)-raices(i))>=vset
               M1(i:N)=1;
               raices(i) = funcionOR(Vin1(i), Vin2(i), M1(i), M2(i), raices(i-1));
            end
        end

    case 2
        M1=zeros(1,N);
        M2=ones(1,N);

        for i = 2:N
           raices(i) = funcionOR(Vin1(i), Vin2(i), M1(i), M2(i), raices(i-1));
            % Comprobamos si M1 cambia a SET
            if (Vin1(i)-raices(i))>=vset
               M1(i:N)=1;
               raices(i) = funcionOR(Vin1(i), Vin2(i), M1(i), M2(i), raices(i-1));
            end
            % Comprobamos si M2 cambia a RESET
```

```
75              if (Vin2(i)-raices(i))<=vreset
76                 M2(i:N)=0;
77                 raices(i) = funcionOR(Vin1(i), Vin2(i), M1(i), M2(i
                      ), raices(i-1));
78              end
79          end
80
81      case 3
82          M1=ones(1,N);
83          M2=ones(1,N);
84
85          for i = 2:N
86             raices(i) = funcionOR(Vin1(i), Vin2(i), M1(i), M2(i),
                   raices(i-1));
87              % Comprobamos si M2 cambia a RESET
88               if (Vin2(i)-raices(i))<=vreset
89                  M2(i:N)=0;
90                  raices(i) = funcionOR(Vin1(i), Vin2(i), M1(i), M2(
                       i), raices(i-1));
91              end
92          end
93
94      otherwise
95          M1=ones(1,N);
96          M2=zeros(1,N);
97
98          for i = 2:N
99             raices(i) = funcionOR(Vin1(i), Vin2(i), M1(i), M2(i),
                   raices(i-1));
100             % En este caso no hay que comprobar nada porque M1 no
                   va a cambiar RESET ni M2 va a cambiar a SET
101         end
102
103 end
104
105 % Representamos las tensiones Vin1, Vin2 y Vx en funcion del
       tiempo
106 figure(1);
107 subplot(2,2,inicio)
108 plot(t, Vin1, 'b-','LineWidth', 2,'DisplayName', 'V_{in1}');
109 hold on
110 plot(t, Vin2, 'g-','LineWidth', 2,'DisplayName', 'V_{in2}');
111 hold on
112 plot(t, raices, 'r-','LineWidth', 2,'DisplayName', 'V_{x}');
113 hold on
114 grid on;
115
116 % Agregamos etiquetas al grafico
117 xlabel('t (s)');
118 ylabel('V (V)');
119 title(['ESTADO INICIAL: M1(0) = ', num2str(M1(1)),' y M2(0)= ',
```

```
    num2str(M2(1))]);
legend('show', 'Location', 'northwest');
sgtitle('Puerta OR');

% Escogemos el nombre del grafico y lo guardamos
namefig=('OR_voltajes.fig');
savefig (namefig);

% Representamos los estados de M1 y M2 en funcion del tiempo
figure(2);
subplot(2,2,inicio)
plot(t, M1, 'b-','LineWidth', 2,'DisplayName', 'M1');
hold on
plot(t, M2, 'g-','LineWidth', 2,'DisplayName', 'M2');
hold on

% Agregamos etiquetas al grafico
xlabel('t (s)');
ylabel('Estado (1 LRS, 0 HRS)');
title(['ESTADO INICIAL: M1(0) = ', num2str(M1(1)),' y M2(0)= ',
    num2str(M2(1))]);
legend('show', 'Location', 'northeast');
sgtitle('Puerta OR');

% Escogemos el nombre del grafico y lo guardamos
namefig=('OR_estados.fig');
savefig(namefig);

% Representamos en un grafico 3D la puerta OR
figure(3)
stem3(Vin1,Vin2,raices,'o','MarkerSize',10,'LineWidth',1)
xlabel('V_{in1} (V)');
ylabel('V_{in2} (V)');
zlabel('V_{X} (V)');
title('Gate OR SUMMARY');
hold on

% Escogemos el nombre del grafico y lo guardamos
namefig=('OR_3d.fig');
savefig(namefig);
hold on

end
```

funcionORideal.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     FUNCION PARA HALLAR DE FORMA ANALITICA EL VALOR DE V_x EN LA
      PUERTA OR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
% La diferencia respecto al programa funcionOR es que se han
    considerado las expresiones para la corriente de un memristor
    ideal en el que el ratio Roff/Ron es grande

% La salida de la funcion es la tension Vx
% La entrada de la funcion son las tensiones de puerta, Vin1 y
    Vin2, y los estados de los memristores M1 y M2
% Si M1=0 entonces M1 esta en estado RESET
% Si M1=1 entonces M1 esta en estado SET
% De forma analoga para M2
function [Vx] = funcionORideal(Vin1, Vin2, M1, M2)

% Parametros de la caracteristica I-V del memristor ideal
Ron=100;
Roff=1E4;

% Establecemos el valor de la resistencia de M1
if M1==1
   R1=Ron;
else
   R1=Roff;
end
% Hacemos lo mismo con M2
if M2==1
   R2=Ron;
else
   R2=Roff;
end

% Hallamos Vx mediante el divisor de tension
Vx=Vin2+(Vin1-Vin2)*R2/(R1+R2);

end
```

evolucionORideal.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      PROGRAMA PARA MOSTRAR LA EVOLUCION TEMPORAL DE LA PUERTA OR
      IDEAL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Se trata del mismo programa que evolucionOR con la diferencia de
    que se llama al programa funcionORideal en lugar de funcionOR

% Definimos el numero de puntos en cada eje
N=1500;
% Hacemos la malla del eje horizontal
t=linspace(0,9,N);
% Escogemos la forma de la onda Vin1
onda1 = @(t) ((t<3)).*0 +...
             ((t>=3) & (t<=6)).*1+...
```

```
13              ((t>6) & (t<9)).*2 ;
14 % Escogemos la forma de la onda Vin2 con la condicion de que Vin2
       <=Vin1 (en otro caso bastaria intercambiar los papeles de M1 y
       M2)
15 onda2 =  @(t) ((t<3)).*0+...
16                ((t>=3) & (t<4)).*0 + ...
17                ((t>=4) & (t<5)).*1+...
18                ((t>=5) & (t<6.1)).*1+...
19                ((t>=6.5) & (t<7)).*0 + ...
20                ((t>=7) & (t<8)).*1+...
21                ((t>=8) & (t<9)).*2;
22
23 % Parametros del memristor
24 vreset=-0.53;
25 vset=0.354;
26 % Definimos un parametro que nos permite modificar la amplitud de
       las ondas onda1 y onda2 sin cambiar su forma y sus proporciones
27 escala=6;
28
29 % Definimos las tensiones de entrada de la puerta OR
30 Vin1=onda1(t).*escala;
31 Vin2=onda2(t).*escala;
32
33 % El vector raices contiene los valores de Vx para cada pareja de
       tensiones (Vin1,Vin2)
34 raices=zeros(1,N);
35
36 % El parametro "inicio" especifica la situacion en la que se
       encuentran inicialmente los memristores M1 y M2:
37 % inicio=1 ---> M1=RESET M2=RESET
38 % inicio=2 ---> M1=RESET M2=SET
39 % inicio=3 ---> M1=SET M2=SET
40 % inicio=4 ---> M1=SET M2=RESET
41
42 % Hacemos un bucle para representar todas las situaciones de M1 y
       M2
43 for inicio=1:4
44
45 % Dependiendo del valor de "inicio" tenemos comportamientos
       diferentes de M1 y M2 (en cuanto a la posibilidad de que
       conmuten entre estados SET y RESET)
46 switch inicio
47
48     case 1
49     % Si ambos estan en RESET entonces M2 va a permanecer en RESET
           pues Vx>Vin2 pero M1 puede cambiar a SET
50         % Inicializamos los estados de los memristores
51         M1=zeros(1,N);
52         M2=zeros(1,N);
53         % Para cada pareja (Vin1,Vin2) calculamos el
               correspondiente valor de Vx
```

```
54          for i = 2:N
55              raices(i) = funcionORideal(Vin1(i), Vin2(i), M1(i), M2
                    (i));
56              % Comprobamos si M1 cambia a SET para actualizar el
                    estado y volver a calcular el Vx correcto
57              if (Vin1(i)-raices(i))>=vset
58                 M1(i:N)=1;
59                 raices(i) = funcionORideal(Vin1(i), Vin2(i), M1(i),
                        M2(i));
60              end
61          end
62
63      case 2
64          M1=zeros(1,N);
65          M2=ones(1,N);
66
67          for i = 2:N
68             raices(i) = funcionORideal(Vin1(i), Vin2(i), M1(i), M2(
                   i));
69              % Comprobamos si M1 cambia a SET
70              if (Vin1(i)-raices(i))>=vset
71                 M1(i:N)=1;
72                 raices(i) = funcionORideal(Vin1(i), Vin2(i), M1(i),
                        M2(i));
73              end
74              % Comprobamos si M2 cambia a RESET
75              if (Vin2(i)-raices(i))<=vreset
76                 M2(i:N)=0;
77                 raices(i) = funcionORideal(Vin1(i), Vin2(i), M1(i),
                        M2(i));
78              end
79          end
80
81      case 3
82          M1=ones(1,N);
83          M2=ones(1,N);
84
85          for i = 2:N
86             raices(i) = funcionORideal(Vin1(i), Vin2(i), M1(i), M2(
                   i));
87              % Comprobamos si M2 cambia a RESET
88               if (Vin2(i)-raices(i))<=vreset
89                  M2(i:N)=0;
90                  raices(i) = funcionORideal(Vin1(i), Vin2(i), M1(i)
                        , M2(i));
91              end
92          end
93
94      otherwise
95          M1=ones(1,N);
96          M2=zeros(1,N);
```

```
97
98          for i = 2:N
99              raices(i) = funcionORideal(Vin1(i), Vin2(i), M1(i), M2(
                    i));
100             % En este caso no hay que comprobar nada porque M1 no
                    va a cambiar RESET ni M2 va a cambiar a SET
101         end
102
103 end
104
105 % Representamos las tensiones Vin1, Vin2 y Vx en funcion del
        tiempo
106 figure(1);
107 subplot(2,2,inicio)
108 plot(t, Vin1, 'b-','LineWidth', 2,'DisplayName', 'V_{in1}');
109 hold on
110 plot(t, Vin2, 'g-','LineWidth', 2,'DisplayName', 'V_{in2}');
111 hold on
112 plot(t, raices, 'r-','LineWidth', 2,'DisplayName', 'V_{x}');
113 hold on
114 grid on;
115
116 % Agregamos etiquetas al grafico
117 xlabel('t (s)');
118 ylabel('V (V)');
119 title(['ESTADO INICIAL: M1(0) = ', num2str(M1(1)),' y M2(0)= ',
        num2str(M2(1))]);
120 legend('show', 'Location', 'northwest');
121 sgtitle('Puerta OR ideal');
122
123 % Escogemos el nombre del grafico y lo guardamos
124 namefig=('OR_ideal_voltajes.fig');
125 savefig (namefig);
126
127 % Representamos los estados de M1 y M2 en funcion del tiempo
128 figure(2);
129 subplot(2,2,inicio)
130 plot(t, M1, 'b-','LineWidth', 2,'DisplayName', 'M1');
131 hold on
132 plot(t, M2, 'g-','LineWidth', 2,'DisplayName', 'M2');
133 hold on
134
135 % Agregamos etiquetas al grafico
136 xlabel('t (s)');
137 ylabel('Estado (1 LRS, 0 HRS)');
138 title(['ESTADO INICIAL: M1(0) = ', num2str(M1(1)),' y M2(0)= ',
        num2str(M2(1))]);
139 legend('show', 'Location', 'northeast');
140 sgtitle('Puerta OR ideal');
141
142 % Escogemos el nombre del grafico y lo guardamos
```

```
namefig=('OR_ideal_estados.fig');
savefig(namefig);

% Representamos en un grafico 3D la puerta OR ideal
figure(3)
stem3(Vin1,Vin2,raices,'o','MarkerSize',10,'LineWidth',1)
xlabel('V_{in1} (V)');
ylabel('V_{in2} (V)');
zlabel('V_{X} (V)');
title('Ideal OR gate SUMMARY');
hold on

% Escogemos el nombre del grafico y lo guardamos
namefig=('OR_ideal_3d.fig');
savefig(namefig);
hold on

end
```

A.2. Puerta lógica TAND

funcionAND.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    FUNCION PARA HALLAR DE FORMA ANALITICA EL VALOR DE V_x EN LA
     PUERTA AND
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% La unica diferencia respecto al programa funcionOR es que se ha
   sustituido Vin-Vx por Vx-Vin en las expresiones de las
   corrientes ya que la polarizacion de los memristores en la
   puerta AND es opuesta a la de la puerta OR

% La salida de la funcion es la tension Vx
% La entrada de la funcion son las tensiones de puerta, Vin1 y
   Vin2, y los estados de los memristores M1 y M2
% Tambien toma como entrada un valor aproximado para Vx
% Si M1=0 entonces M1 esta en estado RESET
% Si M1=1 entonces M1 esta en estado SET
% De forma analoga para M2
function [Vx] = funcionAND(Vin1, Vin2, M1, M2, aprox)

% Parametros de la caracteristica I-V del memristor empleado
v1=-1.1;
vreset=-0.53;
vset=0.354;
v2=0.65;
% Parametro que se repite en las expresiones
A=2.66*10^(-4);
```

```
21
22 % En funcion del estado de M1 la expresion de la corriente cambia
23 if M1==1
24     i1=@(x) ((x-Vin1)<=v1).*(-0.03172+...
25         0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+v1)))+...
26         0.90906./(1+10.^(1.05642.*(0.08094+v1))))-...
27         A*sinh(-2.64134.*v1))+...
28         ((((x-Vin1)>v1)&(x-Vin1)<=vreset)).*(-0.03172+...
29         0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+(x-Vin1))))+...
30         0.90906./(1+10.^(1.05642.*(0.08094+...
31         (x-Vin1)))))-A*sinh(-2.64134.*(x-Vin1)))+...
32         ((vreset<(x-Vin1))&((x-Vin1)<=v2)).*(0.01715.*(x-Vin1)-...
33         A*sinh(-2.64134.*(x-Vin1)))+...
34         ((x-Vin1)>v2).*(-0.01392+0.03932*(v2)-A*sinh(-2.64134*(v2)))
               ;
35 else
36     i1=@(x) ((x-Vin1)<=v1).*(-0.03172+...
37         0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+v1)))+...
38         0.90906./(1+10.^(1.05642.*(0.08094+v1))))-...
39         A*sinh(-2.64134.*v1))+...
40         (((x-Vin1)>v1)&((x-Vin1)<=vset)).*(-A*sinh(-2.64134.*(x-Vin1
               )))+...
41         ((vset<(x-Vin1))&((x-Vin1)<=v2)).*(-0.01392+...
42         0.03932.*(x-Vin1)-A*sinh(-2.64134.*(x-Vin1)))+...
43         ((x-Vin1)>v2).*(-0.01392+0.03932*(v2)-A*sinh(-2.64134*(v2)))
               ;
44 end
45
46 % Hacemos lo mismo con M2
47 if M2==1
48     i2=@(x) ((x-Vin2)<=v1).*(-0.03172+...
49         0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+v1)))+...
50         0.90906./(1+10.^(1.05642.*(0.08094+v1))))-...
51         A*sinh(-2.64134.*v1))+...
52         ((((x-Vin2)>v1)&(x-Vin2)<=vreset)).*(-0.03172+...
53         0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+(x-Vin2))))+...
54         0.90906./(1+10.^(1.05642.*(0.08094+(x-Vin2)))))-...
55         A*sinh(-2.64134.*(x-Vin2)))+...
56         ((vreset<(x-Vin2))&((x-Vin2)<=v2)).*(0.01715.*(x-Vin2)-...
57         A*sinh(-2.64134.*(x-Vin2)))+...
58         ((x-Vin2)>v2).*(-0.01392+0.03932*(v2)-A*sinh(-2.64134*(v2)))
               ;
59 else
60     i2=@(x) ((x-Vin2)<=v1).*(-0.03172+...
61         0.03394.*(0.09194./(1+10.^(38.86159.*(0.6122+v1)))+...
62         0.90906./(1+10.^(1.05642.*(0.08094+v1))))-...
63         A*sinh(-2.64134.*v1))+...
64         (((x-Vin2)>v1)&((x-Vin2)<=vset)).*(-A*sinh(-2.64134.*(x-Vin2
               )))+...
65         ((vset<(x-Vin2))&((x-Vin2)<=v2)).*(-0.01392+...
66         0.03932.*(x-Vin2)-A*sinh(-2.64134.*(x-Vin2)))+...
```

```
67        ((x-Vin2)>v2).*(-0.01392+0.03932*(v2)-A*sinh(-2.64134*(v2)))
              ;
68  end
69
70  % Ahora empleamos la funcion fzero para hallar el valor de tension
        que hace i1=-i2. La funcion de la que queremos hallar su raiz
        es f=i1+i2, f(Vx)=0
71  f=@(x) i1(x)+i2(x);
72  % El segundo argumento de fzero es una estimacion inicial de Vx
73  Vx=fzero(f, aprox);
74
75  end
```

evolucionAND.m

```
1   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2   %% PROGRAMA PARA MOSTRAR LA EVOLUCION TEMPORAL DE LA PUERTA AND %%
3   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5   % Definimos el numero de puntos en cada eje
6   N=1500;
7   % Hacemos la malla del eje horizontal
8   t=linspace(0,9,N);
9   % Escogemos la forma de la onda Vin1
10  onda1 = @(t) ((t<3)).*0 +...
11               ((t>=3) & (t<=6)).*1+...
12               ((t>6) & (t<9)).*2 ;
13  % Escogemos la forma de la onda Vin2 con la condicion de que Vin2
        <=Vin1 (en otro caso bastaria intercambiar los papeles de M1 y
        M2)
14  onda2 =  @(t) ((t<3)).*0+...
15                ((t>=3) & (t<4)).*0 + ...
16                ((t>=4) & (t<5)).*1+...
17                ((t>=5) & (t<6.1)).*1+...
18                ((t>=6.5) & (t<7)).*0 + ...
19                ((t>=7) & (t<8)).*1+...
20                ((t>=8) & (t<9)).*2;
21
22  % Parametros del memristor
23  vreset=-0.53;
24  vset=0.354;
25  % Definimos un parametro que nos permite modificar la amplitud de
        las ondas onda1 y onda2 sin cambiar su forma y sus proporciones
26  escala=6;
27
28  % Definimos las tensiones de entrada de la puerta AND
29  Vin1=onda1(t).*escala;
30  Vin2=onda2(t).*escala;
31
32  % El parametro "inicio" especifica la situacion en la que se
        encuentran
```

```
33 % inicialmente los memristores M1 y M2:
34 % inicio=1 ---> M1=RESET M2=RESET
35 % inicio=2 ---> M1=RESET M2=SET
36 % inicio=3 ---> M1=SET M2=SET
37 % inicio=4 ---> M1=SET M2=RESET
38
39 % Hacemos un bucle para representar todas las situaciones de M1 y
       M2
40 for inicio=1:4
41
42 % El vector raices contiene los valores de Vx para cada pareja de
       tensiones (Vin1,Vin2)
43 raices=zeros(1,N);
44
45 % Dependiendo del valor de "inicio" tenemos comportamientos
       diferentes de M1 y M2 (en cuanto a la posibilidad de que
       conmuten entre estados SET y RESET)
46 switch inicio
47
48     case 1
49     % Si ambos estan en RESET entonces M1 va a permanecer en RESET
           pues Vin1>Vx pero M2 puede cambiar a SET
50         % Inicializamos los estados de los memristores
51         M1=zeros(1,N);
52         M2=zeros(1,N);
53         % Para cada pareja (Vin1,Vin2) calculamos el
               correspondiente valor de Vx
54         for i = 2:N
55             % En cada iteracion se emplea el valor de Vx anterior
                   como aproximante para la siguiente iteracion en
                   funcionAND
56             raices(i) = funcionAND(Vin1(i), Vin2(i), M1(i), M2(i),
                    raices(i-1));
57             % Comprobamos si M2 cambia a SET para actualizar el
                   estado y volver a calcular el Vx correcto
58             if (raices(i)-Vin2(i))>=vset
59                M2(i:N)=1;
60                raices(i) = funcionAND(Vin1(i), Vin2(i), M1(i), M2(
                      i), raices(i-1));
61             end
62         end
63
64     case 2
65         M1=zeros(1,N);
66         M2=ones(1,N);
67
68         for i = 2:N
69            raices(i) = funcionAND(Vin1(i), Vin2(i), M1(i), M2(i),
                   raices(i-1));
70            % En este caso no hay que comprobar nada porque M1 no
                   va a cambiar SET ni M2 va a cambiar a RESET
```

```
71          end
72
73      case 3
74          M1=ones(1,N);
75          M2=ones(1,N);
76
77          for i = 2:N
78             raices(i) = funcionAND(Vin1(i), Vin2(i), M1(i), M2(i),
                  raices(i-1));
79              % Comprobamos si M1 cambia a RESET
80               if (raices(i)-Vin1(i))<=vreset
81                  M1(i:N)=0;
82                  raices(i) = funcionAND(Vin1(i), Vin2(i), M1(i), M2
                       (i), raices(i-1));
83              end
84          end
85
86      otherwise
87          M1=ones(1,N);
88          M2=zeros(1,N);
89
90          for i = 2:N
91             raices(i) = funcionAND(Vin1(i), Vin2(i), M1(i), M2(i),
                  raices(i-1));
92             % Comprobamos si M1 cambia a RESET
93             if (raices(i)-Vin1(i))<=vreset
94                  M1(i:N)=0;
95                  raices(i) = funcionAND(Vin1(i), Vin2(i), M1(i), M2
                       (i), raices(i-1));
96             end
97             % Comprobamos si M2 cambia a SET
98             if (raices(i)-Vin2(i))>=vset
99                  M2(i:N)=1;
100                 raices(i) = funcionAND(Vin1(i), Vin2(i), M1(i), M2
                       (i), raices(i-1));
101            end
102         end
103
104 end
105
106 % Representamos las tensiones Vin1, Vin2 y Vx en funcion del
       tiempo
107 figure(1);
108 subplot(2,2,inicio)
109 plot(t, Vin1, 'b-','LineWidth', 2,'DisplayName', 'V_{in1}');
110 hold on
111 plot(t, Vin2, 'g-','LineWidth', 2,'DisplayName', 'V_{in2}');
112 hold on
113 plot(t, raices, 'r-','LineWidth', 2,'DisplayName', 'V_{x}');
114 hold on
115 grid on;
```

```
116
117 % Agregamos etiquetas al grafico
118 xlabel('t (s)');
119 ylabel('V (V)');
120 title(['ESTADO INICIAL: M1(0) = ', num2str(M1(1)),' y M2(0)= ',
       num2str(M2(1))]);
121 legend('show', 'Location', 'northwest');
122 sgtitle('Puerta AND');
123
124 % Escogemos el nombre del grafico y lo guardamos
125 namefig=('AND_voltajes.fig');
126 savefig (namefig);
127
128 % Representamos las estados de M1 y M2 en funcion del tiempo
129 figure(2);
130 subplot(2,2,inicio)
131 plot(t, M1, 'b-','LineWidth', 2,'DisplayName', 'M1');
132 hold on
133 plot(t, M2, 'g-','LineWidth', 2,'DisplayName', 'M2');
134 hold on
135
136 % Agregamos etiquetas al grafico
137 xlabel('t (s)');
138 ylabel('Estado (1 LRS, 0 HRS)');
139 title(['ESTADO INICIAL: M1(0) = ', num2str(M1(1)),' y M2(0)= ',
       num2str(M2(1))]);
140 legend('show', 'Location', 'northeast');
141 sgtitle('Puerta AND');
142
143 % Escogemos el nombre del grafico y lo guardamos
144 namefig=('AND_estados.fig');
145 savefig(namefig);
146
147 % Representamos en un grafico 3D la puerta AND
148 figure(3)
149 stem3(Vin1,Vin2,raices,'o','MarkerSize',10,'LineWidth',1)
150 xlabel('V_{in1} (V)');
151 ylabel('V_{in2} (V)');
152 zlabel('V_{X} (V)');
153 title('Gate AND SUMMARY');
154 hold on
155
156 % Escogemos el nombre del grafico y lo guardamos
157 namefig=('AND_3d.fig');
158 savefig(namefig);
159 hold on
160
161 end
```

funcionANDideal.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    FUNCION PARA HALLAR DE FORMA ANALITICA EL VALOR DE V_x EN LA
     PUERTA AND IDEAL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% La diferencia respecto al programa funcionAND es que se han
   considerado las expresiones para la corriente de un memristor
   ideal en el que el ratio Roff/Ron es grande

% La salida de la funcion es la tension Vx
% La entrada de la funcion son las tensiones de puerta, Vin1 y
   Vin2, y los estados de los memristores M1 y M2
% Si M1=0 entonces M1 esta en estado RESET
% Si M1=1 entonces M1 esta en estado SET
% De forma analoga para M2
function [Vx] = funcionANDideal(Vin1, Vin2, M1, M2)

% Parametros de la caracteristica I-V del memristor ideal
Ron=100;
Roff=1E4;

% Establecemos el valor de la resistencia de M1
if M1==1
   R1=Ron;
else
   R1=Roff;
end
% Hacemos lo mismo con M2
if M2==1
   R2=Ron;
else
   R2=Roff;
end

% Hallamos Vx mediante el divisor de tension
Vx=Vin2+(Vin1-Vin2)*R2/(R1+R2);

end
```

evolucionANDideal.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    PROGRAMA PARA MOSTRAR LA EVOLUCION TEMPORAL DE LA PUERTA AND
     IDEAL
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Se trata del mismo programa que evolucionAND con la diferencia
   de que se llama al programa funcionANDideal en lugar de
   funcionAND

```

```
% Definimos el numero de puntos en cada eje
N=1500;
% Hacemos la malla del eje horizontal
t=linspace(0,9,N);
% Escogemos la forma de la onda Vin1
onda1 = @(t) ((t<3)).*0 +...
             ((t>=3) & (t<=6)).*1+...
             ((t>6) & (t<9)).*2 ;
% Escogemos la forma de la onda Vin2 con la condicion de que Vin2
   <=Vin1 (en otro caso bastaria intercambiar los papeles de M1 y
   M2)
onda2 =  @(t) ((t<3)).*0+...
              ((t>=3) & (t<4)).*0 + ...
              ((t>=4) & (t<5)).*1+...
              ((t>=5) & (t<6.1)).*1+...
              ((t>=6.5) & (t<7)).*0 + ...
              ((t>=7) & (t<8)).*1+...
              ((t>=8) & (t<9)).*2;

% Parametros del memristor
vreset=-0.53;
vset=0.354;
% Definimos un parametro que nos permite modificar la amplitud de
   las ondas onda1 y onda2 sin cambiar su forma y sus proporciones
escala=6;

% Definimos las tensiones de entrada de la puerta AND
Vin1=onda1(t).*escala;
Vin2=onda2(t).*escala;

% El parametro "inicio" especifica la situacion en la que se
   encuentran inicialmente los memristores M1 y M2:
% inicio=1 ---> M1=RESET M2=RESET
% inicio=2 ---> M1=RESET M2=SET
% inicio=3 ---> M1=SET M2=SET
% inicio=4 ---> M1=SET M2=RESET

% Hacemos un bucle para representar todas las situaciones de M1 y
   M2
for inicio=1:4

% El vector raices contiene los valores de Vx para cada pareja de
   tensiones (Vin1,Vin2)
raices=zeros(1,N);

% Dependiendo del valor de "inicio" tenemos comportamientos
   diferentes de M1 y M2 (en cuanto a la posibilidad de que
   conmuten entre estados SET y RESET)
switch inicio

    case 1
```

```
49      % Si ambos estan en RESET entonces M1 va a permanecer en RESET
            pues Vin1>Vx pero M2 puede cambiar a SET
50          % Inicializamos los estados de los memristores
51          M1=zeros(1,N);
52          M2=zeros(1,N);
53          % Para cada pareja (Vin1,Vin2) calculamos el
                correspondiente valor de Vx
54          for i = 2:N
55              raices(i) = funcionANDideal(Vin1(i), Vin2(i), M1(i),
                    M2(i));
56              % Comprobamos si M2 cambia a SET para actualizar el
                    estado y volver a calcular el Vx correcto
57              if (raices(i)-Vin2(i))>=vset
58                 M2(i:N)=1;
59                 raices(i) = funcionANDideal(Vin1(i), Vin2(i), M1(i)
                       , M2(i));
60              end
61          end
62
63      case 2
64          M1=zeros(1,N);
65          M2=ones(1,N);
66
67          for i = 2:N
68             raices(i) = funcionANDideal(Vin1(i), Vin2(i), M1(i), M2
                    (i));
69             % En este caso no hay que comprobar nada porque M1 no
                    va a cambiar SET ni M2 va a cambiar a RESET
70          end
71
72      case 3
73          M1=ones(1,N);
74          M2=ones(1,N);
75
76          for i = 2:N
77             raices(i) = funcionANDideal(Vin1(i), Vin2(i), M1(i), M2
                    (i));
78              % Comprobamos si M1 cambia a RESET
79               if (raices(i)-Vin1(i))<=vreset
80                  M1(i:N)=0;
81                  raices(i) = funcionANDideal(Vin1(i), Vin2(i), M1(i
                        ), M2(i));
82              end
83          end
84
85      otherwise
86          M1=ones(1,N);
87          M2=zeros(1,N);
88
89          for i = 2:N
90             raices(i) = funcionANDideal(Vin1(i), Vin2(i), M1(i), M2
```

```
                (i));
91              % Comprobamos si M2 cambia a SET
92              if (raices(i)-Vin2(i))>=vset
93                  M2(i:N)=1;
94                  raices(i) = funcionANDideal(Vin1(i), Vin2(i), M1(i
                        ), M2(i));
95              end
96              % Comprobamos si M1 cambia a RESET
97              if (raices(i)-Vin1(i))<=vreset
98                  M1(i:N)=0;
99                  raices(i) = funcionANDideal(Vin1(i), Vin2(i), M1(i
                        ), M2(i));
100             end
101         end
102
103 end
104
105 % Representamos las tensiones Vin1, Vin2 y Vx en funcion del
        tiempo
106 figure(1);
107 subplot(2,2,inicio)
108 plot(t, Vin1, 'b-','LineWidth', 2,'DisplayName', 'V_{in1}');
109 hold on
110 plot(t, Vin2, 'g-','LineWidth', 2,'DisplayName', 'V_{in2}');
111 hold on
112 plot(t, raices, 'r-','LineWidth', 2,'DisplayName', 'V_{x}');
113 hold on
114 grid on;
115
116 % Agregamos etiquetas al grafico
117 xlabel('t (s)');
118 ylabel('V (V)');
119 title(['ESTADO INICIAL: M1(0) = ', num2str(M1(1)),' y M2(0)= ',
        num2str(M2(1))]);
120 legend('show', 'Location', 'northwest');
121 sgtitle('Puerta AND ideal');
122
123 % Escogemos el nombre del grafico y lo guardamos
124 namefig=('AND_ideal_voltajes.fig');
125 savefig (namefig);
126
127 % Representamos las estados de M1 y M2 en funcion del tiempo
128 figure(2);
129 subplot(2,2,inicio)
130 plot(t, M1, 'b-','LineWidth', 2,'DisplayName', 'M1');
131 hold on
132 plot(t, M2, 'g-','LineWidth', 2,'DisplayName', 'M2');
133 hold on
134
135 % Agregamos etiquetas al grafico
136 xlabel('t (s)');
```

```
137 ylabel('Estado (1 LRS, 0 HRS)');
138 title(['ESTADO INICIAL: M1(0) = ', num2str(M1(1)),' y M2(0)= ',
        num2str(M2(1))]);
139 legend('show', 'Location', 'northeast');
140 sgtitle('Puerta AND ideal');
141
142 % Escogemos el nombre del grafico y lo guardamos
143 namefig=('AND_ideal_estados.fig');
144 savefig(namefig);
145
146 % Representamos en un grafico 3D la puerta AND ideal
147 figure(3)
148 stem3(Vin1,Vin2,raices,'o','MarkerSize',10,'LineWidth',1)
149 xlabel('V_{in1} (V)');
150 ylabel('V_{in2} (V)');
151 zlabel('V_{X} (V)');
152 title('Ideal AND gate SUMMARY');
153 hold on
154
155 % Escogemos el nombre del grafico y lo guardamos
156 namefig=('AND_ideal_3d.fig');
157 savefig(namefig);
158 hold on
159
160 end
```

Bibliografía

[1] L. O. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, September 1971.

[2] R. S. Williams *et al.*, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.

[3] R. Tetzlaff, *Memristors and Memristive Systems*. Springer, 2014.

[4] Y. Xiao *et al.*, "A review of memristor: material and structure design, device performance, applications and prospects," *Science and Technology of Advanced Materials*, vol. 24, no. 1, February 2023.

[5] J. Shim *et al.*, "Artificial optic-neural synapse for colored and color-mixed pattern recognition," *Nature Communications*, vol. 9, no. 1, November 2018.

[6] S. Kvatinsky *et al.*, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, 2014.

[7] S. Kvatinsky *et al.*, "MAGIC—memristor-aided logic," *IEEE Transactions on Circuits and Systems*, vol. 61, no. 11, pp. 895–899, November 2014.

[8] S. Kvatinsky *et al.*, "MRL – memristor ratioed logic," *IEEE Cellular Nanoscale Networks and their Applications*, August 2012.

[9] X. Wang *et al.*, "High-density memristor-CMOS ternary logic family," *IEEE Transactions on Circuits and Systems*, vol. 68, no. 1, pp. 264–274, January 2021.

[10] W. Chen, W. Lu, B. Long, Y. Li, D. Gilmer, G. Bersuker, S. Bhunia, and R. Jha, "Switching characteristics of W/Zr/HfO2/TiN ReRAM devices for multi-level cell non-volatile memory applications," *Semiconductor Science and Technology*, vol. 30, no. 7, 2015.

[11] S. Dueñas, H. Castán, H. García, E. Miranda, M. González, and F. Campabadal, "Study of the admittance hysteresis cycles in

TiN/Ti/HfO2/W-based RRAM devices," *Microelectronic Engineering*, vol. 178, pp. 30–33, 2017.

[12] A. Rodriguez-Fernandez, C. Cagli, L. Perniola, E. Miranda, and J. Sune, "Characterization of HfO2-based devices with indication of second order memristor effects," *Microelectronic Engineering*, vol. 195, pp. 101–106, 2018.

[13] J. Niinisto, K. Kukli, M. Heikkila, M. Ritala, and M. Leskela, "Atomic layer deposition of high-k oxides of the group 4 metals for memory applications," *Advanced Engineering Materials*, vol. 11, no. 4, pp. 223–234, 2009.

[14] O. Ossorio, S. Poblador, G. Vinuesa, S. Duenas, H. Castan, M. Maestro-Izquierdo, M. G. Bargallo, and F. Campabadal, "Single and complex devices on three topological configurations of HfO2 based RRAM," *IEEE Latin America Electron Devices Conference (LAEDC)*, pp. 1–4, 2020.

[15] S. Lin, Y. Kim, and F. Lombardi, "CNTFET-based design of ternary logic gates and arithmetic circuits," *IEEE Transactions on Nanotechnology*, vol. 10, no. 2, pp. 217–225, March 2011.

[16] H. Zhang *et al.*, "Implementation of unbalanced ternary logic gates with the combination of spintronic memristor and CMOS," *Electronics*, vol. 9, pp. 542–555, March 2020.

[17] P. C. Balla and A. Antoniou, "Low power dissipation MOS ternary logic family," *IEEE Journal of solid-state circuits*, vol. 19, no. 5, pp. 739–749, October 1984.

[18] M. Khalid and J. Singh, "Memristor based unbalanced ternary logic gates," *Analog Integrated Circuits and Signal Processing*, March 2016.