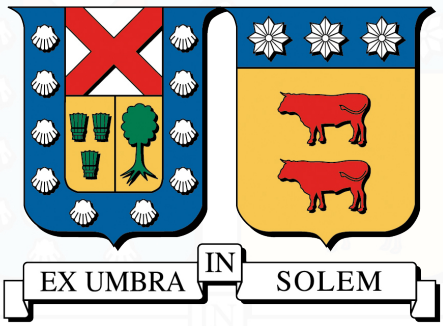
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
Valparaiso - CHILE



Desarrollo de Algoritmos de Síntesis Lógica Aplicados en Topologías de Memorias Resistivas Computacionales ReRAM

Carlos Fernández Hernández

MEMORIA PARA OPTAR AL TÍTULO DE
Ingeniero Civil Electrónico
y al grado de
Magíster en Ciencias de la Ingeniería Electrónica

PROFESOR GUÍA : Dr. Ioannis Vourkas

Septiembre 2022

RESUMEN EJECUTIVO

En la actualidad los sistemas computacionales ven cada vez más difícil la posibilidad de aumentar el desempeño y eficiencia energética. Lo anterior se puede relacionar a la imposibilidad de seguir con la miniaturización de los transistores, componentes básicos de cualquier sistema computacional, producto de que se están alcanzando límites físicos imposibles de sortear. De esta forma eventualmente los beneficios obtenidos de aumentar la concentración de transistores en una misma área se ven atenuados por los costos y/o nuevas problemáticas como, por ejemplo, las corrientes de fuga que aumentan el consumo de potencia. Por otro lado, las arquitecturas convencionales con las que se construyen estos sistemas tienen la característica de separar la memoria y la unidad de procesamiento de datos (CPU), lo que sumado a un dispar crecimiento de la capacidad de almacenamiento y la tasa de transferencia de datos ha producido un efecto de cuello de botella que limita el rendimiento de las CPU.

Para enfrentar estos problemas se han propuesto una serie de soluciones emergentes, dentro de las cuales destaca la posibilidad de realizar computación en la memoria. Esta forma de hacer computación precisamente busca eliminar cualquier tipo de transferencia entre una memoria y CPU fusionando ambos elementos en una sola estructura.

Dentro de las estructuras con capacidad de implementar computación en la memoria se encuentran las memorias resistivas o ReRAM, las cuales se componen de elementos de conmutación resistiva denominados *memristors*. Estos componentes fueron teorizados durante el siglo XX, pero no fue hasta 2008 cuando aquella teoría se vinculó por primera vez con observaciones experimentales en datos obtenidos de *memristors* reales. La principal característica de estos elementos corresponde a la de poder cambiar su resistencia a partir de la aplicación de una señal de voltaje o corriente. De esta forma se pueden utilizar los *memristors* como celdas de memoria, en las los datos almacenados están codificados en el valor de resistencia de los dispositivos. Una de las ventajas del uso de esta tecnología, en comparación con tecnologías convencionales de memorias RAM y Flash, es su alta densidad de integración y la posibilidad de almacenar más de un bit de información en cada celda, de manera no volátil. En conjunto con su aplicación como memoria, se han desarrollado también mecanismos para computar compuertas lógicas utilizando estos dispositivos, permitiendo así la implementación de computación en arreglos de memoria ReRAM.

De esta forma este trabajo se enfoca en dos temas principales. En primer lugar, en el diseño y simulación de una memoria ReRAM que permite implementar las operaciones propias de una memoria las cuales son la escritura y lectura de datos, así como también la computación de compuertas lógicas. Para esto se seleccionó un estilo de lógica con *memristors* que fuera compatible con la topología de memoria utilizada y se diseñó un circuito de control de la memoria que permitiera una implementación funcional del sistema. En segundo lugar, se desarrolla un flujo de síntesis, una herramienta fundamental que permite la computación de funciones lógicas arbitrarias sobre la memoria computacional. El objetivo de esta herramienta es la de establecer un flujo de ejecución de la función lógica a sintetizar, que se traduce en lograr la asignación de recursos de la memoria, así como también establecer los ciclos de ejecución a partir de ciertas restricciones de espacio o de latencia. Para el diseño de esta herramienta se consideran el impacto de la latencia de cómputo de las funciones lógicas a ejecutar y el área de memoria utilizada, logrando grandes mejoras considerables en comparación a otras herramientas ya existentes para computación en ReRAM. Estos ultimo también suelen presentar problemas respecto a los tiempos de síntesis, utilizando cándidas de tiempo poco realistas para aplicaciones reales (del orden de días). De esta forma el trabajo desarrollado propone una estrategia iterativa para enfrentar este problema evitando un impacto sobre la latencia de cómputo de la función a sintetizar.

Finalmente, en este trabajo se utilizan *memristors* reales fabricados por la empresa Knowm para la implementación de un ejemplo práctico de la memoria ReRAM. Para el desarrollo de esta parte se debió estudiar circuitos alternativos a los simulados que permitieran evadir problemáticas que surgen debido a la variabilidad de los dispositivos, así como rangos de resistencia cercanos (R_{OFF}/R_{ON} ratio de solo un orden de magnitud). Se logra implementar una fila de memoria ReRAM con sus respectivos circuitos de control y dos celdas de memoria, dentro de la cual se ejecutan las operaciones propias de una memoria computacional mostrando su funcionalidad.

ABSTRACT

Nowadays, improving the performance and energy efficiency in computational systems is becoming increasingly harder. The former statement is supported by the current difficulty to keep miniaturizing the transistors, basic components of any computational system, this because today's technology is reaching physical constraints in their fabrication. In this way the benefits of increasing the density of transistor in electronic systems are mitigated by the monetary cost of fabrication, or performance issues like the leakage currents which increase the energy consuption. On the other hand, conventional architectures like the Von Neuman, used in most computer systems, have the characteristic of an independant CPU and memory unit, this added to unequal evolution in memory capacity and data transfer bandwidth between memory and CPU, which has produced the so called Von Neuman bottleneck affecting the system performance.

To face the above mentioned issues a large number of emerging solutions have been proposed, being one of them the in memory computating (IMC). This type of computation aims to eliminate the need for data transfer between memory and CPU. Some structures that allow the implementation of IMC are the resistive memories or ReRAM memories, which are made of resistive switching devices called *memristors*. The main feature of such devices is that they have the ability to change their internal state when a voltage or current signal is applied to them. Therefore, in resistive memory applications, data can be stored in form of resistance. One of the benefits of using these devices is the high integration density they have in comparison with conventional memory technolgy like SRAM and DRAM, as well as the ability to store more than 1 bit of information per memory cell (multi-level memory). Together with their apliaction as memory devices, further strategies to compute logic gates have been developed using *memristor* devices, alowing the computation of logic inside the resistive memory arrays.

In this context, this work focuses on two main topics. The first one concerns the design and simulation of a functional ReRAM memory, that allows both the operations of a memory (read and write) and the computation of logic gates. In this direction, a style to do logic with *memristor*, that it was compatible with the selected memory architecture, was selected from the literature. Moreover, peripheral control circuit was designed in a way that allowed the implementation of logic operations, alongside memory operations, where the logic inputs were the data stored in the memory cells in form of resistance. On the other hand, a syntheis flow was developed, a fundamental tool that allows the computation of arbitrary logic function inside the ReRAM memory. The objective of this tool is to define the sequence and timing when the logic gates of a logic function will be computed, and to also asign the resources of the memory that will be used for the computation according to area and latency constraints. For the design of this synthesis tool the latency in the computation of logic functions was the main focus and great imporvement compared to results reported about other existing tools was achived. These existing tools also present issues regarding the synthesis time, some of them requiring an impractical amount of time to synthesize (more than one day). Thus, this work proposed an iterative approach to face the syntehsis problem, avoiding this way the overhead produced over the computing latency of the logic function to synthetize and the impractical synthesis times previously seen in other relevant works.

Finally in this work real *memristors* manufactured by Knwom Inc. were used in a proof-of-concept implementation of ReRAM memory. For the development of this part alternative circuits were studied that could adapt to the specific variability of the devices used as well as the characteristics of them like their resistance values and voltage thresholds. A single row of the memory array with two cells was implemented with their peripheral control circuit. In this implemented circuit write and read operations of a memory were carried out as well as the computation of NOR gates proving the correct functionality of the proposed design.

Índice de Contenidos

0. Introducción **1**
0.1. Contexto Tecnológico Actual . . . 1
0.1.1. Tecnologías Emergentes . . . 5
0.1.2. Límites del Trabajo Propuesto . . . 6
0.1.3. Objetivos del trabajo propuesto . . . 8
0.1.4. Contribución de la tesis . . . 9
0.1.5. Organización del documento . . . 9

1. El Memristor Como Elemento de Memoria **11**
1.0.1. Teoría sobre el memristor . . . 11
1.0.2. Modelos Reales de *memristor* . . . 14
1.0.3. Selección de modelo para aplicación en la memoria . . . 16
1.0.4. Modelo Pershin . . . 16
1.0.5. Modelo VTEAM . . . 17
1.0.6. Modelo de simulación SPICE . . . 19

2. Computación Lógica con *Memristors* **22**
2.0.1. Clasificación de los estilos de lógica . . . 22
2.0.2. Estilo Ratioed Logic . . . 23
2.0.3. Simulando el estilo *"Ratioed Logic"* . . . 25

3. Diseño de Arquitectura de Memoria ReRAM **27**
3.1. Arreglo *Crossbar* de *Memristors* . . . 27
3.1.1. Implementación de computación en la memoria . . . 29
3.2. Evaluación de la fiabilidad de en operaciones lógicas y de memoria . . . 35
3.3. Propuesta de topología alternativa . . . 36
3.4. Implementación practica topología alternativa . . . 38
3.4.1. Primer trabajo practico con *Memristors* . . . 41
3.4.2. Curvas características de los dispositivos . . . 42
3.4.3. Niveles de resistencia y selección . . . 45
3.4.4. Pruebas de computación lógica y de memoria . . . 48
3.4.5. Exploración de dispositivos después de re-forming . . . 52

4. Desarrollo de Herramienta de Síntesis **57**
4.1. Herramientas de síntesis . . . 57
4.1.1. SIMPLE MAGIC . . . 58
4.1.2. Herramienta de síntesis general . . . 59
4.1.3. Herramienta de síntesis SAID (Supergate-Aided Logic) . . . 60
4.2. Herramienta de síntesis propuesta . . . 62
4.2.1. Consideraciones iniciales . . . 62

4.2.2. Desarrollo del algoritmo de síntesis 63
4.2.2.1. Procesamiento de la descripción lógica a sintetizar 64
4.2.2.2. Traspaso descripción NOR/NOT a estructura de grafo 64
4.2.2.3. Estructura de las restricciones para la distribución de compuertas en memoria 65
4.2.2.4. Resolución del problema de mapeo de compuertas lógicas acorde a restricciones 67
4.2.3. Pruebas seleccionadas de funcionamiento del sintetizador 69

5. Caracterización de la Herramienta de Síntesis 73
5.1. Definición de Latencia Real 73
5.2. Simulación de la Salida del Sintetizador 75
5.3. Comparación del Desempeño del Sintetizador con el Estado del Arte 78
5.4. Caracterización de la latencia 80
5.5. Caracterización Tiempo de Síntesis 81
5.6. Limitación de Espacio y Latencia de Computo 82
5.7. Compuertas NOR de Entradas Variables 84

6. Conclusiones y Discusión 87
6.1. Trabajo Futuro 88

Bibliografía 89

Índice de Tablas

1. Comparación desempeño memorias convencionales y memorias emergentes no-volátiles . . 6

1.1. Tabla de parámetros utilizados con el modelo VTEAM 18

2.1. Relación entre entradas lógicas y voltajes de salida con la implementación de una compuerta NOR2 con el estilo *"Ratioed Logic"* 24

3.1. Valor de las señales de control necesarias para las distintas operaciones de memoria y computación lógica 32
3.2. Umbrales de voltaje de los dispositivos *memristors* utilizados, de acuerdo a su hoja de datos. 40
3.3. Tabla de componentes utilizados 48

5.1. Numero de columnas utilizadas por nivel 76
5.2. Operaciones por ciclo y posiciones del arreglo involucrados. Colores identifican cada nivel en donde el color de los ciclos del 1 al 3 identifica al nivel 0, el de los ciclos 4 a 6 al nivel 1 y así sucesivamente. 76
5.3. Desempeño flujo de síntesis propuesto y comparación con SIMPLE MAGIC utilizando conjunto LGsynth91 78
5.4. Desempeño flujo de síntesis propuesto y comparación con SAID utilizando conjunto IWLS'93 79
5.5. Tiempos de síntesis de la herramienta general sin acelerar 82
5.6. Tiempos de síntesis de la herramienta general acelerada y herramienta propuesta 82

Índice de Figuras

1. Evolución de la cantidad de transistores en un procesador 2
2. Tendencia de la evolución de la densidad de potencia 2000-2020 3
3. Diagrama general de la arquitectura de Von Neumann 3
4. Diagrama de la jerarquía de memorias 4
5. Circuitos celdas de memorias de acumulación de carga 4
6. Objetivo del proceso de síntesis desde función lógica a distribución en crossbar 8

1.1. Relación entre variables eléctricas y dispositivos fundamentales 11
1.2. Curva v-i característica de los *memristors* 13
1.3. Comportamiento bipolar reversible de un dispositivo de conmutación resistiva de escala nanométrica TiO_{2-x} 14
1.4. Modelo de memristor presentado por grupo HP 15
1.5. Salidas de de las pruebas de computación 17
1.6. Comportamiento de *memristor* considerando modelo VTEAM parámetros seleccionados . . 19
1.7. Distintos modelos de representación del *memristor* en modelos SPICE 20

2.1. Ejemplo de distintos estilos de lógica 22
2.2. Implementación del estilo *"Ratioed Logic"* 24
2.3. Implementación compuerta NOT y buffer del estilo *"'Ratioed Logic"* 25
2.4. Simulacion LTspice compuerta NOR2 estilo *Ratioed Logic* 26

3.1. Arreglo *crossbar* de *memristors* 27
3.2. Topología de memoria resistiva 1T1R 28
3.3. Topología que permite la implementación del estilo *Ratioed Logic* en una fila de una memoria ReRAM 29
3.4. Diseño funcional de una fila de un arreglo de memoria resistiva 30
3.5. Diseño funcional de una fila de un arreglo de memoria resistiva 30
3.6. Diseño funcional de una fila de un arreglo de memoria resistiva 31
3.7. Diseño funcional de una fila de un arreglo de memoria resistiva 31
3.8. Diseño funcional de una fila de un arreglo de memoria resistiva 32
3.9. Diseño funcional de una fila de un arreglo de memoria resistiva 32
3.10. Simulación ejecución operaciones sobre línea de memoria resistiva 33
3.11. Topología de memoria resistiva funcional 34
3.12. Evolución diferencia de voltaje ΔV_{OL} 36
3.13. Circuito amplificador sumador con *memrsitors* 37
3.14. Circuito amplificador sumador con *memrsitors* 38
3.15. Estructura de los *memristors* SDC 39
3.16. Medición de *memristors* 39
3.17. Configuración circuito para medición de *memristor* en protoboard y pines del Analog Discovery 2 41
3.18. Ejemplo de procesos de forming en un *memristor* 42
3.19. Ejemplo de procesos de forming en un *memristor* 43

3.20. Ejemplo de procesos de forming en un *memristor* . . . 44
3.21. Pruebas SET y RESET dispositivos . . . 45
3.22. Pruebas SET y RESET dispositivos . . . 46
3.23. Pruebas SET y RESET dispositivos . . . 46
3.24. Pruebas SET y RESET dispositivos . . . 47
3.25. Configuración circuito para pruebas de computación y esquemático circuital . . . 48
3.26. Entradas a los dispositivos utilizados en las pruebas de computación . . . 49
3.27. Salidas de las pruebas de computación . . . 50
3.28. Voltaje de salida de los proceso de cómputo de compuerta NOR . . . 50
3.29. Salidas de de las pruebas de computación . . . 51
3.30. Salidas de de las pruebas de computación . . . 51
3.31. Curva característica dispositivos M3 y M4 posterior al re-forming . . . 52
3.32. Salidas de de las pruebas de computación . . . 53
3.33. Salidas de de las pruebas de computación . . . 53
3.34. Entradas a los dispositivos utilizados en las pruebas de computación con dispositivos reformados . . . 54
3.35. Voltaje Voltaje de salida producido por las 50 repeticiones de las pruebas de computación con dispositivos re-formados . . . 54
3.36. Voltaje de salida producido por las 50 repeticiones de las pruebas de computación con dispositivos reformados, solo voltajes de lectura . . . 55
3.37. Histograma que muestra la distribuían de los voltajes de salida para el total de la prueba con dispositivos reformados . . . 55
3.38. Voltaje Voltaje de salida del comparador producido por las 50 repeticiones de las pruebas de computación con dispositivos reformados . . . 56

4.1. Esquema general de síntesis . . . 57
4.2. Flujo de síntesis de la herramienta SIMPLE MAGIC . . . 58
4.3. Flujo de síntesis de la herramienta general . . . 60
4.4. Flujo de síntesis de la herramienta SAID . . . 61
4.5. Ejemplo de mapeo de un Full-Adder de 1 bit utilizando el flujo de síntesis SAID . . . 61
4.6. Circuito lógico NOR/NOT de compuerta XOR y su representación en grafo . . . 62
4.7. Flujo de síntesis propuesto . . . 63
4.8. Compuerta XOR representación grafo y estructura diccionario . . . 65
4.9. Flujo de asignación de posiciones en cada nivel para síntesis de función XOR . . . 69
4.10. Circuitos personalizados para pruebas de flujo de síntesis . . . 70
4.11. Resultado de la síntesis del circuito escalonado . . . 71
4.12. Resultado de la síntesis del circuito incremental . . . 71

5.1. Ejemplo de la transición en la disposición de la memoria . . . 73
5.2. Manejo de salidas con operaciones intermedias en el arreglo . . . 74
5.3. Proceso de síntesis circuito Full-Adder 1 bit . . . 75
5.4. Señales y estados de la memoria durante simulación computación Full-Adder de 1 bit. . . . 77
5.5. Pruebas de caracterización de la latencia del flujo de síntesis desarrollado . . . 81
5.6. Pruebas de caracterización del tiempo de síntesis del flujo de síntesis desarrollado . . . 81
5.7. Ejemplo de asignación de compuertas con celdas de memoria ocupadas . . . 84
5.8. Ejemplo situación con compuertas NOR2 y NOR4 . . . 85

0 | Introducción

0.1. Contexto Tecnológico Actual

Sin lugar a dudas, no hay invención moderna que haya tenido más impacto en nuestras vidas que la del transistor. Desde celulares a televisores, este dispositivo ha sido clave para el desarrollo de la tecnología como la conocemos al ser pieza fundamental para la construcción de los sistemas digitales. La historia de este dispositivo comienza en 1947 cuando Walter Brattain y John Bardeen de Bell Labs, pertenecientes al grupo de William Shockley, tuvieron éxito en sus experimentos con un primitivo transistor de contacto, dando así comienzo a la historia de la micro-electrónica. A partir de esto, el transistor evoluciona permitiendo que en 1961 Robert Noyce presentase al mundo el primer circuito integrado, el cual estaba compuesto por cuatro transistores y dos resistencias. El desarrollo de estos dispositivos no se detiene, y en 1961 se inventa el transistor de efecto de campo metal-óxido-semiconductor (metal–oxide–semiconductor field-effect transistor - MOSFET) que permite una mayor integración de estos dispositivos y sienta la bases para el rápido desarrollo de la industria de circuitos integrados. Noyce, en conjunto con Gordon Moore (ambos habían trabajado previamente con William Shockley), funda Integrated Electronics Corp, mejor conocido como INTEL, empresa que en 1971 presenta el primer microprocesador producido para distribución comercial de la historia, denominado INTEL 4004. Esta CPU de 4-bits contaba con la increíble cantidad de 2300 transistores MOSFET fabricados a partir de tecnología de 10μm.

Con toda seguridad, la miniaturización de los circuitos electrónicos fue un logro gigantesco y, a partir de aquí, comienza a desarrollarse de manera explosiva el campo de la computación, buscando siempre mejorar la capacidad de procesamiento en términos de energía y velocidad, además de capacidad de almacenamiento. En 1965, una época en la que aún la micro-electrónica estaba concentrada en el ambiente militar, Gordon Moore proponía que el futuro de la industria estaba en la integración de más componentes en una misma área de chip, lo que lograría una reducción de costes notable y acercaría esta tecnología al común de las personas. Bajo estas consideraciones, y viendo la capacidad de los ingenieros de la época, Moore realiza su famosa declaración diciendo que *"El número de transistores en un circuito integrado se duplicará cada dos años"* [1]. Esta aseveración que no contaba con un sustento empírico, se transforma en una motivación para toda la industria. Tan solo diez años después demuestra ser cierta, adquiriendo carácter de ley y naciendo así la *"Ley de Moore"*.

Cincuenta y cinco años han pasado desde que se postuló la *"Ley de Moore"* y se ha ido cumpliendo año tras año. Precisamente, en la figura 1 se observa la evolución de la cantidad de transistores en un microprocesador por año. A modo de perspectiva, de este crecimiento se debe considerar que el último procesador anunciado por Apple en 2021, el Apple M1 Max, cuenta con 57.000.000.000 de transistores utilizando una tecnología de 5 nm, lo que significa un incremento sustancial desde el INTEL 4004 y su tecnología de 10μm. Como se puede intuir, la miniaturización de los transistores es cada vez más costosa de realizar, lo cual resulta entendible al considerar que los tamaños de los transistores están llegando al rango de los átomos (aproximadamente 25 átomos de silicio equivalen a 5 nm). De esta forma, actualmente se habla de la desaceleración de la *"Ley de Moore"*, ya que seguir disminuyendo el tamaño de los transistores se traduce en grandes costos de los procesos de fabricación, además de que se están alcanzando los limites fundamentales que impone la física. Por tanto, este postulado está dejando de cumplirse con la misma intensidad que antes.

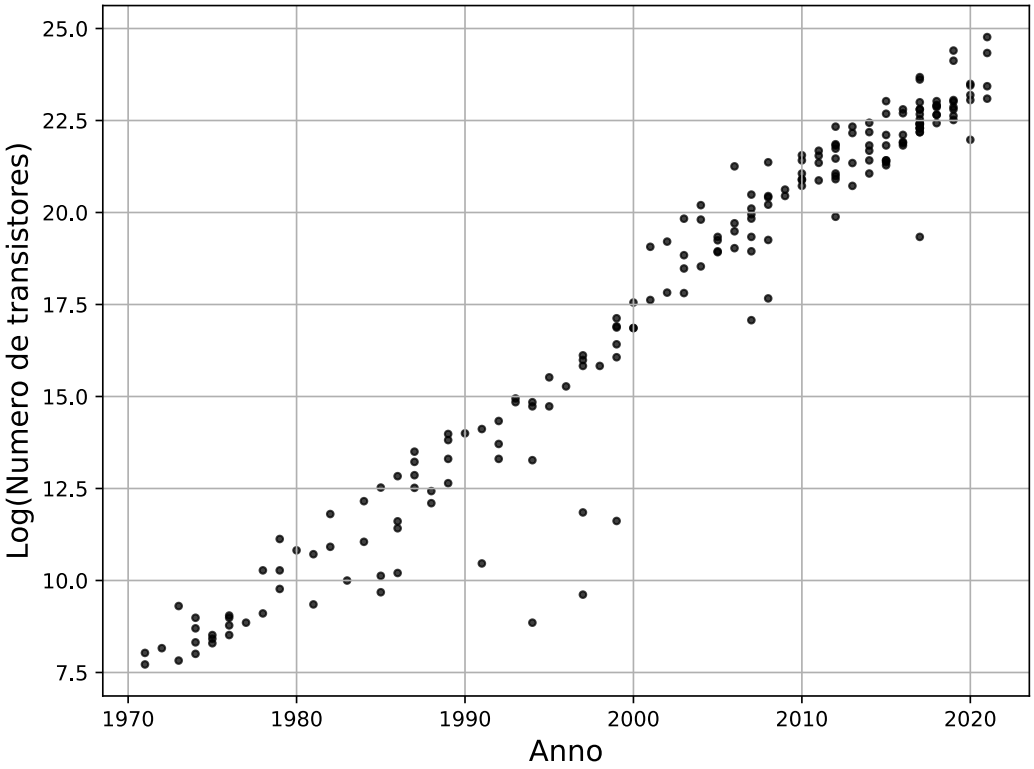


Figura 1: Evolución de la cantidad de transistores en un procesador[2]

El escalamiento de los transistores no solo significa una disminución en los costos de los procesadores, sino que además deriva en un aumento del desempeño de estos. Robert H. Dennard realiza en 1974 [3] la observación de que, a medida que los transistores se hacen más pequeños, también lo hacen la corriente y voltaje de éstos. De esta forma, y considerando que igualmente se aumenta la cantidad de transistores en una misma superficie, se tiene que la densidad de potencia se mantiene constante con la disminución del tamaño de los transistores. Esto significa una era de crecimiento en la capacidad de los procesadores. Al aumentar el número de transistores sin la necesidad de un aumento en el consumo, se logra que estos se vuelvan más eficientes en relación con la energía. Naturalmente, esta situación llega a su fin debido a que el tamaño de los transistores se hace lo suficientemente pequeño como para generar corrientes de fuga, pérdidas que empezaron a predominar frente a la potencia activa del sistema. Es así como la densidad de potencia se deja de conservar entre los cambios de tecnología de los transistores. Esta idea se presenta en la figura 2, en donde se muestra la tendencia de crecimiento de la densidad de potencia y la miniaturización de los transistores en el período 2000-2020. Se hace evidente que alrededor del año 2004 la *"Ley de Dennard"* comienza a dejar de cumplirse.

Con el fin del escalamiento de la potencia acorde a lo propuesto por Robert H. Dennard y con la densidad de potencia como nueva limitante, se busca innovar en la forma de incrementar el desempeño de los chips sin perjudicar su eficiencia. Un impacto directo de esto fue la imposibilidad de seguir aumentando la frecuencia del reloj en los procesadores debido a que esto induce un aumento de la potencia disipada por factores térmicos. De esta manera, para lograr mejoras en el rendimiento de los procesadores y eficiencia energética se plantea la posibilidad de tener una gran cantidad de los transistores del chip apagados u operando a frecuencia bajas en un fenómeno que se le denomina *"Dark Silicon"*[4]. Otra forma de sortear este problema fue la construcción de procesadores de más de un núcleo (Multi-Core Processing), dando el pase a sistemas de computación en paralelo. No obstante, el impacto que tiene la cantidad de núcleos incluidos en un procesador

está gobernado por la *"Ley de Amdahl"*[5], la cual relaciona la cantidad de núcleos con el aumento de desempeño y el porcentaje de operaciones a realizar por el procesador que son paralelizables.

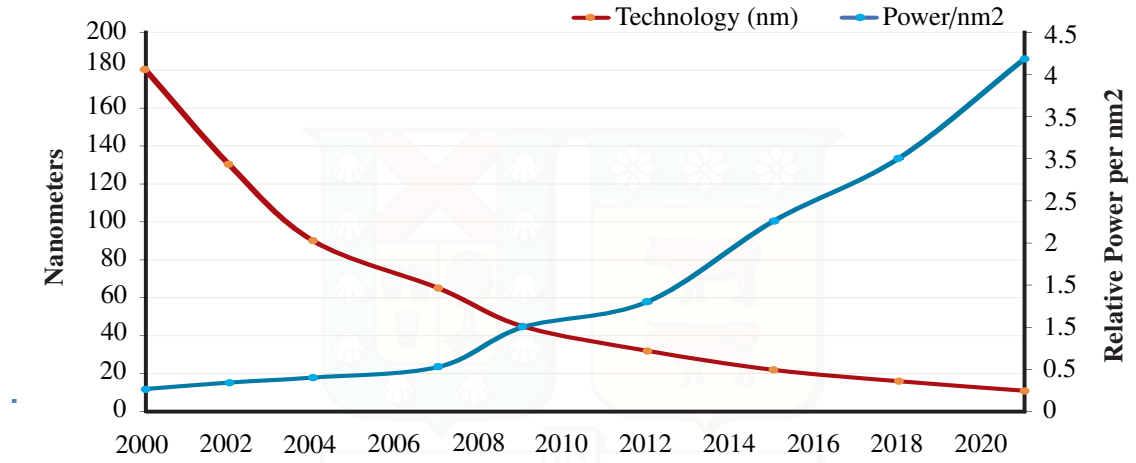


Figura 2: Tendencia de la evolución del tamaño de transistores y densidad de potencia de los chips[6]

El crecimiento de la capacidad de computación actual no esta solamente limitado por los procesadores y la tecnología con la que están estos fabricados, sino que también se debe tomar en cuenta la arquitectura utilizada por los sistemas computacionales actualmente. Durante años, la arquitectura predominante para el diseño de estos sistemas corresponde a la descrita por Von Neumann, la cual posee como principales características el tener la unidad de memoria y procesamiento separadas, y que la memoria almacena tanto datos como instrucciones. Esta arquitectura usualmente se presenta como se observa en la figura 3. La estructura de estos sistemas produce que una instrucción/dato tenga que ser trasladado desde la memoria al procesador para su respectiva ejecución/procesamiento. Es así como, debido al desarrollo dispar entre las frecuencias con las que trabajan los procesadores y la taza de transferencia entre memoria y CPU, es que ocurre un fenómeno que limita el desempeño de los procesadores, el llamado *cuello de botella de Von Neumann*. Este problema aumenta su relevancia si es que se considera algunas de las aplicaciones que se desarrollan actualmente como lo son Big Data, Cloud Computing e Inteligencia Artificial. Estas trabajan de manera intensiva con accesos a la memoria y debido a la gran cantidad de datos utilizados en estas aplicaciones es que se tienen un número considerable de accesos a la jerarquía de memoria. Esto implica una latencia significativa producto de la limitación impuesta por la taza de transferencia entre memoria y procesador, sin contar el consumo energético que se vuelve relevante.

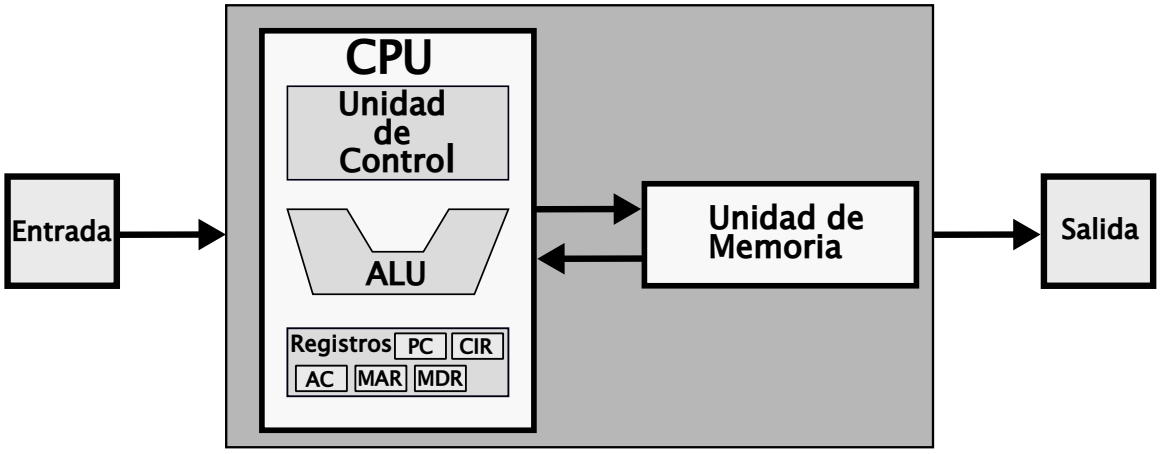


Figura 3: Diagrama general de la arquitectura de Von Neumann [7]

La eficiencia energética se vuelve especialmente importante en sistemas portátiles en los que la

duración de la batería es una de las principales preocupaciones. En [8] se realiza un estudio sobre el consumo de energía para algunas aplicaciones sobre un Chromebook, en el cual se muestra que en promedio el 67 % de la energía del sistema se utiliza en el movimiento de datos entre la memoria principal y las unidades de cómputo.

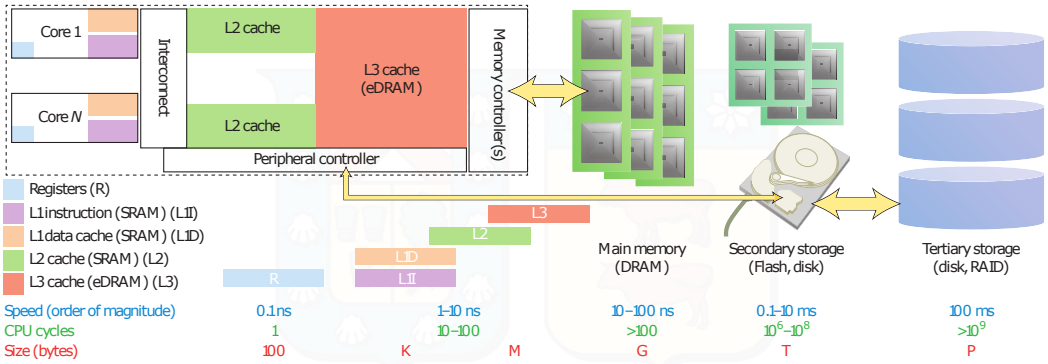


Figura 4: Composición de la jerarquía de memorias utilizadas en sistemas computacionales actuales. Extraída desde [9]

Las tecnologías de memorias actuales también son objeto de interés constante, considerando los problemas de acceso a memoria mencionados anteriormente. Estos se buscan sortear utilizando una combinación de distintas tecnologías de memoria con características complementarias, a lo que se le llama Jerarquía de Memoria.

Para entender el origen de este cuello de botella que ocurre debido a la disparidad entre taza de transferencia memoria-CPU y velocidad de procesamiento de la CPU, es importante tener en cuenta que existen distintos tipos de tecnologías con las cuales se pueden construir las memorias. Es más, el sistema de almacenamiento existente en los sistemas computacionales es una combinación de todas estas tecnologías organizadas de manera jerárquica. Se realiza de esta forma debido a que existe un equilibrio de costos/desempeño que se debe mantener. Esta estructura de memorias se establece como se muestra en la figura 4 y está compuesta por tecnologías que se encuentran en la categoría de acumulación de carga. En esta figura se detalla igualmente la cantidad aproximada de ciclos de la CPU para acceder a la memoria, así como también la capacidad de almacenamiento de las mismas. En este sentido se puede observar que mientras más cerca está la memoria de la CPU los tiempos de acceso disminuyen, no obstante, más costosa se vuelve la fabricación de esa tecnología. Lo anterior se compensa utilizando elementos con poco almacenamiento (en comparación a los elementos de almacenamiento masivo al final de la jerarquía). Por otro lado, aquellas memorias que están más alejadas de la CPU son de menor coste de fabricación, por lo que se pueden utilizar tamaños superiores. Sin embargo, éstas últimas tienen tiempos de acceso con órdenes de magnitud mayores que aquellas de niveles inferiores en la jerarquía. Por lo general, estas memorias están organizadas en arreglos de dos dimensiones con líneas horizontales y verticales denominadas Wordlines(WL) y Bitlines(BL), mediante las cuales se acceden al contenido de las celdas. Las distintas memorias de estos grupos se diferencian debido a la estructura del circuito de la celda que almacena un bit individual. En este sentido, destacan tres tipos de memoria cuyos circuitos se muestran en la figura 5.

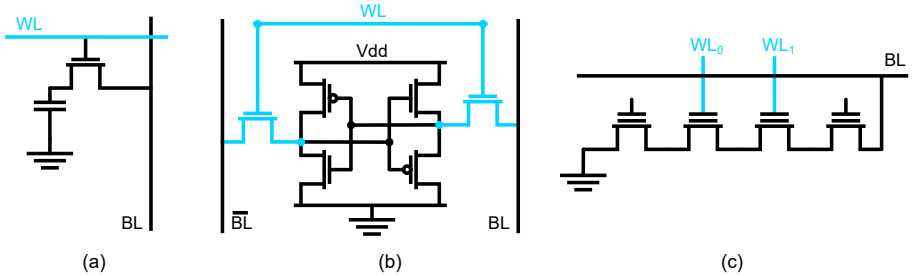


Figura 5: Circuito de las celdas de categoría acumulación de carga (a) DRAM (b) SRAM (c) NAND FLASH

En primer caso, se tiene la memoria RAM Dinámica (Dynamic RAM - DRAM), como se muestra en la figura 5-a. Para el almacenamiento de un bit, este tipo de memoria utiliza un circuito compuesto por un transistor en serie con un condensador cuya carga representa la información almacenada, siendo una memoria rápida y barata de implementar. Sin embargo, ésta tiene algunas desventajas. En primer lugar, es una memoria volátil producto de la utilización de un condensador cuya carga se mantiene por un tiempo acotado. Por lo tanto, se necesita de un proceso de refresco lo cual consume energía, aunque en la práctica la memoria no se esté utilizando. En segundo lugar, la lectura de los datos es destructiva por lo que posterior a esta operación se debe considerar un proceso de re-escritura en caso de ser necesario. Esta tecnología se utiliza para las memorias principales de los procesadores.

Por otro lado, se tiene la memoria RAM estática (Static RAM - SRAM) presentada en la figura 5-b. En este caso, el circuito que compone la celda de memoria está compuesto por un latch y dos selectores haciendo un total de seis transistores. Producto de este número de dispositivos el costo de fabricación es más elevado en comparación a la memoria DRAM que solo cuenta con un transistor y una capacitancia, no obstante, la SRAM a pesar de también ser una memoria volátil no necesita de un proceso de refresco debido a la existencia del latch (mientras exista alimentación). A su vez, tampoco necesita de operaciones de reescritura producto de que la lectura no es destructiva (no altera el estado lógico almacenado en la celda), lo cual hace que esta memoria sea más rápida respecto a la memoria DRAM. Esta tecnología de almacenamiento se utiliza principalmente para los registros y/o el caché de la CPU.

Por último, se tiene también las memorias Flash, las cuales a diferencia de los dos tipos de memorias anteriores entran en la categoría de memorias no-volátiles. En este caso, cada bit se almacena en un transistor, donde cada uno posee una puerta flotante que consiste en una capa especial que permite la acumulación de carga y, por ende, el funcionamiento como memoria. Existen distintos tipos de topología para una memoria Flash, una de estas corresponde a la NAND Flash que se presenta en la figura 5-c. La principal ventaja frente a las otras memorias es precisamente su característica de no-volátil, por lo cual mantiene la información a pesar de no estar conectada a la alimentación. Sin embargo, estas memorias son más lentas que las soluciones anteriores y los ciclos de escritura/lectura van dañando físicamente la celda de memoria de manera que cuenta con una cantidad máxima de ciclos de vida útil. Por estas características, este tipo de memoria está orientada a almacenamiento masivo y es una alternativa más costosa, pero con mejor desempeño en los HDD (Hard Disk Drive).

0.1.1. Tecnologías Emergentes

Como se ha ido introduciendo previamente, en la actualidad se está llegando a límites en distintos aspectos que constituyen los sistemas computacionales, de forma que se hace necesario romper con estas limitaciones para cumplir con la creciente demanda de desempeño que tienen las aplicaciones actuales. Para esto, los cambios deben ocurrir en distintas esferas, desde la arquitectura hasta el desarrollo de software, a nivel de almacenamiento/memoria y en la tecnología utilizada para la creación de estos componentes. En definitiva, se requiere un cambio de paradigma. Bajo este contexto es que ha tomado relevancia el concepto de computación en la memoria (In M emory Computing - IMC), idea que plantea eliminar la distancia entre procesador y memoria a un mínimo, donde el agente de cómputo es precisamente la memoria. Si bien esta idea se ha estudiado en múltiples ocasiones en el pasado[10], problemáticas como la dificultad de integrar estilos de lógica a la tecnología de memoria existente producen que el desarrollo de esta temática no obtuviese suficiente relevancia hasta ahora. Actualmente, para la implementación de este concepto la atención está dirigida a dispositivos de tecnología emergente con características de memoria no-volátil y naturaleza distinta a aquellas de acumulación de carga descritas previamente. Estos elementos se caracterizan por el hecho de que, de forma individual, constituyen celdas de memoria en las que el almacenamiento de la información va asociado al nivel de resistencia del dispositivo.

Existen tres tecnologías que entran en la categoría de memoria resistiva que son importantes de mencionar. En primer lugar, se tiene **Spin-transfer-torque magnetic random access memory (STT-MRAM)**. Las memorias RAM magnéticas (MRAM) trabajan con dispositivos Spin-transfer-torque (STT), los cuales están compuestos por metales ferromagnéticos basados en el concepto de dos capas de polarización magnética, en donde una de estas funciona a modo de referencia y la otra está libre para cambiar de estado. Es así como el

estado resistivo depende de que ambas capas se encuentren en paralelo o antiparalelo. El estado de esta capa es modificable utilizando pulsos de voltaje. En segundo lugar, se tienen las **Phase Change Memory (PCM)**. Estas son memorias compuestas por un dispositivo que basa su comportamiento en las propiedades de ciertos materiales (como $Ge_2Sb_2Te_5$) que puede cambiar de un estado de baja resistencia (estado cristalino) a un estado de alta resistencia (estado amorfo), utilizando calor inducido al material mediante señales de voltaje aplicadas al dispositivo. Por último, se tiene también las **Resistive Random Access Memory (ReRAM)**, un tipo de memoria compuesto por dispositivos denominados **memristors** (resistores con memoria), cuya composición consiste en un óxido de metal entre dos electrodos. De esta forma, la corriente que fluye a través del dispositivo tiene como consecuencia la formación de filamentos que aumenta la conductividad y, por ende, provoca la variación de resistencia.

Como se explica anteriormente, estas nuevas tecnologías de dispositivos comparten la característica de poder modificar su valor de resistencia. De este hecho, surge que tengan la funcionalidad de celda de memoria, en donde los valores extremos de resistencia se asocian a estados lógicos. Por tanto, leer el estado resistivo equivale a leer la información almacenada. En la tabla 1 se muestran las características de estas tecnologías de memoria emergentes en contraste con las tecnologías convencionales SRAM y DRAM, en donde se proyecta una mejora en varios aspectos. El caso de la memoria ReRAM muestra que tiene las mejores características para una mayor integración de todas las tecnologías que se presentan, así como todos los tiempos de escritura/lectura mejoran considerablemente en comparación a la tecnología DRAM y se mantienen en el mismo orden de magnitud de las SRAM. No obstante, en el único apartado en que las memorias emergentes parecieran tener un desempeño considerablemente peor es aquel que corresponde a energía de escritura. Esto se contrarresta al considerar que las tecnologías convencionales poseen otras desventajas, como lo son el caso de las corrientes fuga y los procesos de refresco en el caso de las memorias DRAM, sin contar que las tecnologías emergentes son memorias no-volátiles. Además de esto, la energía utilizada en los procesos de escritura, en el caso de las memorias ReRAM, se compensa al tomar en cuenta la posibilidad de almacenar hasta 7 bits de información en la misma celda.

	Tecnologías Convencionales				Tecnologías Emergentes		
			FLASH				
	SRAM	DRAM	NOR	NAND	STT-MRAM	PCRAM	ReRAM
Tamaño Celda	$> 100F^2$	$6F^2$	$10F^2$	$< 4F^2$(3D)	6~$50F^2$	4~30F^2	4~$12F^2$
Multibit	1	1	2	3	1	2	2-7
Voltaje	<1 V	<1 V	>10 V	>10V	<1.5 V	<3V	<3V
Tiempo Lectura	~1 ns	~10 ns	~50 ns	~10 μs	<10 ns	<10 ns	<10 ns
Tiempo Escritura	~1 ns	~10 ns	10μs - 1 ms	100μs - 1 ms	<10 ns	~50 ns	<10 ns
Retención	N/A	~64 ms	>10 y	>10 y	>10 y	>10 y	>10 y
Duración (ciclos)	>1e16	>1e16	>1e5	>1e4	>1e15	>1e9	>1e6~1e12
Energia Escritura (J/bit)	~1f	~10f	~100p	~10f	~0.1f	~10p	~0.1p

Tabla 1: Comparación desempeño memorias convencionales y memorias emergentes no-volátiles [11]-[12]

0.1.2. Límites del Trabajo Propuesto

Las tecnologías emergentes de memorias mencionadas anteriormente son la base que ayudará a romper con las barreras tecnológicas que están limitando el aumento del desempeño de los sistemas computacionales, mas éstas por sí solas no son suficientes. En particular, si es que se considera el desarrollo de computación en la memoria como una solución al cuello de botella de las arquitecturas Von Neuman, entonces en conjunto a estas nuevas tecnologías de memorias debe desarrollarse una arquitectura apropiada que sea capaz de afrontar las problemáticas características de estos dispositivos emergentes. El trabajo realizado en el contexto de esta Tesis en particular se enfoca en las memorias ReRAM compuestas por los dispositivos de conmutación resistiva conocidos también como **memristors**, que generalmente se organizan en topologías matriciales

conocidas como arreglos de barras cruzadas (crossbars). La razón por la cual se elige esta tecnología es porque ha mostrado ser prometedora y cuenta con desarrollo demostrado en la literatura en aplicaciones como memoria no-volátil, computación neuro mórfica y, por supuesto, computación en la memoria. Cabe aquí señalar, que se trata de una tecnología que cada vez está más cerca a la comercialización masiva, como demuestra el reciente anuncio de la empresa WeeBit Nano de Israel sobre su tecnología ReRAM [13].

El *memristor* es un elemento de dos terminales que es teorizado por primera vez en 1971 por Leon O. Chua[14]. Este trabajo no recibe mucha atención hasta que, en el año 2008, en los laboratorios de Hellwet-Packard, se logra la fabricación de un dispositivo compuesto de dióxido de titanio cuyo comportamiento se conecta precisamente con la teoría de Chua. Esto provoca un gran y renovado interés en este dispositivo que ya contaba con grupos estudiándolo décadas anteriores. Este nuevo componente, además de tener propiedades de un elemento de memoria no-volátil, también tiene otras cualidades como bajo consumo energético, compatibilidad con el proceso de fabricación CMOS y alta densidad de integración [15]. La integración de estos elementos se realiza en estructuras 3D denominadas arreglos crossbar, a partir de donde se pueden construir las memorias ReRAM. El diseño de estos arreglos de memorias no es trivial y deben considerarse varios aspectos, especialmente en lo que refiere a la topología que gobierna las operaciones comunes de una memoria. Uno de los factores más importantes con el cual se debe lidiar es precisamente el comportamiento del *memristor* y sus características internas. Es decir, tener en consideración aspectos como la taza de cambio de la resistencia, rango de valores de resistencia y umbrales de voltaje para la conmutación del dispositivo es fundamental para un diseño correcto. A lo anterior, se debe añadir también la existencia de variabilidad en estos dispositivos, la cual puede ser ciclo a ciclo o entre dispositivos. El primer caso ocurre cuando se tiene un dispositivo que ante una misma señal de voltaje se obtienen diferentes respuestas de parte de éste. Por otro lado, la variabilidad entre dispositivo alude a la diferencia entre elementos de un mismo chip que presentan distintos comportamientos. Ambos tipos de variabilidad afectan en mayor o menor medida a estos dispositivos y puede condicionar su funcionamiento.

Para completar la utilidad de este dispositivo como elemento orientado a aplicaciones de computación en la memoria, se deben idear formas de utilización de los dispositivos de almacenamiento de información para realizar operaciones lógicas. En ese sentido, se han desarrollado una serie de estilos propuestos en la literatura que permiten implementar lógica booleana con *memristors* [16]. La naturaleza de estos estilos y las ventajas que ofrecen varían de acuerdo a una serie de elementos que pueden ser: capacidad de implementación en crossbar, tolerancia a la variabilidad, latencia, codificación de las entradas/salidas en forma de resistencia o voltaje, etc.

No se puede explotar todos los beneficios de la computación en la memoria si es que no se logra computar funciones lógicas arbitrarias de manera organizada. Es por ello que se han ido desarrollando algoritmos de síntesis destinados a estas nuevas tecnologías de memorias que buscan llevar la descripción de una función lógica a un flujo de operaciones que la unidad de memoria sea capaz de implementar, tal de asegurar la ejecución de esta función lógica conforme a restricciones que pueden ser de latencia y/o espacio. Este flujo de operaciones deriva del proceso de mapeado realizado por la herramienta en el que se asigna los elementos de la memoria ReRAM que intervienen en el cómputo de cada compuerta lógica de la función a ejecutar, así como también el instante de ejecución. En este sentido, se puede resumir que el proceso de síntesis desarrollado en tecnologías ReRAM es una combinación de la síntesis lógica y síntesis física utilizada en sistemas computacionales convencionales, esto ya que no solo deben descomponer una función lógica arbitraria en una secuencia de operaciones lógicas compatibles con la arquitectura del sistema PIM, sino que igualmente se debe calcular las propiedades espacio-temporales de todas las operaciones lógicas necesarias para su ejecución dentro del arreglo de la memoria ReRAM. De esta manera, el proceso de síntesis se puede dividir en tres etapas generales. Se parte de un diseño de algún circuito lógico como se muestra en la figura 6-a, el cual posteriormente se transforma como se presenta en la figura 6-b a una representación de alguna estructura de datos que nos permita identificar las distintas operaciones necesarias para su ejecución. Finalmente, en la última etapa se asocian las operaciones necesarias con celdas del arreglo crossbar, esto acorde a las restricciones físicas que este tiene, tal como se refleja en la figura 6-c.

En la literatura se puede hallar una serie de estas herramientas que implementan distintos algoritmos dependiendo del estilo de lógica asumido y la topología de memoria diseñada. Por otro lado, un factor que ha tenido gran impacto en el desarrollo de estas herramientas corresponde al tiempo necesario para completar el

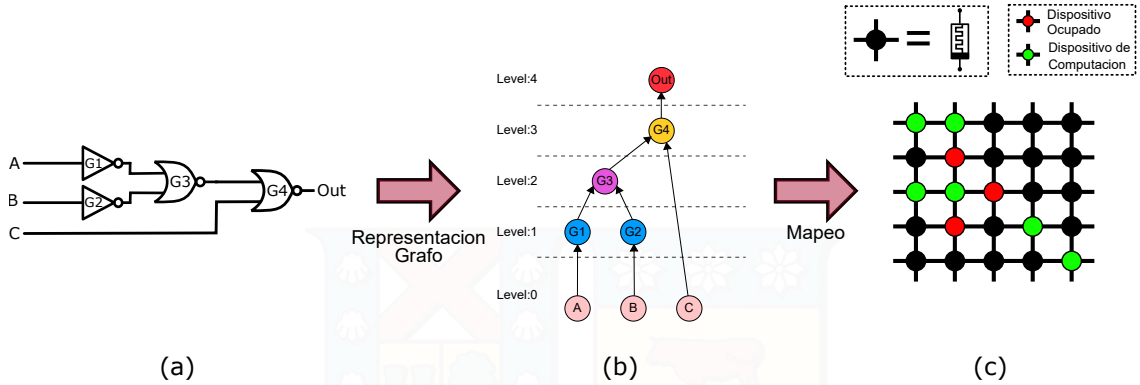


Figura 6: Objetivo del proceso de síntesis desde función lógica hasta distribución en crossbar (a) Función lógica a sintetizar. (b) Representación en estructura de datos de la función lógica a sintetizar. (c) Distribución final de la función lógica en el arreglo crossbar, en donde se tienen identificadas aquellas celdas de memoria que se ocupan en la computación lógica y aquellas ocupadas con datos relevantes.

proceso de síntesis. Existe una variedad de sintetizadores que utilizan elementos de optimización para realizar el proceso de mapeado, el cual es reducido a un problema de minimización de latencia/área. Es así como, a medida que aumenta el tamaño de la función lógica a sintetizar, en términos de número de operaciones lógicas, los tiempos de síntesis crecen considerablemente y es un problema a sortear en el desarrollo de estas herramientas. Por último, como se menciona antes, este flujo de operaciones debe ser traducido a un conjunto de instrucciones entendible por la unidad de control de memoria, de manera que esta genere las señales de control necesarias a la periferia para la ejecución de la función lógica determinada. Esto puede involucrar varias operaciones de escritura y/o de lectura del estado de los dispositivos que participan en el cómputo.

0.1.3. Objetivos del trabajo propuesto

En este trabajo, se busca diseñar los componentes necesarios para implementar aplicaciones de computación en la memoria utilizando la tecnología ReRAM acorde a lo expuesto previamente. De esta forma, como primer objetivo se plantea el diseño y posterior simulación de una topología de memoria computacional ReRAM compuesta por *memristors*. En este sentido, para lograr este objetivo se debe, en primer lugar, buscar un modelo de *memristor* que permita simular el funcionamiento de las celdas que componen la memoria. Para este modelo se deben considerar parámetros de retención, taza de cambio de la resistencia, relación resistencia mínima y máxima, entre otros. Otro elemento que se debe tomar en cuenta al momento de escoger un modelo y sus parámetros es el estilo de lógica utilizado, lo cual definirá el modo de realizar las operaciones de cómputo y posteriormente el diseño del circuito de control. Precisamente, el segundo punto que se debe completar para cumplir con este objetivo corresponde al diseño de los elementos del circuito que componen la periferia y que permitirán la ejecución de las operaciones de cómputo y las propias de una memoria, que son la escritura y la lectura de información. Este diseño se realiza utilizando el software LTspice tal que se pueda simular y comprobar el correcto funcionamiento de topología de memoria utilizando modelos conductuales de todos los dispositivos incluidos los *memristors*.

Como complemento al primer objetivo se pretende implementar una prueba de concepto del sistema de memoria con capacidad de computación, haciendo uso de componentes discretos y *memristors* reales de la empresa Knowm INC[17]. Para esto, se considera el rediseño del circuito de memoria de forma de orientarlo al control de dispositivos reales al alcance y que posean características distintas a las consideradas en el diseño original. Se busca lograr la ejecución de compuertas lógicas y operaciones de lectura y escritura.

A partir de la topología diseñada y comprobado su funcionamiento, el siguiente objetivo consta del desarrollo de una herramienta de síntesis apropiada. Para ello, se espera diseñar un flujo de síntesis y desarrollar los algoritmos de mapeo acorde al estilo de lógica con *memristors* utilizados. Este sistema

tomará una descripción de cualquier función lógica en algún formato determinado y entregará a la salida la distribución de las compuertas lógicas y el orden temporal de ejecución de éstas. Esta distribución será posteriormente procesada para obtener las señales de control que serán entradas a la topología de memoria diseñada.

El tercer objetivo consiste en la validación de la herramienta de síntesis desarrollada. Para esto, se debe generar descripciones lógicas con características de prueba que permitan verificar el correcto funcionamiento de la herramienta desarrollada. Seguidamente, se plantea utilizar la descripción de una función lógica conocida (por ejemplo, full-adder de 1 bit) como entrada al sintetizador, a partir de cuya salida se generan las señales de control necesarias para una simulación en la topología de memoria diseñada.

Como cuarto objetivo, se pretende medir el rendimiento de la herramienta de síntesis desarrollada para poder compararla con otras existentes en la literatura y determinar su impacto en este contexto. Para ello, se utilizará una serie de descripciones comunes en la literatura como benchmarks y se medirán parámetros como latencia de ejecución, área ocupada en el arreglo de memoria para la ejecución y tiempo de síntesis.

0.1.4. Contribución de la tesis

El trabajo desarrollado en esta tesis concluye con una serie de contribuciones a distintas conferencias desarrolladas durante el año 2022. La primera contribución corresponde a la realizada a la conferencia NANO 2022 mediante el articulo [18], en donde se presenta el diseño inicial del arreglo de memoria, un prototipo del flujo de síntesis y el algoritmo de mapeo de funciones lógicas al arreglo. Posteriormente se preparó el articulo [19] en donde se expone la implementación de una fila del arreglo de memoria diseñado, comparando la ejecución de funciones lógicas en este arreglo utilizando el estilo de lógica propuesto y comparado con otros existentes en la literatura. Este artículo se presentó a la conferencia VLSI-SoC 2022 en donde también se hizo una contribución al foro de estudiantes de postgrado mediante el articulo [20]. La última contribución a la fecha corresponde a la realizada a la conferencia DCIS 2022 mediante el articulo [21]. En este último artículo se profundiza en el circuito de la periferia diseñado para el control de la memoria ReRAM y en el cómo se realizan las operaciones de escritura, lectura y computo de compuertas, esto mediante los ejemplos de funciones lógicas y simulaciones en LTSpice.

0.1.5. Organización del documento

El desarrollo de este documento está estructurado con los capítulos que se describen a continuación:

- Capitulo 1: Se explica el comportamiento de los *memristores* y su funcionalidad como dispositivo de memoria. Se realiza también un estudio de los modelos existentes y se selecciona el apropiado para las simulaciones necesarias dentro del trabajo propuesto.
- Capitulo 2: Se estudian diferentes estilos de lógica con *memristores* y se selecciona el estilo más apropiado para realizar computación en la memoria. Se simula en LTspice el estilo de lógica adoptado y el comportamiento con el modelo de *memristor* seleccionado previamente.
- Capitulo 3: Se diseña la topología de memoria que permita implementar el estilo de lógica seleccionado. Específicamente, se diseña el circuito presente en la periferia del dispositivo y se simula su funcionamiento en LTspice. Se describe también la construcción del circuito de memoria en la práctica utilizando elementos discretos y *memristors* de Knowm Inc., y las pruebas de concepto de computación en la memoria realizadas en una porción reducida de la memoria a partir de los circuitos diseñados previamente.
- Capitulo 4: Se desarrolla la herramienta de síntesis para la ejecución de funciones lógicas arbitrarias en el arreglo de memoria. Se estudian previamente herramientas existentes y se define un flujo de síntesis acorde al estilo y topología diseñada. Se concluye con la validación de la herramienta diseñada mediante funciones lógicas con características determinadas.

- Capitulo 5: Se asocia las salidas de la herramienta de síntesis con el circuito de memoria diseñado para determinar el desempeño real en términos de latencia de la ejecución de una función lógica. Posteriormente, se realiza un estudio del desempeño de la herramienta de síntesis en términos de latencia, área utilizada en el arreglo de memoria para la ejecución de la función lógica y tiempos de síntesis, así como una caracterización de su comportamiento. Se emplea lo anterior para comparar con otras herramientas de síntesis disponibles en la literatura.
- Capitulo 6: Se presentan las conclusiones del trabajo realizado, además de explicitar las posibilidades de trabajo futuro en torno al tópico trabajado.

1 | El Memristor Como Elemento de Memoria

1.0.1. Teoría sobre el memristor

En el entorno de la teoría de circuitos, existen tres componentes fundamentales, el resistor, el condensador y el inductor. Cada uno de estos elementos logra relacionar dos de las cuatro cantidades eléctricas fundamentales, voltaje (V), corriente (i), flujo magnético (ϕ) y carga eléctrica (q), las que dan origen a 6 posibles ecuaciones que se relacionan entre sí. La primera de estas relaciones enlaza el voltaje y la corriente dando origen a la resistencia (R), expresado como $V = R * I$. Las siguientes dos relaciones corresponden a $i = L * \phi$ y $q = C * V$ para la inductancia (L) y capacitancia (C), respectivamente. Las últimas relaciones son conocidas en la física y están dadas por $V = \frac{d\varphi}{dt}$ además de $i = \frac{dq}{dt}$. Bajo este contexto, en 1971 Leon O. Chua plantea la existencia de un cuarto elemento fundamental que estaría definido por la relación entre el flujo y la carga eléctrica no considerado previamente, naciendo así el concepto de *Memristor* (M). Este dispositivo de dos terminales presenta la cualidad de ser una resistencia no-lineal con memoria [14]. De esta forma, se completaba el mapa de variables eléctricas y sus relaciones como se presenta en la figura 1.1.

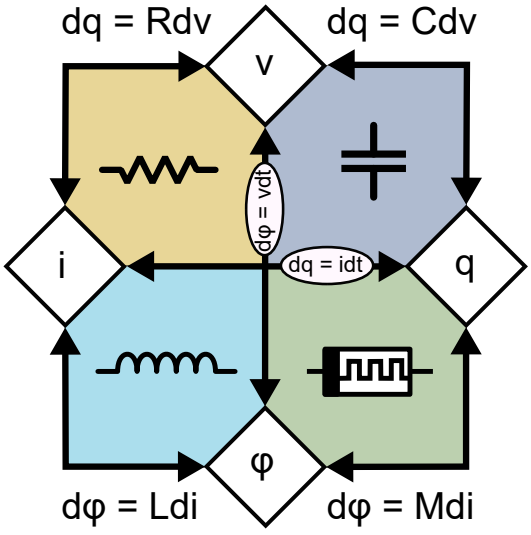


Figura 1.1: Relación entre variables eléctricas y dispositivos fundamentales

Algunos años más tarde de haber planteado la existencia de este dispositivo, Leon O. Chua y Sung Mo Kang [22] generalizan este elemento en una amplia categoría de componentes que tuvieran estas características denominados *sistemas memristivos*. Es así como la representación matemática de estos dispositivos distingue entre señales de entrada correspondientes a corriente y voltaje. Para el caso de un *memristor* controlado por

corriente se tiene que:

$$V = R(x) * i \tag{1.1}$$

$$\frac{dx}{dt} = f(x, i) \tag{1.2}$$

Por otro lado, la representación de un *memristor* controlado por voltaje está dada por:

$$i = G(x) * V \tag{1.3}$$

$$\frac{dx}{dt} = g(x, V) \tag{1.4}$$

En este sentido, se tiene que la memristancia R(x) y la memductancia G(x) dependen de un vector de variables de estado $x = (x_1, x_2, ..., x_n)$. Este último es independiente de las variables eléctricas comunes estando asociado a elementos físicos internos propios al dispositivo, como pueden ser el grosor de algún filamento conductivo, la temperatura, entre otros. No obstante, es la taza de cambio de estas variables de estado la que depende de señales eléctricas como lo pueden ser el voltaje y la corriente, tal como se muestra en las ecuaciones de $\frac{dx}{dt}$ para el dispositivo dependiente de corriente y voltaje.

Como se menciona anteriormente, este dispositivo de dos terminales se caracteriza por establecer una relación entre el flujo magnético y la carga eléctrica, pero esto no es tan evidente en el conjunto de ecuaciones presentadas previamente. Para ello, se consideran las siguientes ecuaciones que definen la relación voltaje-flujo y corriente-carga:

$$q(t) = \int_{-\infty}^{t} i(\tau)d\tau \tag{1.5}$$

$$\varphi(t) = \int_{-\infty}^{t} V(\tau)d\tau \tag{1.6}$$

Se tiene un *memristor* ideal cuando $f(x, i) = q(t)$ y $f(x, V) = \varphi$ [23]. Así, si se diferencian las ecuaciones 1.2 y 1.4, se pueden relacionar con las ecuaciones 1.5 y 1.6, respectivamente, de lo cual se obtiene:

$$\varphi(t) = \int_{-\infty}^{t} V(\tau)d\tau = \int_{-\infty}^{t} R(q(\tau))\frac{dq(\tau)}{d\tau}d\tau = \int_{\varphi(-\infty)}^{\varphi(t)} R(q)dq = \hat{\varphi}(q(t)) \tag{1.7}$$

$$q(t) = \int_{-\infty}^{t} i(\tau)d\tau = \int_{-\infty}^{t} G(\varphi(\tau))\frac{d\varphi(\tau)}{d\tau}d\tau = \int_{q(-\infty)}^{q(t)} G(q)dq = \hat{q}(\varphi(t)) \tag{1.8}$$

En consecuencia, se puede definir al *memristor* como un dispositivo controlado por carga:

$$\varphi(t) = \hat{\varphi}(q(t)) \tag{1.9}$$

Y un *memristor* controlado por flujo:

$$q(t) = \hat{q}(\varphi(t)) \tag{1.10}$$

Diferenciando las ecuaciones anteriores, se puede volver al concepto de *memristor* controlado por corriente y voltaje presentado en las ecuaciones 1.2 y 1.4, respectivamente:

$$V = \frac{d\varphi}{dt} = \frac{\hat{\varphi}(q)}{dq}\frac{dq}{dt} = R(q)i \tag{1.11}$$

$$i = \frac{dq}{dt} = \frac{\hat{q}(\varphi)}{d\varphi}\frac{d\varphi}{dt} = G(\varphi)V \tag{1.12}$$

De esta forma, se obtienen las memristancias $R(q)$ y memductancia $G(\varphi)$ introducidas previamente.

Acorde a Leon O. Chua, una característica fundamental de un memristor es que es capaz de generar una figura de Lissajous que toma valor cero en el origen [24]. A modo de ejemplo, se considera un memristor controlado por corriente que tiene una relación constitutiva, la cual analíticamente se describe como un polinomio cubico de la forma:

$$\varphi = q + \frac{1}{3}q^3 \tag{1.13}$$

Se considera que la entrada es una fuente de corriente dada por:

$$i(t) = \begin{cases} Asin(\omega t), & t \geq 0 \\ 0, & t < 0 \end{cases} \tag{1.14}$$

Luego, a partir de la ecuación 1.5 se puede calcular la carga en el tiempo. De esta forma, y considerando $q_0 = q(0) = 0$, se integra la ecuación 1.14:

$$q(t) = \int_0^t Asin(\omega t) = \frac{A}{\omega}[1 - cos(\omega t)] \tag{1.15}$$

Seguidamente, se sustituye la ecuación 1.15 en la ecuación constitutiva 1.13, obteniendo:

$$\varphi = \frac{A}{\omega}(1 - cos(\omega t))\left[1 + \frac{1}{3}(\frac{A^2}{\omega^2})(1 - cos(\omega t)^2\right] \tag{1.16}$$

Tomando en cuenta la ecuación 1.11, se diferencia la ecuación 1.16, lo cual resulta:

$$v(t) = A\left[1 + \frac{A^2}{\omega^2}(1 - cos(\omega t))\right] sin(\omega t) \tag{1.17}$$

Por lo tanto, al graficar la curva de relación v-i, se construye la forma característica de los *memristors* previamente descrita que se muestra en la figura 1.2.

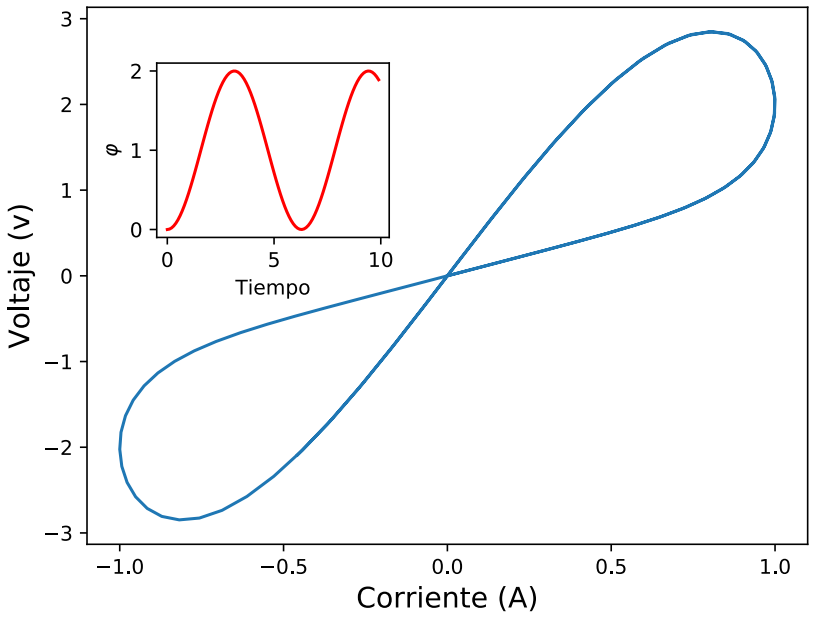


Figura 1.2: Curva v-i característica de los *memristors* que pasa por el punto (0,0). El inset presenta la evolución del flujo en el tiempo de simulación.

En la práctica, es raro encontrar dispositivos experimentales que obedezcan la relación que se expresa en las ecuaciones 1.9 y 1.10. De hecho, es más común modelar estos dispositivos de acuerdo a las ecuaciones 1.1-1.2 o las ecuaciones 1.3-1.3. Modelos de este tipo se presentarán en la sección siguiente y son los que se utilizan para este trabajo.

1.0.2. Modelos Reales de *memristor*

A pesar de que varios grupos de investigación en el mundo trabajaron en el desarrollo y caracterización de dispositivos de conmutación resistiva, no es hasta 2008 cuando el comportamiento de estos componentes fue relacionado con la teoría de Leon Chua de los años 70. Esto fue realizado por los laboratorios de HP en donde se descubrió un dispositivo de estas características utilizando nano-capas de dióxido de titanio (TiO_2) [15]. Este dispositivo está constituido por dos de estas capas de TiO_2, una que actúa a modo de aislante y otra que funciona como conductor utilizando vacantes de oxígeno como agente de dopaje. Al incrementar el voltaje sobre estos dispositivos, las vacantes de oxígeno se desplazan desde la capa superior a la capa inferior cambiando así la proporcionalidad del material conductor y el material aislante, y por ende, cambiando la resistencia del dispositivo.

En la figura 1.3 se muestra algunas características de los dispositivos desarrollados en los laboratorios HP. En la figura 1.3-a se observa una acercamiento a 1x17 puntos de cruce con dispositivos *memristors* TiO_2 fabricados en los laboratorios HP. En la figura 1.3-b se observa la curva I-V de estos dispositivos en estado prístino, es decir, previo a cualquier intento de un proceso de conmutación. En el inset de esta figura, se observa también las capas de material que componen el dispositivo. Por último, en la figura 1.3-c se observa la curva característica de estos dispositivos. En particular, destaca la forma de onda con histéresis tal como se explica en la sección previa. En esta última figura también destacan características que se observan en dispositivos reales como lo son la variabilidad ciclo a ciclo y la asimetría en el voltaje en que comienza la conmutación.

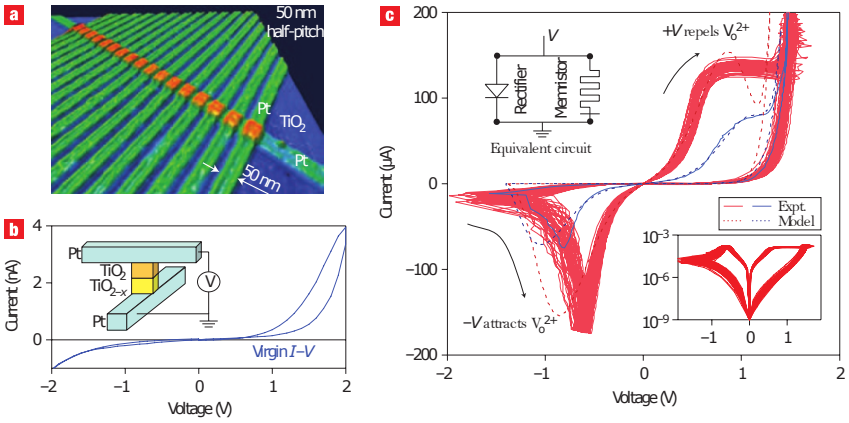


Figura 1.3: Comportamiento bipolar reversible de un dispositivo de conmutación de escala nanométrica TiO_{2-x}. a) Imagen AFM de 1x17 puntos de cruce que contiene un aislante de TiO_2 de 50 nm de espesor entre dos contactos de platino. b) Curva del dispositivo en estado prístino previo a cualquier proceso de conmutación. c) Curvas experimentales (sólidas) y modeladas (punteadas) I-V de conmutación. Curvas rojas representan 50 experimentos de conmutación sobre un dispositivo demostrando los distintos niveles de resistencia del mismo. En el inset inferior se observan razones de conducción ON/OFF de $1x10^3$. En el inset superior se tiene un esquema del circuito equivalente.[25]

Una forma de tener un mejor entendimiento de los dispositivos de conmutación resistiva y descubrir sus capacidades para distintas aplicaciones es utilizando modelos matemáticos que logren replicar las complejas características internas de estos elementos. Para el caso del dispositivo presentado anteriormente, el grupo de HP presenta en [15] un simple modelo en el cual considera una capa de semiconductor de grosor **D** que está ubicada entre dos contactos de metal, tal como se muestra en la figura 1.4-a. La resistencia del dispositivo se modela como dos resistencias variables conectadas en serie. Una región del semiconductor está asociada a grandes concentraciones de dopantes que permiten que funcione como conductor lo que produce que esta sección posea baja resistencia R_{ON}, mientras que la región restante posee una baja concentración de dopantes generando que actúe como aislante al presentar alta resistencia R_{OFF}.

Al aplicar un voltaje sobre los terminales del dispositivo, se logra que la frontera entre ambas secciones se desplace aumentando la concentración de dopantes en el elemento y disminuyendo su resistencia. En este caso, la extensión que alcanzan los dopantes a lo largo del dispositivo está representada por la variable **w(t)**.

De esta forma, si es que se considera un estado de conducción óhmica y un desplazamiento de iones lineal con una movilidad promedio dada por μ_v, se puede modelar este fenómeno de desplazamiento a través de las siguientes ecuaciones:

$$v(t) = (R_{ON}\frac{w(t)}{D} + R_{OFF}(1 - \frac{w(t)}{D})) \tag{1.18}$$

$$\frac{dw(t)}{dt} = \mu_v \frac{R_{ON}}{D} i(t) \tag{1.19}$$

En donde w(t) esta dado por:

$$w(t) = \mu_v \frac{R_{ON}}{D} q(t) \tag{1.20}$$

A partir de esto, combinando las ecuaciones 1.20 y 1.18 se obtiene la ecuación que describe la memristancia del dispositivo. Cuando $R_{ON} << R_{OFF}$, ésta viene dada por:

$$M(q) = R_{OFF}(1 - \frac{\mu_v R_{ON}}{D^2} q(t)) \tag{1.21}$$

Se debe señalar que la variable $w(t)$ está limitada en el intervalo $[0, D]$, donde el crecimiento de esta variable es proporcional a la carga q(t) que pasa a través del dispositivo hasta que su valor se aproxime a D. Es así como el dispositivo tendrá este comportamiento de histéresis simétrica ante una entrada sinusoidal como se observa en la figura 1.4-b. A medida que aumenta la frecuencia de la señal de entrada, esta curva de histéresis degenera en una línea recta. También se pueden lograr curvas i-v asimétricas con señales de entradas distintas como se puede observar en la figura 1.4-c.

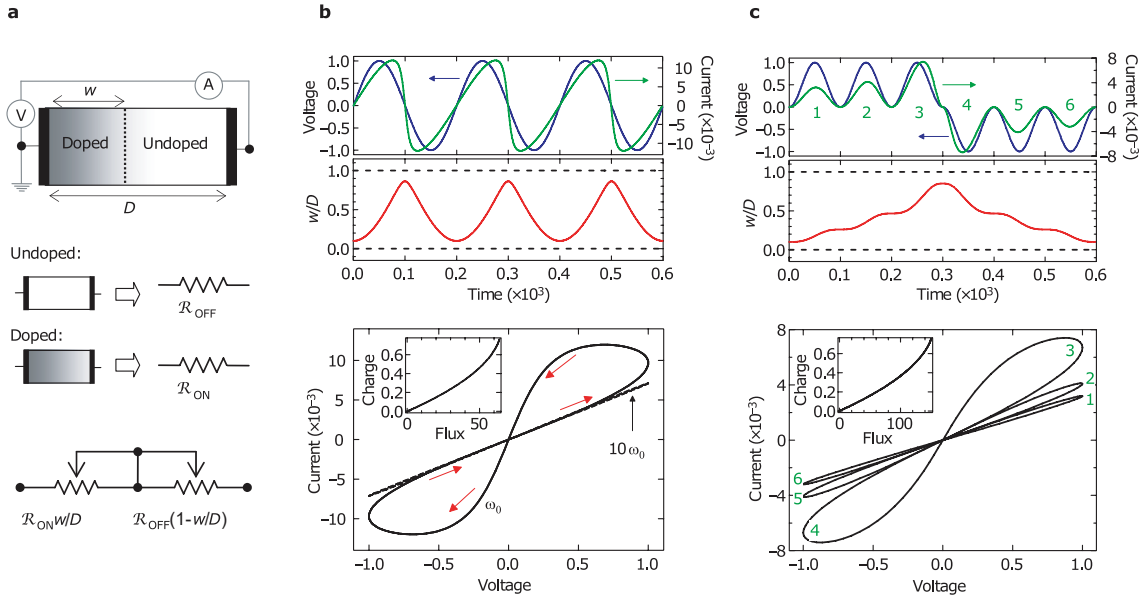


Figura 1.4: a) Diagrama con versión simplificada del circuito que modela un memristor. b) Pruebas al modelo considerando como entrada un voltaje igual a $v_0 sin(\omega_0 t)$ (en azul) y una razón de resistencia de $R_{OFF}/R_{ON} = 160$, en verde la corriente resultante. c) Pruebas al modelo con una entrada igual a a $\pm v_0 sin^2(\omega_0 t)$ (en azul) y una razón de resistencia igual a $R_{OFF}/R_{ON} = 380$. Para todos los casos, v_0 es la magnitud del voltaje aplicado y ω_0 es la frecuencia de la señal. Figura obtenida desde [15]

El modelo presentado es uno de los más utilizados hasta ahora y cuenta además con una implementación en lenguaje SPICE desarrollada en el artículo [26], publicado en la revista Radioengineering en 2009, lo que permite la simulación de este dispositivo. La principal característica de este modelo es presentar una relación proporcional entre la corriente y el ritmo de cambio de la resistencia. Lo anterior no siempre se cumple ya que existen dispositivos que presentan un comportamiento basado en umbrales de voltaje, en donde la resistencia del dispositivo cambia poco o nada si es que el voltaje está por debajo de

un valor específico dado por las características del dispositivo. Este comportamiento de umbrales resulta interesante ya que es idóneo para aplicaciones digitales. Lo anterior principalmente porque permite separar operaciones de lectura, en las cuales no se busca alterar el estado del dispositivo, de operaciones de escritura en donde se busca almacenar algún valor lógico. Este tipo de dispositivos son los que se exploran en secciones posteriores.

1.0.3. Selección de modelo para aplicación en la memoria

En la actualidad, se reconocen distintas composiciones de materiales que se pueden utilizar como base para la fabricación de *memristors*. La diferencia más relevante entre éstos radica en el actuar de los dispositivos ante la aplicación de algún voltaje sobre sus terminales y los mecanismos de cambio de resistencia. En forma general, este proceso puede ocurrir de dos formas. Por un lado, mediante la formación e interrupción de filamentos conductivos entre los contactos metálicos, y por otro, el mecanismo de conducción se puede generar por medio de interfaces conductivas [27]. En la práctica, se tiene que mientras los dispositivos que ocupan el mecanismo de filamentos muestran cambios de resistencia abruptos, aquellos de interfaz poseen una conmutación de resistencia gradual. A pesar de esta diferencia fundamental, se ha observado que ambos tipos de dispositivos presentan umbrales de voltaje que permiten el cambio de la resistencia.

Otra diferencia que se reconoce se relaciona con el mecanismo con el que se puede producir la conmutación, el cual varía de acuerdo al tipo de señal de entrada que se aplique sobre el componente [28]. Para esta situación, se tiene un dispositivo unipolar o bipolar si es que el proceso de conmutación se activa utilizando una señal de una polaridad o de polaridades opuestas, respectivamente. El tercer caso corresponde a los dispositivos no-polares que responden a señales de cualquier polaridad.

Para este caso, se debe considerar que la aplicación a la que se apunta corresponde a la de computación en la memoria. De esta manera, la selección del dispositivo utilizado debe responder a criterios que permitan implementar un estilo de computación lógica con estos dispositivos y repliquen las características de una memoria no-volátil. Al respecto, se puede adelantar que una gran mayoría de estilos de lógica con *memristors* existentes en la literatura basan su funcionamiento en umbrales de voltaje y un tipo de conmutación bipolar. A su vez, una de las principales características que se busca en un elemento de memoria es alta retención y conmutación en el orden de nano-segundos, las cuales se suelen encontrar en dispositivos de filamentos conductivos en los cuales se tengan umbrales fuertes. Por tanto, para este trabajo se considerara un modelo de **memristor** bipolar y umbrales de voltaje, cuya resistencia varíe entre un estado resistivo alto (High Resistance State - HRS con un valor máximo de R_{OFF}) y un estado resistivo bajo (Low Resistance State - LRS con un valor mínimo de R_{ON}).

1.0.4. Modelo Pershin

La conmutación de la resistencia de los *memristors* a raíz de la existencia de umbrales de voltaje es una propiedad que se ha observado en distintos tipos de dispositivos [29]. Esta propiedad es importante pues permite que este elemento tenga propiedades de memoria no volátil, además de ser la base para la implementación de operaciones lógicas. No obstante, capturar las propiedades físicas de los dispositivos en un modelo que pueda replicar el comportamiento dinámico del dispositivo es una tarea sumamente difícil. Es por esto que se desarrollan modelos conductuales que permiten replicar los procesos de conmutación resistiva observados en dispositivos experimentales. En este sentido, uno de los primeros trabajos que plantea un modelo de *memrsitor* basado en umbrales de voltaje es el presentado por Yuriy V. Pershin y Massimiliano Di Ventra en [30]. Este modelo se describe mediante las siguientes ecuaciones:

$$I_M(t) = x^{-1} V_M(t) \tag{1.22}$$

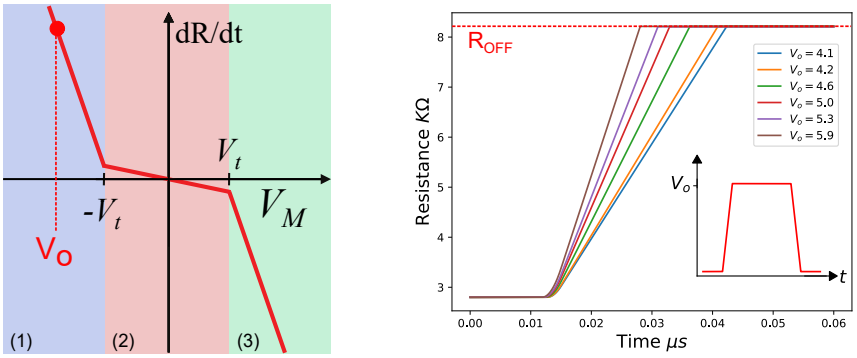
$$\frac{dx}{dt} = f(V_M)[\Theta(V_M)\Theta(R_{OFF} - x) + \Theta(-V_M)\Theta(x - R_{ON})] \tag{1.23}$$

$$f(V_M) = \beta V_M + 0{,}5(\alpha - \beta)[|V_M + v_t| - |V_M - v_t|] \tag{1.24}$$

En este caso, las variables $V_M(t)$ e $I_M(t)$ representan el voltaje y la corriente sobre el *memristor*, respectivamente. Por otro lado, en este modelo la variable de estado x corresponde directamente a la resistencia R_M del dispositivo, mientras que el parámetro v_t configura el umbral de voltaje tanto para el proceso de SET ($R_M \rightarrow R_{ON}$), como para el proceso de RESET ($R_M \rightarrow R_{OFF}$). Como se ha hablado previamente, R_{OFF} y R_{ON} son los valores de máxima y mínima resistencia respectivamente que puede alcanzar R_M, este último limitado entre estos valores mediante la función escalón Θ. La ecuación 1.23 define la taza de cambio de la variable de estado equivalente a la resistencia y depende del voltaje de entrada al dispositivo V_M y su relación con v_t, tal como se muestra en la figura 1.5-a. En esta figura se presenta que dx/dt puede estar en tres zonas dependiendo de V_M dando origen a 4 expresiones, las cuales son:

$$\frac{dx}{dt} = \begin{cases} \alpha V_M(R_{OFF} - x), & V_M < v_t \\ -\alpha|V_M|(x - R_{ON}), & V_M > -v_t \\ \beta V_M(R_{OFF} - x), & V_M > v_t \\ -\beta|V_M|(x - R_{ON}), & V_M < -v_t \end{cases} \tag{1.25}$$

De esta forma, en la ecuación 1.25 se observa los distintos valores que puede adoptar la taza de cambio dependiendo del voltaje V_M. Cuando $|V_M| < |v_t|$, se tiene que dx/dt tiene una relación lineal con el voltaje de entrada V_M, en donde la pendiente está dada por el parámetro α. Por otro lado, si es que $|V_M| > |v_t|$, se tiene también una relación lineal pero esta vez la pendiente es controlada por el parámetro β.



(a) **(b)**

Figura 1.5: 50 ciclos de salidas del comparador a partir de las pruebas de computación. (a) Taza de cambio de la resistencia acorde al voltaje V_m que recae sobre el *memristor*. (b) Respuesta de la resistencia de un *memristor* a pulsos de voltaje V_o (se muestra en el inset) utilizando el modelo de Pershin para parámetros $R_{OFF} = 8{,}2\ k\Omega$, $R_{ON} = 2{,}8\ k\Omega$, $\beta =$ 1e11 $\alpha = 0$, $v_t = 2.2$ V.

Un ejemplo del proceso de conmutación de un dispositivo descrito con este modelo se presenta en la figura 1.5-b. En esta gráfica se muestra la conmutación de la resistencia que ocurre en un dispositivo cuando se tiene como entrada pulsos de distinta amplitud. Para esta prueba, se considera los valores de los parámetros $\beta =$ 1e11 y $\alpha = 0$, así como $R_{OFF} = 8.2\ k\Omega$, $R_{ON} = 2.8\ k\Omega$ y $v_t = 2.2$ V. En este gráfico se destaca cómo la taza de cambio aumenta en la medida que aumenta la diferencia entre v_t y la amplitud del pulso v_o.

1.0.5. Modelo VTEAM

Dentro de los modelos más usados en la literatura y que cumplen con los criterios expuestos anteriormente, se tiene el modelo VTEAM (*Voltage ThrEshold Adaptive Memristor*) [31], el cual busca rescatar las principales características de su modelo homólogo basado en umbrales de corriente TEAM (*ThrEshold Adaptive Memristor*)[32], entre las cuales se encuentran simplicidad, generalidad, precisión numérica y bajo coste computacional. Este modelo ha mostrado una capacidad de replicar comportamiento de dispositivos reales con una precisión tal que logra error RMS menor a 1.5 %.

Tal como su antecesor, este modelo tiene como base una expresión para la derivada de una variable de estado w del dispositivo que representa la extensión de la región de dopantes dentro del dispositivo (región de conductividad), en donde $w \in [w_{off}, w_{on}]$. Partiendo del concepto de un dispositivo controlado por voltaje, se tienen las siguientes ecuaciones asociadas a la tasa de cambio de la variable de estado y la corriente que pasa a través del dispositivo:

$$\frac{dw}{dt} = f(w, v) \tag{1.26}$$

$$i(t) = G(w, v)v(t) \tag{1.27}$$

Desarrollando $f(w, v)$ se tiene que la derivada en el tiempo de la variable de estado corresponde a:

$$\frac{dw(t)}{dt} = \begin{cases} k_{off}(\frac{v(t)}{v_{off}} - 1)^{\alpha_{off}} f_{off}, & 0 < v_{off} < v(t) \\ 0, & v_{on} < v(t) < v_{off}(w) \\ k_{on}(\frac{v(t)}{v_{on}} - 1)^{\alpha_{on}} f_{on}(w), & v(t) < v_{on} < 0 \end{cases} \tag{1.28}$$

En este caso, se tiene que k_{off}, k_{on}, α_{on} y α_{off} son constantes, y tanto v_{on} como v_{off} son los umbrales de voltaje del dispositivo. Las funciones f_{off} y f_{on} representan la dependencia de la tasa de cambio con el estado actual del *memristor*. Además, buscan ser funciones ventanas que restrinjan la conmutación de la resistencia dentro de los límites propios del dispositivo, enmarcados en el intervalo $[w_{on}, w_{off}]$. Existe una gran variedad de expresiones que se pueden utilizar para acotar la tasa de cambio de este modelo, algunas de estas son estudiadas en [33], las cuales se enfocan en mejorar la convergencia de las simulaciones. Existen también otras funciones que buscan agregar características observadas en dispositivos reales que algunos modelos como este no consideran, como puede ser el efecto de la magnitud de las señales aplicadas sobre el comportamiento dinámico del dispositivo, función que se presenta en [34]. No obstante, para el presente estudio se considera una función ideal correspondiente a una ventana rectangular:

$$f_{on}(w) = f_{off}(w) = \begin{cases} 1, & w \in [w_{on}, w_{off}] \\ 0, & w \notin [w_{on}, w_{off}] \end{cases} \tag{1.29}$$

Por otro lado, desarrollando la conductancia $G(w, v)$ de la ecuación 1.27, se tiene la expresión que relaciona $i(t)$ y $v(t)$:

$$i(t) = \left[R_{ON} + \frac{R_{OFF} - R_{ON}}{w_{off} - w_{on}}(w - w_{on}) \right]^{-1} v(t) \tag{1.30}$$

Para este trabajo, los parámetros seleccionados buscan una conmutación de las resistencias en un tiempo de 1 ns, de manera de tener una respuesta cuantitativamente comparable con las tecnologías de memorias existentes actualmente. Para lograr estos tiempos de conmutación, se utilizan aquellos parámetros expuestos en [35] y que se presentan en la tabla 1.1. A diferencia de los valores que se muestran en la tabla, por lo general, es deseable tener un valor alto de resistencia R_{ON} y un valor bajo de umbrales de voltaje con el objetivo de minimizar el consumo de energía. A pesar de lo mencionado anteriormente, se prefiere utilizar los parámetros presentados debido a que vienen a caracterizar el comportamiento de dispositivos experimentales presentados en [36].

Parametro	Valor	Parametro	Valor
R_{ON}	1 kΩ	x_{off}	3nm
R_{OFF}	300 kΩ	k_{on}	-215.2 m/s
v_{on}	-1.5 V	k_{off}	0.091 m/s
v_{off}	0.3 V	α_{on}	4
x_{on}	0	α_{off}	4

Tabla 1.1: Tabla de parámetros utilizados con el modelo VTEAM [35]

Utilizando los parámetros descritos previamente, se puede obtener la curva característica para este modelo bajo dicho régimen. Como entrada, se utiliza una señal sinusoidal de $V_{in} = 2sin(5e3t)$, generando la

huella que se presenta en la figura 1.6-b. Como se puede observar, el régimen de conmutación es asimétrico. Esto se produce directamente por la asimetría en los umbrales de voltaje para los procesos de SET y RESET. Por otro lado, se presenta en la figura 1.6-c la respuesta del dispositivo ante pulso de SET y RESET de 1 V y -2 V, respectivamente. Aquí se observa la conmutación en alrededor de 1 ns.

Como se explica anteriormente, la resistencia de los dispositivos varía entre alta (HRS) y baja resistencia (LRS), en donde la polaridad que define la dirección de la conmutación de estos se identifica a través de una línea negra gruesa en el electrodo inferior (BE) del esquemático. Precisamente, esto se presenta en la figura 1.6-a. Por su parte, los datos almacenados en un *memritor* están representados mediante valores de resistencia. De esta forma, a lo largo de este trabajo se considera que un lógico '0' es equivalente a alta resistencia (HRS), mientras que un lógico '1' es equivalente a baja resistencia (LRS), mientras que cualquier estado intermedio es considerado como un estado no identificado (prohibido) tal como se indica en la figura 1.6-d.

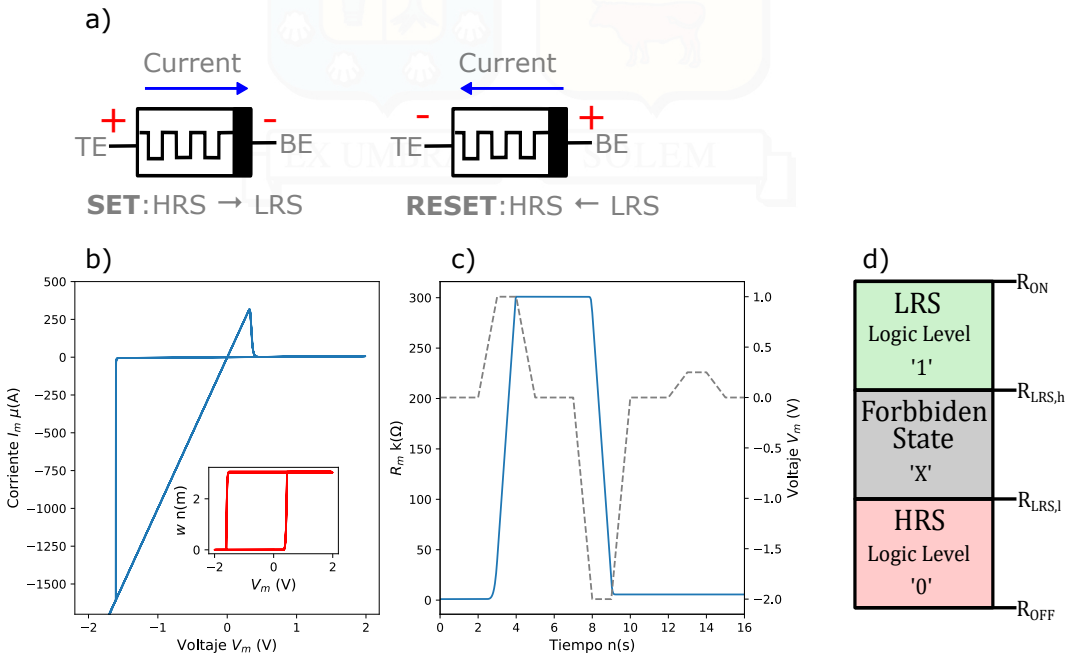


Figura 1.6: Comportamiento de *memristor* considerando modelo VTEAM a partir de los parámetros seleccionados. a) Comportamiento del dispositivo cuando el voltaje sobre los terminales TE-BE (Top Electrode y Bottom Electrode) es positivo o negativo. b) Curva de histéresis que es la huella característica del modelo. El inset presenta la variación de la variable del estado en función del voltaje aplicado. c) Respuesta de la resistencia a pulsos de set y reset consecutivos de 1 ns en alto y 1 ns de tiempos de subida y bajada. d) Representación gráfica del rango de resistencias del dispositivo y la equivalencia con niveles lógicos.

1.0.6. Modelo de simulación SPICE

Las simulaciones contempladas en este trabajo se realizan utilizando la herramienta de simulación de circuitos análogos LTSpice. Esta herramienta basa su funcionamiento en el programa SPICE (*"Simulation Program with Integrated Circuit Emphasis"*), el cual es un ambiente de simulación y modelado de circuitos integrados que permite predecir de manera matemática el comportamiento de distintos circuitos electrónicos.

A pesar de que existen naturalmente modelos precisos para SPICE para la gran mayoría de los dispositivos comúnmente utilizados y para distintos elementos discretos disponibles en el mercado, no ocurre lo mismo con los dispositivos de conmutación resistiva. Al ser un elemento de una tecnología emergente, las herramientas de simulación no incluyen implementación alguna de modelos directamente en sus librerías,

como sí se tiene para otros dispositivos. Debido a lo anterior se hizo necesario expandir estas librerías a través de modelos SPICE que implementaran las ecuaciones de los dispositivos.

Biolek en [26] plantea un primer modelo de SPICE que permite implementar el modelo de *memristor* basado en capas de TiO_2 a partir del modelo presentado en [15], el cual utilizaba una representación de fuente de voltaje controlada. En este sentido, se tienen más estrategias para la implementación de modelos de *memristor* en SPICE, en donde son tres las que destacan. En primer lugar, se tiene la representación usando el resistor, cuyo valor de resistencia varia acorde a las ecuaciones del modelo dependiendo de la variable de estado. Este se presenta en el esquema de la figura 1.7-a. Por otro lado, también se puede usar la representación de fuente de corriente controlada. Esta es la que define la corriente que pasa por el dispositivo dependiendo del valor de resistencia actual y el voltaje sobre los terminales, tal como se presenta en la figura 1.7-b. Por último, se tiene como representación del *memristor* la fuente de voltaje controlada, en cuyo caso se define el voltaje que cae sobre los terminales y depende de la corriente que atraviese el dispositivo, además de la resistencia actual, tal como se muestra en la figura 1.7-c. Todas estas representaciones comparten un elemento en común y este es la representación de la variable de estado. En general, para almacenar esta variable se utiliza un circuito de fuente de corriente conectada en serie con un condensador, presentado en la figura 1.7-d, en donde la ecuación que define la tasa de cambio de la variable de estado controla la fuente de corriente y el valor de esta variable corresponde al voltaje del condensador.

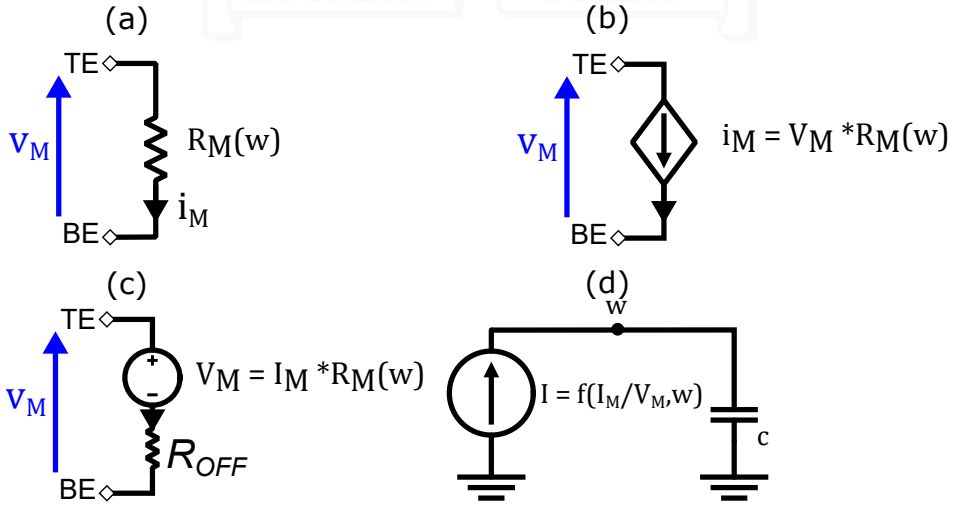


Figura 1.7: Distintos modelos de representación del *memristor* en modelos SPICE. (a) Representación como resistencia variable dependiente de la variable de estado. (b) Representación como fuente de corriente controlada. (c) Representación como fuente de voltaje controlada. (d) Representación del cambio de la variable de estado como fuente de corriente en serie con condensador. Esta fuente de corriente es controlada por la expresión que determina el cambio de la variable de estado del dispositivo.

En particular, el modelo VTEAM utilizado no tiene una netlist SPICE directamente implementable en LTSpice, como sí la tienen otros modelos tales como el de Biolek mencionado previamente o el modelo de Pershin [30]. Precisamente por lo anterior es que se debe generar esta netlist para trabajar con la herramienta LTSpice.

De esta forma, a continuación, se presenta la netlist en SPICE que describe el modelo VTEAM que se utiliza para este trabajo, definido como un subcircuito siguiendo las mismas prácticas que se usaron para implementaciones en SPICE de otros modelos de la literatura. En este caso, al tratarse de un dispositivo de la misma naturaleza, y por simplicidad, se selecciona la representación que utiliza el modelo de Pershin en el que se modela el *memristor* como una resistencia. Específicamente, el nombre que describe el dispositivo es "memristorVTEAM" y se representa como un dispositivo de 3 terminales. El tercer terminal existe solamente para razones de debugging y permite la visualización del valor de la variable de estado.

```
1
2 *Definicion del subcircuito y sus pines.
3
```

```
.subckt memristorVTEAM pl mn x

*Definicion de los parametros utilizados en el modelo.
.params Ron=1e3 Roff=3e5 Won=0 Woff=3e-9 Winit = 0 alpha_on=4 alpha_off=4 Von=-1.5 Voff
    =0.3 Kon=-216.2 Koff = 0.091

*Definicion de fuente de corriente controlada por voltaje
Bx 0 x I=if((V(pl,mn)>0)&(V(pl,mn)>Voff),f3(V(pl,mn),V(x,0)),if((V(pl,mn)<0)&(V(pl,mn)<
    Von),f2(V(pl,mn),V(x,0)),0))

*Definicion de condensador que almacena la variable de estado w
Cx x 0 1 IC={Winit}

*Definicion resistencia de valor variable definida para el dispositivo
Rmem pl mn r= Rm(V(x,0))


*Implementacion de las funciones del modelo VTEAM
.func Rm(x) = {(Ron+(Roff-Ron)*(x-Won)/(Woff-Won))}
.func fon(w1) = {stp(w1-Won)}
.func foff(w2) = {stp(Woff-w2)}
.func f2(v,w) = {Kon*pow(((v/Von)-1),alpha_on)*fon(w)}
.func f3(v0,w0) = {Koff*pow(((v0/Voff)-1),alpha_off)*foff(w0)}
.ends
```

2 | Computación Lógica con *Memristors*

2.0.1. Clasificación de los estilos de lógica

Para lograr aplicaciones de computación en una memoria resistiva ReRAM utilizando *memristors* es necesario establecer una forma de implementar un set de operaciones lógicas con ellos. Anteriormente, se describió que la información almacenada en estos dispositivos está asociada a la resistencia de los mismos, en donde HRS está asociada a un '1' lógico y LRS a un '0' lógico. Esta convención sigue la línea de una amplia cantidad de trabajos sobre computación con *memristors* existentes en la literatura, los cuales, en general, buscan asociar conceptos generales de la lógica digital a esta tecnología.

Se reconocen diferentes estilos con los cuales se pueden implementar operaciones lógicas con *memristors*, los que pueden ser clasificados de acuerdo al tipo de señal que constituye las entradas/salidas (I/O) de la computación lógica. Este tipo de señal puede ser voltaje (V) o resistencia (R), de esta forma, distintas familias de lógica con *memristors* emplean una combinación de entradas y salidas compuestas por estas señales. El estilo de lógica se clasifica como *"Stateful Logic"* si es que ambas entradas y salidas de una operación lógica son representadas como resistencia, es decir, cuando la información lógica que una compuerta recibe como entrada(s) y también el resultado de la operación se expresan en forma de resistencia de los *memristors* que participan en la operación. Por otro lado, se conoce como *"Non-Stateful Logic"* a aquellos estilos de lógica que tengan como entrada niveles de voltaje, es decir, estilos donde la información lógica de entrada se expresa en forma de voltaje aplicado al circuito que implementa la operación lógica.

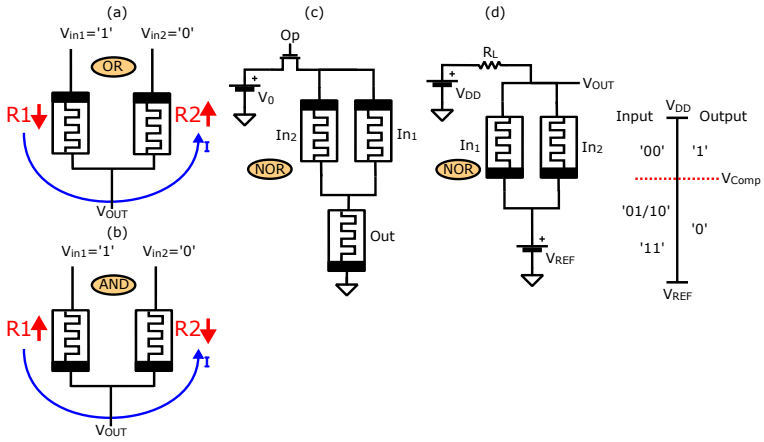


Figura 2.1: Ejemplos de distintos estilos de lógica. (a,b) Compuertas OR/AND implementadas utilizando el estilo MRL[37]. c) Compuerta NOR de 2 entradas (NOR2) implementada con el estilo MAGIC[35]. d) Compuerta NOR2 implementada utilizando el estilo *"Ratioed Logic"*[38]. A la derecha de este circuito, se muestra la relación entre los estados lógicos de entrada, los voltajes de salida esperados y el valor lógico de la salida a partir de un umbral V_{comp}.

En la figura 2.1, se muestran ejemplos de las dos familias de estilos lógicos descritos previamente. En el caso de la figura 2.1-(a,b), se puede observar una compuerta OR/AND implementada utilizando el estilo de

lógica MRL[37], en donde las entradas a las compuertas corresponden a los voltajes V_{in1} y V_{in2}. En este estilo, la computación se lleva a cabo a través de la distribución de los voltajes entre los dispositivos produciéndose la conmutación de ambos, proceso que entrega un voltaje V_{out} a la salida. Por otra parte, en la figura 2.1-c se presenta la implementación de una compuerta NOR utilizando el estilo MAGIC[35]. Este estilo en particular considera como entradas los estados resistivos de ambos *memristors*, de esta forma, la computación de esta compuerta comienza tras aplicar un voltaje V_o que produce la conmutación condicional del dispositivo "Out". Este último dispositivo se encuentra previamente inicializado en baja resistencia y almacena el resultado lógico de salida en forma de resistencia.

Finalmente, se tiene el estilo *Ratioed Logic*[38] presentado en la figura 2.1-d. Este estilo, al igual que el mencionado anteriormente, se basa en una configuración de divisor de voltaje entre un conjunto de *memristors* y un elemento resistivo que se puede implementar mediante un transistor MOSFET polarizado adecuadamente. La figura presenta la implementación de una función NOR cuyas entradas se encuentran almacenadas en forma de resistencia en los *memristors*. Para iniciar la computación de una compuerta NOR con estilo *Ratioed Logic* se utiliza un voltaje de alimentación VDD y voltaje de referencia V_{REF}, esta diferencia cae sobre los dispositivos de entrada conectados en paralelo que, a la vez, están en serie con una resistencia. De esta manera, producto del divisor de tensión que se forma se obtiene un voltaje V_{OUT} característico para cada una de las combinaciones de entradas, es decir, para cada una de las combinaciones de estados resistivos de los dispositivos. Este último voltaje pasa por un comparador donde se obtiene una salida binaria acorde a la tabla de verdad de la compuerta.

En conjunto con los estilos mencionados previamente, existen otros que dan paso a la computación de compuertas lógicas utilizando estos dispositivos, sin embargo, no todos son compatibles con las topologías de memoria comúnmente utilizadas (de las cuales se comenta más adelante). También se debe considerar que la gran mayoría de los estilos existentes se concentran en mejorar métricas de latencia, energía y eficiencia de área, pero que no son las únicas métricas relevantes. Otro factor a tomar en cuenta, algo que no todos los trabajos consideran, corresponde a la variabilidad intrínseca de los dispositivos, la cual se da en cada ciclo de operación y en la confección de cada elemento de manera individual. Además, la durabilidad de los dispositivos es también un factor relevante y se vuelve muy importante si cada operación lógica asume que los dispositivos que participan tienen que conmutar, lo cual en otras palabras significa que cada operación lógica afecta a la vida útil de un dispositivo ReRAM.

2.0.2. Estilo Ratioed Logic

En este trabajo, el estilo de lógica con *memristors* utilizado se denomina *"Ratioed Logic"*[38]. Este estilo se inspira de otros trabajos en los que se logra la computación de compuertas lógicas mediante una operación modificada de lectura, mecanismo que fue presentado en [39]. Esta idea también la comparte el estilo *"Scouting Logic"*[40], el cual utiliza un amplificador sensor de corriente/voltaje para el cómputo de compuertas lógicas. Como se menciona anteriormente, el lograr una computación sin la necesidad de producir la conmutación de resistencia en los dispositivos es una ventaja que permite ampliar la durabilidad de estos dentro de un arreglo de memoria ReRAM.

Para la implementación de este estilo de lógica, se tienen dos posibles esquemas que se presentan en la figura 2.2. Esta puede ser un transistor NMOS conectado en serie con un bloque pull-up de *memristors* o, en su defecto, un transistor PMOS en serie con un bloque pull-down de *memristors*. Los circuitos de ambas implementaciones se muestran en la figura 2.2-a. En la figura 2.2-b, se muestra la implementación del segundo caso en el que se tiene un bloque de *memristors* de pull-down que constituyen una compuerta NOR de 2 entradas (NOR2). El funcionamiento de esta compuerta comienza con la activación del transistor PMOS, cuyo diseño debe estar orientado de tal forma que su resistencia sea equivalente a la resistencia baja (LRS) de los *memristors*. Una vez puesto este transistor en conducción se aplica un voltaje de referencia $V_{ref} = V_{DD} - V_{Read}$ lo que permite obtener un voltaje de salida V_{out} que dependerá de los estados resistivos de los *memristors*. Por lo tanto, considerando R_L como la resistencia del transistor cuando se encuentra en conducción (equivalente a R_{ON}) y R_{eq} como la resistencia equivalente de los *memristors* en paralelo, se tiene

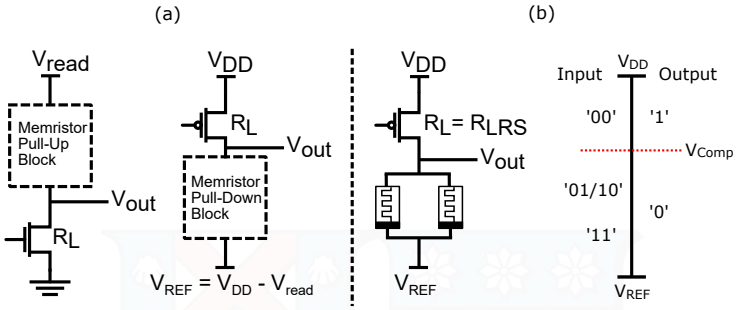


Figura 2.2: Implementación del estilo *"Ratioed Logic"*. (a) Esquemas de implementación del estilo *"Ratioed Logic"* equivalentes. (b) Ejemplo de implementación de compuerta NOR2 utilizando *Memristor Pull-Down Block*. Se explica también la relación entre entradas y voltajes de salida, en conjunto con los valores lógicos de éstas.

que V_{out} está dado por:

$$V_{out} = V_{REF} + (V_{DD} - V_{REF})\frac{R_{eq}}{R_L + R_{eq}} \tag{2.1}$$

Es así como, para reconocer entre '1' y '0' lógico, V_{out} debe pasar por un comparador en donde se considere un voltaje V_{Comp} adecuado que permita distinguir ambas salidas acordes a las entradas (estados resistivos de los *memristors*). Un factor importante que se debe tomar en cuenta al trabajar con este estilo es la relación entre las resistencias LRS y HRS, donde es deseable que $R_{off} >> R_{on}$ para lograr mejorar la robustez de la operación. Asumiendo que se cumple dicha relación, en la tabla 2.1 se muestra las expresiones que toma el voltaje de salida considerando cada una de las combinaciones de entradas cuando se computa una compuerta NOR2 con el estilo *"Ratioed Logic"*. Se tiene que, cuando ambas entradas son lógicos '0', los dispositivos se encuentran en alta resistencia (HRS) y el voltaje de salida es aproximadamente igual al voltaje de alimentación V_{DD}, siendo interpretado como un lógico '1'. Por su parte, si alguna de las dos entradas corresponde a '1' lógico (LRS), se tiene un voltaje $V_{out} < [V_{REF} + (V_{DD} - V_{REF})/2]$ lo que corresponde a un lógico '0'.

Combinación de entradas	R_{eq}	Valor V_{out} esperado	Salida Lógica
'00'	$R_{HRS}/2$	$\approx V_{DD}$	'1'
'01'/'10'	$\approx R_{LRS}$	$V_{REF} + (V_{DD} - V_{REF})/2$	'0'
'11'	$R_{LRS}/2$	$V_{REF} + (V_{DD} - V_{REF})/3$	'0'

Tabla 2.1: Relación entre entradas lógicas y voltajes de salida con la implementación de una compuerta NOR2 con el estilo *"Ratioed Logic"*

De esta forma, para distinguir ambas salidas lógicas el voltaje de comparación debe cumplir con el requisito:

$$V_{DD} > V_{Comp} > V_{REF} + (V_{DD} - V_{REF})/2 \tag{2.2}$$

Por otro lado, este estilo de lógica también soporta el cómputo de la compuerta NOT, cuya topología circuital es equivalente a la de la compuerta NOR2, pero utiliza solamente una entrada tal como se muestra en la figura 2.3-a. En esta representación, se observa también la relación entre la entrada a la compuerta (estado resistivo del *memristor*) y el voltaje de salida, en donde un '0' lógico (alta resistencia) entrega un voltaje cercano a V_{DD}, mientras que una entrada de '1' lógico (baja resistencia) entrega un voltaje cercano a V_{ref}. Similar a lo que se explica previamente, se utiliza el concepto del comparador para diferenciar ambas salidas. Por otra parte, si es que el resultado de esta compuerta NOT se invierte, se puede obtener el valor almacenado dentro del *memristor* que está participando de la operación. El circuito que permite implementar esto se muestra en la figura 2.3-b. Esto es equivalente a tener un buffer o realizar una operación de lectura del dispositivo.

Se puede considerar este estilo de lógica como universal lo que implica que se puede implementar cualquier función booleana. Las salidas que se computan se pueden reutilizar como entradas de compuertas

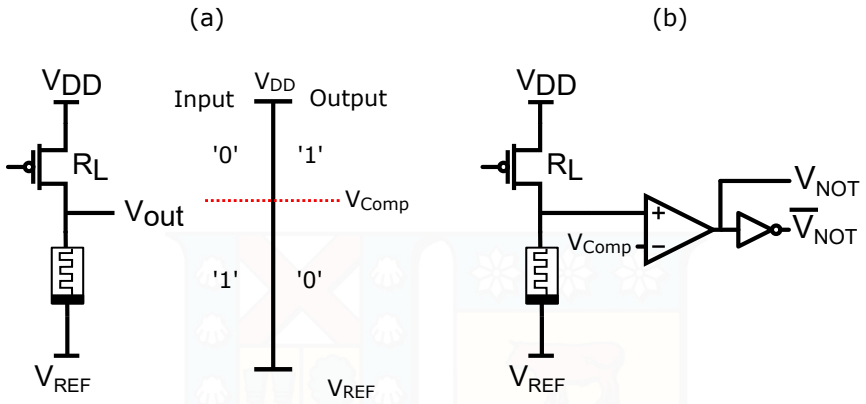


Figura 2.3: Implementación compuerta NOT y buffer del estilo *"'Ratioed Logic"*. (a) Esquemas de implementación compuerta NOT equivalente a compuerta NOR con una sola entrada. Se muestra también la relación de entradas y voltajes de salida, además de la equivalencia lógica de estas últimas. (b) Circuito necesario para obtener el cómputo de una compuerta NOT y un buffer.

lógicas posteriores en implementaciones multinivel. Esto se muestra en el próximo capítulo en donde se introduce la topología de memoria utilizada y la periferia diseñada para el manejo de las operaciones de ésta.

2.0.3. Simulando el estilo *"Ratioed Logic"*

Como se menciona previamente, el estilo de lógica *Ratioed Logic* es aquel utilizado para el diseño e implementación de un sistema de computación lógica haciendo uso de *memristors*. Debido a esto, se considera una simulación en la que se muestra el funcionamiento del circuito que implementa una compuerta NOR2 ocupando el estilo escogido. Para estas simulaciones, se toma en cuenta el modelo de *memristor* VTEAM explicado en el capítulo 1. Se construye entonces el circuito que utiliza el bloque con *memristors* de *pull-down* como se muestra en la figura 2.2-b. En este caso, para el transistor PMOS se emplea una tecnología de 50 nm con una relación W/L=2 (razón entre ancho del transistor y largo del canal) y un voltaje V_{DD} = 1 V. Por otro lado, se elige un voltaje V_{read}=0.2 V, el cual corresponde al máximo voltaje que cae sobre los terminales de los dispositivos. Como criterio para la selección de V_{read}, se contempla el umbral más bajo del dispositivo, el cual según el modelo escogido es de 0.3 V. En consecuencia, dado que el voltaje máximo sobre los *memristors* durante la operación lógica no supera los umbrales de conmutación, se espera que las operaciones de cómputo no produzcan la conmutación indeseada de estos.

Finalmente, utilizando LTspice se realizan tres simulaciones para el circuito de compuerta NOR2 para tres combinaciones distintas de entradas. Los resultados de la simulación se presentan en la figura 2.4, en donde se muestra el voltaje V_{Op} que se aplica al terminal de la puerta (gate) el transistor PMOS para el comienzo de la operación lógica, y los voltajes V_{out} para la combinación de entradas A\B = 0\0, 1\0, 1\1. En esta simulación, se puede notar cómo los voltajes de salida varían de acuerdo a la combinación de entradas tal como se menciona previamente, siendo el caso en donde se alcanza la mayor magnitud aquel correspondiente a las entradas A\B = 0\0. Tal como se esperaba, acorde a la tabla 2.1, este voltaje resulta igual a $V_{out} \approx VDD = 1V$, alejándose de los valores de voltaje de salida para el resto de combinaciones de entradas. De esta forma, considerando que este último es el caso en el que se tiene un '1' lógico, se puede comenzar a hablar de un criterio para la selección de V_{comp}. Tomando en cuenta que los valores de V_{out} para A\B = 1\0 y A\B = 1\1 se aproximan a V_{REF}, el criterio que se utiliza para escoger el voltaje de comparación es el valor medio entre V_{REF} y V_{DD}. Por lo tanto, se tiene que $V_{Comp} = (V_{DD} + V_{REF})/2 = 0.9$(V).

Otro aspecto que es importante resaltar corresponde a la cercanía entre los valores de voltaje V_{out} para A\B = 1\0 y A\B = 1\1. Esto se produce principalmente por el diseño del transistor PMOS para el cual no se toma en cuenta la consideración de $R_L \approx R_{ON}$. Como se muestra en las simulaciones, esto no genera mayor problema si se cumple que $R_{OFF} >> R_L$. En este caso, $R_L = 16\ K\Omega$, lo que es mayor que $R_{ON} = 1\ k\Omega$ pero

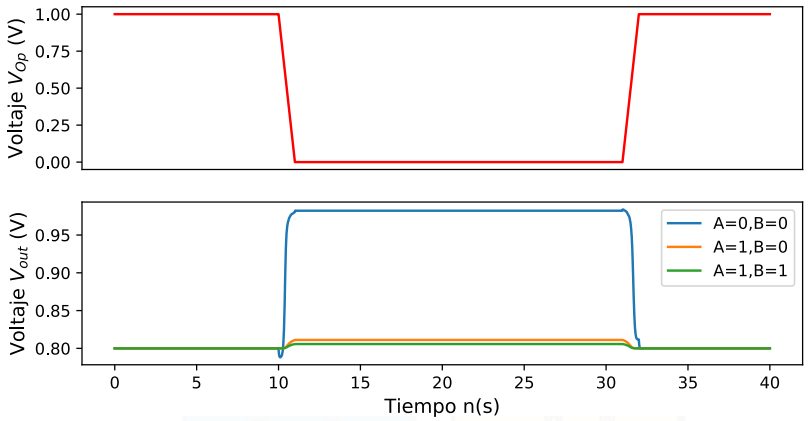


Figura 2.4: Simulación en LTspice de operación de la compuerta NOR2 para (A\B = 0\0,1\0,1\1) usando el estilo *Ratioed Logic*. El tiempo de interés comprende entre los 10 ns y 30 ns cuando V_{Op} es un lógico cero y se realiza la operación lógica.

mucho menor que $R_{OFF} = 300\ K\Omega$, produciendo así que los voltajes para estas combinaciones de entradas estén cerca de V_{REF} y sean similares entre sí.

En resumen, en este capítulo se selecciona el estilo de lógica "Ratioed Logic" que se utiliza para implementar compuertas NOR, NOT y copias de datos en la memoria con ayuda de circuitos de la periferia. En dicho sentido, para llevarlo a una operación más robusta se cumplen con condiciones

3 | Diseño de Arquitectura de Memoria ReRAM

3.1. Arreglo *Crossbar* de *Memristors*

Una de las características de los *memristors* es que su fabricación puede llevarse a cabo en estructuras de arreglos denominadas *crossbar* (barras cruzadas) las cuales son piezas centrales en la construcción de memorias ReRAM. Estas estructuras organizan las celdas de memorias en arreglos de dos dimensiones de forma que aquellas que se encuentren en una misma fila tengan el terminal superior asociado a un nodo común, a la vez aquellas celdas en una misma columna tienen su terminal inferior conectado a un mismo nodo. De esta forma las líneas de las filas se les denomina *"Bit Line"*(BL), mientras que aquellas líneas asociadas a las columnas se denominan *"Word Line"*(WL). Esta es la estructura que se muestra en la figura 3.1-a.

La topología crossbar de una memoria ReRAM presenta una serie de ventajas en comparación a las memorias convencionales como las DRAM y las memorias flash. Dentro de las limitaciones que presentan estas últimas se encuentran las posibilidades de escalamiento, debido a los limites fundamentales que está alcanzando la tecnología CMOS de la cual se componen estas memorias. Lo anterior se contrasta con las tecnologías emergentes como la que se presenta en este trabajo, que permiten una integración tal que alcanzan dimensiones de $4F^2$ (siendo F el ancho de los contactos que construyen las líneas de contacto de los *memristor*)

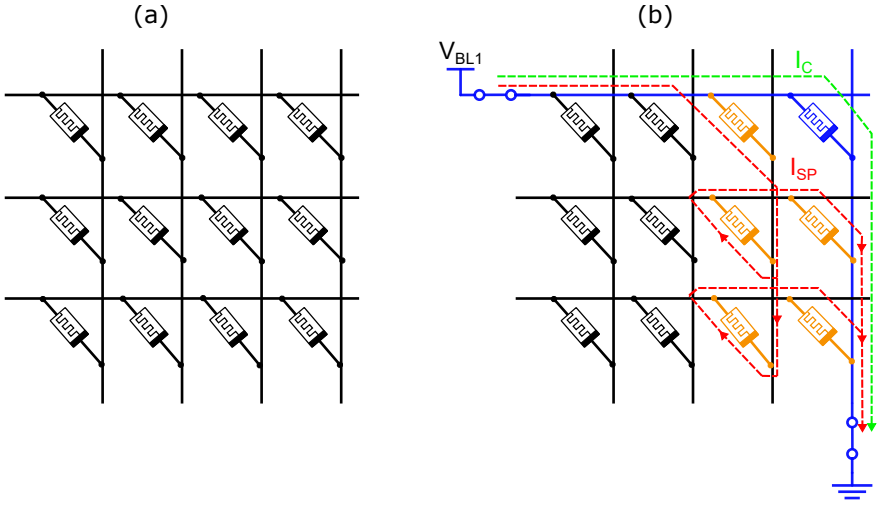


Figura 3.1: (a) Esquema que muestra el circuito de un arreglo *crossbar* de *memristors*. (b) Efecto de corrientes de fuga en el arreglo de memoria

A pesar de las características positivas que presenta esta topología existen ciertos inconvenientes que se deben tomar en cuenta al momento de operar el circuito, siendo el principal de estos las corrientes de fuga o corrientes parásitas (*Sneak path current*)[41]. Este problema surge producto del tipo de conexión que existen entre las celdas de memoria, en donde para realizar una operación y aplicar voltaje sobre las lineas (BL o WL) teniendo como objetivo un *memristor* determinado, la corriente que fluye hacia el nodo de salida no dependa solo de la resistencia del dispositivo objetivo, sino que también del estado resistivo de los demás elementos del arreglo. Esto es precisamente lo que se ilustra en la figura 3.1, en donde se pretende acceder al dispositivo de la posición (1,4) que se muestra en azul y por donde pasa la corriente I_c, pero debido a que prácticamente todos los demás *memristors* del arreglo están interconectados estos actúan como una resistencia parásita. Por lo anterior existe una corriente parásita I_{SP} que también fluye hacia el nodo de salida (se muestra en el camino en color naranja) y puede afectar el resultado de la operación de lectura, así como también alterar el contenido de las celdas en el camino de la corriente parásita.

Una forma de reducir esta problemática de corrientes parásitas en arreglos crossbar pasivos donde existen únicamente *memrsitors*, es añadir un elemento de selección a cada celda de memoria, es decir, en cada cross-point del arreglo fabricar un dispositivo de selección en serie con el *memristor*. Dentro de los elementos más utilizados para este propósito se encuentra el transistor conectado en serie con el *memristor*, lo que permite configurar una topología de memoria resistiva 1T1R como se muestra en la figura 3.2-a. De esta forma cada *memristor* se accederá activando los dispositivos selectores de la columna correspondiente al elemento de interés. Se activa la columna completa debido a que no se puede tener tantas líneas de activación como número de transistores en la memoria, ya que se vuelve inviable la construcción de un circuito de control para el manejo de tantas pistas y señales. A pesar de que la señal de activación llega a todos los transistores de la columnas aquellos que no sean parte de la operación no se verán afectados tal como se muestra en la representación tridimensional de la topología de memoria en la figura 3.2-b, en donde se observa como la corriente se desplaza por la línea de la columna sin pasar a las líneas de fila que no le corresponden, esto producto a la alta impedancia de los transistores en corte. Considerando esto, el trabajo presentado posteriormente asume la topología de memoria 1T1R como la base para la memoria ReRAM utilizada.

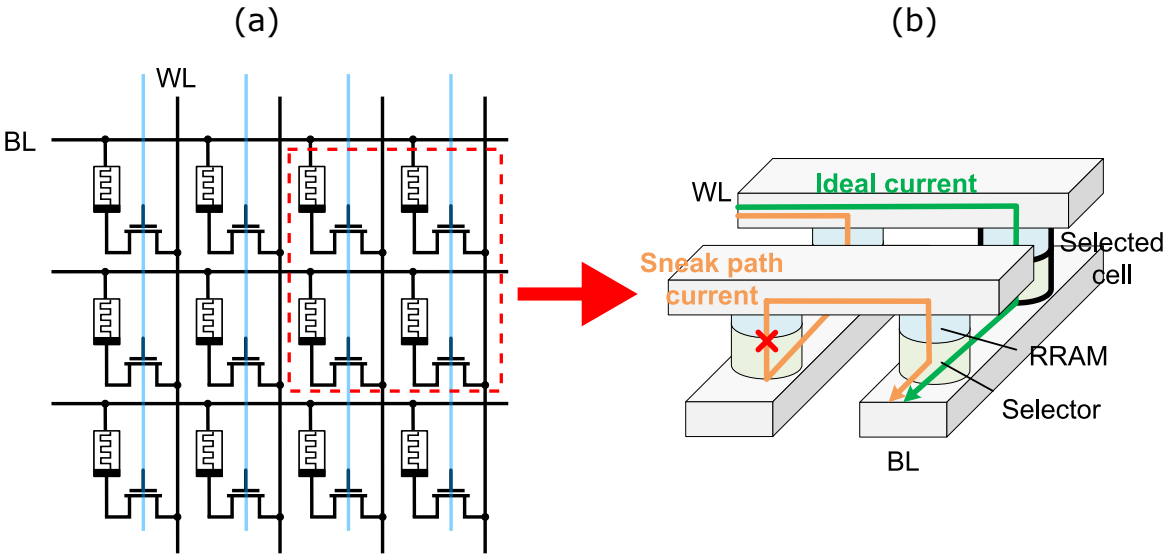


Figura 3.2: (a) Esquema que muestra el circuito de un arreglo *crossbar* 1T1R. (b) Ilustración tridimencional de una sección del arreglo, indicando un ejemplo de cómo los dispositivos selectores desactivados podrían bloquear las corrientes parásitas dentro del arreglo

3.1.1. Implementación de computación en la memoria

Previamente se explicó el funcionamiento de la topología crossbar que es el núcleo de una memoria ReRAM, y la forma de realizar computación lógica con *memristors*, pero ahora es necesario integrar ambos conceptos en un diseño de memoria computacional que permita la funcionalidad completa de una memoria no volátil y a la vez la implementación de computación en la memoria. De esta forma se tiene como base el circuito de compuerta NOR2 presentado en la figura 2.2 desde el cual se puede comenzar a construir una memoria computacional. Precisamente el circuito NOR2 se puede extender para un numero arbitrario de entradas, esto se puede lograr asociando las entradas a las columnas de la memoria tal como se presenta en el circuito de la figura 3.3 en donde el número de entradas de la compuerta NORn a calcular es equivalente al número de transistores que se activen mediante las líneas WL. Como se puede observar este circuito se asocia a una fila de la topología de memoria ReRAM, en el cual se denotan los voltajes y señales de control necesarias para comenzar la computación de una compuerta NORn y obtener el resultado como una señal digital a la salida del comparador. No obstante, no se tiene el circuito necesario para implementar las operaciones propias de una memoria funcional. Las operaciones fundamentales de una memoria corresponden a la escritura y lectura de los datos almacenados, a estas ahora se les suma la operación de cómputo de compuertas lógicas.

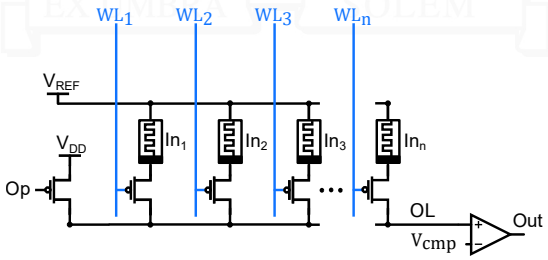


Figura 3.3: Topología que permite la implementación del estilo *Ratioed Logic* en una fila de una memoria ReRAM. Durante una operación NORn, el voltaje en el nodo OL pasa por el comparador y se determina el estado lógico correspondiente.

De esta forma se debe diseñar el circuito de periferia de la memoria que permita la implementación de todas las operaciones básicas de una memoria computacional previamente descritas. Estas operaciones se pueden dividir en dos categorías, que son las operaciones de escrituras y cómputo. Dentro de estas últimas se cuentan las operaciones de lectura, dado que, en el estilo de lógica elegido, y como se explicó en el capítulo 2, la ejecución de una operación lógica es equivalente a una lectura desde la memoria ReRAM. En este sentido vale la pena recordar que la operación de lectura se realiza negando el resultado de la computación de la operación lógica NOT, tal como se expuso en el capítulo anterior (circuito figura 2.3). A su vez para realizar el computo de la compuerta NOT se utiliza el mismo procedimiento que para computar una compuerta NOR, pero solo considerando un *memristor*.

A partir de la descripción anterior se diseña el circuito de periferia de la línea de memoria que permite controlar la unidad de almacenamiento, circuito que se presenta en la figura 3.4. En esta figura se individualiza cada uno de los circuitos de control que componen la periferia de la memoria. En primer lugar, se tiene el circuito de control del bit line (Control BL) en el bloque superior izquierdo. Este bloque es el encargado de definir el voltaje que recae sobre el nodo BL, el cual comparten todos los electrodos superiores de los dispositivos que componen una fila de memoria. Sobre este nodo se aplica un voltaje de referencia V_{REF} en caso de cómputo de compuertas lógicas o lectura, además de un voltaje V_{SET} o tierra si es que se realiza la operación escritura de algún dispositivo. Esta operación de escritura puede provenir del resultado de una operación interna como lo es la lectura de una celda o computo de operación lógica, resultados que quedan retenidos en la señal V_{FF} proveniente del bloque de salida. En su defecto puede ser una operación de escritura proveniente de un valor externo a la memoria, el que está dado por la señal V_W. En la zona inferior izquierda se tienen dos bloques que trabajan en conjunto, uno que implementa los circuitos de control de la línea de salida (Control OL) y un circuito de control auxiliar (Control Auxiliar). Estos circuitos actúan en conjunto y tienen la función de definir el voltaje que recae sobre el nodo OL que comparten los electrodos inferiores de

las celdas 1T1R. El bloque de control auxiliar se utiliza, como se presentará más adelante, para los casos en los que se realice una operación de escritura que implique un RESET de alguna celda, de esta forma para evitar señales negativas que pueden alterar la operación de los transistores, se aplica esta señal positiva en el electrodo inferior. Para el resto de operaciones, en las que se utilice esta línea, se aplica la señal generada por el bloque Control OL, que puede ser tierra para el caso de escrituras que no sean RESET o V_{DD} para el computo de una compuerta lógica/lectura. Por último, se tiene un bloque que permite controlar la señal de salida (Control Salida) al momento de realizar un proceso de cómputo de compuerta lógica y lectura. Este circuito este compuesto por el comparador que permite diferenciar los voltajes generados por las distintas combinaciones de entrada, esto cuando se realiza la computación de compuertas NOR/NOT. Se incluye también un multiplexor 2x1 que permite elegir entre la señal original o la versión negada de la salida del comparador, la cual se utiliza para el proceso de lectura (resultado de negar el computo de una compuerta NOT). Por último en este bloque se incluye también un FLIP-FLOP que permite capturar y mantener el resultado de una operación lógica y de lectura en la señal V_{FF}, la cual es posteriormente realimentada al bloque de control BL.

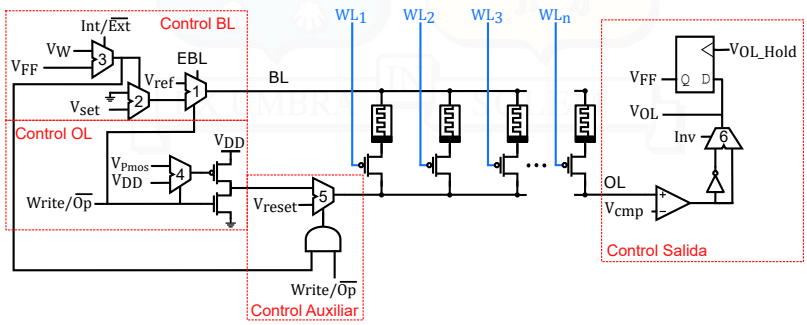


Figura 3.4: Esquemático del driver completo de una fila de un arreglo de memoria resistiva

Para una comprensión completa del uso de este circuito de memoria reducido a una fila, se presentará a continuación las operaciones soportadas, los circuitos involucrados destacados en negro y las señales que se aplican destacados en rojo.

- Operación lógica NOR2: El circuito realizando esta operación se puede observar en la figura 3.5. En este caso se considera una operación entre los dispositivos M1 y M2, de esta forma el computo comienza con la aplicación de EBL = '1' que activa el MUX 1, además se aplica $Write/\overline{Op}$ = '0' que produce que el MUX 1 aplique V_{ref} sobre BL y el MUX 4 active el transistor conectado a su salida para aplicar V_{DD} sobre OL. De igual forma se activan los transistores PMOS asociados a los dispositivos involucrados, en este caso $WL_1 = WL_2$ = '0'. El voltaje resultante en la linea OL (proporcional a los estados resistivos de los dispositivos como se explicó en el capítulo anterior) es capturado por el comparador obteniéndose un valor digital. En este caso el valor de la señal INV = '0' por lo cual pasa la señal de salida del comparador por el MUX 6 y es capturada por el FLIP-FLOP, el cual utiliza V_{OL_Hold} como señal de reloj.

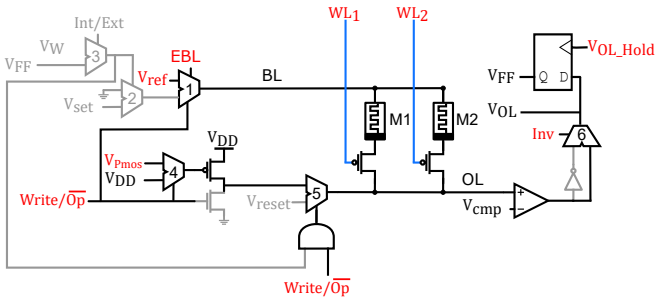


Figura 3.5: Esquemático del driver completo de una fila de un arreglo de memoria resistiva en donde se destacan las partes del circuito que participan en la operación lógica NOR2

- Operación lógica NOT: El computo de esta operación es similar al realizado para la compuerta NOR, con la salvedad que en este caso solamente se utiliza un dispositivo. El circuito computando esta compuerta lógica, teniendo como entrada el dispositivo M1, se puede observar en la figura 3.6. En este caso se activa WL_1 = '0' mientras que WL_2 = 1. El resto de las señales se mantienen igual al caso anterior en el que se computa la compuerta NOR.

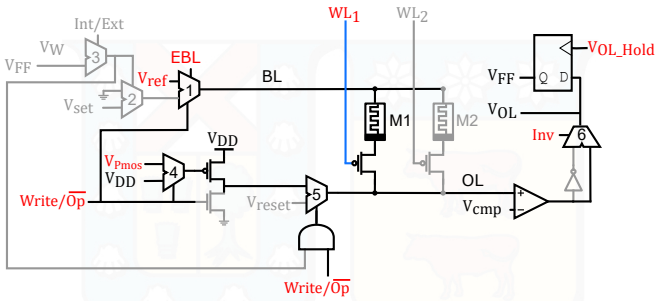


Figura 3.6: Esquemático del driver completo de una fila de un arreglo de memoria resistiva en donde se destacan las partes del circuito que participan en la operación lógica NOT

- Operación de lectura: Como se mencionó en ocasiones previas esta operación es equivalente a negar el resultado del cómputo de la compuerta NOT utilizando como entrada el dispositivo que se quiere leer. En este caso si se quisiera leer el contenido del dispositivo M1, se activaría el circuito que se observa en la figura 3.7. De esta forma al igual que en el caso anterior se tiene WL_1 = '0' y WL_2 = '1', además se tiene que $Write/\overline{Op}$ = '0' al ser un procedimiento equivalente al cómputo de una operación lógica. Por otro lado, el resto de las señales se mantienen igual al caso del cómputo de compuerta NOR y NOT, teniendo como única diferencia el tratamiento de la señal INV. Como se mencionó anteriormente en la operación de lectura se está computando una compuerta NOT y el valor original se obtiene negando este resultado, de esta forma el MUX 6 debe entregar a la salida la versión de la señal de salida del comparador que pasa por el negador, por lo que INV = '1'.

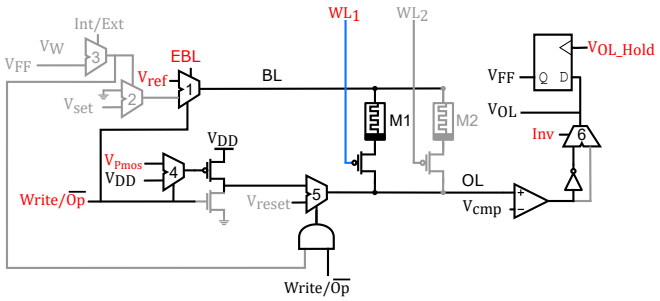


Figura 3.7: Esquemático del driver completo de una fila de un arreglo de memoria resistiva en donde se destacan las partes del circuito que participan en el proceso de lectura de una celda de memoria

- Operación escritura: Esta operación permite almacenar un valor lógico en una celda de memoria, este valor puede provenir de dos fuentes. En primer lugar, se puede tener una escritura interna, en donde el valor que se quiere almacenar proviene del cómputo de una operación lógica o la lectura de un dispositivo en un ciclo previo, en este caso el valor a escribir se encuentra retenido en V_{FF}. En segundo lugar, se puede tener una escritura externa proveniente del elemento de la unidad e control de la memoria, cuyo valor esta dado por la señal de control V_W. Para definir si la escritura es interna/externa se utiliza la señal de control equivalente a $Int/\overline{Ext}$ = 0/1. Además toda operación de escritura comienza con $Write/\overline{Op}$ = '1' y se activa el MUX 1 mediante la señal EBL = '1'.

  A su vez también se tienen dos posibles valores que pueden ser almacenados en la memoria, se puede realizar la escritura de un lógico '1', que equivale a un proceso de SET, o realizar la escritura de un lógico '0', equivalente a un proceso de RESET. Como ejemplo se tiene el caso de que se realice un

proceso de escritura de un lógico '1' sobre el dispositivo M1, en donde las zonas del circuito que se activan se destacan en la figura 3.8. En este caso la señal que sale por el MUX 3 corresponde a un '1' lógico (el valor que se quiere escribir), por lo que este es el que permite que el MUX 2 entregue el voltaje V_{SET} y que la compuerta AND conectado al control del MUX 5 conecte la linea OL a tierra mediante el transistor NMOS. Por otro lado también se puede realizar la escritura de un lógico '0', para este proceso se activa las partes del circuito que se muestran en la figura 3.9. En este caso el valor lógico que sale del MUX 3 es igual a '0', por lo que el MUX 2 conecta la linea BL a tierra y la compuerta AND conectada al MUX 5 produce un lógico '0', de forma que sobre la linea OL se aplica V_{reset} completando la operación de escritura. Se realiza este proceso de escritura diferenciado para evitar utilizar voltajes negativos sobre las celdas 1T1R que generarían problemas con los transistores PMOS.

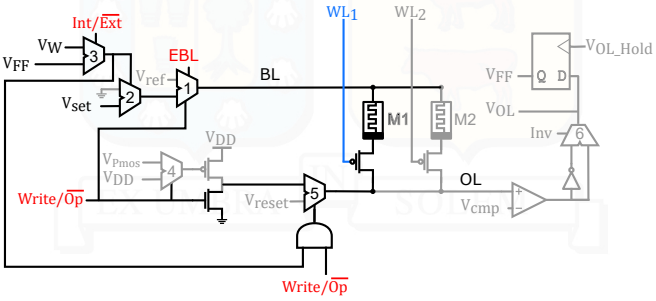


Figura 3.8: Esquemático del driver completo de una fila de un arreglo de memoria resistiva en donde se destacan las partes del circuito que participan en el proceso de escritura SET de una celda de memoria

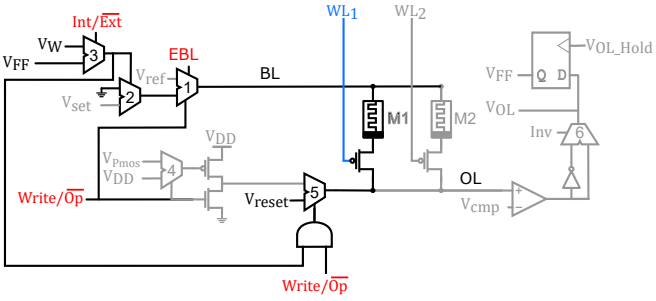


Figura 3.9: Esquemático del driver completo de una fila de un arreglo de memoria resistiva en donde se destacan las partes del circuito que participan en el proceso de escritura RESET de una celda de memoria

A modo de resumen de las operaciones que se pueden realizar en la memoria se presenta en la tabla 3.1 que valores se deben aplicar a cada una de las señales de control para ejecutar las distintas operaciones de una memoria y para la computación lógica.

	Valor de las señales de control					
Operación	$Write/\overline{Op}$	EBL	WL_i	WL_j	$Int/\overline{Ext}$	Inv
Lectura	0	1	1	0	X	1
Escritura Externa	1	1	1	0	0	X
Escritura Interna	1	1	1	0	1	X
NOT/Copy	0	1	1	0	X	0/1
NOR/OR	0	1	1	1	X	0/1

Tabla 3.1: Valor de las señales de control necesarias para las distintas operaciones de memoria y computación lógica

Con el circuito de control para una fila de memoria ReRAM diseñado, se pasa a una fase de simulación para obtener un flujo de operaciones que demuestren el funcionamiento de esta linea de memoria. Al igual que

las simulaciones anteriores estas se realizan en el ambiente de LTSpice utilizando el modelo de *memristors* VTEAM, mientras que para los transistores se utilizó un modelo de canal corto de 50 nm publicado en [42]. Los transistores selectores se diseñaron un un tamaño de W/L = 20 mientras que los otros transistores se diseñaron con W/L = 10. El resto de los componentes corresponden a modelos ideales soportados por LTSpice. Para alimentar los componentes se utilizó un voltaje de alimentación igual a 2.5 V, no obstante V_{DD} del circuito se mantuvo en 1 V. En esta ocasión se han considerado solo dos celdas de memoria (dos WL). En total se realizan 8 operaciones cuya ejecución se muestra en la figura 3.10, en la cual se observa la evolución de los valores de resistencia de ambos dispositivos presentes en la memoria que participan de las operaciones en cuestión, además de las señales de control necesarias para la operación y otras señales de interés.

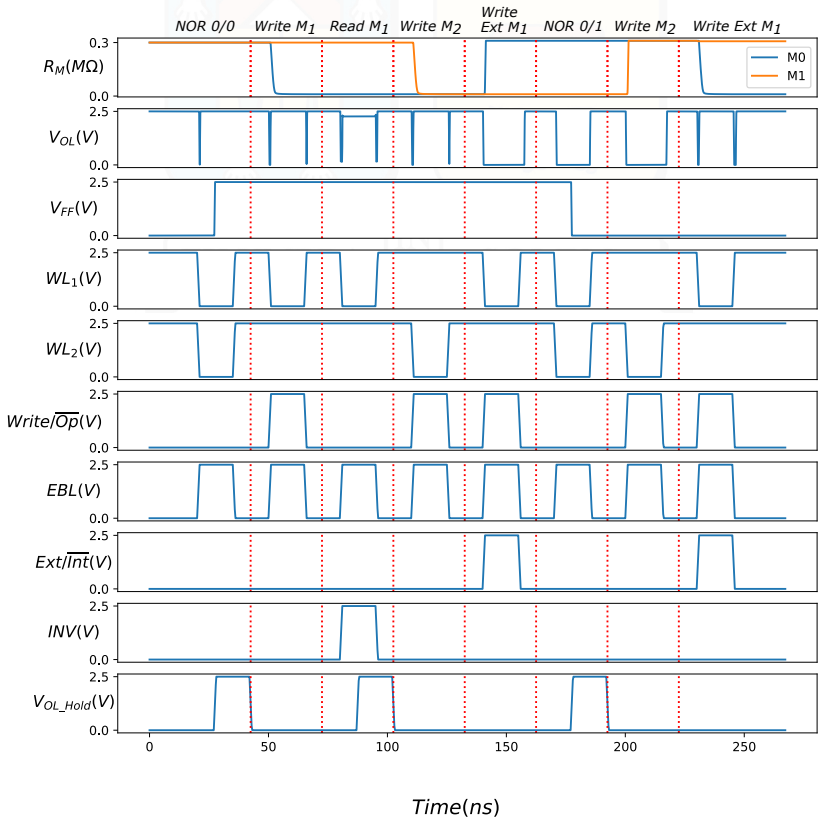


Figura 3.10: Simulación ejecución operaciones sobre línea de memoria resistiva

La serie de operaciones realizadas se pasa a describir a continuación. En primer lugar, ambos dispositivos que serán parte de la prueba se inicializan en alta resistencia (HRS). La primera operación corresponde a un NOR2 entre ambos dispositivos que están en estado 0/0. La operación comienza aplicando $WL_1 = WL_2 =$ '0', $Write/\overline{OP} =$ '0' y $EBL =$ '1'. El resultado de esta operación se puede observar en la señal de interés V_{FF} (durante el tiempo en el que EBL se encuentra en alto) correspondiente a un '1' lógico el cual es mantenido por el Flip-Flop previa captura de V_{OL} con el canto de subida de V_{OL_Hold}. La operación N°2 corresponde a una escritura del resultado calculado anteriormente en el dispositivo M1. Como se observa en la evolución de resistencias R_M, y producto de un resultado igual a '1' lógico en el cómputo anterior, se produce la conmutación del estado resistivo de M1 a baja resistencia (LRS). Para realizar esta operación se aplican las señales $Write/\overline{Op} = EBL =$ '1' y $WL_1 =$ '0', además por tratarse de una escritura interna se tiene $Int/\overline{Ext} =$ '0'. La operación N°3 corresponde a una lectura del dispositivo M1 para corroborar el valor almacenado en la operación anterior. Para esta operación se activa $EBL =$ '1' y se tiene $WL_1 = Write/\overline{Op} =$ '0' además de activar INV = '1' para invertir el resultado. El valor leído se puede observar en la señal V_{OL}, tratándose del '1' lógico obtenido en la operación anterior. Esta lectura también es capturada por el Flip-Flop al activar V_{OL_Hold}, pero como V_{FF} ya se encontraba en el valor '1' lógico no se observan cambios en esta señal. En la operación N°4 se aprovecha el valor leído previamente de M1 (lógico '1') para almacenarlo en

el *memristor* M2 produciendo la conmutación de resistencia de este como se observa en la señal R_M. Para este caso se tienen las señales WL_2 = '0' y $Write/\overline{Op} = EBL$ = '1'. Al tratarse de una escritura interna se tiene que $Int/\overline{Ext}$ = 0. En la operación N°5 se escribe un '0' lógico de manera externa en el dispositivo M1 produciendo nuevamente la conmutación de este como se observa en la señal R_M. Nuevamente para el proceso de escritura se activa $Write/\overline{Op} = EBL$ = '1' y por tratarse de M1 se produce WL_1 = '0'. Al tratarse de una escritura externa se tiene que $Int/\overline{Ext}$ = '1'. En la operación N°6 se realiza el computo de una compuerta NOR2 entre dos dispositivos en estado 0/1. Esta vez el resultado corresponde a un '0' lógico como se corrobora en la señal V_{OL} y es capturado por V_{FF} utilizando la señal V_{OL_Hold} = '1'. En esta ocasión se puede notar el cambio de estado de '1' a '0' en la señal V_{FF}. Al tratarse del cómputo de una compuerta NOR se activa EBL = '1' y se tiene $WL_1 = WL_2 = Write/\overline{OP}$ = '0'. La operación N°7 corresponde a la escritura del resultado obtenido en la operación N°6 en el dispositivo M2. Se puede observar en la señal R_M la conmutación de M2 de baja a alta resistencia por la escritura del '0' lógico obtenido previamente. Para esta operación nuevamente se tiene que WL_2 = '0', $EBL = Write/\overline{OP}$ = '1'. Además por tratarse de una escritura interna se tiene $Int/\overline{Ext}$ = '0'. Por último, en la operación N°8, se realiza una escritura externa de un '1' lógico en el dispositivo M1. Se activa la línea de M1 produciendo EBL = '1' y WL_1 = '0'. La operación se inicia realizando $Write/\overline{Op}$ = '1' y al tratarse de una escritura externa se tienen $Int/\overline{Ext}$ = '1'. El resultado de la escritura se puede observar en la señal R_M con la conmutación de alta a baja resistencia por parte de M1.

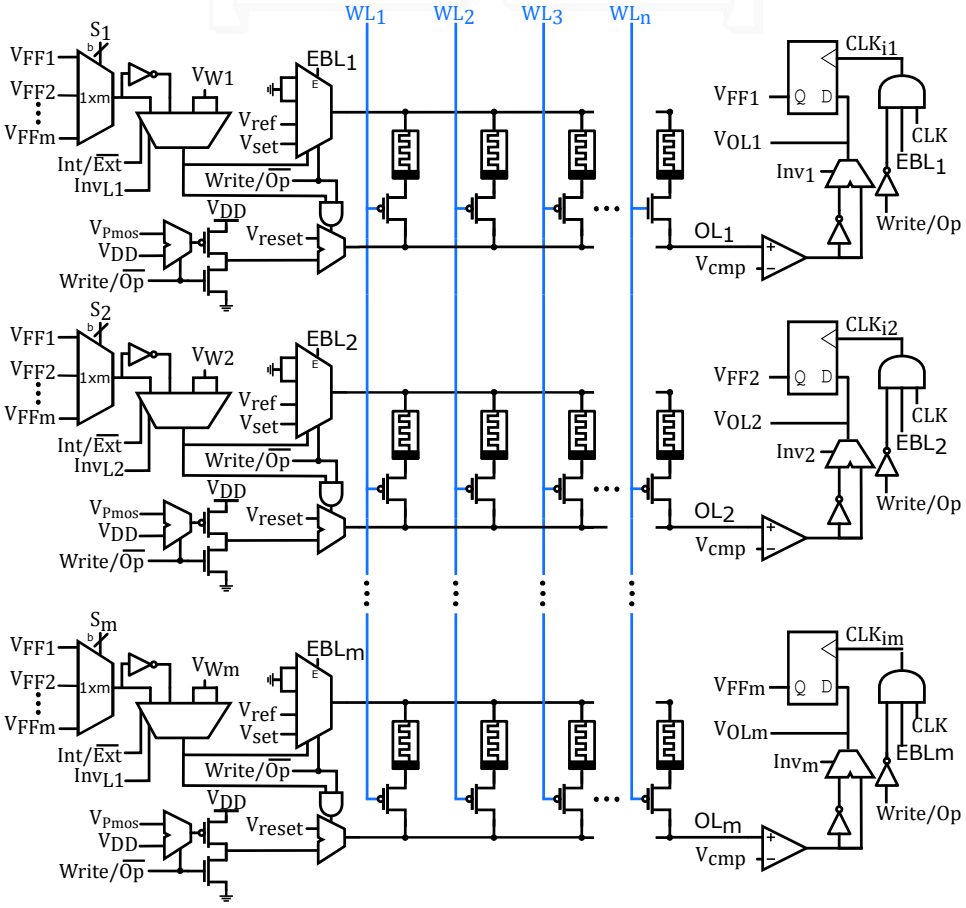


Figura 3.11: Topología de memoria resistiva funcional

Finalmente, para tener un arreglo de memoria resistiva completa se extiende el diseño de una fila de n columnas presentado previamente a un diseño completo en el que se tienen m filas por n columnas. Esto es lo que se presenta en la figura 3.11 en donde se completó el diseño del circuito de memoria de una sola fila para ahora incluir la posibilidad de m salidas. Dentro de los cambios que se hizo en el esquema se tiene la unión de los multiplexores 2x1 a MUX 4x1 y la inclusión de un multiplexor mx1 que permite seleccionar cualquier

salida para almacenarla en cualquier línea. Además se incluyó un circuito que permite generar la señal de control CLk_{i1} de cada Flip-Flop para que este dependiera de las señales de control $Write/\overline{Op}$, EBL_i y de una señal de reloj CLK que permitiera operaciones sincrónicas. Notar que debido a la topología del diseño que permite operar con cada fila de la memoria de manera independiente, se podrá realizar computación en paralelo de m compuertas NOR, esto bajo la condición que las entradas se encuentren alineadas en las mismas columnas. Este último aspecto de computación en paralelo será relevante para la implementación de sistemas más complejos tal como se verá en siguientes secciones.

3.2. Evaluación de la fiabilidad de en operaciones lógicas y de memoria

Un factor determinante en lo que respecta a la fiabilidad de las operaciones de lectura y computación lógica de la memoria es el voltaje de comparación V_{cmp}. Una correcta selección de este parámetro permite que en la etapa de comparación el voltaje V_{OL} para un lógico '0' y un lógico '1' sean claramente distinguibles entre ambos. De esta manera se puede definir el nivel de voltaje $V_{OL1\_1}(V_{OL1\_0})$ correspondiente a un lógico '1'(lógico '0') en donde se busca que la diferencia de voltaje entre V_{OL1_1} y V_{OL1_0} sea la mayor posible. Por otro lado el estilo de lógica *"Ratioed Logic"* plantea en la ecuación 2.1 que el voltaje de salida V_{out} depende directamente de la resistencia equivalente R_{eqn} y por lo tanto de la relación $k = \frac{R_{off}}{R_{on}}$. De esta forma es posible explorar el impacto del parámetro k sobre la ventana de voltaje mínimo $|V_{OL\_1} - V_{OL\_0}|$ que determina la fiabilidad de las operaciones NOR del estilo de lógica.

El computo de la compuerta NOR con entradas '00' significa tener dos *memristors* en alta resistencia R_{off} lo que resulta en un voltaje de salida V_{OL_1}. Por otro lado dentro de las combinaciones de entradas que resultan en lógico '0', sin considerar el fan-in de la compuerta, se tiene que la más crítica es aquella que solo cuenta con uno de los dispositivos en R_{on}, esto pues el correspondiente voltaje V_{OL_0} es el más cercano a V_{OL_1} y resulta en el mínimo margen de voltaje para la selección de V_{cmp}. En este análisis se consideran dos casos: (a) El primer caso correspondiente a $R_{off} >> R_{on}$ (más de dos órdenes de magnitud), y (b) el caso en el que la condición anterior no aplica. De esta forma considerando la ecuación 2.1 y asumiendo $R_L \approx R_{on}$ (el cual en el análisis se simplificara a $R_L = R_{on}$), se tiene que para la combinación de entradas '00' la resistencia equivalente $R_{eq,00} = \frac{R_{off}}{2}$. De esta forma a la salida de nodo de cómputo de la compuerta lógica se tiene el nivel de voltaje dado por:

$$V_{OL\_1} = \begin{cases} V_{DD}, & R_{off} >> R_{on} \\ V_{REF} + (V_{DD} - V_{REF})\frac{1}{\frac{2}{k}+1}, & EOC \end{cases} \tag{3.1}$$

La ecuación 3.1 nos entrega el valor de V_{OL_1} más alto y más bajo que es posible obtener. De esta forma se puede observar la dependencia que este nivel de voltaje tiene de la razón de resistencia k, ya que a medida que este parámetro disminuye su valor el nivel de voltaje V_{OL_1} se aleja de V_{DD} (el máximo nivel de voltaje posible). De igual forma se repite este análisis, pero ahora considerando uno de los dispositivos en estado R_{on}, concretando de esta forma una combinación de entradas lógicas '01'. En este caso se obtiene un nivel de voltaje $R_{eq,01}$ dado por:

$$R_{eq,01} = \begin{cases} R_{on}, & R_{off} >> R_{on} \\ \frac{R_{off}}{1+k}, & EOC \end{cases} \tag{3.2}$$

De esta forma de acuerdo a la ecuación 3.2 en conjunto a la ecuación 2.1 se obtiene el nivel de voltaje V_{OL_0} dado por:

$$V_{OL\_0} = \begin{cases} \frac{1}{2}(V_{REF} + V_{DD}), & R_{off} >> R_{on} \\ V_{REF} + (V_{DD} - V_{REF})\frac{1}{\frac{k+1}{k}+1}, & EOC \end{cases} \tag{3.3}$$

Para el caso de la ecuación 3.3 el factor k no tiene un gran impacto en los valores más altos y más bajos, esto en contraste a lo ocurrido en la ecuación 3.1. De todas formas el caso más relevante es la

diferencia mínima entre V_{OL_0} y V_{OL_1} el cual ocurre en la situación (b) descrita previamente, esto cuando k es relativamente pequeño. Considerando a modo de ejemplo los parámetros utilizados para las simulaciones previas $V_{DD} = 1V$ y $V_{REF} = 0.8$ V y un valor de $k = 10$, se obtiene una ventana de voltaje ΔV_{OL} de $V_{OL\_1} - V_{OL\_0}$ = 0.966 V - 0.895 V = 71 mV. Vale la pena mencionar que esta ventana podría empeorar hasta llegar a niveles de los 20 mV para el peor caso en el que se tiene que k = 2. Para una mayor comprensión acerca del impacto que tiene este parámetro en la diferencia de voltaje, se gráfica en la figura 3.12 la evolución de ΔV_{OL} en comparación a k. Se observa que el valor de la ventana de voltaje ΔV_{OL} se aproxima a $\frac{1}{2}(V_{DD} - V_{REF})$, en este caso 0.1 V, que es cuando k excede los dos órdenes de magnitud (equivalente a decir que R_{OFF} excede en dos órdenes de magnitud a R_{ON}). Por otro lado, cuando k se aproxima a 2, la diferencia de voltaje decrece rápidamente. Vale la pena notar también que esta diferencia de voltaje podría mejorarse escogiendo otros valores de V_{REF}, no obstante, la elección de este voltaje presenta limitaciones importantes, dado que depende directamente de los umbrales de voltaje para SET y RESET de los *memristors*, de forma de evitar que las operaciones realizadas destruyan el contenido de los mismos. Este estudio resalta algunas de las dificultades que implica diseñar este tipo de topología y eventualmente la dependencia que tenga el desempeño de la memoria del *memristor* y sus características.

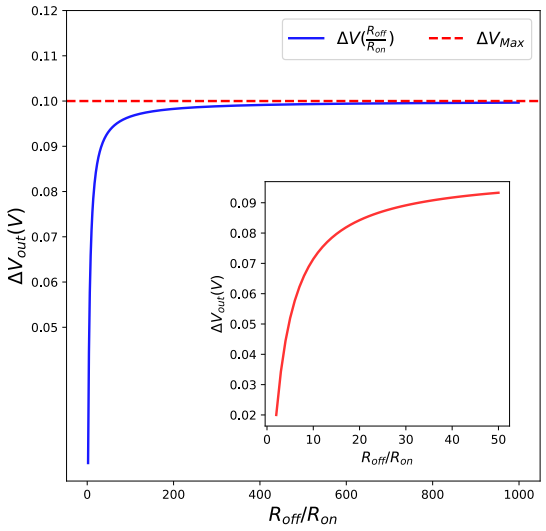


Figura 3.12: Evolución $\Delta V_{OL} = V_{OL\_1} - V_{OL\_0}$ para un rango amplio del parámetro $k = \frac{R_{off}}{R_{on}}$ basados en la ecuación 2.1 asumiendo V_{DD} = 1 V y $V_{REF} = 800mV$. En el gráfico ampliado se observa parte de la curva para valores de k en el rango $2 \leq k \leq 50$

3.3. Propuesta de topología alternativa

El problema descrito en la sección anterior se origina de la dependencia del voltaje de salida de la razón de resistencias k y del hecho de que el parámetro de voltaje V_{REF} era la única variable de diseño. En este sentido ahora se propone una solución alternativa para aumentar la fiabilidad del sistema, la cual está basada en lo mostrado en *"Scouting Logic"* [40]. De esta forma se propone basar la operación de función lógica en el funcionamiento del amplificador sumador, esto como reemplazo del divisor de tensión utilizado hasta ahora en el cómputo de funciones lógicas con *memristors*. En la figura 3.13 se muestra el circuito que combina el amplificador sumador con los elementos de memoria en donde el voltaje de salida está gobernado por la ecuación 3.4.

$$V_{out} = -V_{REF}(\frac{R_F}{R_{M1}} + \frac{R_F}{R_{M2}}) = -V_{REF}\frac{R_F}{R_{eq}} \tag{3.4}$$

De esta forma esta topología permite la computación de la compuerta NOR de dos entradas al activar dos dispositivos y aplicar un voltaje V_{REF} en la entrada, obteniendo de esta forma distintos niveles de voltaje de acuerdo a los niveles de resistencia en los que se encuentren los dispositivos. Comparado con la topología del divisor de tensión utilizada previamente, esta nueva propuesta introduce un parámetro de diseño extra, la resistencia de feedback R_F, la cual puede ser seleccionada según el parámetro k y el valor de voltaje V_{REF}. De igual forma existen limitaciones en el diseño, en particular el valor de V_{REF} el cual debe ser negativo y su valor absoluto no debería superar el valor umbral de cambio de SET/RESET de los dispositivos, en particular el menor valor entre los dos (en valor absoluto).

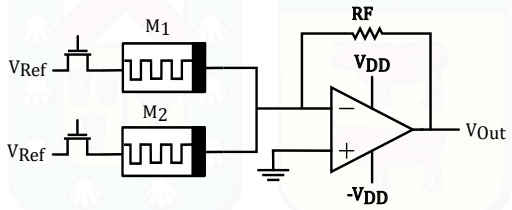


Figura 3.13: Circuito amplificador sumador con *memrsitors*.

En la figura 3.14 se presenta el esquemático del circuito que implementa una sola línea de la memoria ReRAM usando el esquemático alternativo, además se ha agregado el circuito con la topología convencional que se presentó previamente en este trabajo para facilitar la comparación de los elementos introducidos. Dentro de los principales cambios se tiene que la línea OL se conecta a la entrada inversora del amplificador operacional, de esta forma se construye el amplificador sumador cuya salida posteriormente se conecta a un comparador. De igual forma, a pesar de la existencia de una tierra virtual, la línea OL puede ser conectada a tierra al momento de realizar la escritura de un dispositivo. Por otro lado, ya no es necesario aplicar un voltaje V_{DD} en la línea OL, por lo que se reduce el área necesaria en este sentido, compensando así la inclusión de nuevos componentes.

Ahora con esta nueva topología se puede repetir el estudio realizado previamente para observar la mínima ventana de voltaje $|\Delta V_{OL}|$ para la computación de una compuerta NOR2. De la ecuación 3.4 se obtiene que para la combinación de entradas "00" la resistencia equivalente es $R_{eq,00} = \frac{R_{off}}{2}$ y en el nodo de salida se tiene $V_{OL\_1} = -V_{REF}(\frac{2R_F}{R_{OFF}})$. Vale la pena mencionar que en esta topología se tiene que $V_{OL\_1} < V_{OL\_0}$, esto comparado con lo que se planteaba en la topología original en donde ocurría lo contrario. De la ecuación 3.2 y la ecuación 3.4 se puede computar los valores correspondientes para V_{OL_0} a la salida de la compuerta lógica como:

$$V_{OL\_0} = \begin{cases} -V_{REF}\frac{R_F}{R_{off}}k, & R_{off} >> R_{on} \\ -V_{REF}\frac{R_F}{R_{off}}(k+1), & EOC \end{cases} \tag{3.5}$$

De la ecuación 3.5 se tiene que la salida de voltaje siempre dependerá de la razón de resistencias k. Al igual que la topología anterior un valor reducido de k significa un valor reducido de la ventana de voltaje $|\Delta V_{OL}|$. La ventana de voltaje más angosta está dada por $|\Delta V_{OL}| = |V_{OL\_0} - V_{OL\_1}| = |-V_{REF}(R_F/R_{off})(k-1)|$ la cual para $k >> 1$ se obtiene $|\Delta V_{OL}| = |V_{OL\_0}|$, lo cual es proporcional tanto a R_F como a k, por lo que se puede diseñar de manera apropiada. Por ejemplo, se puede asumir $R_F = 50\ k\Omega$, $R_{Off} = 10\ k\Omega$ y $V_{REF} = 100mV$, de esta forma con un $k = 2$ se obtiene $\Delta V_{OL} = 500mV$, lo que nos entrega la mínima ventana posible. No obstante, si es que se tiene un $k = 10$ la diferencia de voltaje aumenta a $\Delta V_{OL} = 4{,}5V$ lo que significa un aumento considerable de la ventana, obteniendo valores mayores que los observados en la topología original. Por otro lado, esta nueva topología entrega un grado de libertad más al momento de diseñar la ventana de voltaje, esto debido a que también depende de la relación entre la resistencia R_{off} y la resistencia R_F. Esta última resistencia se puede diseñar si es que se conoce R_{off} para obtener una ventana de voltaje suficiente y que permita una correcta operación.

Esta nueva topología permite también, si es que se varía el voltaje de comparación, el computo de otras compuertas lógicas sin la necesidad de modificar otras partes del circuito. Por ejemplo, se explicó

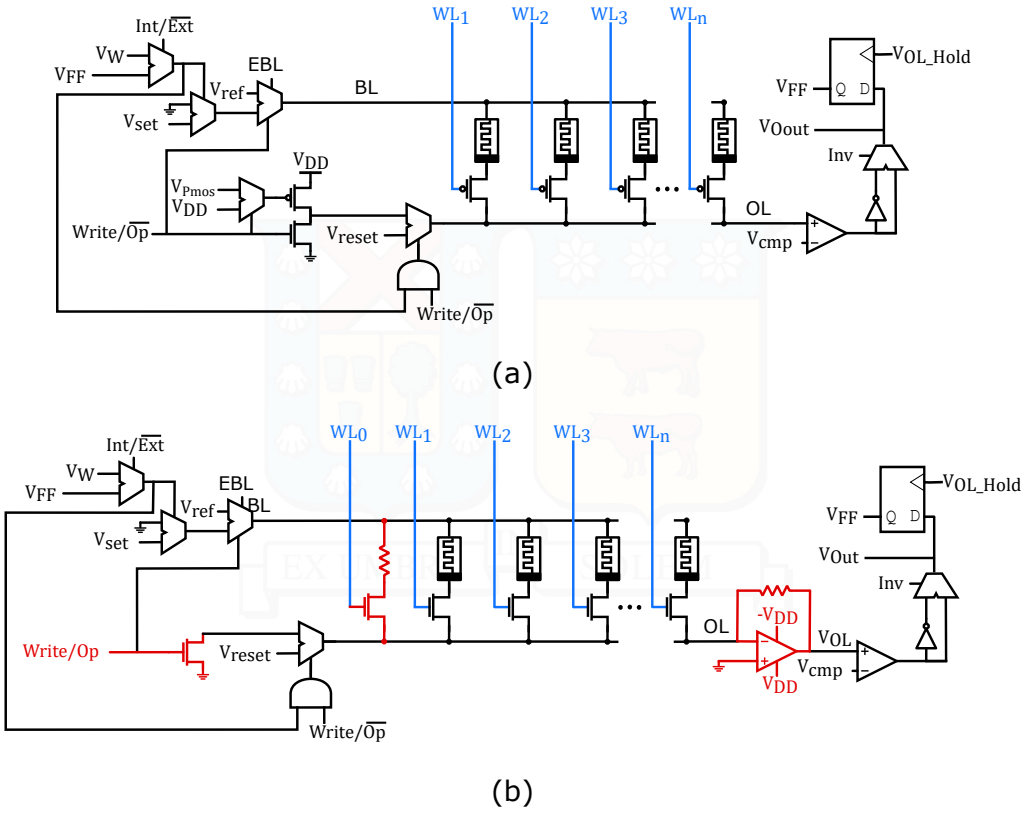


Figura 3.14: (a) Topología original. (b)Linea de memoria acorde a nueva topología del amplificador sumador

previamente que en el cómputo de una compuerta NOR $V_{OL\_1} < V_{OL\_0}$ y en particular cuando las entradas corresponden a '11' se tiene el mayor voltaje, de esta forma si es que se sube adecuadamente el umbral de voltaje se puede transformar el cálculo de la compuerta NOR en el cálculo de la compuerta AND. Esto sin duda entrega mayor versatilidad a esta topología de memoria, no obstante, esto escapa del objetivo de este trabajo.

3.4. Implementación practica topología alternativa

Actualmente se tienen dispositivos de *memristors* disponibles en el comercio, tal es el caso de la compañía *Knowm Inc.*[17] que ha acercado esta tecnología a investigadores y estudiantes sin la capacidades de manufactura. Para este trabajo en particular se tienen a disposición empaquetados DIP cerámicos de 16 pines con dispositivos *memristors* BS-AF-W. Estos dispositivos pertenecen a la categoría de canales autodirigidos (self directed channel - SDC) y se componen de pilas de materiales que conducen iones. Para este caso la estructura interna de estos dispositivos se muestra en la figura 3.15. Estos están compuestos por dos electrodos de tungsteno de los cuales uno de ellos se encuentra en contacto con una capa de metal Ag (plata), al momento de aumentar el potencial del electrodo en contacto con esta capa metálica ocurre la oxidación de la misma y la producción de iones. Una vez formados estos últimos comienzan a desplazarse hacia el electrodo de menor potencial, estos iones al llegar a la capa activa se aglomeran produciendo canales de conducción, aumentando de esta forma la conductividad del dispositivo. Por otro lado, si el mayor potencial se aplica al electrodo más cercano a la capa activa, los iones contenidos en esta emigran en la dirección contraria y disminuyendo así la conductividad del dispositivo.

Para manejar estos dispositivos se deben tener una serie de consideraciones fundamentales para no dañarlos, una de estas corresponde al nivel de corriente con el que se trabaja. De acuerdo al datasheet de los

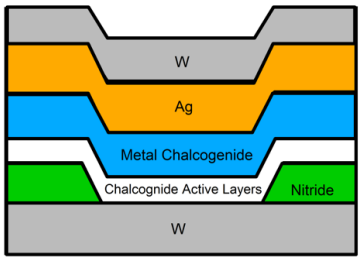


Figura 3.15: Estructura de los *memristors* SDC

dispositivos con los que se dispone, el máximo de corriente que soportan es de 1 mA, por esto un método de limitar la corriente al momento de medir la variación de un *memristor* se hace indispensable. La forma más simple de limitar la corriente sobre estos dispositivos es usando el método de la resistencia en serie. En la figura 3.16-a se observa el *memristor* en serie con una resistencia R_s, esto permite que al aplicar un voltaje V_{In} se produzca un divisor de tensión entre la resistencia conocida y el *memristor*. De esta forma basta con medir el voltaje V_{Out} para saber la corriente que pasa a través del *memristor* y así tener una medición de la resistencia (usualmente llamada "memristencia") del dispositivo.

No obstante, si es que se asume que el resistor esta contactado de forma permanente a los *memristors* para monitorear de manera constante el estado de estos dispositivos durante la aplicación de una señal de entrada para provocar un SET o un RESET, la selección de un valor de resistencia R_S apropiado que permita medir el valor de memristancia puede no ser una tarea fácil. Incluso debido al efecto del divisor de tensión, el voltaje sobre el *memristor* cambia en la medida en que se produce la conmutación de estado del dispositivo. Por esto se vuelve difícil diferenciar aquellos efectos que ocurren debido a las propiedades intrínsecas del dispositivo de aquellos que ocurren debido a la redistribución del voltaje en el divisor de tensión durante el proceso de conmutación.

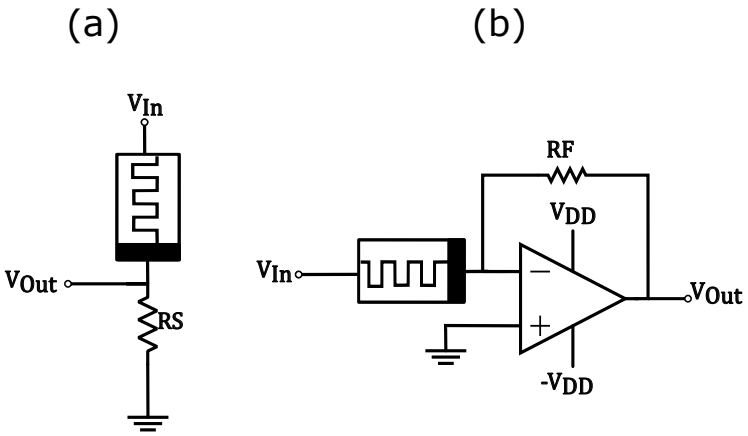


Figura 3.16: (a) Circuito de divisor de tensión (b) circuito de amplificador operacional con resistencia de retroalimentación

De esta forma en [43] se plantea el uso de la topología que hace uso del amplificador operacional con retroalimentación negativa. Este circuito se presenta en la figura 3.16 en donde se conecta el *memristor* al terminal negativo del amplificador operacional, el cual a su vez tiene una resistencia de realimentación R_F. Esta configuración tiene ciertos elementos positivos, en primer lugar, se obtiene un voltaje constante sobre los dispositivos debido a la existencia de la tierra virtual. En segundo lugar, se logra medir la corriente que pasa a través del dispositivo *memristor*, esto debido a que es proporcional a la señal de voltaje V_{Out}. Por último, esta topología permite limitar la corriente máxima que fluye a través del dispositivo que se está midiendo. Esto último se logra a través de la configuración de los voltajes de alimentación $|V_{DD}|$ y resistencia R_F, de esta

forma se define como la corriente máxima que pasara a través del *memristor* es igual a:

$$V_{DD} = \pm R_F I_{MAX} \tag{3.6}$$

Cabe señalar que, mientras el amplificador esté en saturación, no se pueden considerar los datos obtenidos de la medición para calcular la resistencia del dispositivo. De igual forma se puede definir la corriente I_m que pasa a través del *memristor* en estado resistivo R_m como una relación proporcional entre la resistencia de retroalimentación y el voltaje de entrada, esta se define como:

$$I_m = \frac{-R_f * V_{in}}{R_m} \tag{3.7}$$

Con la topología de mediciones seleccionada se procede a armar en una protoboard el circuito de medición correspondiente al amplificador operacional con resistencia de retroalimentación. Para este caso se ha elegido como componente el amplificador operacional OP97, el cual es de baja potencia y alta precisión. El dispositivo seleccionado posee un voltaje de offset de $25\mu V$ y funciona con un rango de voltaje que va desde los ±2.25 V a los $\pm 20V$, para este caso el voltaje de alimentación que se utiliza es de 5 V. Por otro lado, se hace notar que este circuito de medición es idéntico al de amplificador sumador, pero con un solo dispositivo conectada a la entrada inversora. De esta forma la resistencia de realimentación se diseña considerando la operación posterior como sumador. En este sentido es necesario considerar los umbrales de voltaje de los dispositivos los cuales se presentan en la tabla 3.2.

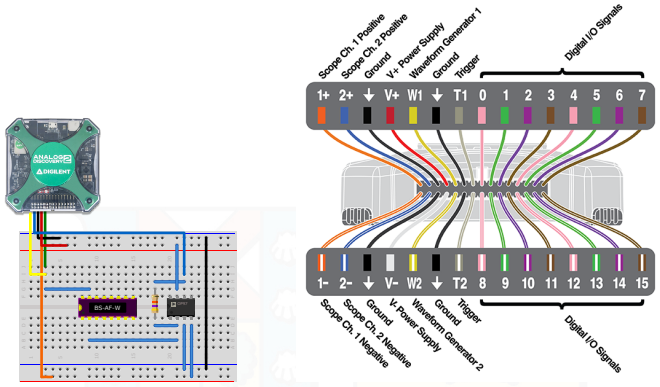
Characteristic	Condition	Min	Typ	Max
Forward Threshold	DC/Quasi-Static	0.15V	0.26V	0.35V
Reverse Threshold	D/Quasi-Static	-0.27V	-0.11V	-0.05V
Cycle Endurance	1.5Vpp, 500 Hz sine wave, 50kΩ series resistor	50M	100M	5B

Tabla 3.2: Umbrales de voltaje de los dispositivos *memristors* utilizados, de acuerdo a su hoja de datos.

De esta forma se puede elegir un voltaje de lectura que no altere el estado interno de los dispositivos, al no superar en valor absoluto el menor de los valores de umbral mencionados anteriormente. En este caso el más alto de los mínimos voltaje que no produce cambios es el que corresponde al umbral positivo de 0.15 V, de manera que se considera como voltaje de lectura 0.1 V. En este sentido como la variable a medir sera un voltaje inversamente proporcional a la resistencia del dispositivo, esto acorde a la ecuación 3.7, se necesita un valor de resistencia R_F que logra amplificar este voltaje para lograr valores distinguibles entre los estados de baja resistencia y alta resistencia, esto sin saturar el amplificador operacional. Naturalmente el máximo voltaje que se obtenga en la lectura del estado de un dispositivo estará dado por la baja resistencia, la cual acorde a los datos proporcionados por el datasheet, se encuentra en el orden de magnitud de los 10e4. De esta forma se elige una resistencia de 47 $k\Omega$, con lo que considerando una resistencia baja de 10 $k\Omega$ del dispositivo y un voltaje de lectura de 0.1 V se obtendría un voltaje a la salida del amplificador operacional de 0.94 V. Se espera que el valor de alta resistencia sea de por lo menos el doble a la resistencia. En consecuencia, se tendría un voltaje a la salida de 0.47, voltajes diferenciales entre ambos.

Con estas consideraciones se planifica una serie de pruebas para corroborar el funcionamiento de la topología del amplificador sumador, y la construcción de un circuito que permita realizar un ejemplo funcional de las operaciones que se pueden realizar en una memoria de una fila y dos celdas. En primer lugar, se busca corroborar las características de los *memristors* y seleccionar aquellos que rinden según lo esperado y que permitan realizar las pruebas adecuadas.

Se construye el circuito en una protoboard distribuyendo los componentes mencionados según indica la figura 3.17-a. Para estas implementaciones practicas se utiliza el Analog Discovey 2 de Digilent, el cual es un osciloscopio USB, analizador lógico y fuente de poder, que se utilizara para alimentar los componentes, generar señales de entrada y medir salidas. La especificación de los pines de este dispositivo se muestran en la figura 3.17-b



(a) (b)

Figura 3.17: (a) Configuración circuito para medición de *memristor* en protoboard. (b) Acercamiento a los pines del Analog Disovery 2

3.4.1. Primer trabajo practico con *Memristors*

Cabe señalar que, los *memristors* originalmente se encuentran en un estado puro de alta resistencia y necesitan pasar por un proceso que permite la creación de un canal conductivo en el dispositivo, este es conocido como el proceso de "forming". Este proceso se puede realizar de dos formas, mediante una rampa de voltaje ascendente o utilizando una señal AC de amplitud incremental. Producto de la variabilidad de los dispositivos no existe un proceso de forming común para cada uno de ellos, por lo que se deben ajustar los parámetros de cada método cada vez que se necesita realizar este proceso en distintos dispositivos. Para este caso se utiliza el mecanismo que considera una rampa de voltaje, ya que este al ser un enfoque más conservador al tener más control sobre los parámetros, ayuda a minimizar la probabilidad de dañar los dispositivos. La rampa se aplicará hasta un voltaje en el que se observe un aumento suficiente en la corriente (en relación al límite de corriente) que atraviesa el dispositivo, será en este punto en el que se considerará el *memristor* como formado.

Cabe destacar que este proceso es crítico para el funcionamiento del dispositivo, pero se tiene la incógnita de cuanto voltaje es necesario aplicar y cuanto es el tiempo de duración de esta rampa, de esta forma para este proceso realizado en los dispositivos disponibles, se comenzara con pruebas valores conservadores de voltaje máximo y tiempo de duración de la señal de entrada, usando primero una rampa de 0 a 1 V en 100 μs. De no lograrse el objetivo se incrementará el tiempo de duración de la rampa en valores de 100 μs hasta los 1 ms, desde donde se comenzará a incrementar el tiempo en valores a un ritmo de 1 ms por prueba. En caso de llegar a los 10 ms y nuevamente no lograr el objetivo del forming, se incrementará el voltaje máximo alcanzado a valores entre los 1 y 1.5 V como máximo. Si es que se ha alcanzado el valor máximo de 1.5 V en la prueba de forming y aun no se tienen resultados, se vuelve a cambiar la variable de tiempo, esta vez aumentando el valor a un ritmo de 10 ms por prueba. En caso de que con este último procedimiento no se logra el forming, se considera que el dispositivo se encuentra defectuoso.

Se muestra este proceso de forming realizado sobre dos dispositivos distintos cuyas curvas de voltaje de entrada y corriente medida se muestra en la figura 3.18. Además, se destaca con una línea vertical gris horizontal y vertical el voltaje en el que comienza el forming y el tiempo en que este ocurre respectivamente. En primer lugar se muestra en la figura 3.18-a un dispositivo que necesitó de una rampa de 0 a 1 V con un tiempo de ascenso de 2 ms para lograr formarse, alcanzando un máximo de corriente de $\approx$40 μA cuando la rampa llega a 1 V. En este caso el tiempo en el que comienza este proceso es en los 0.8 ms y un voltaje de entrada de 0.43 V. Por otro lado en la figura 3.18-b se muestra el proceso de forming de otro dispositivo. En este caso se necesitó de una rampa de voltaje entre los 0 y 1 V en un tiempo 1 ms para completar la formación del dispositivo. En este caso se observa el máximo voltaje alcanzado corresponde a $\approx$92 μA, el cual también corresponde al momento en el que llega a la saturación. Para este segundo caso el tiempo en el que comienza el forming se ubica en los $\approx$0.4 ms y con un voltaje de entrada de $\approx$0.42 V. Como se presentó previamente

ambos voltajes que permiten el forming son cercanos entre si cambiando tan solo la pendiente de la rampa. En el primer caso se necesita una rampa con una menor pendiente en comparación al dispositivo del segundo caso, lo que sugiere un primer dispositivo con una taza de cambio más baja en contraste con el segundo. Se tienen también los casos de la figura 3.18-c y figura 3.18-d para los cuales se utiliza una rampa de 10 ms y 70 ms respectivamente de 0 a 1.5 V. En estos casos el forming necesito de voltajes más altos que en los casos anteriores, llegando a los ≈1 V en el primer caso y ≈1.3 V en el segundo. Es este tipo de variabilidad entre dispositivos con los que se debe lidiar al momento de trabajar con ellos para aplicaciones de computación en la memoria.

En el proceso de forming se puede comenzar a vislumbrar las características que tendrán estos dispositivos, tal como puede ser taza de cambio de la conductividad ante la señal de entrada, valores de resistencia entre otros. En particular para la aplicación que se está apuntando se busca dispositivos con características similares en la respuesta ante una misma señal aplicada, teniendo preferencia por aquellos dispositivos con la taza de cambio más alta o aquellos con umbrales de voltaje bajos, al permitir una mayor libertad en la selección de la duración de las señales de entrada y los valores de voltaje.

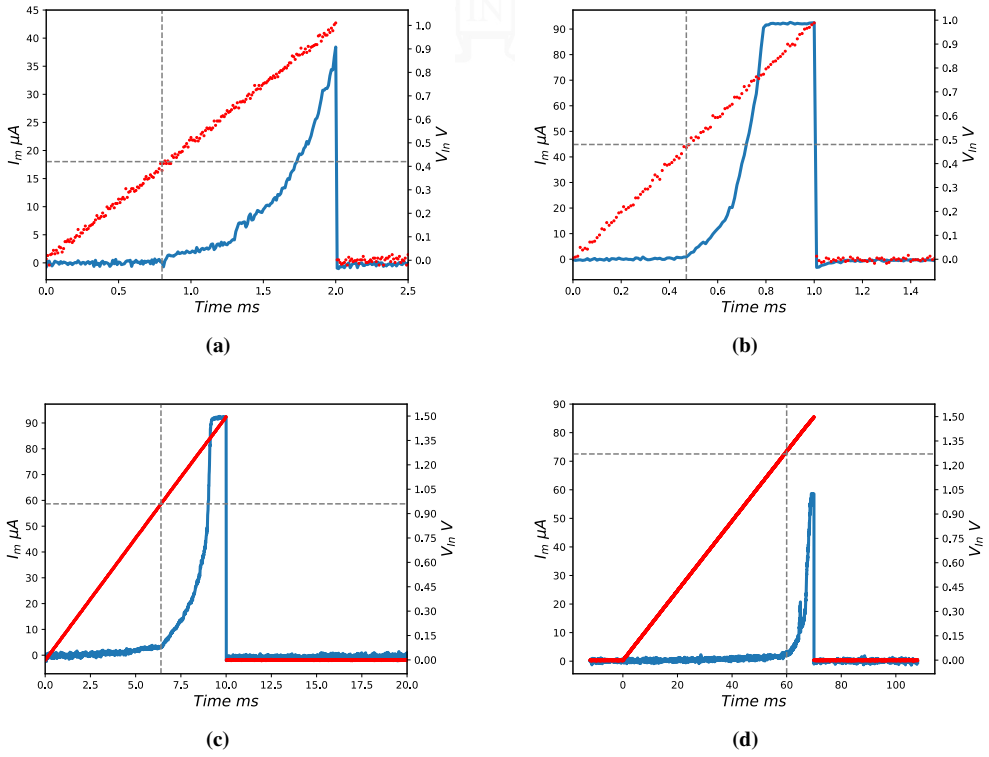


Figura 3.18: Ejemplos de procesos de forming en un *memristor* utilizando rampa de voltaje de 0 a 1 V mostrada en rojo. En azul se muestra la corriente que pasa a través del dispositivo. (a) Proceso realizado en un dispositivo que necesito de rampa de 2 ms (b) Proceso realizado en un dispositivo que necesito de rampa de 1 ms. (c) Proceso realizado en un dispositivo que necesito de rampa de 10 ms. (d) Proceso realizado en un dispositivo que necesito de rampa de 70 ms

3.4.2. Curvas características de los dispositivos

Una vez se haya completado el proceso de forming, la primera prueba que se puede realizar con los dispositivos formados es obtener la curva de histéresis característica que nos permite tener más información acerca del comportamiento de los dispositivos. En este caso se utilizaron los dispositivos del ejemplo (a), (b) y (c) del proceso de forming que se muestra en las figuras 3.18-a,b y c respectivamente. Para estas pruebas las señales de entradas corresponden a 50 ciclos de formas de ondas triangulares simétricas de 1 V de amplitud y

10 ms de periodo. Las tres curvas de histéresis se pueden observar en la figura 3.19, en estos gráficos se ha marcado con flechas la dirección del proceso de SET y posterior procesos de RESET, además se destaca el umbral de voltaje aproximado, tanto positivo como negativo, con una línea vertical punteada. Las primeras dos curvas, como se mencionó previamente, corresponden a los procesos de forming (a) y (b) del punto anterior, estos correspondían a los procesos más rápidos de los cuatros mostrados. Lo anterior se refleja en la rápida conmutación de resistencia de las curvas correspondientes a la figura 3.19-a y 3.19-b los cuales ambos alcanzan la saturación, esto en contraste con el dispositivo que se muestra en la figura 3.19-c con una pendiente de SET menor que en los casos anteriores. Por otro lado, los ejemplos (a) y (b) también tienen características disimiles, ya que el primer caso se puede considerar más lento en el proceso de SET al tener una pendiente menor que el caso (b). No obstante, el caso (a) posee un rango de resistencia mayor observando que la diferencia de la mínima corriente alcanzada en los -1 V y el máximo de corriente alcanzada en los 1 V es de $60\ \mu A$ en contraste con el ejemplo (b) que es de tan solo $30\ \mu A$.

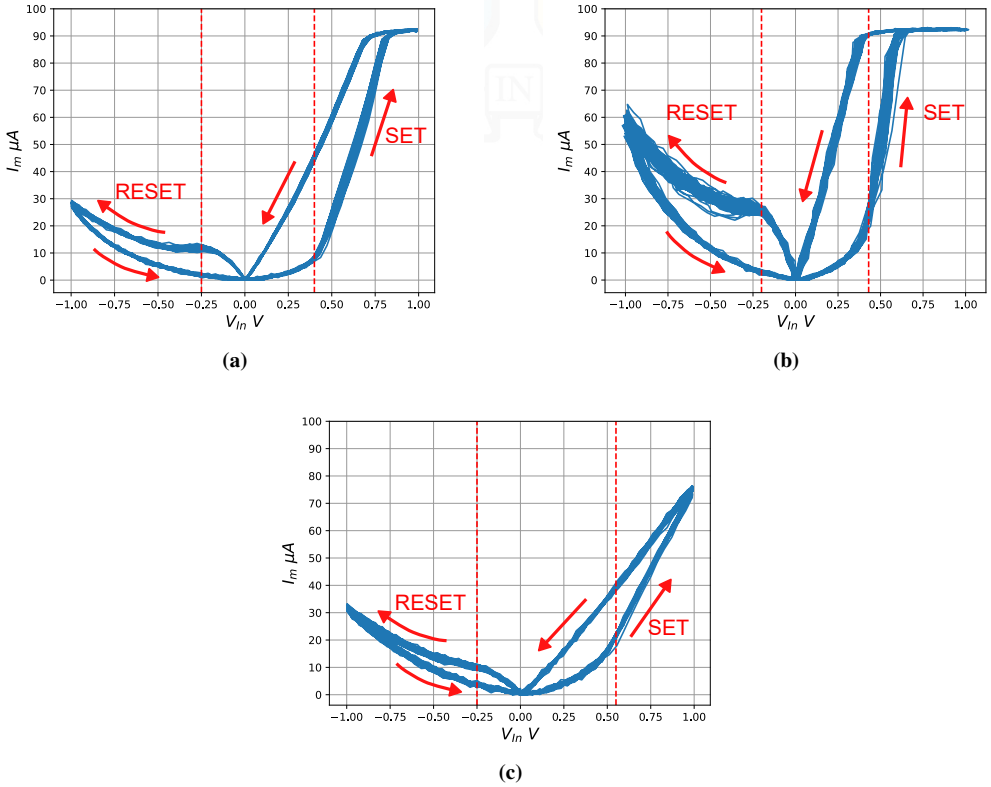
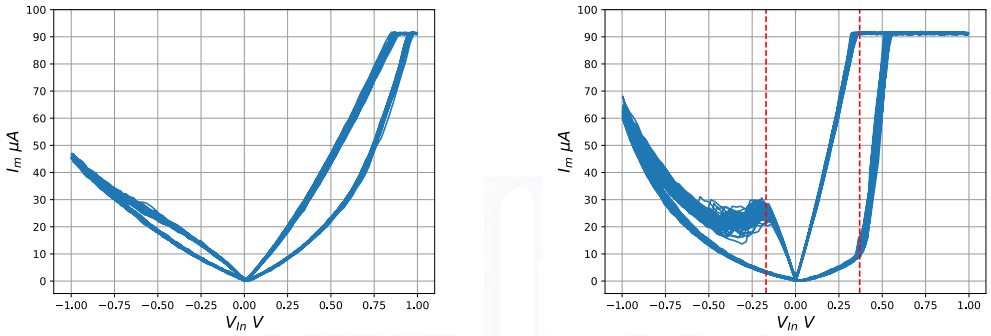


Figura 3.19: Ejemplos de curvas características de histéresis de *memristors* formados. Voltaje de entrada corresponde a una señal triangular de 10 ms de periodo y 1 V de amplitud, en donde se consideran 50 ciclos y se muestra el valor absoluto de corriente. (a) Curva asociada al dispositivo a del proceso de forming. (b) Curva asociada al dispositivo b del proceso de forming. (c) Curva asociada a ejemplo c del proceso de forming.

Estas características de tiempo de conmutación y diferencia de rangos de resistencia son elementos claves al momento de seleccionar los dispositivos adecuados para realizar los procesos de computación en la memoria. En un caso ideal se buscan dispositivos semejantes en estos aspectos, pero como se muestra ambos dispositivos son diferentes a pesar de pertenecer al mismo integrado, y es más, si se observa el caso del ejemplo de la figura 3.19-b se puede observar incluso variabilidad en cada ciclo de operación, por lo que se debe ajustar la operación a la características de los dispositivos disponibles.

Adicionalmente se consideran dos dispositivos cuyo proceso de formación no fueron presentados en las figuras anteriores. La curva característica de estos nuevos dispositivos se presenta en la figura 3.20 las cuales se obtuvieron utilizando 50 ciclos de una señal triangular de 1 V de amplitud y 10 ms de periodo. Se tiene la curva del primero de estos dispositivos en la figura 3.20-a en la cual se observa que si bien logra



(a) **(b)**

Figura 3.20: Curva de histeresis de dispositivos con forming previo. Voltaje de entrada corresponde a una señal triangular de 10 ms de periodo y 1 V de amplitud en donde se consideran 50 ciclos y se muestra el valor absoluto de corriente. (a) Dispositivo 1. (b) Dispositivo 2.

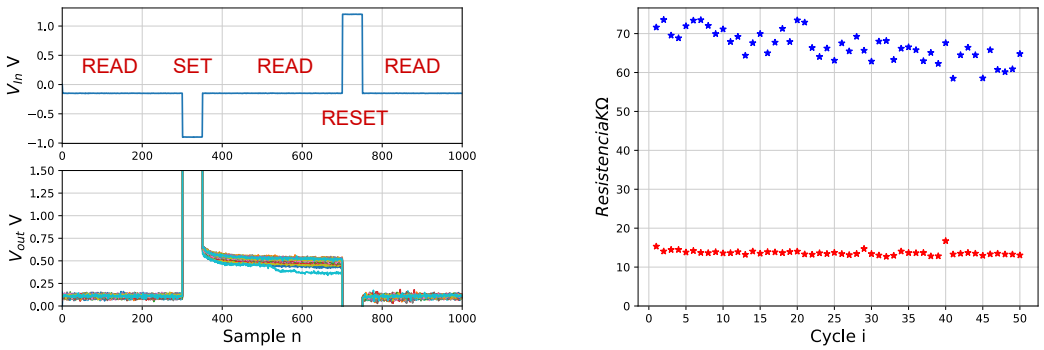
llegar a la saturación, es un dispositivo que no se identifica un umbral fuerte, produciéndose su conmutación lentamente(en comparación a los dispositivos que también saturan mostrados previamente). Lo anterior contrasta con el dispositivo de la figura 3.20-b en donde el dispositivo tiene un umbral claro y logra un proceso de conmutación más rápidamente que el caso anterior. No obstante a pesar de esta diferencia ambos poseen una razón de resistencia similar, esto se observa a partir de la diferencia de corriente $\Delta I_{m,1}$ en los extremos del rango conductivo del dispositivo (diferencia de corriente cuando el dispositivo está en alta y baja resistencia). Considerando esto, se observa que el dispositivo de la figura 3.20-a tiene una diferencia de corriente igual a $\Delta I_{m,1} \approx 40\,\mu A$, mientras que el de la figura 3.20-b tiene $\Delta I_{m,1} \approx 30\,\mu A$. El hecho de ser el par de dispositivos con la razón de resistencia más cercana hace que sean candidatos para el ejemplo práctico de computación en la memoria.

3.4.3. Niveles de resistencia y selección

Las pruebas anteriores dejan claro que se cuenta con 4 dispositivos disponibles para continuar con las pruebas. Se identifica como M1 y M2 a los dispositivos con las curvas características de la figura 3.19-a y figura 3.19-b respectivamente. Por otro lado se identifica como M3 y M4 a los dispositivos con las curvas de la figura 3.20-b y figura 3.20-a respectivamente. El objetivo de esta sección es realizar experimentos que nos entreguen los niveles de resistencia de los dispositivos.

De esta forma con los dispositivos seleccionados se pasa a realizar pruebas con operaciones similares a las que se esperarían realizar en una memoria computacional. Por lo anterior es que se utiliza una señal periódica que introduzca las dos operaciones básicas de una memoria, que son la lectura y escritura. En este caso esta señal es de 10 ms de periodo, en donde los primero 3 ms corresponden a un proceso de lectura, esto seguido de 500μs de un proceso de SET, posteriormente 3.5 ms de lectura nuevamente, seguidos de 500 μs de RESET, para concluir con 250 μs de lectura. Con esto se logra tener una lectura del estado inicial del dispositivo, además de la lectura de su estado final cada vez que se escriba un '1'(SET) lógico y un '0' logico (RESET). De igual manera a partir de la conversión del voltaje de salida del amplificador operacional se puede obtener el nivel de resistencia del dispositivo. Para estas pruebas se ha seleccionado como voltaje de SET -0.9 V, voltaje de RESET 1.2 V y voltaje de lectura -0.15 V. Notar que en este caso el voltaje de SET es negativo, lo cual se debe a la forma en el que fue conectado el integrado correspondiente al *memristor*. El dispositivo quedo con polaridad invertida al estar el terminal inferior conectado a la entrada y el superior a la tierra virtual del amplificador operacional, invirtiendo de esta forma la operación del dispositivo.

El primer elemento que se prueba es el correspondiente al M1, en donde se aplican 50 ciclos de el voltaje de entrada V_{In} y se obtienen la salida V_{Out} del amplificador. De esta forma se se muestran los 50 ciclos de estas señales en la figura 3.21-a.

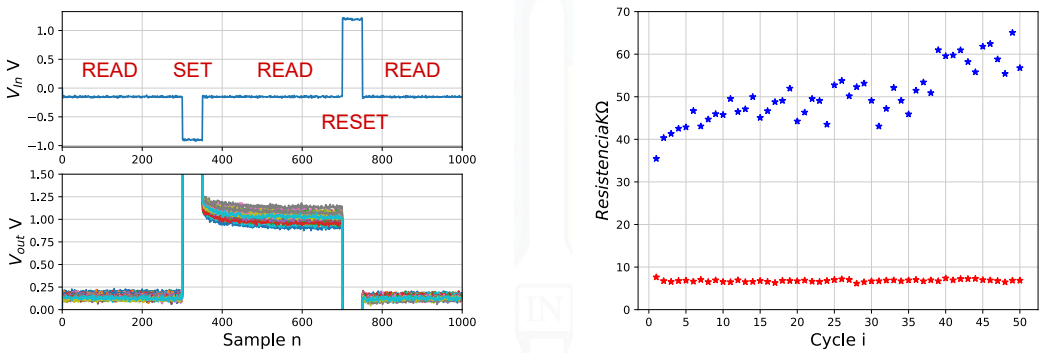


(a) **(b)**

Figura 3.21: (a)Pruebas de SET y RESET del dispositivo M1. (b) Niveles de resistencia por ciclo.

En el gráfico de V_{In} se detallan las operaciones que ocurren en cada sección antes descrita. En el gráfico de V_{Out} se muestra el voltaje de salida del amplificador operacional que se comporta de acuerdo a la ecuación 3.4. De esta forma se observa que el proceso de SET se realiza de forma correcta al observar un aumento considerable del voltaje de salida al momento de leer. Por otro lado, el proceso de RESET vuelve a bajar el voltaje de salida producto de un aumento de la resistencia. Inicialmente el dispositivo siempre comienza en baja resistencia producto de que la última operación que se realiza es la de RESET. En este caso se destaca que la diferencia de voltaje es de ≈0.4 V entre el momento en que se encuentra en alta y baja resistencia el dispositivo. Por otro lado se calcula también la resistencia R_{OFF} y R_{ON} promedio en cada uno de los ciclos, esta evolución se muestra en la figura 3.22-b. Como se puede observar se tiene que $R_{ON} \approx 15$ $k\Omega$ con una baja variabilidad, esto contrasta con la situación de la resistencia R_{off} que posee una mayor variabilidad y no se puede ubicar en un valor estable, lo cual puede deberse a dos fenómenos. En primer lugar, este puede deberse al error de medición producto de que la alta resistencia entrega voltajes bajos. En segundo lugar, esto puede deberse a que el voltaje de SET resulta lo suficientemente alto como para dificultar

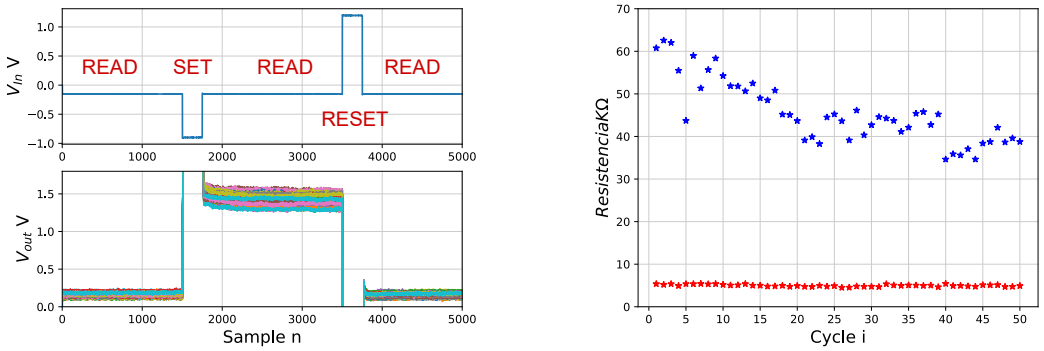
el proceso de RESET, esto a pesar de tener un voltaje de SET menor al de RESET, de esta forma se explica también la disminución en el promedio de R_{OFF} a medida que se suman ciclos a la prueba. Esto último es especialmente relevante, pues hace hincapié en la importancia que tiene la selección de las características de los pulsos, dado que forzar innecesariamente el dispositivo durante SET o RESET puede conducir a fallos permanentes.



(a) **(b)**

Figura 3.22: (a)Pruebas de SET y RESET del dispositivo M2. (b) Niveles de resistencia por ciclo.

Se procede a realizar la misma prueba con el dispositivo M2, en donde las señales de entrada V_{In} y salida V_{Out} se muestran en la figura 3.22-a. En este gráfico la principal diferencia se observa en el voltaje de salida posterior a un proceso de SET en el cual se alcanza un nivel superior a lo mostrado por el dispositivo anterior, lo que indica una resistencia R_{on} menor. Lo anterior se combina con un voltaje de lectura posterior al proceso de RESET semejante al dispositivo M1, lo que entrega una ventana de voltaje de ≈0.9 V mayor que en el caso anterior. Este resultado se corresponde con el cálculo de la resistencia promedio en cada uno de los ciclos de prueba, gráfico que se presenta en la figura 3.22-b. En este caso lo primero que se observa es una disminución de la resistencia R_{On} a un valor ≈ 6,5 $k\Omega$, lo que significa una disminución de casi la mitad de la baja resistencia en comparación al dispositivo M1. Naturalmente esto explica el aumento en casi el doble del voltaje de salida posterior al proceso de SET del dispositivo. En este caso también se observa la variabilidad del dispositivo haciéndose más evidente en los valores de resistencia R_{OFF}. En este caso la resistencia R_{OFF} tiende a crecer en su valor lo que sugiere que en este caso el voltaje de RESET puede tener un valor innecesariamente grande y un valor menor sería suficiente para lograr la conmutación del dispositivo.

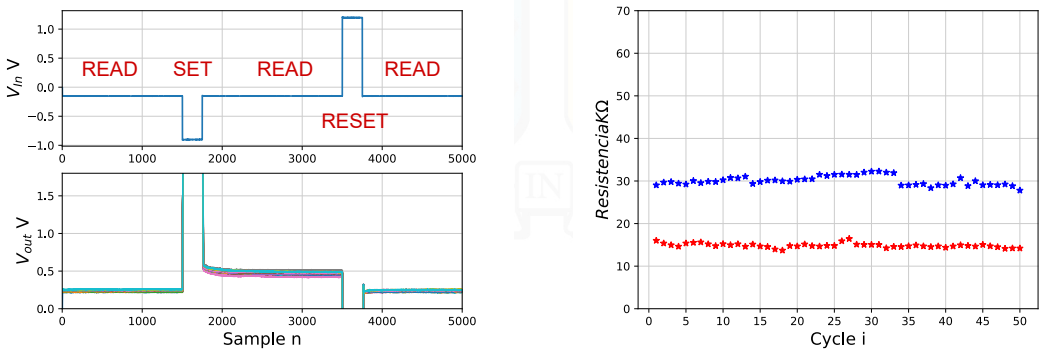


(a) **(b)**

Figura 3.23: (a)Pruebas de SET y RESET del dispositivo M3. (b) Niveles de resistencia por ciclo.

En el caso del tercer dispositivo M3, los resultados posterior a los 50 ciclos de la señal de entrada

usada se pueden observar en la figura 3.23-a. Con este elemento se tiene un comportamiento similar al obtenido con el dispositivo M2. Se puede observar en el voltaje de salida un aumento en su magnitud en la lectura posterior a un proceso de SET, manteniéndose por el contrario el voltaje posterior a un proceso de RESET. En este caso la diferencia de voltaje entre ambos es de $\approx$ 1,35 V. Lo anterior es respaldado por el gráfico de la evolución de las resistencias promedio que se muestra en la figura 3.23-b. En este se observa que la resistencia R_{ON} promedio disminuyo a un valor $\approx$ 5 $k\Omega$, lo que es menor al valor obtenido en el caso del dispositivo M2. La variabilidad de la resistencia R_{OFF} nuevamente se observa en esta figura, esta vez con una tendencia a disminuir su promedio hacia el final de los ciclos de forma más rápida que en el caso de M1.



(a) **(b)**

Figura 3.24: (a)Pruebas de SET y RESET del dispositivo M4. (b) Niveles de resistencia por ciclo.

Por último se tiene el caso del dispositivo M4 cuyas pruebas se presentan en la figura 3.24. En este caso se observa una ventana de voltaje de $\approx$ 0,24 V lo que resulta menor en comparación a los casos anteriores. La razón detrás de esta disminución se puede concluir a partir de la evolución de la resistencia de los dispositivos en cada ciclo la cual se muestra en la figura 3.24-b. En este gráfico se puede observar un aumento en el valor promedio de la baja resistencia, en comparación a los casos anteriores, alcanzando un valor de $R_{ON} \approx 15\ k\Omega$. A su vez se tiene una disminución de la alta resistencia promedio la que alcanzo un valor de $R_{OFF} \approx 30\ k\Omega$. En este caso se tiene un proceso que presenta menos variabilidad en la medición de las altas resistencias que en el caso anterior.

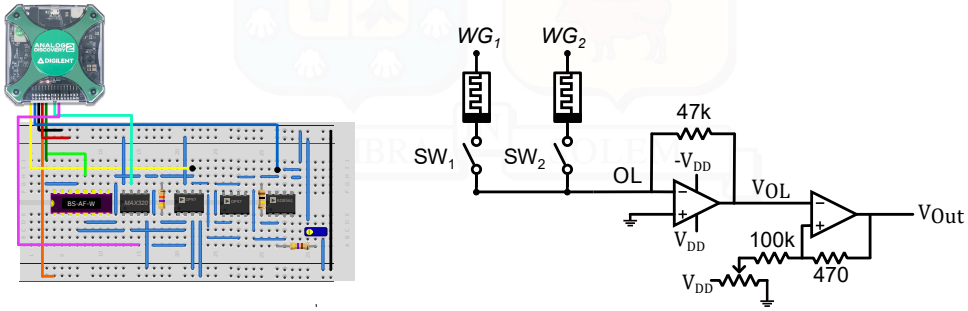
Las pruebas anteriores confirman nuevamente la variabilidad no tan solo entre los dispositivos si no también la variabilidad intrínseca de los mismos que sucede ciclo a ciclo. Por otro lado, también se observó el funcionamiento de la topología del amplificador sumador con un solo dispositivo, recordando por supuesto que esta topología coincide con el circuito de medición seleccionado. En este sentido se puede observar el efecto de los distintos niveles de resistencia R_{ON} y R_{OFF} en los voltajes de salida. Se puede reconocer en el dispositivo M3 la mayor ventana de voltaje entre baja y alta resistencia que difiere considerablemente de la ventana del dispositivo M1, considerando que este último tiene una mayor razón entre sus niveles de resistencia. Esto permite vislumbrar el efecto que tiene la selección de la resistencia de retroalimentación R_F en la topología de memoria. De acuerdo a esto se puede observar que la variabilidad de R_{OFF} cuando estas quedan en un rango de resistencia $R_{OFF} > R_F$ poco efecto tienen en el nivel de voltaje del dispositivo, si es que además se considera un voltaje de lectura bajo como el que se está utilizando. Esto es precisamente lo que se observa con los dispositivos M1 y M3, que a pesar de tener niveles de resistencia R_{OFF} disimiles tienen voltajes de alta resistencia promedios muy similares $\approx$ 0,1 V y $\approx$ 0,15 V respectivamente. En el caso del dispositivo M4 se deja de cumplir esta premisa y se tiene $R_{OFF} < R_F$, lo que resulto en un aumento del voltaje de baja resistencia a 0.23V. Contrario es lo que ocurre con la relación entre la baja resistencia y la resistencia de retroalimentación, en este caso se busca que $R_{ON} < R_F$, de esta forma mientras mayor sea el nivel de baja resistencia mayor voltaje se tendrá a la salida tal como se observó en el caso entre M2 y M3.

Otro factor a tener en cuenta son los niveles de voltaje utilizados para los pulsos de SET y RESET. Como se verá más adelante si es que no se tienen cuidado con la selección de estos valores, se puede lograr

que el dispositivo deje de funcionar correctamente, quedando atrapado en baja o alta resistencia. No obstante, la naturaleza asimétrica de los umbrales de voltaje de este dispositivo y el comportamiento de SET y RESET puede volver confusa la selección de estos parámetros. Como se observó en estos experimentos el proceso de SET y RESET pueden lograr desviaciones en la resistencia de operación, lo que en este caso se destacó en la resistencia R_{OFF} como se ha mencionado previamente.

3.4.4. Pruebas de computación lógica y de memoria

Posterior a la prueba de los dispositivos y conociendo los niveles de voltaje con los que se trabajara de manera individual con cada uno de estos, se avanza con la construcción de la periferia que permita la computación de compuertas lógicas.



(a) (b)

Figura 3.25: (a) Configuración circuito para pruebas de computación lógica. (b) Esquemático del circuito que implementa las pruebas de computación

Respecto a esto se agrega un buffer en la salida del amplificador sumador y posterior a este un comparador con histeresis tal como se muestra en el esquema de protoboard de la figura 3.25. Además de esto ahora los *memristors* se conectan al amplificador sumador mediante un switch que permite desconectar la linea de cada uno de los dispositivos. A pesar de la conexión anterior de igual forma se utiliza un generador de señales para cada uno de los dispositivos para crear las señales de entradas a los mismos, el switch se agrega pues en una instancia final el nodo de entrada sera común para todos los dispositivos, de forma que este componente sera el que permita individualizar la operacion de estos. Un resumen de los componentes utilizados, sus modelos y la función que cumplen se muestra en al tabla 3.3.

Componentes	Modelo	Objetivo
Amplificador Operacional	OP97	Configuración sumador inversor
Amplificador Operacional	OP97	Buffer
Comparador	AD8561	Comparación de voltajes de salida del sumador para computación lógica
Switch	MAX320	Seleccion de *memristors* en operación.

Tabla 3.3: Tabla de componentes utilizados

Con esto en consideración se utiliza el AD2 para generar las señales de entrada $V_{In,1}$ y $V_{In,2}$ que se muestran en la figura 3.26. Con estas entradas se busca lograr realizar las operaciones de computo de compuertas NOR y escritura de un lógico '0' y lógico '1' utilizando proceso de SET y RESET respectivamente, estas operaciones realizadas se detallan tambien en la figura 3.26. El periodo de las señales de entrada que se utiliza es de 10 ms, mientras que los pulsos de SET y RESET tienen una duración de 500 μs, manteniendo los valores de voltajes utilizados previamente.

El conjunto de operaciones realizadas comienza con el computo de la compuerta NOR entre ambos dispositivos, para posteriormente realizar un proceso de SET en ambos elementos provocando así que estos conmuten a baja resistencia, lo que es equivalente a la escritura de '1' lógico en cada celda de memoria. Con

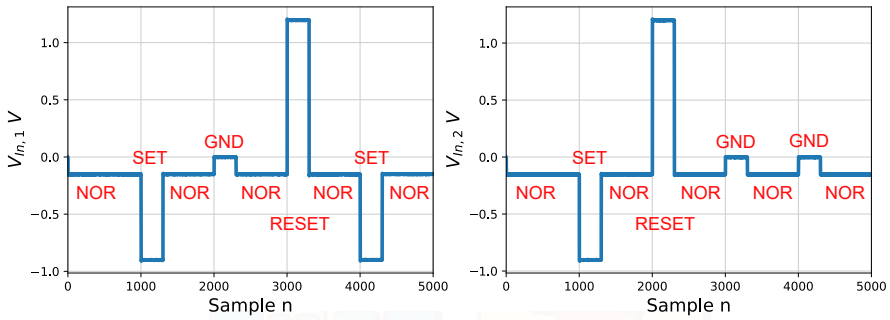
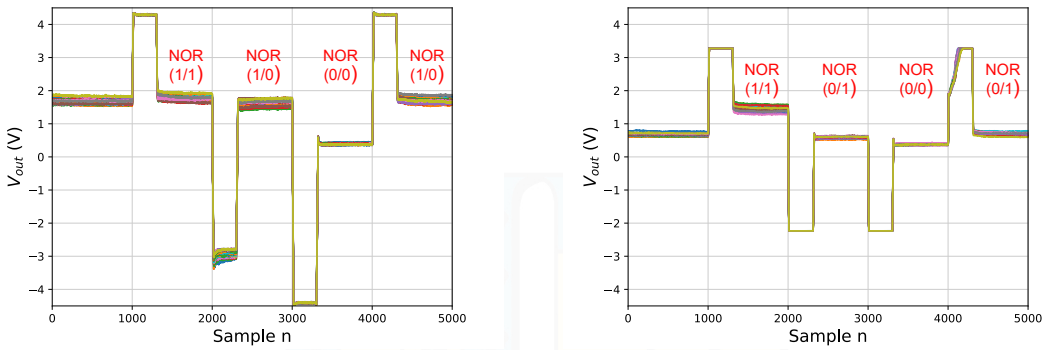


Figura 3.26: Entradas a los dispositivos utilizados en las pruebas de computación $V_{In,1}$ y $V_{In,2}$.

los dispositivos inicializados en un estado conocido de 1/1 se procede a computar nuevamente la compuerta NOR. Lo anterior es seguido de un proceso de RESET en el dispositivo conectado a la entrada $V_{In,2}$ de forma que ahora se tiene como estado de las celdas 1/0. Con los elementos en este nuevo estado se vuelve a computar la compuerta NOR para luego seguir con un proceso de RESET al elemento conectado a la entrada $V_{In,1}$. Con los dispositivos ahora en estado 0/0 se vuelve a computar la compuerta NOR registrando el voltaje de salida. Finalmente se realiza un proceso de SET en el dispositivo conectado a la entrada $V_{In,1}$ dejando los dispositivos en un estado 1/0 con los cuales se computa la compuerta NOR por última vez. Estas pruebas nos permitirán observar los niveles de voltaje que se tendrán a la salida de cada amplificador sumador para cada combinación de entradas en los dispositivos.

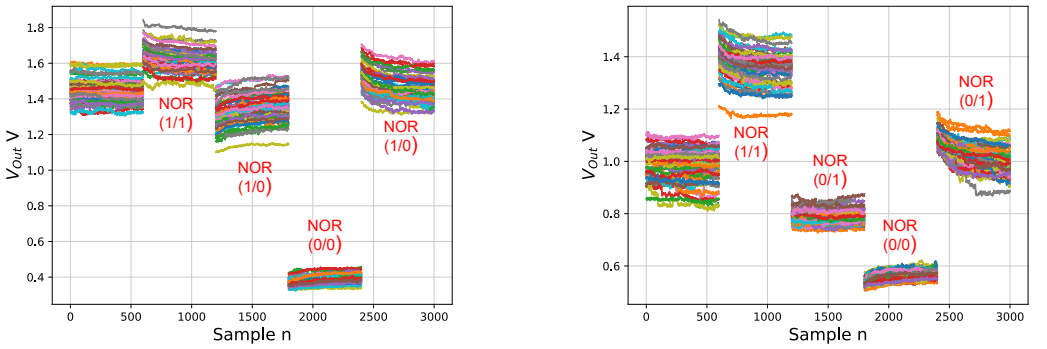
En un principio solo se considera el estudio de los voltajes de salida del amplificador sumador, puesto que aún no se selecciona un umbral de voltaje adecuado. De esta forma se aplica 50 repeticiones de estas señales de entrada que implementan las pruebas descritas previamente, en donde los voltajes de salida se presentan en la figura 3.27 con el detalle del estado en el que se encuentran los dispositivos al momento de realizar el computo de la compuerta NOR. En este caso solamente se trabaja con los dispositivos M3 y M4. De esta forma en la figura 3.27-a se tienen los voltaje de salida resultados de aplicar $V_{In,1}$ a M3 y $V_{In,2}$ a M4. Por otro lado en la figura 3.27-b se tienen los voltajes de salida del amplificador sumador resultados de aplicar $V_{In,1}$ a M4 y $V_{In,2}$ a M3. La diferencia que implica la combinación entre entradas y dispositivos tiene que ver con el momento en el que se realiza el primer proceso de RESET. Esto se refleje principalmente en la diferencia de voltaje de la salida cuando se computa la compuerta NOR estando los dispositivos en estado 1/0 y en 0/1.



(a) **(b)**

Figura 3.27: Voltaje de salida producido por las 50 repeticiones de las señales de entrada al circuito (a) Voltajes salida medidos considerando $V_{In,1}$ como entrada a M3 y $V_{In,2}$ como entrada a M4.(a) Voltajes salida medidos considerando $V_{In,1}$ como entrada a M4 y $V_{In,2}$ como entrada a M3

De esta forma en la figura 3.28, como manera de visualizar la diferencia, se muestran solo los voltajes de salida que corresponden a la computación de compuertas NOR para ambas combinaciones de entradas. Aquí se observa que el nivel de voltaje cuando se computa la compuerta NOR (1/0 - 0/1) es menor cuando se realiza un RESET primero en el dispositivo M3, como se muestra en la figura 3.28-b, que cuando se realiza primero un RESET en el dispositivo M4, como se muestra en la figura 3.28-a. Esto se debe naturalmente a la diferencia en la magnitud de las resistencias para cada dispositivo. Como se observó previamente el dispositivo M3 posee una alta resistencia mayor que el dispositivo M4, el cual a su vez tiene una resistencia baja mayor que M3. De esta forma el voltaje que incorpora M3 al bajar a R_{ON} es mayor que el que incorpora M4 al realizar esta conmutación, lo que se ve reflejado en los voltajes antes mencionados.

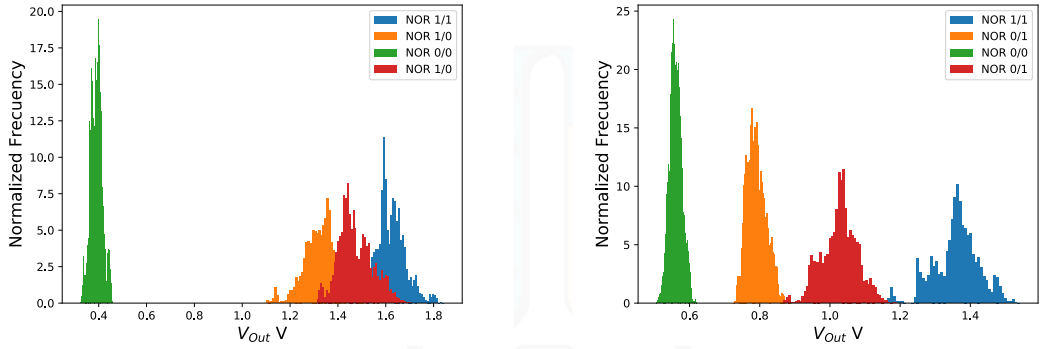


(a) **(b)**

Figura 3.28: 50 ciclos del voltaje de salida de los proceso de cómputo de compuerta NOR. (a) Salidas de las Pruebas considerando $V_{In,1}$ como entrada a M3 y $V_{In,2}$ como entrada a M4.(a) Salidas de las pruebas considerando $V_{In,1}$ como entrada a M4 y $V_{In,2}$ como entrada a M3

El fenómeno observado previamente es sumamente relevante para el proceso de selección de un umbral de voltaje adecuado que nos permita traducir estas salidas análogas a señales digitales. De esta forma los voltajes obtenidos previamente se procesan obteniendo un histograma de los mismos que se presentan en la figura 3.29. En estos histogramas se puede observar la distribución de voltajes para la computación de compuertas NOR con distintas entradas y combinación de las señales de entradas con los dispositivos. En la figura 3.29-a se muestra la distribución de voltajes de salida considerando como entradas a M3 la señal $V_{In,1}$ y al dispositivo M4 la señal $V_{In,2}$(la cual recibe el RESET primero). Como se puede observar en este caso la distancia entre la distribución del voltaje para el computo de NOR (1/0) y la distribución de cómputo con NOR (0/0) es la máxima posible. Por otro lado en el histograma en el que se realiza el RESET primero en el

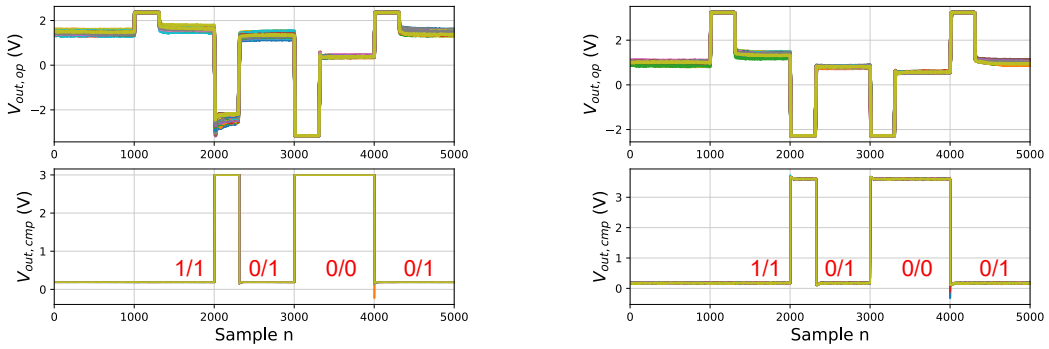
dispositivo M3 mostrado en la figura 3.29-b se obtiene la distancia crítica. Esto sucede entre la distribución de NOR (0/0) y NOR (0/1). Es por esto que este último gráfico es el que nos permite encontrar el umbral adecuado para la etapa de comparación.



(a) **(b)**

Figura 3.29: Histogramas obtenidos a partir de voltaje de salida en las pruebas de computación con M1 y M3. (a) Histograma con salidas de las Pruebas considerando $V_{In,1}$ como entrada a M3 y $V_{In,2}$ como entrada a M4.(b) Histograma con salidas de las pruebas considerando $V_{In,1}$ como entrada a M4 y $V_{In,2}$ como entrada a M3

A partir del estudio anterior se decide seleccionar como umbral de voltaje para el comparador el valor de 0.7 V. De esta forma ahora se conecta la sección del buffer y comparador con histéresis para realizar la computación lógica.



(a) **(b)**

Figura 3.30: 50 ciclos de salidas del amplificador sumador y comparador a partir de las pruebas de computación. (a) Salidas de las Pruebas considerando $V_{In,1}$ como entrada a M3 y $V_{In,2}$ como entrada a M4.(a) Salidas de las pruebas considerando $V_{In,1}$ como entrada a M4 y $V_{In,2}$ como entrada a M3

Se utilizan nuevamente las entradas que se muestran en la figura 3.26 y se captura la salida del amplificador sumador y la salida del comparador, estos se presentan en la figura 3.30. Al igual que se realizó anteriormente en este caso también se considera la relevancia de la combinación de entradas y dispositivos. De esta forma en la figura 3.30-a se presentan las salidas cuando las entradas $V_{In,1}$ y $V_{In,2}$ se aplican sobre M3 y M4 respectivamente. En este gráfico se observa los niveles de voltaje en la salida del sumador en conjunto con el voltaje a la salida del comparador, en este último se destacan los estados de los dispositivos al momento de computar las compuertas NOR. En esta prueba se esperaba que el comparador entregara un '1' lógico cuando los dispositivos se encontraran ambos en baja resistencia o lógico '0/0', lo que finalmente sucede posterior a la segunda operación de RESET para cada uno de los 50 ciclos que se probaron. Este primer caso correspondía al mejor caso al tener la mayor diferencia de voltaje entre el computo de NOR (1/0) y NOR (0/0). El correcto funcionamiento del cómputo lógico en este circuito se replica ahora cuando las entrada $V_{In,1}$

y $V_{In,2}$ se aplican ahora a M4 y M3 respectivamente. Lo anterior se presenta en la figura 3.30-b en donde nuevamente se tiene a la salida del comparador un '1' lógico cuando se computa la compuerta NOR (0/0) para cada uno de los 50 ciclos. Este último caso correspondía al crítico y el que se utilizó para seleccionar el umbral de voltaje, por lo que se esperaba que funcionara de forma correcta.

Posterior a estas pruebas el objetivo era continuar en la construcción de la línea de memoria, pero eventualmente los dispositivos que se estaban utilizando dejaron de conmutar quedando permanentemente atrapados en alta resistencia. Uno de los motivos detrás de esta problemática puede relacionarse a la asimetría de los pulsos de SET y RESET, siendo este último el de mayor resistencia superando los 1 V recomendados.

3.4.5. Exploración de dispositivos después de re-forming

Con los dispositivos utilizados en el punto anterior se intentó un proceso de re-forming para lograr que estos volvieran a operar de la manera que lo estaban haciendo. Para este proceso se aplicaron altos voltajes de SET y RESET que producían que el dispositivo alternara entre un estado continuo de alta y baja resistencia. Finalmente se logró llegar a una operación de histéresis característica de los *memristors* obteniendo las curvas de I-V que se observar en la figura 3.31-a para el dispositivo M3 y figura 3.31-b para el dispositivo M4, en donde se utiliza una curva de voltaje triangular periódica de 0.8 V de amplitud. Como se puede observar la operación de estos elementos reformados a cambiado sustancialmente si es que se compara a los dispositivos originales de la figura 3.20-b y 3.20-a en donde se tiene la curva I-V de M3 y M4 respectivamente. La principal diferencia que se nota entre el comportamiento previo y el actual tiene que ver con el proceso de conmutación. Los umbrales se han desplazado y se necesita menos voltaje para lograr una conmutación completa y la saturación cuando se limita la corriente a 100 μA como se realizaba previamente. Ahora esto último no solo ocurre en la conmutación en un proceso de SET sino que también en el proceso de RESET, esto sugiere que se a disminuido considerablemente el rango de resistencias con el que se estaba trabajando.

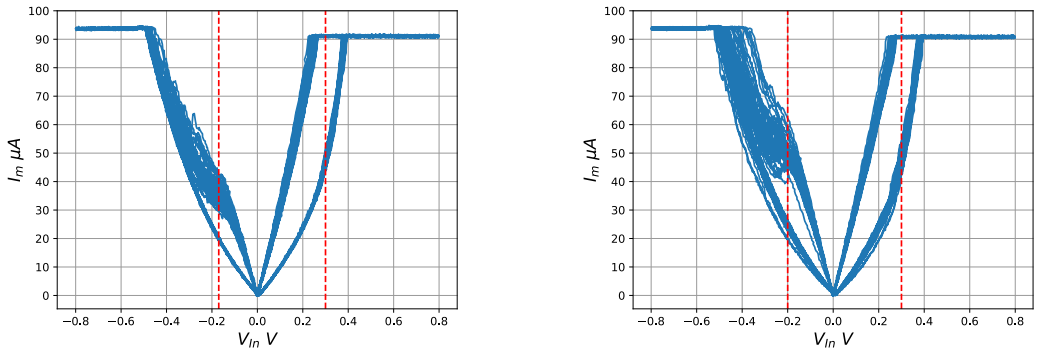
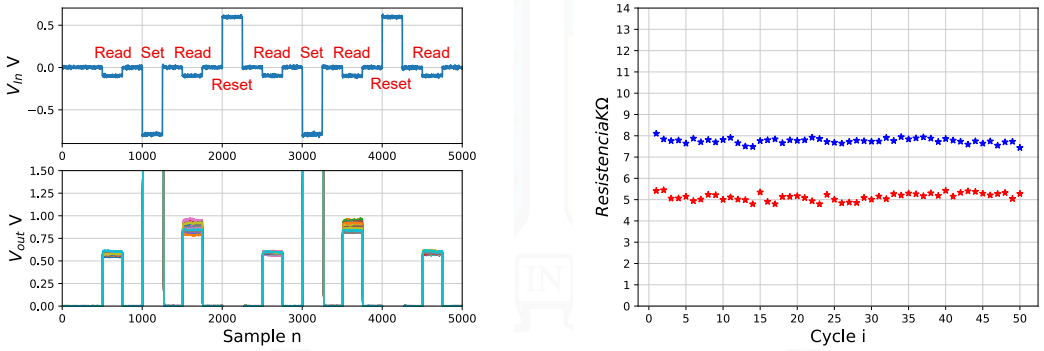


Figura 3.31: Curva característica dispositivos M3 y M4 posterior al re-forming. (a) Curva característica dispositivo M3.(a) Curva característica dispositivo M4

En esta ecuación con el nuevo comportamiento de estos dispositivos se decidió operar con ellos utilizando solamente pulsos de voltaje de un mismo ancho y no señales continuas como se estaba trabajando previamente. De esta forma acorde a la operación a realizar sera el valor de voltaje que tendrá. En este caso debido a la evidente disminución de los umbrales y la taza de cambio de resistencia de los dispositivos, se decidió disminuir los voltajes de SET y RESET a -0.8 y 0.6 V respectivamente favoreciendo en este caso al proceso de SET. El voltaje de lectura se mantuvo en 0.1 V.

Las primeras pruebas se realizan con el dispositivo M3 tal como se muestra en la figura 3.32-a. En este gráfico se presentan la señal de entrada la cual consiste en pulsos de lectura intercalado con pulsos de

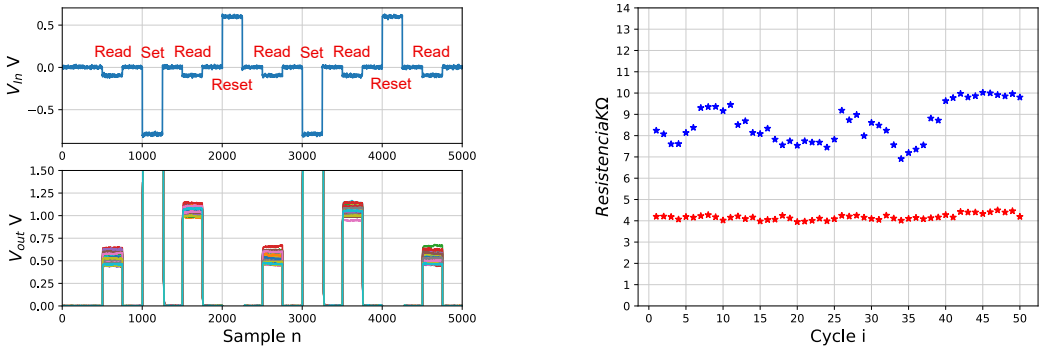
SET y RESET cada uno de 500 μs de duración, en total la señal dura 10 ms y se repite 50 veces la prueba. También se pueden observar los voltajes de salida del amplificador sumador para los 50 ciclos. Con este último se obtiene un promedio de las resistencias para cada uno de los ciclos lo que se muestra en la figura 3.32-b. Como se mencionó previamente el rango de resistencia disminuyo considerablemente alcanzando una razón $R_{OFF}/R_{ON} < 2$. Lo anterior produjo una disminución en la diferencia de voltaje logrando una ventana de ≈0.2 V



(a) **(b)**

Figura 3.32: 50 ciclos de pulsos de SET, RESET y lecturas sobre el dispositivos M3 (a) Voltaje de entrada y salida del proceso de SET, RESET y lectura.(b) Resistencia R_{ON} y R_{OFF} promedio en cada uno de los ciclos de operación

El proceso anterior se repite con el dispositivo M1, en donde los voltajes de salida posterior a las 50 repeticiones de aplicar la entrada antes detallada, se muestra en la figura 3.33-a. En esta ocasión la ventana de voltaje es mayor, siendo esta de ≈0.5 V. Lo anterior se explica si se observa el promedio de resistencia para cada ciclo en la figura 3.33-b, en donde este aumento en la ventana se produce principalmente por la disminución de la resistencia R_{ON} respecto al dispositivo anterior.



(a) **(b)**

Figura 3.33: 50 ciclos de pulsos de SET, RESET y lecturas sobre el dispositivos M4 (a) Voltaje de entrada y salida del proceso de SET, RESET y lectura.(b) Resistencia R_{ON} y R_{OFF} promedio en cada uno de los ciclos de operación

La siguiente prueba corresponde a la operación combinada de los dispositivos, esto de forma que se logre la computación de compuertas lógicas NOR. En este sentido se generan nuevamente entradas $V_{In,1}$ y $V_{In,2}$ que se muestran en la figura 3.34. Esta vez, al igual que en el caso de los dispositivos individuales de esta sección, se utilizan pulsos de un mismo ancho para cada operación, no obstante, se ha disminuido el voltaje de SET y RESET a -0.64 y 0.48 V respectivamente de forma de no desgastar innecesariamente los dispositivos.

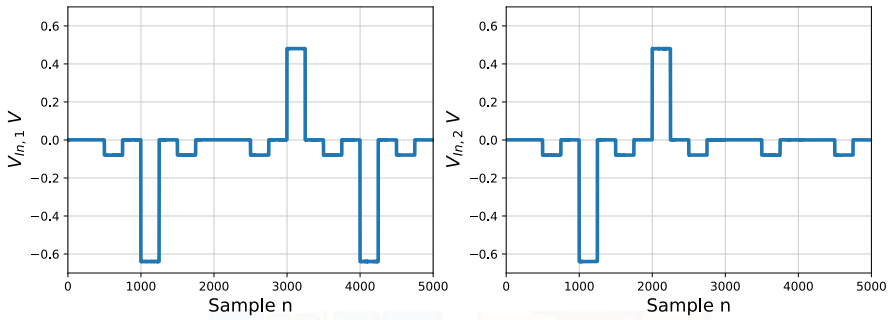
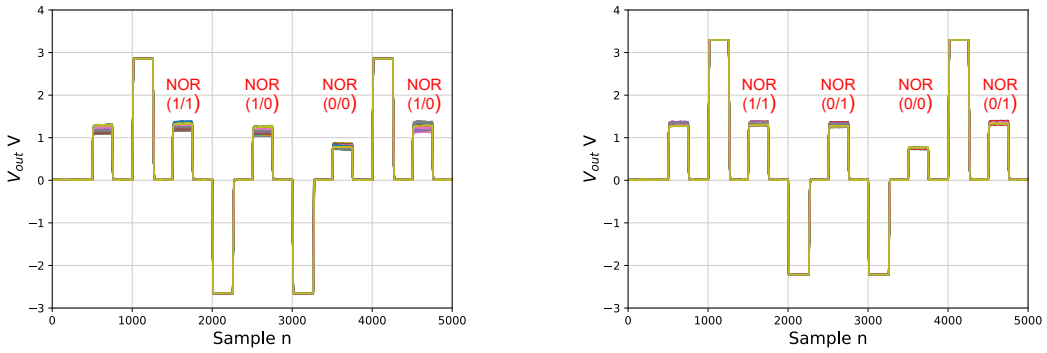


Figura 3.34: Entradas a los dispositivos utilizados en las pruebas de computación $V_{In,1}$ y $V_{In,2}$.

De esta forma se sigue el mismo orden de pruebas que en la sección anterior, en donde se aplican los 50 ciclos de entradas sobre los dispositivos combinados previo a la conexión de la etapa de comparación. Para esto se tomó en consideración también la combinación entre las entradas y dispositivos. De esta forma en la figura 3.35-a se muestra los voltajes de salida del sumador tras aplicar los 50 ciclos de la señal $V_{In,1}$ sobre M3 y $V_{In,2}$ sobre M4. Por otro lado en la figura 3.35-b se tiene la salidas del amplificador sumador tras aplicar los 50 ciclos de la señal $V_{In,2}$ sobre M3 y $V_{In,1}$ sobre M4. En los gráficos se detalla los estados de los dispositivos cuando se computa una compuerta NOR. De esta forma se puede reconocer que el menor voltaje corresponde a NOR (0/0) el cual es reconocible del resto de combinaciones.

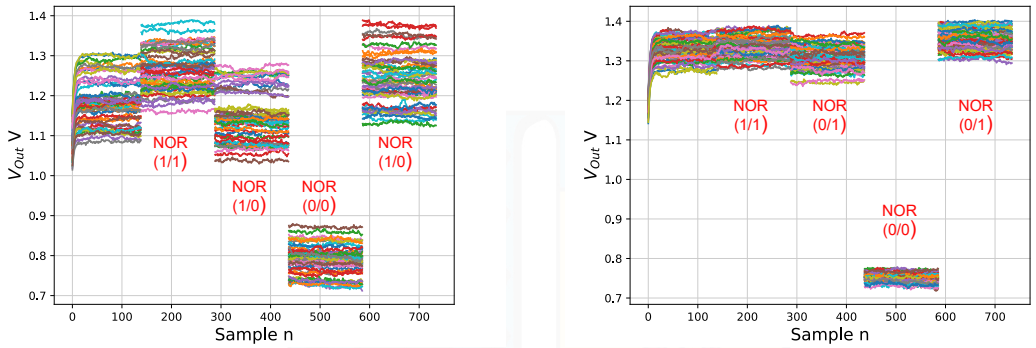


(a) **(b)**

Figura 3.35: Voltaje Voltaje de salida producido por las 50 repeticiones de las pruebas de computación con dispositivos reformados. (a) Voltajes de salida considerando $V_{In,1}$ como entrada a M3 y $V_{In,2}$ como entrada a M4.(b) Voltajes de salida considerando $V_{In,1}$ como entrada a M4 y $V_{In,2}$ como entrada a M3

En este caso a simple vista pareciera que la diferencia de voltaje entre el menor de los voltajes NOR (0/0) y el resto de los cómputos de compuerta NOR es indistinta para los dos casos. No obstante, para corroborar lo anterior se gráfica solamente los voltajes relacionados a los proceso de cómputo NOR para ambas combinaciones de entradas y dispositivos. De esta manera en la figura 3.36-a se puede observar el primer caso, en el cual el primer elemento al que se le aplica un RESET corresponde a M4. Por otro lado en la figura 3.36-b se observa el segundo caso en el que se aplica RESET primero a M3. Al igual que se analizó en la sección anterior con estos dispositivos también se tienen diferencias entre ambos casos, siendo la principal de estas la diferencia entre el voltaje NOR (0/0) y los voltajes de las operaciones adyacentes, en particular con las operaciones NOR (0/1) y NOR(1/0). En este sentido la mejor diferencia de voltaje entre estas operaciones se logra en el primer caso en el que se realiza el RESET al dispositivo M3 primero, como se observa en la figura 3.36-b. Lo anterior es esperable considerando que el dispositivo M4 posee el menor valor de resistencia R_{ON} por lo que tiene el mayor impacto en la suma de voltajes en la salida. Contrario a lo que sucede con el dispositivo M3 cuya mayor resistencia R_{ON} produce un menor voltaje en la salida, como se

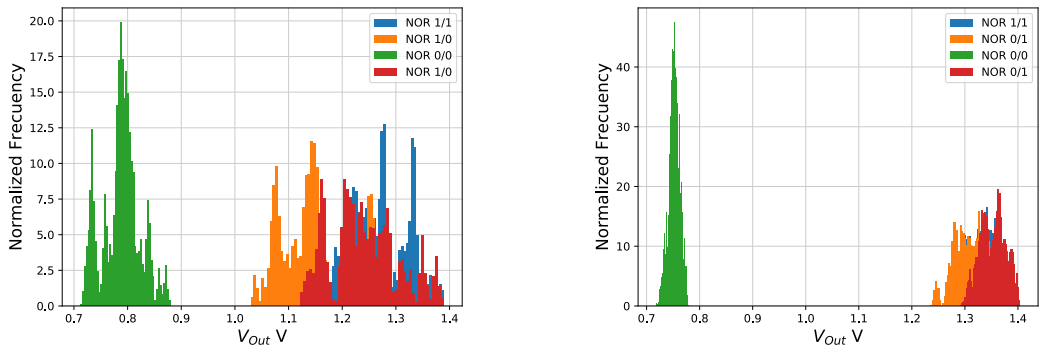
puede observar en la figura 3.36-a para la operación $NOR(0/1)$.



(a) **(b)**

Figura 3.36: Voltaje de salida producido por las 50 repeticiones de las pruebas de computación con dispositivos reformados, solo voltajes de lectura. (a) Voltaje de salida de la lectura considerando $V_{In,1}$ como entrada a M3 y $V_{In,2}$ como entrada a M4.(a) Voltaje de salida de la lectura considerando $V_{In,1}$ como entrada a M4 y $V_{In,2}$ como entrada a M3

Esta vez al igual que en la sección anterior se obtienen los histogramas para cada operación considerando los 50 ciclos de operación. Esta representación de datos se realiza nuevamente para ambas combinaciones de entradas y se muestra na la figura 3.37. El caso en el que $V_{In,1}$ está asociado a M3 y $V_{In,2}$ a M4 se observa en la figura 3.37-a mientras que en la figura 3.37-b se muestra el caso en el que $V_{In,2}$ es entrada de M3 y $V_{In,1}$ es entrada a M4. En los histogramas se observa la misma diferencia analizada en el párrafo anterior, en el que se observó que el realizar un RESET en M4 primero produce una diferencia de voltaje entre NOR (0/1) y NOR(0/0) menor que en el caso de realizar un RESET en M3 primero. De esta forma el primero sera el caso crítico y se utilizara para la selección del umbral de voltaje que permita diferenciar un lógico '0' de un lógico '1' en el cómputo de la compuerta NOR.



(a) **(b)**

Figura 3.37: Histograma que muestra la distribuían de los voltajes de salida para el total de la prueba con dispositivos reformados (50 repeticiones). (a) Histograma de pruebas con $V_{In,1}$ como entrada a M3 y $V_{In,2}$ como entrada a M4.(b) Histograma de pruebas con $V_{In,1}$ como entrada a M4 y $V_{In,2}$ como entrada a M3

A partir de la información entregada por los histogramas se seleccionó un umbral de voltaje adecuado para la etapa de comparación. En este caso este parámetro se eligió con un valor de 0.95 V. De esta forma se agregó la etapa de comparación y se volvió a aplicar los 50 ciclos de las señales de entrada que se muestran en la figura 3.34. Los resultados de estas pruebas se pueden observar en la figura 3.38 en donde se muestra el voltaje de salida del comparador para las pruebas realizadas. Nuevamente se hace la distinción entre los dos casos de combinación de entradas, en donde se puede observar que el computo funciona de forma correcta para los 50 ciclos y para ambos casos. En las figuras se detalla la operación realizada y los estados

de los dispositivos, de esta forma se observa un '1' lógico en el computo solamente cuando el estado de los dispositivos es (0/0) y en ningún otro caso.

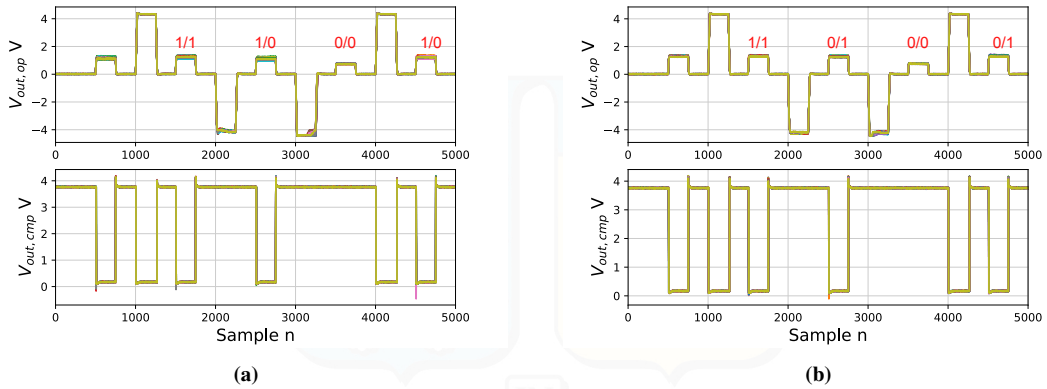


Figura 3.38: Voltaje Voltaje de salida del comparador producido por las 50 repeticiones de las pruebas de computación con dispositivos reformados. (a) Voltaje de salida considerando $V_{In,1}$ como entrada a M3 y $V_{In,2}$ como entrada a M4.(a) Voltaje de salida considerando $V_{In,1}$ como entrada a M4 y $V_{In,2}$ como entrada a M3

Con estos se concluye la etapa de pruebas experimentales en donde se logró utilizar dispositivos *memristors* reales no tan solo para procesos de escritura, sino que también para la computación de compuertas lógicas. Se comprobó la funcionalidad del amplificador sumador como topología de computación lógica, corroborando su utilidad para dispositivos con una baja razón de resistencia y distintas entre sí. En las primeras pruebas se tenía dispositivos con razón de resistencia $\frac{R_{OFF}}{R_{ON}} > 5$ mientras que en la última prueba e tenía $\frac{R_{OFF}}{R_{ON}} < 2$ lo que comprueba la variedad de dispositivos que se puede utilizar para esta aplicación utilizando esta estructura.

Vale la pena también mencionar el impacto de la resistencia mínima en el funcionamiento de la aplicación que se implementó en esta sección. Como se pudo observar es este parámetro el que se mantiene, aproximadamente, entre los dispositivos originales y los que se reformaron en la última parte. En este sentido en este último experimento se lograba una correcta funcionalidad con dispositivos cuya resistencia máxima era muy cercana a la resistencia mínima, es más, se puede considerar que se está trabajando con un proceso de RESET incompleto por lo que no se está alcanzando la resistencia de R_{Off}. No obstante, el correcto funcionamiento se puede explicar debido a que la relación entre estas resistencias y la resistencia de retroalimentación, cuya razón es suficientemente alta como para obtener una ventana que permitiera diferenciar los voltajes de lógico '0' y lógico '1' a la salida del amplificador sumador. En este sentido se puede concluir que es necesario que por lo menos una de los dos estados resistivos de los dispositivos (alta y baja resistencia) mantenga una resistencia menor a la resistencia de retroalimentación del amplificador sumador o en su defecto diseñar esta última de forma acorde. Lo anterior implica que con esta topología no es necesario hacer un proceso de RESET completo, si no que se debe cumplir con esta característica, por lo que se pueden disminuir los voltajes de RESET pudiendo de esta forma disminuir el consumo del sistema.

Por otro lado, no se logró concretar la construcción de una línea de memoria completa, a pesar de esto se logró un ejemplo funcional de lo que significa la operación lógica con *memristors*, además de la posibilidad de incluir operaciones propias de una memoria.

4 | Desarrollo de Herramienta de Síntesis

4.1. Herramientas de síntesis

En las secciones anteriores se habló sobre el funcionamiento del *memristor* y su papel como elemento de memoria. Se demostró también la manera de la cual se puede lograr la ejecución de funciones lógicas con *memristors* y se explicó cómo se relaciona esta funcionalidad con la topología de memoria diseñada. Con esto se puede en efecto diseñar un sistema computacional en base a estas memorias resistivas, de forma de lograr reemplazar las memorias convencionales con una memoria resistiva ReRAM computacional [44] que permitan complementar el funcionamiento de la CPU de un sistema convencional con computación en la memoria. No obstante, hasta ahora se ha hablado solamente de ejecución de compuertas lógicas (NOR2/NOT) estudiadas de manera aislada. Para la ejecución de funciones lógicas arbitrarias se debe diseñar el flujo de ejecución de esta utilizando las compuertas lógicas base que el sistema es capaz de implementar. Naturalmente hacer este diseño de manera manual no resulta practico, así como tampoco óptimo. De esta forma se vuelve una necesidad el tener un sistema que permita la ejecución de funciones lógicas arbitrarias en la memoria diseñada, tomando en cuenta restricciones de espacio y latencia. Este sistema corresponde a las herramientas de síntesis, que en este caso combina la síntesis lógica correspondiente al procesamiento de la función original a computar y de expresiones lógicas equivalentes usando compuertas base soportadas por el sistema, y la síntesis física que corresponde al mapeo espacio-temporal de la ejecución de las compuertas lógicas en el arreglo de memoria.

Más específicamente, la herramienta de síntesis tiene la misión de determinar el flujo de ejecución de una función lógica arbitraria, reservando espacio de la memoria donde se realiza la computación, donde se almacenarán los resultados y además organizar estas operaciones en el tiempo. De esta forma a partir de una implementación multinivel de una función lógica arbitraria la herramienta debe ser capaz de asignar una posición en el arreglo a todas las entradas y salidas de las compuertas que conforman la función lógica a ejecutar. Todo esto mientras se toman en cuenta consideraciones de espacio y latencia para la topología determinada de la memoria utilizada y estilo de lógica asumido.

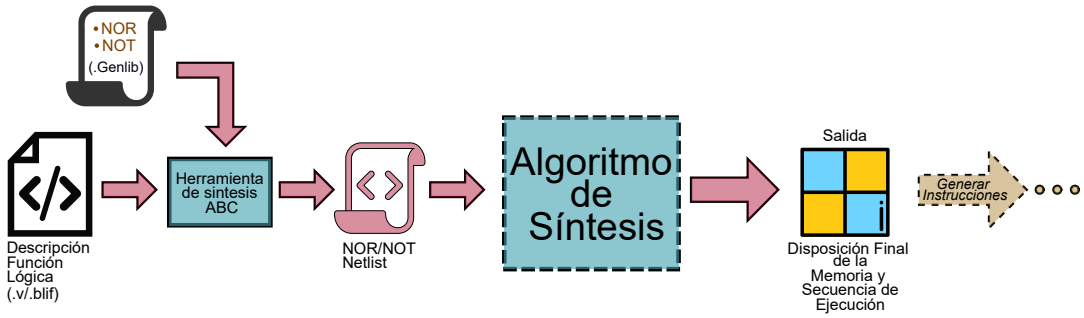


Figura 4.1: Esquema general del proceso de síntesis para ReRAM

En la literatura existen una serie de diseños para el flujo de síntesis, pero la gran parte de ellos comparten algunos elementos generales. En la figura 4.1 se muestra uno de los esquemas que más se repite entre los trabajos existentes, en donde se observan tres etapas fundamentales. La primera etapa está constituida por elementos de pre-procesamiento que buscan tomar una descripción en HDL (Hardware Description Language) de alguna función lógica arbitraria y convertirla a una lista de compuertas compatibles con el estilo lógico utilizado. Una de las herramientas preferidas para esta etapa es el *ABC Tool* [45], el cual es un software para síntesis y verificación de circuitos lógicos secuenciales binarios. Esta herramienta combina distintos algoritmos para la optimización de circuitos lógicos, desde métodos basados en AIG (And-Inverter Graph), retardo óptimo basado en tecnología de mapeo DAG (Directed Acyclic Graph) para Look-up tables, entre otros. La segunda etapa consiste en un algoritmo de síntesis que viene a tomar las cualidades propias de los arreglos de memoria diseñados, así como el estilo de lógica utilizado para generar el orden de mapeo de la función lógica a sintetizar. La tercera etapa corresponde a la encargada de presentar la distribución en la memoria en una representación visual, así como también hacer de interfaz entre el sintetizador y el módulo de control que permite generar las instrucciones para la ejecución de las operaciones lógicas. A continuación, se pasan a detallar algunos métodos disponibles en la literatura que se consideran más relevantes para el trabajo aquí realizado.

4.1.1. SIMPLE MAGIC

Una de las primeras herramientas de síntesis para memorias ReRAM en ser desarrolladas corresponde a SIMPLE MAGIC [46]. Esta herramienta considera como forma de hacer lógica con *memristors* el estilo MAGIC(*Memristor-Aided Logic*) implementando funciones lógicas complejas a través del cómputo de una serie de compuertas NOR, NOT y copy. Lo anterior se realiza a la vez que se minimiza la latencia en la ejecución de la función lógica a sintetizar, a través de la división óptima de esta en una serie de pasos de ejecución reducidos. De esta forma esta herramienta plantea el mapeo como un problema de optimización para obtener la mejor secuencia de operaciones NOR/NOT.

Este problema de optimización se construye mediante una serie de restricciones acordes al estilo de lógica MAGIC aprovechando la capacidad de computación en paralelo de las memorias resistivas. De esta forma esta herramienta comienza con la primera etapa del flujo mostrado en la figura 4.1, en la que se utiliza la *ABC Tool* para procesar la descripción de la función lógica a sintetizar (en formato verilog o blif). Esta herramienta entrega una version de esta descripción lógica que solo cuenta con compuertas NOR y NOT, siendo estas las dos operaciones que el estilo MAGIC puede realizar en el arreglo de memoria. Es con este último elemento que se realiza el mapeo en esta herramienta de acuerdo al flujo presentado en la figura 4.2.

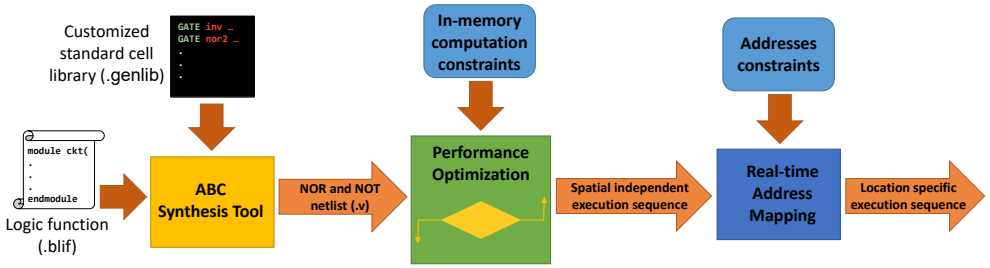


Figura 4.2: Flujo de síntesis de la herramienta SIMPLE MAGIC [46]

Como se mencionó previamente esta herramienta busca sacar provecho de la capacidad de ejecución en paralelo que el estilo MAGIC soporta, para esto el posicionamiento de los datos en los cross-points del arreglo es clave. De esta forma para identificar a cada compuerta lógica j se tienen las siguientes variables:

- $\{(R_{A_j}, C_{A_j}),(R_{B_j}, C_{B_j}),(R_{E_j}, C_{E_j})\}$: Posición fila-columna (R,C) en el arreglo de las entradas A_j y B_j y salida E_j

- T_j: Ciclo de reloj en el que se ejecuta la compuerta j.

Estas variables son las que dan los grados de libertad al problema de optimización el cual se plantea mediante restricciones dadas por criterios de localización, conectividad y tiempo de ejecución de las compuertas. Estas restricciones se plantean como:

- Restricción del tamaño del arreglo: El mapeo de las compuertas debe realizarse en el espacio físico disponible del arreglo. Formalmente para un tamaño igual a Row_{num} x Col_{num} se tiene que la restricción es:

$$(0 < C_{x_j} < Col_{num}) \cup (0 < R_{x_j} < Row_{num}) \tag{4.1}$$

- Restricción de salidas de distintas compuertas: Dos salidas no se pueden asignar a una misma celda de memoria. Esta herramienta no considera reutilizar las celdas ya usadas. Formalmente se tiene que esta restricción esta dada por:

$$\forall E_k, E_j : (C_{E_j} \neq C_{E_k}) \cup (R_{E_j} \neq R_{E_k}) \tag{4.2}$$

- Restricción de celdas usadas por una misma compuerta: Entradas/Salidas de cada compuerta deben ser asignadas a la misma fila y diferente columnas o misma columna y diferente fila. Formalmente se tiene:

$$\forall j : [(C_{A_j} = C_{B_j} = C_{E_j}) \cap (R_{A_j} \neq R_{B_j} \neq R_{E_j})] \cup [(C_{A_j} \neq C_{B_j} \neq C_{E_j}) \cap (R_{A_j} = R_{B_j} = R_{E_j})] \tag{4.3}$$

- Restricción para ejecución simultanea de distintas compuertas: Esta ejecución en paralelo solamente es posible cuando las entradas y salidas, entre sí, de las compuertas a ejecutar en paralelo se encuentran alineadas en las filas o columnas. Formalmente se tiene que:

$$\forall gates\, j,k : T_j \neq T_k \cup \begin{cases} \{[(C_{A_j} = C_{A_k} \cap C_{b_j} = C_{b_k}) \cup (C_{A_j} = C_{B_k} \cap C_{A_k} = C_{B_j})] \cap C_{E_j} = C_{E_k}\} \\ \cap (R_{A_j} = R_{B_j} = R_{E_j} \cap R_{A_k} = R_{B_k} = R_{E_k}) \end{cases} \cup \begin{cases} \{[(\mathrm{R}_{A_j} = R_{A_k} \cap R_{b_j} = R_{b_k}) \cup (R_{A_j} = R_{B_k} \cap R_{A_k} = R_{B_j})] \cap R_{E_j} = R_{E_k}\} \\ \cap C_{A_j} = C_{B_j} = C_{E_j} \cap C_{A_k} = C_{B_k} = C_{E_k}) \end{cases} \cup \tag{4.4}$$

- Restricciones de conectividad: Todas las salidas de una compuerta h corresponde a la entrada de una compuerta j por lo que se deben mapear en una misma celda. Formalmente se tiene que.

$$\forall E_h, x_j \in A_j, B_j : [(C_{E_h} = C_{x_j}) \cap (R_{E_h} = R_{x_j})] \cap (T_h < T_j) \tag{4.5}$$

De esta forma la herramienta busca la combinación de posiciones y tiempos de ejecución que satisfagan el conjunto de ecuaciones que componen las restricciones presentadas anteriormente. Para encontrar una solución a este problema de optimización se utiliza la herramienta *Z3 Theorem Prover*[47] la cual también forma parte de este trabajo y por eso se explicara en secciones siguientes. Dentro de los principales inconvenientes que introduce esta herramienta se encuentra los altos tiempos de síntesis que puede alcanzar, esto a medida que las funciones lógicas a sintetizar aumentan en número de compuertas. Estos tiempos de síntesis pueden volverse poco prácticos alcanzando los días y semanas para descripciones con cientos de compuertas. Por otro lado, esta herramienta se concentra en optimizar la latencia y no considera un uso eficiente del área. En este sentido no se incorpora la reutilización de celdas de memoria en el proceso de síntesis lo que podría llevar a un mejor uso del espacio de memoria.

4.1.2. Herramienta de síntesis general

Otra de las herramientas relevantes desarrolladas previamente es la herramienta de síntesis general para lógica basada en *memristors* presentada en [48]. El principal propósito de este flujo de síntesis es lograr una herramienta universal capaz de adaptarse a las distintas familias de estilos de lógica con *memristors* existentes en la literatura, haciendo la distinción entre lógica *stateful* y *non-stateful*. Lo anterior se realiza a la

vez que se toma en consideración una ocupación realista de la memoria que busca evitar la corrupción de los datos de la memoria, es decir, evitar la pérdida de datos ya almacenados debido a la sobreescritura de ellos con resultados intermedios de la computación lógica en curso.

El flujo de síntesis de esta herramienta se puede observar en la figura 4.3, en donde al igual que sucede en el caso de SIMPLE MAGIC, se considera también una etapa de pre-procesamiento basada en el software *ABC Tool*. En este caso la diferencia radica en que no se realiza para un estilo de lógica en específico por lo que la lista de primitivas lógicas que genere para el proceso de síntesis y mapeo dependerá de las características del estilo lógico. Precisamente para esto se genera una descripción de las primitivas lógicas en la que se extraen parámetros del estilo de lógica y sus características principales, siendo estas: representación de variables, patrón de conexión de compuertas, numero de dispositivos utilizados, resistencia inicial, etc. Como se observa en el flujo de síntesis es este resumen de primitivas el que se procesa para generar las reglas de mapeo y restricciones necesarias, las que también consideran el estado actual de la memoria. Estas reglas y restricciones son llevadas a ecuaciones que son luego solucionadas por una etapa de optimización que también considera el uso del *Z3 Theorem Prover*.

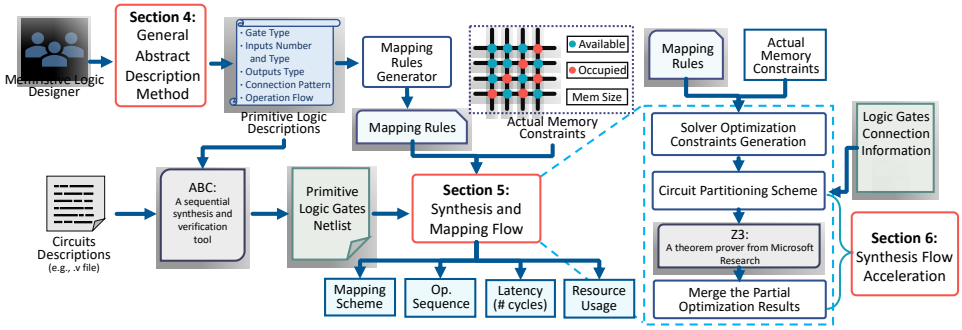


Figura 4.3: Flujo de síntesis de la herramienta general[48]

Otra diferencia fundamental con la herramienta previa es la inclusión de un acelerador de síntesis. Se plantea que esto se vuelve necesario dado que a medida que aumentan la cantidad de compuertas de una descripción lógica, se aumenta el tamaño del espacio reservado dentro del arreglo de la memoria y por consiguiente el espacio a explorar para el mapeo. Esto degenera en tiempos de síntesis restrictivos, del orden de días y hasta semanas. En este sentido en este trabajo se introduce un algoritmo que permite acelerar el tiempo de síntesis, lo que se logra mediante la división del circuito lógico original en sub-bloques los que son sintetizados de manera individual. Lo anterior trae consigo un aumento en el espacio requerido dentro del arreglo y la latencia de ejecución, pero una disminución en los tiempos de síntesis, que es un tema muy relevante también para el trabajo realizado en el contexto de esta Tesis.

Como se explicó esta estructura de síntesis general tiene la principal característica de compatibilizar distintos estilos de lógica en un mismo flujo de síntesis. Por otro lado este trabajo no elabora respecto a el diseño de los circuitos de control de la memoria ReRAM que permitirían implementar distintos estilos de lógica en una misma memoria. Como se explicó previamente en esta tesis, cada estilo de lógica está estrechamente ligada a la topología base de la memoria, definiendo la forma de los controladores de filas y columnas. De esta forma una implementación general difícilmente puede dar una respuesta adecuada a todo tipo de estilos de lógica. Esto es algo que se elabora en esta tesis al diseñar primeramente un circuito de memoria compatible con el estilo de lógica utilizado y el que se utilizara posteriormente como base para el desarrollo de un algoritmo de síntesis.

4.1.3. Herramienta de síntesis SAID (Supergate-Aided Logic)

La siguiente herramienta de síntesis que destaca en la literatura corresponde a la basada en súper-compuertas o SAID [49] (Supergate-Aided Logic) que al igual que en la primera herramienta presentada esta basada en arreglos de memoria que implementan el estilo de lógica MAGIC. Esta herramienta introduce el

concepto de súper-compuerta la que corresponden a *Look-up tables* (LUT) que representan secciones del circuito lógico a sintetizar. Estas súper-compuertas pueden tener un numero arbitrario de entradas y productos internos por lo que pueden representar una amplia variedad de funciones lógicas. Además de esto cada LUT se mapea de manera genérica en el arreglo, de forma que el problema de mapeo se reduce a distribuir adecuadamente cada una de estas súper-compuertas acorde a criterios de latencia y área.

El flujo de síntesis de esta herramienta consta de tres partes que se muestran en la figura 4.4. En la primera etapa de síntesis lógica y optimización se genera la descripción lógica utilizando la herramienta *ABC Tool*, en este caso con la opción de mapeo k-LUT. El procesamiento se realiza sobre una representación AIG(And-Inverter Gates) de la función lógica que transforma el circuito lógico en una grafo dirigido y acíclico (DAG). Esta red de compuertas es posteriormente transformada a k-LUTs dependiendo del tamaño del circuito lógico, en donde se obtienen distintas representaciones para valores de k específicos, este parámetro representando el número de entradas de la súper-compuerta.

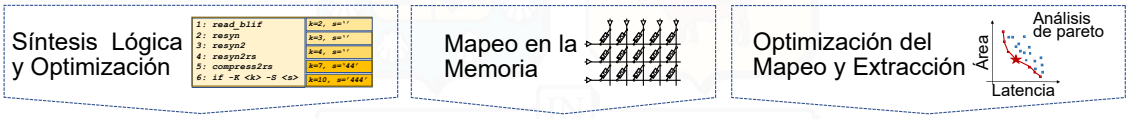


Figura 4.4: Flujo de síntesis de la herramienta SAID [49]

El siguiente paso del flujo de síntesis corresponde al mapeo en la memoria y para esto se implementa un algoritmo que mapea de manera ordenada y según la jerarquía correspondiente las LUT que representan la función lógica. Un ejemplo de este mapeo es el que se muestra en la figura 4.5, en donde se realiza con la descripción de un Full-Adder de 1 bit. Aquí se muestra la función representada utilizando LUTs de distintos tamaños y su posterior mapeo al arreglo de memoria. A su vez se muestra la cantidad de pasos para la ejecución de esta función, que incluyen operaciones de NOT, NOR y de copia, que son las soportadas por el estilo de lógica MAGIC.

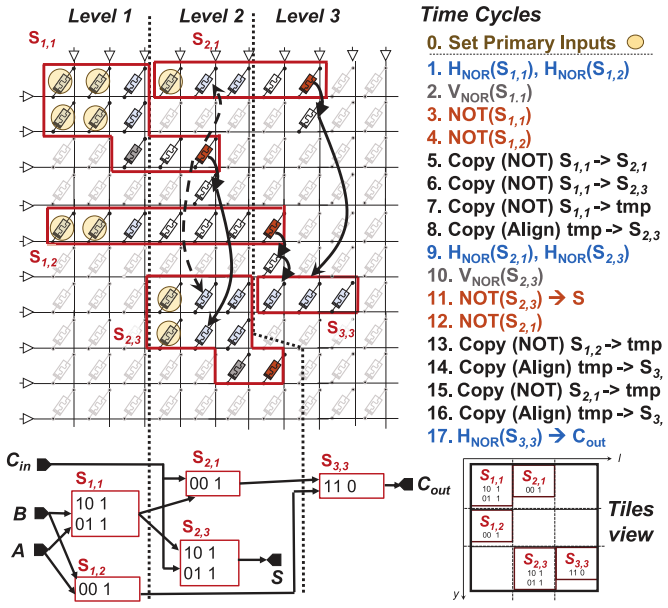


Figura 4.5: Ejemplo de mapeo de un Full-Adder de 1 bit utilizando el flujo de síntesis SAID [49]

El último paso del flujo de síntesis es obtener el mapeo óptimo en términos de área y latencia. En este sentido el parámetro a variar corresponde a k el cual se obtiene mediante un análisis de Pareto, el cual resuelve un problema de multiobjetivo.

Un problema que se puede identificar con esta herramienta de síntesis hace referencia al uso del espacio. Este flujo de síntesis considera una distribución óptima de los super-compuertas, pero no de las

compuertas individuales que la componen. De esta forma logra una minimización de la latencia en desmedro del área utilizada. En este sentido esta herramienta logra un trade-off entre el área utilizada y la latencia variando el parámetro k. Por otro lado, este trabajo no habla sobre el comportamiento del tiempo de síntesis ante funciones lógicas de distinto tamaño, un factor importante como se explicó previamente.

4.2. Herramienta de síntesis propuesta

4.2.1. Consideraciones iniciales

Considerando los antecedentes expuestos acerca de otras herramientas presentadas en este trabajo, se construye un flujo de síntesis orientado al estilo de lógica seleccionado y topología de memoria diseñada. En este caso el estilo *"Ratioed Logic"*, detallado en el capítulo 2, permite la utilización de compuertas NOT y NOR, siendo un estilo de lógica en el que las salidas de las compuertas se interpretan con niveles de voltaje. Además de esto se tiene la posibilidad de ejecutar compuertas en paralelo en el arreglo de memoria. El principal objetivo de este flujo de síntesis es lograr el distribuir las entradas y salidas de las compuertas lógicas que conforman una función lógica arbitraria, esto mientras se toman en cuenta consideraciones de espacio y latencia.

Para el desarrollo del algoritmo de síntesis se considera que un circuito lógico se puede representar como un grafo dirigido y acíclico (DAG). Los nodos de este grafo representan compuertas lógicas mientras que los vértices representan la conexión dirigida entre la salida de una compuerta a la entrada de un conjunto de compuertas de una implementación multinivel. De esta forma se puede tener como ejemplo el circuito lógico NOT/NOR de la figura 4.6-a que implementa la función lógica XOR de dos entradas, mientras que su grafo asociado se muestra en la figura 4.6-b. Esta representación admite tener una estructura de datos que muestre la dependencia entre niveles de las compuertas lógicas (nodos), lo que permite identificar a que niveles pertenece cada una de estas. Lo anterior es relevante para identificar aquellas compuertas lógicas que puedan ejecutarse en paralelo, estas corresponden precisamente a aquellas que se encuentran en el mismo nivel.

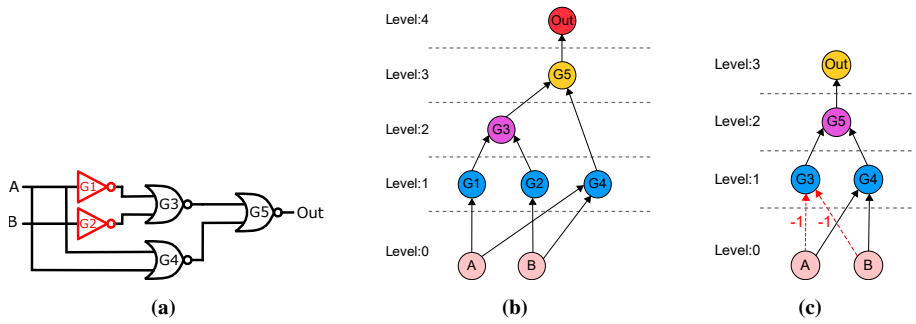


Figura 4.6: (a) Circuito lógico multinivel NOR/NOT de una compuerta XOR de dos entradas. (b) Representación del mismo circuito en forma de grafo, destacando los distintos niveles a los que corresponden las puertas lógicas. Los nodos en el nivel 0 y el nivel 4 representan las señales de entrada y de salida, respectivamente. (c) Representación del mismo circuito en forma de grafo ponderado, destacando los distintos niveles a los que corresponden las puertas lógicas. Los nodos en el nivel 0 y el nivel 3 representan las señales de entrada y de salida, respectivamente. Las flechas rojas representan entradas negadas

A partir de estas definiciones se puede identificar que la mínima latencia posible en la ejecución de la función lógica está dada por el número de niveles totales del grafo menos uno (dado que el nodo que se encuentra en el último nivel representa el valor de la salida). De esta forma se define la computación como una serie de operaciones de cálculo de compuertas NOR/NOT en paralelo en donde cada paso de ejecución corresponde al nivel de la función lógica para la cual se está computando. Considerando esto el problema de mapeo de las puertas lógicas a la estructura de la memoria se puede individualizar para cada nivel, con la condición de mantener la coherencia de la dependencia entre compuertas.

Puesto que se asume una latencia asociada al número de niveles de la función lógica en su representación de grafo, resulta natural pensar que la única forma de aumentar el desempeño en este aspecto es lograr reducir el número total de niveles. Lo anterior resulta imposible si es que se asume un circuito NOR/NOT con un numero óptimo de compuertas, esto dado a que no se podrían simplificar todas las compuertas de un nivel para lograr esta reducción. No obstante, se debe tener en cuenta el circuito de memoria diseñado en el capítulo 3, en donde el circuito de la periferia (específicamente en el bloque de salida) tiene la capacidad de negar el resultado de cualquier operación lógica. De esta forma se puede aprovechar esta funcionalidad y no realizar operaciones NOT en la memoria, si no que directamente en la periferia previo al proceso de almacenado de los datos. Lo anterior permite descartar la compuerta NOT del diseño y focalizar la síntesis en implementaciones que utilizan únicamente compuertas NOR.

El concepto anterior se puede incluir en la representación de grafos a través de una ponderación de los vértices que unen cada nodo/compuerta. De esta forma considerando el circuito de la función XOR de la figura 4.6-a, en donde se destacan las compuertas NOT en rojo, se puede pasar a una representación de grafo como se muestra en la figura 4.6-c, en donde los nodos que correspondían a las compuertas NOT se reemplazan por una ponderación (destacado con lineas rojas punteadas). De este modo al no considerar las compuertas NOT de la descripción original (ya que las señales se pueden negar usando la circuitería en la periferia del módulo de memoria), no solo se está ahorrando en términos de espacio en el arreglo de memoria a ocupar para el cómputo, sino que también se reduce la cantidad de niveles en uno, teniendo un impacto positivo sobre la latencia final. Con esto el algoritmo a implementar trabajará sobre esta estructura para realizar el mapeo de las operaciones lógicas al espacio de la memoria, de acuerdo a los puntos asumidos en esta sección.

4.2.2. Desarrollo del algoritmo de síntesis

El algoritmo de síntesis desarrollado trabajara precisamente con la estructura de grafo dirigido, acíclico y ponderado (DAWG) para realizar el mapeo. El flujo completo de síntesis se presenta en la figura 4.7, cuya implementación se realiza en lenguaje de programación Python. En este esquema se reconocen los bloques esenciales del sintetizador, estas se pasan a detallar a continuación.

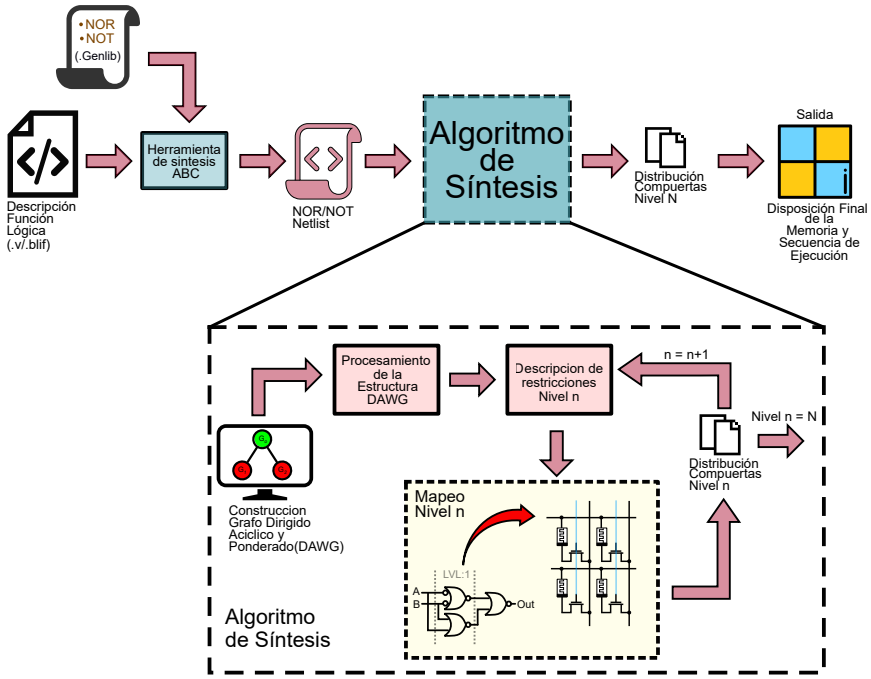


Figura 4.7: Flujo de síntesis propuesto para este trabajo

4.2.2.1. Procesamiento de la descripción lógica a sintetizar

Como entrada al flujo de síntesis se tiene la descripción de una función lógica en verilog o blif, la cual es entrada a la herramienta ABC[45]. Esta última es la encargada de procesar la descripción de la función lógica de entrada, en donde se toman criterios de numero de nodos y niveles para construir una expresión equivalente acorde a la librería utilizada. En este caso la herramienta ABC se utiliza con una librería personalizada que transforma la descripción de la función original a una expresión equivalente en compuertas NOR/NOT (.Genlib). Como ejemplo de este procesamiento se tiene la descripción en verilog de una compuerta XOR presentada a continuación.

```
module Xor (A,B);
//Input Ports Declarations
input A,B;
//Output Ports Declarations
output out;
//Logic
assign out = A^B;
endmodule
```

Naturalmente esta descripción no se puede mapear al arreglo de memoria. De esta forma se procesa utilizando la herramienta ABC obteniendo la siguiente descripción en compuertas NOR/NOT:

```
// Benchmark "notNor" written by ABC on Wed Apr 15 20:40:07 2020

module notNor (
    A, B,
    out  );
  input  A, B;
  output out;
  wire new_n4_, new_n5_, new_n6_, new_n7_, new_n8_;
  inv1 g0(.a(A), .O(new_n4_));
  nor2 g1(.a(B), .b(new_n4_), .O(new_n5_));
  inv1 g2(.a(B), .O(new_n6_));
  nor2 g3(.a(new_n6_), .b(A), .O(new_n7_));
  nor2 g4(.a(new_n7_), .b(new_n5_), .O(new_n8_));
  inv1 g5(.a(new_n8_), .O(out));
endmodule
```

Es este último conjunto de compuertas NOR/NOT que hace de entrada al algoritmo de síntesis desarrollado.

4.2.2.2. Traspaso descripción NOR/NOT a estructura de grafo

Posterior a la etapa de procesamiento de la descripción original de la función lógica se cuenta con su expresión en compuertas NOR/NOT en un archivo verilog, a partir del cual se debe obtener a su representación en grafo. Para construir el grafo se utiliza los diccionarios de Python como contenedor de distintos parámetros de interés para cada nodo de la estructura, información que es esencial para posteriormente lograr una correcta asignación en la memoria. De esta forma considerando las compuertas NOR como los nodos del grafo, se comienza a construir el diccionario utilizando los nodos como llaves, cada uno asociado a un diccionario con la siguiente información como llaves:

- Nivel: Corresponde al nivel en el circuito lógico en el que se encuentra el nodo.
- Entradas al Nodo: Conjunto de nodos cuyas salidas están conectadas con las entradas del nodo en cuestión.
- Nodos de destino: Conjuntos de nodos cuyas entradas están conectadas con la salida del nodo en cuestión.
- Ponderación de los vértices: Corresponde al conjunto de ponderaciones de los vértices que conectan la salida del nodo presente con la entrada de los nodos destinos para identificar las compuertas NOT.

- Nivel Máximo: Corresponde al nivel más alto del nodo cuyas entradas están conectadas con la salida del nodo presente.

De esta forma continuando con el ejemplo de la compuerta XOR, se obtiene la representación en grafo del circuito NOR/NOT de esta compuerta y la estructura de datos que almacena la información recién detallada. Lo anterior se muestra en la figura 4.8 en donde se muestra la estructura de diccionario que contiene la información de cada nodo del grafo ponderado representando a la función lógica XOR.

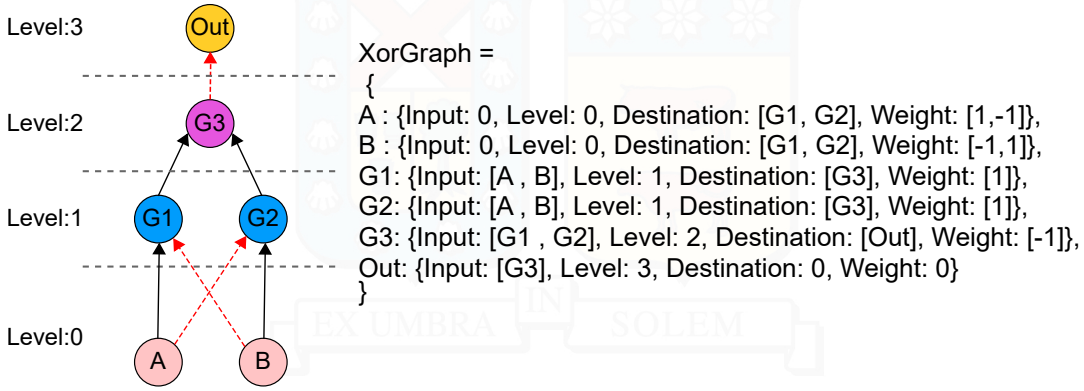


Figura 4.8: Representación en grafo del circuito lógico NOR/NOT de la compuerta XOR y su información contenida en un diccionario de Python, indicando para cada nodo lo que recibe como entrada, el nivel en el que se encuentra, los nodos con los que se conecta la salida de este y su ponderación, donde 1 corresponde a señal normal y -1 a señal negada (flechas rojas en el grafo)..

4.2.2.3. Estructura de las restricciones para la distribución de compuertas en memoria

A partir de la representación en grafo de la función lógica a sintetizar y la información más relevante de cada nodo almacenada en una estructura de datos, es necesario establecer reglas claras para mapear cada compuerta a un espacio específico en el arreglo de la memoria y determinar su tiempo de ejecución. Debido a esto se han definido parámetros que representan a cada entrada y salida de las compuertas, dentro de los cuales se ha fijado la posibilidad de tener repeticiones 'm' de cada nodo 'n', esto debido a la posibilidad de que cada salida de una compuerta sea entrada a más de una compuerta de niveles superiores. De esta forma se buscará mapear en un mismo nivel todas las repeticiones que sean entradas a niveles superiores, buscando así asegurar la ejecución en paralelo de todas las compuertas de un mismo nivel. De esta manera los parámetros de entrada y salidas se pasan a definir a continuación:

- Gn_m_R : Identifica el número de la fila de la repetición 'm' de la compuerta 'n'.
- Gn_m_C : Identifica el número de la columna de la repetición 'm' de la compuerta 'n'.
- Gn_m_T : Identifica el tiempo en el que la repetición 'm' de la compuerta 'n' se ejecuta.

Existen variaciones de esta sintaxis, esto es para el caso de salidas/entradas y compuertas negadas:

- NGn_m_R/C/T: Parámetros de la repetición 'm' de la compuerta 'n'.
- A-m_R/C/T: Parámetros de la repetición 'm' de la entrada 'A'

Bajo estas consideraciones se construyen las restricciones a utilizar durante el proceso del mapeo a la memoria, en donde se toman en cuenta factores relacionados con la topología de memoria y el estilo de lógica ocupado. De modo que, utilizando esta sintaxis para representar las entradas y salidas de las compuertas lógicas, se presentan a continuación las distintas restricciones y ecuaciones que las expresan:

- Tamaño del arreglo de memoria: La primera restricción corresponde a la que impone el tamaño del arreglo de la memoria, dado que esta área limita las celdas a las que pueden ser asignadas las compuertas.

Para un límite de filas equivalente a **Max_Rows** y de columnas equivalente a **Max_Columns** se expresa este restricción como:

$$(Gn\text{-}m\_R <= Max\_Rows) \wedge (Gn\text{-}m\_R > 0) \tag{4.6}$$

$$(Gn\text{-}m\_C <= Max\_Columns) \wedge (Gn\text{-}m\_C > 0) \tag{4.7}$$

- Ejecución de una compuerta lógica: De acuerdo a la topología de memoria y el estilo de lógica utilizado, para un correcto computo de una compuerta NOR es necesario que las entradas de esta estén alineadas en una misma fila y en diferentes columnas del arreglo. De esta forma el destino al que se asigne las compuertas debe cumplir con:

$$(Gn1\text{-}m1\_R = Gn2\text{-}m2\_R) \wedge (Gn1\text{-}m1\_C \neq Gn2\text{-}m2\_C) \tag{4.8}$$

- Ejecución en paralelo de compuertas lógicas (columnas): Para lograr la ejecución en paralelo de todas las compuertas de un mismo nivel, se debe alinear las entradas de estas compuertas en las mismas columnas asegurando cumplir también con la restricción de ejecución individual de compuertas presentada en la ecuación 4.8. Como ejemplo de esta restricción se considera el par de entradas (Gn1-m1/Gn2-m2) de la compuerta Gk1 y las entradas (Gn3-m3/Gn4-m4) de la compuerta Gk2. Ambas compuertas pertenecen al mismo nivel, de esta forma para su ejecución concurrente se plantea la siguiente restricción:

$$\begin{aligned}&\{[(Gn1\text{-}m1\_C = Gn3\text{-}m3\_C) \wedge (Gn2\text{-}m2\_C = Gn4\text{-}m4\_C)]\\ &\vee[(Gn3\text{-}m3\_C = Gn2\text{-}m2\_C) \wedge (Gn4\text{-}m4\_C = Gn1\text{-}m1\_C)]\end{aligned} \tag{4.9}$$

- Ejecución en paralelo de compuertas lógicas (Filas): De forma complementaria a la restricción anterior, para poder ejecutar dos o más compuertas en paralelo, también se debe cumplir con la condición de que las entradas de distintas compuertas sean asignadas a distintas filas. Siguiendo el ejemplo mostrado anteriormente para este caso se presenta la restricción como:

$$Gn1\text{-}m1\_R \neq Gn3\text{-}m3\_R \vee Gn2\text{-}m2\_R \neq Gn4\text{-}m4\_R \tag{4.10}$$

- Restricción de la salida: Este es el último conjunto de restricciones que se consideran, esto debido a que eventualmente durante la síntesis aparecerán nodos que corresponden a salidas lógicas que deben ser asociadas a las respectivas compuertas para evitar utilizar nuevamente estas posiciones de la memoria sobrescribir a estas salidas, perdiendo así los datos almacenados. Para esto se tienen dos restricciones, la primera hace referencia al espacio en el que debería asignarse la salida ya que para este caso se ha considerado que todas sean almacenadas en una misma columna para permitir su lectura concurrente. Para esto se considera el parámetro **OUT_Column** que define la columna asignada para las salidas. De esta forma esta primera restricción de salida se expresa para cada salida Om_C como:

$$Om\_C = OUT\_Column \tag{4.11}$$

Por otro lado, se establece una segunda restricción de salida para evitar sobrescribir los datos computados. Para esto cada compuerta Gn_m_R (distinta a la que genera la salida) debe cumplir con:

$$Om\_R \neq Gn\_m\_R \tag{4.12}$$

El proceso para generar estas restricciones se hace de manera individual para cada nivel n, comenzando desde n = 0. No obstante existe una dependencia entre cada nivel, esto principalmente para las salidas de compuertas que no se utilizan en el nivel de síntesis actual y que deben ser preservadas para niveles posteriores. Es por esto que dentro de la información relevante de cada nodo se incluye el destino de cada compuerta. Esto se considera en cada nivel agregando una restricción extra en caso de ser necesaria que impide el sobrescribir estos datos.

De esta forma, de acuerdo al flujo de síntesis, se generan para cada nivel n un conjunto de restricciones que permiten asignar espacio de memoria para las compuertas asociadas a ese nivel. Los resultados de esta distribución funcionaran de entrada para el proceso de asignación del nivel n+1 siguiente finalizando cuando no haya más niveles con compuertas para almacenar.

4.2.2.4. Resolución del problema de mapeo de compuertas lógicas acorde a restricciones

Para resolver las ecuaciones asociadas a restricciones que se plantearon en el punto anterior y obtener el valor para cada parámetro de las compuertas, se utiliza una herramienta de teorías de satisfacibilidad módulo (satisfiability modulo theories-STM). Los STM son problemas de decisión basados en formulas lógicas construidas a partir de alguna teoría en especifica, las que pueden ser arreglos, vectores de bits, tipos de datos (booleanos, enteros entre otros), etc. En este caso la herramienta utilizada corresponde al *Z3 Theorem Prover*[47] que tiene como objetivo resolver los problemas SMT. Esta herramienta fue desarrollada por Microsoft Research y es utilizada para verificación y análisis de software. Naturalmente en este trabajo el problema de decisión corresponde al de la asignación de las compuertas lógicas un espacio de memoria adecuado, para la ejecución de una función lógica a sintetizar.

El funcionamiento del Z3 en este caso se basa en la generación de un script compuesto por comandos que se agregan a un stack interno de la herramienta, a través de los cuales se construye el problema a resolver. Para este trabajo los comandos utilizados principalmente se revisan a continuación:

- (declare-const a int): Comando utilizado para declarar una variable de tipo Int entero. Las variables Int se usan para declarar las posiciones en el arreglo y tiempo de ejecución.
- (assert (> a 10)): Comando utilizado para declarar una afirmación, en este caso utilizado para establecer las restricciones del problema. En este ejemplo se declara que la constante de tipo int a, debe ser mayor a 10. También existen una serie de operaciones que se pueden expresar en una función, en particular en este trabajo se usaran: $>$, $<$, $\leq$, $\geq$, $=$, AND, OR.
- (check-sat): Este comando se utiliza para corroborar si es que las afirmaciones hechas previamente se pueden satisfacer o no. En el primer caso Z3 retorna sat, en caso contrario se obtiene unsat.
- (get-model): Finalmente en el caso que todas las afirmaciones se logren satisfacer, se puede utilizar este comando para obtener un modelo de las constantes declaradas para la cual todas las formulas son verdaderas, esto quiere decir que se obtienen los valores asignados a cada variable declarada previamente en la resolución del problema.
- (minimize t): Cuando se incluye este comando se obliga a check-sat a encontrar una solución que minimice el parámetro t.

Como se presento previamente, este proceso de síntesis se realiza de forma iterativa a través de los niveles de la función lógica. De esta forma el algoritmo de síntesis desarrollado genera un script con los comandos necesarios planteando el problema de mapeo, el cual es ingresado como entrada a la herramienta Z3 para que logre encontrar una solución a este problema de asignación de recursos de la memoria. A modo de ejemplo se presenta a continuación el script generado como entrada al Z3, para el proceso de síntesis del nivel 0 de la función XOR.

```
;## Proceso de Mapeo Nivel:0
;## Definicion de Parametros
(declare-const COLNUM Int)
(declare-const ROWNUM Int)
(assert (<= COLNUM 1000))
(assert (<= ROWNUM 1000))

;## Definicion de Variables
(declare-const NA-0_Row Int)
(declare-const NA-0_Col Int)
(declare-const NA-0_T Int)
(declare-const A-1_Row Int)
(declare-const A-1_Col Int)
(declare-const A-1_T Int)
(declare-const B-0_Row Int)
(declare-const B-0_Col Int)
(declare-const B-0_T Int)
(declare-const NB-1_Row Int)
(declare-const NB-1_Col Int)
(declare-const NB-1_T Int)
```

```
21
22 ;## Restricciones de Espacio de Memoria
23 (assert (and (> NA-0_Row 0) (<= NA-0_Row ROWNUM)))
24 (assert (and (> NA-0_Col 0) (<= NA-0_Col COLNUM)))
25 (assert (and (> A-1_Row 0) (<= A-1_Row ROWNUM)))
26 (assert (and (> A-1_Col 0) (<= A-1_Col COLNUM)))
27 (assert (and (> B-0_Row 0) (<= B-0_Row ROWNUM)))
28 (assert (and (> B-0_Col 0) (<= B-0_Col COLNUM)))
29 (assert (and (> NB-1_Row 0) (<= NB-1_Row ROWNUM)))
30 (assert (and (> NB-1_Col 0) (<= NB-1_Col COLNUM)))
31
32 ;# Restricciones de Computo de Compuerta Logica
33 (assert (and (= NA-0_Col B-0_Col) (not(= NA-0_Row B-0_Row))))
34 (assert (and (= A-1_Col NB-1_Col) (not(= A-1_Row NB-1_Row))))
35 (assert (and (= B-0_Col NA-0_Col) (not(= B-0_Row NA-0_Row))))
36 (assert (and (= NB-1_Col A-1_Col) (not(= NB-1_Row A-1_Row))))
37
38 ;## Restricciones Destino de Compuertas y Ejecuccion en Paralelo
39 (assert (not(= NA-0_Col A-1_Col)))
40 (assert (or (and (= NA-0_Row A-1_Row) (= B-0_Row NB-1_Row)) (and (= B-0_Row A-1_Row) (=
       NA-0_Row NB-1_Row))))
41
42 ;#Instante de Ejecicion
43 (assert (= NA-0_T 0))
44 (assert (= A-1_T 0))
45 (assert (= B-0_T 0))
46 (assert (= NB-1_T 0))
47
48 ;## Objetivo de Optimizacion
49 (minimize (* COLNUM ROWNUM))
50 ;## Solucion Problema
51 (check-sat)
52
53 (get-model)
```

El proceso para construir este script consiste en primero generar la lista de comando que declaran las variables. Para esto se usan el comando (delcare-const) definiendo en primer lugar las variables COLNUM y ROWNUM que representan el número de columnas y filas respectivamente, lo que se realiza en las líneas 3 y 4. En este caso, utilizando el comando (assert), el número de columnas y filas se restringe a un máximo de 1000, lo que se puede observar en las líneas 5 y 6. Tanto la definición de estos parámetros de filas y columnas, como la restricción del número de ellas, se incluyen en los script generados para cada nivel. A continuación, y utilizando nuevamente el comando (delcare-const) se definen las variables relacionadas a cada nodo presente en el nivel que se está mapeando, lo que se puede observar entre las líneas 9 y 20. Con las variables definidas se plantean ahora las restricciones del problema de mapeo. En primer lugar, se plantean las restricciones asociadas al espacio de memoria, las cuales se encuentran desde la línea 23 a la línea 30 del script. Por ejemplo, en la línea 23 se tiene la restricción (assert (and (> NA-0_Row 0) (<= NA-0_Row ROWNUM))), la cual indica que la variable asociada a la fila de la repetición 0 de la versión negada de A (NA-0_ROW) debe ser un valor mayor a 0 y un valor menor o igual a la variable ROWNUM. El siguiente conjunto de restricciones corresponden a las de cómputo de una compuerta lógica y se encuentran entre la línea 33 y 36. Para entender estas restricciones se puede observar la que se presenta en la línea 34, (assert (and (= A-1_Col NB-1_Col) (not(= A-1_Row NB-1_Row)))), la cual indica que el número de columna de la repetición 1 del nodo A y el de la versión negada de la repetición 1 del nodo B tienen que ser iguales, y a la vez el número de fila de ellos no puede ser el mismo. Posteriormente se tienen las restricciones asociadas al destino de las compuertas y su ejecución en paralelo, esto en las líneas 39 y 40. En este caso a modo de ejemplo se tiene la línea 39, (assert (not(= NA-0_Col A-1_Col))), la cual plantea que la posición de columna versión negada de la repetición 0 de la variable A y la repetición 1 de la variable A no pueden ser iguales. Por último, entre las líneas 43 y 46 se tienen los comandos que presentan los instantes de ejecución. Por ejemplo, en la línea 43 el comando (assert (= NA-0_T 0)), se lee que el tiempo en el que se calcula la versión negada de la repetición 0 de la variable A corresponde al tiempo 0. Finalmente se concluye con aquellos comandos que le indican al Z3 que busque una solución al problema y el objetivo de optimización.

Con este script se logra el mapeo de las entradas de la función lógica que se encuentran en el nivel 0.

Notar que lo que se busca con este algoritmo de síntesis es preparar el entorno en cada nivel para lograr una ejecución concurrente de las compuertas del nivel superior, de forma que se almacenan diversas copias de una misma variable y de su versión negada según sea necesario. De este modo al momento de considerar las señales a las cuales asignar posiciones en el arreglo se están tomando en cuenta el número de conexiones que cada nodo del nivel tenga con nodos de niveles superiores (es de esta característica que aparecen las repeticiones de los nodos).

Como salida de la herramienta Z3 se obtiene la solución para cada una de las variables introducidas en el script, esto se traduce a una posición en el arreglo para cada instancia de los nodos a mapear. En la figura 4.9 se muestra la evolución en el tiempo del proceso de mapeo iterativo de compuertas a un arreglo de dimensiones 2x2, para cada nivel de la función lógica XOR sintetizada. Para el nivel 0 se muestra cómo se asigna inicialmente espacio a las entradas A y B de la función XOR. Tal como se mencionó previamente se consideran las conexiones que estas tengan al momento de realizar el mapeo. Es por esto que se generan versiones negadas de estas que permiten el computo de G1 y G2. De esta forma en este nivel se asigna una celda para las variables A-1 y NB-1 que deben ir en la misma fila al ser entradas a la misma compuerta (Compuerta G1). Sucede lo mismo con NA-0 y B-0, pero para el computo de la compuerta G2. En el caso del nivel 1 solamente se tienen las compuertas G1 y G2 a mapear, al ser entradas ambas de la compuerta G3, también se asignan a la misma fila. Finalmente, en el nivel 2 se asigna la última salida de la compuerta G3, en su versión negada. Con la salida de la compuerta G3 negada asignada al espacio de memoria concluye el proceso de síntesis, pero se muestra un nivel adicional (nivel 3) en donde se identifica la celda en donde se encuentra almacenada la salida, que en este ejemplo simple permanece en la misma posición.

De esta forma, al terminar el proceso de síntesis, se obtiene el posicionamiento y el timing de todas las entradas y las salidas de cada operación lógica a realizar. Es esta información que posteriormente se procesará para interpretar la secuencia de operaciones de memoria (escrituras, lecturas, etc) que se tienen que realizar y los valores de las señales de control en la periferia del arreglo de la memoria, los cuales se podrán traducir en instrucciones para el sistema de la memoria computacional ReRAM. Cabe señalar que, de ahora en adelante, se utilizará la metodología mostrada en la Figura 5.10 para indicar la evolución del proceso de síntesis y el posicionamiento de cada señal en un arreglo crossbar, donde cada cuadrado corresponde a una celda de la memoria.

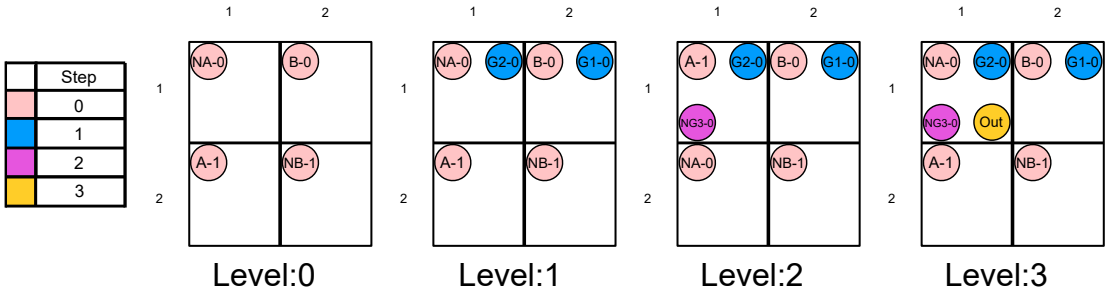


Figura 4.9: Asignación de posición en cada nivel para la ejecución de función XOR.

4.2.3. Pruebas seleccionadas de funcionamiento del sintetizador

Con el flujo de síntesis desarrollado el siguiente paso corresponde a la prueba del mismo utilizando funciones lógicas personalizadas con características distinguibles, esto para corroborar un correcto funcionamiento de la herramienta desarrollada. Para este caso se tomaron en cuenta dos diseños distintos con características notables. El primer circuito se muestra en la figura 4.10-a, este corresponde a una conexión de compuertas en forma de cadena, en la cual cada nivel posee una compuerta individual. Por otro lado en la figura 4.10-b se tiene el segundo circuito a probar, que corresponde a un diseño en el cual el número de compuertas va en incremento a medida que aumenta el número de niveles. Lo anterior se cumple hasta un nivel

intermedio desde donde el número de compuertas comienza a decrecer hasta converger en la salida.

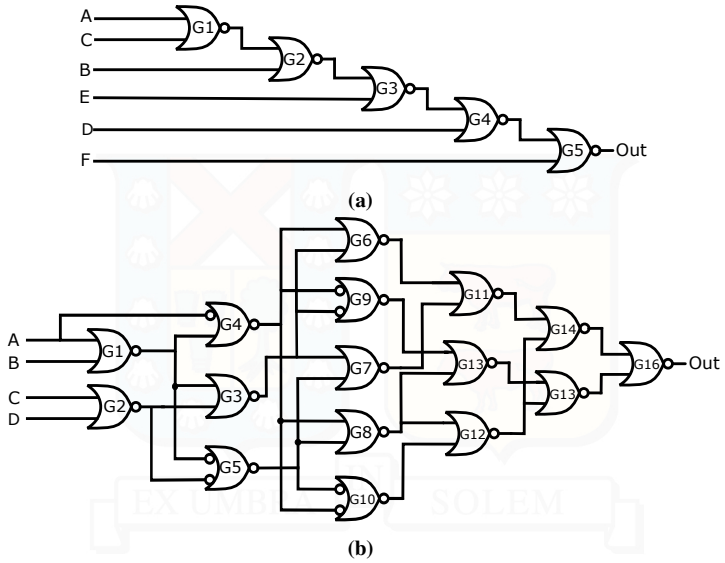


Figura 4.10: Circuitos personalizados para pruebas de flujo de síntesis. (a) Circuito con conexión de puertas en cadena. (b) Circuito con conexión de puertas en forma de árbol de ancho incremental

Son las descripciones de estos circuitos lógicos los que se sintetizan para la ejecución en un arreglo de memoria resistiva de tamaño 500x500. En primer lugar se sintetiza la función lógica correspondiente al circuito escalonado, de esto se puede observar su representación en grafo en la figura 4.11-a en donde destacan las características particulares mencionadas previamente. En la figura 4.11-b se muestra solo la sección del arreglo utilizado por el resultado de la síntesis de esta función lógica, en donde todas las entradas y salidas de las compuertas son mapeadas de acuerdo a lo esperado, es decir, todas las entradas para cada compuerta se mapean en la misma fila, y la salida de cada compuerta que se utiliza posteriormente se mapea acorde a la misma restricción. Por ejemplo, la salida de la puerta G3 (G3-0) se mapea a la misma fila con la señal D (D0) que ambas actúan como entradas a la puerta G4, cuya salida se mapea a la misma fila con la señal F0, etc. La principal característica que se destaca con esta función lógica es la necesidad de mantener las señales almacenadas lo suficiente hasta su ejecución en niveles superiores. Esto se observa con las entradas originales al sistema A-F, para las cuales solamente las variables A y C son entradas a una compuerta del nivel siguiente, mientras que el resto de las entradas se mapean, pero no puede ser sobrescritas hasta que sean utilizadas. Lo anterior es el caso de las variables F0, D0, B0 y E0 que son distribuidas en el nivel 0. Las salidas de compuertas de niveles superiores que van en conjunto con estas entradas originales obligatoriamente se van a mapear acorde a las reglas de ejecución de las compuertas, tal como ocurre en el caso de las salidas G1-0 que debe ir en conjunto con B-0 y así sucesivamente en el resto de niveles.

Se prueba también la síntesis del segundo circuito lógico con característica incremental. En la figura 4.12-a se observa el equivalente del circuito en su forma de grafo, en donde destacan las características previamente descritas, las cuales vienen a ser un diseño más complejo que en el caso anterior al tener más niveles y más compuertas por nivel. En la figura 4.12-b se muestra la sección del arreglo en donde se realiza el mapeo entregado por la herramienta de síntesis para esta descripción lógica, en la cual se observa una asignación de entradas y salidas acorde a lo esperado. En este caso este mapeo destaca la reutilización del espacio de memoria, asimismo, se puede observar que acorde a esto el tamaño máximo del arreglo, que ocupe una función lógica determinada, depende directamente del nivel de la implementación que tenga la mayor cantidad de conexiones con los niveles subsiguientes. Para este ejemplo se tiene que las conexiones de las compuertas en el nivel 2 dictan el tamaño total del arreglo. En este caso son 10 enlaces todos dirigidos al nivel siguiente. De esta forma se tiene, considerando que se está usando solo compuertas NOR de 2 entradas, que el tamaño del arreglo mínimo para la ejecución de esta función lógica es de 5x2 (Filas x Columnas). En este sentido lo anterior permite expresar la máxima área del arreglo que se utilizara, información que se puede extraer directamente de la representación en grafo por lo que se puede saber de antemano si es que la

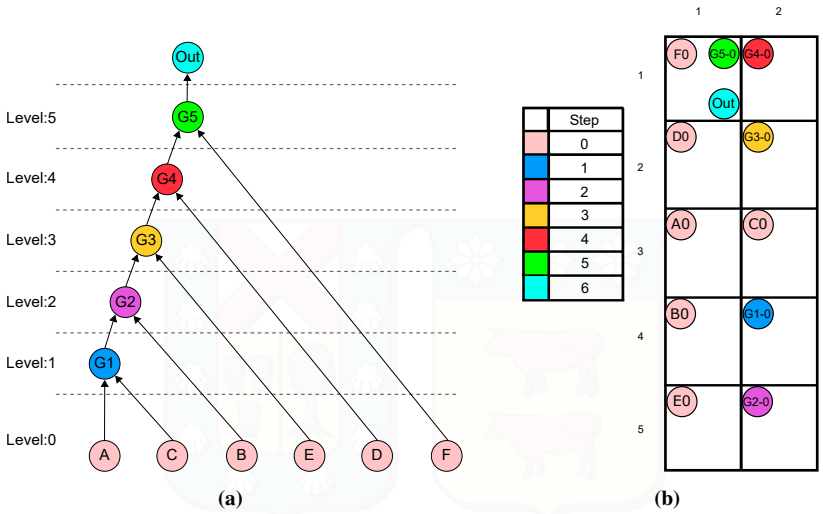


Figura 4.11: Resultado de la síntesis del circuito escalonado. (a) Representación en grafo de la descripción lógica. (b) Distribución de compuertas final de acuerdo al resultado de la síntesis

función lógica a sintetizar se puede ejecutar acorde a las restricciones del arreglo de memoria.

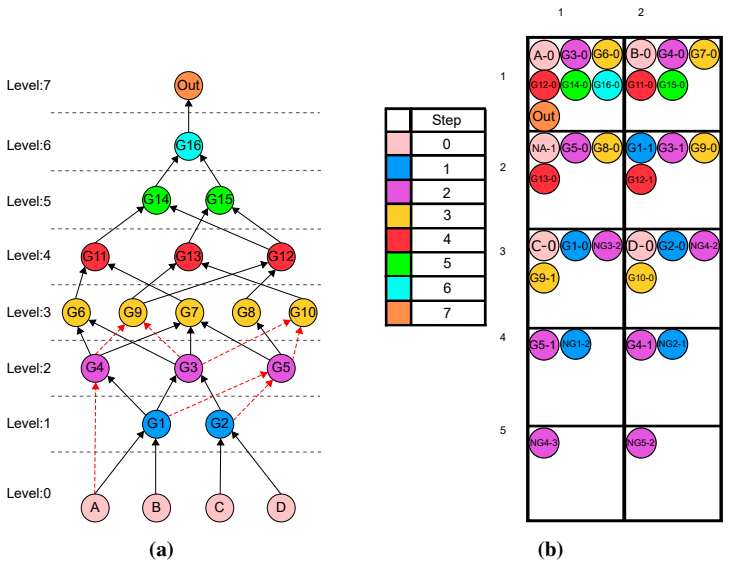


Figura 4.12: Resultado de la síntesis del circuito incremental. (a) Representación en grafo de la descripción lógica. (b) Distribución de compuertas final de acuerdo al resultado de la síntesis

Otro elemento notable que se puede observar, a partir de los ejemplos utilizados, corresponde a la latencia de ejecución del sistema. Hasta ahora consideramos que el número de pasos necesarios para esto corresponde al número de niveles del sistema. Considerando esto se tendría que entre ambas descripciones lógicas aquella que menos tiempo tarda es la función lógica implementada con compuertas en cadena, debido a que solo cuenta con 5 niveles en comparación con los 6 niveles de la implementación en árbol incremental. No obstante, vale la pena mencionar que si es que se obtiene el cociente entre el número de compuertas y numero de niveles igual a μ, la implementación incremental supera ampliamente a la escalonada, con un valor de $\mu = 2.67$ contra una de $\mu = 1$. De esto último se puede concluir que el flujo de síntesis desarrollado logra un mayor provecho de aquellas funciones lógicas con una gran cantidad de compuertas por nivel, lo que puede resultar obvio si es que se considera que se está intentando aprovechar la posibilidad de ejecución de compuertas en paralelo. De igual forma si bien adelante se presentara que el cálculo de la latencia no es

tan directo como se expresó recién, esto nos permite tener una idea del comportamiento del sintetizador y la ejecución de las compuertas, lo que se profundizara en la siguiente sección.

5 | Caracterización de la Herramienta de Síntesis

5.1. Definición de Latencia Real

Concluido el desarrollo y una vez comprobado el funcionamiento del algoritmo de síntesis, se hace necesario caracterizar su comportamiento y medir su rendimiento, esto con el objetivo de poder comparar la herramienta desarrollada con las existentes en la literatura. Previo a esto se debe precisar el mecanismo con el que el circuito de la memoria realiza el computo a partir de la salida del sintetizador, buscando con esto un cálculo exacto de la cantidad de operaciones necesarias en la memoria para la ejecución de una función lógica arbitraria. De esta forma en la sección anterior se consideró que el número de pasos necesarios para computar la función lógica sintetizada era equivalente al número de niveles de esta. Esto no es del todo correcto debido a que con esta suposición se está tomando en cuenta solo las operaciones asociadas al cómputo en paralelo de las compuertas lógicas, dejando de esta forma afuera a las operaciones asociadas al movimiento de datos en los procesos de escritura, y en general la influencia de la propia topología de la memoria resistiva y su circuitería periférica.

Para entender el impacto de estas operaciones en el desempeño de la memoria computacional se considera el siguiente ejemplo. Asumimos que en un momento T = n la situación en el arreglo de memoria es la que se muestra en la figura 5.1, en este ciclo precisamente se requiere computar las compuertas G2 = ($\overline{A1}$ NOR $\overline{B1}$) y G1 = (A0 NOR B0). De esta forma se requiere en el siguiente ciclo T = n+1 que las salidas G1 y G2 queden almacenadas como se muestran en la figura, para esto se requieren operaciones intermedias que permitan llevar el arreglo a este estado.

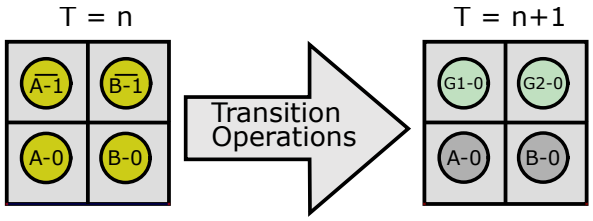


Figura 5.1: Ejemplo de una transición supuesta en la disposición de la memoria en un espacio de tamaño 2x2.

Estas operaciones intermedias que se mencionan hacen referencia a la manipulación de las salidas posterior a cada computo en paralelo, de forma de llevar la distribución de los datos en el nivel siguiente según lo que el sintetizador indica. Lo anterior se explica de mejor forma si es que se considera la topología de memoria utilizada, por esto se presenta en la figura 5.2-a una versión simplificada del circuito de memoria y la distribución de los datos en ella en el tiempo T = n. En este instante T = n se realiza el computo en paralelo de G1 y G2 cuyas salidas lógicas quedan almacenadas temporalmente en el buffer (no se muestra en la figura) en la circuitería periférica de la misma línea del arreglo en forma de voltaje. Estos resultados deben escribirse en el arreglo según lo indica el mapeo realizado por el sintetizador, que para este ejemplo corresponde a la misma fila, más específicamente las posiciones (1,1) y (1,2) del arreglo (según lo que se

muestra en la figura 5.1 para G1-0 y G2-0 en el tiempo T = n+1). Como se necesita escribir información a celdas de la memoria que se encuentran en la misma fila, dado que todos los *memristor* tienen su electrodo superior conectado al mismo driver de escritura, la topología no permite escribir simultáneamente G1 y G2 en un mismo paso. Producto de lo anterior, si bien las señales G1 y G2 se obtuvieron en paralelo y están disponibles en la periferia, se necesitan 2 procesos de escritura independientes y consecutivos para almacenar estos datos en las posiciones correctas en la memoria. En consecuencia, primero en el instante $T = t$ la señal G2-0 actúa como señal de control en el driver de la primera fila y se activa WL_2 para producir un proceso de SET/RESET al *memristor* que se encuentra en la posición (1,2) del arreglo, tal como se muestra en la figura 5.2-b. Asimismo, en el siguiente paso $T = t + 1$ ocurre un proceso similar que produce la escritura de G1-0 al *memristor* en la posición (1,1) del arreglo, según se muestra en la figura 5.2-c. Con los datos almacenados en la posición adecuada, se puede computar el resultado de la compuerta NOR según muestra la figura 5.2-d, cuyo resultado se almacena en el buffer de la primera fila del arreglo.

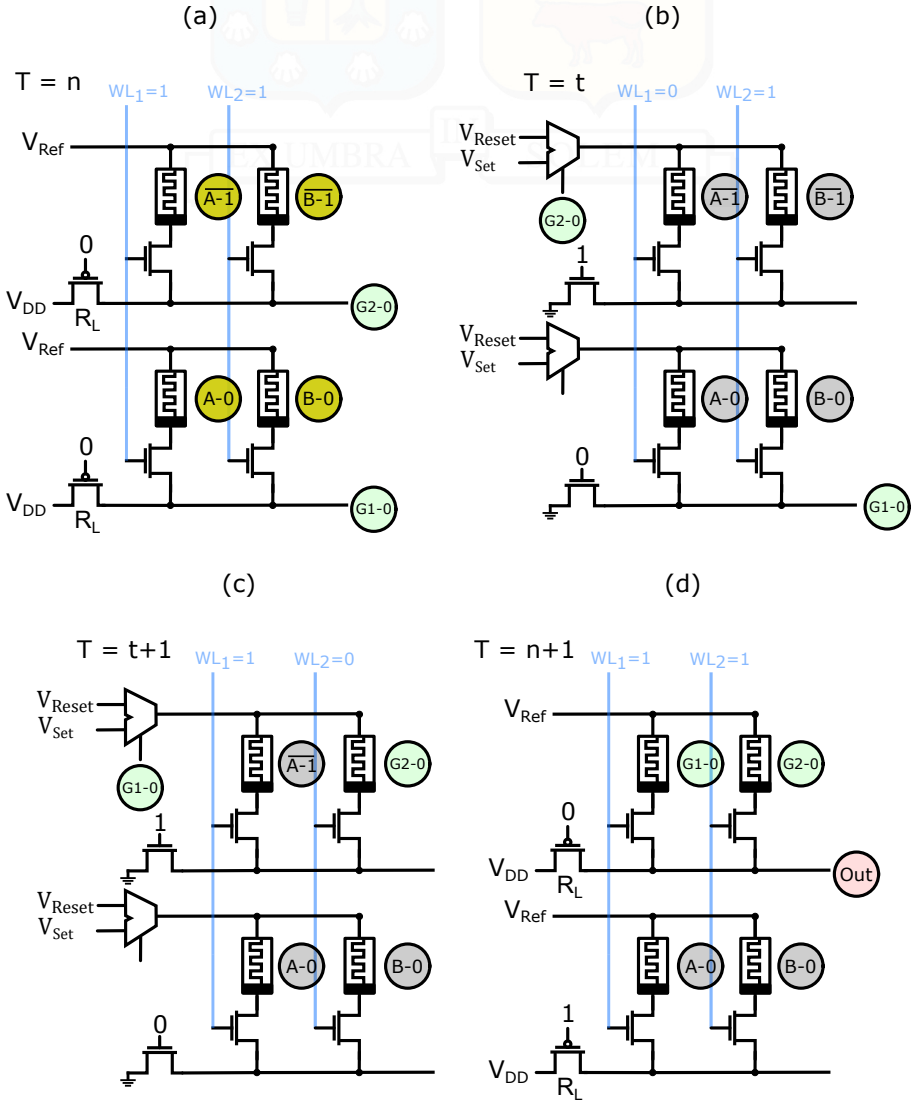


Figura 5.2: Manejo de salidas con operaciones intermedias en el arreglo. (a) Computo de compuertas NOR2 G2 y G1 a partir de una distribución inicial de los datos en la memoria en un tiempo T = t. (b) Proceso de escritura de la salida de G2 en el tiempo T = t. (c) Proceso de escritura de la salida G1 en el tiempo T = t+1. (d) Distribución final alcanzada acorde a la salida del sintetizador, también se computa la compuerta que entrega la salida Out en el tiempo T = n+1

A partir de este ejemplo queda claro que hacen falta operaciones intermedias para preparar el entorno en la ejecución de las compuertas lógicas. Dado que se considera trabajar con compuertas NOR de 2 entradas,

como mucho se necesitará activar 2 columnas para realizar dos escrituras de datos consecutivas, más un tercer paso para realizar el cómputo de la compuerta. Lo anterior resulta en la formula $L_t = 1 + c$ para la latencia real expresada en número de operaciones de memoria, donde c es igual al máximo número de entradas de las puertas NOR que en este caso es igual a 2.

5.2. Simulación de la Salida del Sintetizador

Habiendo precisado el desempeño que se logra en la ejecución de las funciones lógicas a partir de la herramienta de síntesis desarrollada, ahora se plantea la simulación de una función lógica arbitraria acorde a la salida que se obtenga de esta herramienta. Lo anterior se plantea con el objetivo de validar la factibilidad de las operaciones de memoria que conducen a la ejecución correcta de la función lógica. Para este caso la función seleccionada corresponde a la de un Full-Adder de 1 bit, cuyo circuito lógico en términos de compuertas NOR/NOT se muestra en la figura 5.3-a. Tras el proceso de síntesis de esta función lógica se obtiene la representación en grafo que se observa en la figura 5.3-b, mientras que la evolución de la ejecución de las compuertas con el posicionamiento y timing de los datos obtenida se muestra en la figura 5.3-c.

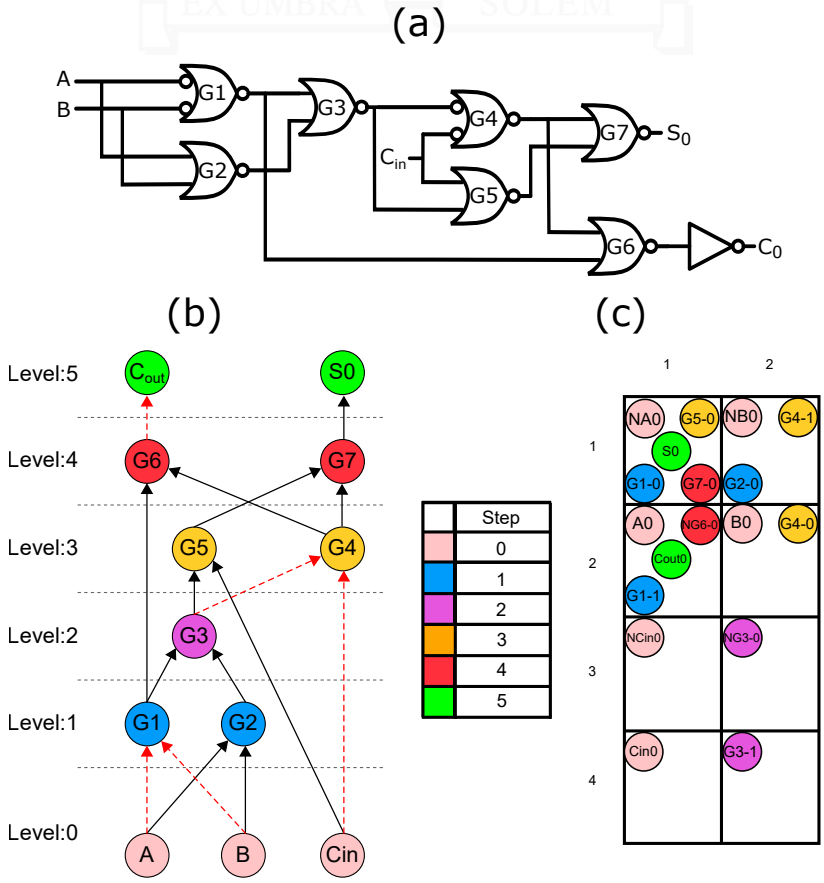


Figura 5.3: Proceso de síntesis circuito Full-Adder 1 bit. (a) Circuito lógico en compuertas NOR/NOT de un Full-Adder de 1 bit. (b) Expresión en grafo del circuito lógico de un Full-Adder de 1 bit. (c) Distribución de compuertas Full-Adder de 1 bit acorde al flujo de síntesis

Como se observa en las figuras mostradas anteriormente, esta función lógica posee 6 niveles, de los cuales cuatro corresponden a operaciones de computación en paralelo, siendo el nivel 4 utilizado para almacenar las salidas y el nivel 5 solamente referencia para la salida. Con respecto al resto de operaciones de escritura se debe considerar el número de columnas activadas en cada nivel. En la tabla 5.1 se muestra la

cantidad de columnas que deben ser utilizadas, de esta forma se obtiene que el número de operaciones de escritura corresponden a 8. En total la ejecución de esta función lógica en un arreglo de memoria tiene una latencia de 12 operaciones, 8 de cómputo y 8 de escritura.

Nivel	0	1	2	3	4	5
# Columnas	2	2	1	2	1	0

Tabla 5.1: Numero de columnas utilizadas por nivel

Con esta información se procede a simular la ejecución de esta función lógica de acuerdo a las disposiciones entregadas por el sintetizador. Esta simulación se realiza tomando en consideración el circuito de memoria presentado en la figura 3.11, simulando una topología de dimensiones 4x2 donde los *memristors* se describen con el del modelo VTEAM[31] acorde a los parámetros mostrados en la sección *Modelo VTEAM* y utilizando la herramienta de simulación LTspice y modelos de dispositivos como se describió en la sección *Pruebas de computación lógica y de memoria*. El flujo de operaciones necesarias para la ejecución de esta función lógica en la simulación se puede observar en la tabla 5.2 en donde se destaca la operación realizada, las columnas involucradas, las filas involucradas y el ciclo en el que esta ocurre. Por ejemplo, en el ciclo 0 se tiene la lectura de las entradas a la función lógica, las cuales se encuentran en las filas 1 a 3 y en la columna 1; posteriormente en el ciclo 1 se escribe en el arreglo NA-0, A-1, NCin-0 y Cin-1, para esto se utilizan las filas 1 a 4 y la columna 1; así sucesivamente con el resto de los ciclos. Cada operación también está asociada a un nivel especifico, en donde los ciclos 1 a 3 están asociados con el nivel 0, los ciclos 4 a 6 con el nivel 1, los ciclos 7 y 8 con el nivel 2, los ciclos 9 a 11 con el nivel 3, y el ciclo 12 con el nivel 4. En donde también cada nivel se identifica con un color especifico que servirá para seguir la simulación posteriormente.

Cycle	Operation	Row	Column
0	Read Input	1-3	1
1	Write NA-0/A-1/NCin-0/Cin-1	1-4	1
2	Write NB0/B1	1-2	2
3	Paralel NOR2 Level 1	1-2	1-2
4	Write G1-0/G1-1	1-2	1
5	Write G2-0	1	2
6	Paralel NOR2 Level 2	1	1-2
7	Write NG3-0/G3-1	3-4	2
8	Paralel NOR2 Level 3	3-4	1-2
9	Write G5-0	1	1
10	Write G4-1/G4-0	1-2	2
11	Paralel NOR2 Level 4	1-2	1-2
12	Write G7-0/NG6-0	1-2	1
Output	Level 5: Output	-	-

Tabla 5.2: Operaciones por ciclo y posiciones del arreglo involucrados. Colores identifican cada nivel en donde el color de los ciclos del 1 al 3 identifica al nivel 0, el de los ciclos 4 a 6 al nivel 1 y así sucesivamente.

La ejecución de las distintas operaciones para realizar estas simulaciones se pueden observar en la evolución de las señales de control y estados internos de las celdas de memoria que se presentan en la figura 5.4. En la figura anterior se encuentran separados por colores los instantes de tiempo en los que se están ejecutando las operaciones asociadas a cada nivel de la memoria, a su vez se delimita la duración de cada ciclo mediante una línea gris punteada.

Como se mencionó antes en esta simulación se computa una función Full-Adder de 1 bit, para este se asumen que en un instante inicial las entradas A = 1, B = 0 y Cin = 0 se encuentran almacenadas en las celdas Rm(1,1), Rm(2,1) y Rm(3,1) respectivamente. En la figura 5.4-a se puede corroborar el estado resistivo de estas celdas para todos los instantes. En el tiempo cero se tiene que Rm(1,1) está en baja resistencia pues

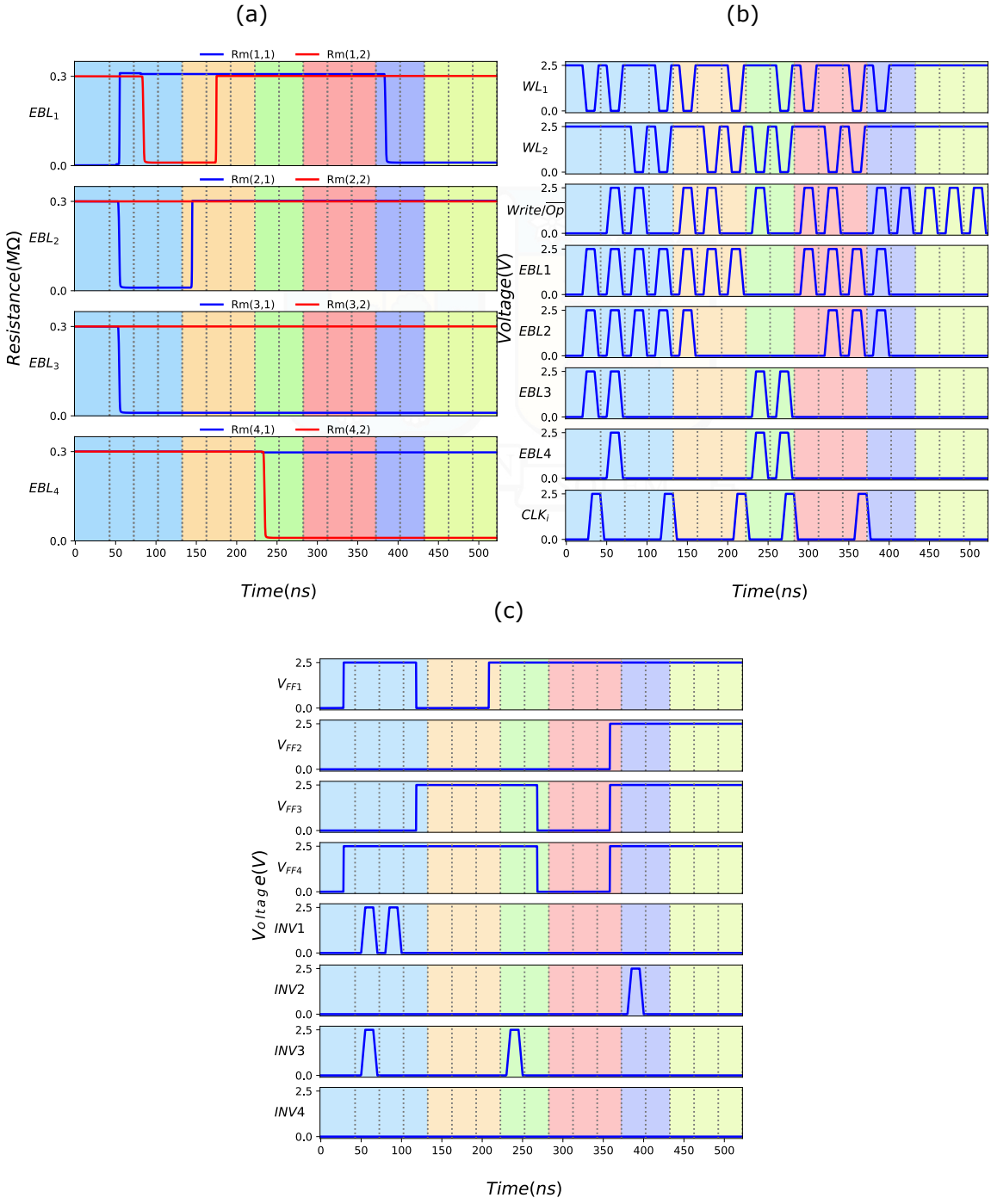


Figura 5.4: Señales y estados de la memoria durante simulación computación Full-Adder de 1 bit. (a) Evolución de la resistencia de los dispositivos en el tiempo. (b) Señales de aplicadas al circuito de control de la memoria. (c) Señales de interés y de control segundarías. Cada color está asociado a un nivel de la función ejecutada, esto acorde a los colores de la tabla 5.2

almacena A = 1, al contrario de los dispositivos Rm(2,1) y Rm(3,1) que se encuentran en alta resistencia al almacenar B = Cin = 0. Acorde a la tabla de verdad este combinación de entradas entregaría las salidas S_0 = 1 y C_{out} = 0, las cuales según el mapeo realizado por el sintetizador se pueden encontrar a en las posiciones Rm(1,1) y Rm(2,1). Lo anterior se corrobora al observar la evolución de las resistencias de los dispositivos

indicados, acabando Rm(1,1) en baja resistencia y Rm(2,1) en alta resistencia como era de esperarse.

Con esta simulación se puede comprobar en primer lugar la capacidad del arreglo de memoria y sus circuitos de control de poder ejecutar una función lógica a partir de la salida del sintetizador. En segundo lugar, también se corrobora el número real de operaciones necesarias para la ejecución de una función lógica, esto acorde a las consideraciones realizadas previamente acerca de las operaciones intermedias, en donde se llegó a que el Full-Adder de 1 bit necesitaba de 12 operaciones para su ejecución, lo que se comprueba al observar la tabla. Vale la pena mencionar que en este último caso se debe descontar la operación del ciclo 0 que corresponde a un ciclo en el que se leen las entradas originales, cuyas posiciones en este caso se eligieron arbitrariamente.

5.3. Comparación del Desempeño del Sintetizador con el Estado del Arte

Para evaluar el desempeño del sintetizador y poder comparar este con el desempeño de otras herramientas existentes en la literatura, se utiliza un conjunto de descripciones lógicas o Benchmarks que se referencia en otros trabajos. Estos Benchmarks consisten en un subconjunto de descripciones de circuitos lógicos obtenidos de LGsynth91[50]. Este conjunto de benchmarks fue creado por investigadores la Universidad de California con el objetivo de entregar a otros investigadores la posibilidad de evaluar sus técnicas de optimización. En esta ocasión son utilizadas por SIMPLE-MAGIC, SAID entre otras herramientas para obtener métricas que permiten medir el rendimiento de los sintetizadores, entre las cuales comúnmente las más importantes se consideran el área, número de *memristors* utilizados dentro del arreglo de la memoria y latencia (expresada en número total de operaciones).

La implementación de estas pruebas se realizará en un computador Intel Core i7-4510U CPU @ 2.00GHz con 16GB de memoria RAM en el sistema operativo Ubuntu. También se debe mencionar que las pruebas se realizaron configurando al sintetizador para encontrar una solución sub-óptima al problema de asignación de compuertas en el arreglo dentro de un área determinada, esto con el objetivo de lograr una ejecución eficiente en términos de tiempos de síntesis. Con esto en consideración se obtienen los datos presentados en la tabla 5.3. En esta tabla se muestran el número de niveles, número de operaciones de escritura, numero de filas utilizadas, número de operaciones totales, numero de *memristors* utilizados y área total necesaria. Además, se presenta el desempeño de la herramienta SIMPLE MAGIC y la comparación entre ambas. Como se puede observar en los resultados de comparación el flujo de síntesis propuesto logra una mejora considerable en los distintos aspectos.

		Proposed Tool						Simple Magic[46]			Comparison		
BM	Gates	Lvl	Write	#Row	Op	Mem	Area	Op	Mem	Area	Op	Area	Mem
5xp1	112	9	19	50	28	99	100	97	142	315	71.13 %	30.28 %	68.25 %
CLIP	152	9	19	57	28	112	114	136	185	444	79.41 %	39.46 %	74.32 %
misex1	78	6	13	28	19	56	56	45	112	294	57.78 %	50.00 %	80.95 %
cm150a	62	9	15	31	24	62	62	51	87	189	52.94 %	28.74 %	67.20 %
parity	76	8	17	16	25	32	32	37	107	240	32.43 %	70.09 %	86.67 %
x2	68	6	13	27	19	52	54	36	86	168	47.22 %	39.53 %	67.86 %
cm162a	60	8	17	21	25	42	42	46	92	186	45.65 %	54.35 %	77.42 %
cm163a	61	7	15	26	22	51	52	45	95	183	51.11 %	46.32 %	71.58 %

Tabla 5.3: Desempeño flujo de síntesis propuesto y comparación con SIMPLE MAGIC utilizando conjunto LGsynth91

Para medir la comparación en el desempeño de latencia se obtiene el cociente $\frac{\#OP_{SIMPLE-MAGIC}-\#OP_{propuesto}}{\#OP_{SIMPLE-MAGIC}}$, de esta forma en la tabla se presenta el porcentaje de mejora en número de operaciones de la herramienta propuesta respecto a la herramienta SIMPLE-MAGIC. En este sentido el método propuesto logra una mejora significativa respecto a esta herramienta con la que se compara, logrando como máximo una mejora de 79.41 % y una mejora mínima de 32.43 %. El máximo de incremento en desempeño obtenido respecto a la herramienta SIMPLE MAGIC se obtiene con la función lógica CLIP, lo que coincide con que esta sea

la función lógica con mayor número de compuertas y el mayor número de compuertas promedio por nivel (Gates/LVl). Lo anterior se explica debido a que el algoritmo propuesto calcula en paralelo el 100 % de las compuertas en un mismo nivel, esto a diferencia de lo que ocurre en SIMPLE MAGIC, obteniendo así una considerable mejora en comparación. Por otro lado, también se logra una mejoría producto de no tener que calcular las compuertas NOT, en este caso para la descripción CLIP estas alcanzaban aproximadamente el 30 % del total de compuertas, las que se calculan en la periferia del arreglo al momento de escribir las salidas de estas compuertas. Si bien no es un comportamiento general, se puede argumentar que el sintetizador obtiene el mejor desempeño respecto a SIMPLE MAGIC en aquellas funciones con una alta cantidad de compuertas promedio por nivel, en donde también se debe considerar el porcentaje de operaciones NOT ahorradas del total de operaciones de la función sintetizada.

	Proposed Tool			SAID		Comparison	
Benchmarks	Levels	Op	Mem	Op	Mem	Op	Mem
5xp1.blif	7	22	121	106	251	79.25 %	51.79 %
9sym.blif	10	30	246	160	1026	81.25 %	76.02 %
b12.blif	9	28	246	26	283	-7.69 %	13.07 %
clip.blif	9	28	306	135	451	79.26 %	32.15 %
con1.blif	4	13	24	5	52	-160.00 %	53.85 %
duke2.blif	9	28	474	300	1632	90.67 %	70.96 %
e64.blif	7	22	639	134	394	83.58 %	-62.18 %
misex1.blif	5	16	44	17	184	5.88 %	76.09 %
misex2.blif	6	18	104	30	125	40.00 %	16.80 %
rd53.blif	7	22	52	31	83	29.03 %	37.35 %
rd73.blif	10	30	230	150	379	80.00 %	39.31 %
rd84.blif	12	36	374	252	745	85.71 %	49.80 %
sao2.blif	9	28	214	79	559	64.56 %	61.72 %
squar5.blif	6	18	73	15	98	-20.00 %	25.51 %
table3.blif	12	37	514	1766	2242	97.90 %	77.07 %
table5.blif	12	36	418	1169	2009	96.92 %	79.19 %
vg2.blif	9	27	190	55	280	50.91 %	32.14 %
xor5.blif	6	18	12	12	35	-50.00 %	65.71 %

Tabla 5.4: Desempeño flujo de síntesis propuesto y comparación con SAID utilizando conjunto IWLS'93

Se compara también el área efectiva utilizada y el número de *memristors* que se necesitan entre la herramienta desarrollada y la presentada por SIMPLE-MAGIC, para estos casos se obtiene una taza de mejora dada por $\frac{Area_{SIMPLE-MAGIC}-Area_{Propuesto}}{Area_{SIMPLE-MAGIC}}$ y $\frac{\#Mem_{SIMPLE-MAGIC}-\#Mem_{Propuesto}}{\#Mem_{SIMPLE-MAGIC}}$. Con respecto al área utilizada el algoritmo propuesto logra una disminución de area de un máximo de 70.09 % respecto a SIMPLE-MAGIC y un máximo de 86.67 % de disminución en los *memristors* utilizados. Esta mejora en la cantidad de área utilizada y el número de *memristors* se asocia a dos factores. En primer lugar, el estilo de lógica utilizado por la herramienta SIMPLE-MAGIC corresponde al estilo MAGIC, el cual precisa de tres dispositivos para llevar a cabo cada operación NOR de dos entradas y dos dispositivos para computar la compuerta NOT. Lo anterior se produce debido a que este estilo considera un dispositivo en donde se obtiene la salida mediante el cambio de su resistencia, esto aparte de los dispositivos que se usan para almacenar las entradas. Lo anterior difiere del estilo "Ratioed Logic" en donde la salida es una señal de voltaje la que puede ser almacenada en cualquier dispositivo. En segundo lugar, la herramienta SIMPLE-MAGIC no considera la reescritura de dispositivos que ya se han utilizado para almacenar entradas como si lo considera la herramienta del presente trabajo, esto permite compactar el área que se está usando, es por este motivo que el número de *memristors* utilizados es un valor igual o muy cercano al que corresponde al área utilizada.

También se realiza una comparación con la herramienta SAID[49], esta vez se utiliza un conjunto de descripciones obtenidas de IWLS'93[51]. Esta comparación se presenta en la tabla 5.4, en donde se toma en cuenta la latencia y el número de *memristors* y se comparan utilizando las ecuaciones $\frac{\#OP_{SAID}-\#OP_{propuesto}}{\#OP_{SAID}}$ y

$\frac{\#Mem_{SAID}-\#Mem_{Propuesto}}{\#Mem_{SAID}}$ respectivamente. Esta comparación, a diferencia de lo sucedía con la anterior, muestra que la herramienta desarrollada no siempre obtiene una ventaja comparativa respecto de la herramienta SAID. La razón de lo anterior puede deberse a dos factores, en primer lugar, se puede tener un bajo promedio de compuertas por nivel, de forma que el proceso de escritura se vuelve predominante en relación a las compuertas que se quiere computar en paralelo. Es el caso de lo anterior para las funciones con1, squar5 y xor5 para los cuales la herramienta desarrollada obtiene los peores resultados. El segundo factor relevante corresponde a la elección de parámetros que realiza la herramienta SAID, para la cual se debe tomar en cuente que su funcionamiento se basa en el mapeo de super-compuertas como se describió en el capítulo anterior. Estas super compuertas tienen una cantidad fija de entradas y mientras mayor sea la cantidad de entradas para cada super-compuerta, menos de estas se tendrán lo que trae consigo una disminución en el número de operaciones que se deben realizar. Este es el caso del benchmark b12 el cual en el trabajo presentado para la herramienta SAID se consideró un número de entradas igual a 10 (el máximo utilizado en ese trabajo). Estos factores se pueden combinar para el resto de los benchmarks por lo que puede explicar la disparidad en la mejora o no del desempeño en términos de latencia. Por otro lado el trabajo presentado en [49] no explica en detalle que sucede con las operaciones de inicialización de las entradas a las distintas súper-compuertas, en este sentido se puede pensar que este proceso podría degradar aún más el desempeño de este trabajo.

Por último, respecto a la comparación del número de *memristors* utilizados en cada benchmark se puede observar también una variabilidad en los resultados. En este sentido el factor más probable para la obtención de buenos resultados en casi todas las funciones probadas corresponde al de la reutilización de dispositivos. Por otro lado, la variabilidad obtenida puede atribuirse al uso de diferentes números de entradas para las súper-compuertas como ya se comentó previamente.

5.4. Caracterización de la latencia

Posterior a la comparación realizada entre el flujo de síntesis desarrollado y otros existentes en la literatura, se pasa a caracterizar la latencia a partir de la prueba de un conjunto mayor de descripciones y con un mayor rango de características en termino de numero de compuertas que componen estas descripciones. Para esto se utiliza un conjunto de 85 descripciones lógicas obtenidas de LGsynth91[50], de las cuales se obtienen variables de interés como, número de operaciones, numero de compuertas NOR2 y numero de niveles.

En un primer caso se gráfica el número de operaciones de memoria totales realizadas, en función de la cantidad de compuertas NOR2 en el diseño, y los resultados se muestran en la figura 5.5-a. Se puede observar que no existe una relación entre estas dos variables, de esta forma se puede concluir que el número de operaciones es independiente del número de compuertas NOR en la función. Por otro lado en la figura 5.5-b se gráfica la relación entre el número de niveles de la descripción lógica y el número de operaciones. En este caso se hace evidente que la relación entre ambas variables es lineal, es más se puede obtener la ecuación de la recta que modela este crecimiento, la cual se presenta en la imagen. La pendiente de esta recta, $\beta = 2{,}92$ está estrechamente relacionada con el máximo número de operaciones por nivel que se presentó anteriormente, el cual era de $L_n = 3$.

A partir de estos resultados se puede concluir que el diseño de flujo de síntesis y posterior ejecución en el arreglo diseñado en este trabajo logra su mejor desempeño cuando se tienen funciones lógicas que presentan un bajo número de niveles con una gran concentración de compuertas lógicas. Lo anterior es evidente si se considera que el diseño propuesto busca aprovechar al máximo la ejecución en paralelo de las compuertas presentes en un nivel. No obstante, estas pruebas toman en consideración la disponibilidad total de un conjunto de *memristors* en un espacio de dimensiones m x 2 dentro del arreglo de memoria para la ejecución de la función lógica, lo que no siempre puede ser cierto, dado que, algunos *memristors* en este espacio podrían mantener datos almacenados y por lo tanto no deberían participar en un proceso de cómputo. En consecuencia, más adelante se observará como la ocupación del arreglo puede tener un impacto en la latencia de cómputo de funciones lógicas, esto en un arreglo de memoria acorde al trabajo desarrollado.

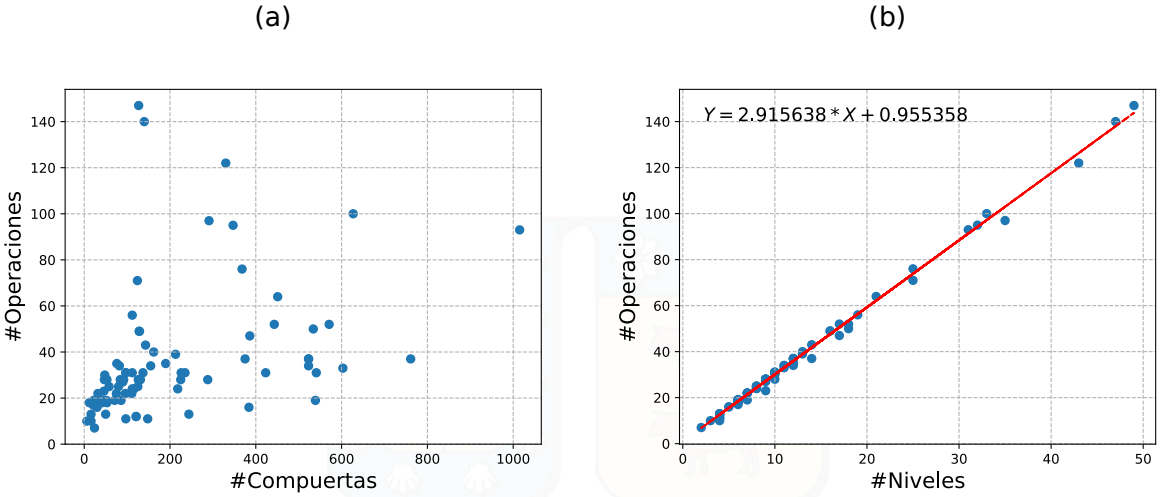


Figura 5.5: Pruebas de caracterización de la latencia del flujo de síntesis desarrollado. (a) Gráfico #Operaciones v/s #Compuertas(NOR2) . (b) Gráficos #Operaciones v/s #Niveles

5.5. Caracterización Tiempo de Síntesis

Otro factor a tomar en cuenta al momento de entender el comportamiento del sintetizador desarrollado corresponde al tiempo que toma el proceso de síntesis en completarse. Herramientas como estas basadas en proceso de optimización del problema de mapeo pueden generar tiempos poco realistas a medida que el número de compuertas y el espacio del arreglo aumenta, dado que aumenta el espacio de posibles soluciones a explorar. Esto precisamente se destaca en la información presentada en la Tabla 5.5 que consiste en datos obtenidos de [48], en donde se muestran tiempos mayores a 1 semana para un diseño de 152 puertas lógicas, lo que resulta un problema en la práctica. En el caso de la herramienta desarrollada se busca reducir el efecto

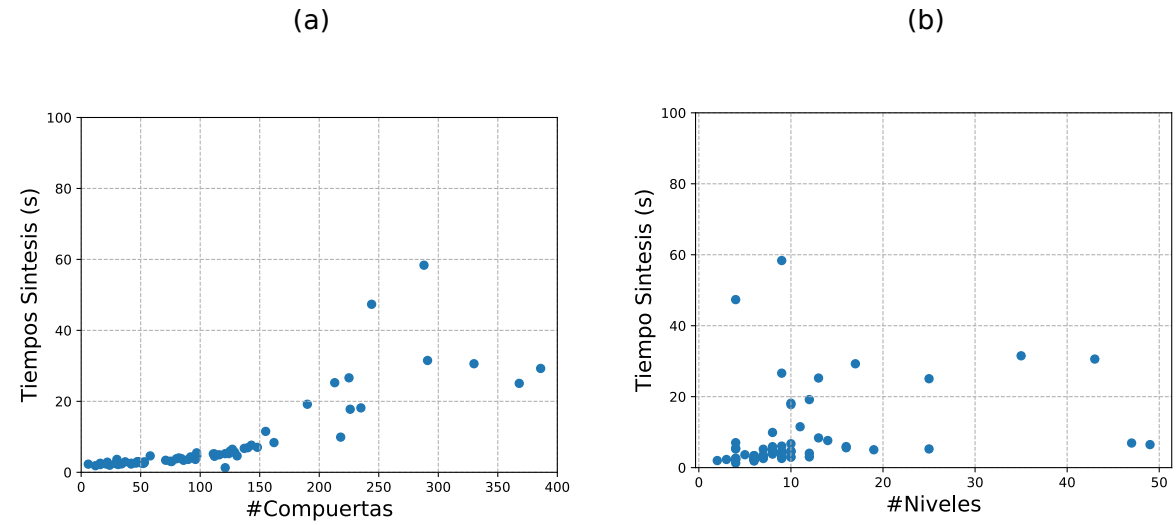


Figura 5.6: Pruebas de caracterización del tiempo de síntesis del flujo de síntesis desarrollado. (a) Gráfico Tiempo de síntesis v/s #Compuertas(NOR2) . (b) Gráfico Tiempo de síntesis v/s #Niveles

de esta problemática a través de dos mecanismos. En primer lugar, el algoritmo desarrollado toma un enfoque iterativo, de manera que la asignación de compuertas se realiza de manera independiente para cada nivel, con excepción de aquellas que dependan de posiciones de compuertas anteriores. Esto ayuda a reducir el número de compuertas que efectivamente se están intentando mapear en un tiempo determinado, lo que naturalmente

tiene un efecto directo en los tiempos de síntesis. En segundo lugar, durante el proceso de mapeo el problema se afronta buscando una solución sub-óptima en términos del área utilizada, a fin de encontrar una solución que logre distribuir las compuertas según las restricciones impuestas, sin la necesidad de contar con un uso óptimo del espacio. De igual forma se logra una mejora en el uso del espacio mediante otros medios, como lo es el limitar el espacio de arreglo usado al momento de definir las restricciones de mapeo.

#Compuertas	Tiempo de Síntesis	Tamaño del Arreglo
11	83	8x8
13	83s	8x8
14	230s	8x8
18	230s	8x8
21	215s	8x8
35	1696	8x16
152	>1 semana	24x24

Tabla 5.5: Tiempos de síntesis de la herramienta general sin acelerar [48]

De esta forma se obtienen los tiempos de síntesis para las 85 descripciones lógicas del sub conjunto Synth91. En la figura 5.6-a se muestra el tiempo de síntesis en comparación con el número de compuertas NOR de las descripciones. Por otro lado en la figura 5.6-b se muestran los tiempos de síntesis comparados con el número de niveles de cada descripción. Como se puede observar de los gráficos, la variable predominante que determina el tiempo de síntesis corresponde al número de compuertas de la función, mientras que el número de niveles no parece tener una influencia directa en este factor. Es importante señalar, además, que los resultados son tiempos de síntesis bajos si se comparan con los presentado en [48], como se muestra en la tabla 5.5.

De igual forma la herramienta desarrollada en [48] enfrenta este problema de tiempos de síntesis a través de un algoritmo que permite acelerar el proceso en desmedro de la latencia de ejecución, por lo que se realiza también una comparación de los tiempos de síntesis de la herramienta desarrollada en este trabajo con los presentados en la versión acelerada de [48]. Para esta comparación se utilizan 5 descripciones lógicas y los resultados se pueden observar en la tabla 5.6 muestran que la herramienta de síntesis desarrollada logra mejores tiempos de síntesis, de como mínimo 40 veces menos.

	General Framework[48]			Propuesto		
Benchmarks	#Op	Area	Tiempo (s)	#Op	Area	Tiempo (s)
5xp1	189	256	475.03	28	100	3.88
Clip	237	324	954.32	28	114	2.84
misex1	94	144	821.34	19	46	2.56
parity	69	144	67.45	25	32	1.69
x2	91	144	273.21	19	54	1.61

Tabla 5.6: Tiempos de síntesis de la herramienta general acelerada y herramienta propuesta

5.6. Limitación de Espacio y Latencia de Computo

Hasta ahora el mapeo al arreglo se había realizado considerando la posibilidad de explorar sin restricciones el espacio de memoria, pero esto no siempre puede ser el caso. De esta forma en la presente sección se estudiará el impacto de tener celdas ocupadas que impidan un mapeo normal de las compuertas de una determinada función acorde a las restricciones presentadas previamente, es decir, el impacto que tiene la imposibilidad de contar con un espacio m x 2 limpio de datos dentro del arreglo de la memoria, para destinarlo al proceso de cómputo.

De esta forma a modo de ejemplo se considera el caso hipotético en el que es necesario mapear las entradas de las compuertas del nivel 1 que se muestran en la figura 5.7-a correspondientes al circuito lógico de la figura 5.7-c. La diferencia en esta ocasión en comparación con otros proceso de síntesis, es que ahora se considera un espacio de memoria disponible de dimensiones 4x4 donde algunas de las celdas están ocupadas tal como se muestra en la figura 5.7-b. Se consideraron que en esta memoria se encuentran almacenadas palabras de 2-bit, y de manera arbitraria se consideró que se encontraban almacenadas 4 de estas palabras en la memoria (1 por columna).

Se debe recordar que originalmente se apuntaba a obtener el máximo provecho a la ejecución en paralelo de las compuertas lógicas de un mismo nivel. Es por esto que en las restricciones de mapeo se incluía una restricción orientada a esto. En este ejemplo en particular se quiere computar las compuertas G1,G2 y G3, y por eso se deben mapear en el arreglo las entradas A, B, C, D. Para ejecutar las tres compuertas, se tienen tres pares de entradas, dadas por (A-0, B-0), (B-1, C-0) y (D-0, C-1), para las puertas G1, G2 y G3, respectivamente. Asimismo, la fila R y columna C donde están asignadas se expresan como [(R1a,C1a),(R1b, C1b)], [(R2a,C2a),(R2b, C2b)] y [(R3a,C3a),(R3b, C3b)]. Lo ideal es que cada entrada sea mapeada acorde a la restricción de que los pares de entrada coincidan en la posición de la columna (presentada en la ecuación 4.9), pero sean destinadas a distintas filas, esto quiere decir que se debe tener:

$$(R1a = R1b) \neq (R2a = R2b) \neq (R3a = R3b) \tag{5.1}$$
$$[(C1a = C2a = C3a) \wedge (C1b = C2b = C3b) \wedge (C1a \neq C1b) \wedge (C2a \neq C2b) \wedge (C3a \neq C3b)] \vee$$
$$[(C1b = C2ba = C3a) \wedge (C1a = C2b = C3b) \wedge (C1a \neq C1b) \wedge (C2a \neq C2b) \wedge (C3a \neq C3b)] \vee$$
$$[(C1a = C2b = C3a) \wedge (C1b = C2a = C3b) \wedge (C1a \neq C1b) \wedge (C2a \neq C2b) \wedge (C3a \neq C3b)] \vee$$
$$[(C1a = C2a = C3b) \wedge (C1b = C2b = C3a) \wedge (C1a \neq C1b) \wedge (C2a \neq C2b) \wedge (C3a \neq C3b)] \vee$$
$$[(C1a = C2b = C3b) \wedge (C1b = C2a = C3a) \wedge (C1a \neq C1b) \wedge (C2a \neq C2b) \wedge (C3a \neq C3b)] \vee$$
$$[(C1b = C2a = C3b) \wedge (C1a = C2b = C3a) \wedge (C1a \neq C1b) \wedge (C2a \neq C2b) \wedge (C3a \neq C3b)] \vee$$
$$[(C1b = C2b = C3a) \wedge (C1a = C2a = C3b) \wedge (C1a \neq C1b) \wedge (C2a \neq C2b) \wedge (C3a \neq C3b)]$$

El cumplir esta restricción nos entregaría el mejor de los casos, pues solamente se necesitarían 3 ciclos para almacenar las entradas y computar la salida. Estas operaciones corresponderían a, en primer lugar, almacenar A-0, B-1 y D-0 en un ciclo en las posiciones correspondientes a la columna 1 y y filas 1-3, posteriormente en otro ciclo se podría almacenar B-0, C-0, C-1 en la columna 2 y filas 1-3 para finalmente ejecutar el computo en paralelo de G1, G2, G3.

Naturalmente la restricción que obliga a las compuertas a estar verticalmente alineadas para la ejecución en paralelo no se podría cumplir, en el caso hipotético de que existieran compuertas ocupadas como se planteó recientemente, es por esto que se debe ajustar este requerimiento de ser necesario. En este caso en particular se debe lograr mapear los pares de entradas a las compuertas en distintos pares de columnas según disponibilidad. Para esto se plantea dividir el total de compuertas que hay en un nivel en grupos suficientes para cumplir con los requerimientos de espacio. Esto se realiza de tal forma que las compuertas de un mismo grupo se puedan ejecutar en paralelo. En este sentido las restricciones para ejecución en paralelo para filas de la ecuación 4.10 y de columnas de la ecuación 4.9, deben ser para los pares de un mismo grupo y de un mismo nivel. En el ejemplo de la figura 5.7 para lograr la ejecuciones de las compuertas del nivel 1 se necesita dividir estas en 2 grupos. Existen tres formas de combinar las compuertas en estos grupos, pero para este caso se ha seleccionado un grupo que contenga solo la compuerta G1 y otro que contenga G2 y G3. En la figura 5.7-b se realiza el mapeo de estos dos grupos, y se puede observar que el primer grupo correspondiente a G1 se puede asignar al espacio de las columnas 1-3 y fila 1, mientras que el segundo grupo se le asigna el espacio de columnas 3-4 y filas 3-4.

Otro factor que se debe considerar con la solución recién planteada al problema de la ocupación en el arreglo de memoria, es que se produce también una disminución de desempeño en términos de latencia, ya que como acabamos de averiguar, no siempre las puertas lógicas de un mismo nivel se podrán ejecutar en paralelo. Para tener una idea general de cómo afecta esta solución a la latencia se introduce el parámetro $d \in [0, 1, ...G - 1]$, que representa la cantidad de veces que se necesita dividir el número de compuertas de

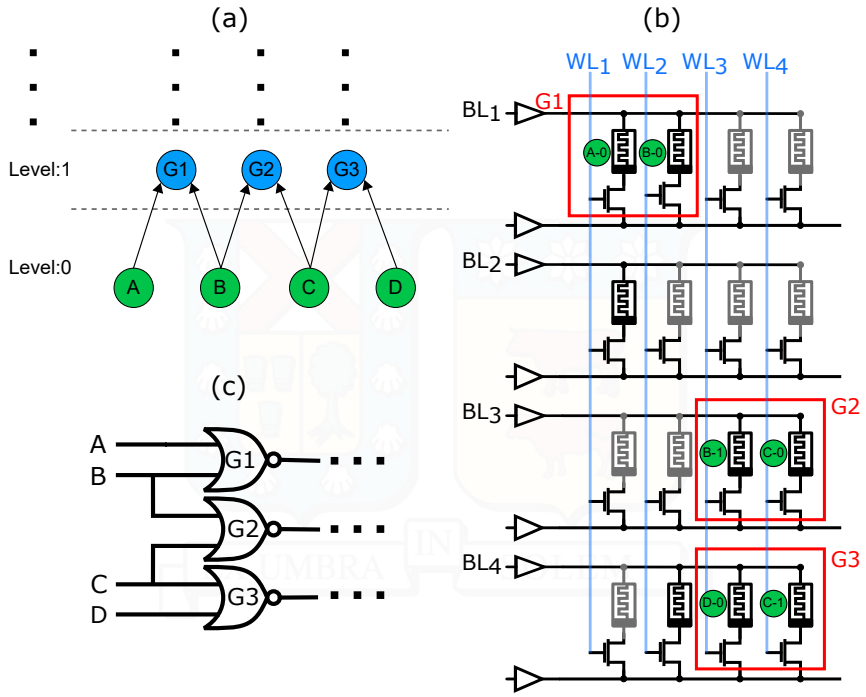


Figura 5.7: Ejemplo de asignación de compuertas con celdas de memoria ocupadas. (a) Representación en grafo de la sección de circuito lógico que se quiere mapear para posterior computo de compuertas lógicas. (b) Esquemático simplificado del circuito del arreglo de memoria 4x4 con celdas ocupadas(en gris) y mapeo posible de entradas a compuertas que se requieren computar. (c) Sección de circuito lógico que se va a computar

un mismo nivel (con un número total de G compuertas) para lograr una asignación exitosa de acuerdo a la ocupación de la memoria. De esta forma en un nivel i con d divisiones en subgrupos de puertas se tiene que el número máximo de operaciones que se necesitan para la ejecución de todas las compuertas del nivel esta dado por la ecuación:

$$L_i = (d + 1) + 2(d + 1) = 3d + 3 \tag{5.2}$$

Esta ecuación refleja el número de operaciones que se necesita para escribir las entradas $W_i = 2(d + 1)$, en donde se tiene que d+1 representa el número de divisiones, mas 1 pues d comienza desde las 0 divisiones y se multiplica por 2 pues se consideran 2 entradas para cada compuerta. Además, se tiene que el número de operaciones para computar las compuertas NOR del nivel i $Op_i = d + 1$, que representa que se tiene una operación cuando no hay divisiones y comienzan a incrementar en 1 a medida que incrementa d. De esta forma en el mejor de los casos se tiene d = 0 y $L_i = 3$ que corresponde al caso en el que no se tiene ocupación en la memoria como se presentó originalmente. El peor de los casos por otro lado corresponde a cuando d = G-1, en este caso se tendría $L_i = 3G$, lo cual es equivalente a computar una compuerta por ciclo desechando completamente la posibilidad de ejecución en paralelo.

5.7. Compuertas NOR de Entradas Variables

Como último tema de estudio sobre la flexibilidad que puede ofrecer la herramienta de síntesis desarrollada, se considera el impacto de utilizar compuertas NOR de más de dos entradas. Hasta ahora se había considerado solamente compuertas de dos entradas, pero el estilo de lógica utilizado es capaz de soportar una cantidad mayor de entradas, esto se traduciría en un principio en la reducción del número de compuertas en una descripción lógica, lo que a su vez significa una disminución en la latencia de cómputo de la función lógica sintetizada y también en el tiempo de síntesis, la cual se demostró depende de la cantidad de compuertas. No obstante, se debe considerar que, dado este escenario, una función lógica difícilmente puede

contener solamente compuertas con una cantidad de entradas fijas, por lo que en un mismo nivel convivirán compuertas con un distinto número de entradas. Esto puede tener un impacto negativo si se considera la topología de memoria utilizada.

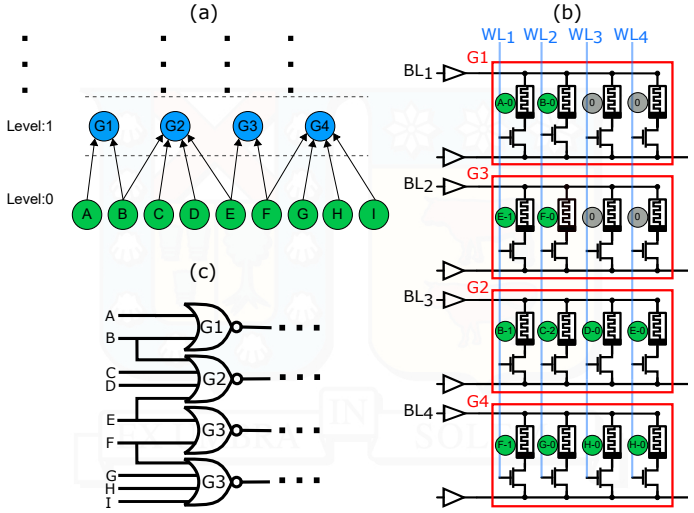


Figura 5.8: Ejemplo situación con compuertas NOR2 y NOR4. (a) Representación en grafo de la sección de circuito lógico que se quiere mapear para posterior computo de compuertas lógicas. (b) Arreglo de memoria 4x4 con distribución de compuertas NOR2 y NOR4. (c) Sección de circuito lógico que se va a computar

De esta forma se considera como ejemplo la necesidad de computar las compuertas del nivel 1, las que tienen las relaciones de entradas que se muestran en la figura 5.8-a y su respectivo circuito en la figura 5.8-c. De esta forma se tienen G1 y G3 que corresponden a compuertas NOR de dos entradas, y las compuertas G2 y G4 que son compuertas NOR de cuatro entradas. Las entradas a estas compuertas deben ser mapeadas de tal forma que se pueda lograr una ejecución en paralelo, pero debido a la diferencia en el número de entradas resulta ser un problema por dos motivos. En primer lugar se incrementaría la dificultad del problema de mapeo, al tener que modificar la restricción para la ejecución en paralelo de compuertas de un mismo nivel (que se muestra en la ecuación 4.9) para lograr considerar compuertas de distintos números de entradas. El segundo problema, y el más relevante, hace referencia al momento de computar las compuertas en paralelo y la topología de memoria diseñada. Debido a la diferencia en el número de entradas se hace imposible el computar en paralelo compuertas NOR2 y NOR4 ya que se necesita activar un numero de columnas equivalente al número de entradas de la compuerta que más entradas tenga. En este caso se tiene compuertas NOR4 y NOR2 por lo que se necesitarían activar 4 columnas lo que impactaría al correcto computo de las compuertas NOR2, ya que además de sus dos entradas correctas, activaríamos 2 columnas adicionales y, por ende, 2 entradas adicionales, cuyo valor almacenado alteraría el resultado.

Como solución a este problema se plantea transformar las compuertas NOR2 en compuertas NOR4 virtuales. Esto se logra inicializando con lógicos 0 las dos celdas adicionales a las celdas donde se encuentran almacenadas las entradas de las compuertas NOR2, cuidando de alinear estas con las columnas de las entradas de las compuertas NOR4. La implementación de esta solución para el ejemplo explicado anteriormente se muestra en la figura 5.8-b. En esta figura se muestra un ejemplo de arreglo con dimensiones 4x4 en donde se mapean las entradas a las compuertas G1,G2,G3 y G4 acorde a las restricciones. En este caso se agrega que para G1 y G3 se debe inicializar con lógico 0 las posiciones de la fila que no corresponden a entradas, logrando así no interferir en la respuesta de la operación NOR2. Esto permite tener 4 entradas alineadas en las mismas columnas para cada una de las compuertas, permitiendo así su ejecución en paralelo.

Otro factor a tener en cuenta al momento de aumentar la cantidad de entradas de las compuertas NOR con las que se están trabajando, es el referente a la latencia de ejecución por nivel, especialmente la referente a la escritura de entradas. Este parámetro está directamente relacionado con la cantidad de columnas que se activen en un nivel, de esta forma este valor es equivalente al que tenga aquella compuerta NOR con mayor número de entradas. De esta forma la latencia en un nivel i será equivalente a $L_i = E + 1$ donde E es igual al

número de entradas de aquella compuerta del nivel que posea la mayor cantidad de estas.

6 | Conclusiones y Discusión

Esta tesis fue desarrollada en el contexto de computación en la memoria, en este caso orientado a implementaciones en memorias ReRAM. De esta forma para lograr este objetivo se enfrentaron tres desafíos principales. En primer lugar, se diseñó una topología de memoria ReRAM compuesta por *memristors* como celdas de memoria. Esto incluyó el diseño del circuito de la periferia que permite el control de la memoria logrando de esta forma realizar operaciones de cómputo de compuertas lógicas mediante el estilo de lógica con *memristors* denominado "Ratioed Logic"[38]. Con este diseño se lograron simular en LTSpice operaciones de computación en la memoria utilizando una combinación de dispositivos implementados con modelos complejos y otros que utilizan modelos conductuales. De forma complementaria a este objetivo se logró implementar de manera experimental una prueba de concepto de lógica con *memristors* y las operaciones de una memoria, esto utilizando dispositivos de la empresa Knowm INC[17]. disponibles en el comercio. Para estas pruebas se planteó una topología alternativa para el computo de compuertas lógicas, la cual busca aumentar la fiabilidad de la computación cuando se tienen dispositivos con rangos de resistencia (entre alta y baja resistencia) variados. En este caso se logró con esta topología el computo de la compuerta NOR2 con dos pares de dispositivos que presentaban razones de resistencia disimiles, uno con una razón mayor a 5 y otro con una razón de resistencia menor a 2. Se pudieron realizar también operaciones propias de una memoria como lo son la escritura y lectura de los dispositivos sin problemas. Estas pruebas permitieron también corroborar la resistencia del estilo de lógica a la variabilidad ciclo a ciclo de cada dispositivo y a la variabilidad de fabricación que afecta a los dispositivos reales.

A partir del diseño de la topología de memoria ReRAM se desarrolló un flujo y algoritmo de síntesis que permite la ejecución de funciones lógicas de múltiples niveles en este arreglo de memoria diseñado. La herramienta completada logra, a partir de la descripción de una función lógica en lenguaje Verilog/Blif, generar la disposición en la memoria de las entradas y salidas de cada compuerta lógica y determina el tiempo de ejecución de las mismas. Para esto se tomó provecho del circuito de periferia diseñado, el cual es capaz de computar la compuerta NOT, lo que permitió disminuir el número de compuertas a computar en la memoria y trayendo consigo una disminución en la latencia de cómputo de la función total. Para el algoritmo de mapeo se utiliza la representación en grafo de la función lógica, más específicamente, la representación de un grafo dirigido y ponderado. Esto permitió obtener la relación entre cada compuerta y asociar cada una a un nivel de ejecución. Lo anterior se realizó para sacar provecho a la computación en paralelo de aquellas compuertas que estuvieran un mismo nivel. Esta estructura de grafo se utiliza también para plantear el problema de mapeo y las restricciones necesarias para el mismo, para esto se tomó un enfoque iterativo, esto quiere decir resolver un problema de mapeo para cada nivel.

Las pruebas realizadas al sintetizador demostraron que el algoritmo desarrollado tiene su mejor desempeño, en términos de número de operaciones (para la ejecución de una función lógica) y comparativamente con otras herramientas de la literatura, cuando se tiene una alta cantidad de compuertas en una función lógica de bajo número de niveles. Por otro lado, también se logra mejorar el uso de área y numero de dispositivos en el arreglo de memoria en comparación a las otras herramientas de la literatura, esto principalmente por que se considera la reutilización de las celdas de memoria. Se debe considerar también que el estilo de lógica utilizado entrega libertades en el mapeo, esto debido a que la salida se obtiene en forma de voltaje y este puede ser almacenado en cualquier celda, además del enfoque de mapeo nivel por nivel. Con esto se logró obtener tiempos de síntesis comparativamente mejores que las herramientas existentes en la literatura. Estas últimas no solo planteaban el problema de mapeo para la función lógica completa (a diferencia del proceso

iterativo que se plantea en este trabajo), sino que también lo hacían como un problema de optimización de latencia, lo que producía un incremento considerable en los tiempos de síntesis. En esta tesis en cambio se trabajó con la solución subóptima del problema de mapeo en el cual se buscaba minimizar el área utilizada, pues se considera que la latencia ya se encuentra implícitamente minimizada al ejecutar en paralelo todas las compuertas en un mismo nivel. No obstante, de igual forma se observó un aumento en los tiempos de síntesis en la medida que se aumenta el número de compuertas de la función lógica a sintetizar demostrando que la dependencia entre ambas variables se mantiene.

En este trabajo se exploran también otras alternativas al funcionamiento del sintetizador, en donde por ejemplo se consideró el impacto de tener un espacio con celdas ocupadas. Se observo el impacto de no poder computar en paralelo todas las compuertas en un mismo nivel, esto con distintos niveles de ocupación. Se observó en este aspecto que a medida que aumenta la ocupación del espacio de memoria, el número de operaciones necesarias para ejecutar una función lógica tiende al número de operaciones en una ejecución en serie, esto quiere decir la ejecución de una compuerta lógica por ciclo. Para evitar la degradación del desempeño en el cómputo de lógica en memorias que pueden estar densamente ocupadas, se puede plantear una arquitectura alternativa en el cual exista dos espacios de memoria independientes, en los cuales uno se dedique exclusivamente al cómputo de lógica mientras que otro espacio se dedique al almacenamiento de datos, de forma que se tendría solamente que resolver el problema de comunicación entre ambos espacios.

Finalmente se estudia la posibilidad de tener también compuertas de distintos tamaños de entrada, en donde se demuestra el impacto negativo en términos de latencia que significa el tener que ejecutar en paralelo compuertas de distintos números de entradas. Se muestra que en estos casos se necesita activar una columna por cada entrada de las compuertas del nivel para almacenar las entradas, esto además de la necesidad de inicializar con lógicos 0 los dispositivos adyacentes a las entradas de aquellas compuertas de menor niveles que también se ejecutaran en un mismo ciclo.

6.1. Trabajo Futuro

El trabajo realizado busca entregar las primeras herramientas que permitan la implementación de aplicaciones enfocadas en la computación en la memoria, de esta forma se prepara el escenario para sistemas de memoria computacional basadas en la tecnología ReRAM, que está cada vez más cerca de su comercialización. Para esto se comenzó desde los más básico, que corresponde al funcionamiento del *memristor* como elemento emergente de almacenamiento, siguiendo con la presentación de cómo se pueden computar compuertas lógicas en elementos de memoria, finalmente se concluyó con un sistema que permite la ejecución de funciones lógicas varios niveles de manera organizada. La extensión natural para este trabajo considera el completar el procesamiento de la salida del sintetizador, de tal forma de generar las instrucciones necesarias para el control de la periferia y la ejecución de las funciones lógicas a sintetizar. Para esto en primer lugar se deben definir un set de instrucciones completo (Instruction Set Architecture- ISA) y como se traducen estas a señales de control en la arquitectura de memoria diseñada.

Como trabajo futuro también se plantea el optimizar el arreglo de memoria diseñado, buscando un mejor uso del espacio y numero de dispositivos. Además de esto queda pendiente el estudio sobre el impacto energético que tiene el uso de la memoria ReRAM diseñada, esto en termino de número de operaciones, buscando una comparación con otras implementaciones y tecnologías convencionales. También esta optimización puede estar enfocada plenamente a la ejecución de las funciones lógicas, diseñando un arreglo con bancos de memoria separados, que permitan espacios dedicados solamente al cómputo de las funciones sintetizadas. A partir de la optimización anterior se pueden evitar los problemas relacionados a la ocupación de celdas de memoria descritos previamente.

Finalmente, otra área de continuación que puede tener este trabajo está relacionado con las mejoras que se pueden incluir a la herramienta de síntesis desarrollada. Especifícamele se puede optimizar el código referente a la etapa de mapeo de las compuertas lógicas, en donde se puede prescindir de la herramienta Z3, cambiando así el algoritmo de mapeo por uno que busque optimizar el tiempo de síntesis.

Bibliografía

[1] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff.," *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006. 0.1

[2] Wikipedia. https://en.wikipedia.org/wiki/Transistor_count, Septiembre 2022. Acceso Septiembre de 2022. 1

[3] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974. 0.1

[4] C. Märtin, "Post-dennard scaling and the final years of moore ' s law consequences for the evolution of multicore-architectures," 2014. 0.1

[5] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), (New York, NY, USA), p. 483–485, Association for Computing Machinery, 1967. 0.1

[6] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, p. 48–60, jan 2019. 2

[7] J. von Neumann, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993. 3

[8] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, *Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks*, p. 316–331. New York, NY, USA: Association for Computing Machinery, 2018. 0.1

[9] H.-S. P. Wong and S. S. Salahuddin, "Memory leads the way to better computing.," *Nature nanotechnology*, vol. 10 3, pp. 191–4, 2015. 4

[10] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation," *Microprocessors and Microsystems*, vol. 67, pp. 28–41, 2019. 0.1.1

[11] B. Li, B. Yan, and H. Li, "An overview of in-memory processing with emerging non-volatile memory for data-intensive applications," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, GLSVLSI '19, (New York, NY, USA), p. 381–386, Association for Computing Machinery, 2019. 1

[12] S. Yu and P.-Y. Chen, "Emerging memory technologies: Recent trends and prospects," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 2, pp. 43–56, 2016. 1

[13] WeeBit Nano. https://www.weebit-nano.com/technology/, Febrero 2022. Acceso Enero de 2022. 0.1.2

[14] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971. 0.1.2, 1.0.1

[15] D. B. Strukov, G. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, 2008. 0.1.2, 1.0.2, 1.0.2, 1.4, 1.0.6

[16] X. Zhu, X. Yang, C. Wu, N. Xiao, J. Wu, and X. Yi, "Performing stateful logic on memristor memory," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 10, pp. 682–686, 2013. 0.1.2

[17] Knowm Inc. https://knowm.org/, Enero 2022. Acceso Enero de 2022. 0.1.3, 3.4, 6

[18] C. Fernandez and I. Vourkas, "Circuit topology and synthesis flow co-design for the development of computational reram," in *22nd IEEE International Conference on Nanotechnology (IEEE-NANO 2022), July 4-8, 2022, Palma, Balearic Islands, Spain*, 2022. 0.1.4

[19] C. Fernandez and I. Vourkas, "Reliability-aware ratioed logic operations for energy-efficient computational reram," in *30th IFIP/IEEE International Conference on Very Large Scale Integration, October 3-5, Patras, Greece*, 2022. 0.1.4

[20] C. Fernandez and I. Vourkas, "On the design and development of a reram-based computational memory prototype," in *30th IFIP/IEEE International Conference on Very Large Scale Integration (student forum), October 3-5, Patras, Greece*, 2022. 0.1.4

[21] I. V. C. Fernandez and A. Rubio, "Design and simulation of peripheral driving circuitry for computational reram," in *37th edition of the Conference on Design of Circuits and Integrated Systems (DCIS 2022), November 16-18, 2022, Pamplona, Spain*, 2022. 0.1.4

[22] L. Chua and S. M. Kang, "Memristive devices and systems," *Proceedings of the IEEE*, vol. 64, no. 2, pp. 209–223, 1976. 1.0.1

[23] I. Vourkas and G. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, vol. 19. 10 2015. 1.0.1

[24] L. O. Chua, "Resistance switching memories are memristors," *Applied Physics A*, vol. 102, pp. 765–783, 2011. 1.0.1

[25] J. J. Yang, M. Pickett, X. Li, D. Ohlberg, D. Stewart, and S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," *Nature nanotechnology*, vol. 3, pp. 429–33, 07 2008. 1.3

[26] Z. Biolek, D. Biolek, and B. V, "Spice model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, 06 2009. 1.0.2, 1.0.6

[27] S. La Barbera, G. Sassine, N. Najjari, M. Minvielle, C. Dubourdieu, and F. Alibart, "Interfacial versus filamentary resistive switching in tio2 and hfo2 devices," *Journal of vacuum science technology B*, vol. 34, p. 012202, 01 2016. 1.0.3

[28] M. Lanza, H.-S. P. Wong, E. Pop, D. Ielmini, D. Strukov, B. C. Regan, L. Larcher, M. A. Villena, J. J. Yang, L. Goux, A. Belmonte, Y. Yang, F. M. Puglisi, J. Kang, B. Magyari-Köpe, E. Yalon, A. J. Kenyon, M. Buckwell, A. Mehonic, A. L. Shluger, H. Li, T.-H. Hou, B. Hudec, D. Akinwande, R. Ge, S. Ambrogio, J. B. Roldán, E. A. Miranda, J. Suñé, K. L. Pey, X. Wu, N. Raghavan, E. Y. Wu, W. Lu, G. Navarro, W. Zhang, H. Wu, R. Li, A. W. Holleitner, U. Wurstbauer, M. C. Lemme, M. Liu, S. Long, Q. Liu, H. Lv, A. Padovani, P. Pavan, I. Valov, X. Jing, T. Han, K. Zhu, S. Chen, F. Hui, and Y. Shi, "Recommended methods to study resistive switching devices," *Advanced Electronic Materials*, 2018. 1.0.3

[29] Y. V. Pershin and M. D. Ventra, "Memory effects in complex materials and nanoscale systems," *Advances in Physics*, vol. 60, no. 2, pp. 145–227, 2011. 1.0.4

[30] Y. Pershin and M. Di Ventra, "Spice model of memristive devices with threshold," *Radioengineering*, vol. 22, 04 2012. 1.0.4, 1.0.6

[31] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015. 1.0.5, 5.2

[32] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Team: Threshold adaptive memristor model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 1, pp. 211–221, 2013. 1.0.5

[33] D. Biolek, Z. Kolka, V. Biolková, Z. Biolek, and S. Kvatinsky, "(v)team for spice simulation of memristive devices with improved numerical performance," *IEEE Access*, vol. 9, pp. 30242–30255, 2021. 1.0.5

[34] C. Fernandez, J. Ortiz, and I. Vourkas, "A voltage-driven window function concept for behavioral memristor device modeling," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020. 1.0.5

[35] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014. 1.0.5, 1.1, 2.1, 2.0.1

[36] J. J. Yang, D. Strukov, and D. Stewart, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, pp. 13–24, 01 2013. 1.0.5

[37] L. Qu, X. Cui, and X. Cui, "A testability enhancement method for the memristor ratioed logic circuits," in *2020 IEEE 29th Asian Test Symposium (ATS)*, pp. 1–6, 2020. 2.1, 2.0.1

[38] M. Escudero, I. Vourkas, A. Rubio, and F. Moll, "Memristive logic in crossbar memory arrays: Variability-aware design for higher reliability," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 635–646, 2019. 2.1, 2.0.1, 2.0.2, 6

[39] I. Vourkas and G. C. Sirakoulis, "A novel design and modeling paradigm for memristor-based crossbar circuits," *IEEE Transactions on Nanotechnology*, vol. 11, no. 6, pp. 1151–1159, 2012. 2.0.2

[40] L. Xie, H. Du Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. AlFailakawi, and S. Hamdioui, "Scouting logic: A novel memristor-based logic design for resistive computing," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 176–181, 2017. 2.0.2, 3.3

[41] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," pp. 156–160, 07 2013. 3.1

[42] R. J. Baker, *CMOS Circuit Design, Layout, and Simulation*. Wiley-IEEE Press, 3rd ed., 2010. 3.1.1

[43] J. Gomez, I. Vourkas, and A. Abusleme, "Exploring memristor multi-level tuning dependencies on the applied pulse properties via a low cost instrumentation setup," *IEEE Access*, vol. 7, pp. 59413–59421, 2019. 3.4

[44] R. B. Hur and S. Kvatinsky, "Memory processing unit for in-memory processing," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 171–172, 2016. 4.1

[45] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification* (T. Touili, B. Cook, and P. Jackson, eds.), (Berlin, Heidelberg), pp. 24–40, Springer Berlin Heidelberg, 2010. 4.1, 4.2.2.1

[46] R. Ben Hur, N. Wald, N. Talati, and S. Kvatinsky, "Simple magic: Synthesis and in-memory mapping of logic execution for memristor-aided logic," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 225–232, 2017. 4.1.1, 4.2, **??**

[47] L. de Moura and N. Bjørner, "Z3: an efficient smt solver," vol. 4963, pp. 337–340, 04 2008. 4.1.1, 4.2.2.4

[48] Z. Zhu, M. Ma, J. Liu, L. Xu, X. Chen, Y. Yang, Y. Wang, and H. zhong Yang, "A general logic synthesis framework for memristor-based logic design," *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019. 4.1.2, 4.3, 5.5, 5.5, **??**

[49] V. Tenace, R. G. Rizzo, D. Bhattacharjee, A. Chattopadhyay, and A. Calimera, "Said: A supergate-aided logic synthesis flow for memristive crossbars," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 372–377, 2019. 4.1.3, 4.4, 4.5, 5.3

[50] S. Yang, *Logic synthesis and optimization benchmarks user guide: version 3.0*. Citeseer, 1991. 5.3, 5.4

[51] K. McElvain, "Benchmark set: Version 4.0," 01 1993. 5.3

EX UMBRA IN SOLEM