Long-Term Memory for Cognitive Architectures

A Hardware Approach Using Resistive Devices

Peng Wang

B.E. (Hons.)

School of Electrical and Electronic Engineering

University of Adelaide

This thesis is submitted for the degree of

Doctor of Philosophy

THE UNIVERSITY *of* ADELAIDE

November 2018

Supervisors:
Dr. Braden Phillips
Prof. Michael Liebelt
Dr. Brian Ng

THE UNIVERSITY *of* ADELAIDE

Declaration

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Acknowledgements

I would like to convey my deepest gratitude to my supervisors Braden Phillips, Brian Ng and Michael Liebelt. I greatly benefited from Braden's guidance and 'cunning plans' in research and enjoyed his sense of humour as well as all the stories he shared. Brian Ng gave me useful suggestions at the beginning of both my undergraduate and PhD study, and Michael Liebelt was always supportive and willing to correct my papers throughout the years.

I acknowledge the contributions of Nate Derbinsky who generously helped me set up the word sense disambiguation experiment. My thanks also extend to Francis Li, Jesse Frost, Mostafa Wasiuddin Numan and Yansong Gao for their four year's mutual support, complaining and insightful discussions in the school office.

I also would like to pass my thanks to the two anonymous examiners for their review and suggestions.

The following fellow researchers have made my coffee break and lunch time enjoyable and rewarding at the University of Adelaide: Nick Lawrence, Cheng Zhao, Chengjun Zou, Wendy Suk Ling Lee. I am also grateful for many friends outside the university, who saved me from my dull hotel life at the beginning of my study in Adelaide.

Finally, I owe the biggest thanks to the support of my Mum and Dad who believed that my undergraduate study in Australia is the best investment in their lives. Their love and guidance have always been with me in whatever I pursuit.

And to my beloved Jiamei. Thank you for all you have done.

Abstract

A cognitive agent capable of reliably performing complex tasks over a long time will acquire a large store of knowledge. To interact with changing circumstances, the agent will need to quickly search and retrieve knowledge relevant to its current context. Real time knowledge search and cognitive processing like this is a challenge for conventional computers, which are not optimised for such tasks.

This thesis describes a new content-addressable memory, based on resistive devices, that can perform massively parallel knowledge search in the memory array. The fundamental circuit block that supports this capability is a memory cell that closely couples comparison logic with non-volatile storage. By using resistive devices instead of transistors in both the comparison circuit and storage elements, this cell improves area density by over an order of magnitude compared to state of the art CMOS implementations. The resulting memory does not need power to maintain stored information, and is therefore well suited to cognitive agents with large long-term memories.

The memory incorporates activation circuits, which bias the knowledge retrieval process according to past memory access patterns. This is achieved by approximating the widely used base-level activation function using resistive devices to store, maintain and compare activation values. By distributing an instance of this circuit to every row in memory, the activation for all memory objects can be updated in parallel. A test using the word sense disambiguation task shows this circuit-based activation model only incurs a small loss in accuracy compared to exact base-level calculations.

A variation of spreading activation can also be achieved in-memory. Memory objects are encoded with high-dimensional vectors that create association between correlated representations. By storing these high-dimensional vectors in the new content-addressable memory, activation can be spread to related objects during search operations. The new memory is scalable, power and area efficient, and performs operations in parallel that are infeasible in real-time for a sequential processor with a conventional memory hierarchy.

Table of Contents

List of Figures

List of Tables

Nomenclature

Roman Symbols

A-E	memory objects
B	the base-level activation function
$\hat{B}$	an approximation of the B
$\lvert E \rvert$	the total number of memory objects
L	the thickness of a memristor's medal-oxide layer
M	a memristor
N	memory depth
Q	a search cue
$Q[\alpha]$	an inverted index list of memory objects containing α
R	resistance of a device
$\mathfrak{R}$	the resistance ratio between the LRS and HRS states
S	a selector
T	a transistor
W	memory width
x	a memristor's state variable
X	a wildcard
Z	an unknown value that could be 0, 1 or a wildcard

Greek Symbols

$\alpha - \varepsilon$	augmentations of memory objects
ε_0	the free space permittivity
$\varepsilon_{\mathrm{r}_{\mathrm{film}}}$	the permittivity of tantalum oxide
λ	half the feature size of a manufacturing process

Superscripts

d	the decay factor in B

Subscripts

$access_i$	the time when the memory element was accessed for the i-th time
cond	conditionally applied voltage
current	the current cycle
i, j	subscript indexes
pd	the propagation delay
read	voltage read
ref	a reference matchline
reset	voltage RESET
set	voltage SET
sp	a sensing point
th	threshold

Other Symbols

$p$IMP$q$	p implies q
$M = [x, y]$	the majority-sum function on inputs x and y
smem	an interface to semantic memory in Soar
xS-yR	a memory cell structure contains x number of selectors and y number of memristors

*x*T-*y*R	a memory cell structure contains *x* number of transistors and *y* number of memristors

Acronyms / Abbreviations

AI	Artificial Intelligence
BE	Bottom Electrode
BLA	Base-Level Activation
CAM	Content-Addressable Memory
CMOS	Complementary Metal-Oxide-Semiconductor
DRAM	Dynamic Random Access Memory
HRS	High Resistance State
LRS	Low Resistance State
LTI	Long-Term Identifier
ML	Matchline
PIM	Processing in Memory
RC	Resistor-Capacitor
RRAM	Resistive Random Access Memory
SA	Sense Amplifiers
SL	Selectline
SRAM	Static Random Access Memory
STT	(Spin-Torque Transfer Devices
TCAM	Ternary Content-Addressable Memory
TE	Top Electrode
WL	Wordline
WME	Working Memory Element
WSD	Word Sense Disambiguation
WTA	Winner-Take-All

Chapter 1

Introduction

An Artificial Intelligence (AI) capable of general, human-like behaviours will require large stores of knowledge. Like a human it will need to remember facts about its world, skills and strategies that have proved effective, and experiential memories. If the AI is to be self-contained, mobile and of a human scale then this memory will need to be compact, dissipate only modest amounts of power, and persist over long time frames. To respond in real-time to unexpected events in a changing environment the AI will need to be able to quickly search its memories to retrieve knowledge relevant to its current circumstances. Microelectronic memories to satisfy these requirements are the goal of this thesis.

AI is transforming our lives, but until very recently, there has been little interest in designing microelectronic hardware specifically for AI. In the last few years that has suddenly changed and major industry players have begun to race to produce chips that better serve the computational requirements of AI than general purpose computers. For example, the latest iPhone includes a neural engine optimised for machine learning algorithms [1] and Google has released details of the custom integrated circuits it uses in its data centres to accelerate neural network computations [2].

Although both of the previous examples, Apple's neural engine and Google's tensor processing unit, accelerate the numerical computations required for a neural network approach to AI, a general human-like AI will also need to store and retrieve symbolic knowledge. At present, AI agents such as the digital assistants Siri or Cortana, make use of the huge storage and computational resources of distributed data centres to handle symbolic queries. How can all this be miniaturised for a mobile, self-contained system?

1.1 Cognitive Architectures

Cognitive architectures are one of the important approaches towards symbolic processing for AI. Important examples such as *ACT-R* and *Soar* demonstrate the potential of cognitive architectures by successfully modelling psychological characteristics of humans [3, 4]. This thesis uses examples from cognitive architectures (mainly Soar) to dictate memory system requirements (see Chapter 2.1) and evaluate the performance of the new memory designs in experiments.

Memories in Soar model human memory by providing information of past experience stored in episodic memory, or general facts about the world in the semantic memory. Human memory is more than just a mechanistic account of information storage and retrieval [5]. Likewise, long-term memory in cognitive systems is expected to quickly examine the available knowledge in a retrieval and apply the right knowledge according to the circumstances. Currently Soar uses an optimised SQLite database to query the information from these two memories. The performance of knowledge retrieval has been impeded by the software implementation sitting on conventional computers [6]. A specialised hardware architecture is needed to sustain the progress of Soar. Cognitive architectures are essentially knowledge based; the capabilities of cognitive agents largely depend on these agents' knowledge size and their abilities to use these knowledge. Thus, one of the many problems that will be faced in designing such a hardware architecture is the need to quickly and intelligently retrieve information from a large knowledge store, much more quickly and intelligently than is likely to be possible with conventional von Neumann computers.

1.2 The Processor/Memory Split

Maintaining the separation of fast processing logic and slow memories in von Neumann computers has lead to what is known as the *memory wall* [7]. Programs in a von Neumann computer need to continuously fetch and store data which results in significant data movement across the memory hierarchy. Meanwhile, a large amount of power is lost on the memory access path. The memory access path is normally a shared bus with limited bandwidth such that a program's data needs to queue to be sent to processors for any computations [8, chap. 2]. Thus, in spite of the large caches, multi-core processors, and out-of-order superscalar pipelines in modern computers, long latency and limited bandwidth to main memory have dominated performance for applications such as database and matrix computations [9].

Also, the execution path of the von Neumann architecture is inherently sequential and hence cannot keep up with the computation requirements of new applications [8, chap. 2]. For instance, a typical smoothing operation on an image involves averaging neighbouring pixel values. For four pixel neighbourhoods, an image with only one thousand by one thousand pixels needs four million such operations. Traditional methods process the data sequentially and often introduce address handling overhead and read latency due to the location addressable nature of the memory.

To hide the long memory latency and sustain the performance, more prediction logic and deeper cache hierarchies are deployed, but this approach is becoming increasingly expensive and inefficient [10]. As the amount of data increases in new applications, memory locality principles are becoming less effective because data access follows less regular patterns [11, 10]. Thus, it is becoming more difficult for the prediction logic to place data at the right memory place in the hierarchy. Deeper cache hierarchies lead to longer memory latency in the worst case [9], and can therefore exacerbate the system performance in solving some emerging data-intensive problems. A natural question thus arises from this discussion: would it be beneficial to move computations closer to memory?

1.2.1 Examples of Processing in Memory

To address the inefficiency in von Neumann architectures, researchers in computer architecture have tried to bridge the performance gap between processor and memory by bringing computation closer to memory. A review of this topic recognised two categories of efforts: (1) providing an intermediate interface (2) integrating logic circuits to the memory chip. The second concept is also called *processing in memory* (PIM).

Examples of proposals at the intermediate layers (often memory controller) between processor and memory include the logic-in-memory-computer [12], a high-speed interface buffer between host processor and memory; *Impulse* [13], a smart controller offering prefetching functions and improvements on cache and bus utilisations; *Centaur* [14], a memory controller on the *Power8* processor that is equipped with a 16 MiB cache to improve memory throughput and latency performance.

A more ambitious effort has sought to move computation all the way into the memory. The main benefit of this approach comes from taking advantage of the available bandwidth when memory and logic are on the same chip. Examples from the literature include *Terasys* [15], which has a single bit ALU that is able to perform specific operations on a row of the local memory; *IRAM* [9, 16], which merged microprocessor and *dynamic random access* *memory* on the same chip and achieved a significant improvement on bandwidth and latency. *HMC* [17], a new memory structure enabled by recent progress in 3D stacking technologies, addresses density issues with PIM chips by connecting logic and memory layers using new interconnects.

Massively parallel processing may be considered extreme examples of the PIM approach. A massively parallel machine can have thousands of processing nodes, each with local memory. It is included in the review to show the potential of a distributed parallel architectures with PIM characteristics. These machines usually provides inter-node communication facilities in addition to memory and logic such that multiple processing nodes can be highly connected to form large arrays [18]. Although massively parallel architectures are often limited by the communication overhead in sending and recieving data [19, chap. 4], their highly connected network of processors is more like a human brain than location-addressable machines with centralised processors. We know very little about how the brain works, but it is generally believed that the computation elements and memory elements in a brain must be closely and densely integrated along all of its three dimensions. One way to approach this is to devise a functional element to match the symbolic data and connections of a semantic network, such that a collection of the highly connected elements in an integrated network is able to perform a specific semantic processing task more efficiently. An important early example of this approach was the *Connection Machines* designed by Thinking Machines Corporation in Massachusetts in the 1980's and early 90's, which were based on the concept of a brain-like thinking machine [20].

Let us come back to the image processing example to illustrate the potential of parallel architectures with PIM capabilities. To match the data structure in image processing, it would be best to put a processor at each pixel of the image so that calculation at the pixel level can be computed locally. With processors allocated to each pixel, it is possible for the smoothing task to reduce the number of sequential operations from four millions to just four, in which all the pixels operate concurrently to share their own values to their four adjacent neighbours.

One common characteristic of all these prior mentioned architectures is their ability to address data directly without much location address handling and with low memory access latency. This is called *content addressability* [10] and can be achieved by distributing many small, cheap, efficient computing elements within the memory.

1.3 A Cognitive Memory

Memories in cognitive architectures model human memory by providing information of past experience or general facts about the world. It has been estimated that a cognitive memory would acquire 96 GiB of data in a few days when performing in a simple knowledge domain [6]. To expand applications, a main focus in the cognitive architecture community is to provide cognitive agents with access to large knowledge stores [21–23]. We believe that to model the human brain, advanced agents will persist for longer time and acquire a larger amount of information to be able to work across multiple knowledge domains in a dynamic environment.

At present, simple agents with limited long-term memory capacity (e.g. up to about 400 MiB) are able to search the memory with access times enabling real-time response. This is not the case for agents with large long-term memories. As knowledge continues to accumulate, the search time increases super-linearly [6]. Moreover, current software implementations do not apply bias mechanisms to all memory items to prioritise potentially useful information when multiple similar search results are found [5, 24]. Nor do they support real-time priming mechanisms to bias knowledge to the current context [21]. However, these bias mechanisms are essential to a cognitive agent's ability to retrieve relevant information from vast quantities of noisy data in a similar way as our brain does.

Recognising these needs, I hypothesise that parallelism at the data level enabled by new PIM schemes offers a solution to search problems over large knowledge stores. Take semantic memory for example. Semantic concepts are lists of records, and a search is carried out by comparing a query against these records. Thus, the core operation of a search is the bit-level comparison. Similar to the image processing example, it would be ideal if compact comparison circuits could be distributed into every bit cell in the memory, such that once a query word is broadcast to every memory word, all bit cells in memory will be compared against query data locally at the same time. Theoretically, such a search will always be able to finish within constant time. A natural implementation of this kind of memory is the *content-addressable memory* (CAM).

The use of content-addressable memory speeds up the search process dramatically, and the saved time could be used for more biology-inspired knowledge retrievals. Optimised database implementations retrieve semantic information from a modest knowledge base in the millisecond range, which is the typical time scale of human cognition. A goal of this study is to propose hardware memories with in-memory parallel searches that can achieve informational retrieval periodically in microseconds or even nanoseconds for the same retrieval complexity. Bias mechanisms in the retrieval process need to reference results generated from search operations to calculate data in memory biasing. Content-addressable memory is able to supply these data earlier, which leaves more time to bias mechanisms.

It is also possible to integrate hardware-based memory bias mechanisms in or close to memory. Cognitive memories model human memory by dynamically updating a record for every memory concept after a memory access. For large knowledge stores containing millions of concepts, the updating activities will involve a great deal of data shuttling through memory hierarchies. By providing hardware support in the memory, the associate computations are decoupled from the centralised processor, thus records of individual memory concepts can be computed simultaneously at their residing locations. Both the CAM and the bias mechanisms mentioned need a large number of computing elements integrated in memory. The trade-off is: with more functions built in to the memory, it is difficult to densely pack a massive amount of memory cells in a single chip with current CMOS technologies.

Emerging nanoscale resistive devices (e.g. memristors) show promising features in the implementation of information computing and storage [25, 10]. This thesis thus, explores the possibilities of merging the memristive computing elements with the memristive storage elements into a homogeneous memory. For instance, the versatility of memristive devices offer the chance to replace the bulky and power-hungry CAM bit cells, as well as the built-in comparison circuits (refer to Chapter 4). Another interesting use of memristors is in the design of analog circuits for the heuristic biasing mechanisms (refer to Chapter 3 and Chapter 6). Indeed, these functions does not have a strict requirement for data precision, and a compact approximate implementation based on memristors will significantly benefit the computation efficiency. Often, these new circuits only consist of a few memristors, and the data stored in these memristors is able to persist for years without power, which is beneficial to long-lived agents.

The aim of this research is to explore architectural techniques and circuit techniques using emerging resistive devices, for supporting fast, efficient and intelligent knowledge retrievals from large cognitive memories. A hardware memory like this will contribute to the core component of many human memory inspired AI systems.

1.4 Original Contributions

This thesis describes a new hardware approach to the long-term memory in cognitive architectures, which performs memory search and cognitive processing operations using a flat resistive content-addressable memory.

Such a memory needs to be optimised according to the processing requirements of cognitive memories. I analysed the knowledge retrieval process in Soar's long-term memory running on conventional computers and identified key performance bottlenecks including the memory search and bias mechanisms in large long-term memories. With the understanding of these higher level requirements, I decided to approach the new cognitive memory using content-addressable memories, which can perform in-memory parallel searches. To the best of my knowledge, the idea of a hardware based content-addressable memory optimised for long-term memory retrievals is new for cognitive architectures. The arguments to support this approach can be found in Section 2.1 and Section 2.2.

However, content-addressable memories using CMOS technologies are prone to cell area and power overheads. A list of existing improvements using emerging devices is reviewed in Section 2.3 and Section 3.1. Memristive devices are particularly promising in this regard, and hence I enhanced a memristive CAM cell using techniques drawn from CMOS technology, as described in Section 3.2. The enhancements described in this thesis include: matchline sense amplifiers for quick sampling and a reference matchline for supporting variation-aware timing control (Section 3.3.2); and hierarchical bitline circuits for scalable and leakage-free read operations (Section 3.3.3). The device level characteristics have a major impact on search latency and sensing margins. I observed that memristors with a low resistive state and a high internal resistance ratio will give a lower latency and a wider sensing margin (Section 3.3.1 and Section 3.4). This work was presented at the *16th IEEE International Conference on Nanotechnology*, with the title of "A Design and Evaluation of Content-addressable Memory using Redox-based Memristive Devices" [26].

To further reduce the cell area overhead, I developed a new CAM cell based on recently published selector-based *resistive random access memories* (RRAM) in Chapter 4 (Section 4.2). An in-depth analysis is given on how to use RRAM cells to build CAM cells that are able to perform either binary or ternary CAM searches. The new cell has the potential to achieve over an order of magnitude density improvement compared with the state of the art CMOS CAM designs (refer to Section 4.2.5). To maintain the memory array area efficiency, I proposed a new bit-serial read scheme that reuses matchlines and sense amplifiers in search operations (Section 4.2.4). In a case study involving four million memory elements, the new memory is estimated to retrieve the required information within several microseconds (refer to Section 4.3).

Apart from storing and retrieving knowledge, long-term memories also perform cognitive processing such as biasing the retrieval process using *activation* values. I investigated dedicated activation circuits using memristors in Chapter 5. The digital version of the activation circuits reduce the storage area of previously proposed CMOS design (Section 5.2). I then approached the activation circuit using analog memristors, which are able to approximate some critical dynamics of activation functions when appropriate voltage biases are applied (Section 5.3). These circuits, to the best of my knowledge, are the first circuits proposed to support activation functions using memristors. A case study using word sense disambiguation tasks suggests that this approach only has a small penalty in reliability compared with other biasing models (refer to Section 5.4.2). However, this approach computes activation values of every memory object in parallel. Also, it represents a step towards a memory closely coupling computation and memory elements. This work is presented in *Electronics Letters*, with the title of "Memristor-based activation circuit for long-term memories in cognitive architectures" [27].

For Chapter 6, I proposed a memory model to support a variant of the spreading activation scheme. Memory objects use the high-dimensional representation as suggested by [28] for a holographic declarative memory. This memory model is entirely new and combines building blocks developed in previous chapters (Section 6.2). To the best of my knowledge, it is also the first memory model that introduces high-dimensional computing to the design of spreading activation schemes. A key building block of the memory model is a circuit that finds a list of candidate objects strongly correlated to the search cue. I proposed an up-counting circuit that finds all objects within a threshold of the object with the strongest correlations (Section 6.3.2). This is done without communicating between the memory objects so the delay is independent of the depth of the memory.

1.5 Thesis Outline

This thesis begins with identifying long-term memory requirements in cognitive architectures. To meet the requirements, I propose several memory circuits based on memristors, each of which either enhances or implements a particular function of the new memory, leading to a hardware-based memory optimised for storing and retrieving a large amount of long-term knowledge.

Chapter 2 reviews the limitations of the current software implementation of cognitive long-term memories and motivates the development of a hardware-supported cognitive memory that is scalable to a large knowledge base. The search for a memory with fast look-up capabilities leads us to content-addressable memories whose quick search comes with a large area and power overhead. The chapter reviews memristor features at the device level in search of possible solutions for CAM's overhead, and also more efficient implementations for other memory functions such as *base-level activation* (BLA) and *spreading activation* mechanisms. The understanding of the top-level architectural requirements, CAM circuit characteristics and memristor's device level behaviours leads to the development of new circuits, techniques and ideas in the following chapters.

Chapter 3 focuses on the design of memristive content-addressable memory, which is the core component of the proposed memory in the thesis. The reduction of the area and power overhead of cells is the foremost objective of this new CAM. This chapter begins with review of CAM improvements using emerging devices. It then proposes a new memory cell which replaces the complementary storage bits using *static random access memory* (SRAM) with two complementary memristors and keeps two access transistors in the cell for comparison operations. Memory operations based on this new memory cell are explained, including read, write and bit-comparison operations. A series of circuits are enhanced for memory search at the array level. This includes: matchline sensing circuits, reference matchlines, and hierarchical bitlines which are used to address the possible leakage current flowing into inactive matchlines in a read operation. The impact of device level characteristics on search performance is also discussed.

Chapter 4 continues to reduce the CAM cell overhead by building upon existing resistive RAMs based on crossbar structures; a special encoding scheme groups two RAM cells into a complementary CAM cell which is able to perform binary or ternary comparison operations. The memory array uses the same peripheral circuits developed in Chapter 3 except for the read circuit. To improve the memory array area efficiency, this chapter proposes a bit-serial read scheme which reuses matchlines and sense amplifiers of search operations. Such a read operation eliminates the need for extra sensing facilities along the highly capacitive selectlines. The proposed CAM array is further developed to form large memory blocks that are able to store four million memory items. This large memory is evaluated in a case study for area and power estimations. The remainder of this chapter discusses and concludes the applicability of existing circuit techniques for improving large CAM performance.

Chapter 5 introduces a memory bias mechanism based on a compact activation circuit, which is used to predict the likelihood that a memory object will be accessed in the next retrieval process. Faithful activation models tend to have computational issues in current software implementations. The proposed activation circuit computes an approximated version of the BLA model by exploiting analog behaviours of memristors; a single memristor is used to store multiple values over time. In operation, the activation values can be dynamically updated according to memory access patterns. A comparison circuit is used to find the memory object with the highest activation value. A case study is conducted to verify the effectiveness of this approximated version of BLA. The word sense disambiguation task is used in the case study where I replace the agent's default activation function with the proposed version modelled in software. The significance of this approach is that the proposed circuits support a parallel computation distributed in the long-term memory with a small accuracy penalty.

Chapter 6 describes an activation spreading scheme to prime the memory to the current context. The chapter begins with an introduction of high-dimensional computing and spreading activations. High-dimensional representation provides useful tools to distinguish or correlate different memory objects. The proposed scheme spreads activation according to the correlation information between objects: a search cue, serving as the activation source, spreads activation to correlated memory objects according to the correlation values generated from the associative memory which is implemented using CAMs. The culmination of this chapter is a memory design incorporating circuits discussed in previous chapters, providing a possibility of closely coupling search, spreading, and bias functions with the content of the memory.

Chapter 7 summarises the work and contributions presented in this thesis. It also provides an outlook of possible future research improvements and directions.

Chapter 2

Introduction to Memristive Long-Term CAMs

2.1 Long-Term Memory in The Soar Cognitive Architecture

A cognitive architecture is a general computation structure that realises a set of basic operation principles of cognition. Agents based on cognitive architectures are able to learn to perform many different tasks when supplied with a baseline of knowledge. The state of art cognitive architectures have demonstrated abilities to perform many human-like behaviours and are seen as one of the most promising approaches in the race to develop artificial agents with human-like intelligence [29, 30]. This thesis focuses on one of the most well-developed and advanced cognitive architectures, Soar, to analyse its memory module requirements and show the need for optimised hardware support as the scale and complexity of cognitive tasks increase.

A top-level system diagram of Soar is shown in Figure 2.1. The cognitive architecture consists of memories, such as episodic and semantic memories, and functional modules, such as a module to realise episodic learning. Memory in a cognitive architecture is typically arranged in two tiers: *working memory* and *long-term memory* [4]. Working memory serves as Soar's short-term knowledge and holds information that is directly relevant to the current problem-solving situation. Working memory contains top-level entities called objects, which are constructed from working memory elements (WMEs). WMEs are also referred to as augmentations of an object. Objects are organised in a single, connected, directed graph, as

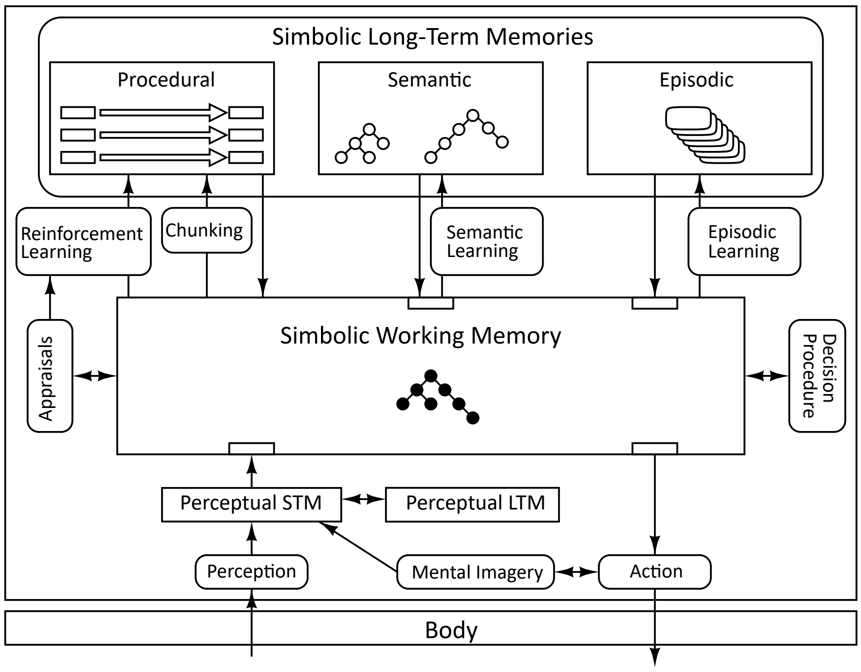


Figure 2.1 The Soar architecture [4]

illustrated in Figure 2.2; each WME contains a specific piece of information arranged as an *Identifier-Attribute-Value* structure. The terminologies involved are shown as follows:

- Identifier (short-term Identifier): each object has a unique and permanent ID as a letter-number pair.
- Attribute: the attribute in a WME is a symbolic constant[1].
- Value: the value of a WME is a symbolic constant or an identifier.
- Symbolic Constants: are ASCII strings, integers or floating point values.

Working memory is in some ways analogical to cache in a conventional computer system, where, when more information is required for processing the current task, the system will issue a command to fetch the missing information from a lower level memory store such as main memory or even storage disks. In cognitive architectures, contents in low level memory stores only change slowly over time and are not accessed for every processing cycle. These lower level memory stores are called long-term memory in Soar. Among all the long-term memory modules in Soar, only semantic memory will be analysed in detail in this

[1]Special cases for Attribute, Value and Symbolic Constant are omitted for the sake of clarity in this general treatment

thesis. Semantic memory was chosen because it is a good case study, representative of the requirements of the other long term memories, but simple enough as a first step.

2.1.1 Semantic Memory

Soar's semantic memory is a large knowledge repository for general facts about the agent's world. These facts can be either pre-loaded to a Soar agent from a knowledge base, or gradually acquired by a Soar agent through its interactions with its environment. In either case, semantic knowledge is independent from the context of when the observation was made [24].

The knowledge representation in semantic memory is similar to that in working memory, with a long-term Identifier-Attribute-Value structure. Such a data structure can also be visualised as a table with fixed tags (fields) and some data words, as illustrated in Table 2.1. New terminologies involved in semantic memory are as follows:

- Long-Term Identifier (LTI): semantic memory indexes its objects by assigning to each object a permanent long-term identifier. To disambiguate from working memory identifiers, these are often written as a letter-number pair prefaced by a @ symbol.
- Activation: each semantic memory object has an associated level of activation stored as a number to bias the memory retrieval operation. The activation value serves as a simple multi-match resolver that, when multiple objects match the search cue, resolves the candidate with the highest activation value.

Table 2.1 A table of semantic memory elements

LTI	Attribute	Value	Activation
@A1	"Name"	"Fruit"	1
@A1	"Type"	"B1"	1
@A1	"Type"	"C2"	1
@B1	"Name"	"Apple"	1
@B1	"Variety"	"Fuji"	2
@B1	"Variety"	"Gala"	2
@C2	"Name"	"Pear"	3
@C2	"Variety"	"Nashi"	3
@C2	"Variety"	"Anjou"	1

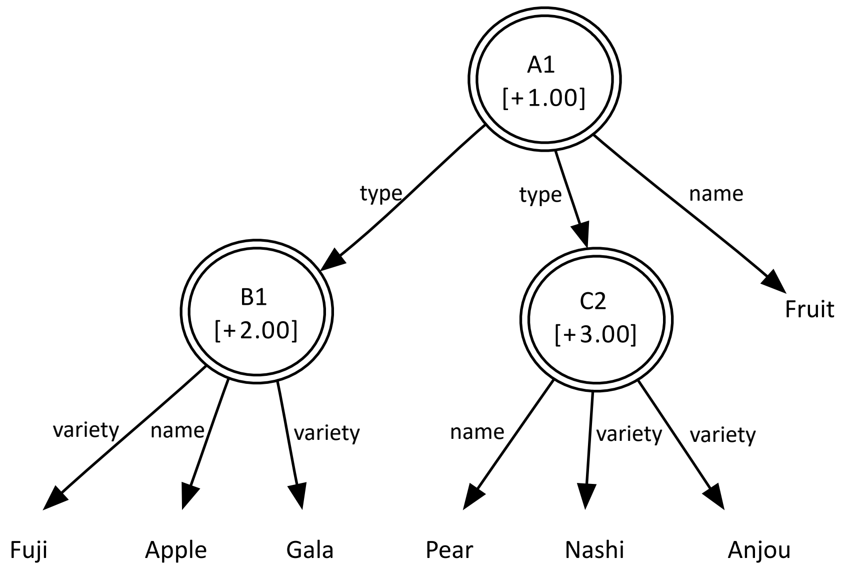


Figure 2.2 A semantic tree network containing memory elements in Table 2.1

2.1.2 Storing and Retrieving Semantic Knowledge

Working memory objects can be stored to semantic memory and semantic memory objects can be retrieved into working memory. Objects are stored and retrieved with only their immediate augmentations, i.e., one layer deep down the working memory tree. A special subtree in working memory, *smem*, serves as an interface to semantic memory. An agent can store, load or search semantic memory by adding command objects into the smem tree. Results from semantic memory also appear as objects in the smem tree.

Knowledge Store

An agent stores an object from working memory to semantic memory by adding a *store* attribute and a value to the smem tree, which creates a smem-store-value WME. The value of the WME is the identifier of the object to be saved to semantic memory, where all WMEs with this identifier are encoded and stored. If the value of the smem-store-value WME is already an LTI, semantic memory will replace and update the existing memory elements associated to this LTI; short-term identifiers assigned to the value of the smem WME will be converted to LTIs and stored as new objects in semantic memory.

Knowledge Retrieval

There are two types of knowledge retrievals from semantic memory: *non-cue-based retrieval* and *cue-based retrieval*. Knowledge retrieval in both types is initiated by creating appropriate commands in the smem tree and completed after the search operation is performed in semantic memory.

Non-cue-based retrieval: the command specifies an LTI in the smem tree. The retrieval performs a search for a semantic memory object with an LTI exactly matching the specified LTI in the smem tree. The corresponding object is then loaded into working memory; all augmentations of this object are created in working memory as new WMEs.

Cue-based retrieval: the command to semantic memory performs a search for an LTI by including search cues composed of WMEs that describe augmentations of an object in semantic memory. Different types of cue may be created by an agent when needed, and this leads to different match requirements:

- A cue WME with a constant value denotes an exact match of attribute and value.
- A cue WME with an LTI as its value denotes an exact match of all three fields: LTI, attribute and value.
- A cue WME with a short-term identifier as its value denotes an exact match of attribute, but with any value.

Note that in non-cue-based retrieval, only the LTI field is searched for the exact match, whereas in cue-based retrieval, only the attribute and value fields are involved in the search depending on the pattern of search cues.

2.1.3 Capacity Requirements

How big a memory would be sufficient in cognitive systems? Soar currently uses an optimised version of an SQLite database to store its semantic memory. The largest semantic store reported for Soar is a 400 MiB database, which contains over 820 000 structures for holding 212 000 word senses in a word sense disambiguation (WSD) task. An estimation of cognitive memory shows that a year of episodic memory is likely to accumulate as large as 42 TiB data from the agent's experience [6]. Although the required amount of knowledge to achieve human-like intelligence still remains unknown, current research efforts on semantic knowledge bases suggest that for more complex tasks that need human-level intelligence, advanced agents will need a significantly larger knowledge base than the WordNet 3 lexicon database in the WSD task or any of the existing structured semantic knoweldge bases. These include ResearchCyc, currently containing over 7 million assertions (facts and rules); Freebase, containing 23 million entries; NELL, holding 50 million candidate beliefs; and IBM's Watson, equipped with 32 servers, each with 256 GiB semantic memory.

Although there are no fixed requirements for memory capacity, the memory size is expected to be significantly larger than the current semantic store and will scale up with the increase of task complexity and the extension of applications. With this capacity requirement, the designs of physical memory structure and memory functions need to consider scalability issues; the memory cell circuits that grow with memory capacity need to have a footprint as small as possible, and the computation, communication demand and power requirements of memory functions also need to be scalable with memory size.

2.1.4 Timing Requirements

Searching a large memory to find objects that match a cue is time consuming. To formulate the problem, consider a naïve search mechanism that traverses through the semantic store, comparing a search cue with each semantic memory object, returning the object that matches all the constraints imposed by the search cue. Without a clever indexing scheme, the average time cost of this approach increases linearly with the size of the memory and the complexity of the search cue. Using a notation from [31], where E represents memory objects in the semantic store, Q represents a search cue, and a represents the average number of augmentations in a memory object, the worst case scenario is thus expressed as: $a|E||Q|$, where the letter with vertical bars represents the number of item in the set. The number of augmentations (constraints) in a search cue does not vary much and the largest cue deliberately constructed in the prior Soar experiments contains ten augmentations. Based on the capacity requirements analysed in the previous section, it is very likely that E will dominate the delay of a retrieval process in semantic memory.

The current Soar implementation uses a specialised indexing method to optimise the speed of a query to its semantic memory. Separate tables called inverted indexes, are created to store lists of semantic memory objects, sorted by augmentations; intuitively, this index inverts the forward index in which data are stored as a list of augmentations per object. Consider a simple example of semantic memory, where the memory objects and augmentations are denoted by letters A-E and Greek letters $\alpha - \varepsilon$. The indexing structure for data retrieval in semantic memory is shown in Table 2.2.

Table 2.2 Tables indexing a simple semantic memory

Forward index table	Inverted index table
A: $\alpha, \beta, \varepsilon, \delta$	α: [A, B, C]
B: α, ε	β: [A, E]
C: α, γ	γ: [C, D]
D: γ, δ	ε: [A, B]
E: β, δ	δ: [D, E]

To demonstrate a retrieval process from the simple semantic memory, a search cue is constructed, which consists of two augmentations α and ε. A successful retrieval needs to fetch any semantic memory objects that contain these two augmentations. Using Q to represent the query plan for this retrieval task, thus $Q[\alpha]$ and $Q[\varepsilon]$ refer to the object lists [A, B, C] and [A, B], currently sorted in the alphabetical order. A retrieval algorithm will successfully locate both objects A and B as candidates. The activation bias mechanism will attach a numerical value to each object and use this activation value to sort each object list. Thus, with activation value 1.23 and 1.01 for object B and A respectively, the $Q[\alpha]$ and $Q[\varepsilon]$ lists will be sorted as [B(1.23), A(1.01)]. According to the search algorithm provided, the object B will be retrieved unambiguously. The data structure behind this simplified algorithm and indexed tables is a B+ tree for speeding up the search process.

Algorithm 1 The retrieval algorithm in the inverted table

```
procedure SEARCH($\alpha, \varepsilon$)                ▷ find a memory item that has $\alpha$ and $\varepsilon$
    $idx \leftarrow 0$
    while $idx <$ sizeof($Q[\alpha]$) do
        pop $Q[\alpha][idx]$
        if $Q[\alpha][idx] \in Q[\varepsilon]$ then
            return Success!
        else
            $idx \leftarrow idx + 1$
        end if
    end while
    return Failure!
end procedure
```

By only examining the retrieval algorithm in Algorithm 1, factors that affect the query time include the size of the candidate element list in Q, the complexity of the search cue. Empirical data from the literature suggests that retrieval time tends to increase linearly with the size of candidate elements in cases where no match for the cue is found. This agrees with the observation above that in the worst case a search requires a scan through the entire list. Furthermore, more complex search cues with larger numbers of augmentations lead to more nested search loops, and hence take more time to resolve. In an experiment with Soar running on a high-performance desktop computer, the time cost is about 0.19 ms for single-augmentation cues, and is 0.5 ms for large cues with ten augmentations [31]. It is difficult to predict the impact of the size of knowledge base on the size of the candidate element list.

Using activation to bias memory retrievals comes at significant computational cost. For a retrieval, activation values for all candidate objects need to be evaluated and sorted. Also, activation values need to be updated after every operation. Existing optimisations focus on limiting the scale of activation updating scheme, the size of the sorting list, or the passive sorting operation which only operates when it is on demand by the query [5, 32]. The requirements for activation mechanisms are covered in Section 2.1.5 on page 20.

The speed of a memory system is measured in latency and throughput of tokens moving through the system. Ideally, memory access time (latency) is expected to be as small as possible. In von Neumann computers, memory access time has been a limiting factor to the overall computer performance, often referred to as the *von Neumann bottleneck* or memory wall as discussed in Section 2.2. To examine the impact of long-term memory retrieval on real-time performance of cognitive systems, it is worthwhile of looking at the operations in Soar's processing cycle and the available time budget for semantic knowledge retrieval.

Analogous to the instruction fetch and instruction execution processing cycle in the von Neumann architecture, Soar's processing cycle is called a *decision cycle*, and every decision cycle, Soar chooses and applies one operator based on the current situation and the goal as shown in Figure 2.3. At the end of the output phase of the decision cycle, semantic memory starts processing the *smem command* structure and then updates the *result* structure with results from the store and retrieval operation. Currently, multiple parallel retrievals are not supported, and only one type of retrieval is allowed to issue each decision cycle. Multiple store commands can be issued in parallel and are processed at the end of each decision cycle. However, timing constraints are less a concern for storing semantic knowledge because storing an object in semantic memory does not involve complex computations and it is expected to take a constant time as memory size increases.

A Soar agent is called *reactive* if it is able to react appropriately to changes in its environment in real-time. To remain reactive, the Soar convention is to maintain a decision cycle time of less than 50 ms [4]. Hence, a natural question to ask is how much delay budget is left for knowledge retrieval operation in semantic memory? The existing SQLite realisation of Soar's semantic memory was tested with a knowledge base containing 185 000

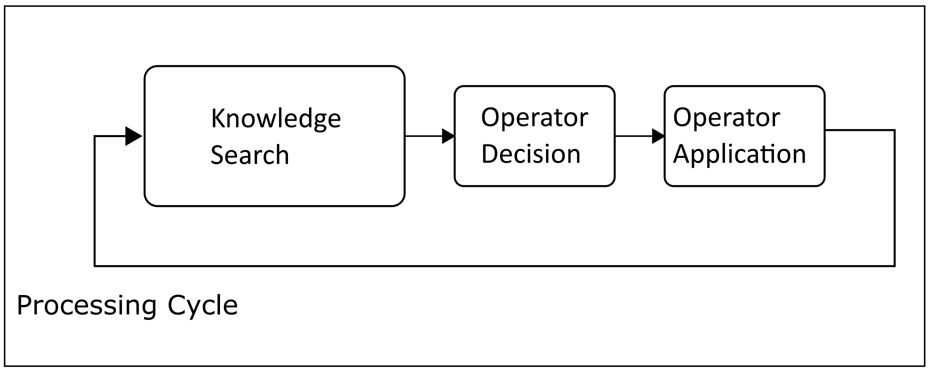


Figure 2.3 Soar's processing cycle based on knowledge search

word senses, which requries about 400 MiB of memory. The query time varied depending on the bias mechanism. The maximum query times (latency) for this experiment are reported in [22, 31], where host computers with 2.8 GHz Core 2 Extreme and 2.8 GHz Intel Core i7 were used respectively. The results varied according to the activation scheme used. For simple activation schemes, the query times were bounded by 0.9 ms; for more complex activation schemes it was about 1.34 ms; for the full-fledged activation scheme, the time taken increased to 13.25 ms, and this delay continues to grow with the size of the semantic memory. These activations schemes are covered in Section 2.1.5.

Thus, the proposed long-term memory must meet the timing requirements by allowing a real-time agent to

- make a complex decision in 50 ms,
- retrieve useful information using an effective biasing scheme, and
- retrieve an item from a semantic store containing tens of millions of items every millisecond or better.

A Proposal for a Hardware Semantic Memory Module

From the perspective of throughput, I hypothesise that more parallelism can be introduced to boost the number of retrieval outputs per unit time and hence improve the overall performance of the system. From the architectural level, the semantic memory retrieval process can be decoupled from the decision cycle, so that a real-time agent in a complex decision making process is able to retrieve an item from a large semantic store in parallel with other processes in its decision cycle. To be more specific, one may expect a hardware-based memory solution that is able to perform activation updating, sorting, and data comparison in parallel with other processes. For example, by the time a search generates a list of candidates in the retrieval process, the activation values have already been computed in memory.

Down to a lower level, a search cue with multiple augmentations can be split and passed to multiple forward indexed memory blocks; each augmentation is compared to the memory store in parallel to avoid nested search loops, and a sum operation at the second stage will determine if a match is found. Like the extra inverted tables required in database implementation, this will inevitably lead to an increase of memory space. However, it allows memory designers to trade space for speed or possibly energy savings as it is more pipelined than the SQLite database version that requires sequential iterations through candidate lists in the inverted table.

At the level of individual forward indexed memory blocks, massively parallel search can be realised by using a distributed array processing scheme instead of the centralised one. Such an array processing scheme is often referred to as *content-addressable memory*, where many data match steps occur in parallel and in place. A more detailed discussion is in Section 2.3.

2.1.5 Activation Requirements

Advanced cognitive architectures, such as Soar and ACT-R include long-term memories in which the retrieval of knowledge is biased by how recently or frequently that knowledge has been accessed. This is achieved by maintaining an activation value for every object in memory. When a memory search matches multiple objects, the one with the highest activation is returned first.

Base-level activation (BLA) is an activation scheme based on human psychology studies and it has been proved a successful part of the knowledge search process in ACT-R [33]. It has also proved a more effective memory bias mechanism than simple alternative schemes in Soar's long-term memory experiments [34, 31]. For Base-level activation in Soar, the activation B of an element in semantic memory is calculated as:

$$B = \ln \sum_{i}^{n} t_i^{-d} \tag{2.1}$$

$$t_i = t_{\text{current}} - t_{\text{access}\_i}$$

- d is a decay parameter.

- t_{current} is the current cycle measured in decision cycles since the agent began execution.
- $t_{\text{access\_}i}$ represents the time (in decision cycles since execution) when the memory element was accessed for the i-th time.

The temporal distance t_i changes in every memory cycle (a decision cycle when memory is accessed) hence the activation B for every memory elements changes every memory cycle. This imposes a computation problem since even the long past access history t_i is part of the calculation. The number of times a memory element may be accessed during an agent's lifetime is unbounded, and this is a problem for scalability.

To summarise, base level activation presents a problem for Soar because of the unbounded storage for storing every access history and the computation complexity of updating and calculating a large number of activation values.

Currently Soar offers an option to use an approximation to BLA as defined in [33]. However, even this simplified scheme still "has severe performance detriment" according to the Soar manual. A *locally efficient* updating scheme is also available in Soar [32]. It is local because only a single object is updated per retrieval by assigning a value greater than the previous largest activation value; efficiency is achieved by caching the largest activation value to avoid searching the memory store. This optimisation turns away from the faithful BLA model to a more efficient model as the activation value is mainly determined by recency instead of a collective effect of frequency and recency as in the original BLA. Chen et al. [35] observed that the computation of activation in cognitive systems is typically performed serially outside of the database. Accordingly they proposed an in-database computation scheme called *SemMemDB* that exploits the advantages of a unified database framework combining computation and storage, as well as an existing query optimiser and execution engine. Still, a significant amount of pre-computation and one-time computation is required in SemMemDB. Although these optimisations have been confirmed to be effective in some experiments, they are based on empirical studies of a small number of applications. An alternative application-independent approach, preferably hardware based, is needed to efficiently expand the application to more general uses.

This thesis approaches this problem using a hardware-based solution, with the aim to realise an approximate version of BLA that efficiently achieves semantic memory retrieval without significant deviation from the behaviour of a full implementation of base-level activation. This requires the circuit to

- reduce the frequency with which activation values are updated to save power,
- limit the number of objects to be updated,

- reduce the storage required for calculating the activation, and
- use only simple arithmetic and thereby save power, area, and storage.

2.2 Limitations of Current Memory Systems

A memory designed for a specific application needs to consider the memory access patterns and the nature of the data involved. As a long-term memory, semantic memory's storage, search, and biasing mechanisms present a particular set of requirements. This section considers how well conventional computing memory systems satisfy the requirements of a semantic memory. One important limitation of conventional memory systems is that storage is physically separated from processing. This was covered in the discussion of the von Neumann bottleneck in Chapter 1. This section covers other characteristics of current memory systems that limit the performance of semantic memory. These include the performance gap between processor and memory, the inability to provide a highly parallel computing, and the volatile nature of RAM.

In the past three decades, the performance of processors has far outpaced memory performance in terms of speed; processors have been able to make more memory requests per second, or request more bytes of data per second, than the memory can provide. For instance, the Intel Core i7 with a 3.2 GHz clock can generate a total peak bandwidth of $409.6\,\mathrm{GiB\,s^{-1}}$. In contrast, the peak bandwidth of contemporary DRAMs is only about 6 % of this bandwidth [8, chap. 2]. As shown in Figure 2.4, the access latency of DRAM has improved by only a factor of 10 over the lest 30 years. Compare this with the memory request time of a uniprocessor core, which has increased by a factor of 10,000 over the same period. Recent multi-core processors have an even higher bandwidth requirements. Memory access latency and bandwidth have been a performance bottleneck for many applications, and have led to many designs that exploit exploiting locality principles using caches. Take database systems for example. Patterson et al. [9] notes that traditional disk-based database systems on average spend up to 75 % time in the memory hierarchy. Many recent improvements have focused on the design of better caches, algorithmic improvements for better cache utilisation and cache-aware data placement [36]. However, these cache dependent improvements have diminishing returns due to cache's scalability and power consumption issues [37].

Modern computers are able to execute programs out of order and execute multiple instructions per clock cycles. However, the centralised processing nature determines that the sequential execution path remains the same: programs in execution keep track of memory

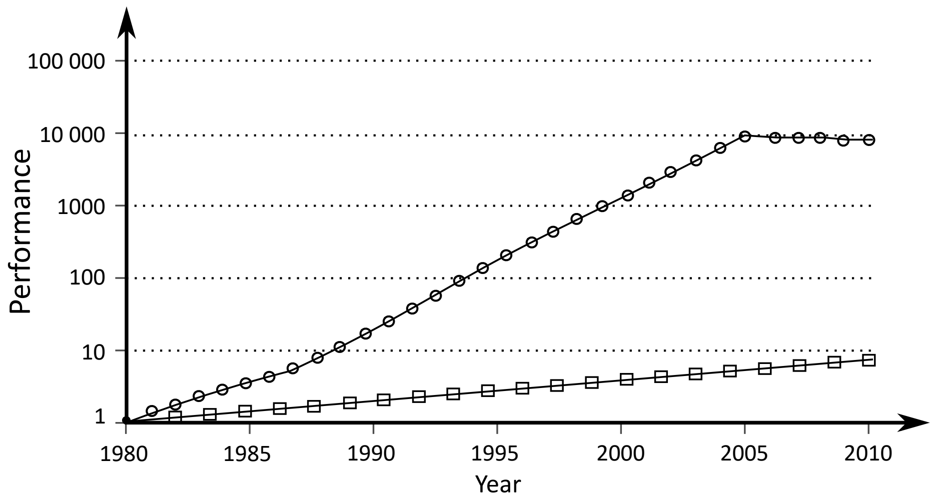


Figure 2.4 The performance gap between uniprocessor and DRAM main memory [8]; the performance gap is measured as the difference in the time between single processor memory requests and the latency of a DRAM access

locations, fetch operands, and store results. The parallelism they achieve is limited compared to the requirements of semantic memory, in which a timely retrieval requires a parallel search through memory. Each memory object, despite being connected in a semantic network, is an independent memory entity that has the same possibility of matching a search cue. Also, each object has an activation value that needs to be recomputed, updated and stored every cycle [4]. In spreading activation, communications between memory objects also occur. For a semantic memory containing millions of objects, it is very unlikely that a handful of centralised processors will be able to manage this level of parallelism through the memory access hierarchy.

Both static and dynamic RAMs are volatile memories that require power to maintain their storage. SRAMs are mainly used in caches, and account for 25 % to 50 % of total power consumption in mobile devices [8]. DRAM's large size and capacitive nature also put much stress on power budgets. An analysis of power consumption on smartphones shows that a 128 MiB DRAM can account for 10 % to 58 % of total power consumption depending on the operating frequency and applications. Modern computers devote most of their transistors to the memory system [18]. However, only a few memory locations are accessed at any given time even when a processor is operating continuously. In other words, billions of transistors are idle, and this results in a huge waste of power, as well as poor utilisation of hardware resources. These statements are especially true when it comes to long-term memories in cognitive systems. As the name suggests, these types of memories are expected to store an enormous amount of knowledge for a long time. An ideal memory for this task would retain information even without power. One possible solution that uses memristive devices as storage elements is presented in Section 2.4.

2.3 Content-Addressable Memory

This section introduces content-addressable memory (CAM), which is a hardware look-up table with built-in logic to quickly perform search operations [38].

Content-addressable memory is a specific implementation of a general class of memory known as *associative memory*. In an associative memory, data is stored, retrieved or modified based on the content itself instead of an arbitrary memory location. The concept of addressing by content has been broadly applied to both software and hardware. The best known software version is a hash table, where objects are located by addresses computed through a hash function. CAM is a hardware version, based on random access memory (RAM) technology. As illustrated in Figure 2.6 on page 25, CAM can be regarded as a table of stored data, usually unsorted. A lookup operation compares a search key against every stored data and returns the locations of any matching data. CAM provides a facility to compare a search cue with every stored word simultaneously, and then to access the words that match the cue.

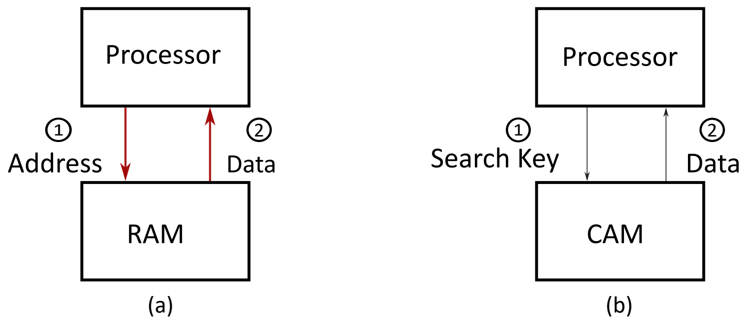


Figure 2.5 The flow of an associative search using RAM (a) and CAM (b)

To better understand the CAM search operation, it is helpful to compare it with the location-addressable RAMs as shown in Figure 2.5(a). A case in point is an associative search. For this discussion, let us assume both RAM and CAM can store the same number of words of data and both can perform a memory operation within the same clock cycle time. In a CAM system as illustrated in Figure 2.5(b), the search key is directly passed to the CAM instead of a reference. The CAM does the low level bit comparison computation and for small CAMs, a search takes only one clock cycle, irrespective of the number of data words. The search time in CAM increases only slowly with table size due to electrical effects such as increased capacitance in data lines. On the other hand, a program is responsible for finding the result in location-addressable computers. Often, the search process is iterative and successive; the algorithm relies on successive approximations, such as hierarchical hashing or binary search, before hitting the best match in the search list in RAM. Each iteration will read a new data word from RAM, which typically takes one, or more clock cycles.

To summarise a CAM's search time is independent from the table size since a CAM searches all stored data words simultaneously. This inherent nature of parallelism provides larger throughput than its software counterparts and location addressable search engines.

2.3.1 CAM Basics

A CAM consists of an array of memory cells, a peripheral circuitry, and an encoder circuitry; data words are stored in a mesh of memory cells, which is connected with horizontal *matchlines* and vertical *selectlines*. The peripheral circuitry normally contains search key registers and matchline sensing circuits. The encoder circuitry has traditionally been implemented as a priority encoder network to generate the search result, i.e. a single address where the best match located.

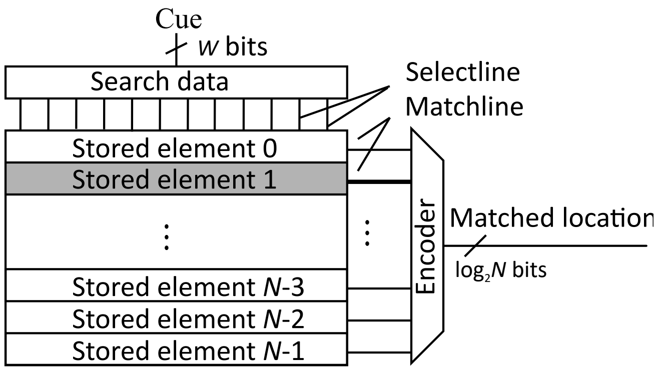


Figure 2.6 A conceptual view of CAMs

A CAM search operation is performed by driving the selectlines to a voltage level according to the search cue. Each CAM cell realises a two-input XNOR/XOR circuit with the stored bit and the search bit as the two inputs. Usually, both stored data and search data are represented in a complementary form, as shown in Table 2.3.

Each CAM cell stores a pair of complementary bits using an SRAM storage cell as shown in Figure 2.7. Within the same cell, extra circuits are also integrated for comparing the connected search bit with the stored bit, as shown in shaded areas in Figure 2.7. Two common

Table 2.3 General binary CAM encoding

stored value	stored data		search data	
	D	$\bar{D}$	Q	$\overline{Q}$
0	0	1	0	1
1	1	0	1	0

cell structures are 10-transistor NOR and 9-transistor NAND cells[2][39, 38]. NOR cells retain a high matchline voltage when stored data match the search data from selectlines; a mismatch creates a pull-down path that discharges the matchline voltage to ground. A NAND cell in bit-comparison operations turns on the matchline pass transistor in a match, and turns it off when a mismatch occurs. The matchline voltage of multiple serially connected NAND cells discharges to ground only when all cells match the search data.

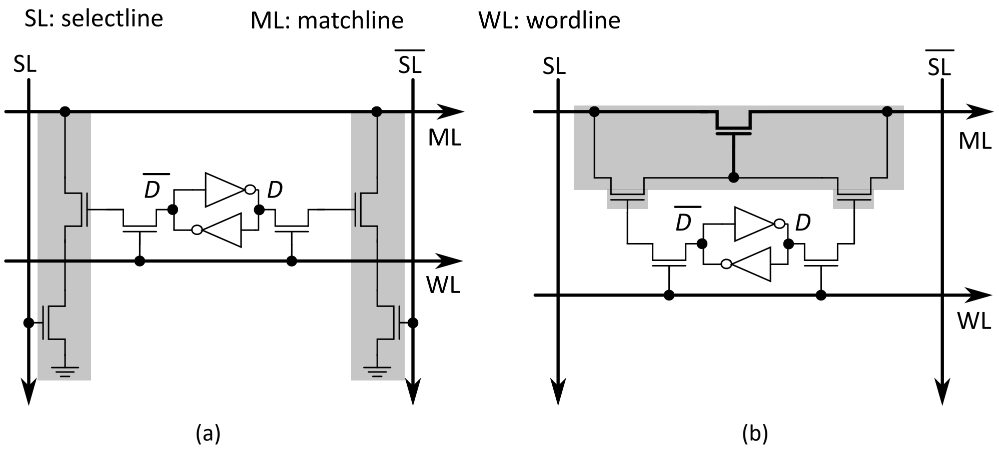


Figure 2.7 CAM cell structure based on SRAM storage: (a) 10-transistor NOR cell; (b) 9-transistor NAND cell; shaded areas show the comparison circuits, hence the pull-down paths created in a comparison operation; SRAM bitlines are omitted for simplicity.

CAMs have been traditionally designed with binary storage cells for performing exact data matching. *Ternary content addressable memory* (TCAM) is a special type of CAM that offers wildcard operations by allowing the addition of a *don't care* value X to both stored data and search data, such that each cell has three possible values: 0, 1, or X. An X cell causes a match in bit comparison, and the same applies to search data when a wildcard bit is present, as shown in Table 2.4. The TCAM in this thesis is defined to allow both stored data and search data to have wildcards [40]. The use of wildcards can be restricted to only stored data or search data [38]. It should be noted however, the cross-coupled inverters in a SRAM

[2]Note that each inverter contains two transistors

Table 2.4 General TCAM encodings

stored data			search data			search result
value	D	$\bar{D}$	value	Q	$\overline{Q}$	
0	0	1	0	0	1	match
			1	1	0	mismatch
1	1	0	0	0	1	mismatch
			1	1	0	match
X	1	1	Z	Z	Z	match
Z	Z	Z	X	1	1	match

Z can be 0, 1 or a wildcard X.

cell can not support the special encoding of X, hence two SRAM storage cells are used when representing wildcards in stored data. A TCAM that only allow the use of wildcards in the search data is not common in the literature. Such a TCAM does not require extra SRAM storage but only needs to modify the search logic driving the two selectlines which can be used to encode the three states: 0, 1, and X. An extensive coverage of CAM and TCAM operations can be found in [38].

The capability of specifying a wildcard in the search data is particularly relevant for information retrieval tasks discussed in this thesis: cue-based retrievals pass incomplete information to the memory which may be rendered as search data with wildcards for a partial match against memory data. This will be covered in Chapter 3 and Chapter 4.

2.3.2 A Review of CAM Applications

CAMs are attractive in many application domains. One of their major uses is in network routers, which are responsible for keeping track of incoming packets' addresses and the corresponding ports in a routing table with thousands of entries [41]. Another use for CAMs is in packet classification (network intrusion detection or IP filtering). Network security features often need to test incoming packets against a set of rules, which in return results in long latency to perform many string matching operations. CAMs are used to accelerate this process. Network intrusion detection systems enabled with CAMs are able to operate securely without long latency [42].

CAMs are also used to build microarchitectural components in modern computers. Palacharla et al. [43] and Buyuktosunoglu et al. [44] proposed issue queue designs using the a RAM/CAM hybrid structure, where CAMs are used for determining the required operands so that an appropriate instruction can be issued to fill up any *holes* in the pipeline due to prior-cycle issues. Translational look-aside buffers sometimes use CAMs to quickly map virtual addresses to physical addresses in current computer architectures [45]. CAMs aligned with SRAM cache-lines are used to accelerate database operations by storing a list of records into the CAM and the corresponding record identifiers into the SRAM; the CAM under search commands will quickly return the matching results, which determines the specific record identifiers [40, 46]. Building on this, CAM provides a hardware-based quick search solution in the field of data mining, where the core tasks often rely on the counting the frequency of existing terms in datasets [47].

CAMs, especially TCAMs, have also been employed to reduce power in other systems such as in the floating-point units inside GPUs, by replacing some of their operations using CMOS logic for the TCAM realization [48]. For instance, resistive RAMs (RRAMs) are used to store the result of common operations, and then consequently the re-execution of the core is replaced by just a search and read inside the TCAMs, which potentially saves power [49].

One example application in image processing is the Hough Transform computation, which has high storage requirements and computing complexity. A CAM-based hardware solution stores the Hough space parameters and performs quick search for voting scanning lines [50, 51].

FPGAs make extensive use of multiple-input-single-output lookup tables for storing arbitrary logic functions. CAM has been proposed as a space-saving alternative to current RAM lookup tables, whose size increases exponentially with the input size of the logic function [52]. For example, a Boolean equation with three variables requires eight entries in the lookup table. For equations with a few terms, most of the entries are a waste of space. Although CAM cells are about twice the size of SRAM cells, the exponential effect of the input size on the number of entries may mean that the use of CAMs brings a significant area improvement.

CAMs are also used in some machine learning approaches. Hyperdimensional computing is a computation scheme used for semantic processing. A core component of this scheme is the associative memory for finding the nearest-match data in the memory. CAM is a direct hardware implementation of the required associative memory [53]. Another machine learning technique, a Self Organizing Map (SOM) is an artificial neural network that provides a way of representing multi-dimensional input data in a much lower dimensionality for data property exploration and visualisation [54]. When an input vector is passed in the training process, every node in the network needs to compare its weight with the input data, after which the algorithm locates the node (Best-Matching Unit) whose weight best matches the input vector. This process is iterative and computationally intensive for large networks. CAMs' capability to find Hamming distance between two vectors is well suited to this task [55].

2.4 Memristors

In the past decade, a variety of resistive nanoscale data storage devices have been developed [25, 56–58]. In [56], Strukov and his colleagues suggested their device exhibited the behaviour of the theorised fourth basic circuit element, the *memristor*. Although this claim has been disputed [59], the term memristor will be used in this thesis as shorthand for any of the diverse integrated circuit structures that can be made to change their resistance by passing a current through them.

A range of resistive programmable devices at the nanoscale can be grouped into memristive devices or memristors in short. Memristive devices are typically fabricated in the wire stack of an integrated circuit using a thin film sandwiched between two metal terminals, as shown in Figure 2.8. Conductive channels in the thin film can be formed or ruptured by applying an appropriate potential difference between the terminals. For metal-oxide memristive devices, the middle layer contains a metal oxide region doped with ion vacancies and an undoped region with normal metal oxide. By injecting current through the top electrode, the charged dopants can be forced into or out of the interface region, resulting in resistive change from one stable state to another state. Often, the change to the resistive state can persist for years. In this design we have chosen to use metal-oxide memristors over phase-change devices and spin-transfer torque devices because they have lower observed switching time and energy than phase-change devices ([58, 60]) and wider resistance range than spin transfer torque (STT) devices ([60, 61]).

This feature provides mechanisms amenable to writing and storing information. In writes, for example, an appropriate positive voltage pulse V_{set} will switch the memristor from its high resistance state (HRS) to its low resistance state (LRS) to store binary value '0', whereas a large enough negative voltage pulse V_{reset} will reset the memristor to its HRS (i.e.,'1'). Being an emerging technology, no particular variety of memristor has yet become a widely deployed industry standard. Variants have different parameters to meet different design requirements.

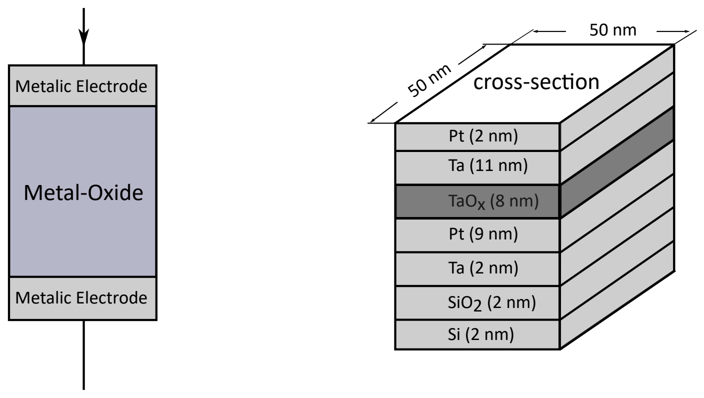


Figure 2.8 A memristor conceptual and physical structure

For the research presented in this thesis I chose to use bidirectional metal-oxide memristors [62, 63], because of their promising features and emerging popularity in published research.

Table 2.5 A Survey of metal-oxide memristive devices

	Strukov	Guan	Lee	Ho	Miao	Torrezan	Tsai
Year	2008	2008	2010	2010	2011	2011	2013
Feature size	10 nm	N/A	30 nm	9 nm	25 nm	2 µm	5.5 nm
Material	TiOx	Cu/ZrOx	TiOx	WOx	TaOx	TaOx	AlOx
Set voltage (V)	1	N/A	N/A	N/A	<2	2	5.5
Reset voltage (V)	-1	N/A	2		-2	-3.3	-3.5
Read voltage (V)	N/A	N/A	0.5	N/A	N/A	N/A	N/A
R_{HRS} (Ω)	N/A	1E+8	1 M	N/A	1E+5	N/A	>10G
R_{LRS} (Ω)	N/A	1E+2	0.67 M	>10M	1E+2	N/A	10M
Ratio (HRS/LRS)	380	1E+6	N/A	N/A	1E+3	N/A	4E+3
Switch time (ns)	100	N/A	10 n	N/A	<2 n	∼100 n	10 n
Capacitance (F)	N/A	N/A	N/A	N/A	N/A	0.12p	N/A
Write endurance	N/A	N/A	1E+12	1E+8	1E+10	N/A	N/A

A survey of memristive devices published in the literature is shown in Table 2.5. The values in this table were used to derive plausable parameters for the memristor models used for simulation in this thesis.

2.4.1 Memristor Modelling

The availability of memristor models that accurately reproduce the behavioural results in simulation is essential to memristor-based circuit designs. A simple memristor model that closely relates to the theoretical memristor proposed by [64] was published in [56]. The behaviours are determined by the following equations:

$$\begin{aligned}
v(t) &= \left[R_{LRS}\frac{x(t)}{L} + R_{HRS}(1 - \frac{x(t)}{L})\right] i(t) \\
\frac{d_x(t)}{dt} &= \mu_v \frac{R_{LRS}}{L} i(t) \\
x(t) &= \mu_v \frac{R_{LRS}}{L} q(t)
\end{aligned}$$

where, $v(t)$ is the voltage bias applied across the memristor and $i(t)$ is the corresponding current; x is the state variable of the device, L is the thickness of the oxide region, and μ_v is the dopant mobility.

In SPICE, a circuit macro can be constructed to reproduce results governed by the above equations. A typical macro for this memristor model consists of current sources and capacitors, as illustrated in Figure 2.9. Based on this, more elaborate variants that support two directional state variable motion and confine the state variable in the range $0 \leq x(t) \leq 1$ are reported in [65, 66]. Memristive effects that capture the non-linear changing rate of dx/dt are included in the model in [67]. Models that closely correlate to fabrication data from specific memristor devices are proposed in [68, 69], where only sinusoidal voltage is used. Device researchers also find the hyperbolic sine function a convenient way to capture memristive behaviours, and a generalised model optimised for repetitive pulse input is reported in [70]. I used the model captured in Figure 2.9 for simulating various circuits proposed in this thesis, as it is general enough to express its important behaviours.

Memristor capacitance is an important parameter that affects power and speed performance of a circuit. It decreases significantly with the size of the memristor. According to [58], memristor parasitic capacitance is modelled by

$$\frac{\varepsilon_0 \varepsilon_{\mathrm{r_{film}}} w^2_{\mathrm{memristor}}}{L}$$

where,

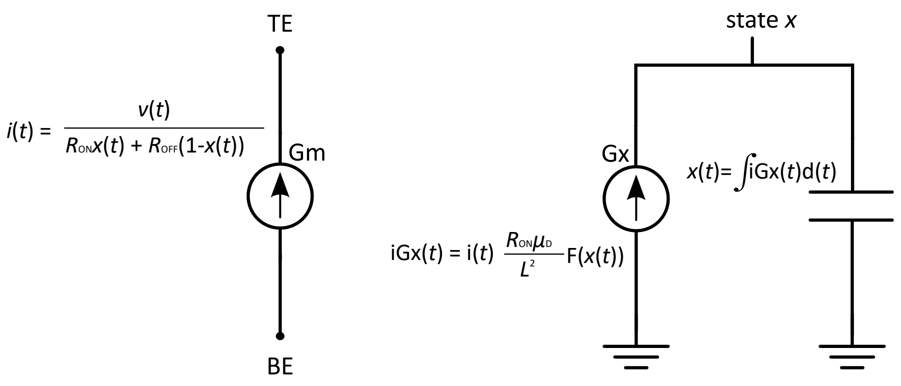

TE
i(t) = v(t) / (R_ON x(t) + R_OFF(1-x(t)))
Gm
BE
state x
Gx
x(t)= ∫iGx(t)d(t)
iGx(t) = i(t) R_ON μ_D / L² F(x(t))

Figure 2.9 Memristor model schematics that produce results in corresponding equations

- ε_0 denotes free space permittivity;
- $\varepsilon_{r_{\mathrm{film}}}$ denotes the permittivity of tantalum oxide, which is assumed 30 in the literature;
- $w_{\mathrm{memristor}}$ is the width of a cross-section on a crossbar;
- L is the thickness of the tantalum oxide layer.

For a half-pitch 50 nm $\times$ 50 nm cross-section with 7 nm tantalum oxide layer, the parasitic capacitance is about 0.1 fF. This figure is becoming smaller as memristor's feature size scales down under 5 nm [62]. Apart from capacitance, HRS, LRS, as well as the HRS:LRS resistance ratio ($R_{\mathrm{HRS}}/R_{\mathrm{LRS}}$) are key memristor parameters throughout all circuit designs in the thesis.

2.4.2 Memristors as Memory Elements

Memristors' primary advantage is their small footprint and the potential of building compact circuits, which can be used for high density memories. RAM is a case in point, and so far, it has been the most extensively studied application. Often, memristor-based RAMs are referred to as resistive RAMs (RRAMs or ReRAMs). Many RRAMs have been proposed in recent years in the literature and rapid progress has been made, as shown in Figure 2.10. These include [57, 71–82]. Many reported RRAM chips are able to store millions of bits, with two exceptions from Micron/Sony and Sandisk, which have a single chip capacity of 16 Gibit and 32 Gibit respectively.

Usually, RRAMs reuse the cell arrangement from conventional RAMs; a typical cell has an access device and a storage element. Early approaches used nMOS transistors as the access device, and such a RRAM cell is referred as the 1T-1R cell structure as shown in Figure 2.11 [71, 73, 75–78].

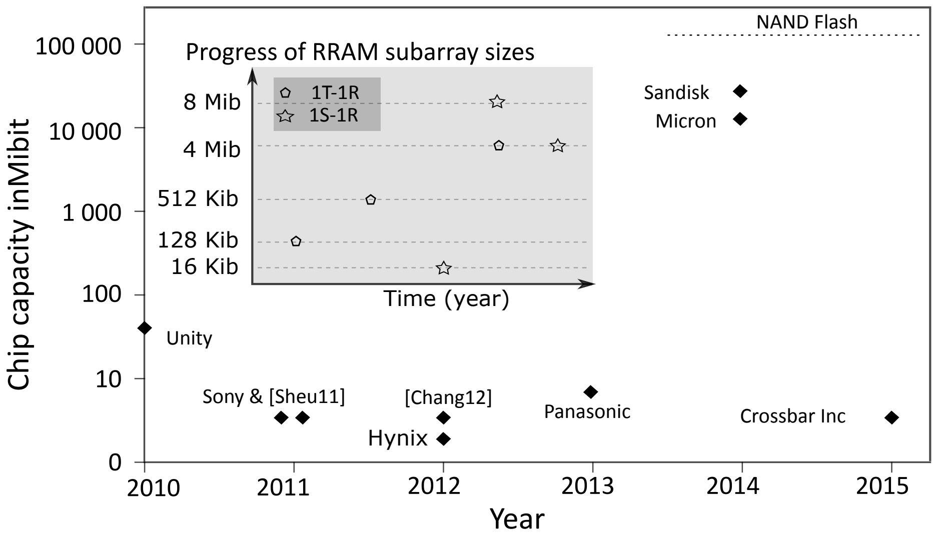


Figure 2.10 The progress of RRAM chip density in recent years

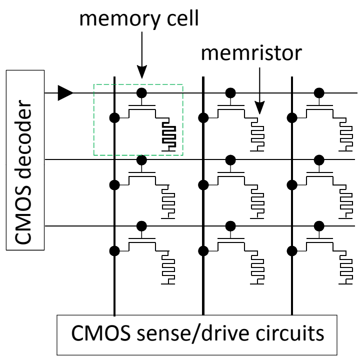


Figure 2.11 A small RRAM array composed of 1T-1R cells

To further reduce the cell footprint, research effort has been made in engineering bi-directional diode-like selector in special rectifying layers integrated with memristors, and such a cell is called the 1S-1R cell structure [74, 83, 79, 82]. As two terminal devices, memristors are suitable for nanowire crossbar architecture, in which an individual memristor in an array is activated by applying appropriate signals to its corresponding horizontal and vertical crosspoints. More details on memory operations and leakage current are covered in the introduction section of Chapter 4. Many of these proposed RRAMs are based on similar circuit principles. Generally, the claims differ in one of the following points:

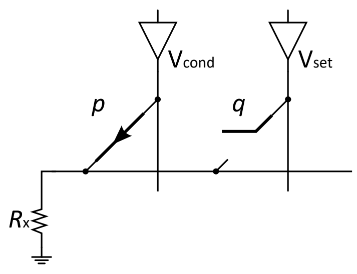
- they used different resistive devices or different switching mechanisms; or
- they proposed different access devices such as transistors or special diodes integrated with the memristor to suppress leakage current; or

- they used different memory arrangement at the array level and re-designed write and read circuits to give a high throughput in write and read.

2.4.3 Memristors as Computing Elements

The success of transistors suggests that next generation devices are likely to be deployed in both memory and computation. Although nanoscale memory is memristors' main application so far, it would be more interesting if the same memristor can serve as the basis for general computation. In fact, memristors' flexibility of being either analog or digital devices provide rich behavioural dynamics for computational purposes, which are explored in a variety of publications.

Analog designs often use a train of fine-tuned pulses to adjust a memristor's internal resistance to a specific value. Authors often claim density and non-volatility advantages over the equivalent transistor implementation with the same precision. For instance, memristors are promising for storing the large volume of long-term constants (e.g. synaptic weights) in neuromorphic systems [84–86]. Such computing schemes update a memristor's resistance by conditionally supplying analog voltage signals from its pre-synaptic and post-synaptic *neurons*. This may appear to be a variant of memristive memory, however, it is included in this review because the updating process often needs similar voltage controls as in required for computing with memristors. More details are provided in a similar analog design in Chapter 3.



$q' \leftarrow p\text{IMP}q$

input	input	output
p	q	q'
0	0	1
0	1	1
1	0	0
1	1	1

Figure 2.12 The basic schematic and truth table of IMP for realising memristive Boolean logics

Digital Boolean logic circuits are algorithmically more flexible than their hard-wired analog counterparts and are often preferred for general computations. Possibilities of realising Boolean logic in memristor arrays have been demonstrated in many publications, often with examples of some primitive logic blocks. Borghetti et al. [87] showed memristive Boolean logic can be achieved via *material implication* (IMP); IMP is executed on two variables p

and q, where p and q are encoded as memristive states. The operation $p$IMP$q$ is equivalent to the Boolean logic $q \leftarrow \overline{p} + q$. If q is cleared to zero before every operation, then IMP simply performs inversion and stores the result in the memristor q. The updated new value is denoted as q'. Hence, q here serves as an intermediate node and can be duplicated for later use. Figure 2.12 shows an IMP circuit setup of two memristors on a crossbar. V_{cond} and V_{set} are applied at the same time, where V_{cond} is an appropriate voltage that drives the horizontal common line to a specific voltage according to the state of p. Depending on the voltage of the common line, hence the state of p, the state of q will be conditionally programmed as shown in the truth table.

There are two major limitations associated with this approach: one is that for the synthesis of any multi-input Boolean functions, a lengthy sequence of voltage biasing operations is required. Typically, a n-input logic function needs $2^{n-1} + 1$ operation steps. Another limitation is that multiple output operations on a common line as shown in Figure 2.12 are impossible. This means that the sequential operations needed for computing a Boolean function can not be performed in parallel [88]. Improvements have been made by either inserting switches to divide the horizontal common line or adding a layer of CMOS logic circuits into the memory array [88, 89], which has a negative impact on the density of an array of memristors.

At the macroarchitectural level, a more complete in-memory computing scheme introduced a controller block as an add-on to the standalone RRAM array [90]. The core operation of in-memory logic computing in this proposal involved three instructions: (1) read operand at address P and convert the resistive content to voltage signal p, (2) read operand at address Q and convert the resistance to voltage signal q, (3) program or write the content at address Z by applying p and q to its two terminals. At the programming stage, the logic operation is a three input majority Boolean function, reformulated as $majority(P, \overline{Q}, Z)$, as shown in Figure 2.13. A disadvantage of this computing scheme is the necessity to perform lengthy sequences of memory read, write operations. For example, a two-input AND operation and an XOR operation need 4 and 7 steps respectively.

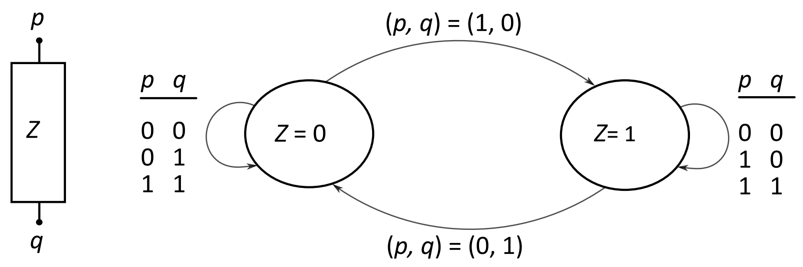


Figure 2.13 The transition of a memristor's states on conditions specified by p and q

2.5 Summary and Conclusion

This chapter introduced some background knowledge from three different research fields: cognitive architectures, content-addressable memory, and memristors.

It began with a brief study of cognitive memory systems, which leads to the understanding of the data structure of memory entities, the memory retrieval process, the associated biasing mechanisms, and current database implementations. Published experiment results based on the current implementation revealed that cognitive tasks are inherently memory-centric and requires fast search engines for high performance information retrievals in real-time. Particularly, the biasing mechanisms pose great computational challenges to complex tasks that need high-fidelity retrievals. Hardware optimised for cognitive memory systems is required to find relevant information in a large memory store.

The core component of this proposed hardware memory is a circuit block that has fast lookup capabilities. The quest for these memory circuits led us to content-addressable memories which are able to perform massively parallel search operations on all stored memory items. However, the benefits come with a cost: the conventional bit-cell is bulky, about twice as large as a static RAM cell, and it consumes much more power as will be shown in Chapter 3.

The latter part of the chapter reviewed behaviours and demonstrated features of memristors at the device level. Many different memristors have been reported recently, and are claimed to be suitable for high density, low power, non-volatile memory as well as digital logic and analog behaviours. The capability of memristors to implement both memory and logic functions is particularly interesting.

In this thesis I hypothesise that they could be used to reduce the CAM cell overhead, and implement evaluation functions in the memory such that cognitive processing features in long-term memory such as the biasing mechanisms can be realised in an efficient way.

To conclude, this chapter broadly reviewed some main concepts which are important for understanding the objectives and motivations of the work presented in this thesis. The current database implementation of long-term memories in cognitive architectures lack some capabilities that are fundamental in dealing with complex tasks that need intelligent retrievals from a large knowledge store. Content-addressable memories are examined as alternative hardware solutions. Memristors are reviewed as the device level implementation technology for realising future cognitive memories.

Chapter 3

Content-Addressable Memory Using Memristors

The core component of an information storage and retrieval system is a fast search engine. This chapter presents a content-addressable memory with high performance search capabilities to support knowledge storage and retrieval in semantic memory. Unlike the hierarchical memory system in modern computers, one can visualise it as a regular flat memory table storing rows of memory objects or records.

A system view of the cognitive memory is shown in Figure 3.1. Recall that memory systems for cognitive architectures typically have a two-tier organisation. Working memory stores information into semantic memory. More importantly, working memory initiates the retrieval process by applying top-down constraints to semantic memory. These cue constraints, sent to CAM's control registers, will serve as the search key for search operations. Once the cue constraints are transferred to the controller, it initiates search operations through the entire semantic memory. In a semantic memory containing an enormous number of memory objects across different knowledge domains, it is very likely that a less discriminative cue will match multiple objects in the memory. These match objects will activate the associated rows in the activation circuits, which use an activation scheme to bias the retrieval with the aim of returning the most relevant matching objects first. Activation circuits are the theme of Chapter 5. The search result after the biasing activation circuits is loaded to the retrieval outcome store. The same type of search operations are applicable to non-cue based retrieval since these can be processed as a simplified version of the cue based retrieval. Thus this chapter focuses on the search operations with search data specified by cue constraints.

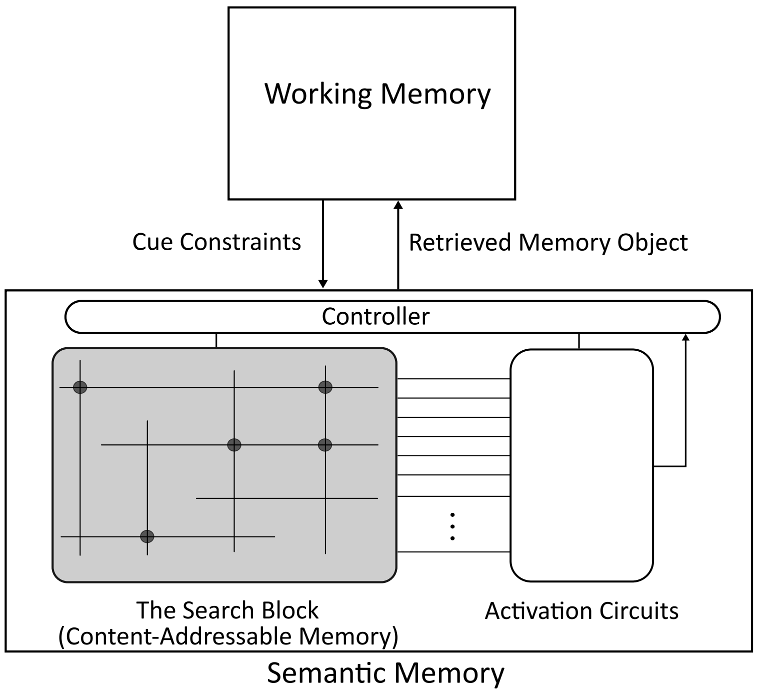


Figure 3.1 The two tier memory system in cognitive architectures with the search block in the shaded area

In store operations, working memory passes memory objects into control buffers, which are then used to store or update the memory content at an available location. Read operations are similar to the final step in a retrieval process, where a specified memory object is loaded to the outcome store for output.

The building blocks of a semantic memory are also shown in Figure 3.1. This chapter contributes to the search block using a memristor-based CAM implementation. The design uses a combination of new circuits and a series of existing techniques reimplemented using memristors to achieve a fast search engine, with which cue constraints are searched against all memory objects in parallel. Also, the memory is designed to be scalable. Scalability is critical in keeping with the primary goal of this thesis: providing efficient hardware support to large declarative memories.

The chapter is organised as follows: the remainder of this section provides a review of related work on improving CAM cells. The remaining sections describe the design of a memristive CAM that will form the basis of other experiments in the thesis. The new CAM uses differential encoding with two memristors per cell, single ended large-signal voltage signalling, and self-referencing timing signals. These design decisions are explained in the following sections. Section 3.2 describes write, read and bit-comparison operations inside memory cells. Building on top of memristive CAM cells for both CMOS and memristive technologies, Section 3.3 moves to the higher level memory array organisation and the operation of various control lines. After a complete view of the memory, the chapter provides simulations with a memory array (Section 3.4) and discussions of the design based on simulation results (Section 3.5).

3.1 Related Work

Memory cells are unit devices in any memory systems for storing binary values. Much effort in the literature has been made to improve CAM cells at the circuit level. Hence it is worth briefly mentioning the inefficiency of CAM cells here. A CAM cell has two functions: storing a single bit and performing bit-comparison between the stored bit and the corresponding search bit. The search performance of CAMs using CMOS technology comes with a great price, as shown in Table 3.1. A typical CAM cell contains a six-transistor SRAM to store two complementary binary values. In addition, four transistors are usually required per cell to support the parallel search. A TCAM cell requires two SRAM storage bits to represent a wildcard. A rule of thumb is that CAMs are about twice the size of SRAM. This area overhead limits the maximum number of CAM cells that can be packed on a single chip [38]. Search operations in CAMs involve all cells in the process, which leads to considerable power consumption. This is another limiting factor for building large CAMs. The capacitive control lines used for a search, including matchlines and selectlines connect to thousands of bits, and are precharged and discharged almost in every search cycle. As a result, the area and power overhead of CMOS CAMs have limited their applications to a small number of timing critical applications with modest storage requirements [40].

Table 3.1 A comparison of CMOS memory technologies with some key parameters in three different memory types according the the report in [91] from 2010.

Memory	Capacity/chip (Mibit)	$/Mibit	Access speed (ns)	Watts/Mibit
DRAM	128	0.08 - 0.16	40 - 80	0.008-0.016
SRAM	9	5.5 - 7.8	3 - 5	0.17 - 0.33
TCAM	4.5	44.5 - 66.7	4 - 5	3.33 - 4.44

Related work on CAM design has sought to reduce the area and power overhead by using more compact cell structures and new power efficient search schemes. This section reviews a series of CAM cells proposed for area and power reduction. Details of power efficient methods are included in Chapter 4.

Hanyu et al. [92] proposed a two-transistor (2T) CAM cell structure based on floating-gate MOS technology. Despite the bit-serial search operation, the cell is able to store multiple values by carefully tuning the input voltage and gate threshold voltage. Arsovski et al. [93] proposed a TCAM cell structure that uses half-latches in places of the cross-coupled inverters. In recent years, researchers working with unconventional technologies, including phase-change devices and spin-torque transfer devices (STT) and memristive devices, have observed behaviours that they claim will be useful for reducing CAM area and power overheads.

Area : the area of a CAM can be reduced by replacing the SRAM storage element in each cell with an alternative storage device. Eshraghian et al. [39] used two memristors as storage elements to reduce cell area by almost half. CAM cells can be made even smaller by simplifying the comparison circuit: [94] proposed a similar 2T-2STT circuit, with two STT devices added in series with the access transistors. A similar structure that enhanced the sensing abilities using phase-change devices can be found in [40, 95]. An extreme example in area reduction is illustrated in [96], where each CAM cell only contains a single memristor. Without the complementary data representation, this compact scheme is only able to detect a mismatch when search data is ‘1’ and stored data is ‘0’. However, it claims to be appropriate for special applications with dense data representations.

Power : the claims of power reduction with emerging devices are usually based on the fact that the energy required to switch a phase-change device, STT or memristor (measured in J/bit) is less than or comparable with the energy required to switch a DRAM cell [62]. Also, these three emerging devices does not need power to maintain stored information hence zero stand-by energy consumption is achievable. This can lead to substantial power savings in large memory systems. Recent study has demonstrated extremely low-voltage operations in read mode [97], which significantly saves power in memories built mainly for read purpose.

3.2 Memory Cell

The cell structure in this work is primarily based on the structure proposed in [94] where the STT device with an undesirable low resistance ratio (<10) between the two states restricted the number of cells that can be attached to a matchline hence the memory array size. Recent advances in memristors provide an opportunity to revive the structure; memristors are used to replace the STT devices to improve the memory width and sensing margin.

The four-transistor two-memristor (4T-2R) cell in this new CAM (Figure 3.2) uses two memristors for storing a pair of complementary binary values, and four transistors for performing comparisons and facilitating read operations.

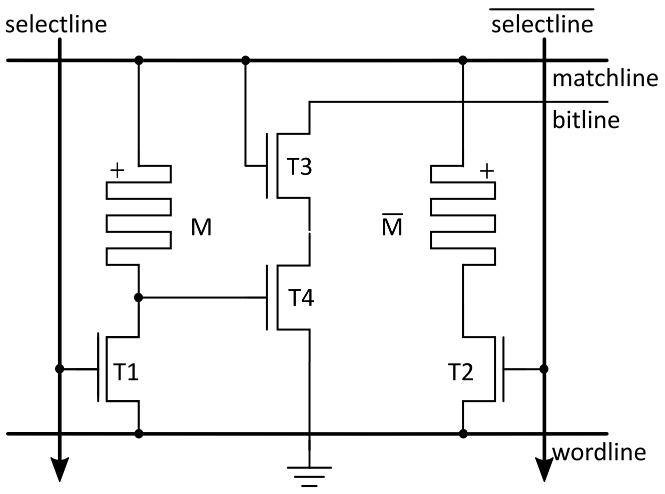


Figure 3.2 Memory cell structure

3.2.1 Write

Write operations use the memristor's electrical properties where conductive channels in the thin film can be formed (SET operation) or ruptured (RESET operation) by applying an appropriate potential difference between the terminals. Conventionally, CAMs have a write port with additional access transistors and wires for write operations, as previously shown in Figure 2.7 on page 26. To avoid these facilities that unduly increase the cell footprint, it is necessary to write multiple cells of a row in bulk. When a row is selected for write, signals along selected columns turn on the connected nMOS transistors in the cell such that the voltage across the in-series memristors is V_{set} (or V_{reset}), which is large enough to alter the memristers' states to LRS or HRS. However, as SET and RESET share a common matchline, writing cells along a matchline is necessarily a two-step operation.

Two write sequences are adopted in the literature of CAM design. Take a row of four cells for example. One way to write the row is to reset the specified columns to '1' so that a four bit row will have a pattern of 'X1X1', with 'X' representing unspecified columns, which will be programmed in the second step to give the result of '0101'. Another way to write a memory row is to reset (ERASE) all the cells along the row to '1' such that the row in example becomes '1111'. A second step sets the specified columns and produces the required result ‘0101’. The advantage of this ERASE-before-SET process is that the ERASE operation can be performed at the same time as waiting for the column signals.

The write operation in this new CAM adopts the ERASE-before-SET method, where ‘0’s and ‘1’s are written into each row in the following sequence:

1. Erase the row – reset all to ‘1’ by driving matchline and wordline and activating all selectlines to close the transistors in series with memristors, so that the voltage across the selected memristors is approximately V_{reset}.
2. Set columns – store ‘0’s to specified columns by driving the matchline and wordline and activating the specified selectlines to ensure the voltage across the selected memristors is approximately V_{set}, which is of opposite polarity to V_{reset}.

3.2.2 Read

Normally, memory objects are stored once and read many times. This requires read operations to be fast, reliable and power efficient. Small signal sensing schemes with differential sense amplifiers are widely used in large RAMs [98]. With the advent of nanometer processes, it is becoming difficult to bias transistors at low voltages, and often more engineering efforts are required in the design of power efficient analog sense amplifiers [99].

The conventional small signal sense scheme (Figure 3.3 b.) in which bitlines are connected directly to the cells, is not well suited to the designs considered in this thesis. In a memory with a large number of rows connected directly to the bitline, a significant amount of current will flow to the inactive matchlines from the activated matchline. Thus, a read operation on a specified matchline will charge all the matchlines. This leads to unnecessary dynamic power consumption. One may propose to swap the positions of the memristor and nMOS transistor so that the nMOS can serve as an isolation transistor between bitline and inactive matchlines. However, this change will result in pull-down paths through memristors to the ground.

There is a trend to favour large signal or single-ended sensing schemes in nanometre processes to tolerate process variations and low power operations [100]. Single-ended sensing schemes usually use full-swing signals compatible with logic-level operations. This makes them suitable for building hierarchical sensing networks which are discussed in Section 3.3.3.

The design in this chapter uses a single-ended voltage sensing scheme for read operations, as shown in Figure 3.2 on page 41. To read a row its matchline is driven high so that T3 is closed, and the bitlines are precharged high. The memristor M and nMOS T1 form a voltage divider. Depending on the memristor's resistive state, the resultant voltage will either open or close T4 to either leave the bitline high or pull it to ground.

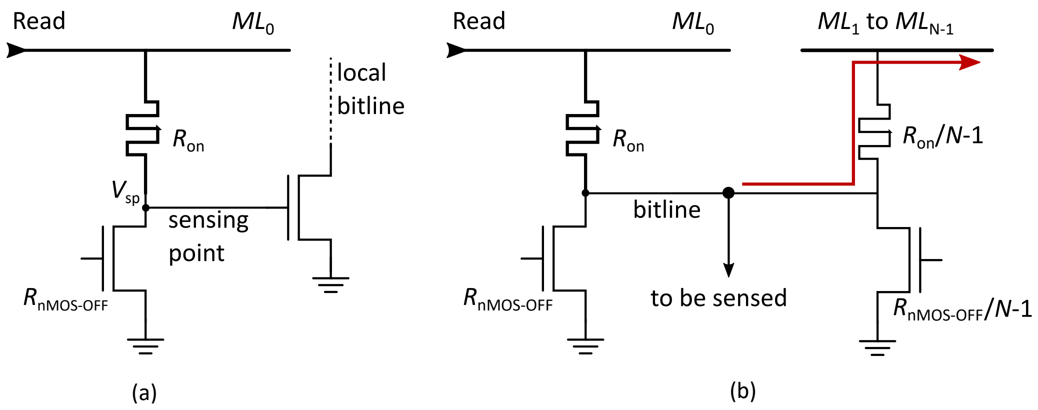


Figure 3.3 Two sensing schemes: (a) single-ended large signal sensing using a voltage divider in the cell; (b) small signal sensing by directly connecting a column of cells to a bitline

As modelled in Figure 3.3(a), a voltage sensing scheme is used for the read. R_{on} represents a memristor's resistance in the LRS state and $R_{\text{nMOS-OFF}}$ represents the equivalent resistance of an nMOS transistor in its OFF state. These two resistors form a voltage divider that converts memristor's state to a valid voltage signal. Unlike the sensing signal in the small sensing scheme, the single-ended sensing signal is gated by a transistor so that each cell only sees a small amount of capacitance. The isolation provided by the transistor also prevents the sensing current flowing into inactive matchlines.

Ideally, the voltage at the sensing point (V_{sp}) is expected to be high for sensing memristor LRS and otherwise a low V_{sp} for sensing HRS. Therefore to get a large sensing margin, it is important to consider R_{nMOS} in this voltage divider. Here, I define the read sensing margin as the difference of V_{sp} levels in sensing two memristor states. The maximum sensing margin can be obtained when the following constraints are met:

$$\frac{R_{\text{nMOS-OFF}}}{R_{\text{HRS}}} \approx 0$$
$$\frac{R_{\text{LRS}}}{R_{\text{nMOS-OFF}}} \approx 0$$

Assuming that the above equations are equal to the same value close to zero, they can be reformulated as follows, with R_{HRS} represented as $\mathcal{R} \times R_{\text{LRS}}$,

$$R_{\text{nMOS-OFF}} = \sqrt{R_{\text{LRS}} \times R_{\text{HRS}}} \tag{3.1}$$

This suggests that for a typical 45 nm process nMOS (width/length = 2/1, $R_{\text{nMOS-OFF}}$ = 1.6 GΩ) and a baseline ratio $\mathcal{R}$ = 1000, R_{LRS} and R_{HRS} need to be 50 MΩ and 50 GΩ respectively. However, the equation used is only useful for analysing static behaviours. Dynamic behaviours need to be considered separately.

The high resistances of LRS and HRS introduce delays to the development of V_{sp}. The first order RC delay suggests that it takes time t_{pd} ($t_{\text{pd}} = R_{\text{memristor}} C_{\text{sp}} \ln 2$) for V_{sp} to rise to the valid voltage level. C_{sp} is the sensing point capacitance. Based on gate and diffusion capacitance extracted from the 45 nm FreePDK model, the time taken to develop a valid V_{sp} is approximately 10 ns for reading an LRS memristor (i.e. 50 MΩ as calculated). This is considerably slower than the read speed in RAMs.

One observation is that sensing an LRS memristor is faster than sensing an HRS memristor. The sensing time of LRS memristors will set the overall sensing time frame for both LRS and HRS because the V_{sp} for HRS memristors does not have enough time to develop a valid voltage and thus remains low throughout the sensing time. Therefore, it is safe to choose a memristor with low LRS resistance to speed up the sensing operation. This observation greatly relaxes the resistance constraint expressed in the equation 3.1. The selection of R_{LRS} and R_{HRS} is important in this new memory as it impacts sensing speed in read, matchline sensing margin in search as well as energy performance which are discussed in more detail in Section 3.3.1 and 3.4.2.

3.2.3 Bit-Comparison

CAMs' parallel search comes from their capability to compare a pair of bits in each memory cell. This bit-comparison function of each cell is implemented using a compact comparison circuit which accepts input from the search data and the stored data in the cell. This section describes a comparison circuit based on memristors and transistors.

This complementary arrangement of two memristors is a typical CAM cell structure that creates a short path between matchline and ground whenever there is a '0-1' mismatch or '1-0' mismatch situation as examples shown in Figure 3.4. Table 3.2 provides the complementary encoding for the bit-comparison operation. Specifically, in the case that a selectline with zero voltage level fails to create a short circuit through a storage memristor (i.e. fails to turn on the transistor in series with the memristor), its complementary selectline will ensure the presence of a short circuit path through the complementary memristor if there is a mismatch.

A CAM search operation is performed by setting the selectlines equal to the search cue. Each CAM cell realises a logic equivalent to a two-input XNOR/XOR circuit with the stored

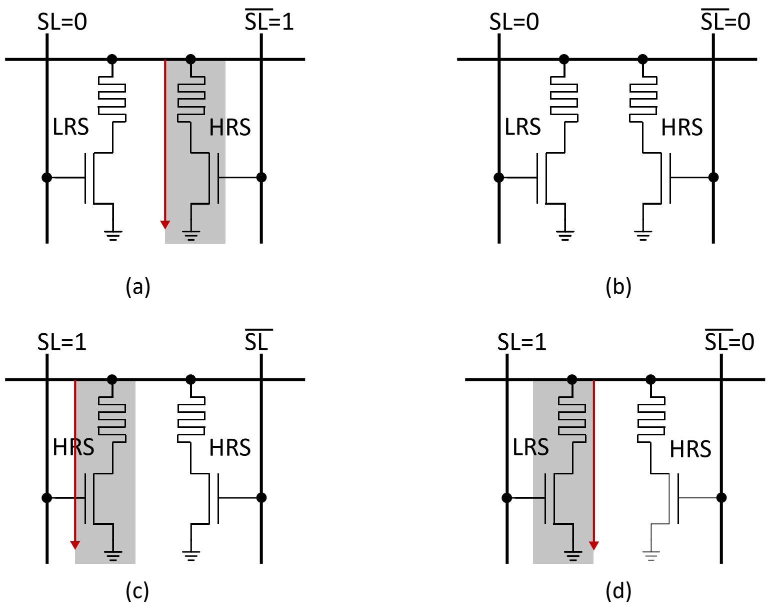


Figure 3.4 Examples of match (a, b, c) and mismatch (d) situations in bit-comparison operations, with discharging paths highlighted in the shaded areas

bit and the cue bit as the two inputs. The matchline-to-ground resistance, $R_{\text{ML-GND}}$, varies with the different input conditions. Consider, for example, the case where both the stored bit and the cue bit are '1' i.e. a match situation. $R_{\text{M}} = R_{\text{HRS}}$ and $R_{\overline{\text{M}}} = R_{\text{LRS}}$. The selectline is driven high to close nMOS N1, and the $\overline{\text{selectline}}$ is driven low to open nMOS N2. The result will be $R_{\text{ML-GND}} \approx R_{\text{HRS}}$. However, in a mismatch situation, one of either the M-N1 or $\overline{\text{M}}$-N2 paths will have a low resistance close to R_{LRS}, which will effectively pull the matchline down.

Table 3.2 4T-2R cell encoding

Stored Data		Search Cue		Match	
Logic	$(D, \overline{D})$	Logic	$(Q, \overline{Q})$	Status	ML
0	(0, 1)	0	(0, 1)	Match	V_{read}
		1	(1, 0)	Mismatch	0
1	(1, 0)	0	(1, 0)	Match	V_{read}
		1	(0, 1)	Mismatch	0
X	(1, 1)	0	(1, 0)	Match	V_{read}
		1	(0, 1)	Match	V_{read}

3.3 Memory Array Organisation and Operations

In many ways the new CAM is organised like a conventional CMOS CAM: memory cells in an array arrangement are connected using various control lines as shown in Figure 3.5. A global precharge control signal charges all matchlines to a high voltage to initialise the search operation, or instead it changes a specified matchline for write and read operations. Matchline sense amplifiers are used to discern match rows from mismatch rows. In addition to the conventional memory array organisation, reference rows are adopted for the generation of timing signals. Another difference is the bitline arrangement which is discussed in more detail in 3.3.3.

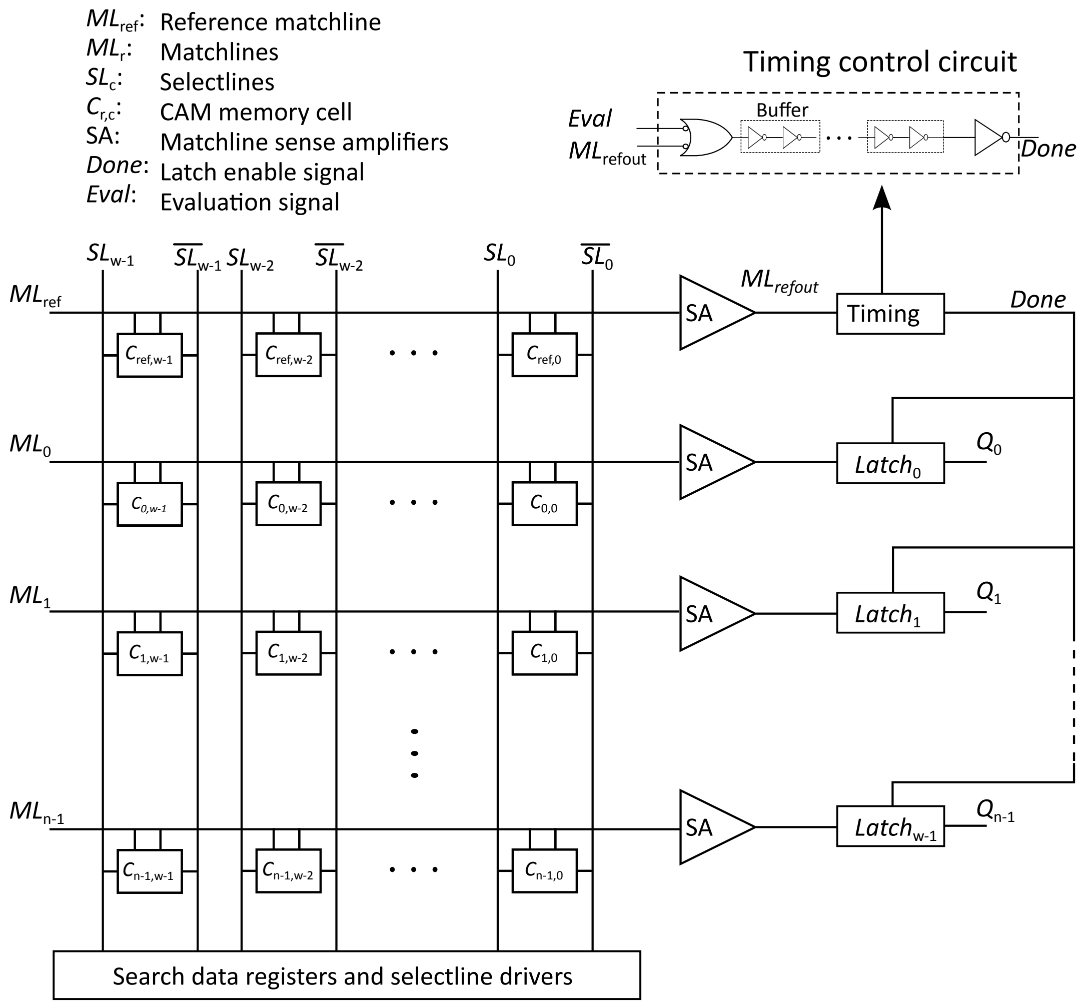

ML_{ref}: Reference matchline
ML_r: Matchlines
SL_c: Selectlines
$C_{r,c}$: CAM memory cell
SA: Matchline sense amplifiers
$Done$: Latch enable signal
$Eval$: Evaluation signal
Timing control circuit
Buffer
$Eval$
ML_{refout}
$Done$
SL_{w-1} $\overline{SL}_{w-1}$ SL_{w-2} $\overline{SL}_{w-2}$ SL_0 $\overline{SL}_0$
ML_{ref}
ML_0
ML_1
ML_{n-1}
$C_{ref,w-1}$ $C_{ref,w-2}$ $C_{ref,0}$
$C_{0,w-1}$ $C_{0,w-2}$ $C_{0,0}$
$C_{1,w-1}$ $C_{1,w-2}$ $C_{1,0}$
$C_{n-1,w-1}$ $C_{n-1,w-2}$ $C_{n-1,0}$
SA
ML_{refout}
Timing
$Done$
$Latch_0$ Q_0
$Latch_1$ Q_1
$Latch_{w-1}$ Q_{n-1}
Search data registers and selectline drivers

Figure 3.5 CAM matchline organisation with the reference row

3.3.1 Search Operations

The search operation looks for rows that match a cue word by comparing the cue against every row in the memory simultaneously. The following steps are performed:

Reset: search results are cleared for the new search

Precharge: matchlines are charged high

Evaluation: selectlines are set to the value of the cue

During the evaluation step, matchlines discharge at different rates according to the number of bits that match in their rows. For rows where one or more bits do not match, the matchlines are pulled low. Matchlines for rows that do match also discharge, but more slowly via memristors in the high resistance state. An example to illustrate the exact match and one-bit mismatch situations is shown in Figure 3.6 on page 47; transistors and selectlines are omitted for simplicity, and LRS memristors with shaded areas are in series with turned-off transistors. Say the four-bit search data is ‘1010’ and this is used to drive corresponding selectlines. Recall that the complementary (HRS, LRS) represents ‘1’ in a storage cells and (LRS, HRS) represents ‘0’. When searching against the search data, cells along matchline ML_i will have exact matches which produces four highly resistive pull-down paths between the matchline and ground. For matchline ML_j, $Cell_{j1}$ does not match the search bit ‘1’ along the column thus turns on a low resistance pull-down path to the ground.

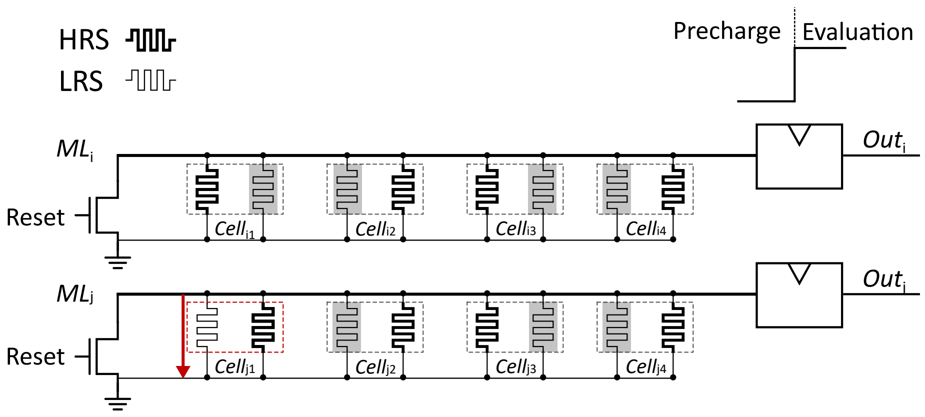


Figure 3.6 A characterised cell circuit illustrating the discharging path in search operations

The generation of correct results depends on precise control of the time at which the matchlines are sampled: if the sampling fires too early, matchlines may not have developed enough voltage difference to be differentiated; if it fires too late, all matchlines are discharged to ground. A *sensing window* in search is defined as the the difference of discharging delays during the evaluation stage between an exact match row and a one-bit mismatch row. Sampling is triggered by the *Done* signal illustrated in Figure 3.5. It is critical the *Done* signal arrives during the sensing window shown in Figure 3.7 so that a one-bit mismatch can be discerned from an exact match. Rows with multiple mismatch cells will discharge much more quickly and are not a concern.

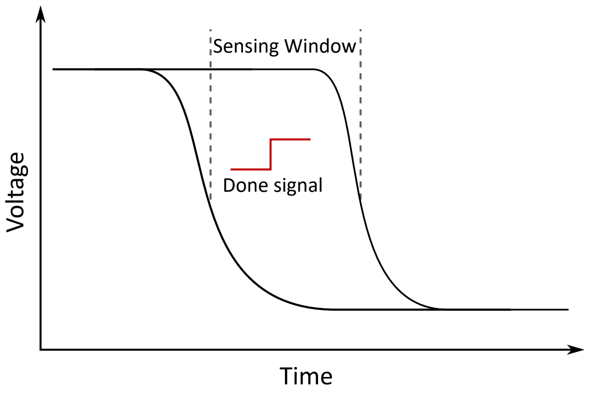


Figure 3.7 An illustration of matchline sensing window and its timing with the *Done* signal in a search cycle

Matchline Sensing Margin

The matchline sensing margin is defined as the voltage difference between a one-bit mismatch and an exact match at the time at which the matchlines are sampled. As stated in [61], the sensing margin is controlled by matchline to ground resistance ($R_{\text{ML-GND}}$).

$$\frac{R_{\text{match}}}{R_{\text{OneMiss}}} = 1 + \frac{R_{\text{HRS}} - R_{\text{LRS}}}{W}(R_{\text{LRS}} + R_{\text{nMOS-ON}})$$

Where, R_{match} is the $R_{\text{ML-Gnd}}$ of an exact-match. R_{OneMiss} is the matchline resistance of one mismatch bit. R_{HRS} is a memristor's resistance in its HRS state and R_{LRS} is its resistance in the *LRS* state. $R_{\text{nMOS-ON}}$ is nMOS's resistance in *ON* state. The W term represents the number of bits attached to a matchline.

Based on this equation, it is observed that the matchline sensing margin is affected by the following factors:

- memristors with large $R_{\text{HRS}}/R_{\text{LRS}}$, denoted as $\mathcal{R}$, improve the sensing margin; and
- increasing the memory width, W, decreases the sensing margin.

It should be noted that the closed nMOS resistance R_{nMOS} (about 5.4 kΩ in the 45 nm process) in series with the memristor sets the lower bound resistance of the pull-down path, which can undermine the benefit of the large sensing margin brought by memristor's inherent resistance ratio. To be more specific, the non-zero R_{nMOS} will be part of the effective resistance ratio which is defined as $\mathcal{R}_e = R_{\text{HRS}}/(R_{\text{LRS}} + R_{\text{nMOS}})$. Thus, the presence of this closed nMOS resistance undermines the already precious resistance ratio $\mathcal{R}$ in real memory operations. The impact of $\mathcal{R}$ is illustrated in Table 3.3, where memories based on memristors with higher $\mathcal{R}$ show more tolerance to memory width scaling.

Table 3.3 Comparison of sensing margin with various memory widths

		Width (bits)			
LRS-HRS(Ω)	$\mathcal{R}_e$	32	64	128	256
100-100K	18	0.17V	0.10V	0.05V	<0.01V
1M-1G	1000	0.50V	0.50V	0.49V	0.36V

Memristors provide a wide range of LRS-HRS pairs, including the possibility of large $\mathcal{R}$ which are observed in a range of metal oxide memristors as reported in publications(up to 1E6) [56, 101, 102]. Larger $\mathcal{R}$ allows CAM researchers to have better design trade-offs between matchline width and sensing margin. These will be further evaluated in Section 3.4.

3.3.2 Reference Matchline

How does the memory know when it is the appropriate time to generate the *Done* signal? A static method to achieve this is by characterising the electrical properties of the match and mismatch rows so that the correct time can be determined by checking the difference of their RC delays in discharging matchline voltages. However, the fabrication of nanoscale devices introduces device variations which reduce the accountability of static circuit characterisations [103].

In each CAM array, one extra row called reference matchline (ML_{ref}) is added to minimize the uncertainty of the arrival time of matchline output caused by device variations. This idea is similar to the use of *replica* cells and bitlines in RAM column circuits to match the delay of the access path in a read sensing operations so that the sense amplifiers know exactly when to start sensing. Here, ML_{ref} contains the replica cells and is deliberately designed to have one mismatch bit by flipping one bit from the search data. During matchline search, it sends an enable signal (*Done*) to every row (matchline) within the memory array, with the purpose of differentiating the exact-match from all mismatch rows.

3.3.3 Bitline Organisation and Operations

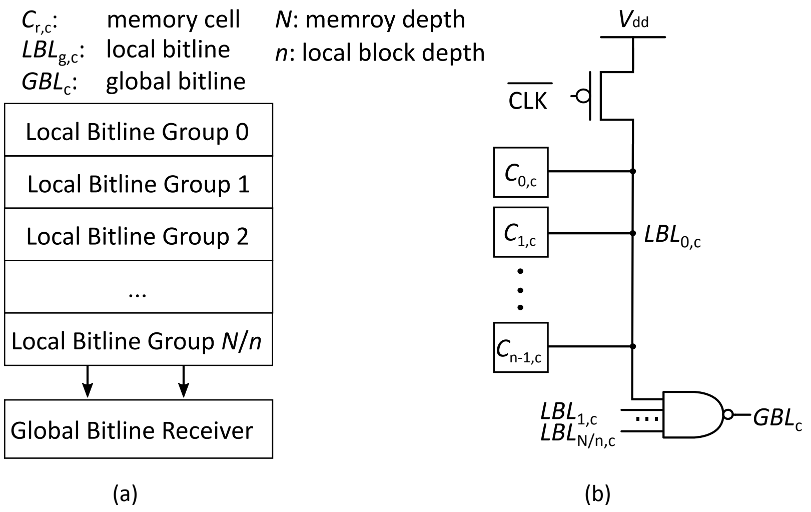


Figure 3.8 Hierarchical bitline illustration

So far, matchlines and selectlines are introduced to serve in write and search operations. One may consider reusing these control lines for read operations as well. Selectlines are isolated from memristors by access transistors thus are unable to read the storage state out of a cell. As briefly mentioned in [40], however, it is possible to sense a cell's state via the matchline by reusing the matchline sense amplifiers. This is essentially the same as a search operation with the exception that only a single matchline and a single cell attached to the matchline is involved. Since all cells in the row share the common matchline, the read operation has to be bit-serial; a matchline with W cells needs to bit-serially cycle through all selectlines to read the entire row into the outcome buffer. The design in this chapter looks for a read scheme that can read out the cells of a row in parallel.

As analysed in Section 3.2, small signal sensing schemes directly connecting to the memory cells will charge unselected matchlines and thus are not suitable for this memory design. To address this problem and to build a deeper memory, I propose to use a hierarchical approach with a large signal sensing scheme, in which memory rows are grouped into local blocks as illustrated in Figure 3.8 [100]. Each local bitline can be functionally viewed as an unfooted dynamic multiplexer comprised of access and drive transistors for each cell. With additional access transistors, each cell only sees a small capacitance. The global bitline network can be viewed as a network that collects data from the groups of local bitlines and produces the final sensing result.

The local bitlines are charged to high voltage before reading. During a read operation, the selected matchline will switch the access nMOS on. The local bitline will either be discharged or maintained depending on the state of their corresponding cell. To alleviate the non-zero voltage noise from unselected matchlines, non-minimal nMOS length transistors are used to ensure that unselected access nMOS will not be unexpectedly turned on, as shown in Figure 3.9. For the 45 nm process with V_{th} = 0.28 V and V_{dd} = 1 V, V_{sp} is arranged to be 1 V for sensing LRS memristors and 0 V for sensing HRS memristors. The presence of a LRS bit in read will discharge the local bitline to ground and pass the signal to the global circuit. The global bitlines can be implemented as dynamic OR gates driven by local bitline outputs.

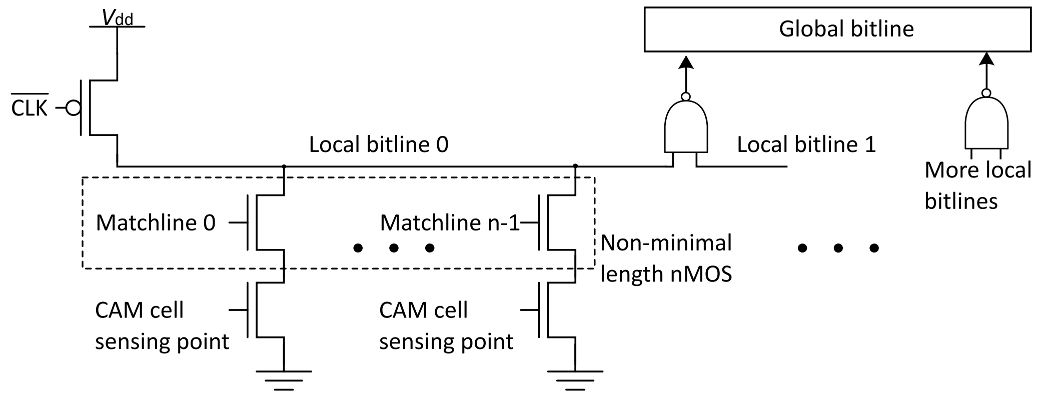


Figure 3.9 Schematic of the hierarchical bitline network with access transistors of a cell

3.3.4 A Summary of Design Decisions

In summary, the preceding sections have explained the design of a new memristive CAM in which the memristors are incorporated into the wire stack of a 45 nm CMOS process.

It is found that memristors' internal resistance and the memory width determine the matchline sensing margin. With a fixed memristor resistive ratio and memory width, running a resistance sweep will be able to find a pair of resistance that gives the best sensing margin (see Section 3.4). A reference matchline is included in the CAM array to provide a self-referenced timing signal which is able to discern the best match from mismatch rows. A large swing single-ended voltage sensing scheme is used to address the issue of leakage current in small sensing schemes. The single-ended sensing scheme is further developed to a hierarchical bitline structure in which sensing results are collected from local bitlines and the logic-level full voltage swing ensures a flexible digital implementation.

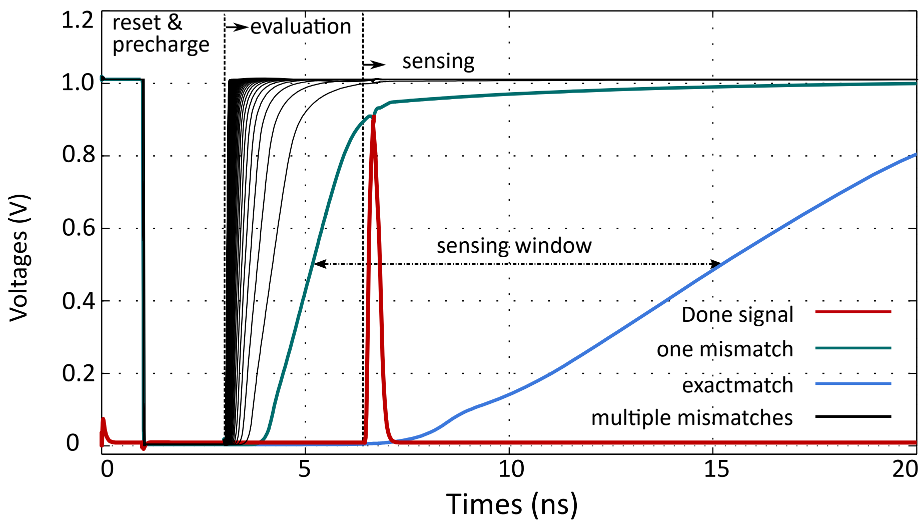


Figure 3.10 Search performance of A 64 × 128 array

3.4 Memory Array: Experiments and Analysis

To gain an insight of circuit performance with different memristor parameters, the new CAM has been simulated at different sizes in SPICE with memristor characteristics reported in [57, 74, 102], and 45 nm transistor models from FreePDK [104]. Simulations to characterise the 45 nm transistor models show the gate capacitance and diffusion capacitance per micron are 0.74 fF and 1.23 fF. Each memristor is shunted with a 0.1 fF capacitor in simulations to account for the memristor internal capacitance. As well as showing the performance of the new CAM with different memristors, the simulations in this section provide an indication of the new CAM array's scalability and energy consumption.

3.4.1 Scalability of The New CAM Array

Search and read operations are affected by memory depth. Write operations work on a specified row in a memory array and hence are not discussed here.

All the simulations of search operations reported here are arranged such that there is one row with an exact match and one row with a one bit mismatch. All other rows use randomly initialised data with an average number of 64 mismatches. Also, a baseline LRS-HRS of 1 MΩ-1 GΩ is used. Figure 3.10 shows that the sensing window is over 10 ns. This large sensing window, compared to the reported 250 ps in the literature [105], is necessary; it allows the memory to broadcast the refence signal to more rows. A characterisation circuit accounting for wire RC delay and load capacitance from each row indicates that 10 ns is sufficient for the reference timing signal *Done* to reach about 1000 rows with a unit drive and four repeaters along the wire.

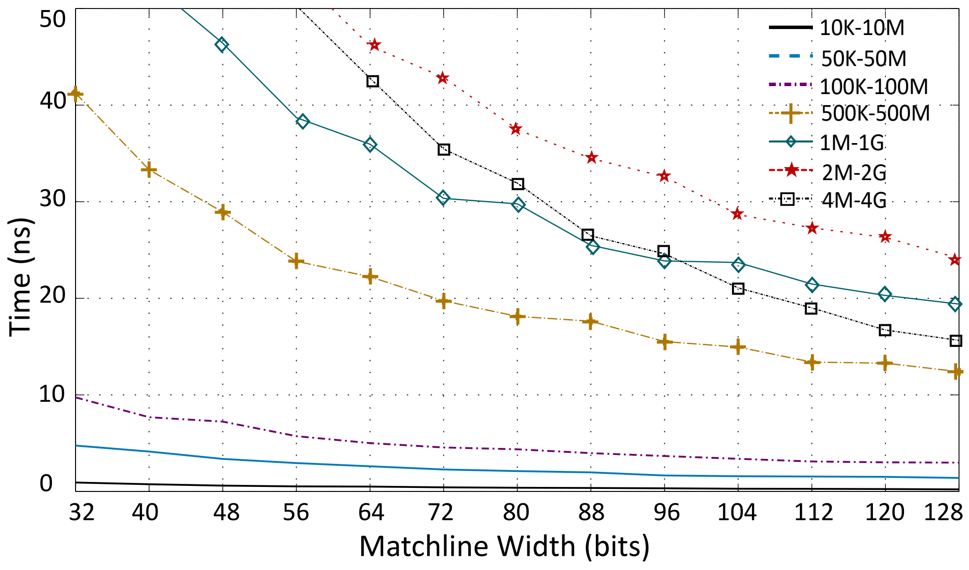


Figure 3.11 Sensing window with different LRS-HRS pairs

To find the LRS-HRS pair that gives the largest sensing window, a LRS-HRS sweep was run. The results are shown in Figure 3.11. With constant $\mathcal{R}$=1000, memristors with LRS-HRS at 2M-2G provide the most tolerance to timing variations without significantly compromising the sensing margin. The resistance ratio set in this experiment represents an achievable value by physical memristors which is in the range of hundreds to 10^6. To the best of my knowledge, no physically fabricated memristors exactly match all these proposed criteria on resistance: the resistance ratio and the resistance values. However, I use these as an indication for memristors with the optimised performance in the remaining part of this section.

To read a specified row, the memory array with 1024 rows is divided into 8 local blocks; each local bitline in a local block is driven by 128 cells along the column. Results from local blocks are collected using standard NAND gates. A total number of 8 NAND gates are used to drive the global bitline in each column.

Parasitic capacitance from the 128 cells attached to local bitlines requires a 0.8 ns precharge operation to obtain the valid voltage along the bitline. In the evaluation stage, the specified matchline is activated for probing the state of the cell. In this setup, it takes about 1.7 ns for the sensing result to propagate to the global bitline.

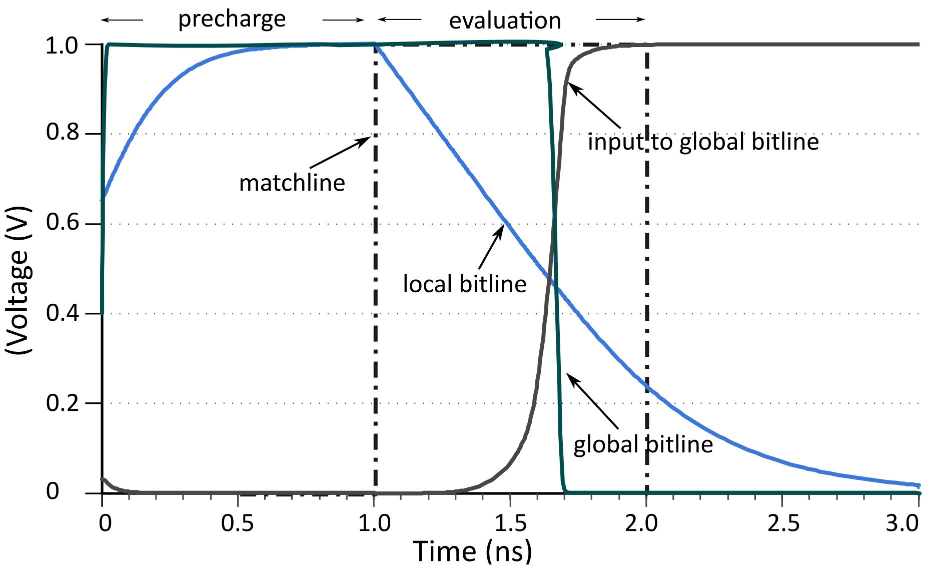


Figure 3.12 A timing diagram for a single two-stage read operation

3.4.2 Energy

The energy consumption in CAM systems is dominated by dynamic dissipation. Of a CAM's three operations, search is the most frequent and hence the focus of this discussion.

Memristor models with $\mathcal{R} \geq 1000$ are examined in a 64 x 128 memory setup. It is shown that matchlines, including memory cells, contribute approximately 60 % of the total energy on average as shown in Table 3.4. Sensing circuits also account for a considerable portion because of the leakage from partially on gates connected with matchlines and *Done* signal lines. The parametrised memristor (2 MΩ-2 GΩ) allows sensing circuits to sample the signal from matchlines and *Done* signal lines at an earlier time, which reduces the leaked energy.

Table 3.4 Energy performance using different memristor models

No. Rows	[57]	[74]	[102]	2 MΩ-2 GΩ
Sensing circuit(pJ)	2.54	2.39	2.49	1.98
Matchline(pJ)	6.31	5.72	4.52	4.67
Selectlines(pJ)	1.22	1.18	1.31	1.15

Experiments with different memory sizes show that the sensing circuit starts to consume noticeably more energy when the memory size increases to 128, mainly due to the load capacitance along *Done* signal lines. To address this issue, multiple reference rows are used to provide local timing signals for subarrays. Experiments with the 256 x 128 memory show that the energy is cut by 22 % by having 4 reference rows, each assigned to a 64 x 128 subarray (see Table 3.5).

Energy consumed per bit generally decreases as the size of the memory increases. In the simulation of the 256 x 128 memory array, the energy is as low as 0.73 fJ/bit/search, which is significantly smaller than that reported for phase-change and STT based CAMs with the same memory cell structure [61, 94].

Table 3.5 Energy performance of different memory sizes (fixed width: 128 bits)

No. Rows	32	64	128	256	256*
Sensing circuit(pJ)	0.81	1.98	5.21	15.28	10.64
Matchlines & selectlines(pJ)	3.01	5.82	7.95	15.80	13.59

* A 256 x 128 memory with 4 reference rows

The simulations indicate that a 256 by 128 CAM block using memristors (2 MΩ-2 GΩ) should operate reliably and power efficiently. It will provide 10 ns of sensing window and a sensing margin of 0.36 V. For a deeper CAM, multiple 256 row blocks can be used in parallel.

3.5 Summary and Conclusion

The chapter presented a memristor-based CAM that achieves high density by removing all but two access transistors. It also examined the impact of memristor characteristics on memory scalability and power performance.

The chapter began with a review of the evolution of CAM memory cells. There is an ongoing effort in the research community to reduce the area overhead of CAM, which has limited broader adoption of this technology. Area improvements have been made by replacing the SRAM and by simplifying the bit-comparison circuit in the cell. There is a trend in current publications to consider CAMs with resistive devices. These new resistive devices are small, can be packed densely in the wire stack of a CMOS process, and are non-volatile.

Section 3.2 described a four-transistor two-memristor (4T-2R) memory cell that replaces the complementary SRAM storage bits and as well as the comparison circuit. Unlike conventional RAMs, the new memory needs to program multiple cells in bulk, and a two-step write operation is required to program a row. A single-ended large signal sensing scheme was adopted for the read operation, because small signal sensing schemes requiring direct connections to memory cells have risks of charging unselected matchlines unintentionally.

Search operations and memory organisation were introduced in Section 3.3. Search involves all memory rows for the purpose of parallel comparison. To detect a match, enough time should present in the sensing window. For individual rows in the memory array, search results depend on the bit-comparison results from the cells attached, which is ultimately determined by the matchline to ground resistance in the evaluation stage in a search. This finding leads to the exploration of device level characteristics for a better sensing margin and power efficiency. The new CAM memory follows the basic structure of a conventional CAM array with the exception of reference rows and hierarchical bitlines. Reference rows provide a way to coordinate the sensing operation across a deep memory array by telling matchline latches the exact time to start storing the sensed results from matchline sense amplifiers. Hierarchical bitlines are introduced to collect the results from large-signal sensing method at the cell level.

The impact of device parameters on memory performance and scalability was analysed throughout Section 3.2 and 3.3. Key memristor parameters were examined in various memory arrays with different sizes in Section 3.4. The experiments used simulation results of sensing windows and sensing margins to explore the design space. The time available in a sensing window during a search determines the maximum memory depth. A simulation among memristors with a 1000 resistance ratio shows that it is possible to find the LRS-HRS pair that gives the largest sensing window hence allowing the largest memory depth. Meanwhile, sensing margins with different memristor parameters were explored to give a wider matchline (i.e. 256 bits). Memristor capacitance and resistance have a direct impact on energy performance. Besides the obvious benefit of small capacitance, appropriate LRS-HRS pairs speed up the sensing operation, and thereby reduce energy consumption. The proposed memristor parameters demonstrate a better energy performance compared to other memristor models. This is achieved mainly by stopping the matchline sensing at an early stage and employing multiple reference rows to minimise the arrival time of the timing signal.

This memristor based CAM achieved the objectives of improving memory area and energy efficiency. In spite of the fact that memory cell area is still dominated by transistors, the cell area overhead is improved by replacing both the storage SRAM cell and bit-comparison circuits with a more compact memristor-transistor hybrid circuit. A slightly improved search energy performance is demonstrated by tweaking device parameters and adopting multiple reference rows. However, it is usually the high level optimisations that can make a more significant difference in energy consumption. These optimisation techniques will be introduced in Chapter 4.

Chapter 4

A 2S-2R TCAM

All of the hitherto proposed designs for resistive CAMs use access transistors in the bit cells. In spite of the engineering effort to build compact MOS/memristor hybrid circuits, for instance by hiding transistors under the memristor array layer [79], the presence of transistors ultimately determines the cell area and contributes to a substantial portion of total energy consumption in the CAM. This chapter presents a more compact CAM cell structure that does not need transistors.

Memristors are passive devices, hence a memristor based memory core needs to have a considerable amount of MOS based peripheral circuitry around it, including various drivers, buffers and sensing circuits to either write data in or read data out of the core. This area overhead to a certain extend negates the density advantage achieved at the cell level. An example given in [106] shows a RRAM array whose area efficiency is lower than its MOS counterpart even though its cell is five times smaller. An area-optimised memristive memory core, or *subarry* in the term of RAMs, has to be large enough so that the area of MOS circuits is not significant compared to the total saved area in the cells.

These considerations lead to two requirements for the CAM design: the CAM cell must be compact to save area and it must be scalable to form a large memory array without bulky MOS circuits built in. These requirements are in line with goals of RRAM designs. The area efficiency issues of RRAMs were recently addressed in work by [79, 81, 82] in which transistors were eliminated from the bit cells by using a 1S-1R structure in which each bit cell contains one access selector in series with one resistive device. These are fabricated as a sandwich of amorphous oxide material between two crossbar nanowires. The reported subarrays in these RRAMs suggest that it will be possible to attain single memory arrays of millions of bits using crossbars. The progress in crossbar-based RRAMs gives hope to building large scale memristive CAMs. This section shows how the same 1S-1R memory array can be used to build a ternary CAM (TCAM) by changing the data encoding and related peripheral logic.

The chapter is organised as follows: the first section presents a review of related work on RRAM operation and leakage current issues. Section 4.2 describes a new crossbar CAM. It begins with cell level operations including write, read and bit-comparison operations. Building on top of memristive CAM cells, the section 4.2.2 illustrates higher level memory array organisations and CAM operations using various control lines. After a complete overview of the memory, the section reports the results of simulations of a subarray and discusses the search performance and the area overhead. Section 4.3 moves on to the analysis of delay and energy issues in large CAMs. The remaining part of the chapter reviews and discusses techniques applicable to the proposed resistive CAMs (Section 4.4).

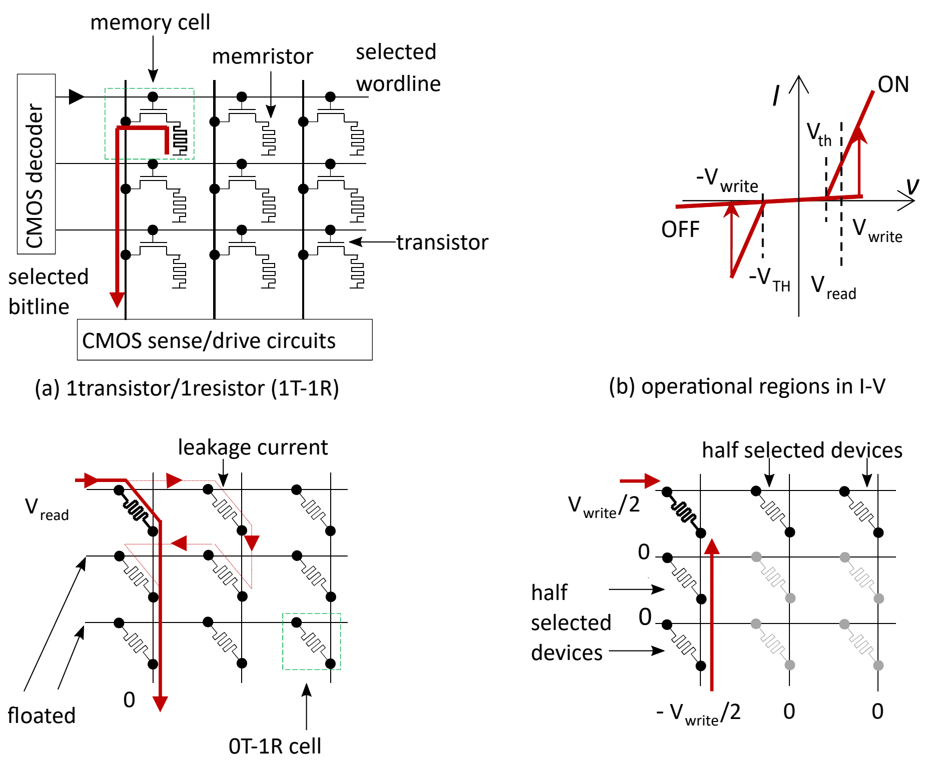
4.1 A Review of Crossbar-Based RRAMs

The proposed CAM array is based on recent work with crossbar-based RRAMs. This section continues the review of RRAM in Chapter 2 by introducing current leakage issues that have been limiting the reliability of RRAM operations and the size of the RRAM array.

The 1T-1R (1 transistor, 1 resistor device) cell structure shown in Figure 4.1 resembles a DRAM cell. When a cell is selected for read or write operations, there is no leakage current flowing into or flowing out from unselected cells as their access transistors are turned off. The presence of these access transistors means the 1T-1R cell cannot take full advantage of the crossbar structure; it is a compromise that trades cell area for more power efficient read and write operations in memory.

A pure memristor cell would be ideal for the crossbar arrangement in terms of cell area. However, without access transistors, leakage current has limited the size of the RRAM array. Large RRAMs can exhibit multiple leakage paths that result in errors in read operations. Specifically, sneak-path current from neighbouring cells flows into the bitline to be read out, and this noise makes it impossible for the sensing circuits to recover the stored information. This is particularly problematic in pure memristor implementations, where no access devices are used to isolate unselected cells from the selected one. An example leakage path is illustrated in Figure 4.1(c).

The source of the sneak-path current from the neighbouring cells in a read operation can be divided into three blocks as illustrated in red circles in Figure 4.2. This array



(a) 1transistor/1resistor (1T-1R)

(b) operational regions in I-V

(c) read operation

(d) write operation

Figure 4.1 RRAM arrays that uses transistors as access devices in (a) and pure memristor-based RRAM arrays in (c) and (d)

characterisation was used in [107] to conclude that the ultimate limit for the crossbar size is the HRS:LRS ratio $\mathcal{R}$: in the worst case when the maximum crossbar size is reached, the sensing current from a HRS cell is as large as the current from the LRS cell. With a fixed $\mathcal{R}$, the maximum achievable sensing margin is quickly diminished as the arrow grows irrespective of reading a single cell or reading a row of cells in parallel.

For a large crossbar we require memristors with a large internal resistance ratio. As shown in the memristor survey in Chapter 2, recent publications report memristors with $\mathcal{R}$ in the range from hundreds to thousands. According to the analysis from [107] which used the characterisation circuits in Figure 4.2, however, this improved ratio is only sufficient to achieve a memory size of maximum 64 by 1024.

To overcome this limitation without changing the characteristic of the device, half-select biasing schemes are used to suppress the leakage current via unselected cells (refer to Figure 4.1(d)). Typically, this access method biases the unselected wordline and bitline to half of the

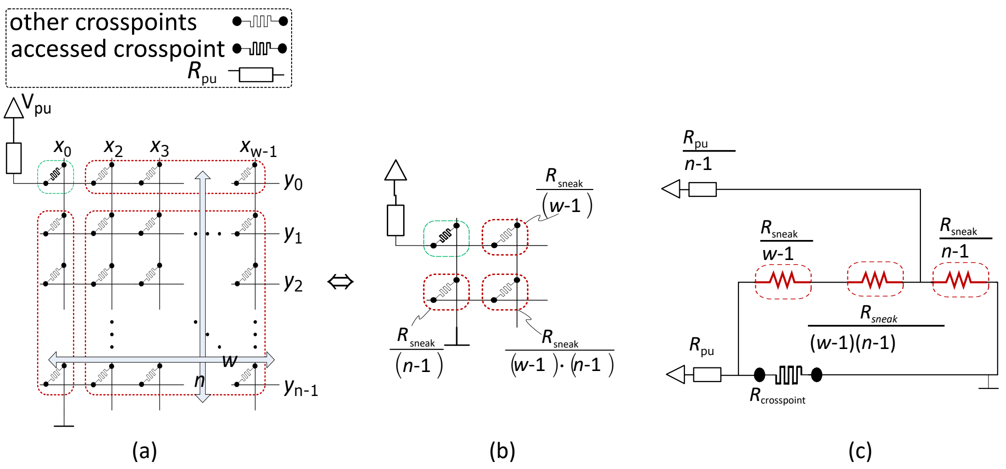

other crosspoints
accessed crosspoint
R_{pu}
V_{pu}
x_0 x_2 x_3 x_{w-1}
y_0 y_1 y_2 y_{n-1}
w n
$\frac{R_{sneak}}{(w-1)}$
$\frac{R_{sneak}}{(n-1)}$
$\frac{R_{sneak}}{(w-1)\cdot(n-1)}$
$\frac{R_{pu}}{n-1}$
$\frac{R_{sneak}}{w-1}$
$\frac{R_{sneak}}{n-1}$
$\frac{R_{sneak}}{(w-1)(n-1)}$
R_{pu}
$R_{crosspoint}$
(a) (b) (c)

Figure 4.2 Equivalent circuits for characterising leakage current in the crossbar-based RRAM array: (a) the equivalent circuit of single cell read; (b) the equivalent circuit of multiple cells read in parallel; (c) the abstracted circuit model for simulation

operational voltage level. Depending on the internal *non-linearity* of the memristors used, the current from a half-biased cell can be suppressed to a low level.

Non-linearity of the internal resistance is an important factor in all memory operations. Multiple groups have reported that it is possible to engineer extra rectifying layers into memristors, by which the desirable non-linearity can be achieved without significantly increasing the device footprint [74, 83, 79, 82]. Using half-select biasing, the voltage across the unselected cells can be kept less than the rectifying threshold and hence the leakage incurred is negligible compared to the current in the accessed cell. The ratio between the current in the selected cell and the unselected cells is called the *selectivity*. State of the art rectifying memristors have been reported with a selectivity over 10^6 [82]. In a broad sense, this can be viewed as an expression of the internal non-linearity of a 1S-1R cell.

The read and write operations using selectors and the half-bias access method in RRAMs remain largely the same and will be covered in Section 4.2.3 on page 62.

4.2 A TCAM Using 1S-1R RRAM Cells

This section presents a new TCAM design using 1S-1R RRAM cells. The new TCAM is based on cells fabricated and measured in [82]. To evaluate the TCAM, a simulation model is used that approximates the reported behaviour of the cells.

4.2.1 Storage Elements

Figure 4.3 shows I-V curves for a 1S-1R storage element measured in [82] and the curves from the simulation model used to generate results for this section. Note that the threshold voltage of the access selector (V_{th}) can be changed according to the thickness of the layers in its construction. The state of the element can be read by applying V_{read} and observing the current. The high resistance state ($I \approx I_{\text{off}}$) is used to store a logic '1', and the low resistance state ($I \approx I_{\text{on}}$) for a logic '0'. The state can be changed to '0' or '1' by applying V_{set} or V_{reset} respectively.

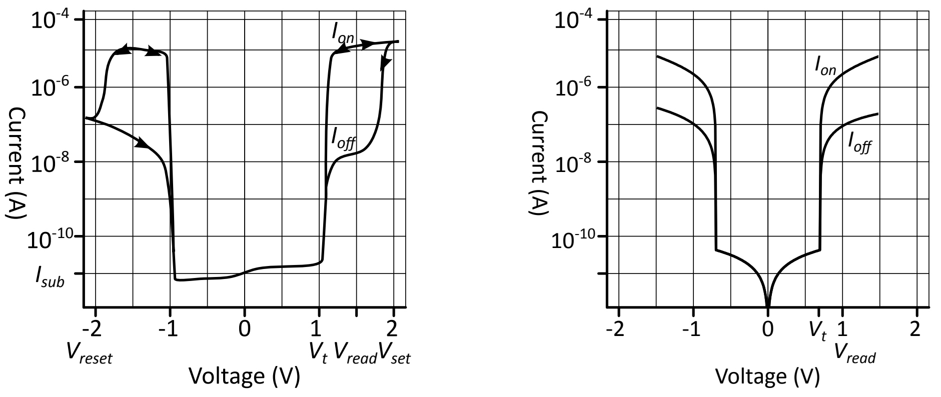


Figure 4.3 1S-1R I-V curves: left as measured in [82]; right as simulated for this design

4.2.2 Memory Organisation

The TCAM organisation is shown in Figure 4.4. The rows of nanowires in the crossbar form the matchlines (MLs), and the columns of nanowires are the selectlines (SLs). A 1S-1R storage element is placed at the junction of each matchline and selectline. Each bit of data in the TCAM is stored as complementary bits on two 1S-1R elements to facilitate search operations. This TCAM is hence composed of 2S-2R cells. The cell for the j-th bit of the i-th word in the memory is connected to ML_i, SL_j and $\overline{SL}_j$. The memory operations write, read and search can be realised by applying different voltage combinations to these lines. As in the RRAM from [82], the TCAM uses a half-bias voltage scheme to suppress the sneak current that can occur when multiple cells are connected in parallel. This requires the storage elements to be constructed with $V_{\text{set}}/2 < V_{\text{th}}$.

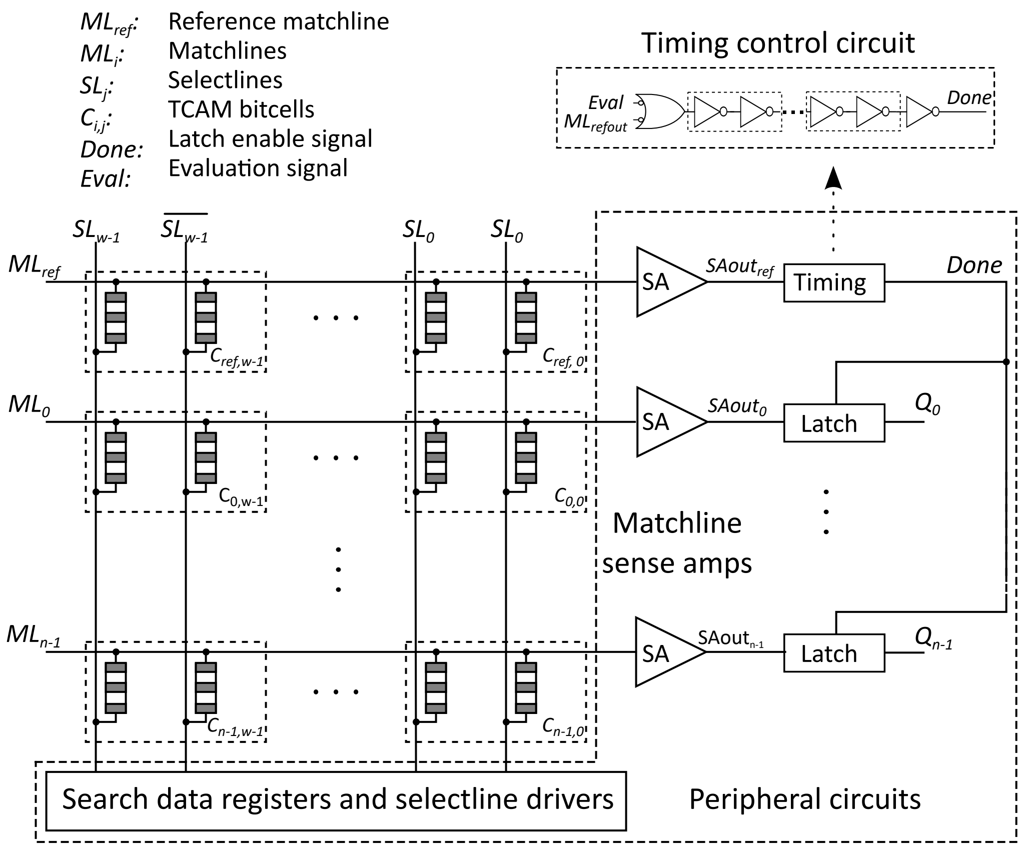


Figure 4.4 CAM matchline organisation with a reference row

4.2.3 Write and Read Operations

Similar to the write operation in the 4T-2R CAM, this design adopts the erase-before-set method, where ‘1’s and ‘0’s are written into each row in the two steps: erase (reset) the entire row and then set specified elements to ‘0’ which is represented by LRS memristors. Unlike the 4T-2R cell where access transistors provide isolation between cells from different rows, cells on crossbar structures are vulnerable to leakage current. A variation of the half-select write scheme is introduced for write operation.

To erase all elements along the i-th row to the HRS, the memory drives ML_i to ground and all selectlines are driven to V_{reset}. All unselected matchlines are set to $V_{\text{set}}/2$ so that all unselected cells are half-selected. Note that the resulting voltage across the unselected cells is less than the threshold voltage of the selectors, so a small current flows through the unselected cells, irrespective of their stored state. To set the element at a specified position (i,j) to LRS, the memory drives ML_i to V_{set} and column j to ground. All other unselected matchlines and selectlines are set to $V_{\text{set}}/2$.

To read out information, two modes are considered: (1) reading out all cells along a wordline simultaneously; (2) reading out a single memory cell at time. The first read mode increases the read throughput at the cost of more sense amplifiers and more instantaneous power dissipation. The second read mode needs fewer sense amplifiers, but is slow when an entire word needs to be read out bit-serially. For simplicity and a demonstration of read throughput, this design firstly considers the first read mode. The bit-serial read operation in the second read mode is introduced together with the search operation because of their common operational requirements.

A word is read out by simultaneously sensing the contents of the cells along the row. To read the i-th word, ML_i is driven to V_{read} where $V_{\mathrm{th}} < V_{\mathrm{read}} < V_{\mathrm{set}}$. Each selectline is pulled to ground via a resistor R_x, and all other unselected matchlines are biased to $V_{\mathrm{set}}/2$ and then left floating. The resistance of the storage elements on the i-th row can be detected by the resulting selectline voltages. Note that because of the complementary data encoding, only half the selectlines need to be observed. To maximise the area saved from this extremely compact cell, this design turns away from the large signal sensing approach and adopts a voltage-mode small signal sensing scheme. Internal sensing is used so that partial swing signals are converted to a full swing signal for inter-block communication.

Seevinck et al. [108] formulated a set of delay models that have been widely adopted for calculating delays in SRAM read operations. The voltage-mode delay in this design is given by

$$t_{\mathrm{delay}} = \frac{R_{\mathrm{wire}} \times C_{\mathrm{wire}}}{2} \times \left(1 + \frac{2(R_{\mathrm{B}}||R_x)}{R_{\mathrm{wire}}}\right)$$

where R_{wire} and C_{wire} are total wire resistance and capacitance. R_{B} is the selectline load resistance and R_x is the resistor in the voltage divider. This equation has been used to estimate the delay of a read operation.

4.2.4 Search Operation

The search operation looks for rows that match a cue word by comparing the cue against every row in the memory simultaneously. A mismatch between the data stored in a cell and the search cue creates a pull-down path that discharges the matchline. When a stored word matches the search cue, the matchline retains its precharged high voltage. The encoding is specified in Table 4.1.

Table 4.1 TCAM data encoding

Stored Data		Search Cue		Match	
Logic	$(D, \overline{D})$	Logic	$(Q, \overline{Q})$	Status	ML
0	(0, 1)	0	$(V_{set}/2, 0)$	Match	V_{read}
		1	$(0, V_{set}/2)$	Mismatch	0
1	(1, 0)	0	$(V_{set}/2, 0)$	Mismatch	0
		1	$(0, V_{set}/2)$	Match	V_{read}
X	(1, 1)	0	$(V_{set}/2, 0)$	Match	V_{read}
		1	$(0, V_{set}/2)$	Match	V_{read}

To perform the search, the matchlines are precharged to V_{read} and then left undriven. The selectlines are set according to the cue as in Table 4.1. In each cell, one of the storage elements will be biased by $V_{read} > V_{th}$ and the other by $V_{read} - V_{set}/2 < V_{th}$. Similar behaviours of matchlines are expected except that a small voltage drop is present across the selectors. If the data does not match the cue, V_{read} will be across the element in the low resistance state, and the matchline will be pulled low, as shown in shaded cells in Figure 4.5. If the data does match the cue V_{read} will be across the element in the high resistance state, and the matchline will discharge only slowly.

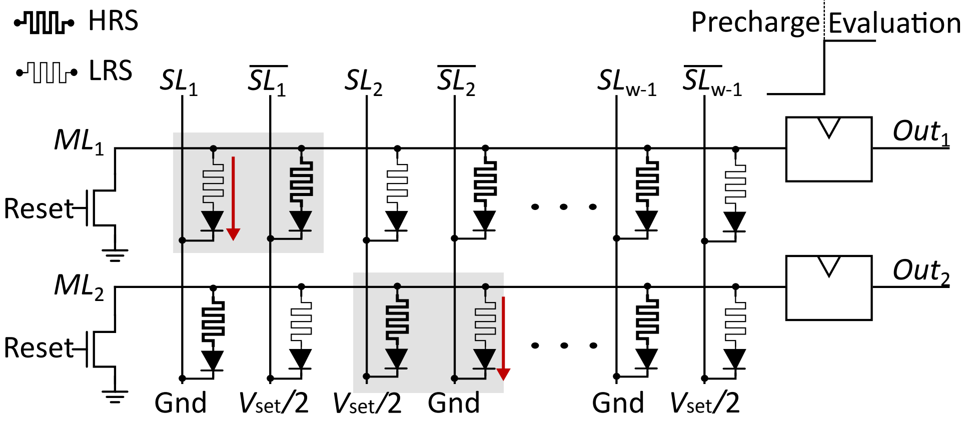


Figure 4.5 The pull-down paths highlighted in shaded cells discharge matchline voltages in a search

During the search operation, matchlines discharge at different rates according to the number of bits that match in their rows. Similar to the memory proposed in Chapter 3, reference rows (ML_{ref}) are used to generate a *Done* timing signal which tells the sensing circuits the correct sampling time. When the sensing window appears in the evaluation stage of a search it is required that the *Done* signal arrives between one-bit mismatch and exact-match rows to discern a one-bit mismatch.

A New Bit-Serial Read

With the reference row described above, it is possible to read a cell by searching along a specified matchline. Recall that a search in a CAM is essentially achieved by measuring discharging rates of each matchline involved. For example, a matchline with a one-bit mismatch discharges faster than an exact-match row and thus triggers an output voltage swing at an earlier time. It is interesting to observe that this is similar to the voltage-mode sensing used in RAMs. The matchline and the sense amplifier used in a search can be reused for reading a cell. Considering all cells share the same matchline, the read operation has to be bit-serial.

Consider a simple example where the word to be read out is [0 1 0 1]. The search data used to drive the selectlines is [1 1 1 1] so that each pair of complementary selectlines creates a single pull-down path in a cell. Readers may find the encodings in Table 4.1 useful to figure out the configurations in matchlines and selectlines. An extra reference row ML_{read} is needed to provide the timing signal as shown in search operations. The reference row for a read is [0 0 0 0], which thus produces a mismatch signal at every bit in this case. To read the first cell, the specified matchline is first charged high and all other matchlines and reference rows in search are left floating after they have first been discharged to ground in the previous search operation. In the evaluation stage, selectline SL_1 is set to ground to open the pull-down path in the first cell. This produces a mismatch in both the specified matchline as well as the reference row ML_{read} which then sends a delayed *Done* signal to the specified matchline to sample the result as shown in Figure 4.6. This process is repeated four times to read out all four bits along the matchline.

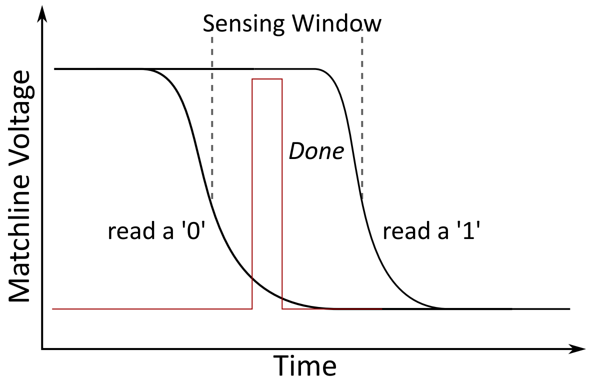


Figure 4.6 An illustration of the timing in a read operation

The time required to read out an entire word depends on the memory width (W). Both the number of clock cycles required, and the clock period increase with W. The clock period must increase because the capacitance of the matchline grows linearly with W. Overall the time taken to read an entire word depends on W quadratically. Another reason to justify the approach is the reuse of matchlines and sense amplifiers. This not only saves space, but also eliminates sensing facilities and their associated capacitance from the selectlines. Note that the selectlines are often more capacitive than the matchlines in deep memories.

4.2.5 Estimates and Simulation Results

SPICE simulations were performed to examine memory search operation and performance. The memristor model from [67] was used for simulating the resistive storage element and parameters from the 45 nm CMOS process from FreePDK [109] were used for transistors and wire loads. The nanowire capacitance and resistance was estimated to be $2.8\,\mathrm{pF\,cm^{-1}}$ and $355\,\Omega\,\mu\mathrm{m}^{-1}$ for regular copper wires with a line width of 15 nm. The selector is difficult to model, but a bi-directional diode approximates the DC behaviours reported in [82] as shown in Figure 4.3 on page 61. A test setup, consisting of a 129 x 128-bit array was used to analyse the search functionality. The circuit array was simulated with 129 distinct pre-stored patterns to cover a complete range of matching scenarios.

At the evaluation stage, multiple mismatches quickly discharge themselves to ground because of the multiple pull-down paths. The reference row, designed with the same RC characteristics as a regular row, is deliberately delayed to trigger the *Done* signal to stop a one-bit mismatch row from proceeding to the output. A full search cycle completes in about 5 ns, and a clear sensing window of 1 ns is present between the one-mismatch and exact-match (see Figure 4.7).

A voltage differential of over 120 mV is observed between the match (V_{mismatch}) and mismatch matchlines (V_{match}) at the instant of the *Done* signal (see Table 4.2).

120 mV is sufficient for reliable detection with 40 mV typically being considered a lower bound for reliable operations [40]. One concern with this non-threshold type memristor model is the state degradation under a voltage bias in search mode. Considering a small voltage bias (0.3 V) across memristors at the evaluation stage for 3 ns in search, the simulations indicate that with the same setup the memristor's resistance is reduced by 2.9 % after performing six million bias operations in searches.

This chapter presented two read operations. A read cycle in the bit-serial read operation is equivalent to a search cycle, which finishes within 5 ns. It takes about 640 ns to read out the entire 128-bit word. In the parallel read operation, only one of two memristors in a cell is probed. The cell current from an LRS memristor is about 32 µA. The wire loading and

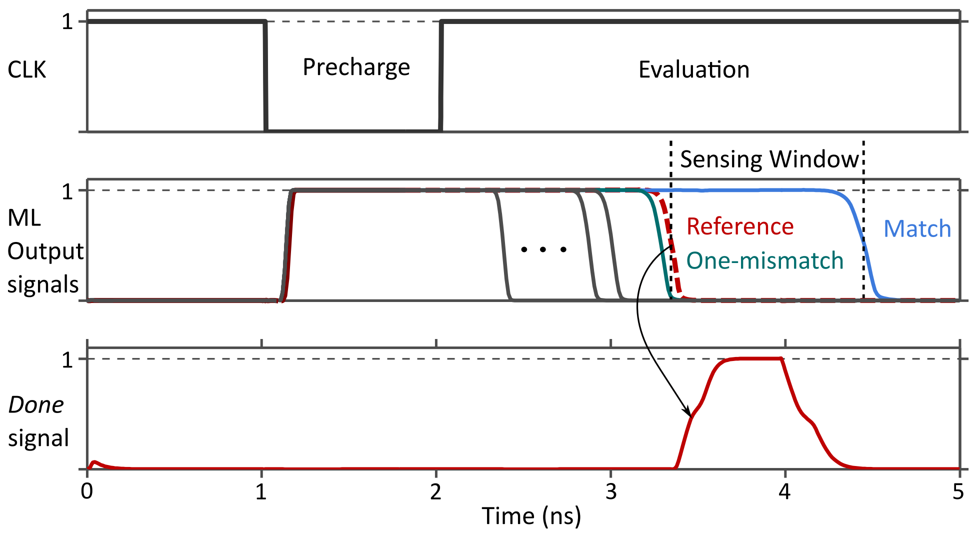


Figure 4.7 Simulated waveforms in a search cycle

Table 4.2 Matchline voltage sensing margin (mV)

No. of Mismatches	$\lvert V_{\text{mismatch}} - V_{\text{match}} \rvert$
1	120
2	137
3	139
4	140
...	...

voltage divider resistance are configured to obtain a maximum selectline swing of about 300 mV. A typical 1 µm selectline has capacitance (C_{wire}) of 2.8 fF and resistance R_{wire} of 355 Ω, and this length is sufficient for about 1024 memory rows. In simulation of a 10 000-row deep and 4-bit wide memory array, the selectline delay of this voltage-mode sensing is about 1.4 ns.

To evaluate the energy performance of the memory array, a memristor capacitance model and measurements from [58] have been used. By extrapolating this according to the ITRS's 5 nm memristor feature size projection, the memristor capacitance is set to 0.1 fF. The resultant capacitance of a 128-bit matchline is about 28 fF.

Simulation results for the energy per search cycle in a 128-bit wide TCAM are shown in Figure 4.8. A 32 Kibit (256 x 128-bit) TCAM subarray consumes about 13.6 pJ. These results include the energy consumed by the peripheral circuits including the selectline. Energy consumed in peripheral circuits becomes dominant in larger arrays, and the calculation shows a linear energy increase at the cell level. Note that the uncertainty of this preliminary estimation is high because of the lack of well-tested electrical models for the nanowires, memristors and selectors. Nonetheless the simulations are sufficient to indicate the trend of the power consumption as the size of the memory is increased. This is shown in Figure 4.8. The peripheral circuits include the selectline drivers, matchline latches and the timing control circuit.

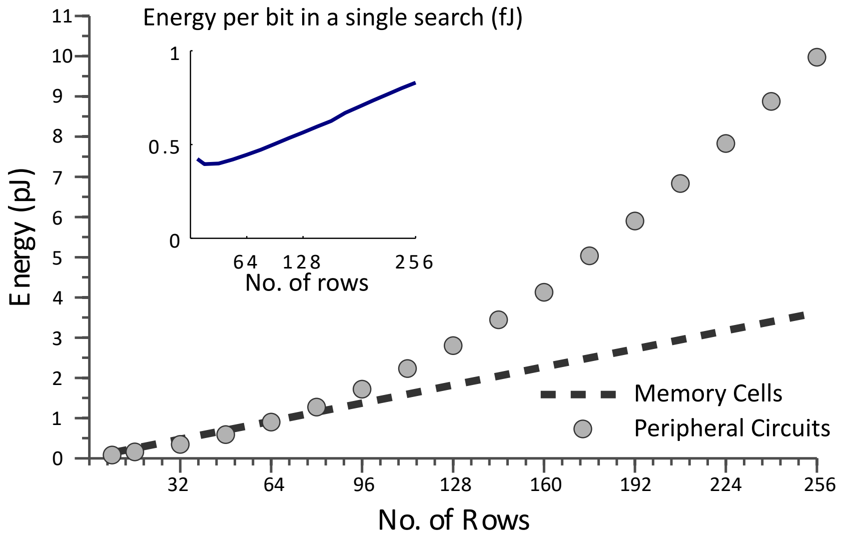


Figure 4.8 Energy consumption of simulated 128-bit wide TCAM with various sizes

Despite the tighter arrangements reported in the publications [110, 111], a 50 nm half-pitch (wire width) is assumed hereafter for estimating the geometry of crossbar based CAMs. Nanowires at the metal layer are orthogonally routed in the crossbar structure which sandwiches the memristors at each crosspoint. Although the size of wires at the metal layer can vary, for convenience reasons, this estimation assumes that the metal wires have coincident edges with a memristor's electrode and metal-oxide regions.

A conceptual view of the cell layout is illustrated in Figure 4.9. A track, defined to be 100 nm in this case, needs to have enough space to contain a wire and the spacing to the next wire. By adopting the Lambda-based design rules, where λ is half the thickness of metal-oxide (its thickness is set to 5 nm), the wire width is 20 λ and a track is 40 λ. A CAM cell occupying two vertical tracks and a horizontal track gives an area of 40 $\lambda \times 80\ \lambda$.

This conservative estimation gives a cell area of 0.02 μm^2, which shows significant density advantage over the 4T-2R approach (0.41 μm^2) in Chapter 3. The area of peripheral circuits per row is estimated to be 5.05 μm^2, based on which the memory density is estimated to be over 10 Mibit mm^{-2}; over an order of magnitude density improvement can be achieved compared to the 640 Kibit mm^{-2} TCAM in the 28 nm process CMOS implementation [112].

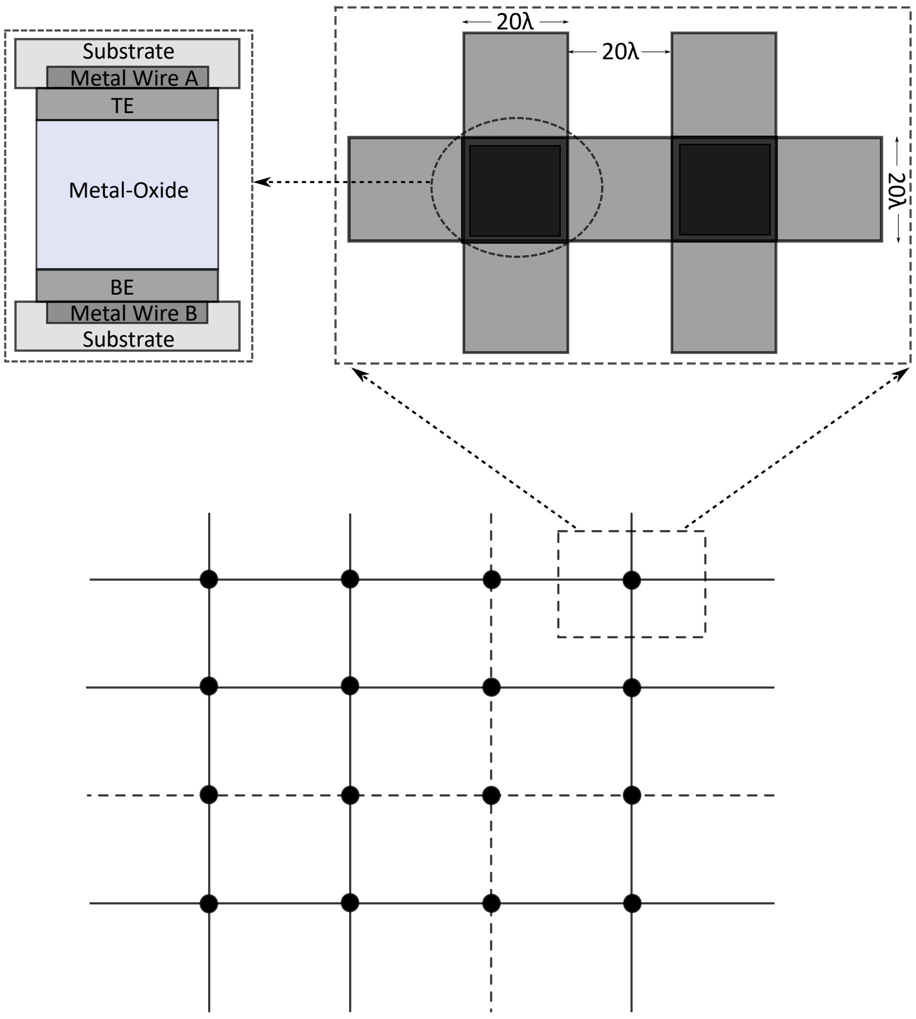


Figure 4.9 Layout of routing tracks and memristors in the crossbar structure

The area improvements can be further enhanced by hiding peripheral circuits under the crossbar array at a separate layer as demonstrated in [81, 80]

4.3 Energy and Delay of Large CAMs

To build a very large CAM it will be necessary to use an organisation like that used for large RAMs in which the memory is broken into subarrays. Subarrays are arranged vertically and horizontally to form arrays (multiple subarrays), subbanks (multiple arrays) and banks (an array of subbanks).

In this section, energy consumption of different memory array components is estimated based on SPICE simulation results of a 1024 x 128-bit test subarray. In the memory core, i.e. the crossbar, energy dissipation along control lines (matchlines and selectlines) is considered. Peripheral circuits, based on transistors, also consume a substantial amount of power. To estimate the energy performance of large scale TCAMs, I used the model from [113], which estimated power consumed by MOS based peripheral circuits including routing and multi-match resolver blocks. The energy consumption of a specified memory size is hence estimated by scaling SPICE measurements and adding up values from various memory components.

4.3.1 Search Delay

Delay in a single monolithic memory subarray is mainly due to the time consumed in charging and discharging matchlines and selectlines, as well as the delay in matchline sensing circuits. Larger capacity memories have more complex organisations that introduce routing delays between subarrays, arrays and banks. For the estimates presented here, it is assumed a chain of 256-bit priority encoders is used to resolve multi-match situations. The delay of this chain contributes to the total delay. The main delay components include:

1. matchline precharge, discharge delay;
2. matchline sensing circuits (sensing, reference signal...);
3. selectline precharge and discharge delay;
4. routing delay between banks and inside banks between arrays;
5. multi-match resolver (e.g. priority encoders) delay.

The array level delay measures of matchlines, selectlines and sensing circuits are obtained from SPICE simulations. Delay arising from routing and encoding circuits are estimated using the model from [113], where an H-tree bank routing organisation is employed. Given a memory configuration, the total delay is estimated based on individual delays from a collection of components.

Delay of Search Operations

Search speed in an array is mainly determined by three sequential search operations, specified as the following:

Reset - The time taken to discharge all matchlines in parallel, although most of them are already discharged low in the previous search operation. Meanwhile, the latched results are cleared. The last delay is negligible, and it does not affect the critical path delay.

Precharge - The time taken to precharge matchline to a valid high voltage V_{read} ($V_{\text{read}} > V_{\text{th}}$) and to charge selectlines to $V_{\text{set}}/2$.

Evaluation - The time taken to (1) discharge half of the selectlines to ground, (2) discharge mismatched rows to ground, (3) charge the *Done* signal to a valid high voltage above $V_{\text{dd}}/2$.

The equivalent circuits for characterising matchline and selectline RC delay are shown in Figure 4.10. Diffusion capacitance from the precharge and discharge circuits are also included in simulations. For a fixed 128-bit memory, both of the matchline precharge and discharge delay take about 0.3 ns, which is a constant delay portion that can be obtained from simulating small size memory arrays.

The *Flight time* along selectlines increases quadratically with the wire length and hence the memory depth. The Elmore delay model reported in [114] (source to sink) is used to approximate the flight time. Simulations show that for a memory array with 1024 rows, it takes about 0.3 ns to charge or discharge the designated selectlines using a 2x unit-sized inverter.

The *Done* global wire is highly capacitive due to the gate capacitance from D-latches and the long wire load. The wire length increases significantly in the CMOS domain; each D-latch with 6 λ track pitch has a height of 1.8 µm. Using a wire with resistance 0.8 $\Omega\,\mu\text{m}^{-1}$ and capacitance 0.1 $\text{fF}\,\mu\text{m}^{-1}$, the Elmore delay for the line of 1.85 mm in length (1024 rows) is about 0.6 ns.

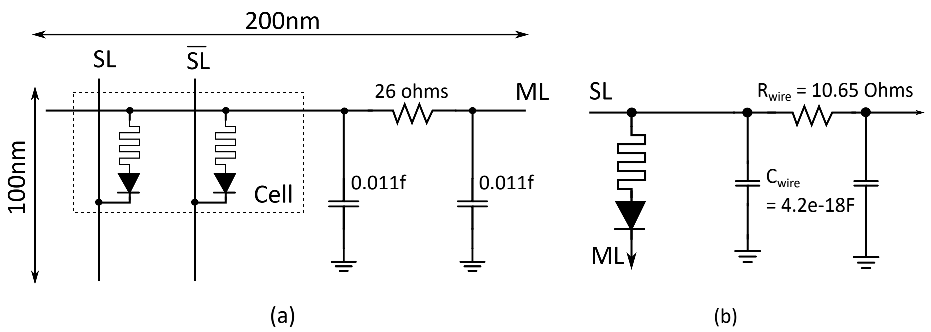


Figure 4.10 The equivalent circuit of a CAM cell (a) and a selectline unit (b)

Table 4.3 Soar semantic memory data size (bits) in a TCAM

	LTI	Attribute	Value
basic requirements	72	80	256
provided in estimation	128	128	128×2

4.3.2 A Case Study

The 1K x 128 TCAM array in the previous section can now be used to evaluate larger memories. In this section, the TCAM array and existing sensing, buffering and routing circuits are used in a case study to evaluate the performance of the proposed TCAM in a cognitive task.

The case study will use a TCAM for the word sense disambiguation task (WSD), which has been used in the literature as a benchmark to compare semantic memory systems [34, 31]. WSD involves searching a dataset for the correct meaning, or sense, of each word in a written text. The WN-LEXICAL dataset in the Soar semantic memory format contains nearly 4 million memory elements (entries) and it is the largest semantic stores reported for Soar [34]. Table 4.3 shows that the LTI column needs 72 bits of data, the Attribute column, with maximum 10 characters in WSD tasks, needs 80 bits, and the Value column will occupy 256 bits (two arrays to accommodate long strings). The LTI and Attribute fields are assigned 128 bits so that three fields can be contained in four uniform 128-bit arrays.

Figure 4.11 shows a block diagram of the TCAM organisation used for this case study. The software estimation model in [113] organises memory in terms of arrays, subbanks, banks, the number of banks; it has a fixed memory structure which, when provided a specific value to each component, calculates the breakdowns of total energy and delay of search operations. To store the semantics data in the WSD task, the memory is arranged as:

Array: a 1024 x 128 monolithic array of memory cells whose performance is estimated from the 256 x 128 subarray.

Subbank: 4 arrays cascaded with size 1024×512, hence matchline is divided into 4 segments

Bank: 4 subbanks organised in an H-tree routing structure

No. of Banks: 64 banks organised in the H-tree structure

In total, each bank contains 64K entries. A module with 64 banks thus can store up to 4 million entries which is the capacity requirement for the WN-LEXICAL dataset. The

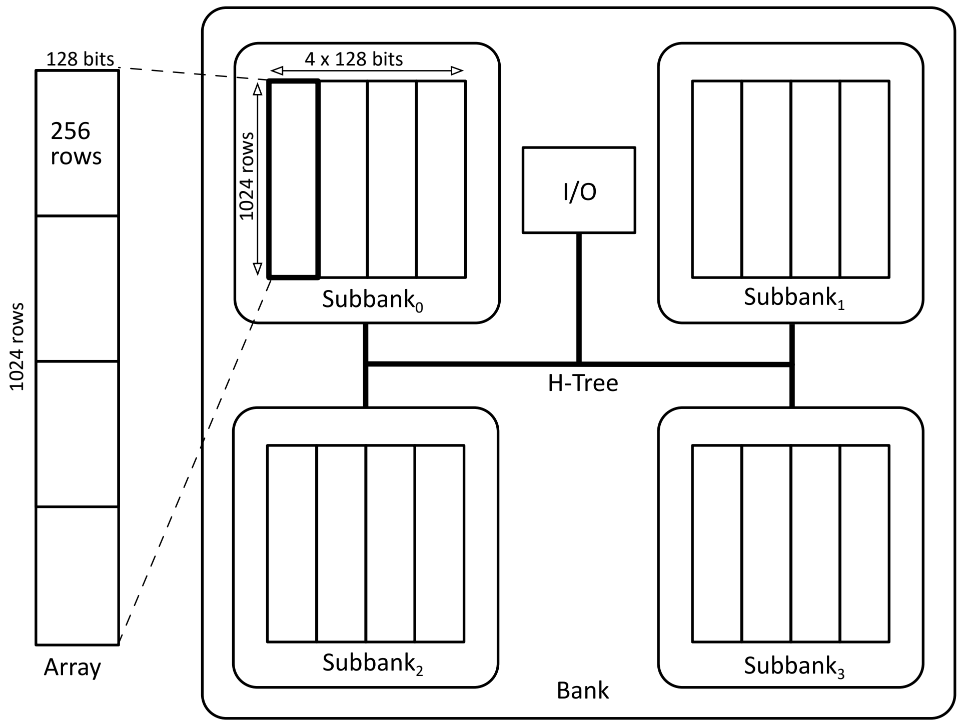


Figure 4.11 An example TCAM in a bank arrangement which consists of four subbanks

estimation model does not include the registers that drive the selectlines. Nor does it include the circuits that prepare the search data at the local registers in each memory array.

Search the Whole Memory

In a brute-force search operation, the search data is fed to search buffers of all memory blocks and these memory blocks are activated and searched in parallel. Table 4.4 gives a summary of the estimates of energy and delay in a search operation. The memory is able to finish the search within 5.1 ns which consumes approximately 725 nJ. Two major components in energy consumption are from the memory array peripheral circuits and various control nanowires connecting with cells which accounts for 84.6 % of the total power. Priority encoders consume about 112 nJ in total. The read operations spend over 99 % of their power in H-tree routing. The search delay inside memory arrays accounts for about 43 % of the total delay. Other delays on the critical path are mainly due to the priority encoder circuit and the routing in subbanks and banks. Compared to reported TCAMs based on the 0.18 µm process, which are typically small in the range of a few Mibit to 80 Mibit [103] (760 mW Mibit^{-1}) and [112](420 mW Mibit^{-1}), the proposed TCAM achieves 82 mW Mibit^{-1}.

Table 4.4 A breakdown of the estimation on energy and delay of a search

Energy:			
	components	nJ	%
in arrays	matchlines	153.5	21.1
	selectlines	144.0	19.9
	reference rows	0.2	< 0.1
out of arrays	priority encoder	112.0	15.4
	peripheral circuits	315.8	43.6
total energy		725.5	
Delay:		ns	
in arrays	reset & precharge	0.6	11.7
	evaluation	1.6	31.4
out of arrays	priority encoder	1.3	25.5
	routing	1.6	31.4
total delay		5.1	

The estimation of the latency of a complete retrieval process also needs the latency of read operations. I use the proposed bit-serial read scheme to derive the delay of read operation. A read cycle in this scheme takes about the same time as a search. Priority encoders are not required because only a single row is activated in read. Thus, reading a row in a memory array with 128 bits takes about 486.4 ns. A memory element with 512 bits is allocated in four memory arrays, which, if configured to read in parallel, will give the same delay as reading a single memory array. Considering the memory object being retrieved has 10 memory elements, a total of 4.9 µs is required for a complete retrieval process.

4.4 Higher-Level Improvements

Although the resistive CAM does not need power to maintain stored information, it consumes a considerable amount of energy when operating the parallel search; all matchlines are precharged high to participate in the search and, except for the exact match row, all matchlines are then discharged to ground at the end of a search cycle. More importantly, all transistor based peripheral circuits are activated along with the control lines. The principle governing dynamic power consumption suggests that the same amount of energy is distributed to every matchline regardless of whether there is a match or a miss. A smarter way to distribute energy would be a selective evaluation scheme in a memory search that only considers possible candidates.

A series of techniques are proposed in the literature along these lines from cell level circuits to architectural ideas. This section reviews some important power saving techniques that adaptively activate parts of memory in a search, and discusses their applications to the resistive crossbar CAM.

4.4.1 Pipelined Matchlines

Matchline segmentation is a common method to enable the selective precharge of matchlines. With segmented matchline, all matchlines are precharged prior to a search. Matchlines that are not pulled low during a search will still leak charge over time, and will need to be fully precharged before the next search. This power dissipation can be reduced by only precharging matchlines for segments that are candidate matches. This selective precharge scheme is shown in Figure 4.12. It has been used in [115–118] as an effective technique for reducing energy consumption of a search.

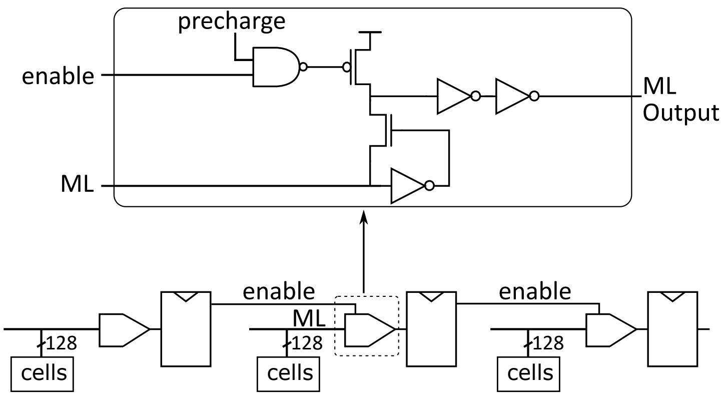


Figure 4.12 An illustration of a three-stage pipeline each of which represents a CAM segment

One implementation of the selective precharge scheme divides a matchline into individual cells and each cell forms a segment, as demonstrated in [116, 117], such that a segment is only activated when there is a match in the preceding cell. A mismatch cell will stop discharging the subsequent matchlines. In search, the evaluation of an exact-match row ripples through each cell along the row. One drawback of this approach is the delay from its bit-by-bit sequential operation. More importantly, the evaluation needs extra circuits built into the cell and along the matchline, which will significantly compromise the area benefits of a crossbar structure.

In a more general selective precharge scheme, an implementation may divide matchlines into any number of segments. Figure 4.12 shows a divided matchline in three 128-bit wide segments. If storage elements are inserted to connect two adjacent segments, the evaluation stage of a search can be pipelined along the matchline in three stages. This is a typical approach that trades area efficiency for power reductions. Results from a 1024 x 144-bit array in [89] show that a five-stage pipelined scheme saves 56 % of the total power consumption [115]. It should be noted that the idea of pipelining matchlines alone does not show a radical difference in terms of power consumption improvements compared to the selective precharge scheme, but it enables the integration of hierarchical selectlines which is able to further reduce the power consumption [38].

The matchline segmentation and selective precharge schemes are well suited to large resistive CAMs because of these memories' poor matchline sensing margins. As analysed in Section 3.3.1 on page 48, matchline sensing margins drop as memory width increases. The maximum width that can be achieved ultimately depends on the memristor's resistance ratio. In the experiments, matchlines with 128 or 256 cells can be reliably attained with acceptable sensing margins which are in line with the results reported in the literature [40]. Despite that the rule of area efficiency requires a matrix of CAM cells to be as large as possible, it is difficult to build a standalone cell array with a width beyond a few hundred bits using current memristors. In other words, large resistive CAM implementations have to divide long words into multiple segments for reliable searches.

Recall that a long-term memory element is a tuple in the form of LTI-Attribute-Value. If each matchline stores one memory element, a natural way to split the matchline would be dividing it into three segments: LTI, Attribute and Value. These three segments then correspond to the record fields in a database system. For generality, I assume a 128-bit segment is wide enough to contain a record field. A memory containing 1000 memory elements will thus be composed of three segments, each of which is 1000 rows deep and 128-bit wide. Each segment also needs a slightly modified matchline sense amplifier and a flip-flop. The sense amplifier is modified to implement a selective precharge, as shown in Figure 4.12. This is done by simply adding a two input NAND gate to accept enable signals from the flip-flop in the preceding segment.

The activity of a typical matchline is shown in Figure 4.13 where three fields of a tuple represent the three segments. The search starts from left to the right. All matchlines are precharged in the first LTI segment to gauge the match potential of each row. Depending on the search results, the subsequent segment of a row will be selectively precharged. In a memory storing rows of random data, the possibility to survive the first segment search is 1

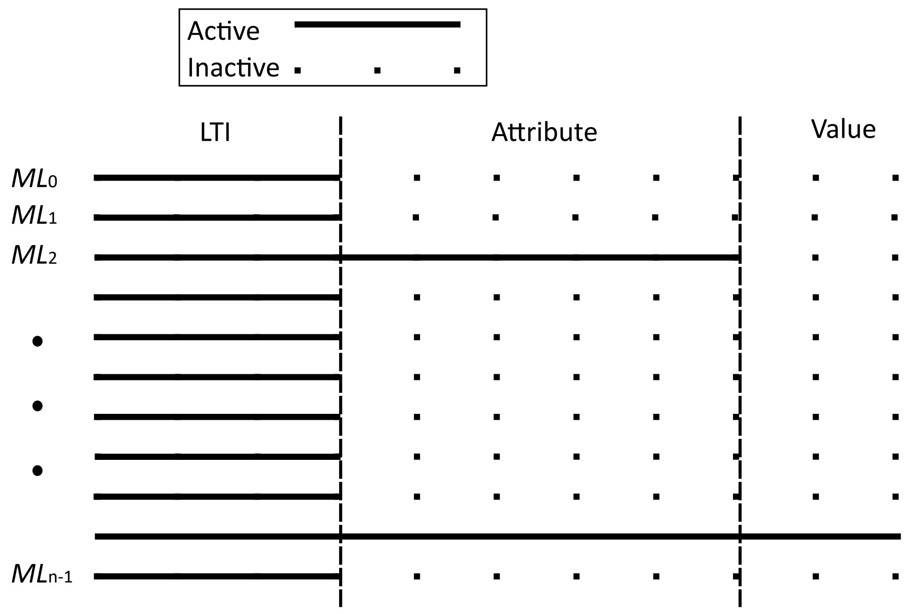


Figure 4.13 An example of a selective precharge scheme pipelined in three stages; active rows represent precharged matchlines, otherwise they remain inactive in search

in 2^{128}. Long-term memory elements are not random data; elements in an object share the same LTI, hence depending on the object size, several rows may survive the search in the first segment and proceed to the second. The majority of rows will not pass the first segments and their matchline will not consume power in the second and third segments.

However, the matchline segmentation with a selective precharge scheme in this new CAM is not as effective as in CMOS CAM designs where the total search energy is reduced by 56 % [115]. The pipelined matchline scheme saves 16 % of total energy consumption in an experimental memory consisting three 64 by 128 segments. The main reason is that energy consumed by transistor-based peripheral circuits accounts for almost half of the total energy. As a result, a large amount of the saved power in inactive matchlines goes into the extra peripheral circuits needed in each memory segment. Also, the first segment is always active in every search regardless of producing a match or mismatch. The third factor is that although the majority matchlines in the second and third segments are not active, half of the selectlines in the two segments are still precharged.

To further improve the design, a few general suggestions are proposed: first, the segment width should be customised accordingly to achieve the best power performance. For example, the initial search in the first segment only needs a few bits to be highly discriminative; a 4-bit matchline in the first search in theory eliminates over 93 % matchlines in the second search. In a memory design as shown above, the design of the first segment involves a careful

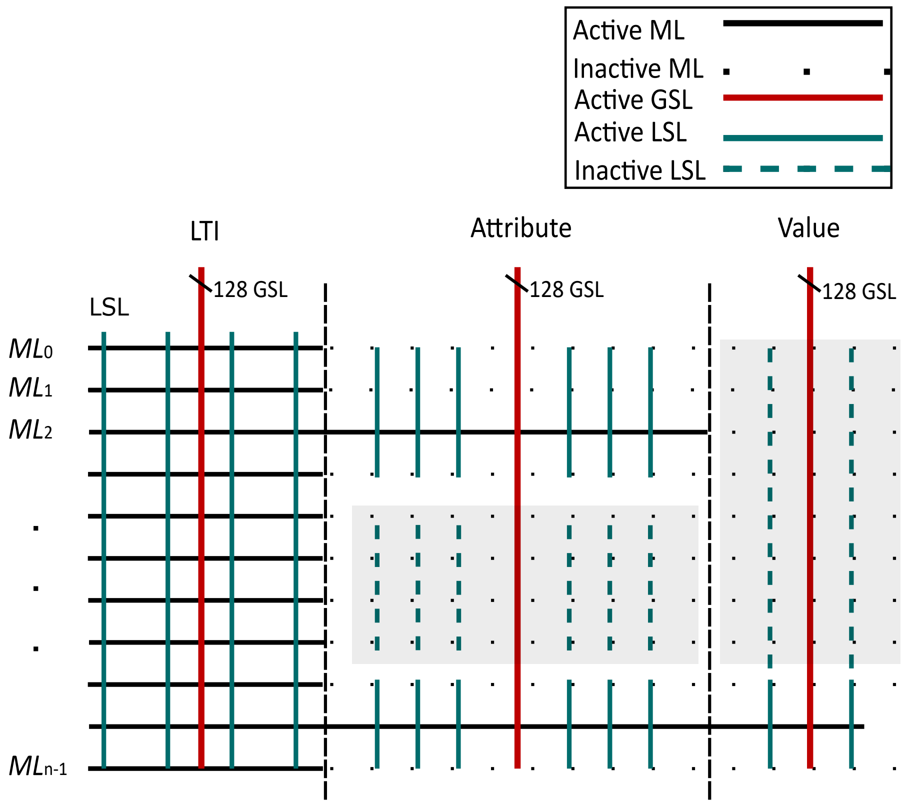


Figure 4.14 An illustration of a memory with pipelined matchlines and hierarchical selectlines; shaded areas are two examples of local segments which contain a subset of matchlines of a segment.

selection of data that is unique to a stored memory element, which is similar to the design of a good Hash function. However, a *collision*, which in this case is a multiple match in the first segment, will not affect the search latency or results but only incur a small power penalty. The specific design is a problem of its own and will not be covered in this thesis. Second, the matchline selective precharge scheme can be extended to include selectlines.

A hierarchical selectline structure is introduced in [119] where the selectlines are divided into multiple shorter local selectlines (LSL) in a way similar to the hierarchical bitline arrangement in the 4T-2R memory described in Chapter 3. Global selectlines (GSL) broadcast the search data into local selectline receivers which depend on enable signals from the previous segment to relay the search data signal to local selectlines. The enable signal from the previous segment is generated by collecting matchline outputs using a wired-OR circuit. A typical example of matchline and selectline activities is provided in Figure 4.14 for illustration. Consider each local segment has 16 rows. As long as there is a match in the previous local segment, the local selectlines in the subsequent local segment will be activated; otherwise both the selectlines and matchlines will be deactivated in subsequent segments as shown by the shaded areas in Figure 4.14. In the application of long-term memories, only a few memory elements with the same LTI are able to survive the initial search in the LTI memory block and most subsequent memory blocks are expected to remain inactive in a memory search. A more detailed coverage of hierarchical selectlines is provided in [119, 115]. The application of this technique to the new CAM is thus to avoid unnecessarily charging selectlines in subsequent segments to further improve dynamic power consumption.

4.4.2 Pre-Computation

A binary CAM supported by pre-computation stores extra information in each word for use in the search operation [38]. Typical data for this extra information is the *Hamming weight* (the number of '1's) of the rest of the data word. This extra information serves as the first segment in the initial search as illustrated in Figure 4.15. When a search cue is passed to the memory, the Hamming weight is counted and will be the search data in Segment 1; the matchlines in Segment 2 are selectively precharged depending on this initial search results.

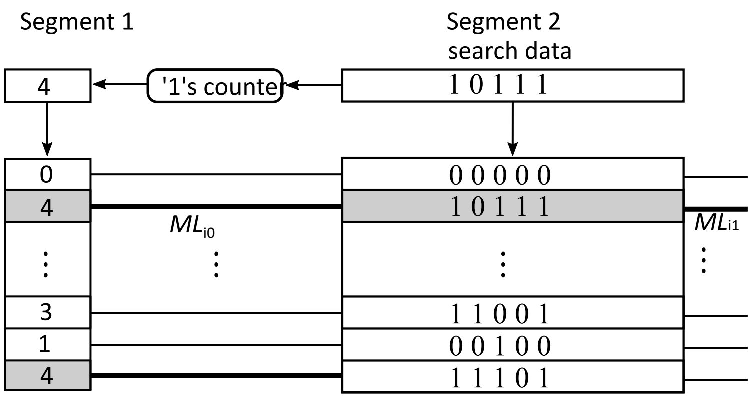


Figure 4.15 A conceptual view of a CAM with pre-computation [38]; the shaded areas indicate a match and the bold solid lines indicate a precharge activity

CAMs make use of pre-computation to reduce power by selectively activating memory blocks. Published results show that the energy consumed in the added peripheral circuits can be amortised by the power saved in memory blocks [120]. It is interesting to observe that rows precharged for search in Segment 2 have the same Hamming weight and this is used to simplify the comparison circuits in the binary CAM cells. A mismatched row in the second segment always has the same Hamming weight as the search cue. In a conventional CAM cell of ($\overline{D}$, D), the memory only needs to evaluate one of the pull-down paths in a cell as illustrated in Figure 4.16. The same 1's count ensures that a mismatch will always have a pull-down path to ground. It should be noted that the cells in Segment 1 still need normal comparison circuits because of the lack of this property.

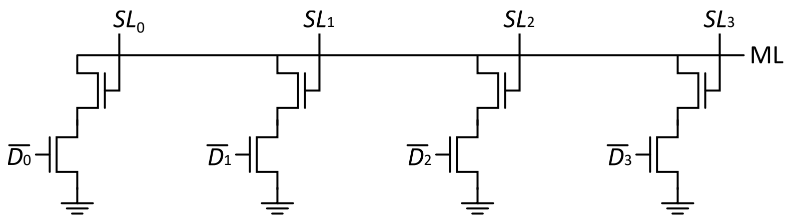


Figure 4.16 An example of the simplified comparison circuits of four cells along a matchline; only one pull-down path is used for a search in each cell [38]

The use of pre-computation is well suited to the resistive CAMs proposed in the thesis, including 4T-2R CAM and the crossbar CAM using 1S-1R RRAM cells. I will consider the latter as an example in this section. Recall that each of the crossbar CAM cell groups two RRAM cells for storing a pair of complementary binary values. With the pre-computation scheme, it is feasible to use just one RRAM cell for a search such that a comparison between (Q, $\overline{Q}$) and (D, $\overline{D}$) is now possible by just comparing Q and D. Now, the memory core is identical to the arrays in RRAMs and half the size of the CAM proposed earlier in this chapter. An encoding scheme without the complementary storage bit and selectlines is provided in Table 4.5.

Table 4.5 Data encoding of simplified cells in binary CAMs with pre-computation

Stored Data		Search Cue		Match	
Logic	D	Logic	Q	Status	ML
0	0	0	$V_{\mathrm{set}}/2$	Match	V_{read}
		1	Gnd	Mismatch	0
1	1	0	$V_{\mathrm{set}}/2$	Mismatch	0
		1	Gnd	Match	V_{read}

An example memory with four-bit words is illustrated in Figure 4.17. The search data is [0 1 1 0], which assumes that the Hamming weight is two in the pre-computation. All six possible combinations are stored in memory, each of which occupies four RRAM cells instead of twelve in the previous CAMs proposed. The simulated circuit confirms the correctness as analysed above.

The compelling part of this circuit simplification is the possibilities to reduce cell area and improve the matchline sensing margin, which is a key factor in building wide CAM arrays. The cell area reduction is obvious as shown in the transistor cell and the memristor

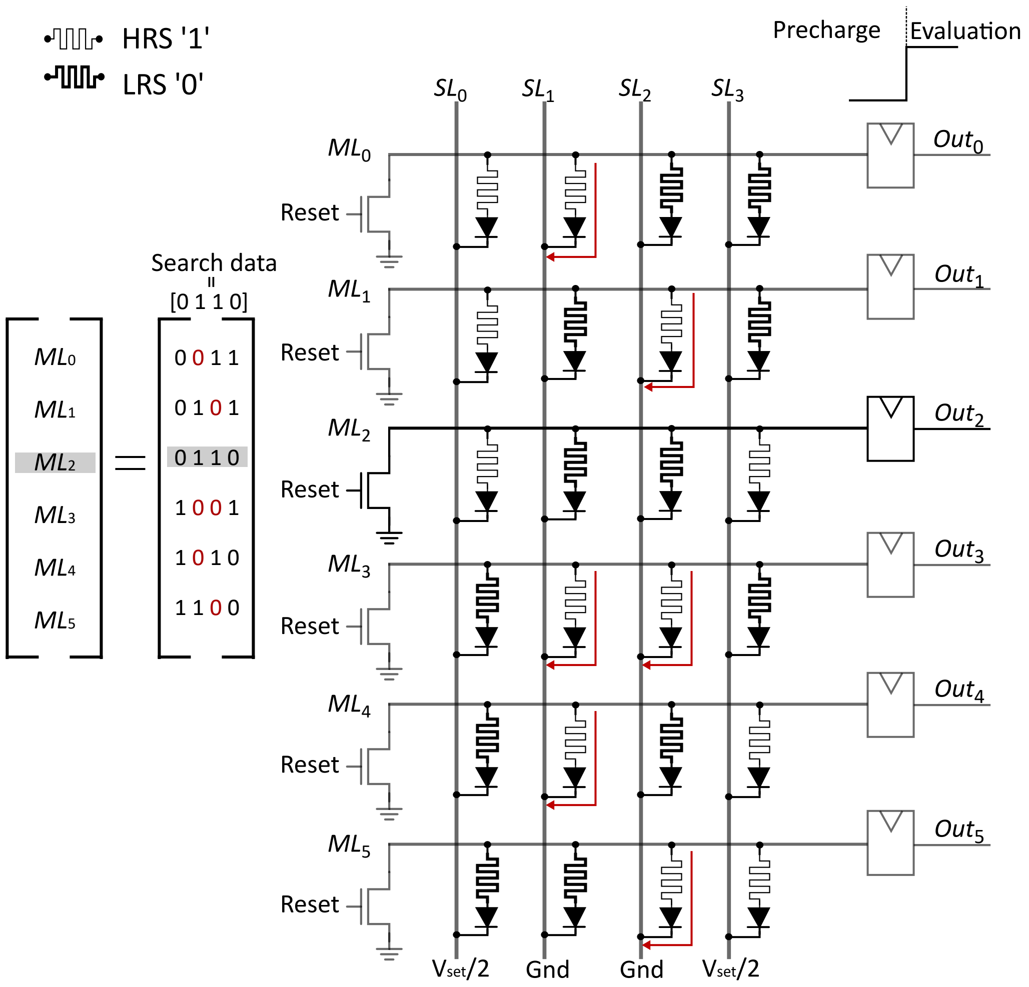


Figure 4.17 Simplified cells in a four-bit wide memory demonstrates search operations; red '0's and arrows illustrate the low resistance pull-down paths, and the shaded area indicates an exact match

cell examples. The possible improvements on matchline sensing margin arise from the conclusion in Section 3.4 that the sensing margin in search along each matchline depends on the matchline to ground resistance between the exact match row and the one mismatch row. The ratio in previous proposed CAMs is $1+\mathcal{R}_{HRS:LRS}/W$ when the both HRS and LRS resistance is significantly larger than the selectline loading resistance. W is the memory width. In this CAM with simplified cells, the matchline to ground resistance of the exact match row doubles because the number of pull-down paths is reduced by half. Thus the resistance ratio between a match row and a mismatch row increases to $1+2\mathcal{R}_{HRS:LRS}/W$. Depending on the device's $\mathcal{R}_{HRS:LRS}$ and the memory width configuration, this change could lead to a boost in matchline sensing margins. For instance, for a memory width of 128 bits and $\mathcal{R}_{HRS:LRS}$ of 128, the sensing margin can theoretically have a 50 % increase.

4.5 Summary and Conclusion

This chapter proposes a compact resistive TCAM based on the crossbar structure. With two RRAM 1S-1R cells representing a CAM cell with a pair of complementary bits, the same search operations as a conventional CAM can be realised by properly controlling matchlines and selectlines. It should be noted that the control over these memory lines is adopted from the same half-biasing scheme used in suppressing leakage current in large RRAM arrays.

CAM operations on crossbar arrays including write, read and search are demonstrated. Because of the commonalities shared by RRAMs and this CAM, write and read circuit techniques in RRAMs can be reused. In addition to the voltage-mode sensing analysed, the chapter shows that it is possible to read a row of data by searching a particular cue against this row bit-serially. The main reason to justify this relatively slow read scheme is the elimination of extra read sense amplifiers inside memory arrays.

A small memory array circuit was simulated in SPICE. The results show that as the array becomes wider and deeper, the peripheral circuits begin to dominate total energy consumption. However, the energy/bit/search is comparable to current volatile CMOS implementations. A conservative area estimation using stick diagrams shows that the proposed CAM achieves over an order of magnitude density improvement compared to the state of the art MOS counterparts reported in the literature. Further implementations that hide peripheral circuits under the crossbar layer promise even more compact memory arrays.

The results for small memory arrays were used to analyse larger memories. Key delay components include search in memory subarrays, multi-match resolution delays and routing delays. Energy estimates show that memory subarrays and their peripheral circuits consume 98 % of the total energy. For a CAM with four million long-term memory elements, the proposed CAM consumes about 10 % of the energy of a conventional CMOS CAM but has a slightly larger search latency. However, the latency of a memory retrieval is significantly improved when compared with the current software approach; the estimation in the case study shows that a retrieval of a modest size memory object is in the range of microseconds, while the latency of a software based retrieval is in milliseconds.

The last section of this chapter focused on reviewing existing circuit techniques applicable to the resistive CAMs proposed in this thesis. Two CAM techniques are shown to be particularly suited to the resistive CAMs: matchline pipelining using selective precharge schemes; and pre-computations before search. Matchline pipelining reduces the energy per search by reducing the switching activity of circuit blocks. Mismatch rows can be detected and disabled in an earlier stage thus saving energy consumptions. Pre-computation also reduces the energy per search. More importantly with crossbar resistive CAMs, using pre-computation is possible to significantly simplify the cell circuit and save the total chip area. The simplified cell circuit also promises to give a better matchline sensing margin.

Chapter 5

Activation Circuits

Chapter 3 and Chapter 4 explained how to build compact CAMs so that information can be densely stored and quickly searched. Semantic memory is more than just an information storage and retrieval system. Long-term memories in advanced cognitive architectures, such as Soar and ACT-R, include biasing mechanisms mimicking the human memory recall process; the retrieval of knowledge is biased by how recently or frequently that knowledge has been accessed. This is achieved by maintaining an activation value for every object in memory. When a memory search matches multiple objects, the one with the highest activation is returned first.

State of the art software realisations of cognitive architectures use carefully optimised databases for their long-term memory as well as the activation biasing schemes. As discussed in Chapter 2, these become a serious constraint for real-time agents or agents that persist for a long time. This chapter shows compact activation circuits can be distributed into CAMs to support cognitive functions in memory. The memory system uses the resistive CAM from Chapter 4 as the storage and search engine as shown in Figure 5.1.

Recall that a search cue passed to the semantic memory is compared against all rows of a record table in CAM at the same time. When a cue only contains partial information about a memory object, for instance, a value of an attribute of an augmentation, the search may generate multiple match records. These will then be passed to the activation circuit to bias the retrieval result, in a similar way to a multi-match resolver in a conventional CAM.

The activation circuit is different to a priority encoder resolver in two ways:

1. the activation circuit uses dynamically updated values and the updating scheme is necessarily a two-step process. When a search cue is passed to the semantic memory,

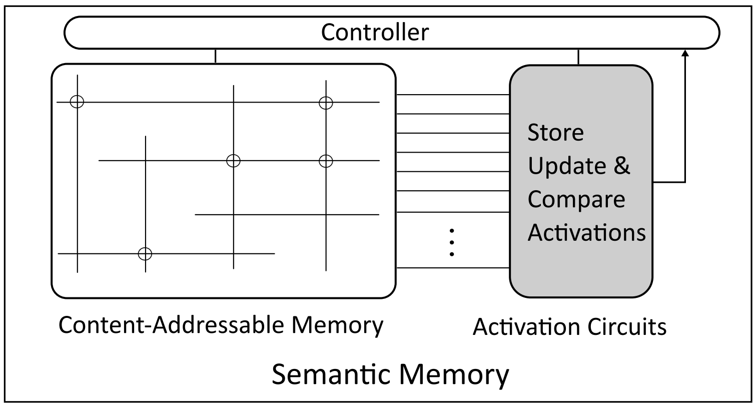


Figure 5.1 Block diagram of the semantic memory in cognitive architectures

the memory first performs a memory search, possibly generating multiple candidates which are biased by their activation values for output; then the activation values are updated based on the outcome of the current search.

2. If the activation circuit generates an incorrect result, it is not a problem. The role of the activation circuit is to order the search results. Sub-optimal ordering may have a performance impact but will not affect the correctness of the system. The situation is analogous to a data cache in a conventional computer system. Cache misses affect system performance, but not correctness

The objective of the present work is to provide hardware support to enable real-time agents with very large long-term memories. For activation blocks in the semantic memory, it is necessary to use dedicated activation circuits to overcome the computational complexity of activation updating in software implementations. One possibility is to build a centralised digital activation processor that computes activation values of multi-match candidates in a sequential manner. Alternatively, one might use a distributed approach, in which a small circuit is associated with each object in memory to store and update its activation. The latter approach is the theme of this chapter.

The remainder of this section reviews related work on a hardware supported activation scheme which stores timestamps in a buffer for the calculation of each activation value. Based on this scheme, an area optimised version of timestamp circuits using memristors is proposed in Section 5.2. The latter part of the chapter considers the analog behaviours of memristors and their application to building activation circuits. Section 5.3 describes a compact circuit with a single analog memristor which is used to store, update and compare the activation value for a memory object. The last section of this chapter provides results of SPICE simulations and a case study.

5.1 Related Work

Recall from Section 2.1.5 on page 20 that base-level activation (BLA) is a commonly used activation scheme that has proved effective for cognitive models [6]; however it is computationally expensive to implement [33].

In an early, unpublished, attempt to support BLA in hardware, members of my research team considered a digital approach using conventional CMOS circuits. Their resulting *timestamp circuit* provides a useful introduction to BLA calculation and is the conceptual starting point for the memristor circuits described later in this chapter.

The authors of the timestamp circuits reformulated the BLA equation Eq.5.1 by assuming that semantic memory accesses occur uniformly in the time period of T_s and it would be sufficient to approximate the activation value by only using the history of the last w memory accesses. Consider the activation of a memory object at time $t = kT_s$, where k is some random number. a_j is defined to be 1 if the object was accessed $j+1$ time periods ago, otherwise $a_j = 0$. The BLA equation (see Eq.2.1 on page 20) can be reformulated as:

$$B = \ln\left(T_s^{-d}\right) + \ln \sum_{j=0}^{w-1} a_j c_j \tag{5.1}$$

where both $\ln\left(T_s^{-d}\right)$ and c_j are constants for a given value of d ($c_j = (j+1)^{-d}$). Thus, for determining the object with the highest activation, it is sufficient to compare their values of

$$\hat{B}_i = \sum_{j=0}^{w-1} a_j c_j \tag{5.2}$$

Considering c_j is a constant, only a_j needs to be maintained as a w-bit timestamp value $(a_{w-1}, ..., a_1, a_0)$ for each memory object. In hardware, a circular buffer is used to store the timestamp for each object in memory, as shown in Figure 5.2. The authors further proposed a more area efficient approach using SRAM for storing and updating timestamp values.

An illustration for testing the effectiveness of this reformulation (Eq.5.2) is shown in Table 5.1, where a 4-bit timestamp is used and the decay factor $d = 0.5$. The constant c_j is precomputed for each time interval. As can be seen from the table, the most active rows are those that have been accessed both frequently and recently (e.g. the row ranked first with (1, 1, 1, 1)). Rows accessed infrequently or less recently have lower values of activation. This

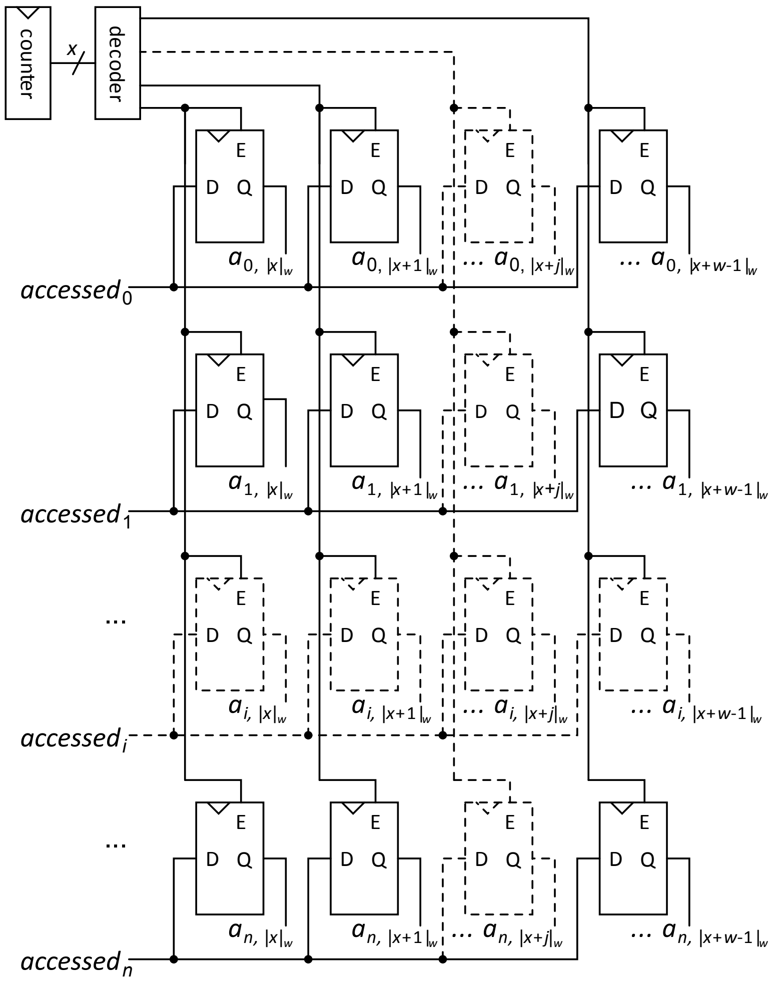


Figure 5.2 Timestamp circuit using a circular buffer of registers. ($|x|_w$ indicates x mod w; $a_{i,j} = 1$ indicates memory object i was accessed at time interval j.)

oversimplified 4-bit access history size is for illustration only, and a record of access history over a wider window will provide a higher precision for resolving multi-match issues.

5.2 Timestamp with Digital Memristors

The semantic memory needs to store tens of millions of memory objects as analysed in Chapter 2. Consider a 10-bit window of access history size for sufficiently maintaining an activation value for each memory object as used in [34]. This leads to hundreds of millions of bits dedicated to storing timestamps alone. Although memory is generally assumed cheap, I propose to use the two-state digital memristors as an alternative storage solution to replace

Table 5.1 A 4-bit timestamp for calculating activation values

Precomputed c_j	1.0000	0.7071	0.5774	0.5000		
Timestamp	0	1	2	3	$\hat{B}$	Rank
	1	1	1	1	2.7845	1
	1	1	1	0	2.2845	2
	1	1	0	1	2.2071	3
	1	0	1	1	2.0774	4
	0	1	1	1	1.7845	5
	1	1	0	0	1.7071	6
	1	0	1	0	1.5774	7
	1	0	0	1	1.5000	8
	0	1	1	0	1.2845	9
	0	1	0	1	1.2071	10
	0	0	1	1	1.0774	11
	1	0	0	0	1.0000	12
	0	1	0	0	0.7071	13
	0	0	1	0	0.5774	14
	0	0	0	1	0.5000	15
	0	0	0	0	0.0000	16

the hundreds of millions of bulky SRAMs (as shown in Figure 5.3) because of memristors' inherent register capabilities. Moreover, the peripheral circuitry used for a crossbar memristor array can be reused to achieve a w-bit circular buffer. In this way, the area overhead will be significantly reduced, and little effort will be needed to re-design the peripheral circuity. Based on the memristive bit cell, an alternative approach using memristive logic operations is investigated.

The idea of using a circular buffer for updating the timestamp is maintained. The counter and decoder are replaced with a self-decoding ring counter. The memristor memory element can be built in the junction of a crossbar structure, which is for area efficiency and consistency with the semantic memory proposed in preceding chapters. Two phases are required to update the timestamp. At the beginning of every cycle, a reset operation is performed to clear $a_{i,0} = 0$ for all i. The second phase sets $a_{i,0}$ to 1 whenever the corresponding memory object has been accessed in the previous memory cycle, i.e., $access_i = 1$.

Specifically, all objects with $accessed = 0$ are reset to 0 in parallel in the reset phase; this is done by applying an appropriate voltage bias across the memristors in the cell bits selected by the ring counter as illustrated in Figure 5.4. At this phase, the memristor in the row with the status $accessed = 1$ is left unchanged since there is no voltage drop across it.

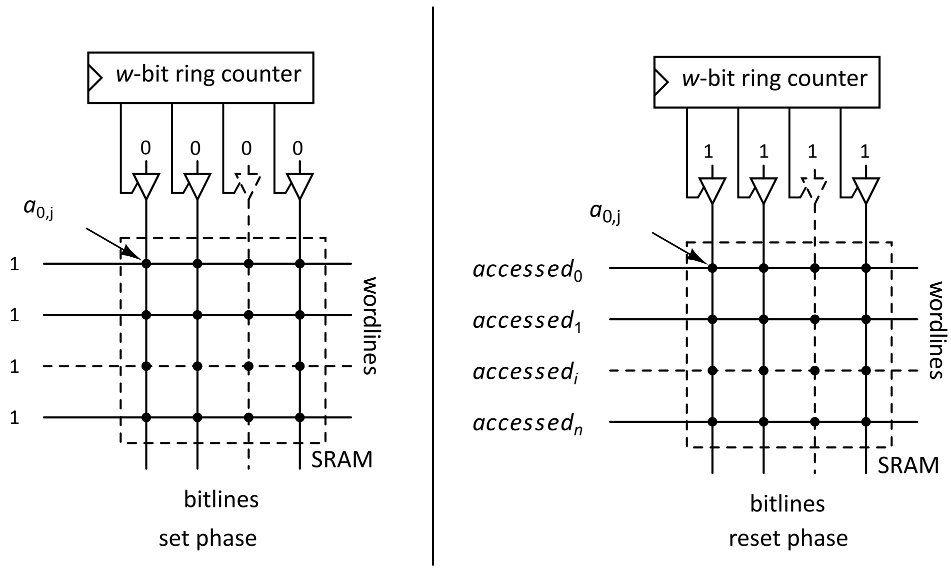


Figure 5.3 Timestamp circuit using SRAMs

In the set phase, different voltage bias is applied across the memristor in the accessed row, which sets its resistive state to represent '1'. Considering the fact that most objects are not accessed in every cycle, the majority of cell bits have the status $accessed_i = 0$. Thus, for the sake of reducing leakage current, memristors with HRS represent the access status '0', and LRS represents '1'.

5.2.1 Timestamp with Memristive Logic

Up to this stage, memristors are only used as storage elements for storing the access history. This section explains how the timestamp scheme can be realised using memristive logic computing. The density benefit is the main reason for exploring this approach.

A straight-forward way to realise the timestamp updating scheme is to use shift registers that shift the access history bit cells by one bit in every cycle. In fact, resistive devices have inherent register-like latching properties that enable them to hold internal states when not biased by high level voltages. Moreover, these stored resistive states are able to perform logic computations. These properties lead to the memristive version of shift registers that are able to update the timestamps of every memory object.

Recall from Section 2.4.3 on page 34 that $p$IMP$q$ is equivalent to the boolean logic $q \leftarrow \overline{p} + q$. With logic duplication (from p to a another memristor y) as shown in Figure 5.5(a), it is possible to build the shift operation: $S_{i+1} \Leftarrow S_i$ and $S_0 \Leftarrow accessed_i$. The operation

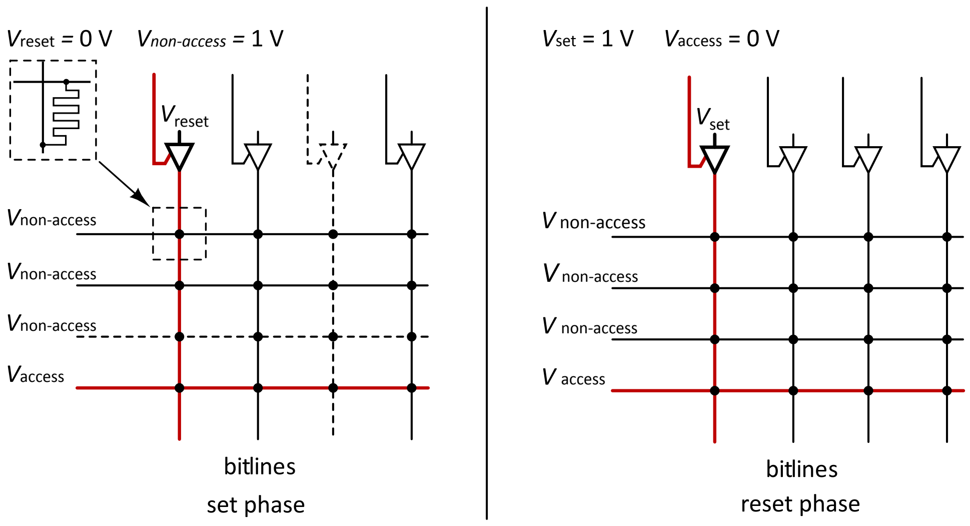


Figure 5.4 Timestamp circuit using memristors

is bit-serial because the bit cells share the same wordline, and memristors lack the gating ability that separates the cells when operated in parallel in this crossbar structure. As a result, a w-bit timestamp needs $w-1$ bit-serial logic duplications, which corresponds to $4(w-1)$ write operations.

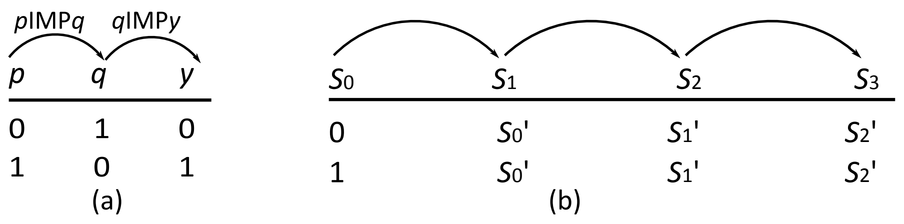


Figure 5.5 Logic duplications (a) and shift operations (b)

A small RRAM with in-memory computing operations suitable for this scheme is described in [121]. The shift operations are sequential, where $n-1$ number of SET and RESET operations need to be performed for completing n bits shift.

5.3 Analog Activation Circuits

So far in this thesis, memristors have been used solely in digital designs. CAMs used memristors in memory cells for storing binary values, and the previous proposed timestamp circuits only used binary memristors to represent only ‘0’s and ‘1’s thus each timestamp needs multiple memristors for storing the binary access patterns. However, the relation between an ideal memristor’s resistance and charge that has passed through it imply that a properly biased single memristor is able to provide analog values represented by its resistance.

The most popular applications of analog memristors are in neuromorphic learning where memristors have been shown to be more area-efficient in analog implementations of synapses models [85, 122, 123]. Typically, a memristor’s state changes in a synapse are used to record a specific learning outcome, which is determined by the state of the pre-synaptic neuron, the state of the post-synaptic neuron and the current state of the synaptic weight [122]. A memristor is good fit for this role because the change of its resistance also depends on three inputs: its current state and two voltages applied to its two terminals. Generally, local CMOS-based neurons are used to supply a train of potentiating voltage pulses to program the resistance of the synapse memristors. Voltage pulses with properly defined duration and width are able to precisely modulate the memristor resistance.

Two observations of memristors and activation models suggest that memristors are applicable to the design of an activation circuit. First of all, both memristor resistance and activation values evolve over time as a function of the external input and the previous state. For memristors, the bias voltage is the external input and the change of resistance depends on the previous resistive state. The change of the activation of a memory object also depends on the external access pattern and the previous activation value, although this change is at a different rate to that observed in memristors. However, the second observation of memristor models is that the non-linear relation between the the bias voltage and the change rate of the resistance provides a tool to tweak the dynamics. It should be noted that none of these observations support a faithful reformulation of the base-level activation model using memristors, but it reinforces the idea that it is possible to support an effective activation model approximated using memristors.

This section explores memristor analog behaviours in the design of an activation circuit which consists of two blocks: an activation cell and a comparison cell as arranged in Figure 5.6. When an ambiguous cue is passed to the memory, the search generates two match candidates: for example, $Object_1$ and $Object_{\text{N-3}}$. The memory then compares these two objects’ activation values based on past access patterns. The winner of this comparison, say $Object_1$, will be successfully retrieved and its activation will be updated in the corresponding activation cell.

An activation cell is attached to the end of each memory row. The activation cells store and update activation values using input signals from the CAM search output, the activation signal $\overline{Act}$ and the deactivation signal $\overline{Deact}$. The activation values of a memory object is stored in a single memristor with continuous resistive states. The updating scheme uses $\overline{Act}$ and $\overline{Deact}$ to drive the memristor's resistance lower or higher to approximate the updating dynamics in BLA.

Apart from having activation storage and update functions, the memory also needs to compare the activation values from match candidates and find the highest. A typical CMOS approach would be to first convert the analog resistance values to digital signals, and then find the highest value by employing a winner-take-all network. I will show in Chapter 6 Section 6.3.5 on page 119 that the CMOS winner-take-all network would not only occupy significant chip area but also introduce a delay linearly proportional to the memory depth. Considering that activation values of each memory object are stored as different resistance in memristors, I use memristor resistance to build a voltage-mode read circuit for each row so that the greatest activation can be discerned from the RC delay triggered by the read signal in Figure 5.6.

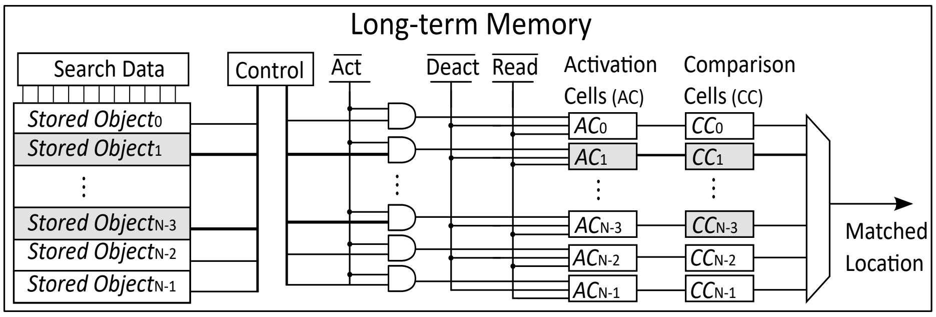


Figure 5.6 Block diagrams of activation circuits for storing, updating and comparing activation values

The core of this activation circuit is a voltage-controlled analog memristor, which stores the activation value for the row. Like the synaptic memristors, the resistance of each activation memristor can be adjusted over a wide range of values by applying voltage pulses [67]. To reduce power dissipation, memristor conductance is used to represent activation, with low conductance corresponding to low activation. At any time most of the rows in memory will be inactive, their activation memristors will have low conductance, and will dissipate little power.

The activation cell is shown in Figure 5.7. It supports three operations:

Activate: a positive voltage pulse greater than the memristor positive threshold voltage V_{TP} is applied to a row when it is accessed to significantly increase its activation value;

Deactivate: a train of negative voltage pulses, which exceed the negative threshold voltage V_{TN}, is applied constantly to all rows in parallel, to decrease all activation values over time;

Read: a voltage pulse is applied for a short period to compare the activations and find the most active row. Although larger than V_{TP}, this pulse has negligible effect on the stored activation values.

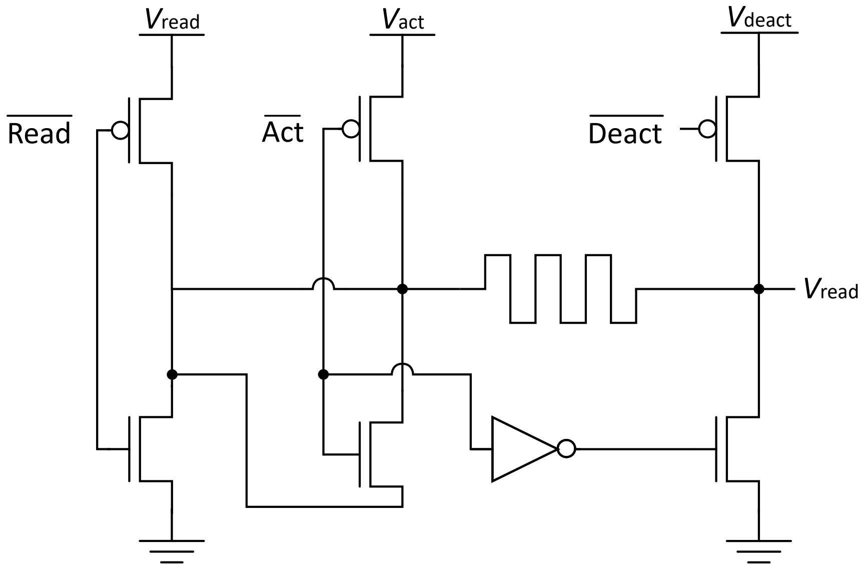


Figure 5.7 An activation cell (AC) for storing and maintaining a row's activation value

The circuit used to find the row with the highest activation is shown in Figure 5.8. In each row, the read voltage, V_{read}, will grow monotonically at different rates according to the conductance of the activation memristor. The first row to reach the input threshold of D-latch will cause the corresponding *mostActive* flag to be latched and sent to the encoder circuit. The $\overline{disable}$ flag will then be propagated to all rows to disable their latches. The characteristic circuit of this RC model shares some commonalities with the CAM discharging model where the determining factor is the total matchline to ground resistance instead of the resistance representing activation values.

It is possible for two or more rows to tie and all be flagged as the most active. This is not a serious problem as the activation value is a heuristic. Agents are designed to be tolerant of multiple memory matches. Like a cache miss in a conventional memory hierarchy, an erroneous activation value or comparison may incur a delay but will not cause the system to fail. This also means that the circuit is tolerant of activation errors arising from memristor variations.

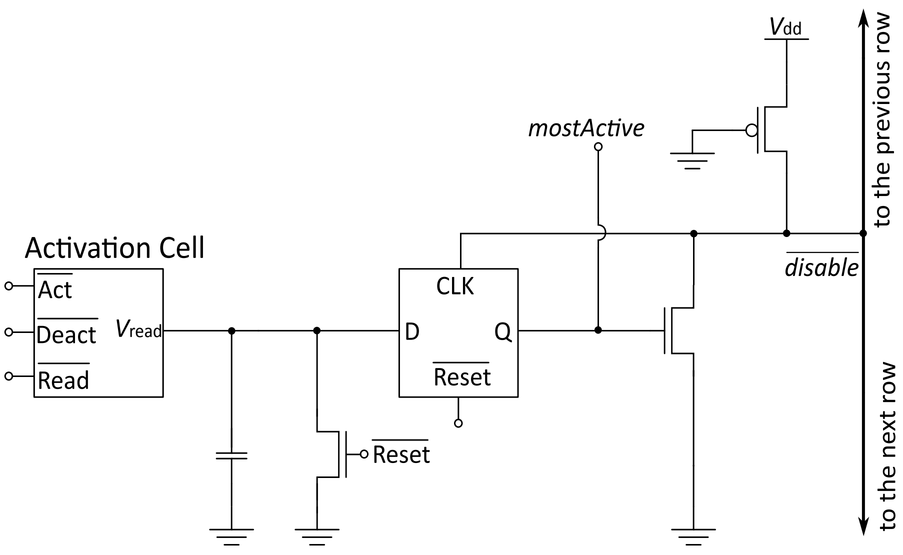


Figure 5.8 A comparison cell (CC) for finding the most active row

5.4 Simulations and Estimations

The activation value of a row should be a function of how recently and how frequently it has been accessed. Various activation functions have been used [34] but the base-level activation function in ACT-R [33] has proved successful and will be used here as a baseline for comparison.

SPICE simulations were performed using a memristor model extracted from a fabricated device [67] and parameters for a 45 nm CMOS process with a supply voltage of 1 V from FreePDK [109]. Figure 5.9 shows the activation function for the memristor activation circuit alongside theoretical BLA when a row is accessed every 100 ms. Deactivation pulses of -1 V for 0.1 ms are applied every 0.4 ms. A 1.8 V activation pulse is applied for 1.5 ms whenever the row is accessed. The memristor-based activation approximately follows the form of the BLA. Both increase significantly when a row is accessed and fall quickly just afterwards. This reflects the recency of access. Both also show a long term increase reflecting the frequency of access.

A circuit of 32 rows has also been simulated. Each activation row is directly hardwired to a matchline output in the CAM, thus each search triggers a row access. Rows are accessed periodically but at different pseudo-random rates between 8 and 33 Hz and for a different random number of accesses from 1 to 10. The experiment assumes that the combinational configuration of access frequency and access numbers in a fixed time frame will illustrate the dynamics of memory activations based on different schemes. Figure 5.10 shows the

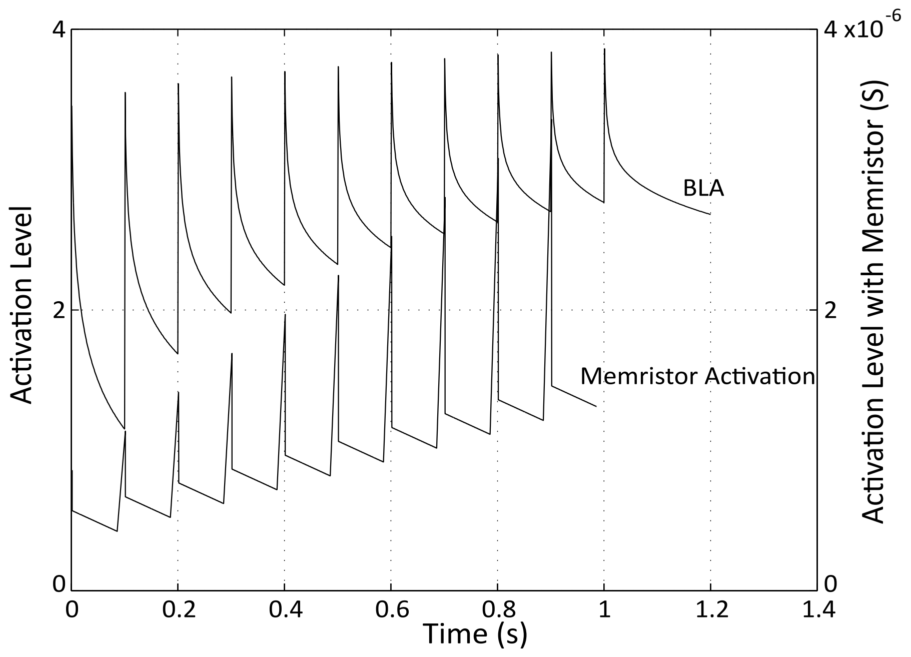


Figure 5.9 Comparison of BLA and memristor activation in the case of uniform memory access

activation values for 4 out of the 32 memory rows. The experiment results are gauged at the time point of one second. The combination effects of frequency and recency makes the row highlighted in red line the most activated row. The activation values for these 4 rows are shown in Table 5.2 along with theoretical values for BLA. At $t = 1$ s the memristor activation function assigns row 2 a higher activation value than row 3; BLA gives row 3 a higher activation than row 2.

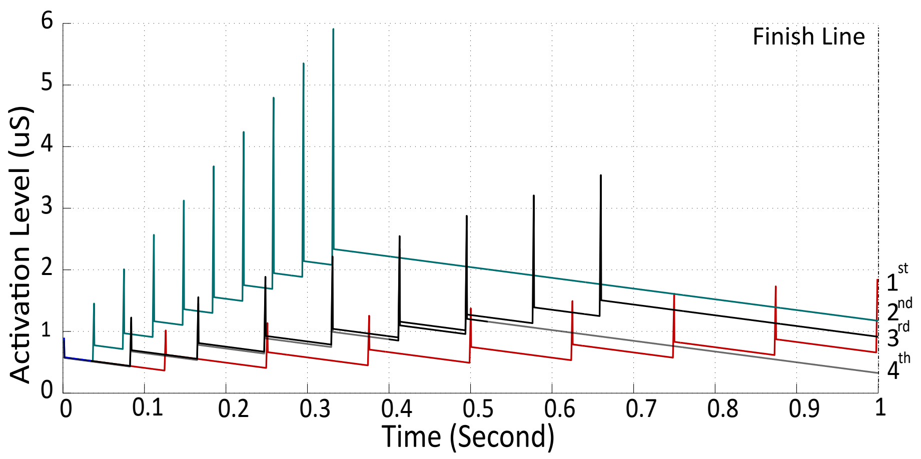


Figure 5.10 Memristor-based activation values for four rows with different access patterns

Table 5.2 Results after updating 4 rows' activations over a 1 second period; the ranking is based on the proposed activation results

ranking	period of a cycle (μs)	no. of access	*BLA*	proposed activation ($\mu\Omega^{-1}$)
1	124800	9	3.62	1.85
2	36800	10	2.4	1.17
3	82590	9	2.44	0.91
4	82590	7	2.1	0.33

5.4.1 Estimations

Energy consumption of the circuit can be estimated from simulations of a single activation row. Each row dissipates energy when performing one of the three operations: activate, deactivate or read. Figure 5.11 shows simulation results over the memristor's resistive range for a 45 nm process. Memristor states 0 and 1 represent two resistive boundaries: high resistance and low resistance respectively. The activate operation has the highest energy dissipation and a row tends to dissipate more energy as it becomes more active. In the worst case, applying an activation pulse to a fully active row can dissipate up to 174.87 nJ. However, most rows will be inactive as confirmed by the memory access patterns observed in [6]. In the experiment reported in that publication, among 821 492 long-term memory rows, only 31 280 rows were accessed, accounting for only 3.8 % of the total storage. Assuming that those accessed rows were fully active throughout the program execution, the activation circuit would consume 0.04J energy.

In simulation of the read operation, when a row is fully active, it triggers the *mostActive* flag after 6.7 ns. In the worst case where all the rows are inactive, this takes about 80 ns. A tie between the two most active rows occurs when the resistance difference between them is less than 4 kΩ.

A rough area estimation of each activation row is given using lambda-based design rules and stick diagrams [124, chap. 1]. Without custom placement, the size of the latch will determine the height of the row and it also occupies a large portion of the total length of the row. Assuming 8λ track pitch, an estimation by counting the number of routing tracks is shown in Figure 5.12. The latch is based on two tristate inverters, one standard inverter and a NOR gate, which occupies an area of $224\lambda \times 56\lambda$ in single-height cells. Without including memristors, which would sit in the wire stack above the CMOS circuits, the row has a $224\lambda \times 56\lambda$ geometry. This corresponds to the area of approximately 200 cells of the new CAM proposed in Chapter 4.

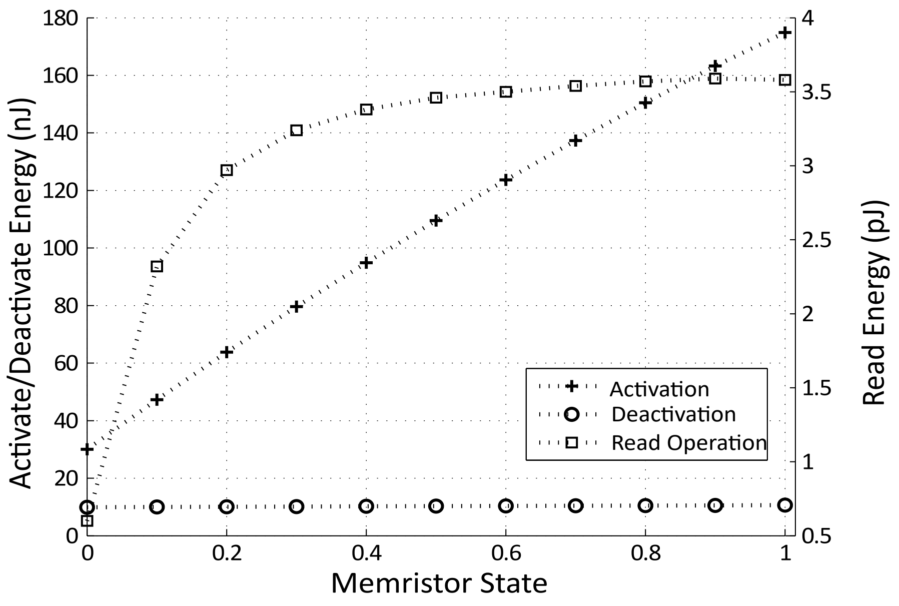


Figure 5.11 Single row energy consumption (average) in activating, deactivating and read operations

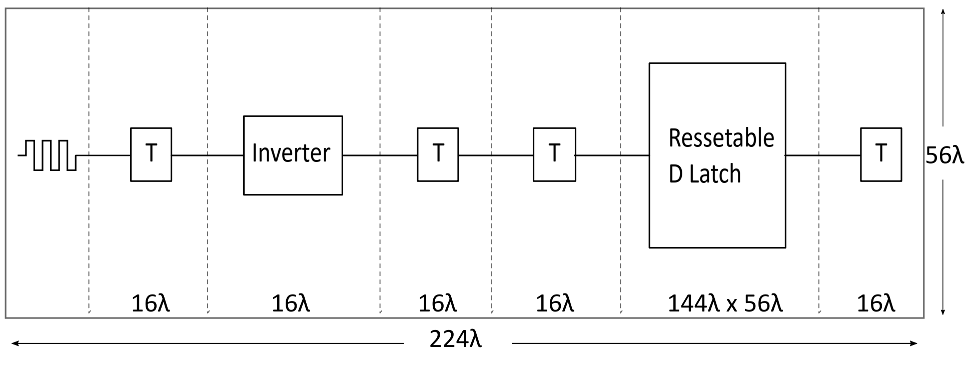


Figure 5.12 A rough area estimation of the activation circuits using lambda-based rules

5.4.2 A Case Study

To evaluate the effectiveness of this circuit-based activation scheme, it is necessary to test it in an appropriate benchmark task. Researchers in [34, 31] conducted experiments of various memory bias models in a word sense disambiguation (WSD) task, which provides a benchmark for evaluating different memory bias algorithms. This section uses the WSD task and the same dataset for a fair comparison. Reported experiment results on BLA are reproduced as a baseline.

WSD is an important and extensively studied research topic in the area of natural language processing. A typical processing flow is to query an agent using an ambiguous word which normally has a list of senses in a machine-readable dictionary; the agent accesses the dictionary and retrieves the sense of the word that is most likely to be correct. For example, the agent may be issued the text of a book about fishing, word by word. When it is given an ambiguous word, such as “bank”, it should identify the most likely possible meaning according to the context. In this case, the text is more likely to mean a river bank than a financial institution. Spreading activation assists this task because words already encountered in the text, such as “water” or “river”, will boost the activation of the desired sense of the ambiguous word. With respect to its application in cognitive systems, previous work uses the ambiguous word as a search cue, and a machine-readable dictionary as the long-term memory (LTM) as illustrated in Figure 5.13. In [34, 31], the test dataset is the *SemCor* textual corpus, which provides sequences of words and the associated correct sense of each word for verification [125]. The dataset stored in LTM is from *WordNet*, which is a lexical database that contains all possible senses for each word in SemCor [126]. The agent with a memory biased LTM retrieves a sense of the cue word and verifies its correctness with the correct sense tagged in SemCor. It is expected that the desired object (the sense of a word in this case) may not always be retrieved. Incorrect retrievals can spread activation to appropriate word senses and, in turn, trigger further incorrect retrievals. To avoid this, the query is read-only and the correct sense of each word is revealed to the agent after each read [34]. The result of each bias model is an accuracy rate based on the number of correct retrievals versus the total number of retrievals.

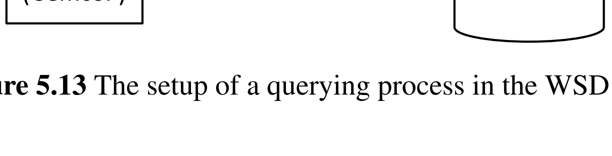


Figure 5.13 The setup of a querying process in the WSD task

Table 5.3 shows that the results reproduced for this experiment alongside the published results. Although the randomness in Soar's operator selection process leads to slightly different results in each run, overall the accuracy rates are highly consistent. This shows that the dataset and configuration is consistent with the published work.

Table 5.3 Reproduced results of three bias mechanisms in WSD retrieval tasks

	Derbinsky and Laird [34]		Reproduced	
	Run 1	Run 10	Run 1	Run 10
Recency	72.34%	74.43%	72.48%	74.52%
Frequency	71.69%	76.53%	71.72%	76.55%
Naïve	74.45%	78.47%	74.85%	78.63%

The memristor circuit model was used to replace the activation function in the the Soar cognitive architecture kernel. In the circuit model implemented in C++, a memristor's conductance represents an activation value and all activation values are initiated with 0.2 µS. This value is bounded in a range from 0.2 µS to 16.7 S. For every memory access, a retrieval function retrieves all this memory element's history (the number of access, timestamp etc.) and converts it to characterized signals for updating memristor's conductance. The conductance is returned as a new activation. As the model does not store intermediate activation values, each memory element's new activation needs to be recomputed using the full list of timestamps.

The results of experiments with different activation functions in [31] were reproduced and extended to show the effect of the memristor activation function. As shown in Table 5.4, the memristor activation circuit achieved good performance in a benchmark task that required 217 171 memory retrievals. Using the memristor activation function, the first retrieved object was the one being searched for 67.12 % of the time. This is not quite as good as base-level activation, which returns the desired object first 74.45 % of the time. The memristor circuit's benefits come at a small but acceptable cost in performance.

Table 5.4 Performance comparison to three memory retrieval bias in [31]

	Recency	Frequency	BLA	The circuit model
Accuracy	72.34 %	71.69 %	74.45 %	67.12 %

5.5 Summary and Conclusion

This chapter has continued the design of long-term memory by exploring the possibilities of a hardware-supported BLA-like memory bias model. The successful incorporation of such a hardware block will avoid the sequential evaluation of computationally complex arithmetic functions in current implementations.

Hardware designs dedicated to the activation bias mechanism have been missing in the literature. Design alternatives for both CMOS and memristive approaches are yet to be explored. This chapter concentrated on designing memristor-based circuits, that could be attached to the rows of a CAM semantic memory block, to store, update and compare activation values. Both digital and analog approaches were considered and investigated using memristors' digital and analog modes. The digital approach is mainly based on unpublished work from my research group, which keeps a record of the access timestamp for each memory object. One of the advantages of using memristors here is to replace the bulky SRAM cell with a more compact memristor for storing timestamps. The required shift operation can be directly implemented on the memristor arrays, although this leads to a sequential manipulation of memristors.

The analog approach achieves good circuit density as a single memristor can be used to encode the analog activation value for an object in memory. Much of the efficiency of this analog design is achieved by approximating the critical dynamics of activation updating activities. With a resistive activation value, it is also possible to use a simple voltage-mode read operation to discern the greatest activation from a table of memory objects. It is not as robust as the digital approach, but as a heuristic, incorrect activation values only incur a search delay. For real-time agents, cognitive architectures such as Soar require long-term memory search latency in the tens of milliseconds. The energy and latency results confirm memristor memories with millions of rows arranged in banks are viable with power dissipation in the order of Watts.

In the last section of the chapter, the effectiveness of this analog circuit was verified using a word sense disambiguation case study. The circuit-based activation model was reimplemented in software and used as the bias mechanism to decide the correct word sense stored in LTM. The circuit model is not as reliable as other reproduced results from the literature. However, the compromise in performance is small, and the proposed circuits support a parallel computation distributed in the long-term memory.

Chapter 6

Memristive Spreading Activation

The use of distributed high-dimensional random vectors, or *hypervectors* in cognitive semantic processing has been proposed in a variety of recent publications [127–130, 53, 131–133]. Kanerva [128] and Plate [127] proposed a computing architecture that lays the foundation of a possible infrastructure for cognitive computing. Their hypothesised computer systematically supports a compositional representation of entities using hypervectors and performs computations using a set of well-defined vector arithmetic operations. Their case studies showed that hypervectors with associated operations can be used to efficiently implement functionality that is otherwise computationally complex. This computing architecture has been adopted in a variety of domains in cognitive computing. For instance, Jones and Mewhort [130] used a similar approach to incorporate word meaning and sequence information into a model lexicon. Gayler [129], Levy and Gayler [134], Osipov et al. [135], Kleyko et al. [133] used hypervectors to construct their holographic neuron models for bio-inspired pattern recognition tasks. Kelly et al. [28] proposed a declarative memory model for ACT-R where memory items are constructed using distributed hypervectors for a recognition memory task. More recently, the underling hardware for supporting such a computing architecture was considered in [131, 132].

This chapter introduces the application of this high-dimensional computing (aka hyperdimensional computing) paradigm in cognitive memories by linking it with spreading activation. A new spreading activation scheme is proposed by combining ideas from high-dimensional computing, content-addressable memory (CAM) and activation. The goal of this scheme is to spread activation from one memory item to semantically bound items.

The approach hinges on the availability of efficient hardware for ultra-wide hypervectors and associated arithmetic operations. In the literature, each memory object is typically represented by a hypervector that is thousands of bits long. This increases the size of the memory required. CMOS memory cells are already relatively large so that in published examples of hyperdimensional classifiers the memories consume two-thirds of the total chip area [132]. The high dimensionality (i.e. high word-length) of hypervectors means that any vector arithmetic operations will expend more energy than typical word length operations, especially in an associative memory search.

Search performance can be improved by using CAMs as the associative memory. The CAMs proposed in Chapter 3 and Chapter 4 search all stored data words simultaneously and provide the search outcomes in a bounded time. The use of CAMs has proved effective in improving the latency of associative search using hypervectors [131]. Laiho et al. [131] and Rahimi et al. [132] demonstrated that CAMs, with word data represented by hypervectors, can be a key component in realising some human-like semantic processing behaviours. This can be a blessing for many applications that rely on quick and intelligent information retrieval from large memory stores. Recent research on resistive devices has shown possibilities of reducing CAM area and power in various ways [136]. This chapter shows how memristive CAMs can be used to efficiently store ultra-wide hypervectors and support semantic processing based on activations.

6.1 Background

Recall that in cognitive architectures, memory activation is used to bias knowledge retrieval so that objects that have been frequently or recently accessed are more likely to be retrieved first. Consider a typical knowledge search process as shown in Table 6.1. The memory objects are denoted by Latin letters A-E and their augmentations are represented by Greek letters (α-ε). If a query is passed to the simple memory store, two search operations are used: the first search selects candidate memory objects that satisfy the query constraints, and the second one searches within the candidate list and sets the object with the highest activation as the output.

A choice between different approaches of finding the best match memory object has been implicitly made here: the best match object is only found among a list of strong-match candidates, thus an object with a high activation does not guarantee that it will be selected for output; it needs to show strong correlation with the search query in the first place. This approach agrees with the biasing schemes used in both ACT-R and Soar's knowledge retrieval process [137].

Table 6.1 An example of cognitive search

Object store	Apply query constraints: $\{\alpha, \varepsilon\}$	Biased by activation: B > C > D > A > E
A: $\{\alpha, \beta, \varepsilon, \phi\}$	Match	Fail
B: $\{\alpha, \varepsilon\}$	Match	Successful
C: $\{\alpha, \gamma\}$	Mismatch	Not participated
D: $\{\gamma, \phi\}$	Mismatch	Not participated
E: $\{\gamma, \phi\}$	Mismatch	Not participated

One can view the activation step as a bio-inspired multi-match resolver. However, the activation step alone does not prime the memory objects semantically linked to the object being retrieved. The retrieved object needs to broadcast its influence by spreading activation to all related memory objects. A successful spreading activation scheme is expected to "extract knowledge that is latent within the structure of an agent's long-term memory" [138]. The general idea of spreading activation through a semantic network consisting of nodes (memory objects) is that when the activation of a node is increased, then the activation of connected nodes is also increased. Consider a semantic network of nodes $N_1 \ldots N_n$ with links $L_{i,j} \in \{0,1\}$. $L_{i,j} = 1$ denotes the existence of a link from node N_i to node N_j. Each node N_i has an associated activation A_i. Whenever the activation of a node N_i changes such that it is greater than a firing threshold–i.e. $A_i > F$–then the activation of all connected nodes N_j with $L_{i,j} = 1$ is adjusted according to $A_j \leftarrow A_j + S_j$ where S_j is the spreading activation N_j receives from source N_i.

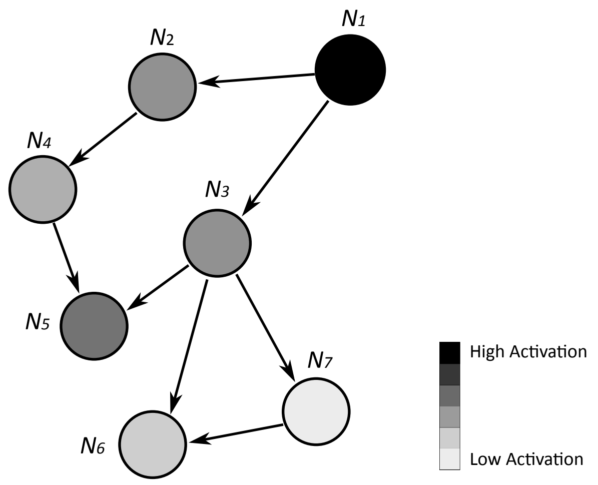


Figure 6.1 An activation spreading example in a simple semantic network

Related Work on Spreading Activation

To reduce the cost of sequentially traversing a network to spread activation values, Douglass and Myers [5] introduced a concurrent scheme for calculating activation values. This is achieved by including a middle-ware framework, through which a multi-threaded programming language called *Erlang* is able to maintain information of individual nodes concurrently. It should be noted that the actual activation calculations are performed serially outside of the database, hence significant communication overhead is expected.

SemMemDB [35] is an initiative that tries to take full advantage of the computational power of databases by performing the activation calculations within databases. The authors observed that the strength of association from node j to i is determined by edge numbers; the association between nodes is network dependent and can be pre-computed and cached in a separate table in the database. The pre-computation is computationally expensive, and though it seems to be a one-time cost for a stable memory store, extra updating mechanisms are needed for changing and evolving long-term memories.

A simplified algorithm from the Soar group [137] spreads activation in the direction of edges and determines the activation value of a recipient based on three factors:

- the initial activation of the source;
- the fan effect - a parent node divides activation equally among all of its children nodes;
- the decay factor, $p < 1$, which exponentially decreases the activation value as activation spreads to deeper nodes.

A simple semantic network is provided in Figure 6.1 to illustrate the flow of activation values and the calculation at the recipient nodes. N_1, as the source node, has an initial source activation of 1. The spread is limited to a depth of three and the decay factor is set to 0.9. The resultant activations of recipient nodes are represented as the intensity of their shades. The calculations of nodes N_2, N_4 and N_5 are:

$$
\begin{aligned}
S_{N2} &= 1 \times \frac{1}{2} \times 0.9 \\
S_{N4} &= S_{N2} \times 1 \times 0.9 \\
S_{N5} &= S_{N4} \times 1 \times 0.9 + S_{N3} \times \frac{1}{3} \times 0.9
\end{aligned}
$$

To prevent unbounded calculation of all linked nodes, the control parameters restricted the spread to a predefined depth and number of nodes. To further reduce the computation workload, a scheme called *Candidate-Only Processing* (also included in ACT-R) is used to spread activation only when there is a small number of nodes that ambiguously match the search cue. In other words, no spreading activation is computed when the cue is unambiguous. Much like the idea of the pre-computation in [35], Jones et al. [137] reduced computation workloads of activation by leveraging the observations made on the processing in long-term memory. For instance, they observed in some applications that long-term memories change only slowly, thus the trace of traversal can be cached for updating activation values.

The software approach often relies on a complex stack of software and hardware including cache hierarchies, and sorted data structures, to achieve adequate search performance. A case can be made for a simpler and more systematic approach that can efficiently maintain activation values. This approach presented in the following sections turns away from software algorithm optimisations and approaches the activation biased memory using a regular, flat, specialised memory hardware that can scale to large memory stores in terms of response latency and power consumption.

6.1.1 Hyperdimensional Arithmetic

Hypervectors are typically in the range of thousands of bits wide. For many applications, this brings enough randomness for creating uncorrelated vectors. For instance, the representational space of 1000-dimensional binary vectors consists of 2^{1000} patterns; a newly created random hypervector will be nearly orthogonal to all other hypervectors that are already in use.

Based on the easiness of creating random representations with hypervectors, Plate [127] and Kanerva [53] formulated a set of vectors algebra operations to create unique hypervectors, correlate multiple hypervectors into one distributed vector, and re-construct constituents from a distributed vector. Potential applications of this hyperdimensional arithmetic have been demonstrated in a series of cognitive semantic processing examples, which include simulations of CMOS circuits to perform the hypervector operations in memory [139, 131].

Two hypervector arithmetic operations are used in this chapter: *majority-sum* (also called *bundling* or *thresholding* in the literature) and *permutation*. Consider three unique hypervectors α, β, and γ. The majority-sum correlates these three hypervectors by performing a majority operation, written as $M = [\alpha + \beta + \gamma]$. An illustration of this operations using 12-bit representations is shown in Table 6.2.

Table 6.2 An example of a 12-bit majority-sum operation

Objects	12-bit hypervector representation											
α	0	1	0	1	0	1	0	1	0	1	0	1
β	0	0	1	0	0	1	0	0	1	0	0	1
γ	1	0	1	1	1	0	0	0	1	1	0	0
M	0	0	1	1	0	1	0	0	1	1	0	1

An important property of this operation is that the sum vector M has the same dimensionality and is similar to all input hypervectors. Also, if a sum vector M contains multiple copies of any component vector, the resultant sum vector is closer to this dominant component vector. Without permutation, M is a bag of data, i.e., two sets of inputs with the same vectors in different sequence will not be discerned. For instance, there will be no difference between $[\alpha+\beta+\gamma]$ and $[\gamma+\beta+\alpha]$. A permuting operation is needed before the sum operation to permute each input hypervector according to their order in the sum operation. The representation after permutation and sum will be $[\alpha+p(\beta)+p(p(\gamma))]$, where p denotes the permutation. One way to realise the permutation is using circular-shift operation, which creates unique hypervectors dissimilar to the original ones. These two operations are essential for encoding local database records to distributed hypervectors. For example, given a unique hypervector for each letter `d`, `o` and `g`, the representation of the word `dog` can be constructed from the hypervectors of these three compositional letters. In a hierarchical encoding process, the representation of `hotdog` can be constructed from the representations of `hot` and `dog`. Thus a compound vector of the complete record `hotdog is-not dog` can be constructed from its three fields `hotdog`, `is-not` and `dog`. How close is the compound vector to the component `hotdog` vector? The Hamming distance measures the number of bit positions in which they differ. If a bit of the compound vector is a 1 and the same bit of `hotdog` is a 0, the same bit in `is-not` and `dog` have to a 1 to constitute a majority, which happens with probability 0.25 in this case. The expected Hamming distance between any compound vector (M) and one of its K number of components (A) is:

$$\delta(M,A) = \frac{1}{2} - \binom{K-1}{(K-1)/2}/2^K$$

where the $\binom{K-1}{(K-1)/2}$ is the binomial coefficient. This equation indicates that as the number of components K increases, more noise is added to the compound vector M and its normalised Hamming distance with any component vector will approach 0.5, which makes it impossible to be discerned. This essentially refers to a capacity issue of hypervectors in bundling. A recent detailed study clarified many aspects of capacity of hypervectors in [140]. The number of components K is set to be 3 in this chapter.

The use of hypervectors and their arithmetic in the realisation of a spreading activation scheme is introduced in Section 6.2. An overview of hyperdimensional computing is provided in Appendix B.

6.2 A New Approach

For the sake of clarity, let us just consider the process of updating the overall activation of a memory object using the spreading activation term. Other components that may change the overall activation, such as decay over time, can be incorporated using similar mechanisms.

The overall activation value for object i is $A_i \leftarrow A_i + S_i$. The activation S_i spreading from source j to memory object i, as defined in [141], is a product of the weight of the source and the strength of association S_{ji} between the source and the object. Since a single search cue is used as the activation source in each search, the equation is reformulated by assigning a unit weight to each source. Thus, the amount of activation spread to a memory object is only proportional to its association strength with the search cue. The association strength in this case is measured by the level of *similarity* between the search cue vector and memory object vectors. In the literature, Hamming distance is typically used as a measure of similarity [132, 133]. To be consistent with the bit comparison operation in the proposed CAM cells, the similarity of two vectors with dimension D in this thesis is defined as:

$$Similarity_i = \sum_{k=0}^{D-1} \text{XNOR}(Cue[k], Object[i][k]) \tag{6.1}$$

where k is the k-th bit of Cue. In the theoretical model implemented in ACT-R, the strength of association between memory objects is set to

$$S_{ji} = S - \left(\ln \left(\frac{1 + Outedges_j}{Edges_{ji}} \right) \right) \tag{6.2}$$

where S is a constant parameter, $Outedges_j$ and $Edges_{ji}$ are the number of edges from object j and the number of edges from object j to object i.

The strength of association S_{ji} in Eq.6.2 shows that the edges connecting the memory object determine its association value with other objects, thus it is safe to say that S_{ji} is network structure dependent. In fact, several publications take advantage of this to reduce the computation cost by pre-computing these association values [35, 137]. In the hyperdimensional spreading activation scheme, as objects' hypervectors are hierarchically encoded from each object's compositional elements, the associations between objects are predefined by the number of common elements they share. The search cue is the source of the spreading activation but also can be viewed as an object. The amount of activation spread to other memory objects is the association between the search cue and the stored objects. As a result, the association S_{ji} is computed in the associative search operation. If the contents of memory changes, there is no need to re-compute S_{ji}. This is in contrast to the pre-computation schemes in which the S_{ji} must be updated to reflect changes in association between memory objects.

To facilitate the spreading activation scheme, the majority-sum operation is used to group similar elements to create correlated memory objects. With an associative memory, the association information could be gathered during the search operation. Section 6.3.1 describes the circuit-level design of the associative memory.

An Illustration

To quickly illustrate the capabilities of this new memory model, this section considers a recognition task using knowledge from a simple semantic store that contains a fragment of knowledge about animals, as illustrated in Figure 6.2. The connection and description between each memory object (each node in Figure 6.2) is consistent with the rules described in [28].

A MATLAB program was used to simulate the hypervector processing. The generation of low level hypervectors involves converting string constants to specific hypervector representations. To represent a higher level memory concept, conditional permutation and majority-sum operations are applied to hypervectors of its compositional components.

Recall how the `hotdog is-not dog` example was constructed from letters to record fields and then to a complete record represented by a compound vector. Following the same encoding process, knowledge about the memory objects `Dog` and `Dolphin` is encoded in two lists of symbolic constants in Table 6.3; through a hierarchical vector permutation and bundling, the program generates the representation for memory objects, which are often augmented by a list of augmentation items. It should be noted that each augmentation's

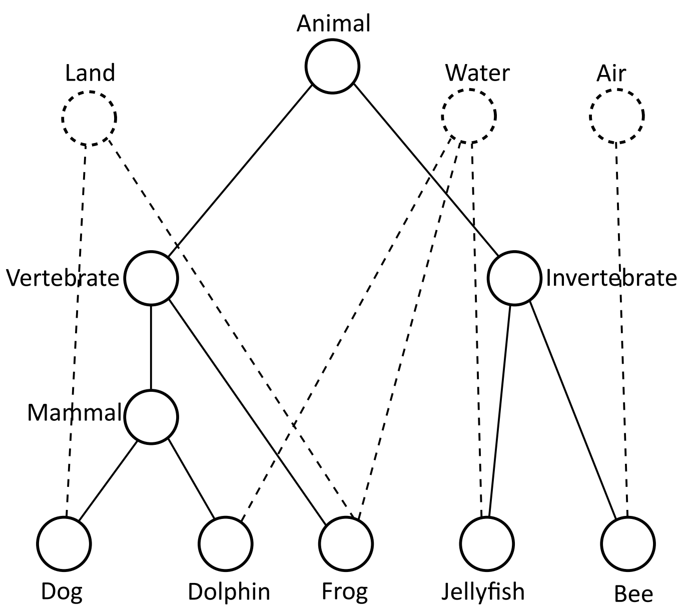


Figure 6.2 A diagram of several animals, their phylogenetic ordering and habitats [142]

internal representations are permuted to differentiate the sequence. At the level of memory objects, the representation is a collection of augmentation items, thus no augmentation is permuted before the majority-sum.

Table 6.3 A sample of memory objects and their representations

Object Name	Augmentation 1	Augmentation 2
Dog	Is Mammal	Habitat Land
Dolphin	Is Mammal	Habitat Water

Hypervector Representations:

Dog	$[H_{\text{Dog}} + [H_{\text{Is}} + p(H_{\text{Mammal}})] + [H_{\text{Habitat}} + p(H_{\text{Land}})]]$
Dolphin	$[H_{\text{Dolphin}} + [H_{\text{Is}} + p(H_{\text{Mammal}})] + [H_{\text{Habitat}} + p(H_{\text{Water}})]]$

After the encoding process, the semantic memory holds a table of hypervectors, each representing a memory object. The query command to the semantic memory includes cues composed of partial information that describes augmentations of an object in semantic memory. The cue goes through the same encoding process as the one describe above, except that the missing field (denoted as '?') is represented by a random hypervector H_{RDM} (could also be an empty hypervector). For instance, the query `Q: ? : Is Mammal : Habitat Water`, interpreted in natural language as: "what is the mammal that lives in water ?", is represented as the hypervector $H_{\text{Cue},1}$ which is:

$$[H_{\text{RDM}} + [H_{\text{Is}} + p(H_{\text{Mammal}})] + [H_{\text{Habitat}} + p(H_{\text{Water}})]]$$

The activation updating activities in a search operation was investigated by firstly passing $H_{\text{Cue},1}$ to the associative memory.

The retrieval process is a two-phase operation. The memory first searches the query hypervector, counts the number of hit events after each search, and retrieves the candidates with the highest activation. Then, the number of hit events are used to update and spread the activation.

Memory objects are assigned zero activation values to start the experiment. For the first query $H_{\text{Cue},1}$, the nearest-match candidate, `Dolphin` was successfully retrieved, with a distance of 8 standard deviations from the mean value (1024-dimensional space with dense binary codes). As shown in Figure 6.3, the activation from the query is spread to all correlated hypervectors. For instance, water-living animals are all recipients of the spreading activation and will be updated accordingly. The second query, `? : Is Invertebrate`, denoted as $H_{\text{Cue},2}$, asks for an animal that belongs to Invertebrate. This less constrained query vector can ambiguously match both `Jellyfish` and `Bee`. However, only the water-living `Jellyfish` is retrieved because it received spreading activation from the previously retrieved water-living animal, which updates it activation to a higher level before the second retrieval. An important design decision is made by assigning one standard deviation tolerance (23 bits) to include all strongly correlated objects in the candidate list, such as `Dog`,`Dolphin`, `Frog` and `Jellyfish` as shown in Figure 6.3. The retrieval process is then biased by the activation values.

6.3 Memory Building Blocks

This section describes a circuit that searches a memory and returns information biased by previous similarity-based activation values. The whole circuit block is partitioned into multiple stages with an associative memory for storing memory objects, counters for storing similarity information, and an activation memory for storing the activation spread from the previous search. The associative memory and the activation memory use memristive storage elements and the remaining blocks uses conventional CMOS logic. The spreading activation scheme requires two search operations as suggested by the example shown earlier in Table 6.1 on page 105. The first search is a CAM search. The second search is the step that biases the result to the most activated object among all match candidates.

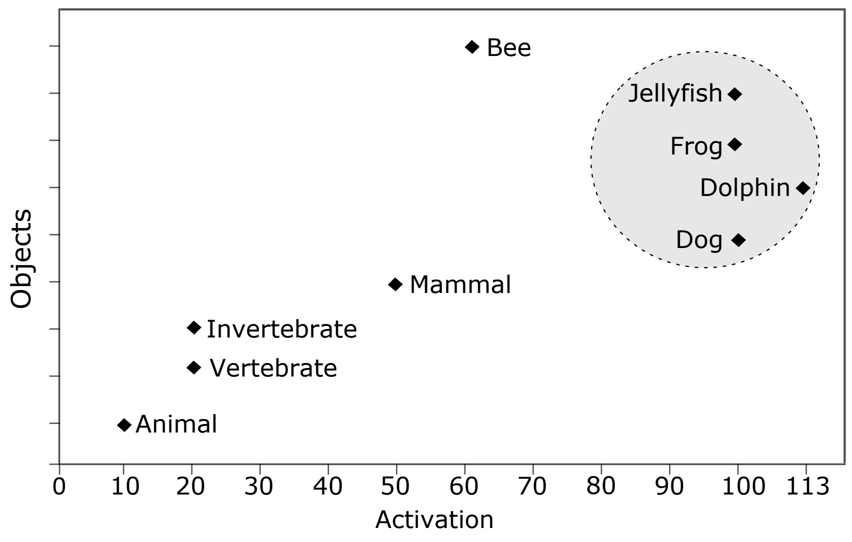


Figure 6.3 An example of object clustering according to activation after the first search

As illustrated in Figure 6.4 a biased knowledge search involves three stages. At the first stage, a bit-serial search operation in the memristive CAM generates similarity values which are then stored in the binary counted buffers after the CAM. At the second stage, these similarity values are examined and a list of candidate memory objects are selected. One of the simplest ways to select these candidates is by assigning a threshold value such that a list of candidate rows with similarity values higher than a threshold are selected for the subsequent bias operations. A discussion on the candidate selection circuit is provided in 6.3.2 on page 116. Candidate rows that survive the threshold operation will activate a subset of the rows in the activation block, which is used to find the highest activation as a single candidate for final output. This block contains a 10-bit wide activation memory and a *winner-take-all* (*WTA*) circuit; the activation memory stores the history of past activation values of every memory object, and when candidates are passed to the activation memory the WTA circuit will find the most activated memory object among all candidates. A *comparator tree* network is one of the possible implementations of a WTA circuit as shown in Figure 6.5 and this chapter proposes alternative WTA implementations in Section 6.3.6 on page 122. After a successful retrieval the similarity values generated in the current search are used to update the activation memory in the activation block so that it biases the result of the next search.

The initial memory search must be handled differently before the activation search because the activation block does not have prior similarity information at the time. Thus the memory disables the 10-bit activation memory of each memory object by initialising all

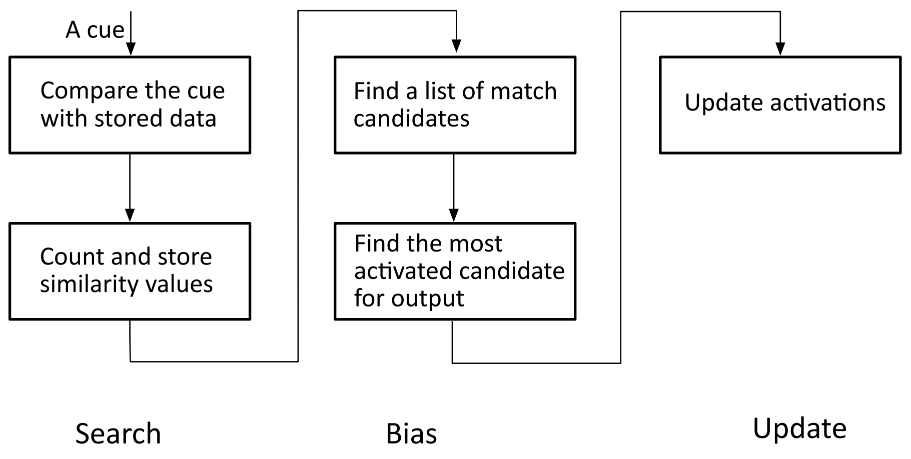

A cue
Compare the cue with stored data
Count and store similarity values
Find a list of match candidates
Find the most activated candidate for output
Update activations
Search
Bias
Update

Figure 6.4 Procedures of retrieval operations in a biased long-term memory

bits to ‘1’s; effectively, the memory finds the nearest-match row only based on the current similarity value in the initial search. The current similarity values are then used to overwrite the initialisation activation values for biasing subsequent searches.

The remainder of this section describes the constituent circuit blocks of this biased retrieval in the long-term memory.

6.3.1 Content-Addressable Memory

CAMs are natural implementation options for hyperdimensional computing, which needs an associative memory to quickly generate association information such as Hamming distance or similarity values for each stored memory object. This section introduces the challenges to building long wordlines in memristive CAMs. The basic idea is to join multiple CAM blocks into a large CAM array for storing long vectors. To serve this purpose, inter-block connection circuits are proposed.

CAM blocks reported in Chapter 4 have a typical memory width of 256 bits, which is far less than the ultra-wide wordline requirement. Despite the potential for minor improvements, practical values of memristor resistive ratio suggest that a standalone CAM block with a memory width of thousands of bits is impractical. An ultra-wide wordline/matchline needs to be divided into multiple segments, in a similar way to the matchline segmentation and pipelining covered in Chapter 4. The communication between the segmented blocks is different. The idea of matchline segmentation is to use a relay chain to reduce unnecessary matchline charging activities at an early stage. The arrangement of CAM blocks in this chap-

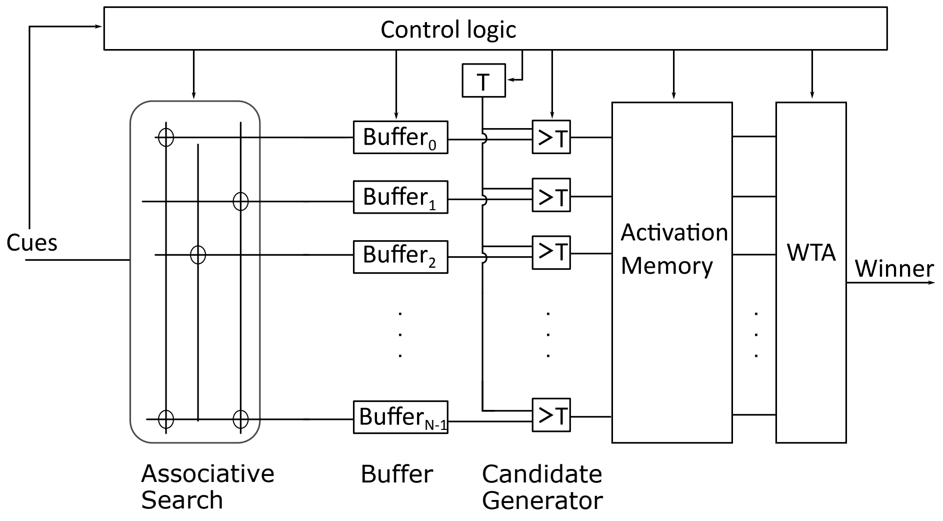


Figure 6.5 The memory circuit blocks

ter is intended for ultra-wide hypervectors where there are no sequential inter-dependencies between blocks; each row in a block sends the binary search output to a global bus which is responsible for aggregating the search results along a memory row. More importantly, the bit-serial search can be performed simultaneously in all blocks with the help of some storage bits and logic circuits.

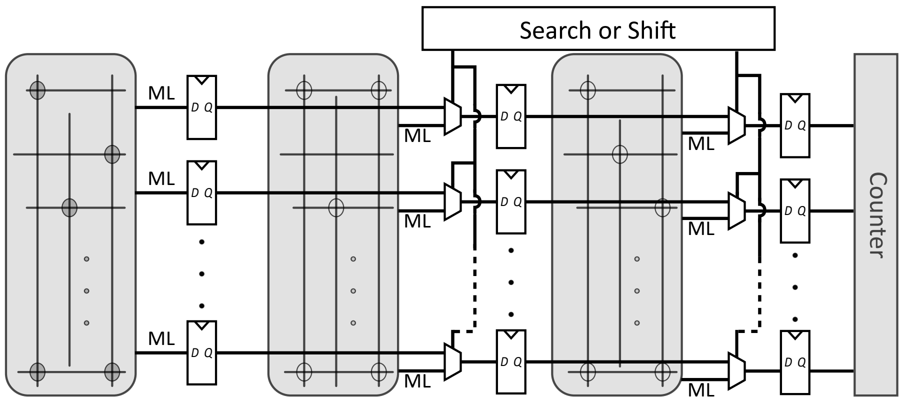


Figure 6.6 A memory with three CAM blocks; shift registers are used to shift out search results from each memory block to the counter

A CMOS circuit is used to accumulate the similarity values from the CAM blocks as illustrated in Figure 6.6. The circuit uses a flip-flop in each CAM block to hold the search result. At the end of each search cycle, search results of individual blocks are stored in parallel and thereafter shifted serially to the counter. For a memory block with a 256-bit wordline, a hypervector of thousands of bits (1000 to 10000) needs 4 to 40 storage bits and hence 4 to 40 cycles of shift operation. Despite the availability of search results at each memory block after a parallel search, the delay of both the shift circuit and the CAM scales linearly with the number of CAM blocks in both circuits because the results are sent bit by bit to the counter.

In a pipelined structure, it is desirable for the CAM blocks to start a new search cycle as soon as the search on the previous bit is finished, so that the subsequent circuits can operate at the same time. Otherwise, CAM blocks have to wait and only start the next bit-serial search when all search results are sent to the counter. Thus, to avoid the sequential operation delay, results from CAM blocks need to be processed simultaneously with the search operation in each block. Such a design requires search results to be stored in parallel immediately after a search, which requires extra storage elements to temporarily hold the information before proceeding to the counter. In this regard, the inter-block connection using shift-registers is well suited because each flip-flop at the local CAM block serves as the result buffer. After passing the search result to the buffer, the internal storage in the matchline sensing circuit is free for storing new search results.

6.3.2 Similarity Buffer

A pipelined structure needs a buffer to store the in-flight similarity value generated at each row of the associative memory. Related work reported in [131, 132, 139] used binary counters to store similarity values and a comparator tree circuit to find the nearest-match row/vector. The binary counter is an intuitive solution that only uses $\log(D)$ bits for counting and storing D-bit hypervectors. This buffer is used to update the activation block as shown in Figure 6.7, where a 10-bit counter is used to store similarity values generated from 1024-bit hypervectors.

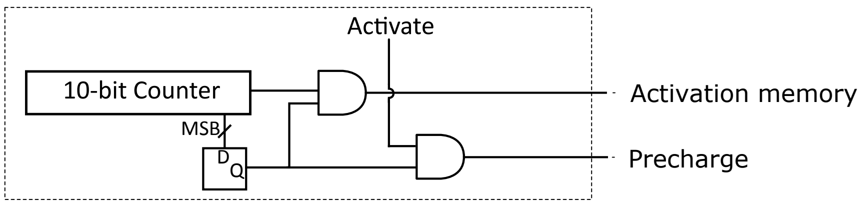


Figure 6.7 The datapath from a 10-bit buffer to activation search

6.3.3 A Candidate Generation Circuit

Unlike the reported applications that require a single winner object after the search, the memory at this stage needs to identify a list of strong-match candidates to proceed to the activation search operations. Thus, it is unnecessary to detect the best match at this point; instead, a circuit that is able to filter out the unrelated data and find a list of strong-correlated candidates is needed.

Such a circuit is a generalisation of the WTA circuit that generates multiple selections (outputs). The design of such a circuit that is able to not only find the row with the greatest similarity value but also keep indexes of k similar candidates is a problem of its own. These are often referred to as k-WTA circuits and serve as building blocks in sorting networks and competitive learning networks in the literature [143–149]. This type of WTA circuit is not the focus of this chapter and this section will illustrate some simple alternatives to demonstrate the proposed spreading activation scheme. The following is a list of options sorted in the order of delay:

1. Finding the k rows with the highest similarity values
2. Finding all rows that are within some threshold of the row with the highest similarity values
3. Finding all candidate rows with similarity greater than some threshold

Finding all candidate rows with similarity greater than some threshold is easier than finding the k rows with the highest similarity values. The former can be done without communicating between the rows and so the delay is independent of the depth of the memory. At this stage, the spreading activation scheme adopts the simple threshold method in this section.

An extremely simple method to find all candidate rows with similarity greater than some threshold similarity values is illustrated in Figure 6.7. The threshold operation simply evaluates the most significant bit of the counter; effectively, any similarity value beyond 512 will proceed to the next stage. To interface with the crossbar based activation memory, which has one-bit serial input, the binary counter needs to have a decoding circuit such as a ring counter to send the $\log(D)$-bit result serially.

6.3.4 Activation Representations

The representation of spreading activation values determines the updating scheme and the interfacing circuits. Similar considerations were made in the activation circuit design in Chapter 5. This section discusses the analog and the digital approaches of storing spreading activation values.

A Discussion of the Analog Circuit

Chapter 5 described a memristor-based activation circuit for maintaining and updating activation values of memory objects; a memristor stores an activation value using its internal conductance, and this value is updated by applying a train of pulses across the device. Based on this circuit, I considered two options for the analog design of spreading activation circuits: (1) for a simplified version of activation scheme where the total value is a simple sum of BLA activation and spreading activation, it is straight-forward to just combine these two type of activation values into one analog memristor representation by reusing the same circuit. (2) alternatively, spreading activations can occupy a separate circuit cell similar to the BLA activation circuit; these two circuits are separately updated, and in the readout operation, they must be serially connected so that the sum of their conductance can be sampled. However, neither of these proposals suits the storage needs of this application. A D-dimensional hypervector needs D distinct resistive states to represent the whole range of similarity values which will require complex tuning and read circuits.

The Digital Circuit

The digital version of the spreading activation memory resembles the structure of a memristor based RAM that records the results from the associative search operation for later processing. The match pattern of each row in the associative memory is serially shifted to the activation array ($\log(D)$ wide), where a column of data is written in parallel as shown in Figure 6.8.

The circuit uses a similar readout operation to that found in RAM. A readout operation in RAM will access a limited number of rows, however, this memory needs to give all the activation information to the subsequent WTA circuit. Thus, using a typical RAM read operation would incur a delay which is dependent on the memory depth. It is desirable to read a column of data simultaneously so that the delay only increases with the memory width, which tends to be a constant number for a specific application. The shift registers with a one-hot bit pattern on the top of the array is for instructing the column-wise read operation.

With a current sensing scheme, sense amplifiers can be arranged at the end of wordlines for sensing the differential of currents along wordlines. Pass-transistors can be used to multiplex the write and read modes (not shown).

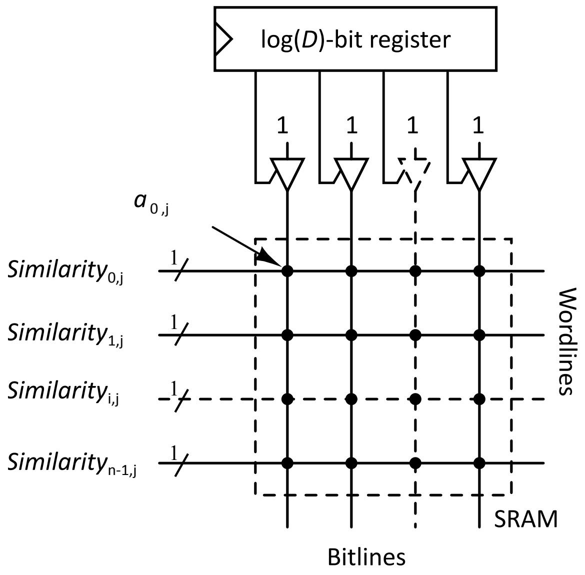


Figure 6.8 A memristor crossbar memory for storing spreading activation

6.3.5 The Winner-Take-All Circuit

The second search needs a WTA circuit to find the highest activation. Unlike the previous threshold circuits that generate multiple candidates, this type of circuit looks for a single winner.

As shown in Figure 6.9, a comparator tree is a straight implementation option. I adopt a CMOS WTA design proposed in [150] where the results propagate from the *most significant bit* (MSB) to the *least significant bit* (LSB). This is a simple means to provide the higher order bits of results to higher nodes of the tree by which the total propagation delay is reduced within the tree. As shown in Figure 6.10, each comparator is implemented using a two input *magnitude comparator* connected with a multiplexor. It accepts quantity inputs like a_i and b_i and will generate outputs with g_i and l_i (for greater than, and less than), which indicate the magnitude relation between a_i and b_i. A winner input is selected for output represented by y_i.

For an associative memory with a depth of N, the number of comparators required is $N-1$. As shown in Figure 6.10, each node in the comparator tree consists of a comparator circuit and a multiplexer selecting the larger input propagating to the higher nodes of the tree.

The overall delay in the worst case, where results must propagate from MSB to the LSB, is the number of stages in a tree of comparator ($\log(N)$) times the worst propagation delay of a $\log(D)$ bit comparator for D-bit hypervectors.

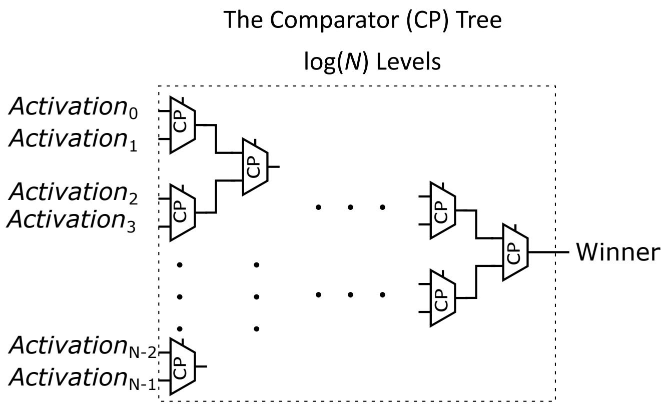


Figure 6.9 The binary counter and comparator tree

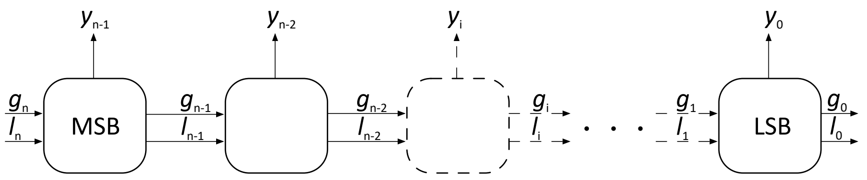


Figure 6.10 The iterative construction of a single comparator propagating results from the MSB to the LSB

6.3.6 Evaluation

In this section the simulation and estimation methodology used in Chapter 4 is used to evaluate the performance of the new spreading activation system.

A memory with 1024-bit hypervectors stored in 256-bit wide CAM needs four such CAM blocks. A bit-serial search cycle takes about 5 ns. The four CAM blocks working concurrently take 256 cycles to perform a search, and 4 cycles are required to shift the results into the counters. This in total introduces a delay of 1.28 µs.

The WTA circuit is another main contributor to the total delay. Consider a memory with depth N and memory width D. In the worst case, each node in the tree has to compare each bit before generating the output. If each bit takes m units of time, the worst case propagation delay of the tree structure is $m\log(D) \cdot \log(N)$. The $m\log(D)$ term is about 1 ns according to the experimental data in [151] for 10-bit width comparators. The $\log(N)$ term decides how many such clock cycles are required to finish the comparison. A memory with one million objects takes about 20 ns to resolve the row with the highest similarity value.

The estimations made here are based on the cell and array layout estimation in Chapter 4 Section 4.2.5 on page 66. A 1024-bit wide CAM needs four memory blocks aligned to construct the matchline. Interconnection logic between memory blocks use CMOS circuits underneath the crossbar structure. Thus the storage of each hypervector occupies an area of $40\,\lambda \times 80000\,\lambda$, which is 100 nm high by 200 µm wide in 45 nm technology; a memristive ternary CAM cell is about 0.02 µm^2.

The state of the art MOS transistor CAM implementation in [112] based on the 28 nm process node realises ternary search using a 16 transistor bit cell with a cell area of 1.62 µm^2. As comparison, a memristive ternary CAM cell is about 0.02 µm^2.

The language recognition task reported in [132] is a suitable case for illustrating the area improvements. It used a similar encoding scheme and memory structure for building a classifier that recognises 21 European languages. The training generates 21 language hypervectors that are stored in an associative memory for similarity search. With a memory width of 10000, this associative memory hosts 210000 bit cells. Memory width (i.e. the hypervector dimensionality) is an application dependent parameter, and it is found that the memory circuit area increases linearly with memory width, which reveals two facts about this memory:

- the amount of area occupied by the peripheral circuits, such as counters, comparators, and encoders, mainly depends on the number of rows in the memory (i.e. memory depth); they remain unchanged when a different memory width is chosen,
- the memory core (i.e. the array of bit cells) often dominates peripheral circuits in terms of area when long hypervectors are used in an associative memory.

Indeed, in [132, 131] the chip area occupied by the associative memory accounts for one third of the total area, which illustrates that hyperdimensional computing is memory-centric. I argue that the crossbar based CAM can significantly reduce this memory area. Specifically, the physical width of the memory can be shrunk from 21.8 mm to 2 mm for the same number of bits (10000) in an ideal implementation. In practical implementations, it is unrealistic to build a complete matchline that connects to thousands of bit cells. Some CMOS storage elements will be required to connect segmented matchlines and this reduces the area efficiency of the memory core in both conventional and crossbar implementations. The vertical direction has limited potential for area reduction since only 21 hypervectors are stored hence 21 memory rows are considered in this case. Nevertheless, the overall area reduction is dramatic and the memory will be able to exploit the saved space by having more sophisticated interconnects and peripheral circuits.

A Discussion of WTA Circuits

The WTA circuit in the proposed activation hardware occupies a significant area. Moreover, its delay is dependent on the memory depth. This section discusses possible improvements of existing WTA circuits and then proposes a counting circuit enhanced for the generation of the candidate list needed in the proposed spreading activation scheme.

One of the common WTA implementations, the comparator tree circuit, has been improved over the years. Since its basic building block resembles an adder circuit, but without the generation of sum bits, many circuit techniques that have been devised for high speed ALU circuits can be applied, such as the look-ahead architecture for improving the propagation delay in [151].

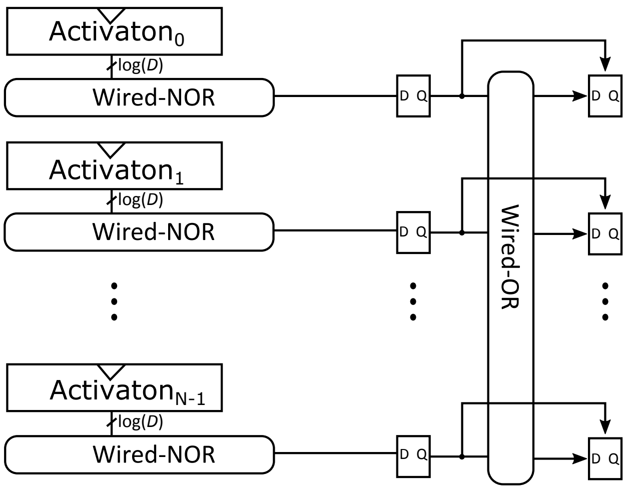


Figure 6.11 Similarity up-counting that finds the maximum

An alternative WTA implementation is the *distance down-counting* circuit used in [55], where, to find the minimal distance the counted distance values are decremented in every cycle until one or more counts reach zero. Recall that Hamming distance is just the complementary value of similarity as defined here. Thus I will discuss the possibility of applying a *similarity up-counting* version of the method here. This up-counting scheme increments the similarity value by one in each cycle until a buffer overflow is detected. The first step of this detection circuit is a wired-NOR gate shown in Figure 6.11, which connects to every bit of the updated similarity values. This arrangement requires rows of similarity values to be accessed at the same time. Thus the array structure in a crossbar cannot be used and CMOS counters are assumed here. The number of clock cycles needed to reach a buffer overflow is equal to the Hamming distance value. In theory, the dimension D decides the upper bound which leads to sequential delays for applications using long hypervectors. In this application, candidates involved in the search of the highest activation row have high similarity values. Thus the average number of cycles required is expected to be less than the mean value of the dimensionality of hypervectors.

6.4 Summary and Conclusion

This chapter described a long-term memory biased by a special spreading activation mechanism that combines ideas from associative memories, memory activation and high-dimensional computing. Circuits developed in previous chapters, such as CAMs and activation circuits, served as the building block of this associative memory and bias mechanism.

Spreading activation and high-dimensional computing are newly introduced concepts in the thesis. The chapter began with a brief review of some basic algorithms that spread activation throughout the semantic network. Existing software implementations of these algorithms are discussed which all rely on databases and centralised computations. The distributed representation with long hypervectors provides a useful alternative, which can be used to group correlated objects or distinguish unrelated objects.

The heart of high-dimensional computing is an associative memory that generates similarity information from which association between different memory objects can be derived. The strength of the association between the searched object and any other objects determines the amount of activation that will be spread to the object. A main difference of this scheme compared to existing algorithms is that the activation spreads to objects that have similar compositional components whereas existing algorithms spread activation according to objects' connections in the semantic network.

The spreading process is done in a memory search, which is a two-phase operation. The memory first searches the query hypervector, counts the number of hit events after each bit-serial search, and detects the candidates within some threshold of the highest activation row. Then, the number of hit events are used to update and spread the activation. A naive recognition task was conducted to illustrate a memory biased by this spreading activation scheme where previous searches were able to bias the following searches and a simple priming effect was thus demonstrated. However, the biasing model using hypervectors is primitive so far and there are a few limitations. First of all, the memory object represented in hypervectors cannot be recalled or reconstructed which limits the applications to recognition tasks. Secondly, it lacks the base activation values and the temporal decay effect. Thirdly, the high-dimensional representation of a memory object with many memory items may lose its correlation with other objects or memory items so that it would not be recognised in a search.

Nevertheless, this activation spreading scheme represents an initiative of a hardware-based memory model that integrates the spreading mechanism with the search operation. Like the parallel search operation, the memory is able to spread activation to all correlated memory objects simultaneously by exploiting available search results. The preliminary results in this chapter suggest the marriage of high-dimensional computing, spreading activation and CAMs provides promising features to implement cognitive long-term memories.

Chapter 7

Summary and Conclusion

The foremost objective of this research has been to design a hardware-based cognitive long-term memory that scales to a large knowledge base but can still be quickly and efficiently searched. This thesis has described a new memory where search and associated cognitive processing are performed using a flat memory structure addressed by content. To support the required operations, a number of new circuit ideas for memory cells and peripheral functions have been described.

The memory structure developed addresses some of the limitations of conventional memory systems when used for cognitive processing. A capable cognitive agent will need a large knowledge base. In conventional technology this would need to use a slow storage class technology such as disks supported by a hierarchy of fast memory caches. The separation between the fast processing logic units and the slow memory system would lead to significant data movement across the memory hierarchy and subsequently more power consumption and operation latency. Cognitive processing often involves access to a large number of memory objects at the same time, which requires massive parallelism at the data level. Von Neumann computers with centralised processing logic, essentially operating in a sequential data fetch, process and store order, are not optimised for the needs of cognitive computations.

The changing nature of computation requires new computing schemes that decouple simple computing resources from centralised processing logic to better fit the characteristics of the data. Many advantages of the new memory, compared to the conventional hierarchical memory, come from closely coupling simple processing units with memory storage elements. Such an arrangement is necessary if the memory is expected to compare a search cue against all memory objects simultaneously, instead of iteratively traversing all possible locations for the best match. The same principle can be applied to cognitive processing: dedicated computation circuits can be distributed to each memory object for maintaining and updating activations in parallel. This in-memory or near-memory processing arrangement eliminates the need to shuffle data between computing units and the memory for memory search or cognitive processing.

The memory achieves its parallel search capabilities using content addressing. Its search power comes from the distributed comparison circuit inside each memory cell which, when passed a binary search cue, is able to compare this against the stored data and generate parity information as the comparison outcome. If search data is broadcast to all rows of the memory table, all the memory cells and all memory rows representing memory entities are searched simultaneously. Thus a memory search now is bounded to a constant time. The price to pay is the complexity of the memory cells which must now include both storage elements and comparison circuits. Traditional CMOS implementations of CAMs often occupy a huge amount of chip area and consume an unacceptable amount of power.

Many alternatives to CMOS CAMs have been proposed in the literature using emerging devices. This thesis shows that memristors are well suited to the design of a more efficient CAM. Memristors have multiple resistive states and hence are natural storage elements. The change of their resistive states also makes them suitable for implementing computing circuits. The thesis shows that the marriage of memristors and CAMs brings the possibilities of new memory cell design. Memristors have a small footprint and can be packed densely on top of CMOS layers. The compactness and resistive characteristics of memristors overcome the area penalty in conventional memory cells by replacing the static RAM storage elements and simplifying the comparison circuit. Another motivation is the power saving potential of memristors which, when used as storage elements, do not need power to maintain their internal states. It should be noted that the passive nature of memristors determines that they still need some CMOS drivers in the periphery circuits.

7.1 Summary

Two new memory cell designs have been proposed to gradually reduce the cell footprint. The basic cell interface of a CMOS CAM cell has been preserved thus control lines such as matchlines and selectlines remain unchanged. The 4T-2R cell described in Chapter 3 builds upon the 2T-2R cell structure often seen in cells using phase-change devices and adds two transistors to address problems of current leakage in read operations. A hierarchical bitline structure is used to collect read results from local memory blocks. Unlike conventional RAMs, the new memory needs to program multiple cells in bulk, and a two-step write operation is required to program a word along the row. With a suitable encoding of memory cells and search data, the memory is able to include a compact comparison circuit in the memory cells which reduces TCAM cell area from conventional twelve transistors to four transistors. Simulations presented in Chapter 3 show that the resistance parameters of memristors determine the selectivity of cells and hence ultimately determine the size of a memory array that can be involved in a read operation. In search, the resistance ratio has a major impact on the number of cells that can be attached to a single matchline. Also, memristors with a low resistance result in lower search latency because of the higher discharging rate in the mismatch cells. The matchline sensing circuit developed in this thesis adopts a self-referenced technique, which provides the timing signal to trigger the sampling process in the sense amplifiers. More importantly, this self-referenced technique has the potential to address the device variation issues associated with memristors.

A more compact CAM is described in Chapter 4 where every two cells of a crossbar resistive RAM are grouped to form a CAM memory cell. By reusing the basic structure of RAMs including bitlines and wordlines, an appropriate voltage bias scheme is able to turn the RAM into a CAM with half the original RAM cell density. The same parallel search capabilities are attained with the assistance of the peripheral circuits from the 4T-2R CAM. Despite the overhead of the CMOS peripheral circuits, estimates show the new CAM achieves over an order of magnitude density improvement compared to the state of the art CMOS counterparts. Estimates of delay and energy consumption were conducted on a CAM storing the entire content of the semantic memory for a word sense disambiguation experiment. The proposed TCAM consumes only a fraction of the energy of its CMOS counterparts but operates at a lower search speed. Nonetheless, estimates show that the retrieval of a modest size memory object with ten augmentations is in the range of microseconds, which is a significant improvement over the millisecond latency of a database implementation. The saved time would allow a cognitive agent to make a decision more quickly, or to use the extra time to make a more carefully considered decision. Many existing CAM techniques are generally applicable to the memristor-based CAMs proposed in this thesis for improvements in energy consumptions. Among them, selective precharge using matchline segmentation and pre-computation have been proved particularly useful for CAMs with interconnected matchlines. The selective precharge scheme ensures mismatched rows are detected and disabled in an earlier stage thus saving energy consumptions. Pre-computation when used in resistive CAMs is more than a low-power design option; it can be used to simplify the cell circuit thus saving the total chip area. The simplified cell circuit also leads to a better matchline sensing margin, which is one of the bottlenecks for CAMs with a larger width.

Hardware support for memory activation and its spreading scheme have been considered. Both digital and analog memristors are capable of storing and updating activation values. Similar to the memristive CAMs, these circuits are CMOS and memristor hybrids. The digital approach is more flexible and more faithful to human cognitive models, but will also need a more expensive winner-take-all circuit to find the most activated object. The analog version achieves much of its efficiency through hard-wiring the updating dynamics of activation. A word sense disambiguation task was conducted to verify the effectiveness of this analog activation. The circuit model is not as reliable as other reproduced results from the literature. However, the compromise in performance is small, and the proposed circuit supports a parallel computation distributed among the long-term memory objects. Both the digital and analog activation scheme represent promising efforts towards in-memory cognitive processing.

Activation needs to be spread to prime the memory objects relevant to the agent's current context. A new spreading activation scheme for a distributed cognitive memory was presented in Chapter 6. The memory combines building blocks from high-dimensional computing, activation and associative memories. The scheme is essentially memory centric and all required operations are carried out in or around associative memories, which are implemented as CAMs. The activation from a search cue is spread to correlated memory objects according to the similarity values generated from CAMs. A series of new circuits have been proposed to aggregate results from segmented CAMs and assist the generation of search candidates. The significance of the hardware-based memory model is that all information required for the spread activation is generated along with the CAM search, thus the memory is able to spread activation to all correlated memory objects simultaneously.

7.2 Conclusion

To conclude, this research proposes a new hardware approach to implement long-term memories using new resistive devices. The new memories are designed to meet the needs of an emerging generation of powerful cognitive agents. They are non-volatile, high capacity, low power, and can be efficiently searched. The new memory also supports contextual priming using spreading activation. The significance of this research is that the use of a flat content-addressable memory provides parallel search capabilities and eliminates the von Neumann bottleneck issues by keeping low level computations within memory. At the same time, distributed hardware blocks supporting cognitive processes such as memory activation scale to large knowledge bases and overcome the limitations of sequential software implementations. At the device level, it is found that the characteristics of practical memristor devices are approaching a point at which they will be useful for large capacity and compact CAMs. Memristor on to off resistance ratio is a critical parameter that determines the search sensing margin and the memory array width. The application of high-dimensional computing not only provides tolerance to memristor variations at the device level but also introduces the possibility of a new hardware accelerated spreading activation scheme.

Looking into the future of computing, it is a fact that dedicated circuit-based accelerators and optimised hardware architectures will better match the needs of emerging applications, especially in the domain of AI. Cognitive memories play an important role in cognitive architectures, and hardware building blocks proposed in this thesis provide an important step towards a more capable cognitive memory that promises to benefit the overall performance of future cognitive agents.

7.3 Future Research

One of the ultimate goals of memory design is to have a brain-like memory. The presented memory in this thesis shows some characteristics observed in biological memories such as parallel search and signs of holding on to a piece of information for the current task. However, the proposed memory is flat and two dimensional while biological memories process information in three dimensions (3D). To achieve the complexity of such memories, a three-dimensional cognitive memory is highly desired to efficiently accommodate more highly connected memory entities, providing the needed infrastructure for brain-inspired cognitive processing. Memristors were born for 3D architectures because of their highly stackable feature. A 3D CAM with either a CMOS/memristor hybrid or a homogeneous memristive structure would be an interesting research topic.

The memory model proposed in Chapter 6 is still primitive and leaves much space for future improvements. At the circuit level, the delay of the WTA circuit is dependent on the memory depth and the footprint of the comparator tree increases with the number of memory rows. However, WTA-like mechanisms that filter out memory objects with weak correlations to the context are indispensable parts of the memory model and other applications mentioned in Chapter 6. A future research topic would be a design of such a circuit that can be gracefully scaled to large memories.

High-dimensional computing provides a rich set of vector algebra tools to process semantic information. Among them, only bundling and permutation are used in this thesis to create random and correlated memory objects. The association between semantically linked objects depends on their common compositional elements instead of their interconnections on the semantic network. One direction of future work might be associating objects according to their connections so that activations are spread via links on the semantic tree network.

Publications

P Wang, BJ Phillips and MJ Liebelt,
"Memristor-Based Activation Circuit for Long-Term Memories in Cognitive Architectures",
Electronics Letters, Volume 51, Number 21, October 2015.

Peng Wang and Braden Phillips,
"Design and Evaluation of Content-Addressable Memory Using Redox Memristive Devices",
16th International Conference on Nanotechnology,
Sendai, Japan, August 2016.

A: Details of The Word Sense Disambiguation Task

The word sense disambiguation experiment used in this thesis was based on that described in [31], for which the source code is available at https://soar.eecs.umich.edu.

Overall, this simulation runs for 20 hours on a Mac (2.3 GHz Intel Core i5, under OS X Version 10.9). With a history size equal to 10, which is the default in Soar [22], an accuracy rate of 67.12 % is obtained.

Two functions in Soar's *smem::act* module (version 9.3.1) were modified for this experiment. The code is provided in Listing 1 and Listing 2.

```

#include "Memristor.h"

inline double smem_lti_calc_base( agent *my_agent, smem_lti_id lti, ↩
    int64_t time_now, uint64_t n = 0, uint64_t access_1 = 0 ){
     double sum = 0.0;

     if ( n == 0 ){
          my_agent->smem_stmts->lti_access_get->bind_int( 1, lti );
          my_agent->smem_stmts->lti_access_get->execute();

          n = my_agent->smem_stmts->lti_access_get->column_int( 0 );
          access_1 = my_agent->smem_stmts->lti_access_get->column_int( 2 ↩
              );

          my_agent->smem_stmts->lti_access_get->reinitialize();
     }

     // get all history
     my_agent->smem_stmts->history_get->bind_int( 1, lti );
    my_agent->smem_stmts->history_get->execute();

    {

        uint64_t available_history = n;

        //initialise memristor
        Memristor activation;
        float act_voltage=1.8;
        float deact_voltage=-1;
        double act_pulse_width=1.5e-3;
        double deact_pulse_width=100e-6;
        uint64_t idx;
        uint64_t period;
        for ( uint64_t i=0; i<available_history; i++ ){
            activation.update(act_voltage,act_pulse_width);
            if(i == available_history-1){
                period = time_now - my_agent->smem_stmts->history_get->↩
                    column_int( available_history-1 );

            }
            else {
                period = (my_agent->smem_stmts->history_get->column_int↩
                    (i+1) - my_agent->smem_stmts->history_get->↩
                    column_int( i ) );
            }

            for (uint64_t j=0; j < period; j++){
                activation.update(deact_voltage,deact_pulse_width);
            }
        }
    }

    my_agent->smem_stmts->history_get->reinitialize();
    return sum = 1/activation.getResistance();
}
```

Listing 1 Memristor-Based Activation Implemented Using the Soar Core Code

```

/*
 * Memristor.h
 *
 */

#ifndef MEMRISTOR_H
#define MEMRISTOR_H
class Memristor{

    public:

        Memristor();
        double getState() const;
        double getResistance() const;
        void update(float Vs, double time);


    private:

        int sgn(double var);                    //internal function
        double sgn2(double var);                //internal function
        double trunc();                         //truncate function
        double stateRate();
        void stateCalc(double time);           //calculate the state
        void resCalc(double time);             //calculate the resistance

        float Vs;
        double state;
        double resistance;
        float alpha;
        float beta;
        float gamma;
        int delta;
        int wmax;
        int wmin;
        float lambda;
        float eta1;
        float eta2;
};
#endif

/*
 *
 * Implementation of Memristor
 *
 * by peng wang Oct 17th, 2014
 */
```

```
#include "Memristor.h"
#include <math.h>

Memristor::Memristor(){

        alpha=0.5e-6;                           //Initialze all parameters
        beta=0.5;
        gamma=4e-6;
        delta=2;
        wmax=1;
        wmin=0;
        lambda=4.5;
        eta1=0.004;
        eta2=4;
                                                //Initialize state and ↩
                                                   resistance
        state=0;
        resistance=5e6;
}

int Memristor::sgn(double var){

            if(var>0) return 1;
            else if(var<0) return -1;
            else return 0;
}

double Memristor::sgn2(double var){

            return (sgn(var)+1)/2;
}

double Memristor::trunc(){

    double temp_1=sgn2(Vs)*sgn2(wmax-state);
    double temp_2=sgn2(-Vs)*sgn2(state-wmin);
    return temp_1+temp_2;
}

double Memristor::stateRate(){                      // the state change ↩
    rate

    double temp_1=trunc();
    double temp_2=sinh(eta2*Vs);
    return lambda*eta1*temp_1*temp_2;
}

void Memristor::stateCalc(double time){          // calculate the state

    state+=stateRate()*time;
}

void Memristor::resCalc(double time){            // calculate the resistance

    stateCalc(time);
    double current =(1-state)*alpha*(1-exp(-beta*Vs))+state*gamma*sinh(delta*Vs);

    resistance=Vs/current;
}

double Memristor::getState() const{

    return state;
}

double Memristor::getResistance() const{

    return resistance;
}

void Memristor::update(float V, double time){

    Vs=V;
    resCalc(time);
}
```

Listing 2 A memristor behaviour model in C++

B: An Introduction to Hyperdimensional Computing

Distributed representation with high-dimensional random vectors has been used in [127, 53] to realise cognitive semantic processing. Researchers have demonstrated that high-dimensional representation can offer robustness and randomness that can be a blessing for several types of semantic processing such as understanding things by analogy, learning to infer. Interestingly, because the high-dimensional representation distributes information over all the components, the integrity of the information decrease in relation to the number of error bits irrespective of their positions. This is often referred to as holographic or holistic representation. An associative memory is a core component in such systems to clean up the noisy item by looking up the real item in the memory store, which is normally a content-addressable memory. Simple semantic processing operations have been demonstrated in [131] using transistor-based CAMs.

All the above characteristics appear to offer a good opportunity for realising long-term semantic memory with memristors because:

- Hypervectors are ultra-wide words with many redundant bits, for which emerging resistive devices have a benefit over transistors because of their small footprint;
- CAM is a core component of high-dimensional computing, and improvements with memristive CAM have been demonstrated; and
- Exclusive-OR multiplication between hypervectors is the key arithmetic operation for hyperdimensional computing and semantic processing. It is also particularly costly in terms of hardware; however a parallel exclusive-OR operation can be embedded within a CAM and this holds the promise to speed up this critical computation.

Background

For example, consider the set of 10 000 dimensional binary vectors. The representational space consists of 2^{10000} patterns called points of the hyper-dimensional space (hyperspace). Each new hypervector (points) is drawn randomly from the hyperspace, and because of its high dimensionality, the new hypervector will be nearly orthogonal to all other hypervectors that are already in use. Denote the number of 1s in a hypervector as k, its *density*. The probability of selecting a random vector of length d with density k, where the probability of a 1 in any position is p, is described by the binomial distribution:

$$P_{\text{binomial}}(k, d, p) = \binom{d}{k} p^k (1-p)^{d-k}$$

An illustration of the binomial probability is shown in Figure 1. The ease of creating new approximately orthogonal vectors is a major reason for using the hyper-dimensional representation.

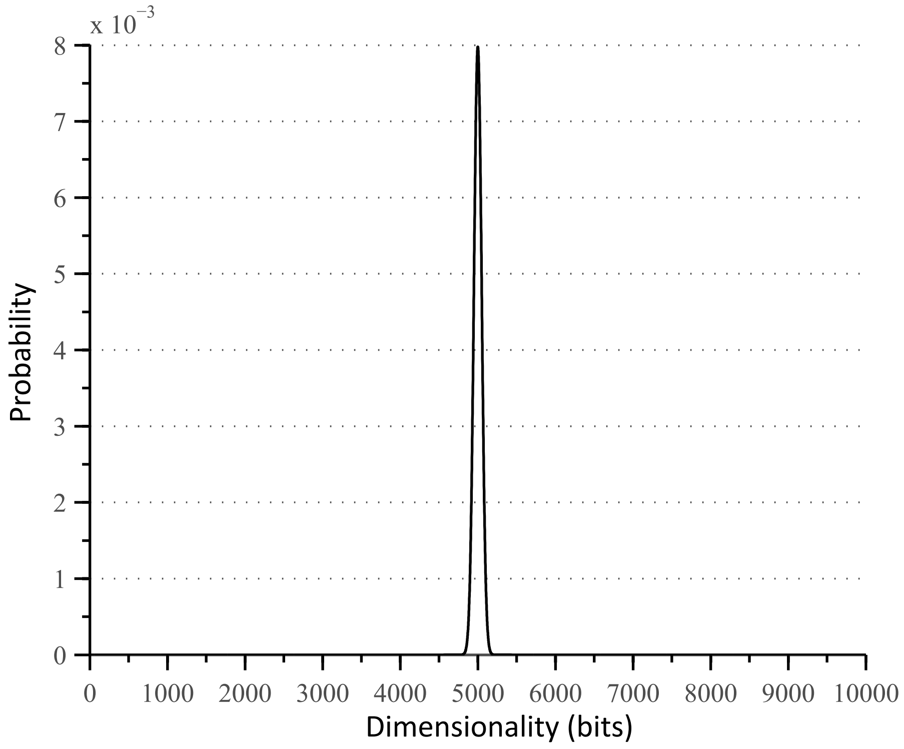


Figure 1 The binomial distribution PDF for k = 10 000 and p = 0.5

The basic entities in hyper-dimensional representation are represented by random vectors stored in the item memory. This is an associative memory that retrieves the memory item most closely matching a noisy search term. The hardware implementation of this memory can be content-addressable memory (CAM) with nearest-neighbour search capabilities.

When creating a hypervector, we need to consider that a hypervector needs to show similarity to related hypervectors, yet it needs to be distinctive from unrelated hypervectors so that it is able to be successfully retrieved after a search operation. Vector addition is used for correlating related hypervectors and exclusive-OR (XOR) for mapping (binding) a hypervector to an uncorrelated pattern.

For example, consider representing memory elements in Soar using these operations. Memory elements have three fields: ID, Attributes and Values. For each of these fields we choose a random vector: X for ID field, Y for Attributes, and Z for Values. The same values X, Y and Z are used to represent all elements in the memory.

Now consider a particular memory element. The contents of its ID field can be represented by a random hypervector A. Similarly the contents of its Attributes and Values fields can be represented by random hypervectors B and C.

The field ID with its content A is represented by $X \oplus A$. Overall the memory element can be represented as:

$$H = X \oplus A + Y \oplus B + Z \oplus C$$

The vector H is referred to as a holistic representation of the memory element because it is made of the bit patterns of its fields and their contents. Being a sum, H is similar to each of the three pairs; the pairs being XOR products, hide the identity of their elements so that H is dissimilar to each of X, Y, Z, A, B, C. Also, information in memory element spans the entire vector in holistic representation.

However, the contents of the fields are not lost and can be recovered by unbinding. For example, to find A, the ID field of the hypervector H with B and C stored in the Attributes and Values fields, we consider XOR(H,X).

$$\begin{aligned} X \oplus H &= X \oplus (X \oplus A + Y \oplus B + Z \oplus C) \\ &= X \oplus X \oplus A + X \oplus Y \oplus B + X \oplus Z \oplus C \\ &= A + R1 + R2 \end{aligned}$$

where A is stored in memory, and $R1$ and $R2$ are random noise and dissimilar to any items in the memory. Recall the equation for calculating the expected Hamming distance between any compound vector and its coponent vectors. In this case, the normalised distance between $A+R1+R2$ and A is 0.25, which is significantly close in the hyperspace. The resultant compound vector $A+R1+R2$ thus can be used as a search query for fetching the real value of A from the CAM where rows of hypervectors are stored.

An Example: What is the Dollar of U.K.?

If we have memory elements with two fields: *country (X)* and *currency (Y)*, Australia (A) with dollar (D) as currency will be represented as: $H_A = X \oplus A + Y \oplus D$. UK ($U$) with Pound ($P$) as monetary unit is : $H_U = X \oplus U + Y \oplus P$.

An associative search: what is the currency of Australia is corresponding to unbinding Y and H_A, which gives a result that recovers D with some random noise $R1$.

$$\begin{aligned} Y \oplus H_A &= Y \oplus (X \oplus A + Y \oplus D) \\ &= R1 + D \end{aligned}$$

What if we ask a silly question: what is the Dollar of U.K.? The result is some random vector and the correct result would not be found.

$$\begin{aligned} D \oplus H_U &= D \oplus (X \oplus U + Y \oplus P) \\ &= R2 + R3 \end{aligned}$$

The language computers use is very literal; Human language tend to be more figurative, and Humans with some basic knowledge of Australian and UK money system can answer the question: Dollar of UK by saying: it is actually Pound.

However, if we provide the holistic system some context, the system would be able the give the correct answer without explicitly recovering the field Y (Monetary) of H_U (UK). The semantic searching is performed by: $(D \oplus H_A) \oplus H_U$, which can be interpreted as: what in UK corresponds to the dollar in Australia?

Representing Substitution

The brain has the ability to understand things by analogy. In computing terms, it means *substituting* a particular value in a field of a record (memory element). For a holistic record, substitution operation on a specific field can be done with XOR multiplication.

Say value A is assigned to field X, represented by the $X \oplus A$ binding. To substitute A with B, we need to restore the field by unbinding with A: $(X \oplus A) \oplus A = X$. Then D is assigned to X: $X \oplus D$. The binding can be written as $((X \oplus A) \oplus A) \oplus D$, which equals to $(X \oplus A) \oplus (A \oplus D)$. Thus substituting value A with value D can be achieved by binding $(A \oplus D)$ to the previously filler-field bound pair $X \oplus A$.

References

[1] Tom Simonite. Apple's 'Neural Engine' Infuses the Iphone with AI Smarts, 2017. URL https://www.wired.com/story/apples-neural-engine-infuses-the-iphone-with-ai-smarts/.

[2] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017.

[3] John R Anderson. Act: A simple theory of complex cognition. *American Psychologist*, 51(4):355, 1996.

[4] John E Laird. *The Soar Cognitive Architecture*. The MIT Press, 2012.

[5] Scott Douglass and Christopher W Myers. Concurrent knowledge activation calculation in large declarative memories. Technical report, AIR FORCE RESEARCH LAB MESA AZ HUMAN EFFECTIVENESS DIRECTORATE, 2010.

[6] John E Laird, Nate Derbinsky, and Jonathan Voigt. Performance evaluation of declarative memory systems in soar. *Ann Arbor*, 1001:48109–2121, 2011.

[7] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, March 1995. ISSN 0163-5964.

[8] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[9] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A case for intelligent ram. *IEEE micro*, 17(2):34–44, 1997.

[10] R. Nair. Evolution of memory architecture. *Proceedings of the IEEE*, 103(8):1331–1345, Aug 2015.

[11] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. *SIGPLAN Not.*, 47(4):37–48, March 2012. ISSN 0362-1340.

[12] Harold S Stone. A logic-in-memory computer. *IEEE Transactions on Computers*, 100 (1):73–78, 1970.

[13] J. Carter, W. Hsieh, L. Stoller, M. Swanson, Lixin Zhang, E. Brunvand, A. Davis, Chen-Chi Kuo, R. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama. Impulse: building a smarter memory controller. In *Proceedings Fifth International Symposium on High-Performance Computer Architecture*, pages 70–79, Jan 1999.

[14] Joshua Friedrich, Hung Le, William Starke, Jeff Stuechli, Balaram Sinharoy, Eric J Fluhr, Daniel Dreps, Victor Zyuban, Gregory Still, Christopher Gonzalez, et al. The power8 tm processor: Designed for big data, analytics, and cloud environments. In *IC Design & Technology (ICICDT), 2014 IEEE International Conference on*, pages 1–4. IEEE, 2014.

[15] Maya Gokhale, Bill Holmes, and Ken Iobst. Processing in memory: The terasys massively parallel pim array. *Computer*, 28(4):23–31, 1995.

[16] Christoforos E Kozyrakis, Stylianos Perissakis, David Patterson, Thomas Anderson, Krste Asanovic, Neal Cardwell, Richard Fromm, Jason Golbus, Benjamin Gribstad, Kimberly Keeton, et al. Scalable processors in the billion-transistor era: Iram. *Computer*, 30(9):75–78, 1997.

[17] Joe Jeddeloh and Brent Keeth. Hybrid memory cube new dram architecture increases density and performance. In *VLSI Technology (VLSIT), 2012 Symposium on*, pages 87–88. IEEE, 2012.

[18] Peter M Kogge. Execube-a new architecture for scaleable mpps. In *Parallel Processing, 1994. Vol. 1. ICPP 1994. International Conference on*, volume 1, pages 77–84. IEEE, 1994.

[19] Donald G Bailey. *Design for embedded image processing on FPGAs*. John Wiley & Sons, 2011.

[20] W Daniel Hillis. *The connection machine*. MIT press, 1989.

[21] Scott Douglass, Jerry Ball, and Stuart Rodgers. Large declarative memories in act-r. Technical report, AIR FORCE RESEARCH LAB MESA AZ HUMAN EFFECTIVENESS DIRECTORATE, 2009.

[22] Nate Derbinsky, John E. Laird, and Ann Arbor Mi. Towards efficiently supporting large symbolic declarative memories. In *In Proceedings of the 10th International Conference on Cognitive Modeling (ICCM*, pages 49–54, 2010.

[23] Dario Salvucci. Endowing a cognitive architecture with world knowledge. In *Proceedings of the Cognitive Science Society*, volume 36, 2014.

[24] JE Laird and CB Congdon. The soar user's manual version 9.4. 0. *Computer Science and Engineering Department, University of Michigan*, 2014.

[25] J. J. Yang and R. S. Williams. Memristive devices in computing system: Promises and challenges. *ACM J. Emerg. Technol. Comput. Syst*, 9(2), 2013.

[26] Peng Wang and Braden Phillips. Design and evaluation of content addressable memory using redox memristive devices. In *Nanotechnology (IEEE-NANO), 2016 IEEE 16th International Conference on*, pages 533–536. IEEE, 2016.

[27] P Wang, BJ Phillips, and MJ Liebelt. Memristor-based activation circuit for long-term memories in cognitive architectures. *Electronics Letters*, 51(21):1639–1641, 2015.

[28] Matthew A Kelly, K Kwock, and Robert L West. Holographic declarative memory and the fan effect: A test case for a new memory module for act-r. In *Proceedings for the 2015 International Conference on Cognitive Modeling (ICCM)*, pages 148–153, 2015.

[29] Pat Langley. The cognitive systems paradigm. *Advances in Cognitive Systems*, 1:3–13, 2012.

[30] Kristinn Thórisson and Helgi Helgasson. Cognitive architectures and autonomy: A comparative review. *Journal of Artificial General Intelligence*, 3(2):1–30, 2012.

[31] Nate Derbinsky and John E Laird. A functional analysis of historical memory retrieval bias in the word sense disambiguation task. *Ann Arbor*, 1001:48109–2121, 2011.

[32] Nate Derbinsky and John E Laird. Computationally efficient forgetting via base-level activation. In *Proceedings of the 11th International Conference on Cognitive Modeling*, pages 109–110, 2012.

[33] A Petrov. Computationally efficient approximation of the base-level learning equation in ACT-R. In *Proceedings of the 7th International Conference on Cognitive Modeling*, Trieste, Italy, 2006.

[34] Nate Derbinsky and John E Laird. A preliminary functional analysis of memory in the word sense disambiguation task. In *Proc. 2nd Symposium on Human Memory for Artificial Agents*, 2011.

[35] Yang Chen, Milenko Petrovic, and Micah Clark. Semmemdb: In-database knowledge activation. In *Flairs conference*, 2014.

[36] Peter A Boncz, Martin L Kersten, and Stefan Manegold. Breaking the memory wall in monetdb. *Communications of the ACM*, 51(12):77–85, 2008.

[37] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 365–376. ACM, 2011.

[38] Kostas Pagiamtzis and Ali Sheikholeslami. Content-addressable memory (cam) circuits and architectures: A tutorial and survey. *Solid-State Circuits, IEEE Journal of*, 41(3):712–727, 2006.

[39] Kamran Eshraghian, Kyoung-Rok Cho, Omid Kavehei, Soon-Ku Kang, Derek Abbott, and Sung-Mo Steve Kang. Memristor MOS content addressable memory (MCAM): Hybrid architecture for future high performance search engines. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(8):1407–1417, 2011.

[40] Engin Ipek, Qing Guo, Xiaochen Guo, and Yuxin Bai. Resistive memories in associative computing. In *Emerging Memory Technologies*, pages 201–229. Springer, 2014.

[41] Anthony J McAuley and Paul Francis. Fast routing table lookup using cams. In *INFOCOM'93. Proceedings. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE*, pages 1382–1391. IEEE, 1993.

[42] Maya Gokhale, Dave Dubois, Andy Dubois, Mike Boorman, Steve Poole, and Vic Hogsett. Granidt: Towards gigabit rate network intrusion detection technology. *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, pages 47–61, 2002.

[43] Subbarao Palacharla, Norman P Jouppi, and James E Smith. *Complexity-effective superscalar processors*, volume 25. ACM, 1997.

[44] Alper Buyuktosunoglu, Stanley Schuster, David Brooks, Pradip Bose, Peter Cook, and David Albonesi. An adaptive issue queue for reduced power at high performance. In *International Workshop on Power-Aware Computer Systems*, pages 25–39. Springer, 2000.

[45] Thomas W Barr, Alan L Cox, and Scott Rixner. Translation caching: skip, don't walk (the page table). In *ACM SIGARCH Computer Architecture News*, volume 38, pages 48–59. ACM, 2010.

[46] Nagender Bandi, Sam Schnieder, Divyakant Agrawal, and Amr El Abbadi. Hardware acceleration of database operations using content-addressable memories. In *DaMoN*, 2005.

[47] William Webber, Alistair Moffat, and Justin Zobel. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems (TOIS)*, 28(4):20, 2010.

[48] A. Rahimi, L. Benini, and R. K. Gupta. Temporal memoization for energy-efficient timing error recovery in gpgpus. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014. doi: 10.7873/DATE.2014.113.

[49] A. Rahimi, A. Ghofrani, M. A. Lastras-Montano, K. Cheng, L. Benini, and R. K. Gupta. Energy-efficient gpgpu architectures via collaborative compilation and memristive memory-based computing. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014.

[50] Mahmoud Meribout, Takeshi Ogura, and Mamoru Nakanishi. On using the cam concept for parametric curve extraction. *IEEE Transactions on Image Processing*, 9 (12):2126–2130, 2000.

[51] Mamoru Nakanishi and Takeshi Ogura. Real-time cam-based hough transform algorithm and its performance evaluation. *Machine Vision and Applications*, 12(2):59–68, 2000.

[52] Somnath Paul and Swarup Bhunia. Reconfigurable computing using content addressable memory for improved performance and resource usage. In *Proceedings of the 45th annual Design Automation Conference*, pages 786–791. ACM, 2008.

[53] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159, 2009.

[54] Teuvo Kohonen. The self-organizing map. *Neurocomputing*, 21(1):1–6, 1998.

[55] S. Rüping, M. Porrmann, and U. Rückert. Som accelerator system. *Neurocomputing*, 21(1):31–50, 1998.

[56] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[57] J. Lee, Jungho Shin, Daeseok Lee, W. Lee, S. Jung, M. Jo, J. Park, K. P. Biju, S. Kim, S. Park, and H. Hwang. Diode-less nano-scale zrox/hfox rram device with excellent switching uniformity and reliability for high-density cross-point memory applications. In *2010 International Electron Devices Meeting*, pages 19.5.1–19.5.4, Dec 2010.

[58] Antonio C Torrezan, John Paul Strachan, Gilberto Medeiros-Ribeiro, and R Stanley Williams. Sub-nanosecond switching of a tantalum oxide memristor. *Nanotechnology*, 22(48):485203, 2011.

[59] Dalibor Biolek, Zdeněk Biolek, Viera Biolková, and Zdeněk Kolka. Some fingerprints of ideal memristors. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 201–204. IEEE, 2013.

[60] ITRS. International technology roadmap for semiconductors 2013 edition emerging research devices. Roadmap report, ITRS, 2011.

[61] Qing Guo, Xiaochen Guo, Yuxin Bai, and Engin Ipek. A resistive tcam accelerator for data-intensive computing. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 339–350. ACM, 2011.

[62] ITRS. International technology roadmap for semiconductors 2013 edition emerging research devices. Roadmap report, ITRS, 2013.

[63] J Joshua Yang, Dmitri B Strukov, and Duncan R Stewart. Memristive devices for computing. *Nature nanotechnology*, 8(1):13–24, 2013.

[64] Leon Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519, 1971.

[65] S. J. Wolf Y. N. Joglekar. The elusive memristor: properties of basic electrical circuits. *European Journal of Physics*, 30(4):661–675, 2009.

[66] Zdeněk Biolek, Dalibor Biolek, and Viera Biolková. Spice model of memristor with nonlinear dopant drift. *Radio engineering*, 18(2):210 – 214, 2009. ISSN 12102512.

[67] Ting Chang, Sung-Hyun Jo, Kuk-Hwan Kim, Patrick Sheridan, Siddharth Gaba, and Wei Lu. Synaptic behaviors and modeling of a metal oxide memristive device. *Applied physics A*, 102(4):857–863, 2011.

[68] Robinson E. Pino, James W. Bohl, Nathan McDonald, Bryant Wysocki, Peter Rozwood, Kristy A. Campbell, Antonio Oblea, and Achyut Timilsina. Compact method for modeling and simulation of memristor devices: Ion conductor chalcogenide-based memristor devices. In *Proceedings of the 2010 IEEE/ACM International Symposium on Nanoscale Architectures*, NANOARCH '10, pages 1–4, Piscataway, NJ, USA, 2010. IEEE Press.

[69] H. Abdalla and M. D. Pickett. Spice modeling of memristors. In *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pages 1832–1835, May 2011.

[70] Chris Yakopcic, Tarek M Taha, Guru Subramanyam, and Robinson E Pino. Generalized memristive device spice model and its application in circuit design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(8): 1201–1214, 2013.

[71] Christophe J Chevallier, Chang Hua Siau, Seow Fong Lim, Sri Rama Namala, Misako Matsuoka, Bruce L Bateman, and Darrell Rinerson. A 0.13 μm 64mb multi-layered conductive metal-oxide memory. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 260–261. IEEE, 2010.

[72] Eike Linn, Roland Rosezin, Carsten Kugeler, and Rainer Waser. Complementary resistive switches for passive nanocrossbar memories. *Nat Mater*, 9(5):403–406, 2010.

[73] Shyh-Shyuan Sheu, Meng-Fan Chang, Ku-Feng Lin, Che-Wei Wu, Yu-Sheng Chen, Pi-Feng Chiu, Chia-Chen Kuo, Yih-Shan Yang, Pei-Chia Chiang, Wen-Pin Lin, et al. A 4mb embedded slc resistive-ram macro with 7.2 ns read-write random-access time and 160ns mlc-access capability. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 200–202. IEEE, 2011.

[74] Y. B. Kim, S. R. Lee, D. Lee, C. B. Lee, M. Chang, J. H. Hur, M. J. Lee, G. S. Park, C. J. Kim, U. I. Chung, I. K. Yoo, and K. Kim. Bi-layered rram with unlimited endurance and extremely uniform switching. In *2011 Symposium on VLSI Technology - Digest of Technical Papers*, pages 52–53, June 2011.

[75] Kuk-Hwan Kim, Siddharth Gaba, Dana Wheeler, Jose M. Cruz-Albrecht, Tahir Hussain, Narayan Srinivasa, and Wei Lu. A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. *Nano Letters*, 12 (1):389–395, 2012.

[76] Wataru Otsuka, Koji Miyata, Makoto Kitagawa, Keiichi Tsutsui, Tomohito Tsushima, Hiroshi Yoshihara, Tomohiro Namise, Yasuhiro Terao, and Kentaro Ogata. A 4mb conductive-bridge resistive memory with 2.3 gb/s read-throughput and 216mb/s program-throughput. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 210–211. IEEE, 2011.

[77] Hyung Dong Lee, SG Kim, K Cho, H Hwang, H Choi, J Lee, SH Lee, HJ Lee, J Suh, S-O Chung, et al. Integration of 4f2 selector-less crossbar array 2mb reram based on transition metal oxides for high density memory applications. In *VLSI Technology (VLSIT), 2012 Symposium on*, pages 151–152. IEEE, 2012.

[78] Meng-Fan Chang, Che-Wei Wu, Chia-Cheng Kuo, Shin-Jang Shen, Ku-Feng Lin, Shu-Meng Yang, Ya-Chin King, Chorng-Jung Lin, and Yu-Der Chih. A 0.5 v 4mb logic-process compatible embedded resistive ram (reram) in 65nm cmos using low-voltage current-mode sensing scheme with 45ns random read time. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 434–436. IEEE, 2012.

[79] Akifumi Kawahara, Ryotaro Azuma, Yuuichirou Ikeda, Ken Kawai, Yoshikazu Katoh, Yukio Hayakawa, Kiyotaka Tsuji, Shinichi Yoneda, Atsushi Himeno, Kazuhiko Shimakawa, et al. An 8 mb multi-layered cross-point reram macro with 443 mb/s write throughput. *IEEE Journal of Solid-State Circuits*, 48(1):178–185, 2013.

[80] R. Fackenthal, M. Kitagawa, W. Otsuka, K. Prall, D. Mills, K. Tsutsui, J. Javanifard, K. Tedrow, T. Tsushima, Y. Shibahara, and G. Hush. 19.7 a 16gb reram with 200mb/s write and 1gb/s read in 27nm technology. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 338–339, Feb 2014.

[81] Tz-yi Liu, Tian Hong Yan, Roy Scheuerlein, Yingchang Chen, Jeffrey KoonYee Lee, Gopinath Balakrishnan, Gordon Yee, Henry Zhang, Alex Yap, Jingwen Ouyang, et al. A 130.7-mm^2 2-layer 32-gb reram memory device in 24-nm technology. *IEEE Journal of Solid-State Circuits*, 49(1):140–153, 2014.

[82] Sung Hyun Jo, Tanmay Kumar, Sundar Narayanan, and Hagop Nazarian. Cross-point resistive RAM based on field-assisted superlinear threshold selector. *IEEE Transactions on Electron Devices*, 62(11):3477–3481, 2015.

[83] J. Joshua Yang, M.-X. Zhang, Matthew D. Pickett, Feng Miao, John Paul Strachan, Wen-Di Li, Wei Yi, Douglas A. A. Ohlberg, Byung Joon Choi, Wei Wu, Janice H. Nickel, Gilberto Medeiros-Ribeiro, and R. Stanley Williams. Engineering nonlinearity into memristors for passive crossbar applications. *Applied Physics Letters*, 100(11): 113501, 2012.

[84] Greg S Snider. Spike-timing-dependent learning in memristive nanodevices. In *Nanoscale Architectures, 2008. NANOARCH 2008. IEEE International Symposium on*, pages 85–92. IEEE, 2008.

[85] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B. Bhadviya, Pinaki Mazumder, and Wei Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Letters*, 0(proofing):null, 2010.

[86] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert M Shelby, Irem Boybat, Carmelo Nolfo, Severin Sidler, Massimo Giordano, Martina Bodini, Nathan CP Farinha, et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558(7708):60, 2018.

[87] Julien Borghetti, Gregory S Snider, Philip J Kuekes, J Joshua Yang, Duncan R Stewart, and R Stanley Williams. 'memristive'switches enable 'stateful'logic operations via material implication. *Nature*, 464(7290):873–876, 2010.

[88] Eero Lehtonen, Jussi H Poikonen, and Mika Laiho. Applications and limitations of memristive implication logic. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on*, pages 1–6. IEEE, 2012.

[89] Kyosun Kim, Sangho Shin, and Sung-Mo Kang. Stateful logic pipeline architecture. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 2497–2500. IEEE, 2011.

[90] Pierre-Emmanuel Gaillardon, Luca Amarú, Anne Siemon, Eike Linn, Rainer Waser, Anupam Chattopadhyay, and Giovanni De Micheli. The programmable logic-in-memory (plim) computer. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, DATE '16, pages 427–432, San Jose, CA, USA, 2016. EDA Consortium.

[91] Ashish Goel and Pankaj Gupta. Small subset queries and bloom filters using ternary associative memories, with applications. *ACM SIGMETRICS Performance Evaluation Review*, 38(1):143–154, 2010.

[92] Takahiro Hanyu, Naoki Kanagawa, and Michitaka Kameyama. Non-volatile one-transistor-cell multiple-valued cam with a digit-parallel-access scheme and its applications. *Computers & electrical engineering*, 23(6):407–414, 1997.

[93] Igor Arsovski, Trevis Chandler, and Ali Sheikholeslami. A ternary content-addressable memory (tcam) based on 4t static storage and including a current-race sensing scheme. *IEEE Journal of Solid-State Circuits*, 38(1):155–158, 2003.

[94] Shoun Matsunaga, Kimiyuki Hiyama, Atsushi Matsumoto, Shoji Ikeda, Haruhiro Hasegawa, Katsuya Miura, Jun Hayakawa, Tetsuo Endoh, Hideo Ohno, and Takahiro Hanyu. Standby-power-free compact ternary content-addressable memory cell chip using magnetic tunnel junction devices. *Applied Physics Express*, 2(2):023004, 2009.

[95] Jing Li, Robert K Montoye, Masatoshi Ishii, and Leland Chang. 1 Mb 0.41 μm^2 2T-2R Cell Nonvolatile TCAM With Two-Bit Encoding and Clocked Self-Referenced Sensing. *IEEE Journal of Solid-State Circuits*, 49(4):896–907, 2014.

[96] Eero Lehtonen, Jussi H Poikonen, Mika Laiho, and Pentti Kanerva. Large-scale memristive associative memories. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 22(3):562–574, 2014.

[97] Meng-Fan Chang, Jui-Jen Wu, Tun-Fei Chien, Yen-Chen Liu, Ting-Chin Yang, Wen-Chao Shen, Ya-Chin King, Chorng-Jung Lin, Ku-Feng Lin, Yu-Der Chih, et al. 19.4 embedded 1mb reram in 28nm cmos with 0.27-to-1v read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 332–333. IEEE, 2014.

[98] Neil HE Weste and David Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education, 2011.

[99] Eric S Fetzer, David Dahle, Casey Little, and Kevin Safford. The parity protected, multithreaded register files on the 90-nm itanium microprocessor. *IEEE Journal of Solid-State Circuits*, 41(1):246–255, 2006.

[100] Benjamin Stolt, Yonatan Mittlefehldt, Sanjay Dubey, Gaurav Mittal, Mike Lee, Joshua Friedrich, and Eric Fluhr. Design and implementation of the power6 microprocessor. *IEEE Journal of Solid-State Circuits*, 43(1):21–28, 2008.

[101] Cheng-Lin Tsai, Feng Xiong, Eric Pop, and Moonsub Shim. Resistive random access memory enabled by carbon nanotube crossbar electrodes. *Acs Nano*, 7(6):5360–5366, 2013.

[102] Weihua Guan, Shibing Long, Qi Liu, Ming Liu, and Wei Wang. Nonpolar nonvolatile resistive switching in cu doped. *Electron Device Letters, IEEE*, 29(5):434–437, 2008.

[103] Igor Arsovski, Travis Hebig, Daniel Dobson, and Reid Wistort. A 32 nm 0.58-fj/bit/search 1-ghz ternary content addressable memory compiler using silicon-aware early-predict late-correct sensing with embedded deep-trench capacitor noise mitigation. *Solid-State Circuits, IEEE Journal of*, 48(4):932–939, 2013.

[104] James E Stine, Jun Chen, Ivan Castellanos, Gopal Sundararajan, Mohammad Qayam, Praveen Kumar, Justin Remington, and Sohum Sohoni. FreePDK v2. 0: Transitioning VLSI education towards nanometer variation-aware designs. In *Microelectronic Systems Education, 2009. MSE'09. IEEE International Conference on*, pages 100–103. IEEE, 2009.

[105] Abbas Rahimi, Amirali Ghofrani, Kwang-Ting Cheng, Luca Benini, and Rajesh K Gupta. Approximate associative memristive memory for energy-efficient gpus. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1497–1502. EDA Consortium, 2015.

[106] Cong Xu, Xiangyu Dong, Norman P Jouppi, and Yuan Xie. Design implications of memristor-based rram cross-point structures. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[107] A. Flocke and T. G. Noll. Fundamental analysis of resistive nano-crossbars for the use in hybrid nano/cmos-memory. In *ESSCIRC 2007 - 33rd European Solid-State Circuits Conference*, pages 328–331, Sept 2007.

[108] Evert Seevinck, Petrus J van Beers, and Hans Ontrop. Current-mode techniques for high-speed vlsi circuits with application to current sense amplifier for cmos sram's. *IEEE Journal of Solid-State Circuits*, 26(4):525–536, 1991.

[109] James E Stine, Ivan Castellanos, Michael Wood, and Fred Love. FreePDK: An Open-Source Variation-Aware Design Kit. In *IEEE International Conference on Microelectronic Systems Education*, pages 173–174, 2007.

[110] Cong Xu, Pai-Yu Chen, Dimin Niu, Yang Zheng, Shimeng Yu, and Yuan Xie. Architecting 3d vertical resistive memory for next-generation storage systems. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '14, pages 55–62, Piscataway, NJ, USA, 2014. IEEE Press.

[111] A. Ghofrani, M. A. Lastras-Montano, and K. T. Cheng. Toward large-scale access-transistor-free memristive crossbars. In *The 20th Asia and South Pacific Design Automation Conference*, pages 563–568, Jan 2015.

[112] Koji Nii, Teruhiko Amano, Naoya Watanabe, Minoru Yamawaki, Kenji Yoshinaga, Mihoko Wada, and Isamu Hayashi. 13.6 A 28nm 400MHz 4-parallel 1.6 Gsearch/s 80Mb ternary CAM. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 240–241. IEEE, 2014.

[113] Banit Agrawal and Timothy Sherwood. Ternary cam power and delay model: Extensions and uses. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(5):554–564, 2008.

[114] William C Elmore. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of applied physics*, 19(1):55–63, 1948.

[115] Kostas Pagiamtzis and Ali Sheikholeslami. A low-power content-addressable memory (cam) using pipelined hierarchical search scheme. *IEEE Journal of Solid-State Circuits*, 39(9):1512–1519, 2004.

[116] Aristides Efthymiou and Jim D Garside. An adaptive serial-parallel cam architecture for low-power cache blocks. In *Proceedings of the 2002 international symposium on Low power electronics and design*, pages 136–141. ACM, 2002.

[117] Charles A Zukowski and Shao-Yi Wang. Use of selective precharge for low-power content-addressable memories. In *Circuits and Systems, 1997. ISCAS'97., Proceedings of 1997 IEEE International Symposium on*, volume 3, pages 1788–1791. IEEE, 1997.

[118] Mohsen Imani, Abbas Rahimi, Pietro Mercati, and Tajana Simunic Rosing. Multi-stage tunable approximate search in resistive associative memory. *IEEE Transactions on Multi-Scale Computing Systems*, (1):17–29, 2018.

[119] Kostas Pagiamtzis and Ali Sheikholeslami. Pipelined match-lines and hierarchical search-lines for low-power content-addressable memories. In *Custom Integrated Circuits Conference, 2003. Proceedings of the IEEE 2003*, pages 383–386. IEEE, 2003.

[120] Chi-Sheng Lin, Jui-Chuan Chang, and Bin-Da Liu. A low-power precomputation-based fully parallel content-addressable memory. *IEEE Journal of Solid-State Circuits*, 38(4):654–662, 2003.

[121] Haitong Li, Kai-Shin Li, Chang-Hsien Lin, Juo-Luen Hsu, Wen-Cheng Chiu, Min-Cheng Chen, Tsung-Ta Wu, Joon Sohn, S Burc Eryilmaz, Jia-Min Shieh, et al. Four-layer 3d vertical rram integrated with finfet as a versatile computing unit for brain-inspired cognitive information processing. In *VLSI Technology, 2016 IEEE Symposium on*, pages 1–2. IEEE, 2016.

[122] Greg Snider. Instar and outstar learning with memristive nanodevices. *Nanotechnology*, 22(1):015201, 2010.

[123] Yuriy V Pershin and Massimiliano Di Ventra. Experimental demonstration of associative memory with memristive neural networks. *Neural Networks*, 23(7):881–886, 2010.

[124] Weste Neil HE et al. *Cmos Vlsi Design: A Circuits And Systems Perspective, 4/E.* Addison-Wesley, 2011.

[125] Rada Mihalcea. Semcor semantically tagged corpus. 11 1998.

[126] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[127] Tony A Plate. Holographic reduced representation: Distributed representation for cognitive structures. 2003.

[128] Pentti Kanerva. Binary spatter-coding of ordered k-tuples. In *International Conference on Artificial Neural Networks*, pages 869–873. Springer, 1996.

[129] Ross W Gayler. Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience. *arXiv preprint cs/0412059*, 2004.

[130] Michael N Jones and Douglas JK Mewhort. Representing word meaning and order information in a composite holographic lexicon. *Psychological review*, 114(1):1, 2007.

[131] Mika Laiho, Jonne K Poikonen, Eero Lehtonen, Mikko Pänkäälä, Jussi H Poikonen, and Pentti Kanerva. A 512$\times$ 512-cell associative cam/willshaw memory with vector arithmetic. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pages 1350–1353. IEEE, 2015.

[132] Abbas Rahimi, Sohum Datta, Denis Kleyko, Edward Paxon Frady, Bruno Olshausen, Pentti Kanerva, and Jan M Rabaey. High-dimensional computing as a nanoscalable paradigm. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2017.

[133] Denis Kleyko, Evgeny Osipov, Alexander Senior, Asad I Khan, and Yaşar Ahmet Şekercioğğlu. Holographic graph neuron: A bioinspired architecture for pattern processing. *IEEE transactions on neural networks and learning systems*, 28(6): 1250–1262, 2017.

[134] Simon D Levy and Ross Gayler. Vector symbolic architectures: A new building material for artificial general intelligence. In *Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, pages 414–418. IOS Press, 2008.

[135] Evgeny Osipov, Asad I Khan, and Anang Amin. Holographic graph neuron. In *Computer and Information Sciences (ICCOINS), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[136] Tony F Wu, Haitong Li, Ping-Chen Huang, Abbas Rahimi, Jan M Rabaey, H-S Philip Wong, Max M Shulaker, and Subhasish Mitra. Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study. In *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, pages 492–494. IEEE, 2018.

[137] Steven J Jones, Arthur R Wandzel, and John E Laird. Efficient computation of spreading activation using lazy evaluation. *Ann Arbor*, 1001:48109–2121, 2016.

[138] John R Anderson. A spreading activation theory of memory. *Journal of verbal learning and verbal behavior*, 22(3):261–295, 1983.

[139] Mohsen Imani, Abbas Rahimi, Deqian Kong, Tajana Rosing, and Jan M Rabaey. Exploring hyperdimensional associative memory. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pages 445–456. IEEE, 2017.

[140] E Paxon Frady, Denis Kleyko, and Friedrich T Sommer. A theory of sequence indexing and working memory in recurrent neural networks. *Neural computation*, 30 (6):1449–1513, 2018.

[141] Dan Bothell. Act-r 6.0 reference manual. *Working Draft*, 2004.

[142] Dominic Widdows and Dominic Widdows. *Geometry and meaning*, volume 773. CSLI publications Stanford, 2004.

[143] Kiichi Urahama and Takeshi Nagao. *k*-winners-take-all circuit with o (n) complexity. *IEEE Transactions on Neural Networks*, 6(3):776–778, 1995.

[144] Jui-Cheng Yen, Jiun-In Guo, and Hun-Chen Chen. A new *k*-winners-take-all neural network and its array architecture. *IEEE Transactions on Neural networks*, 9(5): 901–912, 1998.

[145] Barbaros Sekerkiran and Ugur Cilingiroglu. A cmos *k*-winners-take-all circuit with o (n) complexity. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(1):1–5, 1999.

[146] Bruce D Calvert and Corneliu A Marinov. Another *k*-winners-take-all analog neural network. *IEEE Transactions on Neural Networks*, 11(4):829–838, 2000.

[147] Corneliu A Marinov and Bruce D Calvert. Performance analysis for a *k*-winners-take-all analog neural network: basic theory. *IEEE Transactions on Neural Networks*, 14 (4):766–780, 2003.

[148] Pavlo V Tymoshchuk. A discrete-time dynamic *k*-winners-take-all neural circuit. *Neurocomputing*, 72(13):3191–3202, 2009.

[149] Jun Wang. Analysis and design of a *k*-winners-take-all model with a single state variable and the heaviside step activation function. *IEEE Transactions on Neural Networks*, 21(9):1496–1506, 2010.

[150] Makoto Ogawa, Kiyoto Ito, and Tadashi Shibata. A general-purpose vector-quantization processor employing two-dimensional bit-propagating winner-take-all. In *VLSI Circuits Digest of Technical Papers, 2002. Symposium on*, pages 244–247. IEEE, 2002.

[151] David C Hendry. Comparator trees for winner-take-all circuits. *Neurocomputing*, 62: 389–403, 2004.