Access-rate guaranteed memory controller

BERTA MORRAL ESCOFET
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Access-rate guaranteed memory controller

Berta Morral Escofet
be2103mo-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Babak Mohammadi

Examiner: Pietro Andreani

September 17, 2018

Printed in Sweden
Tryckeriet i E-huset, Lund

"Hope is the dream of a waking man"
Aristotle

Acknowledgements

I would like to thanks all the people that have helped me in any way to bring this project to its end. Firstly, to my supervisor Babak, who has always found time to discuss all the problems encountered along the project. Also, to Xiao, Kevin, and Roshan who patiently have assisted me with all the Cadence technical problems. And to Cati, for sharing with me all of those long days at work while making them a fun experience.

I would also like to thanks all my friends that have made these two years and incredible adventure.

Finally, thanks to my always supporting family which I am genuinely grateful for having them, and in special to Carles who has been at my side helping and supporting me all along the path.

Abstract

On-chip memory plays an important role in system-on-chip (SoCs) being in most cases the dominant part in both area and power. Additionally, it determines the overall system's speed. As a result, new memory architectures and technologies have been developed over the years in order to improve the overall system performance.

This project introduces the design and implementation of a memory controller algorithm that increases the throughput while guaranteeing a fix access-rate of the memory. The work is focused on incrementing the number of read operations, i.e., being capable of performing two-read operation per clock cycle, with the use of single-port memory banks.

Three different algorithms are implemented (XOR solution, Word Addition solution (WA), and Bit Addition solution (BA)) and are compared in terms of area, power and speed. Moreover, they are compared to the conventional two-port memory solution.

In addition, a simple BIST (built-in self-test) engine has been implemented in order to perform a basic functionality test in memory. The BIST module is integrated into the Word Addition solution.

The project concludes that the area per bit of the three solutions decreases as the size of the memory increases. However, it is by increasing the number of rows that the lowest cell area per bit values are achieved.

The three solutions reduce the power and the area compared to the conventional two-port memory solution, with the XOR solution being the most area and power efficient. Even though the area and power increase between solutions is significant, when considering the memory block system (memory controller and memory banks), the overall area and power difference is negligible. Moreover, the three solutions have shown to be able to work at higher speeds than conventional 28nm SRAM.

Also, the system with the integrated BIST into the memory controller has an area and power significantly smaller compared to the conventional two-port memory solution. Finally, the memory controller speed is not affected by the BIST.

Popular Science Summary

The ever-increasing connectivity between devices and amount of data transferred and processed in processing units implies stricter demands on memories. Because of this, during the past decades, an increasing tendency of the memory's area to the total area ratio of a processing unit has been observed. For this reason, novel techniques and architectures have been developed to produce smaller and more power efficient memories. Furthermore, memories can represent the main data-processing bottleneck. This master thesis proposes a design and implementation of a memory controller that allows multiple-access to its embedded memory without significantly increasing its area. Hence, the benefits of the memory's architecture are kept, while its utilization and data rate capabilities are enhanced. The proposed memory controller stores an additional set of values that allow recovering any data from the memory without having full access to it. Therefore, two values can be retrieved at the same time, one from the memory itself, and the other from the memory-controller additional stored data.

Table of Contents

List of Figures

List of Tables

Glossary

1R operation one-read operation. 5, 16, 28

1W operation one-write operation. 5, 28

2R operation two-read operation. 16, 19, 21, 28

BA Bit Addition Solution. xii, 15, 33, 40, 42, 43, 47, 48

BIST Build-In-Self-Test. v, ix, xiii, 2, 23, 33, 34, 44, 45, 47

BL Bitline. 5

BLB Bitline-inverse. 5

DP-SRAM Dual-Port Static Random Memory. 2, 6, 7, 44

FSM Finit State Machine. 34

HDL Hardware Description Language. ix, xi, 5, 23, 24, 33

LUT Look-Up-Table. 12

NB number of banks. 24, 29, 33, 37, 39, 41

NR number of rows. 24, 29, 33, 37, 39, 41

PnR Place and Route. ix, xii, xiii, 33, 42, 43

RBL Read bitline. 6

RTL Register Transfer Level. 33

RWL Read wordline. 6

SoC System-on-Chip. 10

SP-SRAM Single-Port Static Random Memory. 2, 5–7, 15, 23, 41, 44

SRAM Static Random Memory. v, ix, 5–7, 9, 47, 48

TP-SRAM Two-Port Static Random Memory. xii, 2, 6, 7, 41–44, 48

WA Word Addition Solution. xii, 15, 33, 40, 42, 43, 47, 48

WBL Word bitline. 6

WL Word Length. 7, 18, 20, 24, 29, 33, 37, 39, 41

Chapter 1
Introduction

Over the years, the number of electronic devices and their performance demands has increased dramatically. To meet these stricter demands, the functionalities implemented in a system on chip (SoC) have increased, and with that the amount of data that it needs to process. In order to process data with low latency, the on-chip memory in the SoCs has been forced to increase, becoming in many cases the dominant part of the die.

This can be seen in Figure 1.1, which shows the percentage of area assigned to on-chip memory regarding the overall SoC area over the years. It is possible to see the monotonic tendency of the on-chip memory, reaching nowadays in average, 68% of the total SoC area [1].

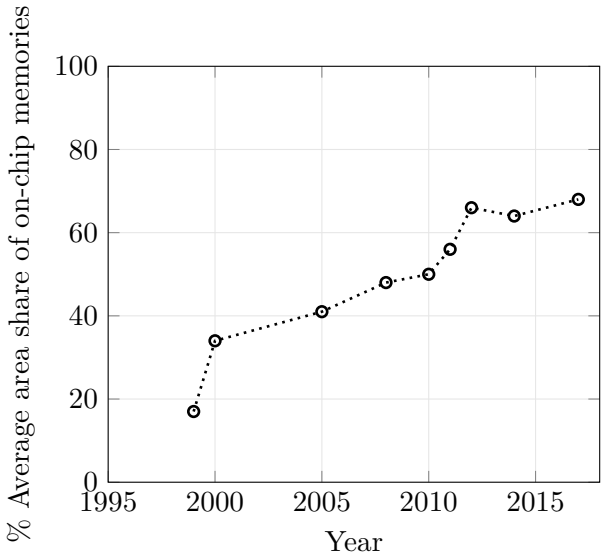


Figure 1.1: On-chip memory die area over the years.

Similar to the area, memory is often the block that consumes the major part of the total power in a SoC [12].

1.1 Motivation

Throughout the years, memories have been an important topic of research because they are the main bottleneck in most of the digital designs regarding area, power, and speed. To accommodate all the different memory requirements, the main focus has been on the memory architecture and technology. Several memory architectures (e.g., DP-SRAM and TP-SRAM) and technologies including new bit cells have been developed, achieving better and better throughput every time.

SP-SRAM architecture is the most area efficient, and the least power-consuming memory configuration. However, it only allows one read/write access per clock cycle. To further increase memory throughput, single-port memory configurations evolved into multi-port memory architectures, e.g., DP-SRAM [6]. For instance, DP-SRAM allows either to read or write two values simultaneously. Yet, these new configurations came at the cost of larger area and more power consumption.

Instead of finding a new memory architecture, this project focuses on how to improve the throughput of the memory by enhancing the memory controller. The advantages of this approach are:

- Reduction of the area. The penalty of using dual-port memory architecture is 50% to 100% compared to single-port memory architecture [2].
- Increased throughput. Allowing to perform more operations simultaneously.
- Not changing the memory architecture. Dual-port memory architecture is not always available in all technologies.

1.2 Main Goal

The main goal of this project is to utilize and study a memory controller algorithm to increase and guarantee the throughput of on-chip memories. The idea is to increase the throughput while still assuring a fixed access-rate without changing the memory architecture itself. The study case is done using SP-SRAM.

To perform the project, several solutions have been designed and implemented in 28nm CMOS technology, and compared in terms of area, power, and speed. Moreover, the memory controller includes a simple memory BIST to test the memory behavior.

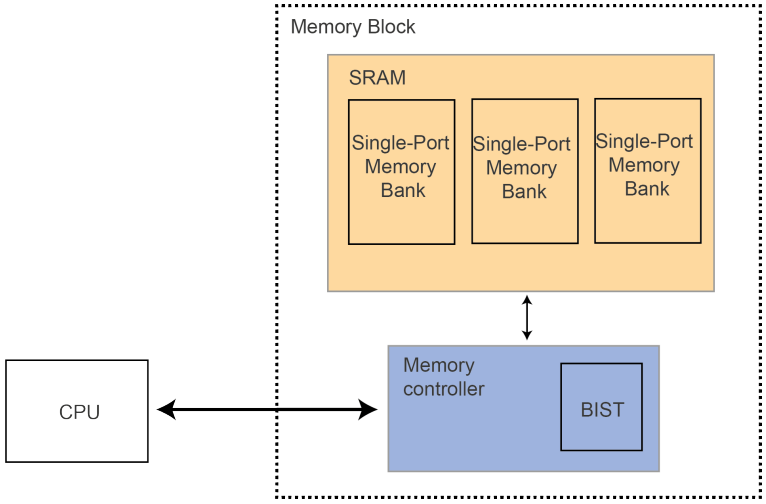


Figure 1.2: On-chip memory solution. The project scope is shown inside the dashed box.

Chapter 2
Relevant Concepts

Hardware description language - (HDL) Computer language used to describe the hardware of digital circuits, [21].

Cell Area Referred to the total area of a gate. Then the complete cell area is pointed to the addition of all gates used in the design.

Memory Block In this project we will refer as memory block to the memory banks and the memory controller, if not mentioned otherwise.

2.1 Memory architectures

Single-Port SRAM - (SP-SRAM) Figure 2.1 shows the conventional 6T SP-SRAM schematic of a bitcell, which is basic block element to store a binary value. The Bitline (BL) and Bitline-inverse (BLB) are used to either read or write into the bitcell once the wordline is activated, [3].

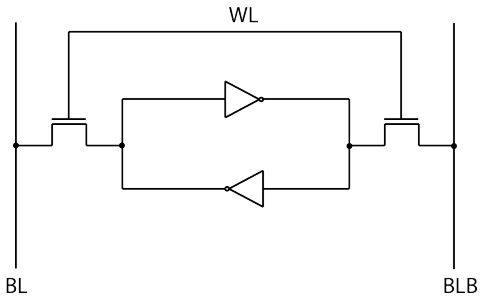


Figure 2.1: Single-Port SRAM.

The primary disadvantage of this architecture is that only one operation at a time is allowed. It means that either 1R operation or 1W operation is supported.

Dual-Port SRAM - (DP-SRAM) Figure 2.3 shows the conventional 8T DP-SRAM schematic of a bitcell. The DP-SRAM behavior is similar to the conventional 6T SP-SRAM but allows two parallel operations per clock cycle instead (using word bitlines 1 (WBL1s) and word bitlines 2 (WBL2s)).

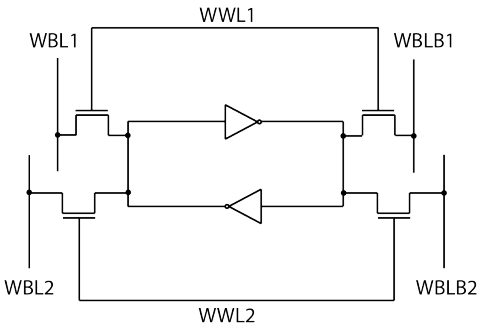


Figure 2.2: Dual-Port SRAM.

The major disadvantages of this architecture are the increase of area and power consumption, [4]. Moreover, simultaneous accesses at the same row affect to the memory stability [7].

Two-Port SRAM - (TP-SRAM) Figure 2.3 shows the conventional 8T TP-SRAM schematic of a bitcell. TP-SRAM behavior is similar to the conventional 6T SP-SRAM but allows two parallel read operations per clock cycle instead. The two parallel read operations are performed via WBLs and read bitline RBL. Moreover, read wordline RWL must be activated in order to perform the read through RBL, [5].

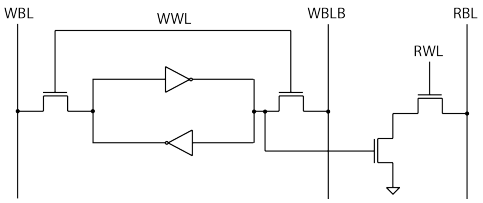


Figure 2.3: Two-Port SRAM.

The main disadvantages of this architecture are the increase of area and power consumption.

Table 2.1 sums up the SRAM architecture concepts previously explained:

SRAM Architecure	Num. Read operations	Num. Write operations	Simultaneous Read-Write operations
SP-SRAM	1	1	-
DP-SRAM	2	2	1-1
TP-SRAM	2	1	1-1

Table 2.1: SRAM architecture operations.

2.2 Parts of a memory

Memory Bank A Memory Bank refers to a memory unit. It is a portion of a memory that has the same functionality as the memory itself. Usually, an SRAM is made of several Memory Banks.

Row Row refers to a position where a concatenated number of bits conforming a word(words) is stored. Yet, in this project a row contains only one word and it is selected to read or write at the same time. Memory Banks are made of several rows.

Word Length (WL) Word Length (WL) is referred to the size of the word stored in a memory row.

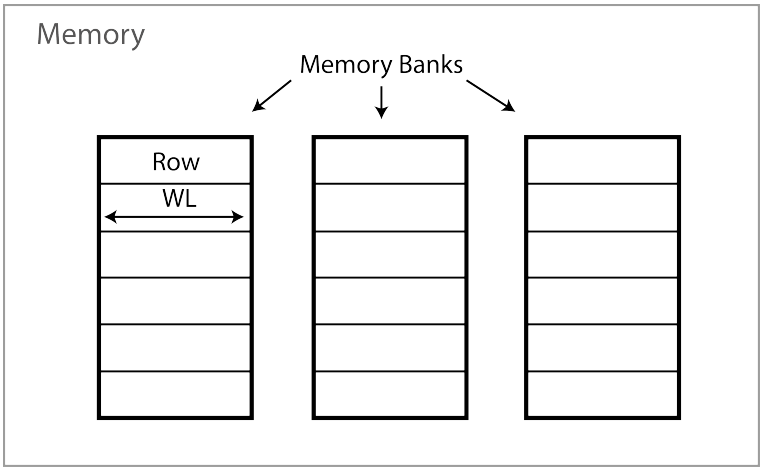


Figure 2.4: Parts of the memory.

Chapter 3
Background

3.1 Random-Access-Memory (RAM) Definition

A memory can be defined as an electronic design that can store information. There are many kinds of memories for different needs.

In this project we are focusing on RAM (Random-Access-Memory). One characteristic of this type of memories is that the time needed for reading or writing is independent of the physical data location. Moreover, RAM memories are usually volatile, which means that when the power is removed the data is lost.
There are two main types of RAM

- DRAM (Dynamic-Random-Access-Memory)
- SRAM (Static-Random-Access-Memory)

The main difference between them relies on its topology: DRAM needs the data stored to be refreshed periodically in order to prevent losing it. Yet, SRAM does not need a periodically refreshing control, so once data is stored it remains until the power is removed, [22].

3.2 Main concerns in SRAM

Memory plays an important role in digital systems, not only because they are the major area and power consumers on the die, but also it determines the overall system speed. Therefore, it is crucial to have following constrains in mind when designing, [10]:

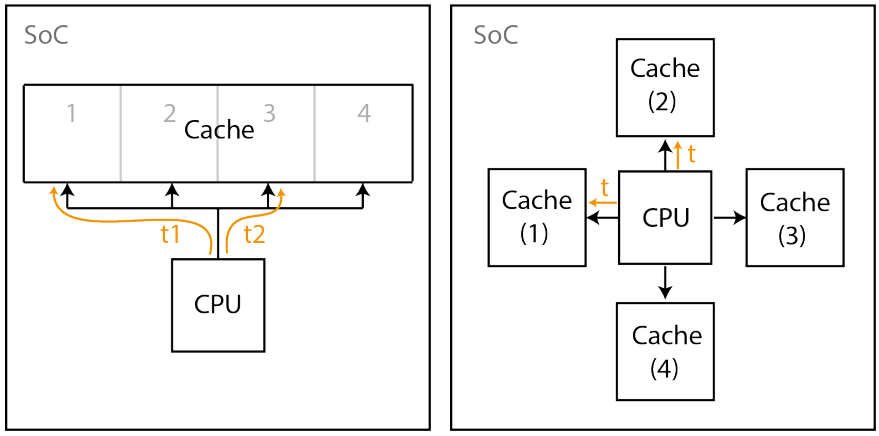
- Area
- Power consumption
- Speed
- Reliability

3.3 Available solutions

3.3.1 Distributed Cache Management - SoC

The access-time to a memory is dependent on the distance between the processor and the memory, therefore having a large memory may lead to a non-uniform latency [8]. The Distributed Cache Management method tackles this drawback. This method is based on splitting up and distributing the cache memory throughout the SoC. The processor sees one cache memory, yet the access-time is reduced as a result of the evenly distribution over the die [14].

Figure 3.1 shows the main idea of this method. In Figure 3.1a it is seen that the memory access-time is different depending on where data is read/written (t1 $\neq$ t2) as the path length differs. However, using the Distributed Cache Memory method, 3.1b, this problem is solved as the distance between the memory and the processor is the same.



(a) Non-Distributed Cache Management **(b)** Distributed Cache Management

Figure 3.1: Comparison of memory access-time between non-distributed and distributed method.

3.3.2 Interleaving memory banks

Interleaving memory banks is a method that is spreading the memory addresses between the memory banks. In other words, it does not assign the addresses in the memory banks sequentially.

For instance, if we have four memory banks of 16 bytes each, in block oriented method the addresses in the memories are assigned consecutively: 0-15 first bank, 16-31 second bank, and so forth. On the contrary, if interleaving method is used, address '0' is assigned to the first memory bank, address '1' is assigned to the second memory bank, and so forth.

This method increases the throughput significantly when the required data is a set of consecutive words. The reason for this is the possibility to access each memory bank independently [15].

Figure 3.2 shows the interleaving memory banks concept. In 3.2a we can see the advantage of using this method when a set of consecutive words is required, resulting in providing all the data in one clock cycle. However, this method does not guarantee the throughput, as it is shown in Figure 3.2b where the data is provided in two clock cycles instead of one. This is due to the fact that two data values from the same memory bank are requested.

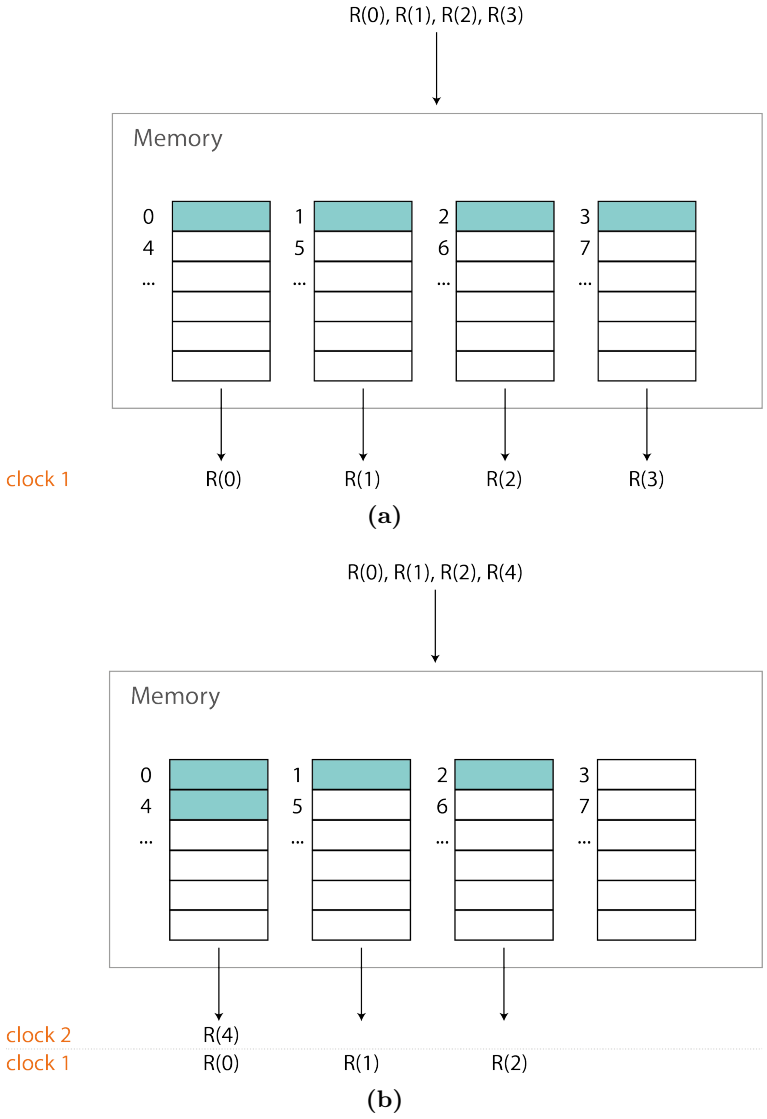


Figure 3.2: Interleaving memory banks method.

None of the solutions previously explain (Distributed Cache Management and Interleaving memory banks) can assure the access-rate of the memory, and therefore not suitable solutions for this project.

3.3.3 Memory algorithm

In [16], two main algorithms to increase the throughput of RAM memories are described.

The main purpose of the first algorithm is to increase the number of write operations. To achieve this behaviour, an extra memory and a LUT to have full control of state of the memory are needed. In general terms, when a two-write operation is generating a conflict, meaning that they must be written in the same memory bank, one of the data-write is stored in the extra memory, reflecting this change in the LUT.

The purpose of the second algorithm, explained in the section 4.1, is to increase the number of read operations. This algorithm allows to recover any value from an array of values, due to the addition of a memory space where the XOR of all values in the array is stored. This method allows to perform two read operation simultaneously.

The research in [16] has been a primary source of inspiration and the starting point for this project.

3.4 Alternative solutions

Moreover, three alternative solutions were explored in order to analyze the feasibility of using them for the project purpose.

3.4.1 Compression

The first solution taken into account was to compress the input data and store it in an extra memory bank. Unfortunately, the compression ratio of a compression algorithm is not deterministic since it depends on its input values. Therefore, this solution is not suitable for the study case, because the additional memory would be required to have the same size as the main memory to cover for the worst case scenario (i.e., no compression possible). If the extra memory size was set according to the compression ratio average, there would be a risk of not being able to store all the data [17].

3.4.2 Error detection and correction

Next solution was error detection and correction. Some of the existing methods such as VRC, LCR and CRC for detection and Hamming code for correction, [20] were carefully studied. Once it was known how these methods work, the combination of some of them were considered as a possible solution. This type of methods were found to be unsuccessful since they required the same amount of bits as the saved data.

3.4.3 Hash Functions

Last solution to explore was hash functions. They were considered because they give the possibility to get a fix output data size from an arbitrary input data size, [18].

Usually, hash functions are used to compare two files without knowing its content. Hence, they are designed to prevent being reversed, which was the main problem found for this project, [19].

Chapter 4
Algorithm Design

This chapter explains the selected algorithms and analyses them.

The main purpose of this project was to find an alternative approach of the algorithm described in 3.3.3. The focus was on finding an algorithm to increase the throughput in terms of performing either one write operation or two read operations at a time.

The major drawback in this performance relies on how to perform two read operations at the same time from the same memory bank, having in mind that the memory bank has a SP-SRAM architecture.

The idea behind this is to autocorrelate the data stored in the memory so that any value from a memory bank can be indirectly known by reading values from other memory banks.

Three different algorithms are chosen:

- XOR - Memory algorithm mentioned in 3.3.3.
- Word Addition (WA).
- Bit Addition (BA).

In the following subsections each considered algorithm is explained in detail.

4.1 XOR

This solution is the one presented in 3.3.3, the basic idea is to use an extra memory bank with the same size as the other memory banks. However, the extra memory bank will store the XOR of the other memory banks.

Write Operation In Figure 4.1, the write operation is shown. There is one input data, data_in(W_A1), which indicates that a value needs to be written (W) in the first row of the bank named 'A', i.e., address 'A1'. To perform a write operation four tasks must be done:

1. Write the value "data in" (d_in) at the specified address: "A1".
2. Read the values from all the other banks at the specified "data in" row address: "B1" and "C1".

3. Perform the XOR of the values previously read and "data in": ([B1] $\oplus$ [C1] $\oplus$ [D_in]) [1].
4. Write the result of the XOR into the extra memory bank in the specified "data in" row address: "Extra1".

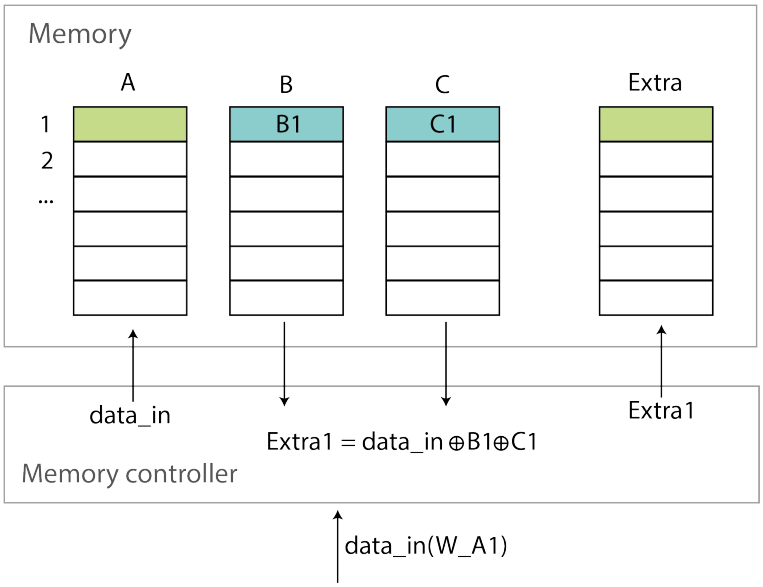


Figure 4.1: Algorithm XOR - write operation

Read Operation The read operation leads to three different scenarios:

- 1R operation.
- 2R operation from two different banks.
- 2R operation from the same bank.

For the first two cases, nothing special needs to be done, as the values can be read without conflict from different banks. The problem is faced when 2R operation should be performed from the same memory bank. Figure 4.2 is used to explain the algorithm to solve the third case. In the figure, it is possible to see four signals - two of them are the input addresses and the other two the output values. The input data, "addr1 = R_B1" and "addr2 = R_B3" indicate that two values need to be read (R) from the bank named 'B' from the first and third position, respectively. The output values "data1_out" and "data2_out" will contain the data read from "B1" and "B3". To perform the read operation, the following steps must be done:

1. Read the value from the first specified address: "data1_out = [B1]".

[1] In the following explanations, we will use "[X]" to designate the value content of address "X".

2. Read the values from all the other banks at the third row : "A3" and "C3".
3. Read the value from the extra memory bank at the third row: "Extra3".
4. Perform the XOR of all values previously read, with the exception of the value from the memory bank that generates conflict ('B'), and the value from the extra memory bank: "data2_out = [A3] $\oplus$ [C3] $\oplus$ [Extra3]".

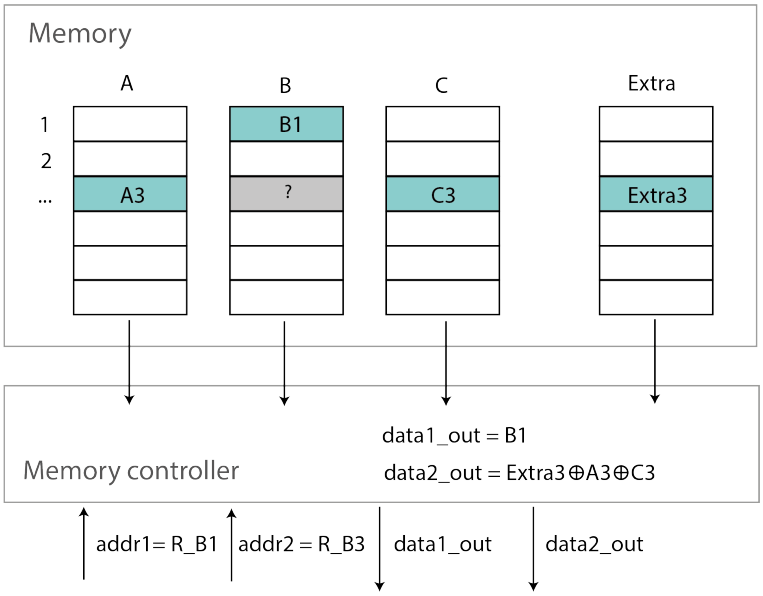


Figure 4.2: Algorithm XOR - read operation

Using the XOR technique, the second value can be "recovered" without reading the value from its memory bank, therefore the problem of reading two values from the same memory bank is solved.

4.2 Word Addition

As in the previous solution, the basic idea is the use of an extra memory bank with the same size as the other memory banks, however, for every row, the addition of all its words will be stored.

It is important to highlight that the extra memory bank has the same size as the other memory banks used in the system. Hence the carry-out is not considered and not required for this algorithm.

To explain the reason of why the carry-out is not needed, the following example is used. Following abbreviations are needed:

- R_F (Final Result) - Result from the addition of all memory banks.
- R_E (Extra memory bank Result) - Value stored into the extra memory bank.

- R_P (Partial Result) - Result from the addition 'N-1' memory banks. Being 'N' the total number of memory banks.
- R_{value} (Value Result) - Result from the operation. The "unknown" value recovered.

For instance, having the scenario of four memory banks of WL=2, and each of them having a value stored A=2, B=3, C=3 and D=1 ($Bank_i$). The result from the addition of all memory banks is $R_F = 9$ (b1001) but the value stored in the extra memory bank is the WL LSB bits from the result $R_E = 1$ (b01). Table 4.1 analyses case by case, but before it is important to highlight the equation used to get the results:

$$R_{value} = \begin{cases} R_E - R_P, & \text{if } R_E \geq R_P \\ (2^{WL} + R_E) - R_P, & \text{if } R_E < R_P \end{cases} \tag{4.1}$$

Since R_F must always be bigger than R_P, we can assume that when R_E is smaller than R_P an overflow has been produced. Therefore the result, R_{value}, must be calculated adding the overflow to the subtraction of R_P.

Case ($Bank_x$)	$\mathbf{R_P = (\sum_{i=0}^{2} Bank_i) - Bank_x}$	**Comparison R_P and R_E**	$\mathbf{R_{value}}$
$[A]$	7(b0111) $\rightarrow$ R_P = 3(b11)	$3 > 1$	4+1-3= 2
$[B]$	6(b0110) $\rightarrow$ R_P = 2(b10)	$2 > 1$	4+1-2= 3
$[C]$	6(b0110) $\rightarrow$ R_P = 2(b10)	$2 > 1$	4+1-2= 3
$[D]$	8(b1000) $\rightarrow$ R_P = 0(b00)	$0 < 1$	1-0= 1

Table 4.1: Example size of extra memory bank - Word Addition.

The reason is that a bit pattern is repeated every 2^N, therefore for an addition or subtraction of values smaller than 2^N, the bit pattern configurations are unique. Hence, storing 2^N is enough to "recover" any value from the memory.

Write Operation In Figure 4.3, the write operation is shown. There is one input signal, data_in(W_A1), which indicates that a value needs to be written (W) in the first row of the bank named 'A', i.e., address 'A1'. To perform a write operation four tasks must be done:

1. Write the value at the "data in" (data_in) specified address: 'A1'.
2. Read the values from all the other banks at the specified "data in" row address:' B1' and 'C1'.
3. Perform the addition of the values previously read in step 2 to "data in": "[B1] + [C1] + [D_in]".
4. Write the result of the addition into the extra memory bank in the specified "data in" row address: "Extra1".

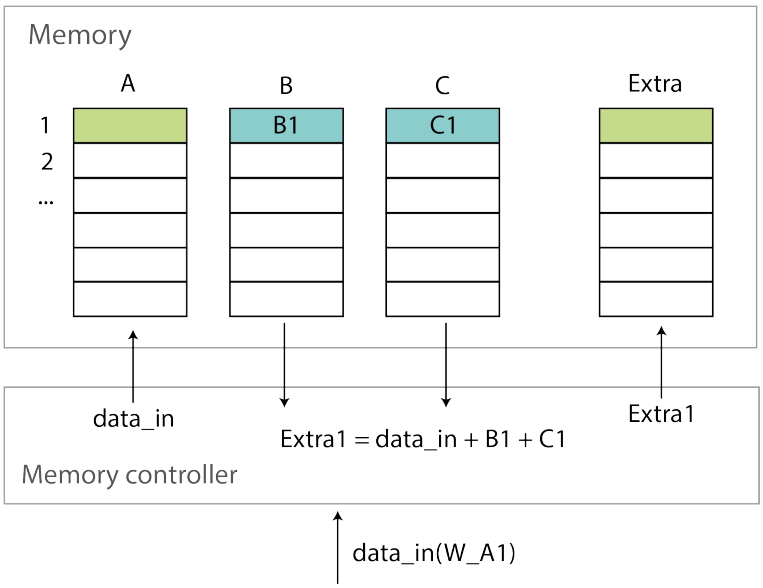


Figure 4.3: Algorithm Addition Words - write operation.

Read Operation The read operation leads to the same scenarios as in the case explained in subsection 4.1. Figure 4.4 is used to explain the algorithm when 2R operation are performed from the same memory bank. In the figure, it is possible to see four signals - two of them are the input addresses and the other two the output values.

The input data, "addr1 = R_B1"and "addr2 = R_B3" indicate that two values must be read (R) from the bank named 'B' from the first and third row, respectively. The output values "data1_out" and "data2_out" contain the data stored in "B1" and "B3". The following steps must be done in order to perform a two-read operation:

1. Read the value from the first specified address: "data1_out = [B1]".
2. Read the values from all the other banks at the third row : "A3" and "C3".
3. Read the value from the extra memory bank at the third row: "Extra3"
4. Addition of the values: "Rp = [A3] + [C3]"
5. Subtract the addition done in step four to the extra memory bank value: "data2_out = [Extra3] - [Rp]"

Using the Word Addition technique, the second value can be "recovered" without reading the value from its memory bank, therefore the problem of reading two values from the same memory bank is solved.

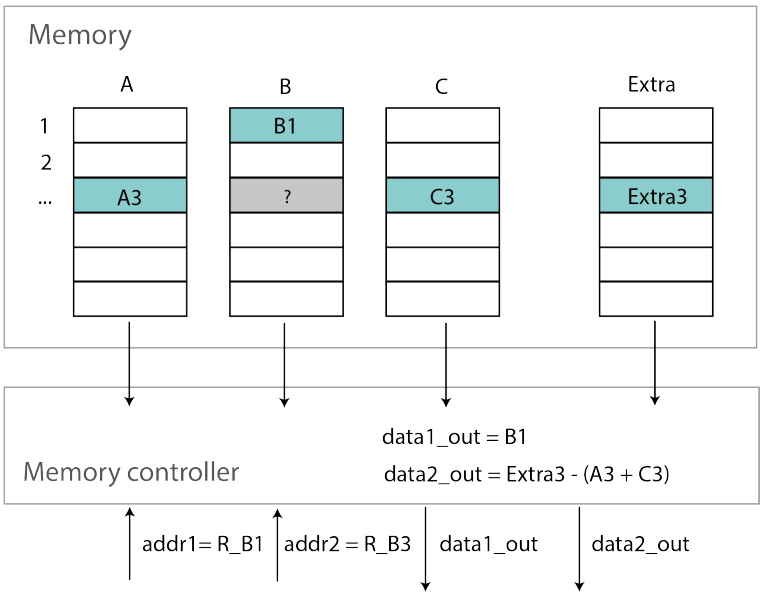


Figure 4.4: Algorithm Addition Words - read operation

4.3 Bit Addition

The Bit Addition solution uses an extra memory bank, as the other solutions previously explained but in this case, the size may be different from the memory banks size. In this situation, the addition of each position of all the bits in a row that share the same bit position in the word is stored (for each row).

It is important to notice that the size of the extra memory bank is $log_2(NB)/log_2(2) \times WL$. $log_2(NB)/(2)$, number of bits required per addition of each position of all the bits in a row that share the same bit position in the word. As in the Word Addition Solution 4.2, the carry-out is not stored in the extra memory bank. In this case, the carry-out refers to all MSB bits bigger than $log_2(NB)/log_2(2)$ generated as a consequence of each addition.

For instance, if there are four memory banks (B1, B2, B3, B4) each having WL = 1, in the extra memory bank only 2 bits per row are stored ($log_2(4)/log_2(2) \times 1 = 2$). In consequence, when the result of the addition is '4' (b100) the stored value is '0' (b00). Storing the value '0' does not create a collision between results and the reason is because only two possible combinations have '0' as a result:

- All memory banks values is '0'.
- All memory banks values is '1'.

Hence, if a '0' value is stored in the extra memory bank and the result from the addition of N-1 banks is '0', the value stored in the "unknown" memory bank must be '0'. Whereas if the result from the same addition is '3', the value stored in the "unknown" memory bank must be '1'.

Write Operation Write operation is shown in Figure 4.5. There is one input data, data_in(W_A1), which indicates that a value needs to be written (W) in the first row of the bank named 'A', i.e., address 'A1'. To perform a write operation in 'Bit Addition solution' five steps need to be done:

1. Write the value at the "data in" (d_in) specified address: "A1".
2. Read the values from all the other banks at the specified "data in" row address: "B1" and "C1".
3. Perform the addition of each bit position of each row read: "Extra(1) = [d_in(1)] + [B1(1)] + [C1(1)]" , "Extra(0) = [d_in(0)] + [B1(0)] + [C1(0)]".
4. Concatenate the result values from the addition: "Extra1 = [Extra(1)] & [Extra(0)]".
5. Write the addition result into the extra memory bank in the specified "data in" row address: "Extra1".

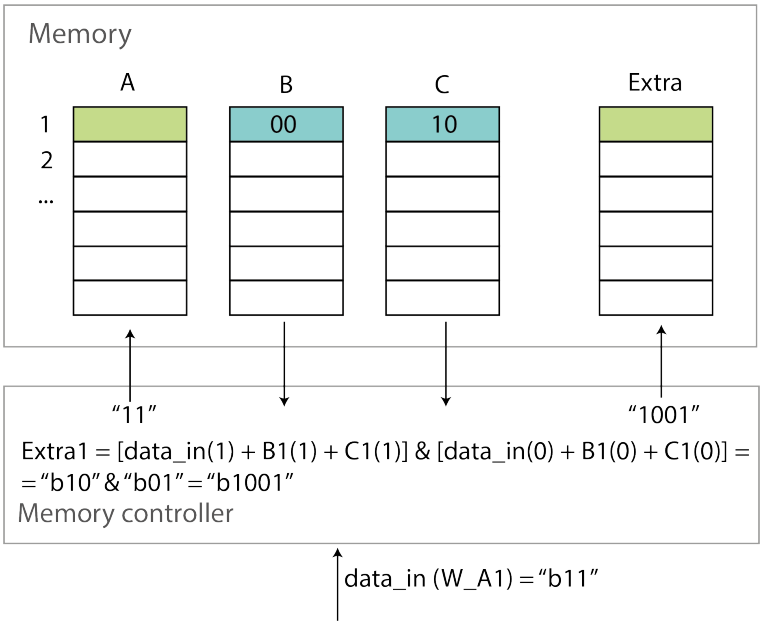


Figure 4.5: Algorithm Addition Bits - write operation

Read Operation The read operation will lead us to the same scenarios as in the case explained in subsection 4.1. Figure 4.6 is used to explain the algorithm when 2R operation are performed from the same memory bank. In the figure, it is possible to see four signals - two of them are the input addresses and the other two the output values.

The input signals, "addr1 = R_B1"and "addr2 = R_B3" indicate that two values must be read (R) from the bank named 'B' from the first and third rows,

respectively. The output values "data1_out" and "data2_out" contain the data stored in "B1" and "B3". The following steps must be done in order to perform a two-read operation:

1. Read the value from the first specified address: "data1_out = [B1]".
2. Read the values from all the other banks at the third row : "A3" and "C3".
3. Read the value from the extra memory bank at the third row: "Extra3"
4. Perform the addition of each position of all the bits in a row that share the same bit position in the word except the memory bank already used in step 1: "B3(1)= [Extra3(1)] - ([A3(1)] + [C3(1)])", "B3(0)= [Extra3(0)] - ([A3(0)] + [C3(0)])"
5. Concatenate the result values from the addition: "data2_out = [B3(1)] & [B3(0)]".

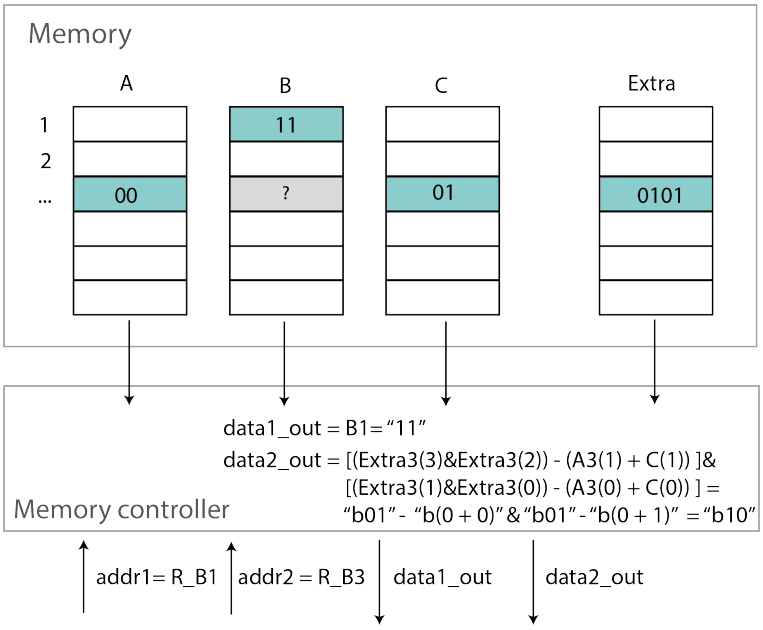


Figure 4.6: Algorithm Addition Bits - read operation

Using the Bit Addition technique, the second value can be "recovered" without reading the value from its memory bank, therefore the problem of reading two values from the same memory bank is solved.

Chapter 5
Implementation Phase

This chapter explains how the project has been realized, explaining in detail each of its parts. It is divided in three main blocks: HDL Implementation, Behavioural HDL Test, and BIST.

- HDL Implementation: The VHDL implementation of each solution is described.
- Behavioural HDL Test: How the behavioural test has been done for each solution.
- HDL: The BIST is explained in detail.

Figure 5.1 shows an overall picture of the proposed memory solution. The memory contains several SP-SRAM memory banks and the memory controller with an integrated memory BIST. In this solution, both the CPU and the memory banks communicate with the memory controller, which properly handles the communication between them.

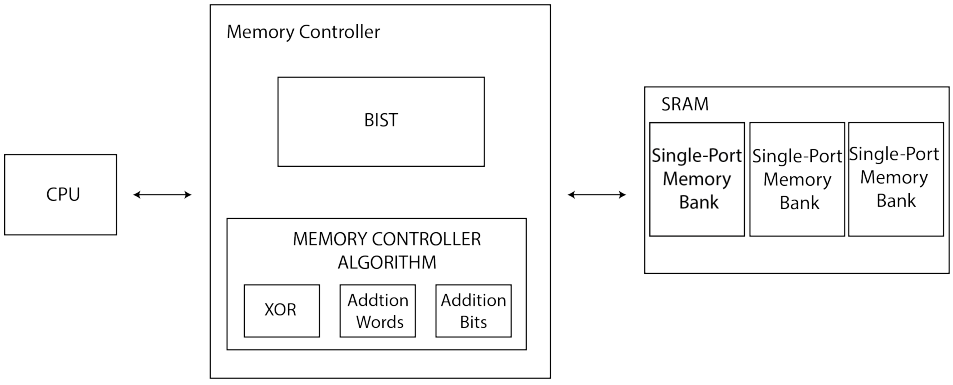


Figure 5.1: HDL Overview of the project.

5.1 HDL Algorithms Implementation

Before going into details in this section, it is important to have in mind that all designs have been developed to perform the operations within a clock cycle. The reason is to avoid having two different clock domains between the memory controller and the memory and to determine the maximum achievable clock frequency. Knowing the maximum frequency not only gives us an idea of what kind of memories the memory controller can be integrated into but also the worst case scenario for the area. The reason for the area penalty is due to the extra hardware needed in order to perform all the operations within one clock cycle.

Figure 5.2 shows the top view of the memory controller implemented. In general terms, it has several control signals, a bus with the memory addresses and an in-out bus for the data.

The three solutions are implemented having the following generic parameters:

- Word Length (WL).
- Number of memory banks (NB).
- Number of words per memory bank (NR).

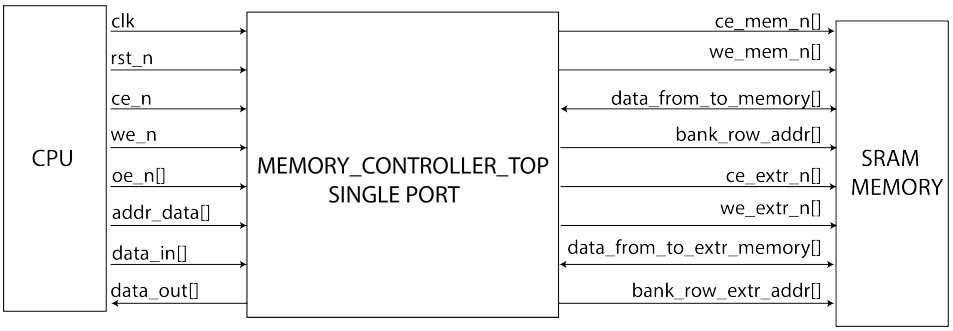


Figure 5.2: Top memory controller overview.

Figure 5.3 shows the top view for all the solutions (XOR, Words Addition, and Bit Addition). All solutions will be explained under the same section as they share the same main blocks, the only difference between them is the behavior of the 'output_memory_controller' (XOR, Word Addition, or Bit Addition).

The top design is divided into two main blocks - 'input_memory_controller' and 'output_memory_controller'.

This block not only has its instantiations but also handles the in/out signals connected to the memory banks, and registers the input and output signals connected to the CPU.

In order to handle the in/out signals, tristate buffers have been used. During reading operations, they will be in high impedance, and thus the memory will be allowed to write into them.

The signals connected to the CPU are registered in order to have a stable value within a clock cycle.

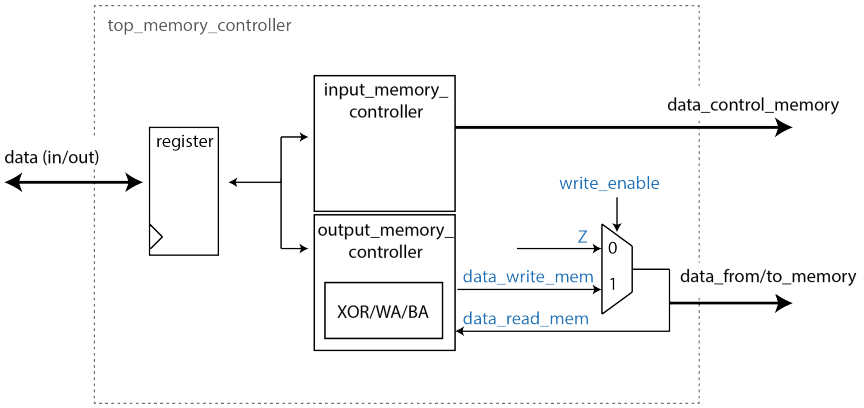


Figure 5.3: Top design of memory controller solutions - Implementation.

5.1.1 'input_memory_controller' Block

The input block ('input_memory_controller') is a combinational block responsible for configuring the output control signals going into the memory banks. Hence, it has to activate or deactivate the memory bank and the specific rows that must be read or write.

Figure 5.4 shows the behavior of the 'input_memory_controller' block. The block depends on three input signals: "we_n_reg" - Enable Write operation (Active low), "oe_n_reg" - Enable read/2-read operation(Active low), and "in_address _reg" - Specifies the memory address [Bank ("in_bank_addr1" and "in_bank_addr2") and Row ("in_row_addr1" and "in_row_addr2")]. Depending on the configuration of the input signals, it takes us to four different situations: 1-read operation, 2-read same bank operation, 2-read different bank operation and 1-write operation.

To be able to control output memory banks behavior for this four situations, three different output signals have been set:

- "ce_mem_n" - Memory bank enable bus - Active low. Enables or Disables the memory banks of the system for either reading or writing, e.g., "ce_mem_n = b1010" means that memory banks 1 and 3 are enabled.
- "we_mem_n" - Write memory bank enable bus - Active low. Enables or Disables the memory banks of the system for writing, e.g., "we_mem_n = b1010" means that memory banks 1 and 3 are enabled to be written.
- "bank_row_addr" - Configure the row for each bank that must be read or written. The bit position in the bus is referred to the memory bank and the value of the bit(bits) refers to the row, e.g. "bank_row_addr = b1001" means that the row to read or write from memory bank 1 is '1' from memory bank 2 is '0' from memory bank 3 is '0' and from memory bank 4 is '1'.

The same signals with the same behavior but with different names have been used to activate or deactivate the extra memory bank.

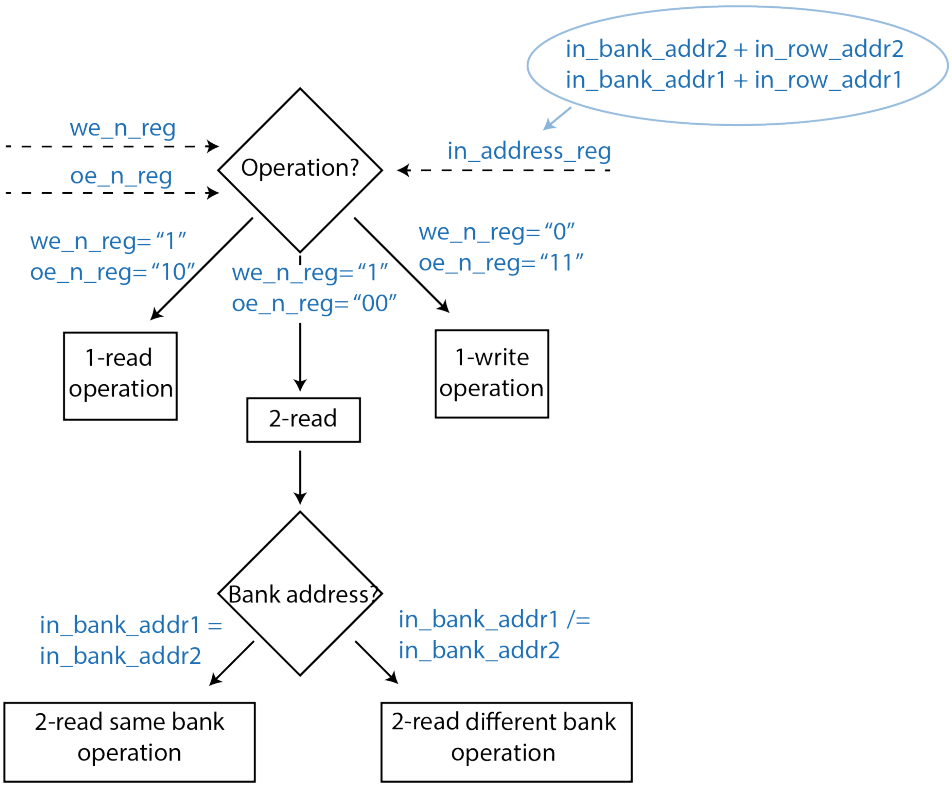


Figure 5.4: Input memory controller block - Flow diagram.

Figure 5.5 shows an example of how the output signals are configured in each situation. Each box represents a memory bank, therefore, four memory banks are considered in this example. Moreover, coloured boxes mean that a specific signal (ce_mem_n and we_mem_n) for a particular memory bank is enabled and white boxes mean that is disabled. Additionally, the numbers inside the boxes correspond to the row that must be read or write. An empty box means that the memory bank is not required for that operation.

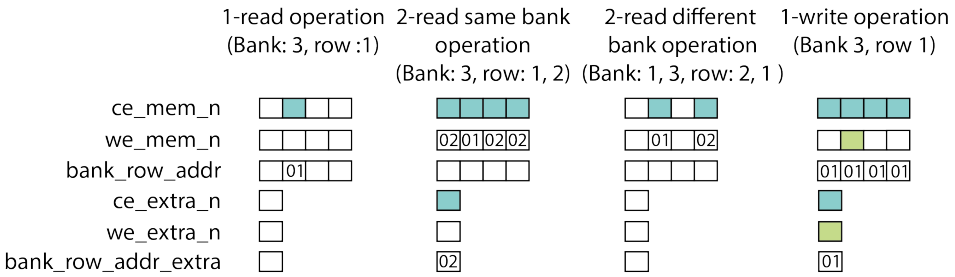


Figure 5.5: Output signals configuration example from input memory controller block.

Figure 5.5 is used as an example to explain each situation:

- 1-read operation - To be able to read the value bank 3 row 1 must be enabled. Therefore the configuration of the output signals are: "ce_mem_n = b1011", "we_mem_n = b1111" and "bank_row_addr = X1XX"[1]. The extra memory bank is not required in this operation.
- 2-read same bank operation - In this operation all memory banks must be enabled using row 2 except the memory bank 3 which uses row 1 instead. Therefore the configuration of the output signals are: "ce_mem_n = b0000", "we_mem_n = b1111" and "bank_row_addr = 2122". The extra memory bank needs to be enabled reading the row 2. Hence, the configuration of the extra memory bank output signals are: "ce_extra_n = 0", "we_extra_n = 1" and "bank_row_addr_extra = 2"
- 2-read different bank operation - In this case the Banks 1 and 3 and rows 1 and 2 respectively must be enabled. Therefore the configuration of the output signals are: "ce_mem_n = b1010", "we_mem_n = b1111" and "bank_row_addr = X2X1". The extra memory bank is not required in this operation.
- 1-write operation - To write into the memory bank all of them must be enabled using row 1. Moreover, memory bank 3 must be enabled to be written. Therefore the configuration of the output signals are: "ce_mem_n = b0000", "we_mem_n = b1011" and "bank_row_addr = 1111". The extra memory bank must be enabled to be written into the row 1. Hence, the configuration of the extra memory bank output signals are: "ce_extra_n = 0", "we_extra_n = 0" and "bank_row_addr_extra = 1"

[1] X is considered as not relevant value

5.1.2 'output_memory_controller' Block

The output block ('output_memory_controller') is a combinational block responsible for providing the output data signals going to both the CPU and the memory banks.

Figure 5.6 shows its block diagram. The output block gives the right input data to the 'XOR/Word Addition/Bit Addition' Block ('calculation block') and configures the output data coming from these blocks depending on the scenario (1W operation, 1R operation, 2R operation - same bank or 2R operation - different bank). Therefore, not only configures the output data, but also configures the input data for the 'calculation block' ("data_calc").

Input data from 'calculation block' ("data_calc") is configured differently depending on the operation. If it is a 2R operation from the same bank ('B_X'), then the input data is configured as the data from N-1 memory banks and '0' instead of the memory bank 'B_X' data. If it is a write operation writing into the memory bank 'B_X', input data is configured as the data from N-1 memory banks and "[data_in]" instead of 'B_X' data.

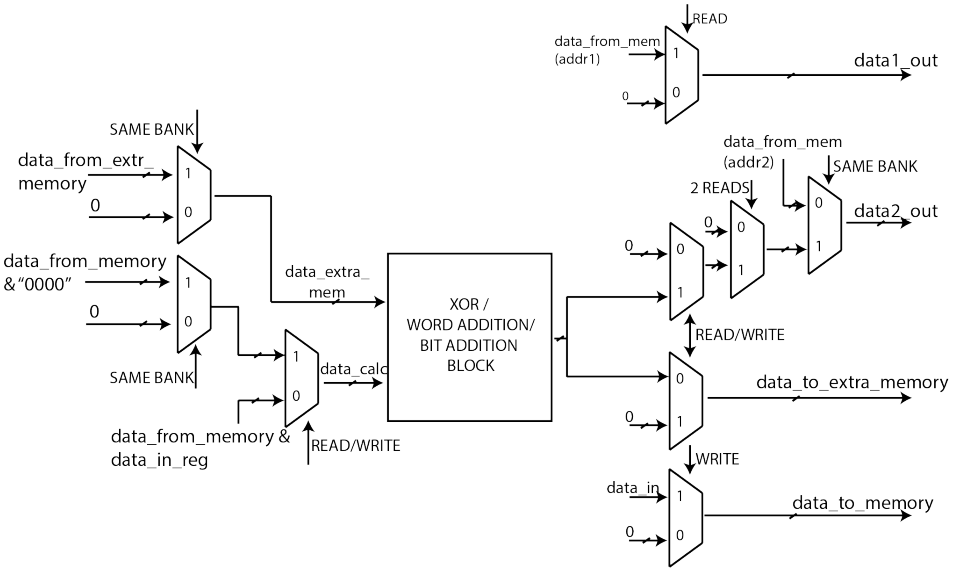


Figure 5.6: Output memory controller block - block diagram.

XOR Block This block is responsible to perform the XOR of all words given as an input and provide the output value that is either stored in the extra memory bank or provided as the second data value read to the CPU.

Figure 5.7 shows the top view of this block. According to the scenario (write/read) the output of this block ("result") is given by the 'XOR Calc Block' - Write operation -, or from the result of the XOR operation between the result of the 'XOR

Calc Block' and the "data_extra_mem[2]" - Read operation.

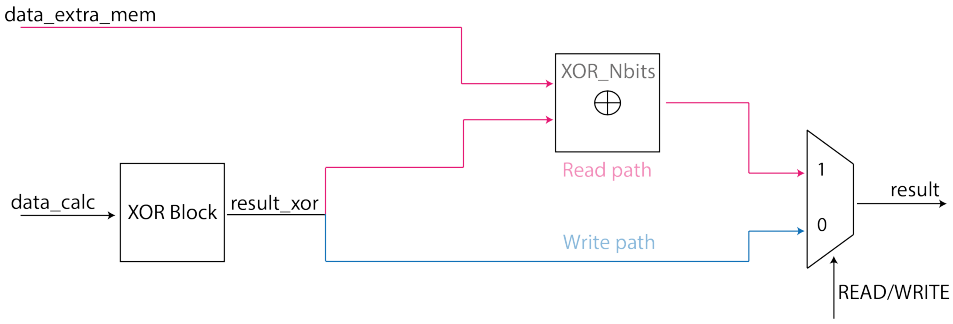


Figure 5.7: Top XOR - block diagram.

To perform all the XOR's, a pure combinational tree structure has been used. The idea of using this method is, to have the result as soon as possible and within one clock cycle. This block is highly dependent on the generic parameters (WL, NB and NR). Consequently, the hardware needed in the tree structure block is increased or decreased significantly based on this.

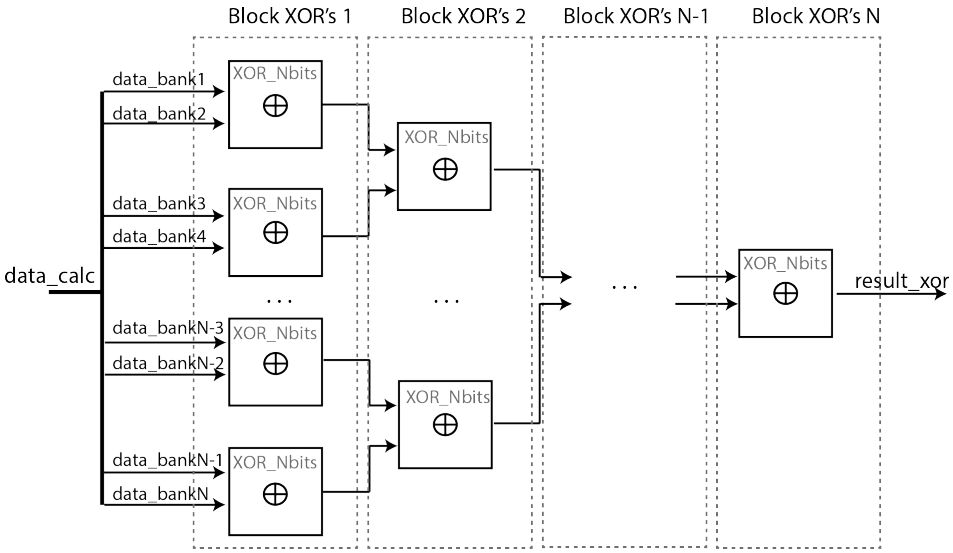


Figure 5.8: Tree structure XOR - block diagram.

Word Addition Block This block is responsible to calculate the output value that is, as in the XOR Block, either stored in the extra memory bank or provided as the second data value read to the CPU.

[2]"data_extra_mem": data read from the extra memory bank

Inside this block there is another block, named Adder Block. This block is responsible of performing the addition of the data given as "data_calc".

In this case, there are two different scenarios to take into account - read and write. The write path (blue line in the Figure 5.9), the result is the 'Adder Block' result ("result_adder"). For the read path (pink line in the figure 5.9), the result will be the subtraction of "data_extra_mem - result_adder". To do this operation, the same full_adder as the addition part has been used. In order to do it, two's complement transformation must be applied to "result_adder" signal ("not(result_adder) + 1").

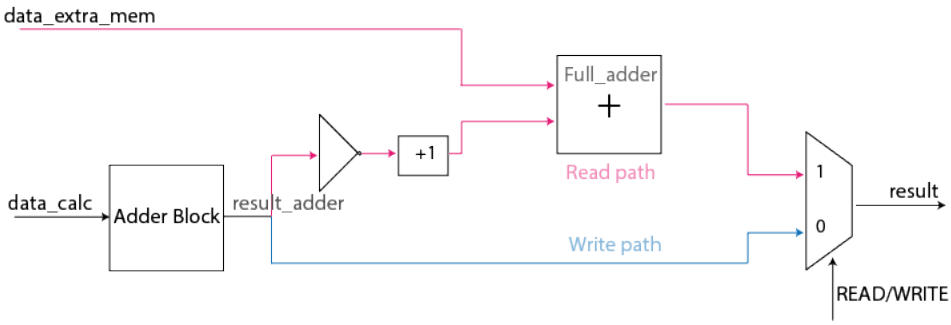


Figure 5.9: Word Addition - block diagram.

Adder Block Figure 5.10 shows its block diagram. The goal of this block is, given "data in" which contains the data from a row of 'N' number of memory banks, it provides as a result the addition of all of them. As in the XOR Block design, the addition will be performed using a combinational tree structure. This block is also highly dependent on the generic parameters configuration used.

Full Adder To perform the addition as fast as possible, a carry look-ahead method has been used. The idea of this method is to calculate the carry-out in advance based on the input signals, more information can be found here [9].

As the method becomes more complex as the number of bits increase, a cascade of 4 bits carry look-ahead adders has been used to generate all the adder sizes needed.

Bit Addition Block This block, that is shown in 5.11, is responsible for calculating the output that will be either stored in the extra memory bank or provided as a second data value read to the CPU. This block has two inputs: "data_calc" and "data_extra_mem" and one output "result_addition", which is dependent on the operation performed. The behaviour of this block is as follows: "data_calc" is a bus that contains the data from a row of several memory banks. This signal goes into a block named 'Generate Vectors' which reorganizes the data in order to perform the addition correctly. Then, it gives as an output, WL number

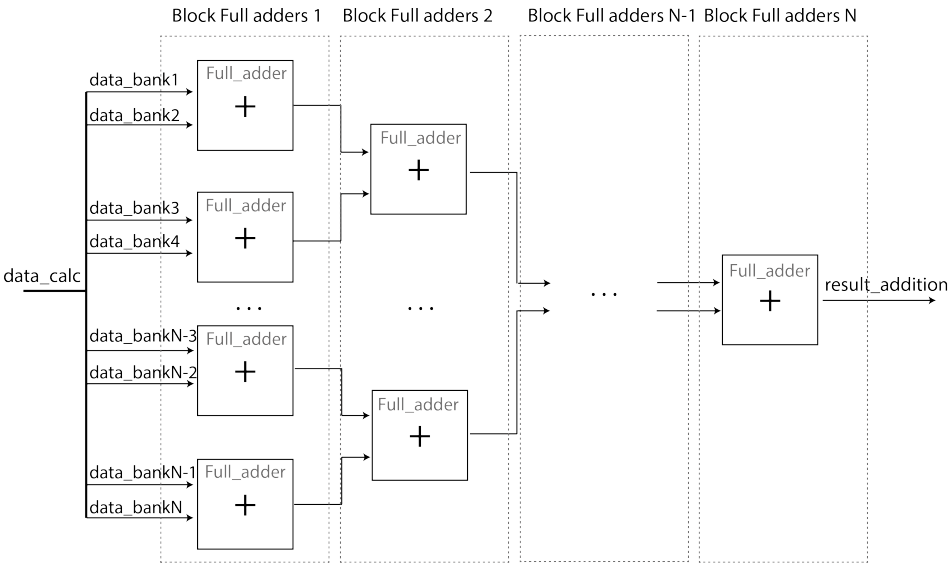


Figure 5.10: Adder Block - block diagram.

of vectors and each of them containing all the bits that share the same bit position in the word.

Once the vectors with the right data have been generated, they are provided as an input to the next block named 'Adder Generator'. This block is responsible for doing the addition of all provided vectors and has an output bus which contains the result of all the additions concatenated.

The result from the block depends on the operation to perform. If a writing operation must be done, the output signal from this block "out_addition_bits" is the result obtained from the 'Adder Generator' ("result_addition). However, if it is a reading operation instead, the output is taken from the "data_read" signal. The "data_read" value is generated by comparing "data_extra_mem" to "result_addition" for each bit position of the word. Therefore if they have the same value the result is '0'; otherwise, it is '1'.

Adder Generator The Adder Generator block is shown in Figure 5.12. This block is responsible of generating all the hardware needed to perform the addition of all vectors.

It generates WL number of tree structure blocks, the same tree structure as in the previous solutions.

It is important to highlight that the adders inside the tree structure block differ from stage to stage, i.e., in the first stage, 1-bit adders are used but in the second stage, 2-bit adders are used instead and so forth. This is because in these operations the carry out needs to be taken into account (until $log_2(NB)/log_2(2)$ bits), therefore every time an addition is performed the result from that is : result from the addition plus carry out. Hence, the input for the next addition will increase one bit.

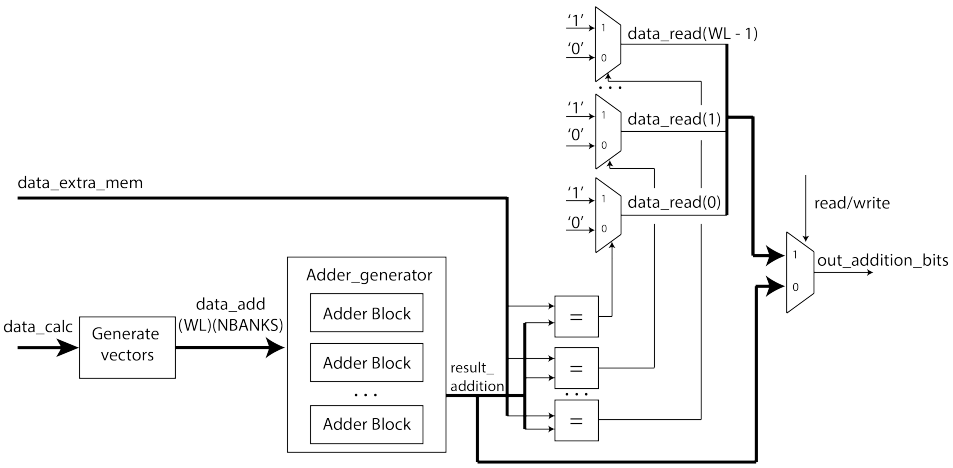


Figure 5.11: Bit Addition Block - block diagram.

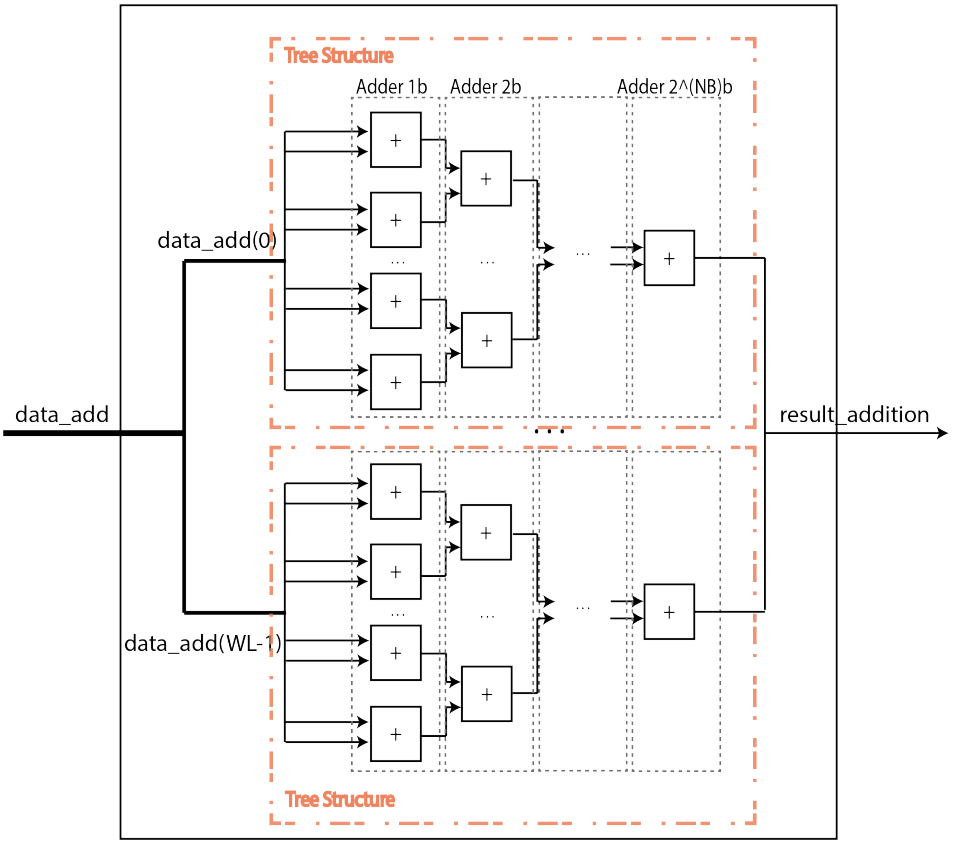


Figure 5.12: Adder Generator - block diagram.

5.2 Behavioural HDL Test

In this work, the behavioural test is one of the essential parts, because different solutions (XOR, WA, BA) and each of them for different cases (Word Length - Number of banks - Number of words), and for all the phases in the flow (Pre-synthesis, Post-synthesis and Post-PnR) must be carefully tested. As a result, every implementation done must be verified in each phase of the flow (Pre-synthesis, Post-synthesis and Post-PnR).

Figure 5.13 shows the scenario created in this project for behavioral testing. It has three different parts: (I) Generation of the stimulus (Python), which is dependent on the generic parameters (NB, NR, and WL) and the number of tests configured by the user, (II) Algorithm programmed in Python, and (III) TCL Testbench to communicate with the RTL designs.

The stimulus file generated at the first stage is used as an input for the Python algorithm and the TCL Testbench. The output from both solutions are compared and the result is be given based on that.

It is important to highlight that one scenario per solution has been generated with its own Python algorithm.

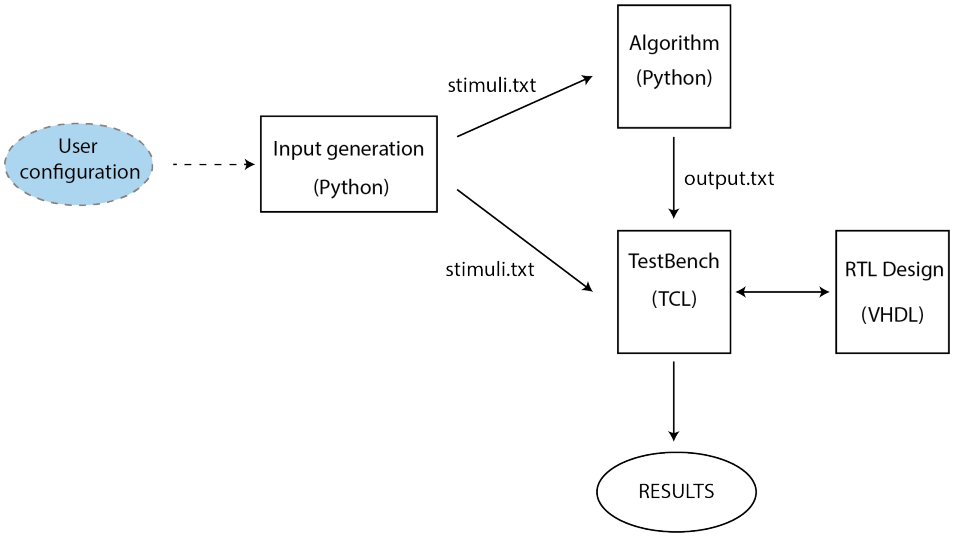


Figure 5.13: Behavioural Testing Scenario.

5.3 BIST

In this project, a simple BIST has been designed and integrated in the Word Addition solution. In general terms, the main purpose of this block is to write four different pre-defined patterns in the memory banks and verify the result. The patterns used in this BIST are:

- All Zeros ("0000").
- All Ones ("1111").
- Checkboard ("0101").
- Reverse Checkboard ("1010").

This BIST works as follows: firstly, an enable signal must be activated to start using it ("enable_n = 0"). Once the enable signal has been triggered, the FSM of the BIST starts and cannot be stopped. It begins writing into all rows of the memory banks, but switching from one pattern to the other, e.g., if it starts writing the first row of all memory banks with the pattern 'ZEROS' then, the pattern written in the second row is 'ONES', and so forth, until all the rows are written with one of the patterns.

After that, the read operation begins. In this operation, it reads row by row and verifies if the values read are correct. It continues this process until an error is found and stops the memory testing and reports "error found" or all the rows have been verified but testing is not finished and reports "no errors found and continuing", or the testing is finished without any errors and reports "no errors found".

At this point, if the testing needs to continue, it switches to the write operation, but in this case it starts writing the first row with the next pre-established pattern, e.g., if the previous pattern written in the first row was 'ZEROS', the new pattern written is 'ONES', and the same happens for all the following rows. This procedure continues until all patterns have been written in all the rows.

5.3.1 Integration BIST with the Memory Controller

The BIST module has been integrated as a block in the memory controller design. At the top module two new signals have been added: "enable_n" - Enable/Disable the memory test and "data_out_bist" - Gives information about the memory test (Error - No error - No error and not finished). Moreover, more control logic to select either the output data from the memory controller itself or from the BIST has been added.

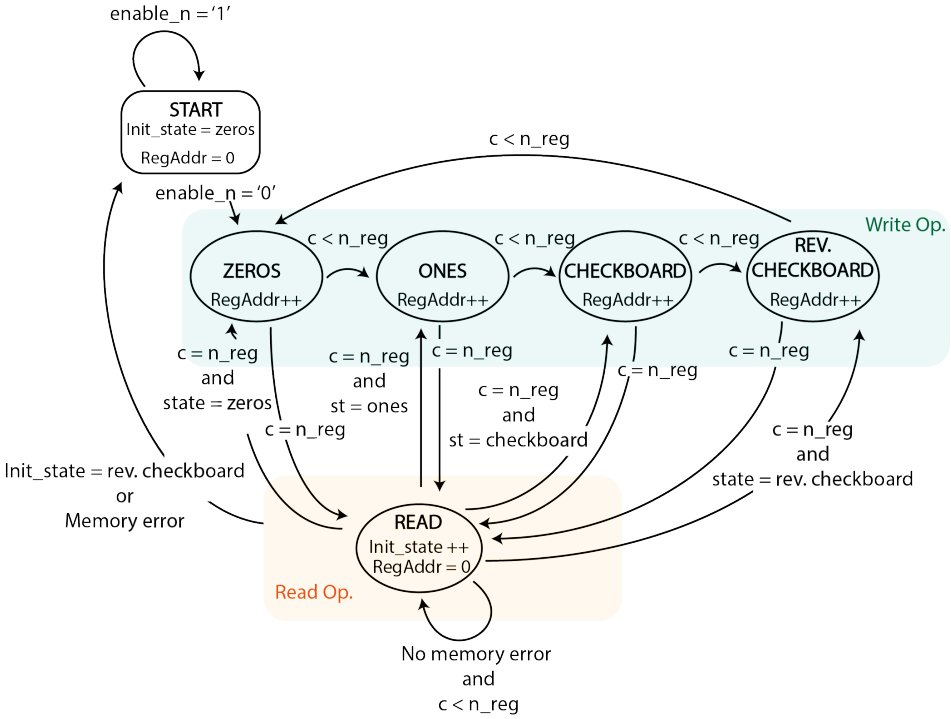


Figure 5.14: FSM BIST.

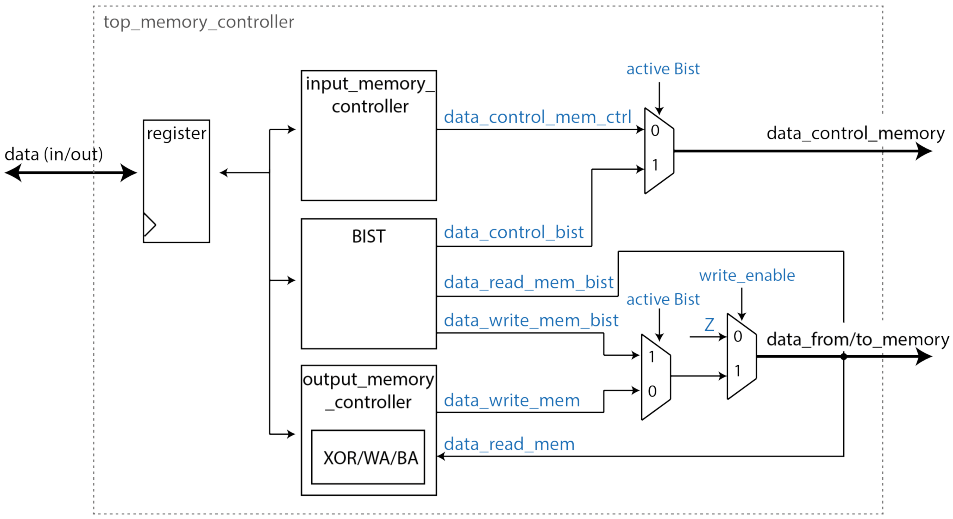


Figure 5.15: Top Memory Controller with the BIST.

Chapter 6
Results

This chapter summarizes the results taken from all the designs developed for this project.
The chapter has been divided in two main parts:

- Synthesis Results.
- Place and Route Results.

It is important to have in mind that all solutions have been implemented using a clock period T=3ns, if not mentioned otherwise.

6.1 Synthesis Results

6.1.1 Memory controller solutions

In this section, it is described how the post-synthesis cell area per bit varies for each solution implemented between each generic parameter (WL, NR, and NB) of the design. The three solutions are compared at the same time since they experience a similar area increment behavior.

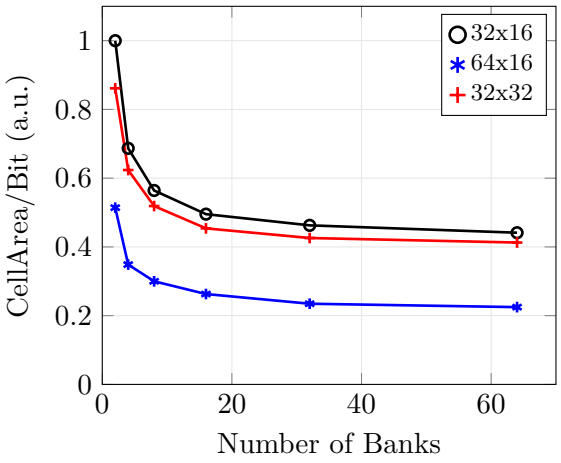
The methods are compared by analyzing the average amount of cell area required to store one bit, which is referred to as 'Cell Area/Bit'.

The memory controller areas are obtained using the following configurations:

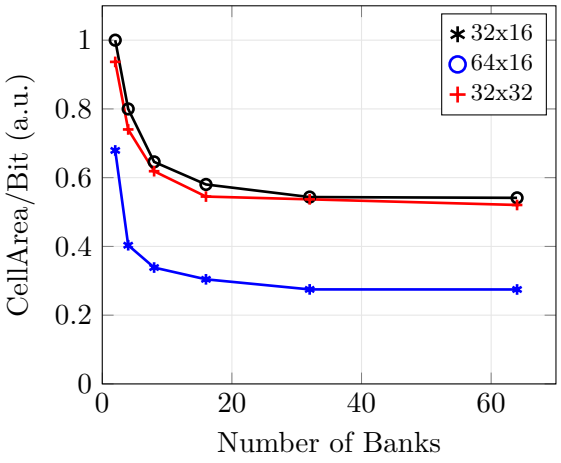
- 3 different sizes of memory banks: 32×16, 32×32 and 64×16 (NR×WL).
- 7 different values of NB: 2, 4, 8, 16, 32, and 64.

For each NR×WL configuration, the area has been obtained for all the NB values.

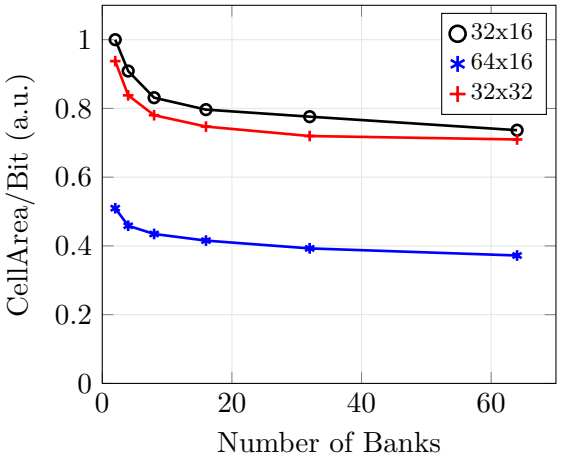
Figure 6.1 shows the three graphs, one for each solution, and in each of them, three different curves that represent the different memory bank configuration. The tendency of all three curves is logarithmic, having the 8-NB value as the border where the cell area per bit saturates and becomes a steady value. Therefore, for low NB values, the memory controller cell area per bit is not as good as for bigger NB values, having the turning point at 8 NB.



(a) XOR Solution



(b) Word Addition Solution



(c) Bit Addition Solution

Figure 6.1: 'CellArea/Bit (a.u.)' result for different configurations.

The memory bank configuration '32×16', which corresponds to the black curve in all three graphs, is used as a reference to compare the different generic parameters.

To compare the parameter WL, the reference curve and the memory bank configuration '32×32' (red-line in all three graphs) are used. The shape of the curve for both configurations is the same, but the red-line is shifted towards the bottom, meaning a reduction of Cell Area/Bit. Then, increasing the memory bank size by incrementing the WL, the memory controller cell area per bit improves slightly.

The parameter NR is compared using the reference curve and the blue curve which corresponds to the last memory bank configuration '64×16'. The curve corresponding to the '64×16' memory bank configuration is also shifted towards the bottom but more abruptly than in the previous case. This means that increasing the memory size by incrementing NR, reduces the overall area share of the memory controller, i.e., the area cost of the memory controller/bit is reduced.

To sum up, increasing the total memory size by incrementing the NB can be approximated to a logarithmic function, having the NB = 8 as the turning point where the 'Cell Area/Bit' becomes a stable value. Furthermore, the increase of the memory bank size by changing the WL or the NR parameters makes the memory controller cell area per bit to decrease as well. Although the total memory size is the same for both configurations, '32×32' and '64×16', increasing the NR instead of the WL decreases the Cell Area/Bit more significantly.

6.1.2 Comparison between memory controller solutions

In this subsection, the three memory controller solutions are compared to each other.

Figure 6.2 shows 'Cell Area/Bit' for different memory sizes in each memory controller solution. In general terms, the XOR solution is the most area efficient solution, followed by the Word Addition solution (28.5% bigger than XOR solution). Finally, bit addition solution has the lowest area efficiency among three approaches (39% bigger than XOR solution).

It is important to highlight that for low NB the Bit Addition solution cell area per bit becomes smaller than the Word Addition solution and close to XOR solution. This is due to the fact that lower NBs result in lower logic complexity, and consequently, smaller area for the memory controller. Moreover, the Bit Addition solution when NB = 2 has the same behavior as the XOR solution. This is because the result of the addition of two bits without the carry-out is the same as the result of the XOR.

Figure 6.3 shows the propagation delay of each design for the longest combinational path. Looking at the figure, all three solutions have monotonic behavior, increasing the propagation delay as the NB increments. XOR is the solution with the lower combinational time, followed by the Bit Addition solution. The Word Addition solution has the worst timing properties.

It is important to highlight that for low NB values, the maximum combinational time for the Bit Addition solution is similar to the XOR solution. The same reason as previously explained in the area comparison applies in this case.

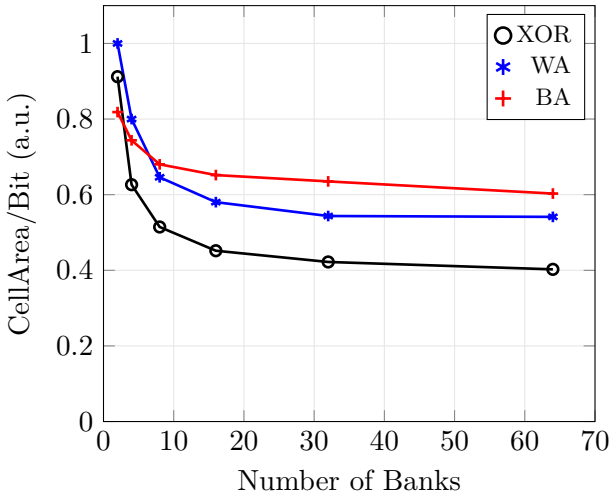


Figure 6.2: Area comparison between memory controller solutions for 32×16 configuration. XOR solution is the most area efficient.

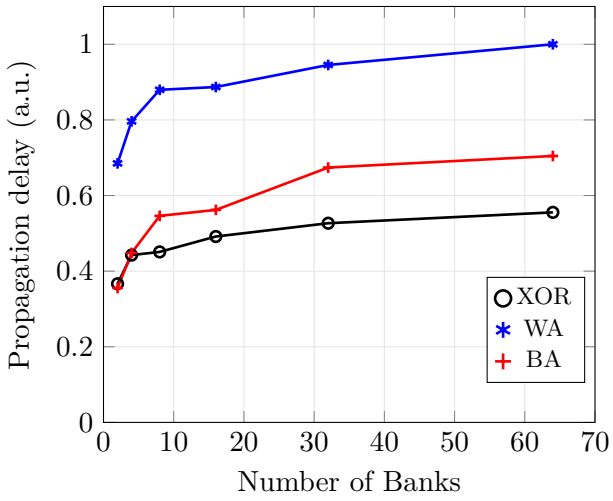


Figure 6.3: Time comparison between memory controller solutions (XOR, WA and BA) for 32×16×16B configuration. XOR solution has the lowest combinational time.

The Word Addition solution has the longest combinational path since it has to perform the addition of larger numbers than the Bit Addition solution.

6.1.3 Comparison Memory Block to the TP-SRAM solutions

This section compares the entire memory block solution to the TP-SRAM solution. The entire memory block solution is comprised by the memory controller and the SP-SRAM banks. The memory configuration used for this section is the following: 32×16×16B (NR×WL×NB).

Figure 6.4 shows the portion occupied of the total area by each part of the memory block solution: memory controller and the SP-SRAM banks for the XOR solution. The percentage of the area assigned to the memory controller decreases when larger memories are used.

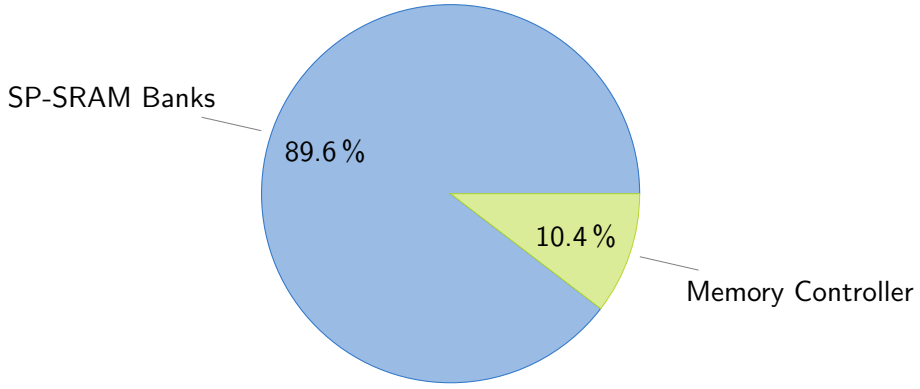


Figure 6.4: Area occupied by each part of the memory block for the XOR solution and using 32×16 configuration.

Figure 6.5 shows the ' Area/Bit' for four different solutions:

- XOR Memory Controller solution + SP-SRAM banks.
- Word Addition Memory Controller solution + SP-SRAM banks.
- Bit Addition Memory Controller solution + SP-SRAM banks.
- TP-SRAM bank solution.

In all cases, the solution with the memory controller and SP-SRAM banks are more area efficient (for example the area in XOR solution is reduced by 45%, using 32×16×16B configuration) than the conventional TP-SRAM solution and it increases when larger memories are used.

Furthermore, it is important to notice the small area difference between solutions, especially the XOR and Word Addition solutions. As the memory controller area is a small part of the total memory area, the difference between them when the entire memory block is taken into account is less than 2%.

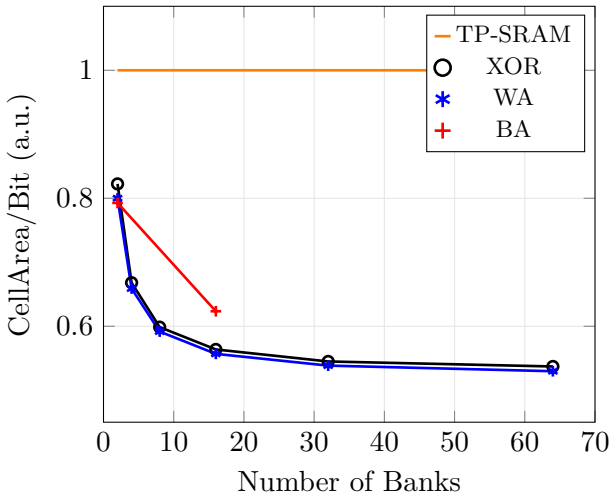


Figure 6.5: Area comparison between entire memory block solutions (XOR, WA and BA) to Two Port solution. All three solutions are smaller than TP-SRAM solution.

6.2 Place and Route Results (PnR)

The results from the PnR are done for one configuration: 32×16×16B. The reason for that is the amount of work that the PnR requires for each setting.

Figure 6.6 shows PnR after nano-route using 8 metal layers. The figure corresponds to the Word Addition memory controller solution for 32×16×16B configuration. As the memory controller is part of the memory block, which consists of the memory controller and the memory banks, the power rings, and pin placement are not final. For the same reason, power strips, and the IO pads are not utilized in the PnR.

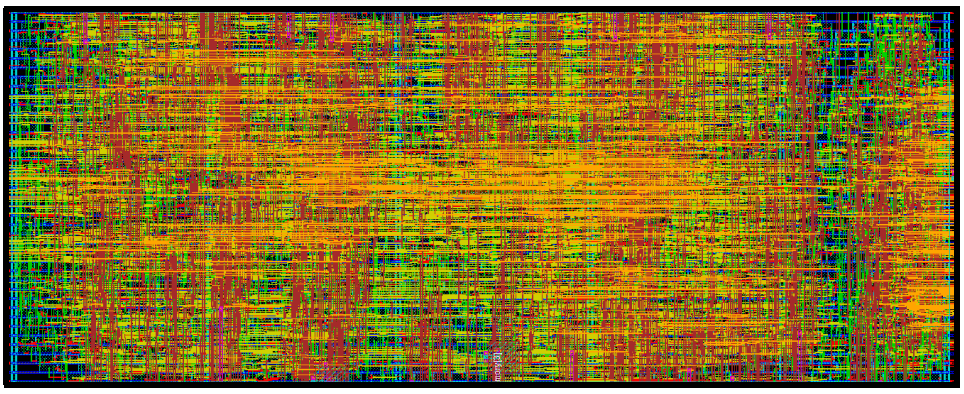


Figure 6.6: Core of PnR after nano-route for Word Addition memory controller solution, 32×16×16B configuration.

6.2.1 Memory controller

The area after the PnR with an utilization of 68% increases significantly compared to the post-synthesis results. However, the area increment of the memory block solution is not more than 4%. Therefore, the designs after the PnR are still area efficient compare to the conventional TP-SRAM solution.

Table 6.1 shows the maximum frequency of the memory controller in each solution after the PnR. This frequency is much higher than the typical SRAM frequencies (800MHz-1.1GHz) for 28nm.

Solution	**Max. frequency [GHz]**
XOR	2.5
Word Addition	1.25
Bit Addition	2

Table 6.1: Maximum frequency results for each memory controller solution after PnR for 32×16×16B configuration.

Figure 6.7 shows the power results of the memory controller for each solution after the PnR. It is important to mention that the power from the memory controller solutions are obtained from Encounter tool, and the power from the memory banks are obtained directly from the datasheet.

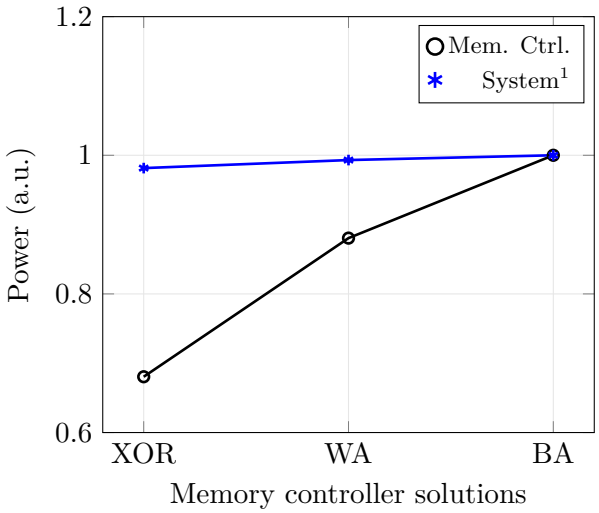


Figure 6.7: Power results from PnR for 32×16×16B configuration.

The power difference of the memory controller between solutions is significant, yet when the memory block solution is considered it is almost negligible. Moreover, all three solutions reduce the power consumption by at least 35.7% compared to

the conventional two-port memory solution[2].

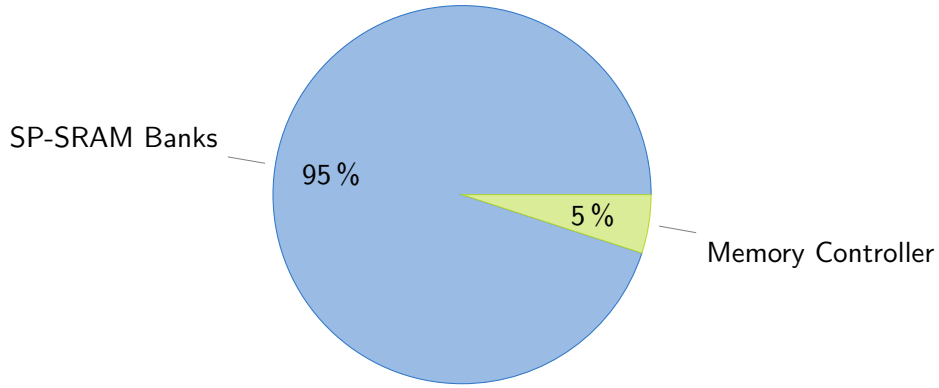


Figure 6.8: Average of the three solutions power consumption for each part of the memory block solution using 32×16×16B configuration.

6.2.2 BIST

Table 6.2 shows the results of the Word Addition solutions with BIST block integrated.

The addition of the BIST block makes the memory controller to increase 30% which corresponds to an increment of the 4.7% in the memory block solution (Memory controller, BIST, and SP-SRAM Banks). Furthermore, the solution studied in this subsection is 35% smaller than conventional TP-SRAM solution.

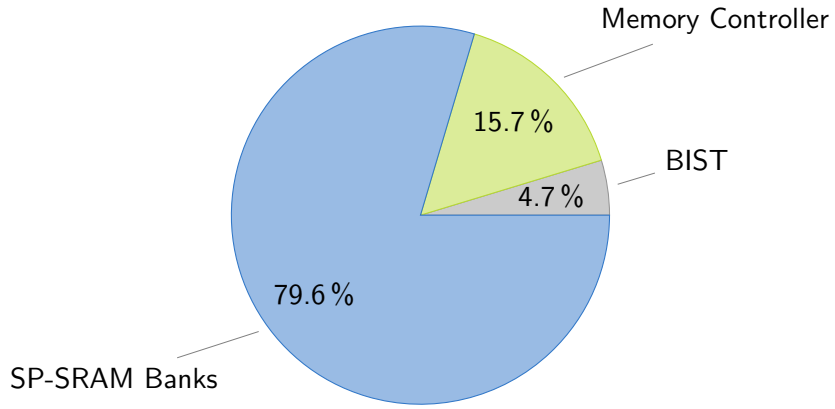
The slack is not being affected because the critical path is in the addition block.

The memory controller power is increased by 66%. This is mainly caused by the sequential part that the BIST is adding. Although there is a significant increment in the memory controller power due to the BIST, the resulting overhead considering the memory block solution corresponds to 3.3%. As a conclusion, the solution studied in this subsection reduces the power consumption by 34% compare to the conventional Two-Port memory solution.

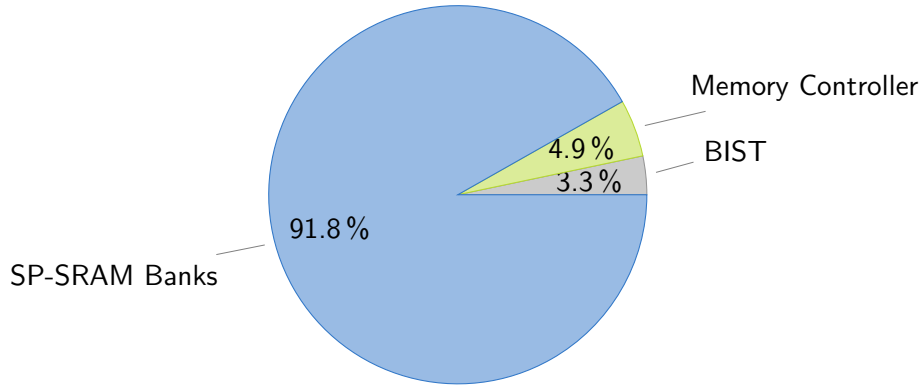
[2]TP-SRAM power consumption data has been deduced by extrapolating the power data from SP-SRAM and DP-SRAM

Number of gates Mem. Ctrl.	Number of gates Mem. Ctrl. with BIST	Slack [a.u.]	Total Power Mem. Ctrl. with BIST [a.u.]
2934	3814	0.8111	1.0266

Table 6.2: Results from place and route using Word Addition solution including the BIST module into it and using 32×16×16B configuration.



(a) Area occupied by each part of the system



(b) Power consumed by each part of the system

Figure 6.9: System results compressed by Word Addition with BIST solution using 32×16-16B configuration - Place and Route.

Chapter 7
Conclusion

In this project, three different memory controller algorithms are proposed in order to increase the throughput of a SRAM in terms of performing a two-read operation or one-write operation at a time. All of them have been analyzed in terms of area, power, and speed.

XOR solution has demonstrated to be the most area and power efficient compared to the other two solutions exposed. Even though, the area overhead of the memory controller for the other two solutions with regard to the XOR solution is 28.5% and 39% respectively, when the entire memory block solution is taken into account, is not more than 2% in the Word Addition Solution (WA) and 9% in the Bit Addition Solution (BA).

From the power consumption perspective, the Word Addition solution (WA) has an overhead of 30%, and the Bit Addition solution (BA) has an overhead of 47% with regard to the XOR solution. The power consumption penalty when considering the whole memory block solution is negligible.

Additionally, the maximum frequency, using 32×16×16B configuration, for each solution, is 2.5GHz for XOR solution, 1.25GHz for Word Addition solution (WA), and 2GHz for Bit Addition solution (BA). This frequency is much higher than the typical SRAM frequencies (800MHz-1.1GHz) for 28nm.

To sum up, the three solutions can reduce the area compared to the conventional Two-Port SRAM solution by up to 39% (cell area after synthesis and using 32×16×16B configuration). Additionally, and similar to the area, the three solutions reduce the power consumption compared to the conventional two-port SRAM solution by 36% (power consumption average of the three solutions using 32×16×16B configuration).

Furthermore, a BIST has been integrated into the Word Addition solution and for 32×16×16B configuration. The penalty for adding this module is 30% in the memory controller area, which corresponds to an overhead of 3% in the entire memory block solution. The slack is not being affected by this module as the longest combinational path is due to the addition block. Regarding the penalty in power, the memory controller experiences an increment of 66% due to the addition of the sequential logic. However, the overall overhead considering the entire memory block solution (memory controller with BIST and single-port SRAM banks) is 3.3%.

Overall, the entire memory block solution with the integration of the BIST into

the memory controller reduces the area by 35% compared to the two-port SRAM solution. The slack is not being modified with its incorporation. Regarding the power consumption, the penalty is 3.3% of the entire memory block solution but the power consumption also reduces by 34% compared to the conventional two-port SRAM solution.

Figure 7.1 sums up the percentage of improvement for each memory controller solution compared to the conventional two-port SRAM solution.

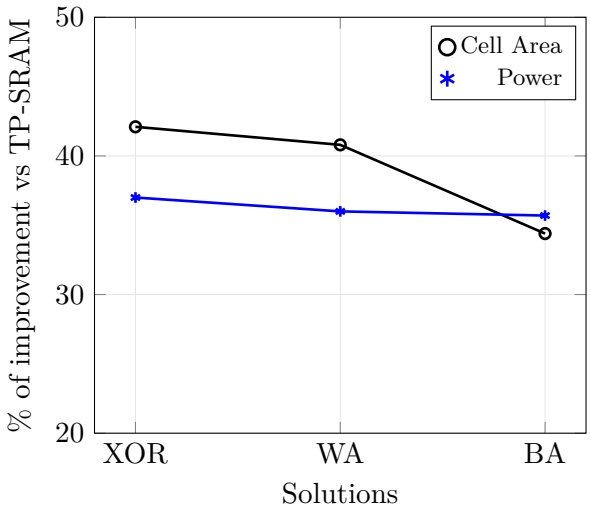


Figure 7.1: Comparison of the entire memory block solution with the different memory controller solutions to the conventional two-port SRAM solution for 32×16-16B configuration.

7.1 Future Work

- Reduction of the area by reducing the combinational logic.
- Exploring new algorithms that allows more than two-read operation.
- Exploring new algorithms to increase the number of writes per operation.
- In Section 3.4 we concluded that compression, error detection, and hash functions were not suitable for this thesis purposes. However, a method that could ensure the codification of the data with a deterministic resulting memory size could be a potential method. Obviously, the memory required size should also be inferior than simply duplicating the data.

References

[1] [Online]. Available: http://electroiq.com/blog/2014/02/the-most-expensive-sram-in-the-world-2-0/

[2] You Li and Xiangqing He, *"A novel area-efficient and full current-mode dual-port SRAM,"*2008 International Conference on Communications, Circuits and Systems, Fujian, 2008, pp. 1079-1082.[Online]. Available: http://ieeexplore.ieee.org.ludwig.lub.lu.se/stamp/stamp.jsp?tp=&arnumber=4657955&isnumber=4657700

[3] Kaur Ramandeep, Fell Alexander, Rawat Harsh, *"A 6T SRAM cell based pipelined 2R/1W memory design using 28nm UTBB-FDSOI"* 2015, 2015 28Th IEEE International System-On-Chip Conference (SOCC), System-On-Chip Conference (SOCC), 2015 28Th IEEE International, p. 310, IEEE Xplore Digital Library.

[4] Yokoyama Yoshisato, Ishii Yuichiro, Okuda Haruyuki, Nii Koji, *"A dynamic power reduction in synchronous 2RW 8T dual-port SRAM by adjusting word-line pulse timing with same/different row access mode",* 2017, 2017 IEEE Asian Solid-State Circuits Conference (A-SSCC), Solid-State Circuits Conference (A-SSCC), 2017 IEEE Asian, p. 13, IEEE Xplore Digital Library.

[5] F. Bai et al., *"A two-port SRAM using a single-port cell array with a self-timed write-after-read control scheme to save 47% area & 63% standby power,"* 2017 IEEE 12th International Conference on ASIC (ASICON), Guiyang, 2017, pp. 426-428.

[6] *"Understanding Asynchronous Dual-Port RAMs,"* Cypress Semiconductor Corporation.[Online]. Available: http://65xx.unet.bz/ds/an1048.pdf

[7] Mohammadi Babak, *"Ultra-low Power Design Approaches in Memories and Assist Techniques,"* Lund Unversity, 2017-06-16 pp 28.

[8] Mohammad Hammoud, Sangyeun Cho, Rami Melhem, *"C-AMTE: A location mechanism for flexible cache management in chip multiprocessors"*, Journal of Parallel and Distributed Computing 2011, pp. 889-896. [Online], Available: http://www.sciencedirect.com/science/article/pii/S0743731510002418

[9] *"Introductory Digital Systems Laboratory"* MIT, 2006. [Online], Available: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-111-introductory-digital-systems-laboratory-spring-2006/lecture-notes/l8_9_arithmetic.pd

[10] Singhee, A, & Rutenbar, R 2010, *Extreme Statistics In Nanoscale Memory Design*. [Elektronisk Resurs], n.p.: Boston, MA : Springer US, 2010., Library catalogue (Lovisa).

[11] G. Anandharaj and R. Anitha, *"A Distributed Cache Management Architecture for Mobile Computing Environments,"* 2009 IEEE International Advance Computing Conference, Patiala, 2009, pp. 642-648. doi: 10.1109/IADCC.2009.480908

[12] A. P. C. M. Michael Price, James Glass, *"A 6 mW, 5K-Word Real-Time Speech Recognizer Using WFST models,"* in 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014, pp. 454–456.

[13] M. Gautschi, D. Rossi, and L. Benini, *"Customizing an open source processor to fit in an ultra-low power cluster with a shared L1 memory,"* in Proceedings of the 24th edition of the great lakes symposium on VLSI, 2014, pp. 87– 88. [Online]. Available: `http://dl.acm.org/citation.cfm?id=2591569`

[14] P. P. Damodaran, S. Wallentowitz and A. Herkersdorf, "Distributed cooperative shared last-level caching in tiled multiprocessor system on chip", 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2014, pp. 1-4. [Online]. Available: `https://ieeexplore.ieee.org/document/6800297/`

[15] Lenis, J, and Senar, M 2017, *"A performance comparison of data and memory allocation strategies for sequence aligners on NUMA architectures"*, Cluster Computing, 20, 3, pp. 1909-1924, Inspec, EBSCOhost, viewed 18 May 2018.

[16] Sundar Iyer and Shang-Tse Chuang, *"High speed memory systems and methods for designing hierarchical memory systems"*, United States Patent Application Publication, Pub. No.: US 2011/0022791 A1, Pub. Date: Jan. 27,2011.

[17] M. B. Lin and Y. Y. Chang, *"A New Architecture of a Two-Stage Lossless Data Compression and Decompression Algorithm,"* in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 17, no. 9, pp. 1297-1303, Sept. 2009. [Online]. Available: `http://ieeexplore.ieee.org.ludwig.lub.lu.se/stamp/stamp.jsp?tp=&arnumber=4801589&isnumber=5208586`

[18] T. Nakamura, K. Iwai, T. Matsubara and T. Kurokawa, *"Implementation of Hash Function Generator on Schematic to Program Translator(SPT),"* 2017 Fifth International Symposium on Computing and Networking (CANDAR), Aomori, 2017, pp. 469-474.

[19] M. Turčaník, *"Hash function generation based on neural networks and chaotic maps,"* 2017 Communication and Information Technologies (KIT), Vysoke Tatry, 2017, pp. 1-5.

[20] I. Amro, *"Speech Compression Exploiting Hamming Correction Code Compressor,"* 2013 Fifth International Conference on Computational Intelligence, Communication Systems and Networks, Madrid, 2013, pp. 234-238.

[21] [Online]. Available: `https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_hdl_overview.htm`

[22] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolić, *"Digital integrated circuits: a design perspective,"* Chennai, Delhi: Pearson, 2003, pp.623-721.

Printed by Tryckeriet i E-huset, Lund 2018

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2018-668
http://www.eit.lth.se