High Bandwidth Distributive In-logic Memory

WENPENG SONG
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

High Bandwidth Distributive In-logic Memory

Wenpeng Song
we3543so-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Hemanth Prabhu, Erik Larsson

Examiner: Pietro Andreani

March 5, 2020

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Static Random Access Memories (SRAM) are widely used for on-chip caches in modern computing systems, and occupy a significant portion of the chip's area and power consumption. The traditional "von-Neumann" architecture which been used in modern computing systems also leads to a limitation of data transfer rate between memory and processing unit. These issues are more pronounced in data-intensive applications.

This thesis presents an implementation and evaluation of a new approach for designing SRAM which can be used similarly as "in/near-memory" architecture. This approach reduces the peripheral circuit of the small storage memory by preserving the memory cell array in SRAM as a custom cell and replacing the remaining modules with digital circuits. The SRAM designed with this approach needs a smaller area than conventional SRAM if the storage capacity of 2K or less. At the same time, power consumption keeps at the same level or even less. Therefore, it is more efficient to replace large memory with multiple new SRAMs to avoid the limitation of bandwidth. Furthermore, the SRAM designed with this approach can be distributed in a common logic design like a standard cell. This reduces the data transmission path, which further increases speed and reduces power consumption. Moreover, this approach is not algorithm specified and can be used in any algorithm without extra modification.

Acknowledgments

I would like to express my thanks to my supervisor Hemanth Prabhu (Xenergic AB), professor Erik Larsson (Lund University). Thanks a lot for their continuous help and precious support throughout this project period. I would also like to thank Joachim Rodrigues (Lund University) for his valuable advice during this project.

Special thanks to Babak Mohammadi, CEO of Xenergic AB, for his comprehensive support during this time. And I am also grateful for the help during my Master's studies from Helene von Wachenfelt, International Master's Coordinator at Lund University.

At last but not least, I would like to thanks all my friends, colleagues and my family for their supporting during this period.

Popular Science Summary

The project presented in this report is an implementation and evaluation of a new method to design the memory which can be used in chips. The implemented memory achieves a higher density, lower power consumption compared to the original memory. And these advantages are valid for the memory with a storage capacity of 2kb or less.

This memory can be used in designs of Machine Learning (ML), Artificial Intelligence (AI) or other designs that required heave access to memory. The designs using this memory will have a smaller size and less power consumption than original memory. This means these designs could be used in smaller devices or have more powerful functionality with the same device size. And it can also support these devices to be used longer than usual under the same amount of power.

These advantages will result in lower manufacturing costs and better user experience for equivalent devices. At the same time, the devices with ML and AI can become more popular and more versatile. Moreover, this memory is easy to use and can be modified/generated by digital designers.

Table of Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

ASIC Application Specific Integrated Circuit

BL Bit Line

CNN Convolutional Neural Network

DRC Design Rule Checking

EDA Electronic Design Automation

HDL Hardware Description Language

LEF Library Exchange Format

LIB Liberty Timing File

ML Machine Learning

PNR Place and Route

RTL Register Transfer Level

SoC System-on-a-Chip

SRAM Static Random Access Memory

WL Word Line

Chapter 1
Introduction

Static Random Access Memories (SRAM) are the most popular type of embedded memories in modern SoCs and widely used for on-chip caches in modern microprocessors. SRAMs occupy a significant portion of the chip's area [1] and power consumption [2] in modern SoCs, which becomes a bottleneck of modern computing systems. The modern computing systems normally using the traditional "von-Neumann" architecture which the memory and the processor are physically separate. This architecture also leads to a limited bandwidth for data transfer based on the memory IO. These issues are more pronounced in data-intensive applications (like AI and ML) as they need bigger memories, which lead to long-distance data movement that requires more time and energy [3][4].

One of the promising approaches for improvement is the concept of "in/near-memory" computation that the computation can be done inside or close to the memory array. The "in/near-memory" architectures have much fewer data transferred over long distances than conventional architecture. And the memory access is not limited by it I/O which improves the throughput. However, these approaches typically need modification on the memory structure based on the computation algorithms. As an example, the Conv-SRAM [5] was designed for the computation for the convolutional layer of CNN, which embedded analog computations in the SRAM bit-cells. These designs require the work of algorithms modification and mixed with SRAM structure. Also, the modified SRAM can only be used in these specified computations.

The "in/near-memory" architectures are very efficient in speed and power consumption, but the limitation of the algorithm makes more development cost. So, we want to make a general SRAM that can be used for "in/near-memory" architectures without algorithms limitations. As other approaches like Conv-SRAM normally make the modifications based on the SRAM with six transistor-based (6T) bit-cell since it has a small cell area and compact lithography-friendly layout [6] which leading to high-density memory arrays. Our approach is based on the 6T SRAM bit-cell as well, but digital design around the memory arrays instead of analog modification to make it general to use and easy to compact with other logic designs to work as "in/near-memory" architecture.

In this thesis, We will first introduce more about how our approach can

work as “in/near memory” and the difference between our approach, others approach, and conventional SRAM. After that, We will introduce the basic structure and functionality of conventional 6T SRAM to explain how we implement this approach. Then follow the general design flow to test and find out what other work needs to be implemented for our approach and how we implemented them. In the end, We will show some evaluation of our approach.

"Von-Neumann" and “In/Near-memory” Architectures

"Von-Neumann" architecture is a computer architecture that includes ALU, Control Unit, Memory, Input and Output mechanisms which is the basic architecture of the modern computing system. The separated memory and CPU in this architecture leads to a limitation of the data transfer rate between them. Normally, the data transfer rate between memory and CPU is very low compared to the workload of the CPU. But, in some situations that required to perform small processing on large amounts of data (AI, ML), this limits the CPU process speed. This bottleneck becomes more of a problem since the CPU speed and memory size increased much faster than the data transfer rate. And this is more pronounced in data-intensive applications as mentioned before since they require big memories which leads to longer data transfer paths. Besides this increasing gap between CPU speeds and memory access times (“memory wall” [7]), the increasing power dissipation also becomes a problem. There is much research on associative memory circuits and alternative computing architectures. [7][8]

The system energy and data transfer rate are limited by memory size and I/O bandwidth. The concept of “in/near-memory” is one of the promising approaches to circumvent this limitation. The “in/near-memory” architecture brings the memory closed to or among the processing unit which decreases the data transfer path. And this can even avoid the bandwidth limitation in some approaches that embedded the computation with memory array directly. As mentioned before, these “in/near-memory” approaches normally based on the modification of the desired algorithm. They can access the memory array directly during the computation to avoid the I/O bandwidth limitation. But, they can only work for desired computation.

Register-heavy solutions is another one of the approaches to circumvent the limitation of “Von-Neumann” architecture. As an example, the Graphical Processing Units (GPU) includes a very large number of registers files that allow high parallel computing. The multi-port memories can also help. But, the parallelization and high throughput of the data leads to high power consumption [3].

Our approach

The current bottleneck of computing architecture (from memory side) is caused by the increased memory size and the I/O limitation. The increased memory size increases the data path inside the memory, locating time, and power consumption per access. This further increases the "memory wall". The I/O limited the parallelization of the access which limited the computation bandwidth. Multi-port memories can make some help, but the computation bandwidth expectation grows much more as the AL and ML.

What if using multiple small memories instead of these limited number of large memories in the computing architecture? The parallelization won't be any problem in this situation. But, as we know, small memory requires much more area than large memory for the same amount of stored bit. So, the optimization of physical size for small memory is required for this idea. Furthermore, the distance between the processing unit and memories in conventional architecture still influences the datapath. So, we want to bring those small memories into the processing unit, like register files, to reduce the datapath.

Based on the previous information, our approach combined the idea of the "in/near-memory" and the register-heavy concept. This approach splits the memory into small memories and then distributed them to the processing unit or other logic. Different from computing in-memory, our approach can act as memory in-logic. The splitting of memory will solve the bandwidth problem and the distribution in the processing unit will reduce the "memory wall". The critical of this approach is the density of small memories and the method of distributing.

There is much research about optimizing memories power consumption and area like [9] and [10]. These optimizations usually require trade-offs on the price in terms of performance or area [11]. These optimizations are mainly targeting big capacity memories (1-2kb or bigger). The small capacity memories used in ASIC designs are usually standard cell registers which are not so efficient. As shown from a trade-off discussion in 65 nm [9], the standard cell latch-based memories have a density advantage in the size of 2KB and below. The discussion is focused on SRAM which is the main product in Xenergic AB. So if we rebuild the SRAM structure by standard cells and the custom cell that keeps the high-density part of SRAM, the new SRAM should have the chance to get a better area efficiently than traditional SRAM for small capacity, Since it won't have that much peripheral circuit. And to make the new approach easy for distributing, it is better to build the SRAM structure with digital logic except for the high-density part (memory array).

Chapter 2

Basic Implementation

The basic structure of a SRAM usually contains Decoder, Memory Cell Array, Sense Amplifier, and Driver as shown in Figure 2.1.

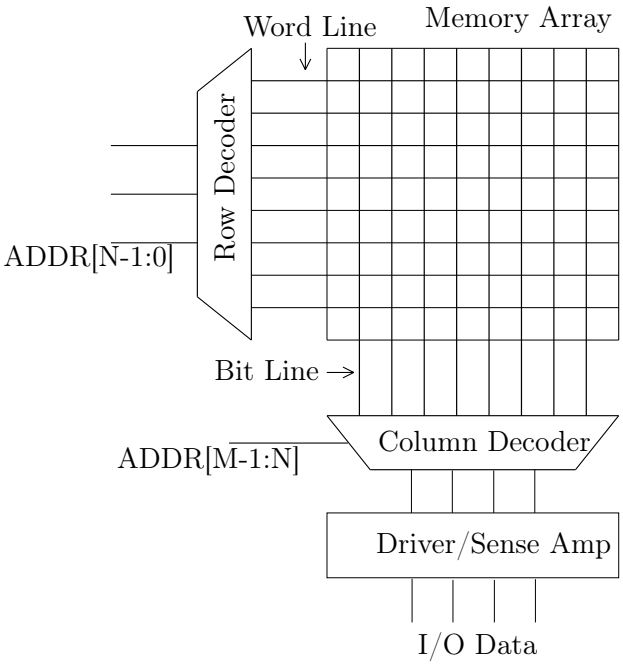

Figure 2.1: SRAM Structure

Generally, in SRAM, the memory cells array is the most difficult to optimize in density without changing the structure of the memory array. And it usually has the highest density in SRAM. So, this design will keep the memory array as a custom cell as mentioned before. Other parts were implemented by Register Transfer Level (RTL).

Memory Array

In Figure 2.2, a typical 6T SRAM cell is shown. This cell is the basic cell used to consist of the Memory array. A set of opposite states, which is 1-bit data, can be stored in each cell ($Q/\overline{Q}$). By pulling up word line, the cell state is operated differently depending on the status of the $BL/\overline{BL}$.

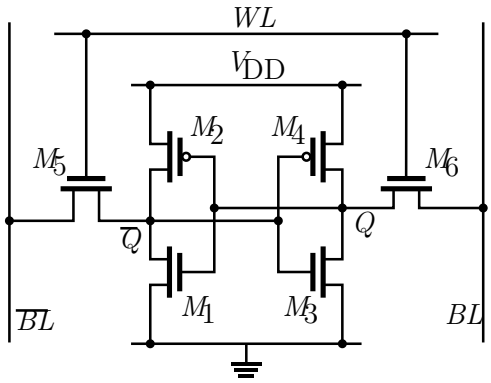

Figure 2.2: 6T SRAM Cell

Memory cells are connected to the same row by sharing the same word line. Memory cells are connected to the same column by sharing the same bit line and inverse bit line. The size of a memory array is the number of the memory cell in one row multiply the amount of the memory cell in one column.

The behavior module of the memory array in SRAM therefore has word lines, bit and inverse-bit lines as pins (Figure 2.3). The functionality of the memory array is to act as storage cells. The memory array should have different operations depends on various combinations of pins. These operations are what the behavior module needs to achieve.

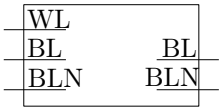

Figure 2.3: Memory Cells Array Module

The word line is the primary control of memory cells in the memory array. If the word line of a memory cell is low, nothing will happen to the memory cell, and the bit lines will not have any changes. If the word line of a memory cell is high, the memory cell could be operated on the different status of bit lines.

Read operation happened if the two bit-lines have the same level, which will drive the two bit-lines based on the cell value as shown in Figure 2.4. Write

operation happened if the two bit-lines have the opposite level, which will change the cell value based on the bit line's level as shown in Figure 2.5.

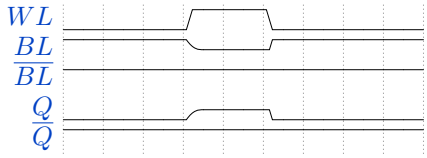

Figure 2.4: Read Operation Signal Waveform

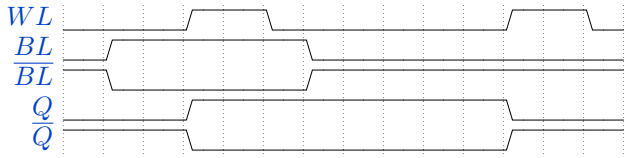

Figure 2.5: Write Operation Signal Waveform

Decoder and Sense Amplifier

The decoder module decodes the address to WL that be used in memory cells array. It needs an enable signal as input to control the timing for decoding. This module is combinational in the design, the module decodes the current address to WL when the enable signal is high. Different from the ordinary design method, here we automatically generate the RTL with direct mapping through a script that converts the address to WL. The generated decoder module will be transferred into the gate level with the help of the synthesis tool to have a better solution.

The sense amplifier module in our approach is also combinational. It gets the BL and BLN from memory cells array and compares each pair of bits to generate the output data. The output is equal to BL if and only if BL and BLN are opposite to each other. As this module is combinational, the output data only valid when WL is activating. So, a latch register is needed to latch the output data with the same enable signal as the decoder. The enable signal needs to go through several gates from different modules to generate the output data to output latch register and this causes a delay which ensures the functionality of the output latch register. The signal waveform of the output register looks like as in the Figure 2.6.

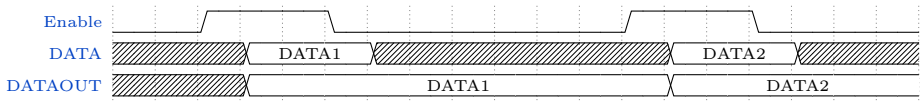

Figure 2.6: Output Register Waveform

Driver

The RTL design in this project is used to drive the macro of the memory cells array and acts as standard SRAM. Usually, A SRAM has five basic input ports (clock[CLK], address[ADDR], input data[DATAIN], write enable[WEN], chip select[CSN]) and one output port (output data[DATAOUT]). So, our design here should work as Figure 2.7

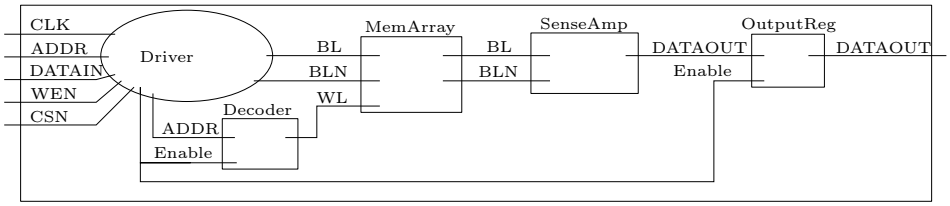

Figure 2.7: RTL Exclude Driver

The driver is the most critical part of this RTL design. It is used to get the right input signal during each clock rising edge and process them, then based on the reading or writing request generates signals to decoder and memory cells array. The write operation is easy to control, but there was a tricky part for the read operation which is the pre-charge. Pre-charge both bit lines to the same level then cut-off before the word line is active during the read operation to reduce the read delay. The cut-off during the operation will need a delayed clock signal as a control and also needs the help of tri-state buffers to perform the cut-off. So, a submodule was created for the tri-state control. This submodule is used to control the pre-charging of the BL. All BL signals connected to tri-state buffers with enable signal Tri-Ctrl. Tri-Ctrl is always high during the writing operation. In the reading operation, it is a pulse, as shown in Figure 2.8.

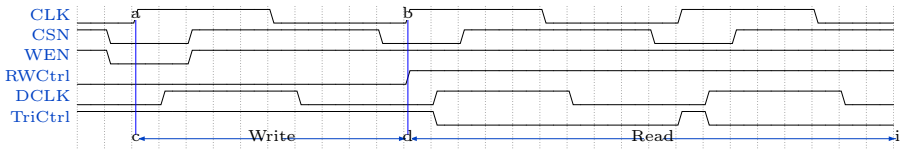

Figure 2.8: Tri-Ctrl Signal from Driver Waveform

The WL should be activated after the cut-off, which means enable signal for decoder should become high after the cut-off during the reading operation. A pulse is needed for the enable signal of the decoder. Another submodule for

the pulse generation is needed. This submodule generates two signals with different delays from the clock. One of them is the pulse signal (DCLK2) which works as a control signal of Decoder and Output Register. The other is the delayed clock (DCLK1) which functions as pre-charge control inside the Driver block.

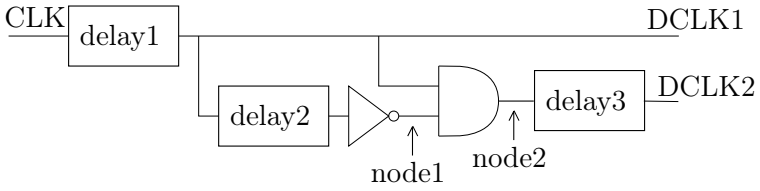

Figure 2.9: PulseGen Logic Design

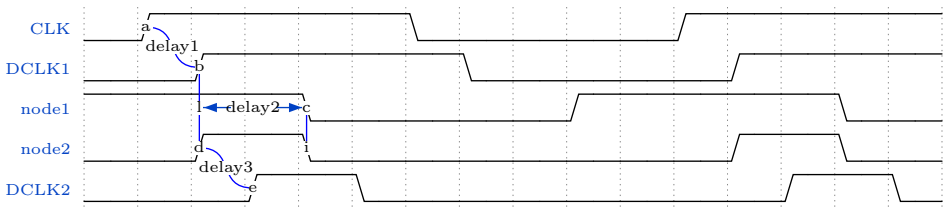

Figure 2.10: PulseGen Waveform

As shown in Figure 2.9 and Figure 2.10, delay1 is the period for pre-charge, delay2 is for WL active, delay3 is to control the WL actives after the pre-charge be cut off to avoid the half-select issue.

During the write operation, the WL (of bit-cell in macro) is active after the BL (of bit-cell in macro) has the right voltage and de-active after the process done (before the BL changes). The BL is controlled by the driver that triggers it at the rising edge of the clock. DCLK2 (rising edge) needs to be delayed (delay3) for the period of transition time from driver to the macro to make sure that the BL of bit-cell in the macro will have the voltage before WL actives. The width (delay2) of DCLK2 should be wider than the writing timing of bit-cell. During the read operation, the BL is precharged and cut off before WL is active. The delayed time (delay1) depends on the pre-charging time for the bit cell. The rising edge of DCLK1 is for cutoff the pre-charge.

RTL Main Design

The main design is shown as in Figure 2.11 after all modules be implemented. It has five input ports (CLK, ADDR, DATAIN, WEN, CSN) and one output port (DATAOUT). It includes seven blocks: pulseGen, decoder, driver, tridriver, sense amplifier, output register, MemArray. The MemArray is the behavior module for the macro from Xenergic's SRAM, and that is the only custom cell.

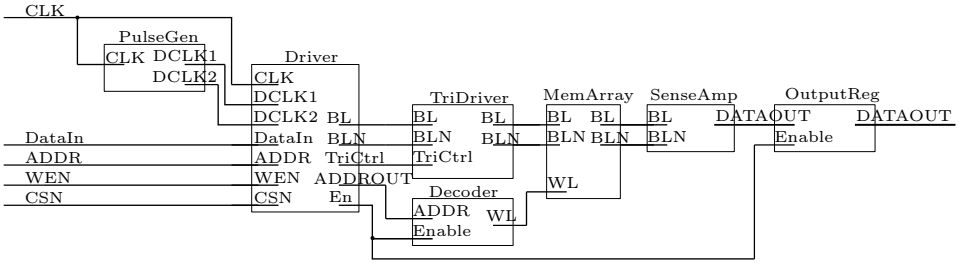

Figure 2.11: Main Design

Chapter 3

Related Implementation

Functional verification and testing are usually required after the RTL design is completed to ensure proper functionality during ASIC design flow as shown in Figure 3.1. A test bench needs to be created to check on the functionality. RTL code and test bench are simulated using HDL simulators for verification and testing. If there is a deviation, it needs to be modified in time for the subsequent steps to proceed normally.

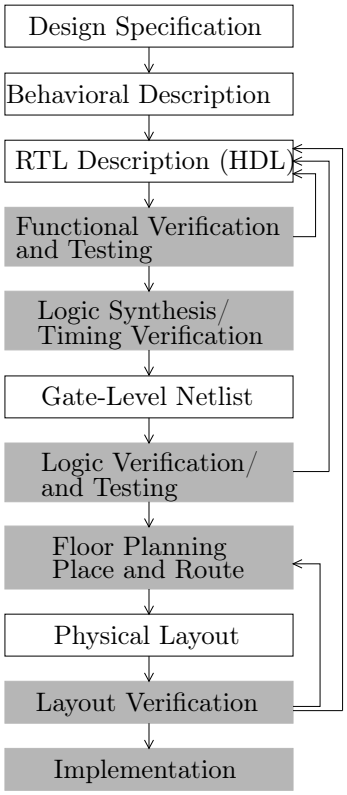

Figure 3.1: Design Flow

After that, logic synthesis which is to transfer RTL hardware description and standard cell (gate level, including macros) libraries (timing/power information) to a gate-level netlist. The resulting gate-level netlist is a complete structural description with standards cells only at the level of the design. It can also generate design constraints (.sdc) based on libraries after optimized the design. The timing information for the design also could be verified during the synthesis. In this step, the Liberty Timing Files are required for all cells.

After the gate-level netlist is tested and verified, The layout generation is the last process of turning a design into a manufacturable file. The first step for layout generation is floor planning. The floor planning includes the size of the core, the distance between the core and I/O pad, core rail, and some other related settings. Then place the design into the floor and connect (routing) them as described in the netlist with the help of placement tools. After the timing analysis and optimization, the layout could be generated. This step needs the LEF file which describes the physical information of cells/macros, LIB file which describes timing/power information, a netlist, and design constraints after synthesis.

So, we need to implement the LIB and LEF file for the custom cell which is the memory array to make our approach work in order to make further evaluations.

Liberty Timing File (LIB)

The LIB file is an ASCII representation of the timing and power parameters associated with ant cell in particular semiconductor technology. These parameters are obtained by simulating cells under a variety of conditions. The data is represented in the LIB format.

For each cell of the library, it describes the port, cell type, operating conditions, power consumption, and timing modeling. The LIB file contains timing models and data that need to be calculated, including I/O delay paths, timing check values, interconnect delays. The I/O delays and timing check values are calculated between each instance. The delay in the circuit path depends on the electrical behavior of the interconnections between the cells.

```
library (name) {
... library based attribute ...
cell (name) {
... cell based attribute ...
pin (name | name_list) {
... pin description ...
}
}
cell (name) {
... cell based attribute ...
```

```
bus (name) {
... bus based attribute ...
pin (name | name_list) {
... pin description ...
}
}
}
}
```

The basic format of the LIB file is listed above. Each level has its attribute settings, including operating conditions, time information, power consumption information. See the appendix for more syntax details.

Timing Generation

There have two timing variables that need to be set in the LIB file, transition time and delay. The transition time here used was provided by the Xenergic directly. The timing calculation is focused on the delay of output pins. Xenergic provided delays with a special size of bit cells. Assume the delay is linearly increased (it should be a little less than that) with the size of bit cells. The worst-case from the provided values were used to generate the delay values.

Power Calculation

Leakage and internal power are the two needs in the LIB. Same as the delays, Xenergic has the leakage current for a single bit cell. The leakage can be calculated by using the formula IV for single bit cell, then multiply by the number of bit cells. The capacitance for a special size of bit lines and word lines were provided. They were used to generate the capacitance for pins of the word line and bit line. Then use the formula CV^2 to calculate the internal power of input pins.

Library Exchange Format (LEF)

According to the previous mentioned, LEF is a necessary file to generate a layout. LEF defines the elements of an IC process technology and associated library of cell models [12]. It gives the view of cells in PNR.

There are usually "technical" LEF files and "cell library" LEF files. The technical LEF file contains all LEF technical information for the design, such as layout and routing design rules and layer process information. The cell library includes the info about boundary, blockage, pin position, and the metal layer of cells.

In this project, only the part of the MACRO (cell library) describing the custom cell needs to be generated, so here is a brief introduction to the MACRO

part. The rest of the LEF sections (technical library) that define layers and rules are not described here.

```
MACRO macroName
[CLASS
{ COVER [BUMP]
| RING
| BLOCK [BLACKBOX | SOFT]
| PAD [INPUT | OUTPUT |INOUT | POWER | SPACER | AREAIO]
| CORE [FEEDTHRU | TIEHIGH | TIELOW | SPACER | ANTENNACELL | WELLTAP]
| ENDCAP {PRE | POST | TOPLEFT | TOPRIGHT | BOTTOMLEFT | BOTTOMRIGHT}
}
;]
[FOREIGN foreignCellName [pt [orient]] ;] ...
[ORIGIN pt ;]
[EEQ macroName ;]
[SIZE width BY height ;]
[SYMMETRY {X | Y | R90} ... ;]
[SITE siteName [sitePattern] ;] ...
[PIN statement] ...
[OBS statement] ...
[DENSITY statement] ...
[PROPERTY propName propVal ;] ...
END macroName
```

As shown in the syntax example above, the LEF of Macro contains the definition of CLASS, FOREIGN, ORIGIN, SIZE, SYMMETRY, SITE, PIN, OBS.

- CLASS specifies the macro type. Type BLOCK is used in a hierarchical design. Type CORE is used for standard cells and should always contain a SITE definition.
- FOREIGN specifies the foreign (GDSII) structure name with a default offset 0 0.
- ORIGIN determines how the macro align with DEF placement point.
- SIZE specifies placement boundary, width BY height.
- SYMMETRY specifies which orientations can be used during the placement. X for North and Flipped South; Y for North and Flipped North; X Y for North, Flipped North, South, Flipped South.
- SITE specifies the associated location that will be used in the placement.
- PIN defines pins. It must be included in the LEF and should specify all pins, including power pin.
- OBS specifies sets of blockages for the macro.

We need to do some extra works to make it work as a standard cell. The size of the macro usually is greater than the standard cell, which means it can not be

placed between standard core rails. The power rail is needed if the macro placed over rows of core rail. So the LEF file needs to have an embedded power rail which can be connected to the standard core rails during the place and route, as shown in Figure 3.2.

The blockage was created as the location of the macro to avoid routing inside the macro. As shown in Figure 3.3, the smaller blockage for layers from and below Metal3 is the macro. The overall blockage for layers from and below Metal2 is used for power ring.

The LEF file generated here is only to have an overview of the expected layout and also for the evaluation of the whole design. There was no layout modification or generation in this project.

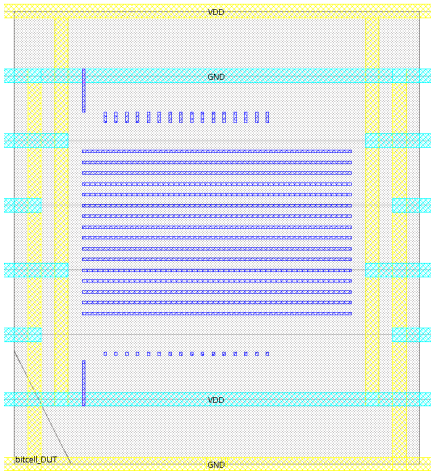

Figure 3.2: LEF Power Ring

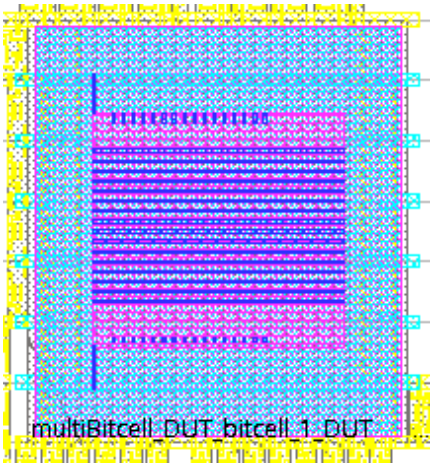

Figure 3.3: LEF Blockage

Design for Multiple Macros

Another design is implemented for driving multiple Macros, as shown in Figure 3.4. This design is used for further evaluation. The design concept is still the same. There is one change in driver for the address. The address is split into two parts. The lower part (passing to the decoder) is used to generate WL for Macro, which depends on the size of a macro. The upper part (passing to MultiMacro) is used to select macro which to be active. MultiMacro block (Figure 3.5) works similarly to the column decoder.

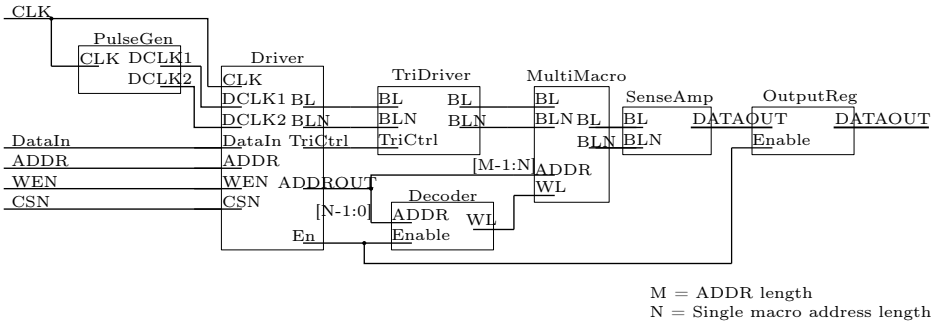

Figure 3.4: Design for Multiple Macros

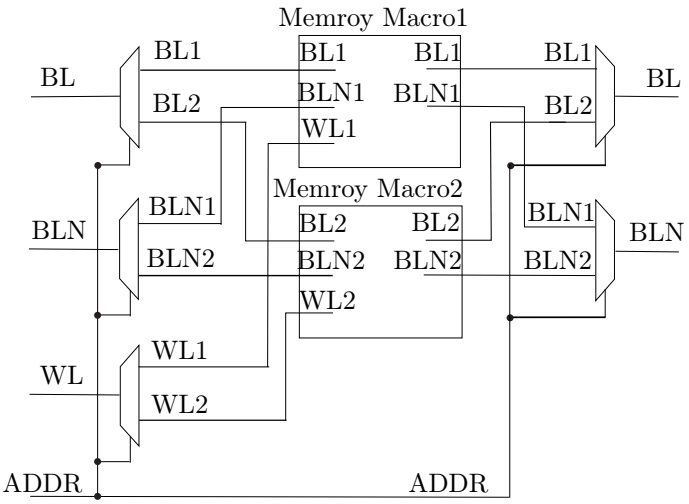

Figure 3.5: MultiMacro Block

Chapter 4
Test & Evaluation

This chapter describes the tests that were conducted during the project. Some results from the test also shown in this chapter. The last part of this chapter explains the settings for the evaluation.

Functionality Test

The implementation of the design must be tested to ensure proper operation before any evaluation is performed. The test went through all the steps of ASIC design flow and with the help of simple testbench to check the logical result.

Synthesis

To ensure that the design works after synthesis, the PulseGen block needs to be set as do not touch and written in the gate level with delay cells to keep the delays after optimized. The tri-driver also needs to be set as don't touch since the control single is a small pulse that will be optimized away during synthesis. The synthesis also requires to allow the latch to happen (hdl_latch_keep_feedback true) to keep the output latch register works.

Place & Route

During the place and route, the LEF file is tested first after the placed design. The memory array block, which is imported from the LEF file, should align with core rail. Also, power rails connect to the right pins from memory array, as shown in Figure 4.1.

The tools can move the memory array during optimization as a standard cell. The clock tree optimization gave the wrong result since the output register is latch register with enable signal which is generated from the clock. So the enable signal needs to be ignored during the clock tree optimization to obtain the correct result. The following command is for ignoring the pin during timing optimization.

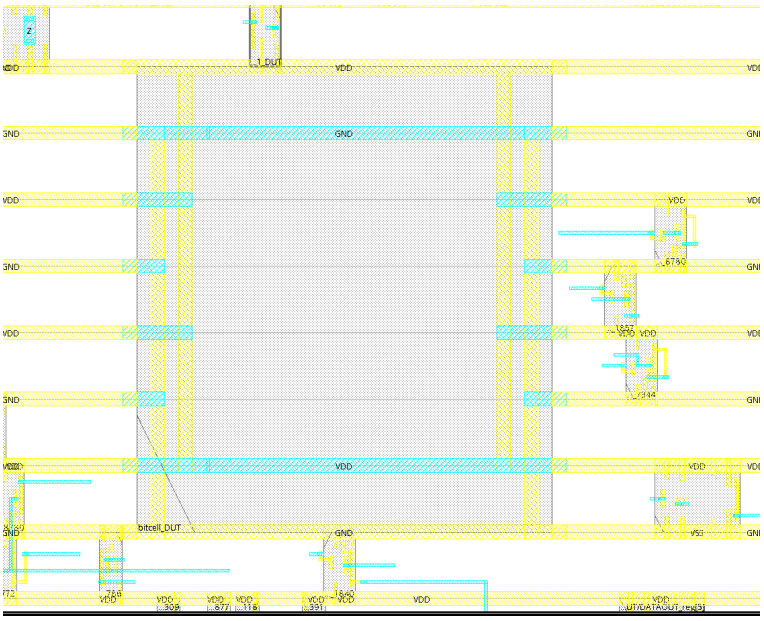

Figure 4.1: LEF IN PNR

set_ccopt_property sink_type -pin [PIN] ignore

As the memory array block is higher than one core rail height, and it is not like a multi-row standard cell, which has power rail inside with the same height at the same location. The power rail generated by the tools which overlap with the memory array block should be removed after all optimization. The following command that means trim off all nets inside the selected area and is used for both VDD and GND with pattern 00.

trim_pg -net VDD -type stripe -layer M1 -area [dbGet selected.box] -pattern 00

Post Simulation

The post-simulation has been done after the netlist and SDF file had been obtained from PNR. A simple test bench was created to do the test. The test bench does all write then all read with input data as the same as the address.

As shown in Figure 4.2, all the input signals were given valid values before clock rising edge and hold for some time after clock rising edge so the input can meet the setup/hold timing.

As shown in Figure 4.3, the read delay for reading from address 1 is 447ps in this set of tests, which matches the result from synthesis and PNR.

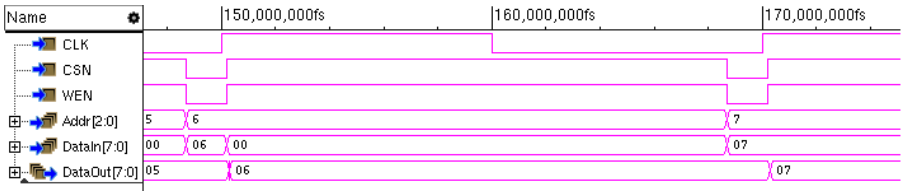

Figure 4.2: Post Simulation Write Waveform

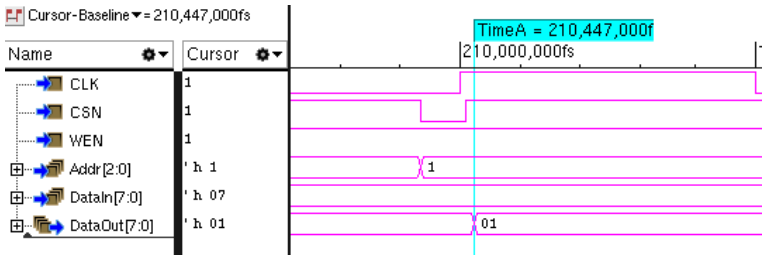

Figure 4.3: Post Simulation Read Waveform

Schematics Simulation

Since the memory array used for post-simulation is the behavior module, the simulation can only prove that the design will send right signals to the memory array and process signals from the memory array and gain the correct result. Schematics simulation is required to make sure that the design can drive the Bit-cell.

By import of physical netlist (from PNR) into Virtuoso, the whole design will have a schematic with the layout. After creating a simple test bench with schematics, the simulation can be done.

Figure 4.4 shows a simple test result. CSN in the test has been set to 0 to make the simulation easy since it is only the functionality test. The first part is writing different values into different memory cells. Another part is reading from different memory cells.

Figure 4.5 shows details about the reading operation. At each rising edge of the clock, BLs were precharged to high. After WL actives, one of BL should be pulled down, and the storage cell which stores low should go a little bit higher then back to low during the pulldown period.

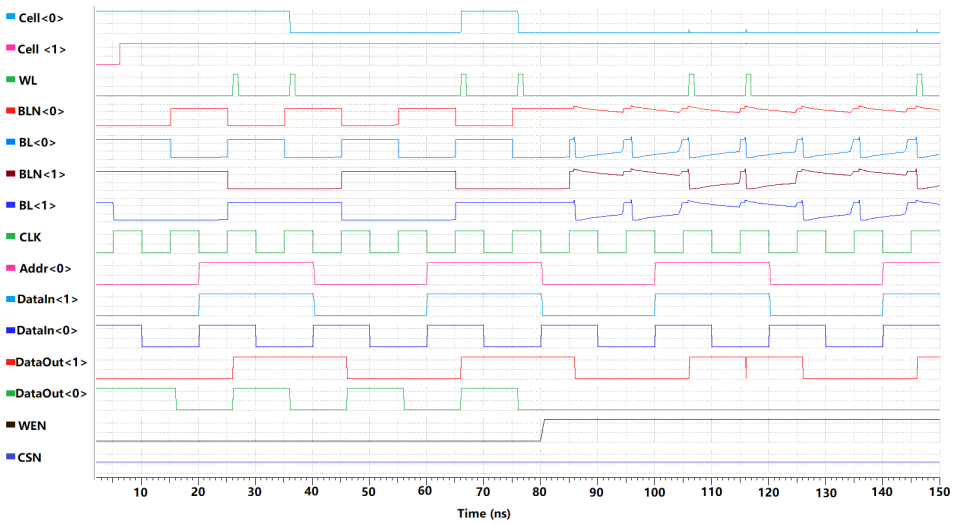

Figure 4.4: Schematics Simulation Waveform

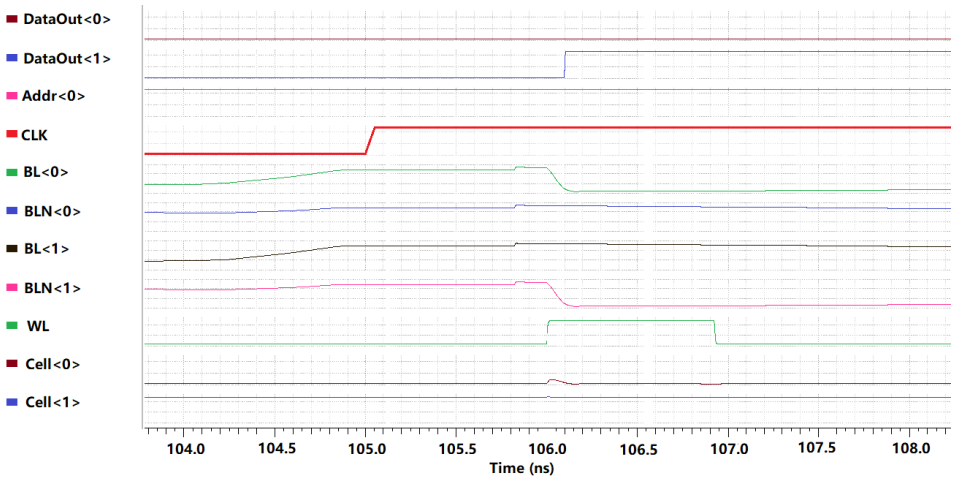

Figure 4.5: Schematics Simulation Read Waveform

BIST Test

After all previous tests, the BIST test became the last functionality test. The new DIL-SRAM has been connected to March C BIST and re-runs previous flow to Post Simulation. The design has been successfully merged with BIST logic (distributed in BIST) in PNR. Moreover, the simulation has been done without error.

Evaluation

Before the evaluation, all these designs (including LEF and LIB generation) can be generated automatically through python script with a single configuration. The address and data width of one macro and number of macros can be set in the configuration. To avoid invalid data, every single configuration during the evaluation reaches post-simulation. And the post-simulation result should be checked.

The evaluation is focused on area and power (leakage, dynamic power). To be able to evaluate performance as comprehensively as possible, the evaluation was set up with different combinations of Address width (single macro), data width and the number of the macro. The following ranges were used for the combinations.

- Address width of a single macro starts from 3 to 9.
- Data width starts from 8 to 64 (step by the power of 2).
- The number of macros starts from 1 to 4 (step by the power of 2).

Because the decoder of the design can only synthesis-able up to 9-512 (memory issues of the tool), that the address width of each macro can run only up to 9. 512 WLs is enough for single macro since the new memory focuses more on the small size. The test has $7 \times 4 \times 3 = 84$ combinations which are from 8×8 to 4096×32 bits.

After collecting the above evaluation results, the obtained results are compared and analyzed with the original memory data. The comparison is discussed in the next chapter. And according to the confidentiality agreement, all the data shows here and the following chapter for the comparison are normalized.

Area and Timing for Single Macro

As shown in Figure 4.6, the density of memory cell (bit/area) increases with memory size (word $\times$ I/O). The increase of words gives more effect on density than I/O. It happens since the increase of word only (without considering the memory macro's size) changes the design of the decoder. However, the I/O needs to change the sense amplifier, output register, and driver. The increase in I/O needs more gates than the word. On another hand, the increase of word leads to

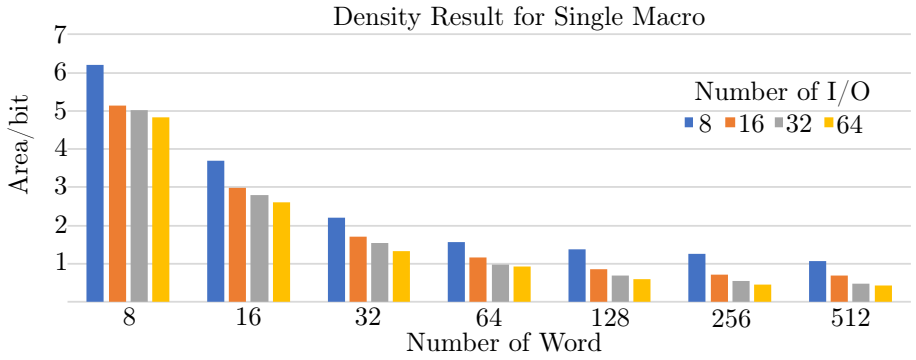

Figure 4.6: Density Result for Single Macro

a much more extended clock period due to the read delay from a longer bit line as shown in Figure 4.7.

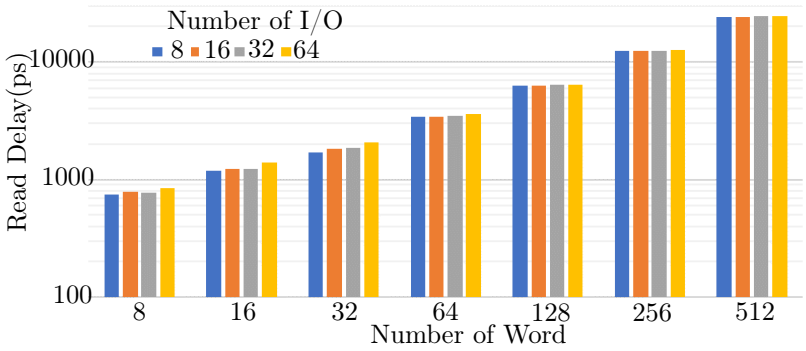

Figure 4.7: Timing Result for Single Macro

Area and Timing for Multiple Macros

Comparing the results from Figure 4.6, Figure 4.8 and Figure 4.9, the area is larger for multiple macros than single for the same memory size (word × I/O). The increase of area is related to the MUX logic does not exist in the single macro design. By making a more detailed check, the area of multiple macros (compared to the same size of single macro or less mux) started to be smaller in a specific setting. That is because the amount (area) of delay cells for pulse generator starts to be increased more than the mux logic due to the read delay (pulse width) increasing with length of word line for single macro. Furthermore, the length of I/O also affects the density increasing point since the area of the mux is highly dependent on it.

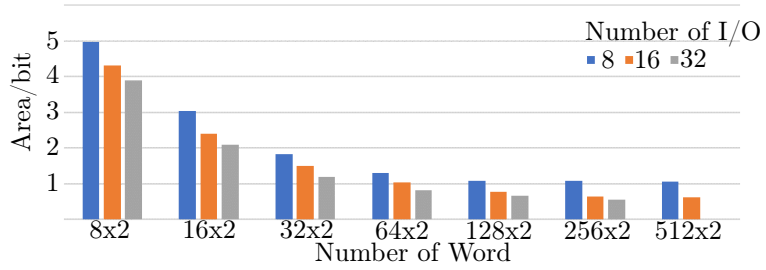

Figure 4.8: Density Result for Two Macros

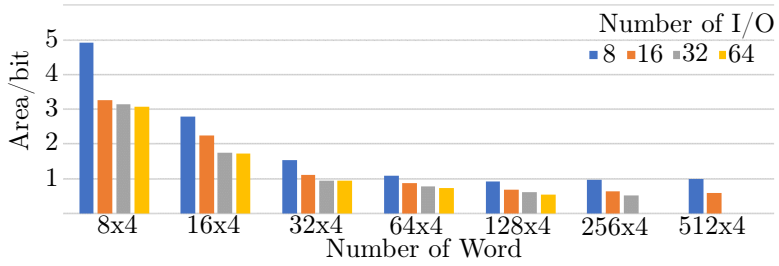

Figure 4.9: Density Result for Four Macros

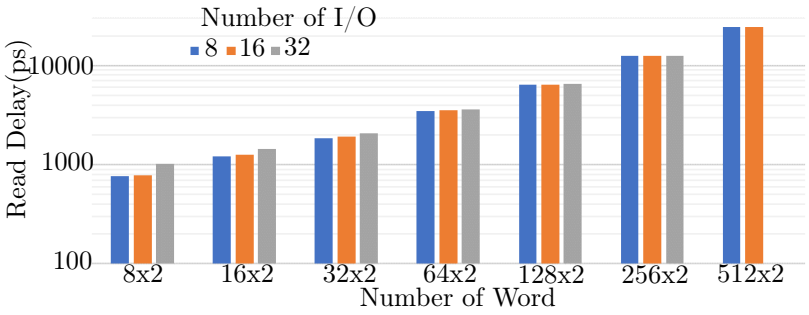

Figure 4.10: Timing Result for Two Macros

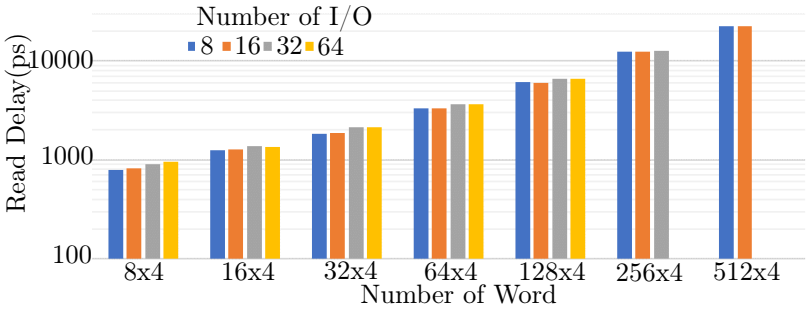

Figure 4.11: Timing Result for Four Macros

Comparing the results from Figure 4.7, Figure 4.10 and Figure 4.11, the read delay is reduced for the same memory size if the number of macros increased since it relies on the word length of each macro. As an example, 32×8 with a single macro has a timing period for 1686 which is 1221 for two macros. The reason is that the word length of the macro is 32 for the single design but 16 for the two macros design. So 32×8 with two macros has a similar timing as 16×8 with single macro, but a little larger due to the extra mux logic. The same situation happens to four macros design, as shown in Figure 4.11.

Power Consumption

The results are shown in Table 4.1, Table 4.2 and Table 4.3. Leakage was increased based on the length of the word, length of I/O, and the number of macro. Same as the area, the length of I/O has the most significant impact on leakage and the length of the word has the least impact. As the result for dynamic power after synthesis was not reasonable, only the dynamic power for reading/writing, which got after import specified VCD file, was considered here. Read operation has more dynamic power than write due to the pre-charge operation. The length of I/O has more impact on dynamic power than the length of the word. The increase of mux (keep the same memory size) reduced the dynamic power since only one of the macros works at a time.

Word	I/O	Dyn Power (normalized/Mhz)			Leakage (normalized)		
		Read	Write	Synthesis	Read	Write	Synthesis
8	8	13.63	8.35	28.18	216	241	228
8	16	25.89	13.29	18.82	387	440	436
8	32	53.07	23.38	33.74	726	834	827
8	64	94.79	42.72	64.53	1410	1622	1610
16	8	14.34	8.37	13.88	238	264	263
16	16	28.22	14.53	21.12	408	461	457
16	32	54.49	23.87	116.75	748	855	798
16	64	100.21	42.92	226.72	1432	1644	1530
32	8	18.69	12.43	40.52	266	290	276
32	16	32.22	16.53	23.39	436	487	482
32	32	61.51	25.48	139.14	776	881	824
32	64	109.94	44.27	69.14	1458	1670	1658
64	8	25.96	17.88	56.32	327	353	339
64	16	40.63	22.15	98.08	497	550	521
64	32	72.32	30.95	44.09	837	944	937
64	64	130.28	52.00	355.87	1521	1732	1618
128	8	37.65	27.56	30.54	391	417	416
128	16	56.14	31.79	147.11	562	614	587
128	32	95.46	40.46	52.80	902	1009	1003
128	64	169.17	61.53	525.66	1587	1799	1686
256	8	62.57	48.52	143.80	558	585	568
256	16	88.76	52.73	246.04	729	782	750
256	32	143.53	61.35	71.03	1070	1178	1168
256	64	248.39	80.05	866.64	1757	1968	1851
512	8	110.37	88.57	83.36	863	891	883
512	16	152.02	92.75	90.72	1036	1089	1079
512	32	237.71	101.34	811.08	1379	1487	1423
512	64	410.06	119.58	136.43	2054	2282	2264

Table 4.1: Memory Array with Single Macro Result —Power

Word	I/O	Dyn Power (normalized/Mhz)			Leakage (normalized)		
		Read	Write	Synthesis	Read	Write	Synthesis
8x2	8	14.78	8.87	32.72	283	312	296
8x2	16	26.85	13.46	23.63	509	567	555
8x2	32	53.46	22.99	42.64	951	1071	1048
16x2	8	17.23	10.64	16.49	303	331	320
16x2	16	30.03	14.99	68.56	524	583	556
16x2	32	58.27	23.97	136.31	970	1090	1033
32x2	8	20.76	12.76	45.04	335	363	347
32x2	16	35.40	16.99	80.88	557	615	589
32x2	32	67.39	25.79	158.57	1004	1124	1067
64x2	8	29.35	18.39	61.20	376	404	389
64x2	16	47.94	22.61	34.23	599	657	646
64x2	32	87.50	31.23	53.24	1041	1163	1139
128x2	8	46.71	28.11	93.70	651	679	663
128x2	16	74.10	32.29	46.47	872	930	913
128x2	32	131.29	40.86	298.16	1316	1435	1377
256x2	8	79.90	49.16	156.76	1033	1064	1041
256x2	16	123.76	53.31	68.43	1268	1332	1300
256x2	32	213.78	61.85	87.22	1737	1872	1813
512x2	8	144.01	89.54	100.90	1787	1822	1793
512x2	16	219.65	93.74	111.57	2076	2148	2101
512x2	32	373.04	102.29	239.46	2649	2801	2737

Table 4.2: Memory Array with 2 Macros Result —Power

Word	I/O	Dyn Power (normalized/Mhz)			Leakage (normalized)		
		Read	Write	Synthesis	Read	Write	Synthesis
8x4	8	15.18	9.28	17.08	312	345	346
8x4	16	27.52	13.99	68.75	552	620	603
8x4	32	54.43	23.52	48.35	1037	1177	1136
8x4	64	120.43	44.70	93.91	2022	2295	2201
16x4	8	17.47	10.91	40.72	342	375	362
16x4	16	30.52	15.37	33.24	584	651	627
16x4	32	58.97	24.44	55.99	1068	1207	1162
16x4	64	116.32	36.08	289.54	2042	2322	2246
32x4	8	21.14	13.13	47.91	356	389	376
32x4	16	35.94	17.37	33.08	599	667	639
32x4	32	74.65	26.62	164.77	1091	1223	1186
32x4	64	142.94	46.16	331.36	2073	2336	2257
64x4	8	29.77	18.83	64.68	410	443	431
64x4	16	48.48	23.06	39.98	649	716	693
64x4	32	95.37	32.10	216.90	1139	1268	1236
64x4	64	180.83	51.55	114.14	2103	2372	2281
128x4	8	47.49	28.49	97.07	760	793	780
128x4	16	75.53	32.69	167.16	1006	1070	1052
128x4	32	136.76	41.62	76.41	1484	1621	1580
128x4	64	253.88	60.47	126.58	2452	2727	2635
256x4	8	80.72	49.58	159.47	1239	1273	1253
256x4	16	125.28	53.79	270.77	1508	1579	1551
256x4	32	216.48	62.47	489.40	2046	2192	2155
512x4	8	144.83	89.87	286.42	2194	2236	2218
512x4	16	221.24	94.04	117.17	2551	2637	2567

Table 4.3: Memory Array with 4 Macros Result —Power

Chapter 5
Comparison and Analysis

The last part of this project is to compare with standard SRAM in the same technology. The following comparison was made based on the original SRAM that Xenergic supplied and the new implemented SRAM.

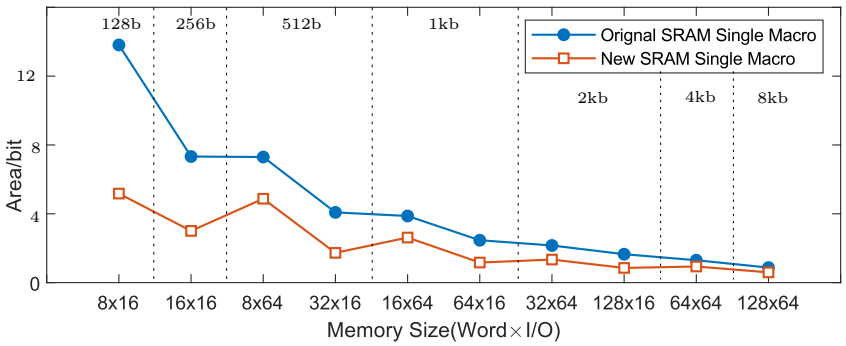

Figure 5.1: AreaSingleMacro (normalized)

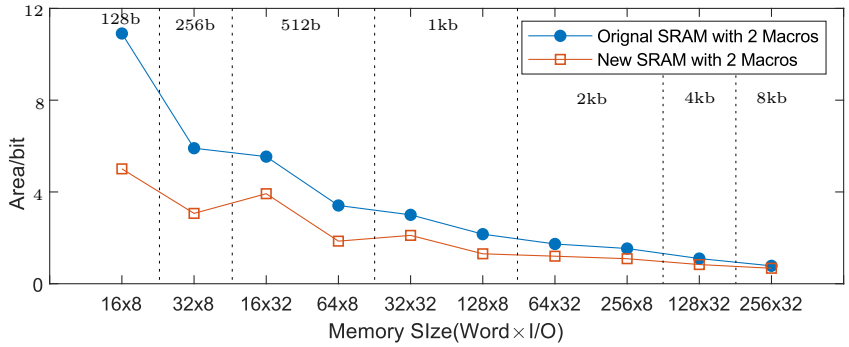

Figure 5.2: Area2xMacro (normalized)

As shown in Figure 5.1 to Figure 5.3, the blue line is for the original memory, and the orange is for the new memory. It is not difficult to see that new memory has

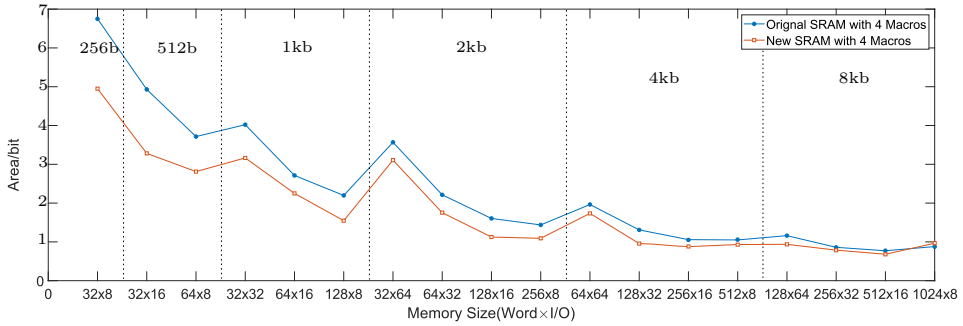

Figure 5.3: Area4xMacro (normalized)

a higher density in small capacity areas. At about 2kb, it gradually began to lose its advantage. They all have a higher density for multi macros than single macro, and keeps the same shape as the single macro, but get a little bit closer to each other.

The area reduced around 60 percent for 128b and 256b with a single macro and 30 percent for 512b and 1kb. The Area reduced 50 (128b, 256b) and 30 (512b, 1kb) percent for 2 macros. Around 20 percent area reduce for four macros within 1kb.

The new SRAM has a smaller advantage in quad-macro. This is because the logic for quad-macro becomes much complex than single and dual macros, which takes more area. This is also a reason that the test only runs till 4 for the number of the macro.

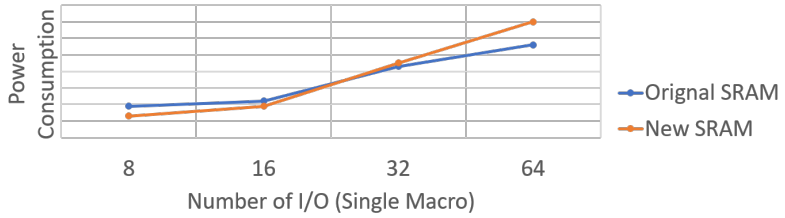

Figure 5.4: Power Comparison (normalized)

After being compared with the data from Xenergic, the total power from the New SRAM is almost the same or smaller if the I/O is 32 or less, and the memory size is 8192 or less (Figure 5.4). The power consumption is about 10 percent less for 16 I/O and 30-40 percent less for 8 I/O. The new memory consumes less power with less I/O and consumes more power with more I/O.

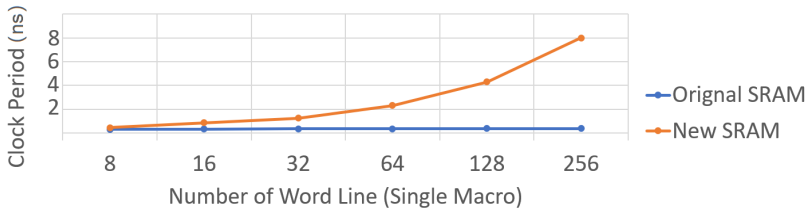

Figure 5.5: Speed Comparison (normalized)

The clock period for the original SRAM is generally around 0.4ns to 0.6ns. The New memory's clock cycle is generally the same level as traditional memory if the word of a single macro is 8 or less. The more Word length, the longer the period, as shown in Figure 5.5.

Decrease / Increase	Word Length (Single Macro)	Data Length (Single Macro)	Number of Macro (No more than 4)	
Word Length (Single Macro)		Increase	Increase	**Density**
		Decrease a lot	Decrease a lot	**Speed**
		Decrease	Decrease	**Power**
Data Length (Single Macro)	Decrease		Decrease a little	**Density**
	Increase a lot		Same	**Speed**
	Increase		Increase	**Power**
Number of Macro (No more than 4)	Decrease	Increase a little		**Density**
	Increase a lot	Same		**Speed**
	Increase	Decrease		**Power**

Figure 5.6: Trade-off Discussion for Specific Capacity of Memory

We can see that there are some trade-offs between speed, power consumption, and memory size. As shown in Figure 5.6, assume we have a specified capacity for memory. The increase of word length in a single macro gives a longer clock period but more area reduction and less power consumption. The increase in data length in a single macro gives higher power consumption and less area reduction but higher speed. The increase in macros gives a little bit less area reduction but higher speed.

Even though there are trade-offs, we do have some combinations which have a similar speed but less power consumption than the original. These can be used for high bandwidth memory. Such as 8×32 for a single macro ($2 \times 8 \times 32$ for two macros, $4 \times 8 \times 32$ for four macros) which has a similar clock period and power consumption.

As the result mentioned before, the new SRAM indeed has a higher den-

sity for size 2k or less. However, the speed of the new SRAM was limited by the length of the word. It would need more time (smaller area) than the original if the word length increased. It is still a trade-off between speed or area.

The power consumption of new SRAM is good for 8k (or less) if the I/O is 32 (or less). This is another trade-off between speed and power consumption (more I/O means less word length, which gives higher speed).

Based on the current result we got, our approach should be efficient to use as the replacement of large SRAM than conventional small SRAM. With the help of mixing logic and memories, the data transfer path reduced which reduced the time and power spent during the transfer path.

Chapter 6
Conclusion

This project makes the implementation and evaluation of a new approach to construct SRAM. The implementation includes reconstructing the entire SRAM structure through RTL and behavior (memory cell) with Verilog, modified LEF/LIB files, automatically generation by Python. The evaluation is done by following the design flow (synthesis, PNR, post-simulation/schematic-simulation).

Based on all the works and results, the implemented SRAM is successfully working as expected. Even though there are some trade-offs between power consumption and access speed, the implemented SRAM is a high-density memory for small size (2k or less). It can have more than 20 percent area reduction for SRAM with 1k storage. Furthermore, the implemented memory can be mixed into other digital designs (act as in-logic memory) and handled by the PNR tools directly. By using multiple of the implemented SRAM in design will lead to a high bandwidth distributive in-logic memory with higher density and no extra (or less) power consumption than using multiple standard SRAM.

Chapter 7
Future Work

Current work is only compared with conventional SRAM. It should be compared with full standard cell memory (latch-register based) as well. And the work should have more evaluation and analysis between traditional architecture and the new approach.

The LEF file used in this project wasted more area than expected. It can be reduced by re-adjusting the power connection. It should be modified through layout by analog designer directly. The timing information used in the LIB file also can be modified based on reality. The delay controls in the pulse generator can be modified as well. These will further increase density and frequency.

The current implementation only tested in specified technology. It should be tested and modified for other technologies to make it generic in the future.

References

[1] Simevski Aleksandar, "Architectural framework for dynamically adaptable multiprocessors regarding aging, fault tolerance, performance and power consumption," 2015, Brandenburg University of Technology Cottbus-Senftenberg

[2] Benmoussa Yahia, "Performance and Energy Consumption Characterization and Modeling of Video Decoding on Multi-core Heterogenous Mobile SoC and their Applications," 2015, 10.13140/RG.2.1.4094.0640.

[3] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, San Francisco, CA, 2014, pp. 10-14.

[4] G. Kestor, R. Gioiosa, D. J. Kerbyson and A. Hoisie, "Quantifying the energy cost of data movement in scientific applications," *2013 IEEE International Symposium on Workload Characterization (IISWC)*, Portland, OR, 2013, pp. 56-65.

[5] A. Biswas and A. P. Chandrakasan, "Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications," *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, San Francisco, CA, 2018, pp. 488-490.

[6] M. Khare et al., "A high performance 90nm SOI technology with 0.992 μm^2 6T-SRAM cell," *Digest. International Electron Devices Meeting*, San Francisco, CA, USA, 2002, pp. 407-410.

[7] Wm. A. Wulf, S. A. McKee, "Hitting the memory wall: implications of the obvious", ACM SIGARCH Comp. Arch. News, Vol 23, (1), pp 20-24, 1995.

[8] M.V. Wilkes, "The memory wall and the CMOS end-point", ACM SIGARCH Comp. Arch. News, Vol 23, (4), pp.4-6, 1995.

[9] O. Andersson, B. Mohammadi, P. Meinerzhagen, A. Burg and J. N. Rodrigues, "Ultra Low Voltage Synthesizable Memories: A Trade-Off Discussion in 65 nm CMOS," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 6, pp. 806-817, June 2016.

[10] O. Andersson, B. Mohammadi, P. Meinerzhagen and J. N. Rodrigues, "A 35 fJ/bit-access sub-VTmemory using a dual-bit area-optimized standard-cell in 65 nm CMOS," *ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC)*, Venice Lido, 2014, pp. 243-246.

[11] A. Bhaskar, "Design and analysis of low power SRAM cells," *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, Vellore, 2017, pp. 1-5.

[12] "LEF/DEF Language Reference", Product Version 5.7, November 2009, Cadence Design Systems

Appendix A

LIB Syntax Example

Example 1-1 General Syntax of the Library Description
Liberty User Guide, Volume 1 Version 2017.06

```
library (name) {
   technology (name) ; /* library-level attributes */
   delay_model : generic_cmos | table_lookup | cmos2 |
   piecewise_cmos | dcm | polynomial ;
   bus_naming_style : string;
   routing_layers (string) ;
   time_unit : unit ;
   voltage_unit : unit ;
   current_unit : unit ;
   pulling_resistance_unit : unit ;
   capacitive_load_unit (value, unit) ;
   leakage_power_unit : unit ;
   piece_type : type ;
   piece_define ("list") ;
   define_cell_area (area_name, resource_type) ;
      /* default values for environment definitions */
   operating_conditions (name) {
      /* operating conditions */
   }
   timing_range (name) {
      /* timing information */
   }
   wire_load (name) {
      /* wire load information */
   }
   wire_load_selection () {
      /* area/group selections */
   }
   power_lut_template (name) {
      /* power lookup table template information */
   }
   cell (name1) {/* cell definitions */
      /* cell information */
   }
   cell (name2) {
      /* cell information */
   }
   scaled_cell (name1) {
      /* alternate scaled cell information */
   }
```

```
   type (name) {
      /* bus type name
   }
   input_voltage (name) {
      /* input voltage information */
   }
   output_voltage (name) {
      /* output voltage information */
   }
}
```

Example 3-2 Setting Library-Level Default Attributes for a CMOS Library
Liberty User Guide, Volume 1 Version 2017.06

```
library (example) {
   ...
   /* default cell attributes */
   default_cell_leakage_power : 0.2;
   /* default pin attributes */
   default_inout_pin_cap : 1.0;
   default_input_pin_cap : 1.0;
   default_output_pin_cap : 0.0;
   default_fanout_load : 1.0;
   default_max_fanout : 10.0;
   wire_load (WL1) {
      ...
   }
   operating_conditions (OP1) {
      ...
   }
   default_wire_load : WL1;
   default_operating_conditions : OP1;
   default_wire_load_mode : enclosed;
   ...
}
```

Example 4-1 cell Group Example
Liberty User Guide, Volume 1 Version 2017.06

```
library (cell_example){
   date : "August 14, 2015";
   revision : 2015.03;
   cell (inout){
      pad_cell : true;
      dont_use : true;
      dont_fault : sa0;
      dont_touch : true;
      area : 0;/* pads do not normally consume internal
      core area */
      cell_footprint : 5MIL;
      pin (A) {
         direction : input;
         capacitance : 0;
      }
      pin (Z) {
         direction : output;
         function : "A";
```

```
        timing () {
           ...
        }
     }
  }
  cell(inverter_med){
     area : 3;
     preferred : true;
     pin (A) {
        direction : input;
        capacitance : 1.0;
     }
     pin (Z) {
        direction : output;
        function : "A' ";
        timing () {
           ...
        }
     }
  }
  cell(nand){
     area : 4;
     pin(A) {
        direction : input;
        capacitance : 1;
        fanout_load : 1.0;
     }
     pin(B) {
        direction : input;
        capacitance : 1;
        fanout_load : 1.0;
     }
     pin (Y) {
        direction : output;
        function : "(A * B)' ";
        timing() {
           ...
        }
     }
  }
  cell(buff1){
     area : 3;
     pin (A) {
        direction : input;
        capacitance : 1.0;
     }
     pin (Y) {
        direction : output;
        function : "A ";
        timing () {
           ...
        }
     }
  }
} /* End of Library */
```

Printed by Tryckeriet i E-huset, Lund 2020

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2020-749
http://www.eit.lth.se