DESIGN AND IMPLEMENTATION OF A LOW-POWER RANDOM ACCESS MEMORY GENERATOR

Deborah Capello

LiTH-ISY-EX-3318-2003
Linköping, 2003
DESIGN AND IMPLEMENTATION OF A
LOW-POWER RANDOM ACCESS
MEMORY GENERATOR

Master thesis in Electronics Systems
at Linköping University
by

Deborah Capello

LiTH-ISY-EX-3318-2003

Supervisor: Henrik Ohlsson
Examiner: Prof. Lars Wanhammar
Linköping, 6th of June 2003
Design och implementering av en lägeffekts-RAM-generator

Design and Implementation of a Low-Power Random Access Memory Generator

In this thesis, a Static Random Access Memory generator has been designed and implemented. The tool can generate memories of different sizes. The number of words that can be stored can be chosen among powers of 2 and the number of bits per word can be up to 48.

The focus of the thesis was to find an adequate structure for the generated memories depending on the size, and develop a memory generator that implements the structures, which has been thoroughly done. The single circuits used in the generated memories can be substituted with better circuits as well as adapted to other processes.

All circuits apart from a block decoder circuit have been developed. The memory generator was not supposed to automatically produce a complete layout, and some manual interventions on the memories generated by the tool are necessary. The tool requires to be developed further to minimise this manual intervention on the generated memories. The complete memories generated have not been tested because of their complexity, but tests on circuits as well as many parts of the memories have been carried out.

During the thesis work, a large amount of tasks had to be carried out and a lot of issues had to be dealt with, which has been a problem. The tool used for the implementation has powerful features for both analog and digital electronic design, but has stability problems with large designs, which has been a big obstacle in this work.

SRAM, generator, low-power, layout
ACKNOWLEDGEMENTS

It is now time to thank all that have contributed to this master thesis. It seems appropriate to start with the person that suggested this very stimulating work which suited me perfectly, my examiner professor Lars Wanhammar.

A big big big thank has to be given to my supervisor, Henrik Ohlsson, who stood by me with excellent suggestions and comments whenever I needed them, no matter whether he had time to deal with all my questions or not. I was once more lucky to get the best possible supervisor. Nice to see that all questions did not prevent him from finishing his licentiate work.

Furthermore, I would like to thank for the technical support: Weidong Li, who had to listen to all my bug complaints and helped me among other things with layout questions; Robert Hägglund for his help with my questions on analog design, SKILL and Cadence; Emil Hjalmarson for sharing his SKILL expertise; and all others at Electronics Systems that always did their best to help, Jonas, Greger, Peter, and Ola.

A major thanks has to be given to Nils Funke and Terese Ekebrand who did not only help me with debugging, by being insightful opponents on my master thesis, and by giving me helpful tips on SKILL coding and layouting, but also helped me through this work with their friendship and moral support. Even the rest of the coffee break team, Jonas, Micke, and Stefan, deserve my thanks for cheering me up during lunches and coffee breaks.

Reading this long thesis can hardly be done for fun. So a big thank to Janne and Rikard for proofreading it anyway, even if it was not their main field of interest.

At last but not least, I would like to thank Janne once more for backing me up all along with all practical matters I did not have time for during this thesis work. Without him I would have practically starved a long time ago.
# Table of Contents

1 Introduction ............................................................................................................. 1

1.1 Background.......................................................................................................... 1

1.2 Requirement Specifications for the Memory Generator ...................................... 2
  1.2.1 Memory Size .................................................................................................. 2
  1.2.2 Low-Power .................................................................................................. 3
  1.2.3 Flexibility .................................................................................................... 3
  1.2.4 Memory Control ......................................................................................... 3
  1.2.5 Speed .......................................................................................................... 4
  1.2.6 Chip Area .................................................................................................... 4

1.3 Limitations and Further Requirements on the Project ..................................... 4

1.4 Overview ............................................................................................................ 5
# Table of Contents

## 2 Methodology and Tools

### 2.1 Cadence and its Features

2.1.1 Views in Cadence

- Schematic View
- Layout View
- Extracted View
- Analog Extracted View
- Configuration View

2.1.2 Working with Cadence

2.1.3 Simulation Possibilities

2.1.4 How to Create a Circuit

- Manual Creation of a Circuit
- Programming a Circuit

### 2.2 A Short Description of Skill

### 2.3 Process Description

### 2.4 Procedure

2.4.1 Steps Towards a Memory Generator

- Step 1 - Literature Study
- Step 2 - Hands-On Acquisition of Knowledge on SRAMs
- Step 3 - Choices for Memory Design
- Step 4 - Floorplanning
- Step 5 - Placing Parts
- Step 6 - Refining the Circuitry
- Step 7 - Timing Phase

2.4.2 Comments on the Steps
# 3 SRAM - A Theoretical Background

## 3.1 An Introduction to Random Access Memories

- **3.1.1 Performance Measures for RAMs**
- **3.1.2 Static and Dynamic RAMs**
- **3.1.3 Operation of an SRAM Cell**

## 3.2 A Simple Model for SRAMs

- **3.2.1 Structural Description of a Simple SRAM**
- **3.2.2 Power Consumption in the Simple SRAM**
- **3.2.3 Delay in the Simple SRAM**

## 3.3 Structural Variations in SRAMs

- **3.3.1 Memory Squaring**
  - Power Consumption in Memory Squaring
  - Delay in Memory Squaring
- **3.3.2 Divided Word Line**
  - Power Consumption in the Divided Word Line Structure
  - Delay in the Divided Word Line Structure
- **3.3.3 Hierarchical Word Line**
- **3.3.4 Partitioned Memory Architecture**
  - Power Consumption in the Partitioned Memory Architecture
  - Delay in the Partitioned Memory Architecture
- **3.3.5 Hierarchical Bank Structure**
  - Power Consumption in the Hierarchical Bank Structure
  - Delay in the Hierarchical Bank Structure
3.4 Circuit Variations ............................................................... 34
  3.4.1 SRAM Memory Cell .................................................. 34
     Resistive-Load Cell .................................................. 34
     Full CMOS SRAM Cell ............................................. 35
  3.4.2 Decoders ................................................................. 37
     Simple NOR Decoder .............................................. 38
     Clocked NOR Decoder ............................................ 39
     NAND Decoder ...................................................... 40
     NAND Decoder with Predecoder ............................. 41
  3.4.3 Enable Circuit .......................................................... 42
     Simple NAND Enable ........................................... 42
     Word Line Enable Circuit .................................... 43
  3.4.4 Sense Amplifier .......................................................... 43
     Bitline Decoupled Latch Type Sense Amplifier 44
     Buffered Sense Amplifier ..................................... 45
     Sizing a Sense Amplifier .................................... 45
  3.4.5 Write Circuit ............................................................. 46
     Differential Input Write Circuit ......................... 46
     Single-Ended Write Circuit ............................... 47
  3.4.6 Pull-Up Circuit .......................................................... 48
     Simple Pull-Up Circuit ....................................... 48
     Switchable Pull-Up Circuit .................................. 48
  3.4.7 Buffers ................................................................. 49
     One-Directional Buffers ....................................... 49
     Bi-Directional Buffers ....................................... 50
4 ANALYSIS

4.1 CHOICE OF STRUCTURE FOR THE MEMORY ....................... 51
4.2 CHOICE OF CIRCUITS FOR THE MEMORY .......................... 53
  4.2.1 CELL ........................................................................ 54
  4.2.2 ROW DECODER ....................................................... 54
  4.2.3 ENABLE .................................................................... 55
  4.2.4 SENSE AMPLIFIER.................................................... 55
  4.2.5 WRITE CIRCUIT......................................................... 55
  4.2.6 PULL-UP CIRCUIT .................................................... 56
  4.2.7 BUFFERS .............................................................. 56
    ADDRESS AND CONTROL BUFFERS ....................... 56
    DATA BUFFERS ......................................................... 57
4.3 DEFINITION OF THE STRUCTURE LIMITS ....................... 57
5 IMPLEMENTATION

5.1 THE STRUCTURAL IMPLEMENTATION ........................................... 59

5.1.1 CREATING BASIC PARTS .................................................. 60
5.1.2 CREATING A BLOCK OF PARTS ....................................... 61
5.1.3 CREATING A PARTITION OF BLOCKS OF PARTS ............. 61
5.1.4 CREATING BRIDGES ...................................................... 62
5.1.5 CREATING SEGMENTS .................................................... 63

BASIC BLOCK .......................................................... 64
SEGMENT 1 ............................................................... 65
SEGMENT 2 ............................................................... 65
STANDARD VERTICAL SEGMENT .................................. 66
STANDARD HORIZONTAL SEGMENT .............................. 66

5.1.6 CREATING A FLOORPLANNING CELLVIEW .................... 67

ONE-PARTITION MEMORY FLOORPLANNING .............. 67
ONE-BASIC BLOCK MEMORY FLOORPLANNING ......... 68
TWO-BASIC BLOCKS MEMORY FLOORPLANNING ...... 68
TWO-SEGMENT 1 MEMORY FLOORPLANNING ............ 69
STANDARD VERTICAL FLOORPLANNING .................. 69
STANDARD HORIZONTAL FLOORPLANNING ............. 70

5.1.7 WIRING ............................................................... 71
5.1.8 BUFFER SIZING ...................................................... 71
5.1.9 PLACING DECODED BUFFERS .................................. 73

INPUT AND OUTPUT PINS ........................................... 73
BUFFER STRUCTURE ................................................. 73
BUFFER SIZING ...................................................... 74
ADDRESS AND CONTROL BUFFERS .......................... 75
DATA BUFFERS ....................................................... 75
PART ARRAY ......................................................... 75
WIRING WITHIN BUFFERS ........................................ 76
5.1.10 Control Circuit ................................................................. 77
    Timing the Block Enable Signals .............................. 77
    Control Logic on the Memory Enable Signal .... 78
5.1.11 Evaluation of the Structural Implementation .... 79
    Structure in General ............................................ 79
    Input and Timing Circuit ................................. 79
    Buffers ...................................................... 80
    Summarising ................................................... 80

5.2 Implementation of the Circuits .............................. 81
5.2.1 Implementation of the Cell ................................... 81
    Description of the Cell Implementation ............... 81
    Evaluation of the Cell Implementation ............ 82
5.2.2 Implementation of the Row Decoder .................... 82
    Description of the Row Decoder ....................... 82
    Evaluation of the Row Decoder ....................... 83
5.2.3 Implementation of the Enable Circuit ............... 83
    Description of the Enable Circuit ................. 83
    Evaluation of the Enable Circuit ................ 84
5.2.4 Implementation of the Sense Amplifier .............. 84
    Description of the Sense Amplifier ............. 84
    Evaluation of the Sense Amplifier ................ 85
5.2.5 Implementation of the Write Circuit ................ 85
    Description of the Write Circuit ................. 85
    Evaluation of the Write Circuit ................ 85
5.2.6 Implementation of the Pull-Up Circuit .............. 85
    Description of the Pull-Up Circuit ............... 85
    Evaluation of the Pull-Up Circuit ............... 86
5.2.7 Implementation of Block Decoders ................. 86
# 6 Conclusions

## 6.1 Implementation versus Requirement Specifications

6.1.1 Memory Size ............................................................... 89
6.1.2 Low-Power ................................................................. 90
6.1.3 Flexibility .................................................................. 90
6.1.4 Memory Control ....................................................... 90
6.1.5 Speed ........................................................................... 91
6.1.6 Chip Area ................................................................... 91

## 6.2 Possible Improvements and Future Work

6.2.1 Necessary Additional Work ............................................ 91
   Buffers .......................................................................... 92
   Timing and Input Circuit .................................................. 92
   Block Decoder ................................................................. 92
   Supply Connections .......................................................... 92
   Testing ........................................................................... 92
   User Manual .................................................................... 92
6.2.2 Improvements to Minimise Manual Intervention ... 92
   Block Decoder ................................................................. 92
6.2.3 Improvements on the Actual Circuits ......................... 93
6.2.4 Comments on Future Work .......................................... 93

## 6.3 Problems Encountered During Implementation

## 6.4 Final Words

# References

# Appendices

## Appendix 1

Inverter Sizing Data

## Appendix 2

List of Code Files
### Table of Figures

3 SRAM - A Theoretical Background

| Figure 3.1 | SRAM cell in its context | 19 |
| Figure 3.2 | Simple SRAM block | 20 |
| Figure 3.3 | Simple structure with column decoding | 25 |
| Figure 3.4 | Divided word line architecture | 27 |
| Figure 3.5 | Hierarchical word line structure | 29 |
| Figure 3.6 | Partitioned memory architecture | 30 |
| Figure 3.7 | Hierarchical bank architecture | 32 |
| Figure 3.8 | General SRAM cell | 34 |
| Figure 3.9 | Resistive load SRAM cell | 35 |
| Figure 3.10 | Full CMOS SRAM cell | 36 |
| Figure 3.11 | Simple NOR decoder | 38 |
| Figure 3.12 | Clocked NOR decoder | 39 |
| Figure 3.13 | NAND decoder | 40 |
| Figure 3.14 | NAND decoder with predecoder | 41 |
| Figure 3.15 | Simple NAND enable | 42 |
| Figure 3.16 | Word line enable circuit | 43 |
| Figure 3.17 | Bitline decoupled latch type sense amplifier | 44 |
| Figure 3.18 | Sense amplifier in a context | 44 |
| Figure 3.19 | Buffered sense amplifier | 45 |
| Figure 3.20 | Differential input write circuit | 47 |
| Figure 3.21 | Single-ended write circuit | 47 |
| Figure 3.22 | Pull-up circuits | 48 |
| Figure 3.23 | Bi-directional buffer | 50 |
5 IMPLEMENTATION

FIGURE 5.1 BASIC PART STRUCTURE ........................................... 60
FIGURE 5.2 BASIC PARTITION STRUCTURE ................................. 62
FIGURE 5.3 BRIDGE ................................................................... 63
FIGURE 5.4 DIVIDED BASIC BLOCK............................................. 64
FIGURE 5.5 WHOLE BASIC BLOCK.............................................. 64
FIGURE 5.6 SEGMENT 1 ............................................................. 65
FIGURE 5.7 SEGMENT 2 ............................................................. 65
FIGURE 5.8 STANDARD VERTICAL SEGMENT .............................. 66
FIGURE 5.9 STANDARD HORIZONTAL SEGMENT ....................... 66
FIGURE 5.10 ONE-PARTITION MEMORY FLOORPLANNING ............. 67
FIGURE 5.11 ONE-BASIC BLOCK MEMORY FLOORPLANNING ......... 68
FIGURE 5.12 TWO-BASIC BLOCK MEMORY FLOORPLANNING ....... 68
FIGURE 5.13 TWO-SEGMENT 1 MEMORY FLOORPLANNING ........... 69
FIGURE 5.14 STANDARD VERTICAL FLOORPLANNING .................. 70
FIGURE 5.15 STANDARD HORIZONTAL FLOORPLANNING .......... 70
FIGURE 5.16 BUFFER STRUCTURE ............................................. 74
FIGURE 5.17 BUFFER ARRAY..................................................... 76
FIGURE 5.18 TIMING OF THE ENABLE SIGNALS........................... 77
FIGURE 5.19 CREATING AN ENABLE PULSE ............................... 78
FIGURE 5.20 MEMORY ENABLE INPUT CIRCUIT .......................... 78
FIGURE 5.21 ROW DECODER FROM SCHEMATIC TO LAYOUT ...... 82
INTRODUCTION

Nowadays, mobile solutions for computers, telephones, and other technical devices are a necessity and, in order to allow for maximal running time, it is necessary for these applications to keep power consumption low. Information needs to be saved in high-speed memories for fast access. The memories in mobile devices often contribute to a large portion of the total power consumption. For this reason, using low-power memories in the design yields a large reduction of the total power consumption. In this thesis, a generator for design and layout of low-power random access memories is implemented.

1.1 Background

This thesis work has been carried out at the division of Electronics Systems at Linköping Institute of Technology. The division focuses its research on design and implementation of signal processing and communication systems. This research concentrates on digital signal processing (DSP) systems, analog and digital filters, as well as analog, digital, and other kinds of mixed circuits. Low-power consumption implementations and efficient design are pursued.
In DSP systems, random access memories (RAM) are often needed and the division of Electronics Systems needs to have access to low-power RAMs of different sizes. It is within this framework that a RAM generator, i.e., a program that creates a layout of a RAM of a specified size, has been developed. The memories generated by the tool will be embedded on chip together with other circuits.

A top-down design of the memory generator has been applied during the project and this design flow is discussed in Section 2.4.

The reader of this thesis is expected to have the basic technical knowledge acquired during a technical master program as well as basic knowledge of electronic design.

### 1.2 Requirements Specifications for the Memory Generator

The requirement specifications for the RAM generator are derived from the requirements on the memories it should generate:

#### 1.2.1 Memory Size

- **Requirement 1** - The number of words of the generated memories shall be flexible.
  
  ✓ The number of words can only be powers of two, that is 2, 4, 8,..., in order to simplify the addressing of the memories and their architecture.
  
  ✓ The range of words may vary from 2 to 10k but the memory generator will be optimised to work best when designing memories in the lower part of this range, that is, up to a few k-words.

- **Requirement 2** - The number of bits per word of the generated memories shall be flexible.
  
  ✓ The number of bits can take all possible values between 8 and 48.
1.2.2 **Low-Power**

- **Requirement 3** - The generated memories shall be designed as a low-power implementation.

1.2.3 **Flexibility**

- **Requirement 4** - The basic parts used by the generator in the memories shall be easily interchangeable in order to be able to update the produced memories to new technologies and with better structures for the parts. This puts a requirement on the modularity of the generator.

- **Requirement 5** - The memories shall be optimised to drive a specified load at the memory data I/O of up to 1 pF.

1.2.4 **Memory Control**

- **Requirement 6** - The generated memories shall have the following control inputs:
  - ✓ Enable signal - clocks the reading/writing cycles and enables the memory.
  - ✓ Write signal - high means writing to, low means reading from the memory.
  - ✓ Address bus.

- **Requirement 7** - The generated memories shall have the following input/outputs:
  - ✓ Data bus.

- **Requirement 8** - The timing of the generated memories shall be controlled simply by an enable signal as follows:
  - ✓ When the enable signal has a high flank, the reading/writing of a new word starts.
  - ✓ The complete cycle of the enable signal must be at least as long as the memory read/write time.
  - ✓ Addresses, write signal, and data are required to be stable on the input before the high enable flank arrives and until shortly before the next high flank.
When the enable signal is low and an already started cycle is finished, the memories shall be disabled.

1.2.5 Speed

- **Requirement 9** - Speed requirements on the designed memories were not clearly specified from the beginning of the project. The only formulated requirement has been that the generated memories shall be acceptably fast, which towards the end of the project turned out to mean a reading/writing time under 20 ns, implying that the memories shall be used for a speed of up to 50 MHz.

1.2.6 Chip Area

- **Requirement 10** - No area limits have been imposed on the generated memories. Nevertheless, a well-designed and compact layout is desirable since the size of the memories influences the power consumption.

1.3 Limitations and Further Requirements on the Project

The time constraints dictate certain limitations on the project and these are summarised here:

- **Limitation 1** - The basic parts of the memories do not need to be flexible, i.e., they are the same regardless of memory size. Their sizing is therefore supposed to work acceptably well on the largest possible memory. This means some inefficiency with respect to power consumption, area, etc.

- **Limitation 2** - As discussed in Section 1.2, the basic parts shall be replaceable. For this reason, no direct requirements are put on them other than that they must be able to be switched off when they are not used in order to keep down power consumption.

- **Limitation 3** - The input and control circuit of the memory does not need to be automatically generated. The circuit parts shall be available and only one timing proposal shall be done that, hopefully, works for the largest block. Fine tuning might be necessary in order to get full functionality.
and keep power consumption down.

- **Limitation 4** - The memory generator does not need to produce a complete layout. Some manual interventions might be necessary as well as some testing.

Limitations given by the available tools are discussed in Section 2.1.

Apart from requirements on the generated memories and on the memory generator, there are some requirements on program coding.

- The code must be well-commented and well-structured in order to allow for future improvements.
- The code must be as modular as possible in order to allow for certain substitutions.

### 1.4 Overview

In this section, an overview of the thesis is given.

In Chapter 2, the approach to the project is discussed, which includes the description of the top-down design flow. Technical tools and features used in the project are discussed. Furthermore, the used programming language, SKILL, is presented. The chapter ends with a description of the main features of the used layout process, Thompson’s 0.18-µm HCMOS8D process, which influence the memory design.

In Chapter 3, a theoretical background relevant to the implementation is given. The chapter begins with an introduction to Random Access Memories (RAM), and to Static RAM (SRAM) in particular, in Section 3.1. A simple structural SRAM model is described in Section 3.2 and variations to that model and their influence on delay and power consumption in Section 3.3. The chapter ends with a description of different circuits needed in an SRAM in Section 3.4.

In Chapter 4, the theoretical background of the preceding chapter is interwoven with some practical outcomes from testing and simulation of different memory parts. The discussion aims at the choice of a structure for the memory in Section 4.1 and at the choice of circuitry for the memory in Section 4.2.
Chapter 5 focuses on the implementation of the memory. The implementation of the structure of the memory is described and then evaluated in Section 5.1. In Section 5.2, the same is done for each of the sub-circuits in the memory. The chapter ends with a description of how analog problems in the generated memories are handled.

Chapter 6 concludes the thesis by, in Section 6.1, comparing the implementation with the requirements discussed in Section 1.2. Possible deviations and their causes are also discussed here. Possible errors and problems present in the generator at the time of writing are highlighted in Section 6.2. Suggestions on what could and/or should be done to improve the memory generator are found in the same section. In Section 6.3, the problems encountered during the implementation phase are treated.
In this chapter, we introduce the reader to the methodology used in this work. The chapter starts with a description of the tools and their features that are relevant for the work, in order for the reader to get an idea of what possibilities and restrictions were available during the work. The chapter ends with a description of the design steps in the project.

2.1 CADENCE AND ITS FEATURES

The tool used in this project has been a Cadence electronic design package, as it is the standard tool used at the Electronics Systems division for layout and as Cadence is one of the major companies delivering platforms for electronic design. In this thesis, the package will be referred to as Cadence. In this part, the different features and limitations of Cadence of interest for the project are presented.
2.1.1 Views in Cadence

In Cadence a circuit can be described in many different ways. Those of relevance for this work are presented in the following.

**Schematic View**

A netlist is a list where elements used, e.g., transistors, wires, capacitances, resistances, and pins, are included as mathematical models with specific connection points. Such a netlist is in Cadence shown graphically in a schematic view, where all models are symbolically represented. It is also possible for the user to create parts and include them in the design, which allows a hierarchical structure.

**Layout View**

A layout view specifies several layers of different materials, with a given shape, size and position. These specifications are followed when fabricating a chip. The choice of layers and shapes depend on what logical/analog function the chip is meant to have. Transistors are built by placing specific layers onto each other in a specific way. The placed objects can be given a name and each of them gets a so called *database number* by Cadence so that they can be uniquely identified by the program.

In order for the layout to work when manufactured, some given rules called *design rules* need to be followed. These rules depend on the process used, even though many rules are similar in different processes.

As in the schematic view, we can produce one part and insert it with varying orientation into other layout views, hence an object can be instantiated as many times as needed.

There are also layers that are not translated into real layers when the chip is produced. One example is the “align” layer, which can be used as a boundary for a layout block. When a layout is designed, an “align” rectangle can be placed around it and, for example, be used as a reference when aligning two parts to each other. If used properly, two objects can be placed next to each other without breaking any design rules. In this way, objects made of simple “align” rectangles can be used while designing complex systems where it is not exactly known what the layouts of the objects look like.
Other aligning references can also be used. A cell may be designed with its $V_{DD}$ and GND pins at a fixed place of the cell so that terminals of different cells may be aligned to each other.

All “real” layers have resistive and capacitive characteristics so the layout will differ from an ideal schematic view because of its parasitic capacitances and resistances.

**Extracted View**

Cadence allows translation of a layout into a netlist. This process is called extraction. The parasitics of choice found in the layout can be included in the netlist. The parasitics extraction possibility is of extreme interest in the memory generator, which will become clear in Section 3.2. For very large layouts, as a memory layout often is, an extraction can require much computing time. Cadence is not stable on such large layouts and, if run on computers with insufficient memory, the computer can crash.

**Analog Extracted View**

The extracted view can be translated into an analog extracted view by Cadence, which is in many ways very similar to the extracted view but is better suited for simulation.

**Configuration View**

A configuration view is a copy of the schematic view where all parts found in the view can be instanciated in different views. For instance, a part can be instanciated as schematic, another as analog extracted, etc.

**2.1.2 Working with Cadence**

Normally, while designing a chip layout, we are interested in making sure that the logical circuit works correctly. This is done by simulating the schematic circuit. The simulation tool used for this thesis work is called Affirma Analog Circuit Design Environment. After a layout view of the circuit is created, a design rule check (DRC) can be done to check whether the layout follows the given design rules. If the layout passes this check, an extraction into an extracted view can be done. Cadence also provides the possibility to make a so called Layout Versus Schematic (LVS) check. This check compares the extracted netlist with the schematic netlist and checks if the logical function
of the layout is the same as for the schematic. Both DRC and LVS are very powerful tools, but, especially in LVS, error messages are sometimes misleading, and finding the real errors can be very time consuming.

When LVS gives a correct answer, a backannotation of the parasitics back into the schematic view can be done. This means that the values of the parasitics at each pin can be shown in the schematic view.

### 2.1.3 Simulation Possibilities

The Affirma simulation tool has a large amount of useful features. The most important one is the possibility to simulate the design taking into consideration all parasitics found in the schematics by simulating the analog extracted view. It is possible to simulate an analog extracted view taking into account the analog characteristics of a layout. This kind of simulation is often more time consuming than the simple simulation of a schematic view. A configuration view can be simulated containing different views for each part.

### 2.1.4 How To Create a Circuit

In Cadence a circuit can be created either manually or by a program. In the following both solutions are discussed.

**Manual Creation of a Circuit**

A circuit in Cadence can be created manually by adding and positioning each part by hand. The generation of a cell by hand in the layout view means that every database object has a certain coordinate value. Moving a database object might introduce DRC errors and with them the need of manually moving other database objects. As long as we do not expect cells to change and the layout is not repetitive, layout done by hand can be a good option.

**Programming a Circuit**

Cadence also offers the possibility to create circuits by writing a program code. This can either be done as simple commands in the command window or by writing a program. The programming language used is called SKILL. A short description is given in Section 2.2. Programming can be done on schematic circuits as well as on layout circuits. Here, only layout programming is considered, as no schematic programming has been done in this thesis work.
When schematic needed to be generated in order to test some parts of the memory, the generation was done with an already existing program.

Programming layout gives the user one extra feature compared with drawing. It is possible to place objects relative to other objects. This means that if the program places the first object in some other place, it will place the next object in another place too, according to how it is meant to be placed relative to the first object. It is possible to force the alignment to be updated in the already given layout when a change in it is done. This function is, however, not stable and causes Cadence to crash. The only alternative is to regenerate the layout modifying the first object there and replacing the second according to the first one.

Of course, it is also possible to program in terms of direct coordinates too. This kind of programming works well as long as changes in the direct coordinates are not done. Changing coordinates often causes Cadence to crash. For this reason, such kind of programming should be avoided when possible.

Using other programming platforms, it is common programming practice to pre-define all global constants instead of inserting them everywhere in the program. The user/programmer must anyway be aware of that, if this method is used in Cadence, changes in such constants between runs of the program often cause the program to crash.

### 2.2 A Short Description of SKILL

The Cadence package uses SKILL as a working language. SKILL is a variation of LISP, improved with some C-commands and features required for hardware description, which are the most used commands.

Hardware description commands in SKILL for layout are based on one out of two kinds of objects. As we mentioned in Section 2.1.1, every object in each layout has a database number, so each object can be seen as a database object. This number is given to objects no matter whether they are created manually or by programming. Some SKILL functions use database numbers to manipulate objects. Some kinds of database objects, such as pins, can take connectivity names, and some, such as wires, cannot. When an object is programmed onto a view, it is also catalogued as a rod object and gets a rod object name. The rod object name is assigned to the object automatically if the user does not choose to assign one to it. The only objects used in the
layout of the memories which do not get a rod object name are contacts. Note that standard high-speed transistors placed by hand get rod object names as they are parametrised cells and have been programmed when they were created. A rod object can be handled with a series of rod object functions, such as aligning functions and functions that return the coordinates of an object, which are often very useful.

Sometimes objects used in standard cells, as for instance drains and sources in some transistors, do not have rod object names, which implicates that they cannot be addressed by rod object name, and no rod object functions can be applied to them. When required, a name can be assigned to them by a command, which is not always a trivial process.

2.3 PROCESS DESCRIPTION

A chip can be produced in different processes with different characteristics. In newer processes the geometries are often smaller than in older processes. Transistors as well as wires become smaller and in this way the same amount of circuitry can be placed onto a much smaller chip area. Furthermore, the number of possible metal layers seems to grow with new technologies.

For this project the process used is a CMOS 0.18-μm, but, as specified in Section 1.2, the generator is meant to be able to be suited to newer processes as well, with few changes to the structure and by substituting the basic parts. There is reason to believe that newer processes will tend to develop in the same direction as for 0.18-μm compared to the older 0.35-μm, so hopefully the assumptions taken in this project will still be valid when changing the process.

The 0.18-μm CMOS process used here is Thompson’s technology. The number of metal layers available here are six. The lowest layer has the highest resistivity and capacitivity, as it lies closest to the substrate, while the highest has the lowest. It is for this reason good to reserve the two highest levels for positive and negative supply and use middle level layers for long signal paths.

The natural choice for the project has been to choose metal 5 for long distance GND and metal 6 for long distance V_{DD} paths. Metal 1 has been used for short distance GND and V_{DD} in order to reduce the number of contacts in the cells.
The low level wires can be as thin as 0.32 µm, which means that the resistances of the wires can be relatively large. An even higher resistance is found in poly wires, which may be as small as 0.18 µm. In this process capacitances between wires in the same layer are much larger than between wires in different layers. If we have two wires of the same layer next to each other the total capacitance can become nearly twice as large as for a single one. This means that busses with wires on minimum distance from each other have very large capacitances. On the other hand, two parallel wires of two different metals do not influence capacitance notably. Many factors influence the final values of capacitances and resistances, so it is not an easy task to calculate a resistance or capacitance on a wire without extracting it.

On the contrary to what used to be true for older technologies, the gate capacitances for transistors have become very small. When connecting a gate to a long wire, the gate capacitance can practically be neglected.

The minimum transistor length, $L_{\text{min}}$, is 0.18 µm and the minimum width, $W_{\text{min}}$, is 0.28 µm. No easily accessible information about transistor models is available, which is a major problem when calculating the right size for a transistor in a circuit based on a given capacitance. The needed information has been found by simulation of an inverter with a load on the output. The speed results from this simulation are found in Appendix 1. The n/p-ratio is not linear for the transistor as the ratio between rise and fall time varies with the load capacitance but its value is in most cases smaller than 3 and can be approximated to 3.

### 2.4 Procedure

As mentioned in Chapter 1, a top-down design approach has been striven for during this project. Sometimes some detours from the top-down approach have been taken in order to increase the understanding of the relationship between different factors.

#### 2.4.1 Steps Towards a Memory Generator

The process towards a memory generator can be seen as a series of steps, described in the following.
STEP 1 - LITERATURE STUDY

A literature study was the basis for the whole work. Here, a large amount of information on implementation of memories was collected.

STEP 2 - HANDS-ON ACQUISITION OF KNOWLEDGE ON SRAMs

In order to increase the knowledge on memories and get some hands on experience, a schematic and layout draft of the main circuits was done by hand. Extractions and simulations on analog extracted views were done and the theories took a more concrete form. No effort was put into sizing circuits properly. The only requirement on the layouted circuits was that DRC and LVS would give no errors so that an extraction would be possible. The circuits were also tested as whole rows and columns of different sizes. This stage was not in accordance with the top-down approach, but at the time it was deemed necessary in order to get an approximate idea on dimensions, power consumption, etc., due to the fact that memories do not lie in the main knowledge field at the division of Electronics Systems and that the author did not have any previous experience of working in Cadence or with the 0.18-μm process.

STEP 3 - CHOICES FOR MEMORY DESIGN

Combining the theory with the test results from the first and the second step, it was possible to choose the general structure of the memory.

STEP 4 - FLOORPLANNING

A floorplan for the whole memory was proposed and programmed in layout using only empty rectangles of “align” layers. It was at this point that a decision to where all circuits would be placed was taken and the layouted memories were only seen as composed of blocks. A preliminary capacity for the blocks was decided. The only requirement on the block was that it may not be wider than its height. Also busses were seen as “align” rectangles.

Afterwards, the blocks were floorplanned, that is, an “align” rectangle was placed for each block of the circuitry. At this point no logical testing was possible.

STEP 5 - PLACING PARTS

The blocks were then filled by the program with the preliminary designed parts. Still no effort was put in sizing them or have them properly designed.
They were only seen as black boxes with a name. The placing of buffers and busses in the bus rectangles was planned as well as that of the timing circuit.

**STEP 6 - REFINING THE CIRCUITY**

The contents of the black boxes were refined and circuits sized and designed so that all pieces would fit together without giving DRC or LVS errors. Buffers were designed and placed, and wires were drawn.

**STEP 7 - TIMING PHASE**

When all circuitry was in place, testing was aimed at finding out a proper timing, and a control circuit was designed. At this point, parasitics played an extremely important role and each block was simulated as schematic as well as analog extracted views. A configuration view with a model of the whole memory block was created and some circuit parts were substituted with their parasitics in order to simplify the circuit. A temporary timing was found with a schematic simulation and a more accurate was found with an analog extracted simulation. At the same time a draft of a flexible timing circuit was designed.

**2.4.2 COMMENTS ON THE STEPS**

All steps, apart from step 2, followed a top-down approach. That step would not have been necessary if the knowledge of memories and tools had been better to begin with. It was a very time consuming step because of stability problems with Cadence when extracting large circuits (see Section 2.1.2). In retrospect, modelling should have been used more at this stage as the information obtained from not-properly designed circuits is not precise anyway. That type of analysis could have been done in step 7. A computer with a larger memory would have increased the design efficiency. In the later stages of the project a computer with larger memory was used, which resulted in a big improvement.
SRAM - A THEORETICAL BACKGROUND

3.1 AN INTRODUCTION TO RANDOM ACCESS MEMORIES

A memory that stores data and permits its modification as well as its retrieval is a read-write memory (RWM) [9]. Ideally, access to data takes the same amount of time no matter what stored data is being accessed. For this reason a commonly used definition for this kind of memory is Random Access Memory (RAM). Even if the term RAM is suitable for a variety of memories, it is traditionally exclusively used to define RWMs. A RAM is volatile, that is, the data stored on it is lost when the supply voltage is switched off [13].

3.1.1 PERFORMANCE MEASURES FOR RAMS

The performance of a RAM can be characterised by a number of properties. In the following section some of these properties will be presented.
One important characteristic of a memory is the amount of data which can be stored. This can be measured in the amount of *words*, \(n\), that is, a basic entity which can be addressed. A word consists of \(m\) *bits* [13]. The amount of words multiplied by the word size, \(n \cdot m\), yields the memory’s storage capacity in bits.

It is also of interest to know the number of *input and output ports*. Ports are often bi-directional, but for memories with high bandwidth requirements different ports are used for each direction.

The speed of the memory is often a fundamental characteristic and can be measured as [13]:

- **Read Access Time** - the time it takes to retrieve data measured from the moment when the read request is done to when the data is stable at the output.
- **Write Access Time** - the time it takes to store data measured from the moment when the write request is done to when the data is finally stored in the memory.

Ideally, for a RAM these two measures coincide.

- **Cycle Time** - the minimum time required between successive reads and writes. Normally, this time exceeds the read/write access times.

### 3.1.2 Static and Dynamic RAMs

In some memories, the data is stored and indefinitely retained in the cell as long as the supply voltage is on [9]. In this case, we have a *Static Random Access Memory (SRAM)*. In other memories, where the data vanishes from the cell with time, the cell contents have to be refreshed, i.e., rewritten into the cell, regularly in order to keep data. Such a memory is called *Dynamic Random Access Memory (DRAM)*. The main difference between these two memories is how the cells are designed.

A DRAM cell is more compact than an SRAM cell as it stores its value onto a small capacitance in the cell. However, in order to get a good compact DRAM capacitor cell, a special process is required. In this project, a standard CMOS process is used. Hence, the SRAM is the only alternative. In the rest of this thesis, only SRAM memories will be discussed, but much of what is written here also applies to DRAMs.
3.1.3 Operation of an SRAM Cell

The basic structure of an SRAM data storage cell consists of a simple latch circuit with two stable operating points. These two points store the states of the cell, or the value of the bit. Two switches controlled by a so called \textit{word line} (WL) connect the two points to two complementary bit lines, which throughout this thesis will be called \textit{bit line} (BL) and \textit{bit line} (\overline{BL}). When the WL is low, the cell is keeping a value. When reading or writing, that is, while the cell is active, the WL is high. A generic structure for an SRAM cell is shown in Figure 3.3.

During its static operation BL and \overline{BL} are kept at a given voltage, a pre-charged level. When the cell stores a ‘0’, the input node of the upper latch is low and the one of the lower latch is high. When the cell holds a ‘1’, the input node of the upper latch is high and the input node of the lower latch is low.

When writing a ‘0’ to the SRAM cell, the BL is forced to a logic low by the write circuit. When writing a ‘1’, it is the \overline{BL} that is forced to a logic low. When reading a ‘0’, the cell pulls the BL down, while when reading a ‘1’ the cell pulls the \overline{BL} down.

The only difference is that when writing it is the writing circuit that does the work and when reading, it is the cell [9].
3.2 A SIMPLE MODEL FOR SRAMS

In this part, a simple model of an SRAM is presented. This model is very similar to how small SRAMs are designed, but is not realistic for larger memories. However, it constitutes a good starting point for understanding how memories work. First, a structural description is given and then the sources of power consumption in an SRAM are described.

3.2.1 STRUCTURAL DESCRIPTION OF A SIMPLE SRAM

In its simplest form, a memory core can be seen as an array consisting of \( n = 2^N \), where \( N = 0, 1, \ldots \) rows and \( m \) columns, as in Figure 3.2. Every element of this array corresponds to a cell, where one data bit is stored.

Every row corresponds to a word and all cells on the row are connected to each other by a WL. Only one word line at the time can be activated. Each column is connected to one BL and one \( \overline{BL} \). As the wires are connected to each cell, apart from their own capacitance and resistance due to their length, they will also get a capacitance from each cell.

Figure 3.2 - Simple SRAM block, adapted from [6], p. 429, and [11], p. 240
Each BL and BL has a given stand-by charge, and the circuit responsible for keeping the charge constant at that value while the array is not used is the so-called pull-up circuit.

One row at a time is selected by entering an address composed of \( N \) bits. This address is then translated by a row decoder which activates a specific row. Without such a row decoder, the amount of pins necessary to address each row would soon be too cumbersome, or even impossible to put into practice.

While reading from a cell, the signals on the BL and BL are amplified by a sense amplifier, which basically is a differential amplifier with BL into one input and BL into the other. In this simple memory model, there will be one sense amplifier for each column. The function of a sense amplifier can be to speed up the reading of data or to convert a differential signal into a non-differential signal.

While writing into the cell, a signal is put onto the BL and BL by a write circuit. In this simple memory model, there will be one write circuit for each column.

The signals retrieved from the cells or to be stored into the cells are transmitted to/from the input/output ports on input/output lines (I/O lines). For larger memories the degradation of the signal on the lines can be large and the need for I/O drivers increases. Also degradation on address signals need to be alleviated and this is done by address buffers. Buffers occupy a large chip area. Furthermore, some input circuit, that also times up the memory, might be required. Neither buffers nor input circuit are shown in Figure 3.2.

### 3.2.2 Power Consumption in the Simple SRAM

In order to study the dynamic power consumption in a RAM, we can fit a simplified model of the memory presented by Itoh et.al. [12] to the simple memory above. The simplified SRAM can be divided into three blocks: a memory cell array, decoders and periphery, where the periphery corresponds to all circuitry which is not directly included in the other two blocks, e.g., drivers and buffers. We assume that all cells on a selected row in the array are active, but that only one row at a time is activated.

The general power formula is

\[
P = V_{DD} \cdot I_{DD}
\]  

(3.1)
where \( V_{DD} \) is the operating supply voltage for the memory and \( I_{DD} \) is the sum of the currents in the memory. We are going to analyse the different terms of \( I_{DD} \) during a read cycle.

We start by analysing the array block. During the read operation, there is a current through the active cells, \( i_{\text{active}} \). The total current through the active cells is \( m \cdot i_{\text{active}} \), where \( m \) is the number of bits in a word. Furthermore, we have a retention current in the inactive cells, \( i_{\text{hold}} \). The number of inactive cells is given by the number of inactive rows, \( n - 1 \), multiplied by the number of columns in the array, or the number of cells in a row, \( m \). This implies that the total retention current is \( (n - 1) \cdot m \cdot i_{\text{hold}} \). The total current in the array is then

\[
I_{\text{array}} = m \cdot i_{\text{active}} + (n - 1) \cdot m \cdot i_{\text{hold}} \tag{3.2}
\]

Note that, normally, \( i_{\text{active}} \) is much larger than \( i_{\text{hold}} \).

The decoders have a given capacitance at their output node, \( C_{\text{rowdec}} \), and a supply voltage, \( V_{DD} \). The current increases with the operating frequency of the memory, \( f \). The higher the operating frequency, the larger the current in the decoder blocks. The frequency is the inverse of the cycle time, \( t_{\text{cycle}} \). In the simple memory above, there are \( n \) row decoders. The total current through the decoder blocks is then

\[
I_{\text{rowdecs}} = n \cdot C_{\text{rowdec}} \cdot V_{DD} \cdot f \tag{3.3}
\]

The periphery block has in its simplified form a total capacitance, \( C_{\text{periphery}} \) and a supply voltage \( V_{DD} \). The current depends on the frequency, \( f \). Furthermore, there is a static, or quasi-static, current, \( I_{\text{DCperiphery}} \), which is due to sense amplifiers, write circuits, etc. Essentially, \( I_{\text{DCperiphery}} \) does not depend on \( f \), and it can be neglected for high frequencies. The total periphery current is then

\[
I_{\text{periphery}} = C_{\text{periphery}} \cdot V_{DD} \cdot f + I_{\text{DCperiphery}} \tag{3.4}
\]

\( I_{DD} \) is

\[
I_{DD} = I_{\text{array}} + I_{\text{rowdecs}} + I_{\text{periphery}} \tag{3.5}
\]

Insertion of equation (3.2), (3.3), (3.4) in (3.5) gives
In this thesis we will be concerned with how to reduce the different parts of $I_{DD}$. The formula above implies that the current increases with increased size of the memory, as some terms depend on $m$ and $n$.

### 3.2.3 Delay in the Simple SRAM

When we analyse the delay in this simple RAM, we can distinguish two contributing paths: the address path, i.e., the path from the address input to the WL, and the data path, i.e., the path from the memory cells to the I/O ports of the memory.

The address path goes through address buffers, decoders and finally gets to the WL. The delay on this path hence depends on the delay of the buffers, the delay of the decoders, and the delay on the WL.

The data path goes from the cell through the BL and $\overline{BL}$ to the sense amplifiers, and onto the I/O lines. I/O drivers can also be found on the data path. The delay on this path therefore depends on the delay on the BL and $\overline{BL}$, in the sense amplifier, on the I/O lines, and of the drivers.

When the size of the memory increases, the number of rows and/or columns in the array increases. This means that there are more cells on each WL and BL/$\overline{BL}$ and their length grows. At the same time, every cell adds capacitance on the line. Furthermore, the smaller the width of the wire, the larger the resistance of the wire becomes.

According to the RC-line model presented in Dally and Poulton [7], the delay on such a line is

$$\tau_{RC-line} = 0.4 \cdot l_{line}^2 \cdot R_{line} \cdot C_{line}$$  \hspace{1cm} (3.7)

where $l_{line}$ is the length of the line in $\mu$m, $R_{line}$ its resistance per $\mu$m and $C_{line}$ its capacitance per $\mu$m. This means that the delay will increase quadratically with increasing length and linearly with resistance and capacitance of the line. The value of $R_{line}$ decreases with an increasing width while the value of $C_{line}$ increases with it.

The delay on the WL is then
\[ \tau_{WL} = 0.4 l_{WL}^2 R_{WL} C_{WL} \]  \hspace{1cm} (3.8)

and on the BL and BL, assuming symmetry

\[ \tau_{BL} = 0.4 l_{BL}^2 R_{BL} C_{BL} \]  \hspace{1cm} (3.9)

For very large memories, built according to the conventional model presented in this thesis, the delay on the WL and BL/BL would become too large and their speed performance too low.

### 3.3 Structural Variations in SRAMs

In this section, variations to the simple model are described. The power consumption model proposed by Itoh et.al. [12] is adapted to suit each structure for easy comparison.

#### 3.3.1 Memory Squaring

Suppose that we have a large amount of words in the memory. The array would then get a too large aspect ratio.

A common practice is to place more than one word on each line, which means that we place \( k = 2^K \) words in one line, where \( K = 1, 2, \ldots \). In this thesis, this structure is called memory squaring. In memory squaring, we can use one sense amplifier and one write circuit for \( k \) words, which increases area efficiency, and the sense amplifier and write circuits may be \( k \) times wider than the cells.

This solution, shown in Figure 3.3, calls for the addition of another circuit, the column decoding circuit. The circuit connects the BL/BL that should be active to the sense amplifiers and write circuits. This circuitry is often composed of a column decoder, decoding the address into the active columns, and
a column select circuit forwarding the signals between the BL/BL of the
selected column and the other circuits [5].

![Figure 3.3 - Simple structure with column decoding, adapted from [5], p. 2](image)

**POWER CONSUMPTION IN MEMORY SQUARES**

The current formulas presented in Section 3.2.2 can be rewritten for this
architecture. The array current becomes

$$I_{array} = m \cdot k \cdot i_{active} + \left(\frac{n}{k} - 1\right) \cdot m \cdot k \cdot i_{hold}$$  \hspace{1cm} (3.10)

The contribution from the hold current hardly increases, but the active current
contribution does increase with a factor $k$.

The decoder current is here composed of the current from the row decoder
and from the column decoder.
\[ I_{\text{decoder}} = \left( \frac{n}{k} \cdot C_{\text{rowdec}} \cdot k + m \cdot k \cdot C_{\text{coldec}} \right) \cdot V_{DD} \cdot f \] (3.11)

While the number of row decoders decreases with a factor \( k \), the total capacitance for the row decoder will be \( k \cdot C_{\text{rowdec}} \). In total, the row decoder contribution is unchanged, but there will be an increased contribution from the column decoder.

The rest of the currents are unchanged, even if the power consumed for driving signals over the I/O lines can increase when their length increases. Summarising, the total \( I_{DD} \) increases with this structure.

**DELAY IN MEMORY SQUARING**

The delay of the column select is added to the data path. The column decoder is not on the critical address path as it works in parallel to the row decoder and will not contribute to the delay. The delays on the lines are modified to

\[ \tau_{WL} = 0.4 \cdot (l_{WL} \cdot k)^2 \cdot R_{WL} \cdot C_{WL} \] (3.12)

\[ \tau_{BL} = 0.4 \cdot \left( \frac{l_{BL}}{k} \right)^2 \cdot R_{BL} \cdot C_{BL} \] (3.13)

As long as \( k \cdot l_{WL} \cdot R_{WL} \cdot C_{WL} \) is less than \( l_{BL}/k \cdot R_{BL} \cdot C_{BL} \), the total delay on the lines is decreased by this solution compared to the simple model. Note that if the size of the memory increases, the delay on the I/O lines will increase.

**3.3.2 DIVIDED WORD LINE**

If many cells are connected to a WL, the WL might become very long and the total resistance and capacitance may become very large. If we divide the WL into \( n_b \) smaller WLs, so called *local word lines* (LWL), and connect them all onto one long, wider, low resistive, and low capacitive WL, a so called *global word line* (GWL), via a so called *block decoder*, as shown in Figure 3.4, we reduce the load the row decoder has to drive as well as the total resistance over the WL. Only the cells on one LWL are active at the same time. This solution results in a *divided word line architecture* and has a two-level
hierarchy. Its simplest form is obtained by dividing a WL into two LWLs and placing the global row decoder in the middle [10].

![Divided word line architecture](image)

**Figure 3.4 - Divided word line architecture, adapted from [10], p. 250**

**POWER CONSUMPTION IN THE DIVIDED WORD LINE STRUCTURE**

The current formulas will be modified for the divided word line structure in the following. The array current is

\[
I_{array} = m \cdot i_{active} + m \cdot \left( \frac{n}{n_B} - 1 \right) + \frac{n}{n_B} \cdot (n_B - 1) \cdot i_{hold} \quad (3.14)
\]

where the first part is due to the active cells, one row in one array, the second part is due to the hold current for the inactive cells on the active LWL and the last is due to the inactive cells in the inactive LWL. Simplified

\[
I_{array} = m \cdot i_{active} + m \cdot (n - 1) \cdot i_{hold} \quad (3.15)
\]

The array current is hence unchanged compared to the simple model.

\[
I_{decoder} = \left( \frac{n}{n_B} \cdot C_{GWL} + n_B \cdot C_{LWL} \right) \cdot V_{DD} \cdot f \quad (3.16)
\]

where \(C_{GWL}\) is the load capacitance on the global row decoder and \(C_{LWL}\) is the load capacitance on the local row decoder, which is about as large as \(C_{rowdec}\).
A suitable layer and width for GWL to keep $C_{GWL}$ small must be used. However, as not the cell but only the decoders are connected to it, this capacitance will be much smaller than $C_{WL}$. The part of the current due to the local row decoders in the formula is only valid if only the decoders on one row of the active GWL consume current. Otherwise, it must be multiplied with $n/n_B$. In the first case the current due to the decoders will clearly be reduced.

For the periphery currents the same reflections done in memory squaring are valid.

**Delay in the Divided Word Line Structure**

The address path passes through two decoders rather than one as in the simple model SRAM: the global word decoder and the local word decoder, which increases the delay on the address path.

Furthermore, the delay on the total WL (GWL and LWL together) is also increased compared to the simple model.

$$
\tau_{\text{TWL}} = 0.4 \cdot ((l_{GWL})^2 \cdot R_{GWL} \cdot C_{GWL} + (l_{LWL})^2 \cdot R_{LWL} \cdot C_{LWL}) \quad (3.17)
$$

where $l_{GWL}$ is a little shorter than $l_{WL} \cdot n_B \cdot R_{GWL}$ and $C_{GWL}$ are much smaller than $R_{WL}$ and $C_{WL}$. $l_{LWL}$ is about as long as $l_{WL}$ while $R_{LWL}$ and $C_{LWL}$ are about as large as $R_{WL}$ and $C_{WL}$. So the delay on the total WL is similar to the one in the simple SRAM plus the delay on the GWL.

The delay on the bit lines is reduced by this structure to

$$
\tau_{\text{BL}} = 0.4 \cdot \left(\frac{l_{BL}}{n_B}\right)^2 \cdot R_{BL} \cdot C_{BL} \quad (3.18)
$$

As long as a correct $n_B$ for the blocks is selected, the delay reduction on the bit line will be larger than the increase in the total word line delay and in the address path delay through decoders. Even here an increase of delay on the I/O lines might also occur, if the lines become longer compared to the simple case.

**3.3.3 Hierarchical Word Line**

For very large memories, in the order of some megabits, one more level must be introduced in the word line. The architecture obtained is a **hierarchical**
word line structure. There will be a global word line (GWL), a sub-global word line (SGWL) and a local word line (LWL), as shown in Figure 3.5. Here, besides a global row decoder and a local row decoder as in the divided word line architecture, a block group decoder is required. Each GWL is connected from the global row decoder to all sub-global row decoders, while SGWL is connected to each local row decoder, and each LWL to the cells in its array. There is one global, one sub-global and one local row decoder activated in each cycle. The global row decoder gets the row addresses, the sub-global a block group select signal and the local row decoder a block select signal, as shown in Figure 3.5.

For this kind of solution the same observations on power consumption and delay for the divided word line structure apply with the addition of one more level similarly to what was shown in Section 3.3.2.

![Hierarchical word line structure](image-url)

Figure 3.5 - Hierarchical word line structure, adapted from [11], p. 251
3.3.4 Partitioned Memory Architecture

The memory can also be divided into partitions that look much like the simple SRAM described in Section 3.2.1, which results in a partitioned memory architecture. Only one of the partitions can be activated during one cycle. If all circuitry such as sense amplifiers, write circuits, etc. can be switched off when not required, there will only be one active circuit in each cycle. A block decoder will control which block to activate depending on the block addresses. Figure 3.6 represents a partition architecture, where each partition is shown as a dashed rectangle and the block decoders as small rectangles that output the enable signal to the partitions (shown as an arrow in the figure).

\[ I_{array} = m \cdot i_{active} + m \cdot (n - 1) \cdot i_{hold} \]  

(3.19)

Figure 3.6 - Partitioned memory architecture, adapted from [13], p. 558

Power Consumption in the Partitioned Memory Architecture

Similarly to how shown in (3.14) and (3.15), the current consumed by the array will be

\[ I_{array} = m \cdot i_{active} + m \cdot (n - 1) \cdot i_{hold} \]  

(3.19)

The decoder current for a partitioned memory is
where $C_{\text{blockdec}}$ includes all capacitances a block decoder has to drive.

### Delay in the Partitioned Memory Architecture

The delay of the address path in the partitioned memory architecture is increased by the block decoder compared to the simple model. Before each decoding within the block can be done, the partition must first be activated. Afterwards, within each partition, the address path will be the same as for the simple memory model. The delay of the data path will hence become

$$\tau_{BL} = 0.4 \cdot \left(\frac{l_{BL}}{n_p}\right)^2 \cdot R_{BL} \cdot C_{BL}$$

(3.21)

Compared to the simple memory model the delay on the I/O lines can increase if the distances from the input of the memory to that of the partitions is larger than in the simple model case.

Given a length, $l_{IO}$, a capacitivity, $C_{IO}$, per $\mu m$, and a resistivity, $R_{IO}$, per $\mu m$, the value of the delay on the I/O lines will be

$$\tau_{IO} = 0.4 \cdot l_{IO}^2 \cdot R_{IO} \cdot C_{IO}$$

(3.22)

However, the capacitances of the I/O lines are not as large as those on the bit lines, so the increase in delay of these lines will in many cases be compensated for by the decrease in delay of the bit lines.

### 3.3.5 Hierarchical Bank Structure

Another commonly used structure according to Ellis [8] is a hierarchically ordered partitioning of a memory. This structure can be explained with a tree structure. The leaves are banks containing a series of blocks similar to the ones described in Section 3.3.4. These banks are connected two and two. A decoder is placed at each connection and selects one of the two banks. The nodes are then recursively connected two and two with one more decoder until all leaves converge to one and the same node.
All data and address signals start out at the top node and, according to the last address on the address bus, one of the two ways is chosen. The data and address signals will then be driven by I/O buffers into or/and from that direction only. Now a new node is reached and the next last address will decide the direction to take. This process is carried out recursively. At the end only one bank will be reached by the signals on data and address busses.

When floorplanning this structure the blocks will be placed similarly to what is shown in Figure 3.7. A possible path is highlighted with a darker colour.

Figure 3.7 - Hierarchical bank architecture, adapted from [8]
In the figure one partition block from Figure 3.6 is divided into two. The reason for this placement is that in this way the path to each block has exactly the same length because of the symmetry. This allows read and write operations to take approximately the same time for all addresses.

An important factor here is the area. Each driver may require a large amount of area and cannot always be placed down to the lowest level. This is why the basic block for this structure is often not a partition but a block of partitions.

**Power Consumption in the Hierarchical Bank Structure**

The hierarchical bank structure is basically a partition structure, with the only difference that the information on the lines is not forwarded to all partitions and all block decoders, but only to the block of partitions which contains the partition that has to be activated.

The saving in power consumption is due to two factors:

- Only block decoders from one block of partitions is active.
- The power consumed to drive I/O lines decreases, as only the lines going to the active block of partitions must be driven instead of all lines.

**Delay in the Hierarchical Bank Structure**

The delay for this structure is the same as for the partition structure apart from the delay on the I/O lines. For the sake of illustrating how the delay on the I/O lines is influenced, assume that there are a given number of nodes, \( n_N \), and that, ideally, the length between all nodes is the same.

\[
\tau_{IO} = 0.4 \cdot \left( \frac{l_{IO}}{n_N + 1} \right)^2 \cdot R_{IO} \cdot C_{IO} + n_N \cdot \tau_{driver} \tag{3.23}
\]

where \( \tau_{driver} \) is the delay through each driver.


**3.4 Circuit Variations**

In this part of the thesis, only circuits that can be turned off so that they do not consume power when not used for read/write operations are discussed. The main focus is on the memory cell, as it is the circuit central for a memory.

**3.4.1 SRAM Memory Cell**

As described in Section 3.1.3, an SRAM cell is basically composed of two latches. Normally, a latch is built of one n-transistor working as a driver device and one load, as shown in Figure 3.8. How the load is implemented is what characterises the cells described in this section.

![Figure 3.8 - General SRAM cell, [9], p. 418](image)

**Resistive-Load Cell**

One type of cell uses a resistor implemented in polysilicon as a load. Such a cell, shown in Figure 3.9, has only four transistors, which allows the cell layout to become very compact. According to [9], the value of the load
resistance must be kept relatively low in order to avoid too large pull-up times as well as to keep noise margins at an acceptable level.

![Resistive load SRAM cell diagram](image)

The stand-by operation of the cell can be illustrated with the case when a ‘1’ is stored. Then M1 is turned off, while M2 is conducting. This means that the load resistor on the left will have a steady-state current, which corresponds to \( i_{\text{hold}} \). Keeping the current low requires a large resistive load, which implies a trade-off with speed and low-noise performance of the cell. Even if the \( i_{\text{hold}} \) for one cell can be in the order of nA, with a large amount of cells the total static current will be significant [9].

**FULL CMOS SRAM CELL**

In a *full CMOS SRAM cell*, p-transistors are used as loads. This cell has therefore 6 transistors, which affects the area negatively. With new processes, this
has become less of a problem. However, the noise margins for the cell are large and it can work at low $V_{DD}$ [9].

The stand-by operation is illustrated with the case where a ‘1’ is stored. The left point will store a ‘1’ and the right a ‘0’. This implies that M1 is off, M2 on, M5 on and M6 off. In this way, there is no way for the current to go from $V_{DD}$ to GND. As the nodes are charged respectively discharged already, no current is needed for charge and discharge. The only power consumed is due to the leakage current of the transistors, which is very small. Hence, this type of cell is low-power at steady state [9].

The transistor sizing is based on the following criteria: the read operation should not destroy the value stored in the cell, and the cell must allow data to be changed during write [9].

The length, $L$, of the transistors is the minimum length in order to keep the area down. Furthermore, full symmetry in the widths, $W$, of the transistors is required in order to avoid mismatches and errors in the stored values. This means that $W_{M1} = W_{M2}$, $W_{M3} = W_{M4}$ and $W_{M5} = W_{M6}$.

A read operation is carried out. Assume a ‘1’ is stored in the cell. At the beginning, the left operating point is then a ‘1’ and the right a ‘0’. When M3 and M4 are turned on by the WL, the BL stays at its $V_{DD}$ level while the
voltage of the BL is slowly discharged through M2 and M4. Note that the
decrease of the voltage on the BL is not very large during read because of the
large capacitance on the line. The difference in potentials between BL and BL
is in the order of a few mV during reading. If $W_{M4}$ is too large compared to
$W_{M2}$, the voltage at the right operation point might exceed the threshold
voltage of M1, which would cause an unwanted change in the stored data.
During proper operation, the transistor M1 will stay in cut-off mode [9].

Now assume that a ‘0’ is to be written to the cell and the initial stored value is
a ‘1’. As above, M1 and M6 are off, while M2 and M5 operate in linear
mode. The voltage on BL is forced to a logic ‘0’ by the write circuit. When
WL goes high, M3 and M4 are turned on. If the relationship between M1 and
M3 is correctly designed to work in read, the right operation point will
remain below the threshold voltage of M1 and the transistor will not be
turned on. The left operation point on the other hand will have to be driven
down under the threshold voltage of M2 and turn it off. When the left operation
point has reached the threshold voltage for n-transistors, the M3 will
operate in the linear region and M5 in the saturation region. M1 is subse-
quently turned on. When the ratio between $W_{M5}$ and $W_{M3}$ is properly estab-
lished, the writing onto the circuit will be correct [9].

According to [5], a cell can be tested for proper function by putting a voltage
ramp on the WL of a fully symmetric cell, but also of a mismatched cell. If
the cell is stable it will pass the test and work even when mismatches are
present. Such tests in a 0.18-μm process showed that the combinations in
Table 3.1 are the most stable as they passed all worst case mismatches.

<table>
<thead>
<tr>
<th>Alternative</th>
<th>$\frac{W_{M5}}{W_{M3}}$</th>
<th>$\frac{W_{M1}}{W_{M3}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>3.0</td>
</tr>
<tr>
<td>2</td>
<td>1.5</td>
<td>3.0</td>
</tr>
<tr>
<td>3</td>
<td>2.0</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Table 3.1 - Stable sizing of full CMOS SRAM cells in 18-μm process

### 3.4.2 Decoders

Decoders of various kinds are used while designing a memory. This section
focuses on the decoder as a circuit which takes a given number of address
bits, \( n_A \), and returns an enable signal on only one of its \( 2^n \) outputs. Such a decoder can be used as a row decoder, partition decoder, block decoder, etc.

Each output is activated by a certain combination of addresses. For example, if \( n_A = 6 \) and the output to activate is number 10, the binary address will be \( A = (001010) \). In order to get a ‘one’ on output 10, the following function must be implemented

\[
OUT_{10} = \overline{A}_5 \cdot \overline{A}_4 \cdot A_3 \cdot \overline{A}_2 \cdot A_1 \cdot \overline{A}_0 \tag{3.24}
\]

Instead of an AND function, this can be implemented as a NOR-function

\[
OUT_{10} = A_5 + A_4 + \overline{A}_3 + A_2 + \overline{A}_1 + A_0 \tag{3.25}
\]

This means that one function must be implemented for each output [13].

**SIMPLE NOR DECODER**

Implementing a decoder with \( n_A \) address bits implies that, if a NOR-function as in (3.25) is chosen, \( 2^n \) NOR gates with \( n_A \) inputs are needed. A schematic structure of a *simple NOR decoder* for \( n_A = 6 \) and \( OUT_{10} \) is shown in Figure 3.11.

![Simple NOR decoder schematic](image)

**Figure 3.11 - Simple NOR decoder, adapted from [13], p. 592**

There, it becomes clear that the evaluation of the output value is very fast, as the output node can be discharged through as many transistors as the number of input address bits that are high. For the active output the value will be
ready at the output directly after precharge and is ready at the output node before all other outputs [13].

As row decoders are in the critical path of the address path, a fast decoder implementation is important for fast memories.

However, there is a major drawback with this decoder circuit. For each address evaluation cycle, the output nodes need to be precharged by activating the p-transistor. During the evaluation phase, the output nodes of all gates will be discharged apart for the one node that should be enabled, which means that $2^a - 1$ need to be recharged during the next stage. Hence, each low output contributes to the power consumption. Minimising the number of active decoders as well as the output capacitance on each output node is necessary in order to avoid large power consumption.

**Clocked NOR Decoder**

An alternative, proposed by [6], is a *clocked NOR decoder*, where both the p-net and the n-net are clocked, as in Figure 3.12. In this way, the decoder can be deactivated no matter what the address inputs are by keeping the clock low. Short circuits from $V_{DD}$ to GND are avoided by the two complementary clock transistors.

![Figure 3.12 - Clocked NOR decoder, adapted from [6], p. 430](image-url)
In this solution, an inverter is added on the output, yielding an inverted enable signal, and a feedback from it to a p-transistor for speeding up the charging of the node during precharge.

This solution allows for a two-stage decoding. The decoders can be grouped and each group can be activated only when one of the gates in the group is the one to enable.

**NAND Decoder**

Another solution is to choose the function in (3.24) and implement the decoder with \(2^n_A\) NAND gates with \(n_A\) inputs, which results in a NAND decoder. In this case all outputs will be high, except for the active one. A NMOS schematic structure for the NAND solution is shown in Figure 3.13.

There, it becomes clear that the larger the amount of addresses the slower the decoder will be, as the number of transistors in series grows.

The advantage of this solution is that only the active decoder bit will be discharged while all the others keep their charge, which implies that only one bit consumes power.
NAND Decoder with Predecoder

A way to solve the speed problem for a NAND decoder is to reduce the amount of inputs to the NAND gate. The logic function in (3.24) is equivalent to the following one:

\[ \text{OUT}_{10} = (A_5 + A_4) \cdot (A_3 + A_2) \cdot (\overline{A_1} + A_0) \]  

(3.26)

where the pairs of addresses are the input to a NOR gate and the resulting values from each NOR are input to a NAND gate. This makes predecoding possible: all combinations of address pairs, \( A_1 + A_0 \), \( \overline{A_1} + A_0 \), \( A_1 + A_0 \), \( \overline{A_1} + A_0 \), are calculated. Then, each input to a NAND gate is connected to the NOR output that gives the right function for each bit. A NAND decoder with predecoder is shown in Figure 3.14 using logic gates.

Figure 3.14 - NAND decoder with predecoder, adapted from [13], p. 593
Predecoding implies that evaluation of a combination is done only once instead for in each bit, which keeps the number of transistors in the design down. Calculating the number of transistors involved in the evaluation of the function in the simple NAND decoder for the 6-address example, will give $2^6 \cdot 6 = 384$ n-transistors. For a decoder with predecoding in CMOS-technique, there will only be $2^6 \cdot 3 + 4 \cdot 3 \cdot 4 = 240$ transistors.

Halving the number of inputs to the NAND gates reduces the delay through the gate by a factor of four. Furthermore, the capacitance on the address line from the NORs to the NANDs is also reduced, as the load due to the inputs is one of every four NANDs, compared to the capacitance without NOR, where each address sees the load from all NANDs.

One extra feature can be added by including an enable signal to the predecoders. In this way, the whole decoder may be turned off for power saving. Consequently, this solution is used in all large decoders [13].

### 3.4.3 Enable Circuit

In the structures described in Section 3.3, the need for a decoder that allows a signal to be forwarded only if a certain part must be active has become clear. The circuit doing this kind of decoding is in this thesis called an *enable circuit*. Such enable circuits also fulfils the following functions:

- Buffering up a signal on a line.
- Timing the output signal, i.e., even if the original signal is high during a given period, the signal can be kept high during a shorter period, which can be useful for reducing power consumption in some cases.

#### Simple NAND Enable

The simplest enable circuit is a two-input NAND gate where one input is the signal to be forwarded and the other is an enable signal. This simple enable circuit is found in Figure 3.15.

![Simple NAND Enable](image)
WORD LINE ENABLE CIRCUIT

A specific enable circuit for the WL is described in [6] and shown in Figure 3.16.

![Word line enable circuit diagram]

This word line enable circuit gets the row enable signal from the row decoder as input and inverts it onto the WL if the enable signal is low. The enable signal could be the output from a NAND with, for instance, a row select and a block select as input, which in the figure above is shown in the dashed rectangle. No inverter would be needed because of the enable low operation. In this case the signal from the row decoder must be low when the row should be active. This circuit works well with the clocked NOR decoder described in Section 3.4.2.

3.4.4 SENSE AMPLIFIER

Sense amplifiers (SA) are used in SRAM memories in order to speed up the read from the cell. According to [14], the sensing of the data stands for a considerable portion of the total delay and power consumption. As discussed above, while reading from a cell the signal difference between BL and BL is in the order of a few mV and the load on the lines is large. This means that SA must be able to amplify very low swing signals with large capacitances on the input lines.


**Bitline Decoupled Latch Type Sense Amplifier**

One common structure in SRAM literature is the so called *bitline decoupled latch type sense amplifier*. This structure is shown in Figure 3.17.

![Figure 3.17 - Bitline decoupled latch type sense amplifier, adapted from [14]](image)

This SA can detect very low swing, has fast access time and the discharge on the BL/BL can be kept to a minimum. When using this SA, care must be taken not to turn it on before the required voltage has been reached on the bit lines [14].

This SA is put into a context by [5], adding a switch to the output data lines in order to allow for bi-directional functioning of the data lines. This is shown in Figure 3.18.

![Figure 3.18 - Sense amplifier in a context, adapted from [5], p. 73](image)
**Buffered Sense Amplifier**

Another SA structure, which is supposed to work when the ratio between $V_{DD}$ and the threshold voltages of the transistors is low, is the *buffered sense amplifier* proposed by [6] and shown in Figure 3.19.

This kind of SA has circuits on the output which provide for the translation of the differential signal on the bit lines into a low-high data signal and a tristate output for bi-directional functioning of data lines.

This SA is enabled by a low enable signal into $M_5$. As long as $M_5$ is disabled, $M_1$ and $M_2$ are conducting and the output nodes are low. The NOR outputs a high signal which keeps the tristate buffer disabled. When the required input swing is on the lines, $M_5$ can be activated which feeds current to the SA. The transconductances of $M_1$ and $M_2$ as well as the positive feedback load amplify the signal swing on the lines. When any of the two output nodes becomes high, the NOR gate activates the tristate and the amplified, non-differential signal is found on the output. According to [6], this SA might need some extra circuitry in order to guarantee glitch-free operation.

**Sizing a Sense Amplifier**

Sizing an amplifier is a difficult task, and simulation is an important stage in the process. In this section, some rules of thumb that apply to both amplifiers presented here are given:

- Symmetry must be kept in the sizes, i.e., $W_{M_1} = W_{M_2}$, $W_{M_3} = W_{M_4}$.
- The lengths of the transistors should be larger than $1.5 \cdot L_{\text{min}}$.
- The width of the enable transistor must be twice as large as the transistor it is connected to.

Furthermore, the following relationships must be kept into account:
- The smaller the input transistors, the smaller the capacitance at the input.
- The larger the input transistors, the larger the amplification.

### 3.4.5 Write Circuit

The write circuit discharges the BL when a ‘0’ is written and $\overline{\text{BL}}$ when a ‘1’ is written to the cell.

#### Differential Input Write Circuit

The circuit in Figure 3.20 is a differential input write circuit and works basically in the same way as the bitline decoupled latch type SA presented above, only in the other direction. The input to the latches will be taken from a data line and its complementary line.

As long as the enable signal is low, M5 is turned off and the two data lines are disconnected from the write circuit as M8 and M9 are turned off. M6 and M7 will also be off and, therefore, the bit lines are also disconnected. With this circuit, the four lines are in tristate as long as the enable is low.

When the enable signal turns high, current will start to flow through all the above mentioned transistors. The differential swing on the data lines will set the latches in the right state. The value on the drain between M1 and M3 will be a ‘1’ if a one has to be written and a ‘0’ otherwise. The opposite is true for the drain between M2 and M4. In this way, M10 is activated when writing a
‘0’ and M11 when writing a ‘1’, and the BL or \( \overline{BL} \) respectively will be discharged.

**SINGLE-ENDED WRITE CIRCUIT**

In Figure 3.21 a write circuit with a single data input line is shown. The single-ended write circuit takes two input signals: a write low and a data. When the write signal is low and data is ‘0’, the output of the first NOR will be high and the transistor connecting BL to GND is turned on. A ‘0’ is written into the cell. If the data is ‘1’ the output of the first NOR will be low, so the inputs to the second will both be low and its output will be high, which turns on the transistor connecting BL to GND and a ‘1’ is written to the cell.
The outputs of the NOR gates can be connected to more than one BL or BL switch, as shown in the figure above.

### 3.4.6 Pull-Up Circuit

The pull-up circuit in a memory array restores the original voltage level on the bit lines after a reading or a writing. In this section, the two pull-up circuits in Figure 3.22 will be presented. Their function is to pull up the voltage value on the bit lines to $V_{DD}$.

![Figure 3.22 - Pull-up circuits](image)

**Figure 3.22 - Pull-up circuits**
- (A) Simple pull-up circuit, adapted from [9], p. 428
- (B) Switchable pull-up circuit, adapted from [5], p. 73

#### Simple Pull-Up Circuit

The simplest possible pull-up circuit is composed of only one p-transistor with a gate connected to GND, as shown in Figure 3.22 (A). The transistor will conduct as soon as the value on the bit lines is less than $V_{DD}$ and the capacitances on the lines will be recharged. The advantage of the circuit is its simplicity. Its disadvantage is that when other circuits (cells, write circuits) try to discharge the lines, the pull-up transistor will try to charge them. A direct connection between $V_{DD}$ and GND is created, and unnecessary power is consumed. In order to counterbalance this phenomena, the size of the transistor must be kept at a minimum.

#### Switchable Pull-Up Circuit

The circuit proposed in Figure 3.22 (B) works similarly to the one described above, but has three transistors whose gates are connected to an enable signal.
The two upper transistors recharge the bit lines when an enable low signal is on their gate. The lower transistor assures that the two lines have the same charge when the transistor is on. The circuit should ideally only be turned on when this will not result in a direct connection between $V_{DD}$ and GND, in which case the transistor size may be larger than in the preceding circuit, increasing the speed of the recharge.

### 3.4.7 Buffers

As discussed above, the resistances and capacitances on the lines due to the wires’ resistive and capacitive properties as well as to the loads on the lines influence the speed on the lines as well as the quality of the signals. One solution is to size gates so that the wanted rise/fall time or the wanted delay is achieved. When the lines are too long, buffers are a solution commonly used in chip design. This means that a line is divided into shorter lines and these lines are then connected to each other with a repeater/buffer [7]. When the lines are one-directional, the buffers have to buffer up the signal in one direction, while when bi-directional the buffering has to be done either in one or the other direction.

**One-Directional Buffers**

The simplest one-directional buffering device is an inverter. The inverter size is chosen depending on the load capacitance so that the required rise and fall times are achieved. When large capacitances are driven, more than one inverter of decreasing size are placed in series. Placing an even number of inverters in series brings about a non-inverting buffering of the signal. The delay on the line will be according to (3.23), and the rise/fall time will be shorter as the load capacitance to be loaded will be smaller [13].
**Bi-Directional Buffers**

In order to be able to drive in only one direction at a time cross-coupled tristate buffers can be used, as shown in Figure 3.23.

![Bi-directional buffer diagram](image_url)

Figure 3.23 - Bi-directional buffer

In this way, we can have either of the tristate buffers driving the signal into the chosen direction, or both being deactivated.
4

ANALYSIS

An important part of this project is to choose a general memory structure that minimises the power consumption of the generated memories. The choice will be motivated in this chapter. However, the reasons for the choice of specific preliminary circuits and decisions specific for the chosen circuits will also be discussed here.

4.1 CHOICE OF STRUCTURE FOR THE MEMORY

As discussed in Section 3.3.1, many of the sources of power consumption in memory squaring are multiplied by the factor $k$. So the larger the number of words in a row the larger the power consumption. For this reason, as long as area is not an issue, memory squaring is not an optimal solution and will therefore not be implemented. However, as area might be of importance in the future, implementing it might become unavoidable. The design of the memory should allow for the addition of this feature.

A divided word line structure is a better solution for this implementation at time of writing. The given choice for the number of cells on the WL is the number of bits per word, so the length of the WL will vary. As clarified in Section 3.3.2, this structure cannot be indefinitely expanded into large...
memories. For this reason, the divided word line structure is only used for very small memories and for basic partitions.

A strategy is also required when expanding the memories. The partitioned structure is a superior solution to the hierarchical word line structure in this application for many reasons:

- The partitions can be seen as independent parts that may be placed where required without the bounds given by the hierarchical word line structure, which makes it easy to expand the memory keeping the aspect ratio within the 0.5-2 range. To do this in a hierarchical word line structure requires the addition of new SGWL and of new rows. The circuits on the BL/BL would hence require resizing.
- The size of the partitions can be kept within a given range and still keep the required aspect ratio. No changes in the parts are then needed.
- In the partition only a relatively short GWL and one LWL are driven at a time, while in the hierarchical word line structure a long GWL, one SGWL and one LWL are driven.
- The hierarchical word line structure needs one more metal layer compared to the partitioned structure.

Hence, the partitioned structure has been chosen for the expansion of the memory.

As became clear in Section 3.3.4, the lengths of the I/O lines increase with the size of the memory, which require that buffers are included along the line.

In Section 3.3.5, a hierarchical bank structure was introduced. According to Section 3.3, the only relevant difference which distinguishes it from the partitioned structure presented in Section 3.3.4 is the way the I/O lines are driven. The hierarchical bank structure forwards information on I/O busses only to or from the block which is to be activated, while the partitioned structure forwards information to or from all blocks. Actually, the hierarchical bank structure can be seen as a partitioned structure where the decoding has been spread along the way. This solution has many advantages compared to the simpler partitioned structure:

- The power used to drive the lines is minimised by only activating the lines to the active partition.
- The block decoder at the partition entrance has lower complexity and the area of the block decoder can be kept down.
Only the block decoders in the active bank are activated instead of all, implicating power saving.

As buffering is required on the line, decoding can be added there. The only possible disadvantage is that the buffers become more complex because of the decoding. In this application, decoders will therefore be placed along the I/O lines as according to the hierarchical bank structure.

One question that has not directly been treated in the theoretical section is whether the data lines should be differential or not. It is evident that with differential data lines, the data buss width doubles and with it the number of lines to be driven, which means large area especially for long words. The advantage of this solution would be that no full swing on the lines is needed in order to transfer information, saving power. Driving such a line with a small swing is a difficult task and for robustness as well as simplicity reasons a non-differential data line was chosen.

One important structural choice to be made is where to place the memory circuits. In order to avoid large busses, all row decoding and the lowest block decoders are placed close to the array. This also applies to write and SA circuits, since this model is more easily fitted together with a non-differential data bus and also because the vicinity to the array allows for lower swings and less risk for noise.

Within each partition, the structure described for the simple memory will be applied as no reasons for other choices are known to the author.

The question is now where the limits for each structure used in the design shall be put. This must take into account the kind of circuits used.

### 4.2 Choice of Circuits for the Memory

A preliminary choice of the circuits was done in step 2, described in Section 2.4.1 and the aim was not to achieve the optimal circuits, but to have some circuits to simulate in order to examine different structures. In this section, the choices done are discussed together with possible better alternatives or improvements when applicable.
4.2.1 Cell

In a large memory, the number of inactive cells is large. In order to keep power consumption down, the $i_{hold}$ for each cell must be very low. Here again, as area was not an issue, the full CMOS cell was a given choice.

4.2.2 Row Decoder

According to Section 3.4.2, a common choice for memory designers is a NAND decoder with predecoding. It keeps power consumption to a minimum. The choice for this application has nevertheless fallen onto a clocked NOR decoder for a number of reasons:

- A predecoder structure demands area outside the area dedicated to the row decoder. An alternative would be to place such a predecoder at the input of the memory or to place it in the available area under the row decoder, next to the write circuit and SA blocks. The first solution would imply larger busses, drivers, etc. The space might be insufficient for the second solution. The chosen structure could easily be fitted in an area dedicated to it next to the array, which makes it the better choice for a modular circuit.

- The speed of the NOR decoder and of the clocked NOR decoder is much higher than the other structures.

- Layout of the NOR decoder and of the clocked NOR decoder was deemed simpler than the layout of the predecoded NAND.

- The NOR decoder and the clocked NOR decoder has the possibility to vary the number of addresses without the speed of each decoder bit being influenced negatively, which is not the case for the NAND / predecoded NAND solution.

- The power consumption of the clocked NOR decoder can be kept low by division into blocks which are activated one at a time, as described in Section 3.4.2, when an improvement is required. The blocks can easily be placed next to the decoder bits.

Before the choice was made, simulations aiming at verifying its suitability of a clocked NOR decoder in a low-power application was verified. The following has been observed:
• There is a small increase in power consumption in each decoder part due to the addition of an extra address bit.

• The addition of an extra address bit means that the number of decoder bits are doubled. Therefore, the actual increase for the whole block is very large.

• As long as the number of decoder bits active at one time can be kept low (up to 64 rows), this solution will have an acceptable power consumption. For larger blocks, division into smaller blocks becomes necessary.

4.2.3 **Enable**

As a divided word line structure has been chosen, a gate for the selection of the active LWL is required. Given the choice of the clocked NOR decoder, which gives a low active signal on the output, the word line enable circuit is best suited for this purpose, as it is inverting, and has less transistors than the simple NAND enable. Furthermore, simulation shows that this circuit is very low-power. This enable block will be used to select the LWL as well as timing WL high pulse.

4.2.4 **Sense Amplifier**

The choice of the amplifier has been based on the following arguments:

• Given the choice of a non-differential data bus, the easiest solution would be the buffered sense amplifier, as it already gives a single-ended data bit.

• The enable signal for the bitline decoupled latch type SA will have five transistor gates as a load, while there will only be one transistor gate as load when using the buffered SA.

• The buffered SA offers buffering onto the data line. If more than one partition is placed in a bank, they will share the data line and the load for the SA will be large.

The buffered SA is the most suitable choice for this application.

4.2.5 **Write Circuit**

The single-ended write circuit has been chosen for this application since:
It is best suited for the non-differential data bus.

The load on the enable line is lower than for the differential circuit.

It is simpler to size and layout compared to the analog differential circuit because of its digital operation.

### 4.2.6 Pull-Up Circuit

The pull-up chosen here was the simple pull-up circuit in Figure 3.22 (A). The only reason this circuit was chosen is for its simplicity:

- It needs no timing as it is always active.
- It takes no effort to layout and works in all situations.

It is anyway advisable to update the circuit by using the structure presented in Figure 3.22 (B), as this circuit can be switched off when other circuits are pulling down BL/BL, avoiding direct current flow from VDD to GND. As the pull-up circuit would not counteract the circuits pulling down the lines, the memory could be speeded up by this solution.

### 4.2.7 Buffers

The buffers needed on the I/O lines are the one-directional address and control bit buffers, and the bi-directional data buffers.

In this application, decoding buffers are needed because of the structure chosen in Section 4.1.

**Address and Control Buffers**

As address and control signals are one-directional, the buffers used for these signals only have to buffer up signals in one direction. As the hierarchical bank architecture was chosen, these buffers need to be decoding. Instead of a couple of inverters in series, an enable circuit as the one in Figure 3.15 is used. One input to the NAND gate is the enable signal, i.e., the last address bit choosing which bank to activate. The other input is the signal to be buffered. Even if the different signals drive different loads, it would be too complicated to have buffers of different sizes, so all buffers in one group will have the same size.
CHAPTER 4 - ANALYSIS

**DATA BUFFERS**

Data buffers are bi-directional. Using the bi-directional buffers shown in Figure 3.23, the decoding can be done by controlling the enable signals to the two tristate buffers. In order to accomplish this, a NAND gate for each direction will be used. The inputs to the gate enabling the tristate to the arrays are the write enable and the last address bit choosing bank, the inputs to the other gate will be the complementary write enable and the last address bit. Note that these buffers are inverting. This means that the data signal might be inverted. This is, however, not a problem. If there are an even number of buffers on the I/O line, the original bit will be stored in the cell, otherwise its complement will. In the second case, the stored bit will be inverted on the way back. This means that the data signal on the output will be the same as the one on the input, even if the stored value is complementary.

4.3 **DEFINITION OF THE STRUCTURE LIMITS**

Taking into consideration the used circuits, limits must be established for when each of the structures chosen above should be used. Results of simulations are used for this purpose. Note that all simulations have been done with minimum width transistors, as increased width implicates increased power consumption.

The given choice for the number of cells on a LWL is the number of bits per word, as the memory squaring has been excluded in Section 4.1. The optimal number of LWLs on a GWL would be the one that allows for acceptable speed and keeps the aspect ratio of the block between 1 and 2. As the actual limit for a LWL lies slightly above 48 bits per line, the optimal solution for the maximum word length would be to have two LWL, one on each side of the global row decoder with an enable bit allowing the output on one of the two LWLs. In order to keep the design simple, the decision has been made to use this solution for all word lengths. It is evident that for short words this solution will incur large relative area overheads. So the design of the generator must take into consideration that the number of LWL can easily be increased in the future. At the time of writing, this seems as a good preliminary solution as long as area is not an issue.

The height of the arrays is basically given by the pull-up circuit and the write circuit. An acceptable speed can be kept for blocks up to 64 rows. For 128 rows, the speed degenerates. Because of this together with the limit given by
the row decoder in its simplest form discussed in Section 4.2, a maximal block height of 64 is chosen.

A change of the circuits may change the optimal maximal height. The only limitation on how to choose the maximum height is that it should never be smaller than half the maximal width of the block to avoid unmatched aspect ratios.

This fact also limits how many partitions that can be placed within each basic bank. The number of partitions will be doubled as long as the total width is less then or as large as twice the height. If the number of partitions has to be doubled after that limit has been reached, the partition row will be placed on two rows. The data lines along the blocks do not allow for any I/O line decoding if the size of the memory has to be doubled once more. This whole block must be the basic bank. From now on, for each doubling of size, one decoding node is added.

However, if the number of partitions required to reach the size of the memory can be placed on one or two rows, a decoding node will be placed in the middle of the row or one of the rows, and each half of the row(s) will belong to different banks.
The program takes as an input the size of the memory in number of words and bits per word. The number of words are rounded up by the generator to the nearest power of two. Then, the number of words to be fitted into a partition is calculated, depending on the maximum height for a partition, and the number of LWL in the partition. A list of all source code files is included in Appendix 2.

5.1 The Structural Implementation

The structure of the memory is decided recursively based on the size of the memory and of the basic partition. A basic partition is constructed in two steps: by first creating a basic part and then the partition. Afterwards, the memory is expanded by creating what is here called segments until the memory has reached half its required size. In the last step, what was called a floorplan is done, where the last segments are instantiated twice and all input circuits and possible lines are supposed to be placed. The floorplanning cellview will contain the entire generated memory. In order to align all segments properly, keeping into account the wire busses, so called bridges are used. As buffers and control circuit strongly depend on the structure used, these two circuits will be discussed in this section.
5.1.1 Creating Basic Parts

The first requirement for the memory generator is to create cellviews for each of the parts to be included in the basic partition. The only thing that has to be included in each cellview is a $V_{DD}$ terminal, placed as a top right corner of the cell, and a GND terminal, placed as a bottom left corner. The centre of the pins gives the limit of the cellview. All $V_{DD}$ and GND terminals shall have a width and height of 0.6 μm, use “metal1”, be of InputOutput type, and have the same rod object name. In Figure 5.1, the limits for a cell are shown, given its pins.

![Figure 5.1 - Basic part structure](image)

When layouting the parts, it is important to keep in mind that the pins will be the aligning reference and ensure that they are placed properly and that no DRC error due to putting the parts together appear.

When describing the pins of all parts, $V_{DD}$ and GND are not listed since all parts have them and they always have the same characteristics.

Some parts are supposed to be placed on the same row as the SRAM cell and some on the same column. In the first case, as for row decoder and enable circuit, the height of the parts shall be the same as the one of the SRAM cell. In the second case, as for pull-up circuit, write circuit, and SA, the parts shall have the same width as the SRAM cell.
5.1.2 Creating a Block of Parts

A program procedure creates a block of parts by calculating the size of the cell limit. This procedure can be run in “dummy” mode or in “layout” mode.

In “dummy” mode, the part size is measured along the limits of the cell and then multiplied with the number of parts that, according to what calculated at the beginning, need to be placed horizontally and vertically. An “align” rectangle will be created with the obtained dimensions in a new cellview. The cellview for the basic part does not have to contain anything but the two terminals in Figure 5.1.

In “layout” mode, the basic part will be instantiated with a proper name and numbering as many times as needed. Alignment will be done on the V_DD and GND pins. When aligning two parts vertically, one of them will be mirrored along the x-axis so that the two parts share the same supply wire. In order to create a block in “layout” mode, the cell for the part should at least contain all input and output pins for the cell. The pins of the cells will then be propagated to the block of parts with a bus name as, e.g., A<x> or with a normal pin name as ENABLE.

5.1.3 Creating a Partition of Blocks of Parts

When all blocks of parts are created, no matter if in “dummy” or “layout” mode, they are aligned to each other by the program according to the simple SRAM model. The “align” rectangles will be used as a reference for the alignment.
An example of a whole block with 64 words à 36 bits, generated in “dummy” mode is shown in Figure 5.2.

The figure also shows the proportions taken by each block, given the circuits used, are shown. All supply wires on the outside of the block are $V_{DD}$ wires.

Around the whole partition, an “align” rectangle is placed. On the left and right of the array, extra place is left for $V_{DD}$ wires, else the rectangle limits are the ones given by the part blocks’ rectangles.

If the words in a memory can all be placed in one array without exceeding the height limits, only one array with the number of words calculated above will be used.

### 5.1.4 Creating Bridges

A bridge is a pin row with an upper and a lower $V_{DD}$ pin as well as a centred GND pin. Pins for the required number of addresses and for all control signals are placed between the higher $V_{DD}$ and the central GND pin while data
pins are placed between the central GND and the lower V_{DD} pin. A bridge is shown in Figure 5.3.

Bridges are used for three purposes:

- Align segments to each other.
- Indicate where buffers shall be placed.
- Act as start and/or end point when placing busses.

Supply pins are 0.6 μm and in “metal1”, and all other pins are 0.32 μm and in “metal3”. By default, the distance between pins is the minimum distance. A greater distance can be chosen between data buss pins, address/control pins, or both. This feature is introduced since buffers sometimes need more place than a pin row with minimum distances can offer.

The centre of the lower and higher V_{DD} are the align points for the bridges.

5.1.5 Creating Segments

The way a segment is created depends on which segment it is. The first segment is called “basic block”, while all others will be called “segment x”, where x = 1, 2,... The placing of the buffers is also indicated in the figures. Apart from the “basic block”, the segments will be created alternatively vertically or horizontally. In the vertical case, the preceding segment will be
instantiated once as the “upper segment” and once vertically mirrored as the “lower segment”. In the horizontal case, the preceding segment will be instantiated once as the “left segment” and once horizontally mirrored as the “right segment”.

**Basic Block**

In a basic block, partitions are placed next to each other as long as there are partitions left to place and as long as the width of the basic block does not become larger than twice its height. If the basic block contains all partitions needed or half the partitions needed, a bridge will be placed in the middle and used as the input to the partitions as in Figure 5.4.

![Figure 5.4 - Divided basic block](image)

Otherwise all partitions will be aligned to each other according to the partition rectangles, as in Figure 5.5.

![Figure 5.5 - Whole basic block](image)
No buffers are placed there.

**SEGMENT 1**

If the memory is at least four times larger than the “basic block”, the first segment is created vertically with two “basic blocks”. It will not have the place needed for buffers, as the data I/O will be found all along the lower part of the “basic block”. This segment is shown in Figure 5.6.

![Segment 1 Diagram](image-url)

Figure 5.6 - Segment 1

**SEGMENT 2**

Segment 2 is created horizontally and aligned with two bridges, one to the left and one to the right of the segment. A buffer block will be placed on each side. At the entrance of the segment one buffer block driving both preceding buffer blocks is placed. This segment is shown in Figure 5.7.

![Segment 2 Diagram](image-url)

Figure 5.7 - Segment 2
STANDARD VERTICAL SEGMENT

For all segments $x$, where $x = (2y + 1)$ for all $y = 1, 2,...$, the preceding two segments are placed vertically and aligned with a simple bridge, containing one buffer block at the entrance. This kind of segment is shown in Figure 5.8.

![Figure 5.8 - Standard vertical segment](image)

STANDARD HORIZONTAL SEGMENT

The standard horizontal segment can be described as the vertical one, apart from the fact that they are placed horizontally next to each other as shown in Figure 5.9.

![Figure 5.9 - Standard horizontal segment](image)
5.1.6 Creating a Floorplanning Cellview

While generating the memory, the floorplanning cellview is where the whole memory is found. What the memory looks like depends on how many partitions are included in the memory.

One-Partition Memory Floorplanning

If all words can be placed in one or two arrays, the floorplanning will include one partition, the input buffer and the input circuit with control circuit and memory I/O pins. These different parts will be placed according to Figure 5.10 (A) for one array and as in Figure 5.10 (B) for two arrays.

If the block is small, the buffer block and the control circuit will probably be larger than the partition.
**ONE-BASIC BLOCK MEMORY FLOORPLANNING**

If all words can be placed in one “basic block”, a divided “basic block” is placed in the floorplanning cellview. The rest of the circuits will be placed in the middle of the “basic block” as shown in Figure 5.11.

![Figure 5.11 - One-basic block memory floorplanning](image)

There will be one buffer activating the left side and one activating the right.

**TWO-BASIC BLOCKS MEMORY FLOORPLANNING**

If two “basic blocks” are needed, a similar structure as for a one-basic block floorplanning is used. The only difference is that the left buffer block will drive the left half of both basic blocks and the right one both right halves. This is shown in Figure 5.12.

![Figure 5.12 - Two-basic block memory floorplanning](image)
CHAPTER 5 - IMPLEMENTATION

TWO-SEGMENT 1 MEMORY FLOORPLANNING

As segment 1 has no place for buffers, all buffers driving the segments must be placed in the floorplanning cellview. One left buffer block drives the left segment and one right buffer block drives the right segment. These are placed, as shown in Figure 5.13, in the vicinity of the entry bridges of the segments.

![Figure 5.13 - Two-segment 1 memory floorplanning](image)

The control and input circuit will be placed between the two segments aligned to the lower segment limit.

STANDARD VERTICAL FLOORPLANNING

For larger memories, the floorplanning is standardised. The only difference is between vertical and horizontal floorplanning, i.e., if the last two segments are placed in a column or in a row, similarly to how segments were created. In the vertical case, an upper segment and a mirrored lower segment are placed above each other. They are aligned to each other using an entry bridge aligned with the entry bridges of the two segments. The control circuit is aligned to
the rightmost limit between the two segments. Thereafter a buffer block is placed. This is shown in Figure 5.14.

**Figure 5.14 - Standard vertical floorplanning**

**STANDARD HORIZONTAL FLOORPLANNING**

The basic idea for the horizontal floorplanning is the same as for the standard vertical floorplanning. The only difference is that the segments are a left and a right one and the control circuit is aligned to the lowest limit between the two blocks. This is shown in Figure 5.15.

**Figure 5.15 - Standard horizontal floorplanning**
5.1.7 WIRING

A number of procedures perform the wiring of the memory. The basic procedure creates a path. This procedure can calculate the parasitic capacitance on the created path by using a constant for each metal. The value of this constant is difficult to determine exactly as it depends on many factors. There are two different constants: one for parallel wires at minimum distance and one for more distant wires. The first constant must take into account that the bits on the busses can switch into different directions, doubling the actual capacitance between the bus wires. This procedure can even calculate the wire capacitances without actually placing the wire, if required. This procedure is used in other procedures connecting straight busses, corner busses, T-busses, basic block busses and so on. In order to work, these procedures need all pins to connect to be in place.

The wiring is done step by step during the creation of the segments when possible. When busses go from, to or between buffer blocks, wiring is not possible until buffers are in place and only the parasitics are calculated.

While placing buffers in each segment, the missing wires are also placed. “Metal 3” is used for long distance wires and “metal 2” is used for crossing points, i.e., only for shorter paths.

5.1.8 BUFFER SIZING

An automated buffer sizing is a very complicated task for many reasons:

- The real capacitance on a wire is hard to determine without extraction.
- Gate capacitances as well as output capacitances are hard to determine without extraction as extraction shows that gate and output capacitances do not vary logically with the size of the buffers.
- Exact equations for current through transistors and their exact parameters are not known.
- As long as the exact buffer size is not known, it is not possible to exactly know the wire lengths.
- Each buffer has a different load.
Required modelling has furthermore increased the uncertainty of the result. The wire capacitance has been approximated from extractions. Since gate and output capacitances are relatively small compared to wire capacitances, all these capacitances have been approximated with a maximum value instead of being calculated exactly. A standard inverter has been simulated with different sizes and different load capacitances and a list, where the minimum size for an inverter given a load and a minimum speed, has been created. This will be loaded when loading the program, so the size will be calculated by fetching a value from this list.

When loading the program a large number of lists are created, where each buffer will be an element in each list. There are some lists containing buffer placing information, some with sizing information, some with gate and output capacitance information, some with wire capacitance information and some with total capacitances.

When running the floorplanning procedure and creating the segments and the final floorplanning cellview, all buffers are inserted and an element for each buffer is inserted in each list. The start value is zero. The placing information is added as well as the wire capacitance information during the floorplanning.

When the preliminary floorplanning is finished, all wire capacitance information should be available. A buffer placing procedure can be run.

All buffer widths can be calculated taking into consideration what load each buffer has. The buffer at the memory I/O takes into account the specified load the data lines must drive on the output. The size of all the data buffers is based on the largest load, and the same for all address and control buffers, and all decoders. The process of calculating the right maximum capacitance for each type of buffer and decoder becomes very uncertain because of the complex load relationships and the uncertain start values. Note that each data buffer before and after the one to be placed affects the load of the actual buffer. Furthermore, the load of the address/control buffers depends on the data buffers and on their decoders.

The maximal total capacitances are calculated before each sizing is done. The sizing of the n-transistor for each part of the buffer is stored in the sizing lists as if all buffers were inverters, here called $W_{N_{basic}}$. After each sizing, all capacitances that the sized part influences are updated and the next sizing is done.
5.1.9 Placing Decoded Buffers

The decoded buffers are seen as a block containing all data, address and control buffers, and decoders. They will have an input and an output pin row, and all wiring within the block can be ignored outside the block.

A procedure that places the buffers has to be run when all sizes according to the above are calculated as for an n-transistor in an inverter. This procedure starts from the first buffer, creates it, places it into the proper segment according to what is stored in the buffer placing lists, places the missing wires and continues until all buffers and wires are placed.

The following sections will discuss the different parts of the buffer block.

Input and Output Pins

An output pin row and an input pin row are needed. They need to be of the same size as the bridge the buffer is supposed to be aligned to, i.e., the number of addresses must be the same and so also the distances between the pins. The output pins will be aligned in the row on the left and the input pins on the right.

Buffer Structure

The decoded address and control buffers are placed next to the address and control pins, while the data buffers are placed next to the data pins. As the address/control buffers take much less area than the data pins, there will be
space for the decoders to the data buffers on the same height as the address buffers, as shown in Figure 5.16.

The limit for each block is given vertically by the $V_{DD}$ and GND pins. Horizontally, it is given by the segments/floorplanning size. There are no automatic checks for the horizontal limit. If not enough place is available, the distance between the pins will have to be increased manually so that some extra space will be given to the bus.

**BUFFER SIZING**

When the buffers are created, each stored size is manipulated in order to get the real sizes for each transistor, taking into account that the n/p-ratio is approximately 3. Table 5.1 shows how the transistors are calculated from the stored value for each type of buffer part.

<table>
<thead>
<tr>
<th>Buffer Type</th>
<th>$W_N$</th>
<th>$W_P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverter</td>
<td>$W_{Nbasic}$</td>
<td>$3W_{Nbasic}$</td>
</tr>
<tr>
<td>Tristate</td>
<td>$2W_{Nbasic}$</td>
<td>$6W_{Nbasic}$</td>
</tr>
<tr>
<td>NAND</td>
<td>$2W_{Nbasic}$</td>
<td>$3W_{Nbasic}$</td>
</tr>
</tbody>
</table>

Table 5.1 - Sizing circuits given the width for the n-transistor of an inverter
**ADDRESS AND CONTROL BUFFERS**

The address and control buffers are made in the same way and have the same size. The first part is the inverter. This part will be the smallest of the parts of the buffer. The second part is the NAND. Both parts are created automatically.

These two buffer parts are only built using normal high speed transistors with contacts on drain and source. As these two parts are often much smaller than the data buffers, a compact design is not as important as for the data buffers. The inverter and the NAND gate are not placed next to each other since the lengths of the wires driven by the first buffer and the last in a line will be more similar.

**DATA BUFFERS**

The data buffers normally require much more place than the address busses as they are bi-directional and two tristate buffers are needed. Furthermore, a tristate buffer has 4 transistors and the size is often much larger for the p-transistors in a tristate (see Table 5.1) than in an inverter or NAND gate given the same load. The inverter to the one enable input also requires area. For this reason, compactness becomes of importance.

For this kind of buffers, as well as for the decoders, transistors without contacts are used, which allows for the transistors to be directly connected at the drain/source next to each other, even when they have different sizes. Contacts are placed later where required. The tristate gates to both directions as well as the NAND gates to both directions are created automatically. The tristates and their inverters are placed next to each other to reach compactness as well as to avoid problems with the feedback. The feedback requires that all metals up to “metal 3” are used in the tristate block. The only possibility for connection to input and output pins is with “metal 4”. Contacts at the outputs and inputs from the “metal 3” pins to the “metal 4” wires are required. As all buffers are enabled by the same signals the NANDs are the same for all tristates to the same direction. For this reason only two gates are required.

**PART ARRAY**

Each part of a decoded buffer is placed in a part array whose height is the distance between $V_{DD}$ and GND pins in the pin row. After each part is generated with the right sizing, and its size is available, given the available height for the whole part block, an array with a given number of full rows and columns,
as well as of incomplete ones is calculated. Afterwards the parts are placed one after the other according to the matrix. In Figure 5.17, an example of such an array is shown.

![Buffer array diagram](image)

Each buffer part will be named according to the signal it is supposed to buffer up. When the part is placed, it will be connected to the preceding array of parts buffering the same signal.

The array is placed in the right place from the beginning, aligned to the preceding part array or pin row so that no DRC errors appear.

**Wiring within Buffers**

All wiring to the preceding part with the same name is done while placing each bit. The last wiring is needed from the input pin row to the preceding part.

For each connection, two metal layers are used: “metal 2” or “metal 3” for the vertical connections and “metal 3” or “metal 4” for the horizontal ones. In order to avoid unwanted connections, all “metal 3” connections will be placed at the same height as their corresponding pin in the input/output pin rows.

The only not yet automated connection is that of the enable pins to tristates and NAND gates.
5.1.10 CONTROL CIRCUIT

Address and data busses do not require any control logic at the input of the memory. The only signal that requires it is the enable signal, which is the signal that controls the memory. In this implementation, timing is done with a delay circuit.

TIMING THE BLOCK ENABLE SIGNALS

Timing is done by sending an enable pulse to each circuit that require enabling. The circuits need to be enabled at different points of the cycle and during a specific length of time. Simulations according to Step 7 described in Section 2.4.1 gave the timing scheme in Figure 5.18.

![Figure 5.18 - Timing of the enable signals](image-url)
Each pulse can be achieved by applying a start step up and a stop step down into a NAND gate and then inverting the output as shown in Figure 5.19.

> ![Figure 5.19 - Creating an enable pulse](image)

For this purpose a delay circuit built of inverters in series is used. If a step is placed on the input of the first inverter, the step will be found inverted and delayed at the output of each odd inverter and only delayed at the output of each even inverter. Start signals are hence found after the appropriate amount of even inverters and stop signals after the appropriate amount of odd inverters.

**CONTROL LOGIC ON THE MEMORY ENABLE SIGNAL**

The memory enable signal is required to give a pulse at the beginning of each cycle.

The circuit in Figure 5.20 is used to achieve this operation.

> ![Figure 5.20 - Memory enable input circuit](image)
Soon after the enable goes high the output from the first NAND gate will go low and the output node Q of the cross-coupled inverters will be set to one. When the enable goes high again, the output of the NAND will also go high and the signal at the output is kept as long as the reset signal goes low, which will happen with a delay due to the inverters at the input of the lower NAND gate.

5.1.11 Evaluation of the Structural Implementation

At the time of writing, the status of the structural implementation is the following:

Structure in General

- The basic partitions are generated properly.
- The segment production works properly.
- Bridges of the right sizes are placed correctly.
- Floorplanning works apart from the placing of the control circuit, the placing of the buffers, and wiring of the input circuit. The reason for that is that the control circuit should be adjusted to the needed timing before generation and the buffers have to be aligned to the timing circuit.
- Wiring works correctly as long as all pins required are available.

Input and Timing Circuit

- The input circuit is not implemented in layout.
- A delay circuit can be produced given the available area and the number of delays needed for each start and stop signal. The input and output pins are also placed properly.
- Placing of the gates at the output of the delay circuit is not automated.
- The connection from the delay circuit to the gates is not automated. But as the outputs for the start and stop signals can be found everywhere in the delay circuit, it would be too hard to optimise wiring from there to the NAND gates. For this reason, the connection is better left to manual intervention.
The timing proposed above is not fully tested, which means that some more simulation is required and tests aiming at finding out the applicability of the timing should be done before a proper evaluation of the proposal can be done.

Buffers

Buffers are generated and placed, but a large number of details require to be fixed.

- Wiring capacitance calculations work automatically given that capacitance constants for the lines are specified.
- Buffer sizing is still not working correctly, given all capacitances, as the relations between loads of the buffers are intricate: Total load capacitances for each buffer should be calculated automatically; Given the right capacitances, the sizing should be automated.
- When manually specifying their size, the buffers are generated and placed correctly.
- The only check to be done is whether there is enough space. If not, the distance between pins in bridges shall be increased.
- The user should be able to vary the width between address or/and data pins of a specific bridge and possible buffers aligned to it.
- Enable signals within the buffers are not yet connected automatically. Automating this function would save a lot of time in the long run.
- A low-active circuit should be placed at the input of the buffers activating each low-active segment, otherwise the decoding does not work properly. Without this circuit, both segments in a segment or floorplanning will be activated, which is not desirable.
- The connection of VDD/GND in buffers is not automated, which it should be.
- The placing of the buffer in the floorplanning cellview aligned to the control circuit should be automated.

Summarising

Summarising, at time of writing there is no reason to be dissatisfied with the chosen solutions for the structure, even if some improvements are required in
order to generate fully operational memories with minimal manual inter-
vention.

## 5.2 Implementation of the Circuits

Most subcircuits have been layouted by hand. All subcircuits are layouted so 
that there are no DRC errors neither when aligning with other instances of the 
same circuit according to the placement of the V_DD and GND pins nor when 
aligning two blocks of parts along the “align” rectangles as described above. 
When not otherwise specified, transistor sizing has been done according to 
the following sizing criteria:

- Minimum length is used.
- The n-transistor width is chosen according to the load the circuit has to 
drive and to Appendix 1.
- Rise and fall time around 1 ns.
- When two transistors are set in series their size is doubled.
- The ratio between n-transistors and p-transistors is chosen to 3.

### 5.2.1 Implementation of the Cell

#### Description of the Cell Implementation

The 6-transistor cell is implemented with the following transistor widths:

<table>
<thead>
<tr>
<th>Size</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W_{N1}, W_{N2}</td>
<td>0.70 \mu m</td>
</tr>
<tr>
<td>W_{N3}, W_{N4}</td>
<td>0.28 \mu m</td>
</tr>
<tr>
<td>W_{P1}, W_{P2}</td>
<td>0.42 \mu m</td>
</tr>
</tbody>
</table>

Table 5.2 - Widths for the SRAM cell

which was one of the possible alternatives discussed in Section 3.4.1, given 
that the smallest of all transistors is set to minimum width. All transistor 
lengths are L_{min}. This choice has been tested by simulation with a ramp of 
1 ns, which gives correct operation for both read and write.
CHAPTER 5 - IMPLEMENTATION

EVALUATION OF THE CELL IMPLEMENTATION

Extensive simulations of the cell in different contexts show proper operation. The layout of the cell can be compacted substantially. As will be shown below, the height competitively of the parts on the same row is much easier to achieve than the width competitively of the ones in the same column. The cell can easily be compacted in height without major changes in the structure of the basic partition.

5.2.2 IMPLEMENTATION OF THE ROW DECODER

DESCRIPTION OF THE ROW DECODER

The clocked NOR decoder is composed of three different parts:

- The p-net part including clock and feedback transistor.
- The n-part of the clock and the output inverter.
- The address transistor.

The division into parts is shown in Figure 5.21.

![Row decoder from schematic to layout](image-url)
These parts are layouted by hand and then programmed into a whole row decoder into one cellview. Each address transistor is connected to either \( A^{<x>} \) or \( \overline{A^{<x>}} \) at the top level by the row decoder procedure, as this connection depends on where the part is placed.

The addition of the predecoding NANDs can be done on the left of a block of rows.

The sizing of the transistors in this block is shown in Table 5.3.

<table>
<thead>
<tr>
<th>Size</th>
<th>Width (( \mu m ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W_{M1} )</td>
<td>0.28</td>
</tr>
<tr>
<td>( W_{M2} )</td>
<td>1.12</td>
</tr>
<tr>
<td>( W_{M3},W_{M4} )</td>
<td>0.56</td>
</tr>
<tr>
<td>( W_{M5} )</td>
<td>1.28</td>
</tr>
<tr>
<td>( W_{M6} )</td>
<td>0.44</td>
</tr>
</tbody>
</table>

Table 5.3 - Widths for the clocked NOR decoder

**EVALUATION OF THE ROW DECODER**

The row decoder has not been tested completely, but some rows have and they are correct. The address bits and the p-net parts can easily be compacted in height if the cells are updated to more compact ones. The n-net bit must be developed horizontally instead of vertically if compacting is required.

**5.2.3 IMPLEMENTATION OF THE ENABLE CIRCUIT**

**DESCRIPTION OF THE ENABLE CIRCUIT**

The sizing of the enable circuit is done according to the sizing criteria above. The resulting sizing is:

<table>
<thead>
<tr>
<th>Size</th>
<th>Width (( \mu m ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W_N )</td>
<td>0.44</td>
</tr>
<tr>
<td>( W_P )</td>
<td>1.32</td>
</tr>
</tbody>
</table>

Table 5.4 - Widths for the enable circuit
CHAPTER 5 - IMPLEMENTATION

EVALUATION OF THE ENABLE CIRCUIT

The enable circuit is, according to simulation, low-power and consumes the same amount of power independently of the enable pulse length. With this sizing it can drive a whole row of 48 bits with a rise and fall time of approximately 1 ns.

5.2.4 IMPLEMENTATION OF THE SENSE AMPLIFIER

DESCRIPTION OF THE SENSE AMPLIFIER

The SA has been sized using simulation results and the rules of thumb described in Section 3.4.4. With matched transistors, the SA worked well during simulation with smaller transistor widths. When mismatch was applied, the sizes required to be increased. Increasing length from \(1.5 \text{ } L_{\text{min}}\) to \(L = 0.32 \mu \text{m}\), increased operation margins.

The logic at the output was sized given a load capacitance for the SA of approximately 200 fF according to the logic sizing criteria above. The resulting sizing is summarised in Table 5.5:

<table>
<thead>
<tr>
<th>Size</th>
<th>Width (( \mu \text{m} ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(W_{M1}, W_{M2})</td>
<td>1.28 ( \mu \text{m} )</td>
</tr>
<tr>
<td>(W_{M3}, W_{M4})</td>
<td>6.0 ( \mu \text{m} )</td>
</tr>
<tr>
<td>(W_{M5})</td>
<td>12.0 ( \mu \text{m} )</td>
</tr>
<tr>
<td>(W_{NORN})</td>
<td>0.28 ( \mu \text{m} )</td>
</tr>
<tr>
<td>(W_{NORP})</td>
<td>1.68 ( \mu \text{m} )</td>
</tr>
<tr>
<td>(W_{TRISTATEN})</td>
<td>1.4 ( \mu \text{m} )</td>
</tr>
<tr>
<td>(W_{TRISTATEP})</td>
<td>4.2 ( \mu \text{m} )</td>
</tr>
<tr>
<td>(W_{INVN})</td>
<td>0.28 ( \mu \text{m} )</td>
</tr>
<tr>
<td>(W_{INVP})</td>
<td>0.84 ( \mu \text{m} )</td>
</tr>
</tbody>
</table>

Table 5.5 - Widths for the SA circuit

The SA is prone to be resized for possible adjustments. For this reason, this circuit has been programmed, even if not at a completely abstract level. Many constants are still found in the code.
CHAPTER 5 - IMPLEMENTATION

EVALUATION OF THE SENSE AMPLIFIER

The SA is stable for mismatch of up to 10%. The only proven negative factor with this SA is the large area requirement due to the large transistor sizes, which makes compacting the width very difficult. It is important to enable the SA first when the necessary swing is ready on BL and BL.

5.2.5 IMPLEMENTATION OF THE WRITE CIRCUIT

DESCRIPTION OF THE WRITE CIRCUIT

All transistors of the write circuit have been sized to minimum width as simulation showed that this sizing gives acceptable results. The resulting sizing is:

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_N$</td>
<td>0.28 µm</td>
</tr>
<tr>
<td>$W_P$</td>
<td>0.28 µm</td>
</tr>
</tbody>
</table>

Table 5.6 - Widths for the write circuit

The circuit has also been programmed, for the same reason and in the same way as the SA circuit.

EVALUATION OF THE WRITE CIRCUIT

The write circuit works as required and is, according to simulation, low-power. For a faster circuit the width of the enable transistors can be increased.

5.2.6 IMPLEMENTATION OF THE PULL-UP CIRCUIT

DESCRIPTION OF THE PULL-UP CIRCUIT

The pull-up transistors are sized in order to minimise power consumption. The width chosen was 0.44 µm, as simulation gave acceptable rise times for this value, and the length was chosen to minimum length.

The GND wire is implemented in “metal 1” and “metal 5” with contacts in between. This wire can be used to distribute GND to all GND wires within the partition by drawing wires in “metal 5” from there to all “metal 1” wires and placing a contact at the intersection.
In order to keep the contours of the partitions as V\textsubscript{DD}, an upper V\textsubscript{DD} wire is added, even if it does not have any other use. A “metal 6” wire could be placed above it and used similarly to the GND wire.

**Evaluation of the Pull-Up Circuit**

This pull-up circuit is unfortunately the source of unnecessary power consumption and slows down the fall times on BL and BL. From all other points of view, this circuit seems to work good enough. The rise time on the lines is not 1 ns, given that there are 64 rows in one array, but this is not needed for correct operation, given the way the timing works.

**5.2.7 Implementation of Block Decoders**

At time of writing, no block decoder has been implemented and it is left to manual intervention. The reason for this choice is that the implementation of the block decoder is complex and there was not enough time for it. The implementation should deal with the following issues:

- The block decoder should take addresses and control signals as inputs.
- The block decoder should only forward the input signals when the partition addresses have the combination that activate that partition.
- When the partition is active, the block decoder should only forward the input signals to the row decoder and to either the left circuit blocks (as the write block, the SA block, etc.) or to the right, depending on the address.
- The signals forwarded should have the same delay and rise/fall time independently of the load for each signal so that the timing of the control signals is not affected by this circuit.
- The enable signals, except for the clock, should be inverted, i.e., the enable signal at the input of the block decoder is high when a certain circuit block is active and low otherwise, but the output should be low when the blocks are active and high otherwise.
- The block decoder should deliver the addresses to the row decoder as well as their complement.
- It should be placed so that it fits under the row decoder and enable blocks and between left and right write circuits and SA.
The inputs should be placed along one line on the lower limit of the partition.

The outputs should be placed where the inputs to the circuit parts are. Especially, keeping the timing could be a difficult issue.
During this thesis work a memory generator has been developed. The generator works in its whole, but does not fully automatically deliver a complete circuit. A few circuits still need to be designed ad hoc, and some functions do not work properly yet. The aim of this chapter is to summarise the results and the current status of the work.

6.1 IMPLEMENTATION VERSUS REQUIREMENT SPECIFICATIONS

Taking into account the limitations discussed in Section 1.3, a memory generated by the tool fulfils the requirements discussed in Section 1.2 according to the following:

6.1.1 MEMORY SIZE

- Requirement 1 - The memory generator translates any number of words into the nearest power of two by rounding upwards. There are no lower or upper limits for how large the memory may be. This requirement has
CHAPTER 6 - CONCLUSIONS

hence been fulfilled.

- **Requirement 2** - The memory generator can take any word length. The dimensioning of the parts is done for a maximum of 48 bits. Larger words would slow down the memory and the aspect ratio might become too large. For small words, the distance between pins in bridges will require to be increased so that the buffers have enough space. This requirement has also been fulfilled.

### 6.1.2 Low-Power

- **Requirement 3** - The structure of the generated memories is chosen in order to keep power consumption low. The circuits are designed so that they can be turned off when not used. A timing circuit controls when circuits will be turned on and off. However, the input and timing circuit is not generated automatically, which was foreseen as a possibility in limitation 3. Oversizing of circuits according to limitation 1 may often occur. However, in its whole, this requirement has been fulfilled.

### 6.1.3 Flexibility

- **Requirement 4** - The generator is modularised so that some parts can be changed without changing the whole structure as long as some rules are followed. For the basic parts the V_{DD} and GND pins must be placed in a specific place, the height of the row decoder gate and of the enable circuit must be the same as the height of the cell, and the width of the SA, the pull-up circuit and the write circuit must be the same as the width of the cell. The code allows for changes in parts as each part corresponds to a procedure or a group of procedures. This requirement has been fulfilled.

- **Requirement 5** - The buffers can be automatically designed for sizes that allow to drive a load on the data lines of 1 pF. The process of sizing the buffers does not work properly and requires hands on intervention, according to limitation 3. This requirement has only been partly fulfilled.

### 6.1.4 Memory Control

- **Requirement 6 and 7** - Inputs and outputs follow the requirements.
 Requirement 8 - The proposed input circuit for the enable signal controls the memory according to the specifications. The enable input circuit is however not implemented in layout. A proposal of a timing circuit and a circuit that has to be finetuned before generation is implemented. Apart from the limitations above, this requirement is fulfilled.

6.1.5 SPEED

 Requirement 9 - A cycle time for read/write within a partition is approximately 5 ns. The delay on the I/O lines has to be less than 15 ns for this requirement to be fulfilled. This can be obtained by changing the speed of the buffers. The memory parameters can be specified by the user so that this requirement is fulfilled.

6.1.6 CHIP AREA

 Requirement 10 - The cells are not compact and the area of the memory becomes unnecessarily large. This affects power consumption and speed negatively. Even if no area requirement was put on the memory, this is clearly a problem.

6.2 POSSIBLE IMPROVEMENTS AND FUTURE WORK

There are some improvements that are required in order for the memory generator to work acceptably, some that can be done to minimise manual intervention and some that can be done to improve the circuitry used. These three kinds of improvements will be discussed separately.

6.2.1 NECESSARY ADDITIONAL WORK

Some improvements must be done within some specific parts of the generated memories.
BUFFERS
A large amount of tasks need to be carried out on buffers in order to minimise manual intervention, see Section 5.1.11.

TIMING AND INPUT CIRCUIT
Also in the timing and input circuits, some work is left to do in order to minimise manual intervention. The specific tasks are described in Section 5.1.11. Furthermore:
- An I/O model should be created that can be used to find the complete cycle time including the delay on the lines.
- The placing of the gates at the output of the delay circuit should be automated.

BLOCK DECODER
- At least input and output pins for the block decoder must be placed. With this solution, each generated memory needs a block decoder to be laid-out ad hoc.

SUPPLY CONNECTIONS
- Connections to VDD and GND should be automated.

TESTING
- LVS system tests on blocks of different sizes, on buffers, and on timing and input circuits should be done.
- Simulations on whole blocks should be done if possible.

USER MANUAL
- A user manual should be written for the final version of the program.

6.2.2 IMPROVEMENTS TO MINIMISE MANUAL INTERVENTION

BLOCK DECODER
- An automated block decoder should be placed in each block. The issues
to be dealt with in this process are described in Section 5.2.7.

6.2.3 IMPROVEMENTS ON THE ACTUAL CIRCUITS

- Except for the pull-up circuit, all structures chosen for specific circuits are good. Some parts are, however, not layouted in a sufficiently compact way, especially the SRAM cell. Compacting the cell vertically would be a good measure for decreasing power consumption.

- If the cell is compacted in height, also the row decoder and the enable circuit must be compacted.

- The maximum number of rows in a block can probably be doubled.

- The timing must be adjusted to fit the changes.

- The pull-up circuits could be provided with a switch in order to avoid unnecessary power consumption. The number of control signals to be timed then increases with one.

- The row decoder can be divided into blocks where only the interesting block is activated. This would effectively reduce the power consumption, but at the same time it would influence the speed negatively.

6.2.4 COMMENTS ON FUTURE WORK

There are a few features that I believe are better left to manual intervention rather than being automated. These are:

- The connection from the outputs of the delay circuit at the memory input to the input of the gates that create the enable signals.

- Finetuning of the timing circuit is hard to automate. To check timing for each specific application implies that more optimal solutions may be reached. Keeping the design of the enable input circuit open for changes and finetuning, the generated memories can be made faster and more low-power than if only one or a few fixed delays are available.

- Even if a horizontal compaction of the cell would be good, compaction of the SA and write circuit can be time consuming.
6.3 Problems Encountered During Implementation

The major problem with this implementation is the large amount of small tasks to be carried out, as well as the large amount of analog problems with capacitances and delays in the memory which must be taken care of. This made it impossible for me to foresee all possible problems that would appear.

The instability of Cadence has slowed down the process appreciably. A large amount of time has been spent on restarting the computer, restarting Cadence, reloading the program, re-running the program in order to get all variables on place, etc. At time of writing, after having switched to a newer version of Cadence as the old one was too unstable, Cadence has had to be restarted 255 times because of crashes due to problems described in Section 2.1 and Section 2.2. Also finding errors from misleading error messages has taken a lot of time.

6.4 Final Words

Although this did not come as a surprise, the project was too large to be carried out within a 20 weeks period. The programmed tool does not at time of writing deliver fully working memories, but with the improvements described above, the generator can be used for producing low-power memories of different sizes. Even if finetuning of the timing minimises the power consumption and maximises speed, a few stable timings proven to work on a fabricated chip could be used for more standardised but less optimal solutions.
REFERENCES


Appendix 1

Inverter Sizing Data

\[ t = \max(\text{fall time, rise time}) \]

n/p-ratio approx. 3

\[ W_P = 3 \ W_N \]
### APPENDIX 1

<table>
<thead>
<tr>
<th>$t$ [s]</th>
<th>$W_N$ [$\mu$m]</th>
<th>$C_{LOAD}$ [fF]</th>
</tr>
</thead>
<tbody>
<tr>
<td>250 p</td>
<td>0.28</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>0.44</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>0.56</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>0.72</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>0.84</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>1.12</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>1.28</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>1.40</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>1.56</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>1.68</td>
<td>116</td>
</tr>
<tr>
<td>500 p</td>
<td>0.28</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>0.44</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>0.56</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>0.72</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>0.84</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>138</td>
</tr>
<tr>
<td></td>
<td>1.12</td>
<td>158</td>
</tr>
<tr>
<td></td>
<td>1.28</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>1.40</td>
<td>199</td>
</tr>
<tr>
<td></td>
<td>1.56</td>
<td>219</td>
</tr>
<tr>
<td></td>
<td>1.68</td>
<td>240</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t$ [s]</th>
<th>$W_N$ [$\mu$m]</th>
<th>$C_{LOAD}$ [fF]</th>
</tr>
</thead>
<tbody>
<tr>
<td>750 p</td>
<td>0.28</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>0.44</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>0.56</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>0.72</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>0.84</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>212</td>
</tr>
<tr>
<td></td>
<td>1.12</td>
<td>243</td>
</tr>
<tr>
<td></td>
<td>1.28</td>
<td>274</td>
</tr>
<tr>
<td></td>
<td>1.40</td>
<td>306</td>
</tr>
<tr>
<td></td>
<td>1.56</td>
<td>337</td>
</tr>
<tr>
<td></td>
<td>1.68</td>
<td>369</td>
</tr>
<tr>
<td>1000 p</td>
<td>0.28</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>0.44</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>0.56</td>
<td>161</td>
</tr>
<tr>
<td></td>
<td>0.72</td>
<td>199</td>
</tr>
<tr>
<td></td>
<td>0.84</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>280</td>
</tr>
<tr>
<td></td>
<td>1.12</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>1.28</td>
<td>363</td>
</tr>
<tr>
<td></td>
<td>1.40</td>
<td>405</td>
</tr>
<tr>
<td></td>
<td>1.56</td>
<td>446</td>
</tr>
<tr>
<td></td>
<td>1.68</td>
<td>488</td>
</tr>
<tr>
<td>$t$ [s]</td>
<td>$W_N$ [$\mu m$]</td>
<td>$C_{LOAD}$ [fF]</td>
</tr>
<tr>
<td>--------</td>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td>1500 p</td>
<td>0.28</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>0.44</td>
<td>190</td>
</tr>
<tr>
<td></td>
<td>0.56</td>
<td>246</td>
</tr>
<tr>
<td></td>
<td>0.72</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>0.84</td>
<td>366</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>428</td>
</tr>
<tr>
<td></td>
<td>1.12</td>
<td>491</td>
</tr>
<tr>
<td></td>
<td>1.28</td>
<td>554</td>
</tr>
<tr>
<td></td>
<td>1.40</td>
<td>618</td>
</tr>
<tr>
<td></td>
<td>1.56</td>
<td>682</td>
</tr>
<tr>
<td></td>
<td>1.68</td>
<td>746</td>
</tr>
<tr>
<td>2000 p</td>
<td>0.28</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>0.44</td>
<td>251</td>
</tr>
<tr>
<td></td>
<td>0.56</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>0.72</td>
<td>402</td>
</tr>
<tr>
<td></td>
<td>0.84</td>
<td>483</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>565</td>
</tr>
<tr>
<td></td>
<td>1.12</td>
<td>648</td>
</tr>
<tr>
<td></td>
<td>1.28</td>
<td>732</td>
</tr>
<tr>
<td></td>
<td>1.40</td>
<td>816</td>
</tr>
<tr>
<td></td>
<td>1.56</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td>1.68</td>
<td>984</td>
</tr>
<tr>
<td>2500 p</td>
<td>0.28</td>
<td>217</td>
</tr>
<tr>
<td></td>
<td>0.44</td>
<td>314</td>
</tr>
<tr>
<td></td>
<td>0.56</td>
<td>407</td>
</tr>
<tr>
<td></td>
<td>0.72</td>
<td>506</td>
</tr>
<tr>
<td></td>
<td>0.84</td>
<td>608</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td>1.12</td>
<td>816</td>
</tr>
<tr>
<td></td>
<td>1.28</td>
<td>921</td>
</tr>
<tr>
<td></td>
<td>1.40</td>
<td>1027</td>
</tr>
<tr>
<td></td>
<td>1.56</td>
<td>&gt; 1000</td>
</tr>
<tr>
<td></td>
<td>1.68</td>
<td>&gt; 1000</td>
</tr>
<tr>
<td>3000 p</td>
<td>0.28</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td>0.44</td>
<td>383</td>
</tr>
<tr>
<td></td>
<td>0.56</td>
<td>494</td>
</tr>
<tr>
<td></td>
<td>0.72</td>
<td>614</td>
</tr>
<tr>
<td></td>
<td>0.84</td>
<td>736</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>861</td>
</tr>
<tr>
<td></td>
<td>1.12</td>
<td>988</td>
</tr>
<tr>
<td></td>
<td>1.28</td>
<td>1115</td>
</tr>
<tr>
<td></td>
<td>1.40</td>
<td>&gt; 1000</td>
</tr>
<tr>
<td></td>
<td>1.56</td>
<td>&gt; 1000</td>
</tr>
<tr>
<td></td>
<td>1.68</td>
<td>&gt; 1000</td>
</tr>
</tbody>
</table>
APPENDIX 2

LIST OF CODE FILES

SUPPORT FILES
- variables.il
- mos_creation.il
- pin_row.il
- create_contact.il
- sizing.il
- support.il
- list_procedures.il
- wiring.il

START FILES
- memory.il
- memory_dimensioning.il

BUILDING CIRCUITS
- write_circuit.il
- sense_amp_circuit.il

BUILDING CIRCUIT BLOCKS
- building_row_column.il
- array.il
- enable.il
- pullup.il
- row_decoder.il
- sense_amp.il
- write.il
APPENDIX 2

BLOCK CONSTRUCTION
- block_construction.il
- block_dimensioning.il

SEGMENT CONSTRUCTION
- segments.il

FLOORPLANNING
- floor_planning.il
- new_floorplan.il

BUFFER PROCEDURES
- buffer_placing.il
- buff_nand.il
- addr_buff.il
- buffer.il
- data_buffer.il

INPUT AND CONTROL CIRCUIT PROCEDURES
- control_circuit.il
- timing.il

LOAD CAPACITANCE CALCULATIONS AND SIZING
- capacitances.il
- gate_sizing.il
The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Deborah Capello