# Spice Circuit Reduction for Speeding Up Simulation and Verification

CANCAN YIN
MENGLIN WANG
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

# Spice Circuit Reduction for Speeding Up Simulation and Verification

Cancan Yin
ca8267yi-s@student.lu.se
Menglin Wang
me3457wa-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor:
Joachim Rodrigues
joachim.rodrigues@eit.lth.se
Hemanth Prabhu (Xernergic AB)
hemanth.prabhu@xenergic.com
Xiao Luo (Xernergic AB)
xiao.luo@xenergic.com

Examiner:
Erik Larsson
erik.larsson@eit.lth.se

June 13, 2019

Printed in Sweden
Tryckeriet i E-huset, Lund

# Acknowledgements

We would like to first thank our supervisor Xiao Luo and Hemanth Prabhu at Xenergic for their continuous help and precious support through this project. We also would like to thank Babak Mohammadi, CEO of Xenergic, for always offering us his support and answering our questions patiently. We wish to thank Joachim Rodrigues, our supervisor at Lund University, Faculty of Engineering, for giving us valuable advices. We are grateful to Xenergic for offering us this great opportunity and also to our colleagues for sharing their experience with us.

We would like to express our gratitude to Lund University for giving us the chance to take this Master Program, and also thanks to our classmates who have spent this unforgettable two years with us.

At last but not least, we want to thank our families for their love and support that encourage us to overcome all the difficulties.

# Abstract

The focus of this work has been to implement a generic netlist reduction engine to speed up circuit simulations. The netlist reduction techniques are further optimized for Static Random-Access Memory (SRAM), wherein we exploit the repetitive pattern of the circuit.

There are many driving factors for developing a netlist reduction engine for SRAM simulations. In today's System on Chip (SoC), SRAM sizes are in megabyte ranges to support ever-increasing demands for features. The increasing size of SRAM makes it one of the biggest contributors of power consumption and area of a SoC. Many of today's state-of-the-art SoCs have on average more than 50% area and power consumption due to SRAMs. Hence it is very crucial to run full simulations of SRAM to check functionality, timing and power numbers. Unfortunately, due to the huge size of SRAM, it is unfeasible to simulate the whole SRAM, since it would take in the order of months to perform simulations. Also, a typical SRAM needs to be run for different corners, which is performed by Monte Carlo simulations, which is even more computationally intensive. Tackling these issues is the key focus of this thesis. We perform this by exploiting the iterative nature of the SRAM circuit.

The design is implemented in Python and verified on Xenergic's latest SRAM by using Cadence® simulation tools. The reduction engine has shown to provide a speed up of around 95% (using Spectre® simulator) for a 4 kb SRAM. The tolerance of simulation results between the original SRAM netlist and a reduced netlist are below 5%. Furthermore, there is no difference when it comes to SRAM functionality via the digital interface. In addition, time for the whole reduction process is far less than saved simulation time for a large scale circuit.

# Popular Science Summary

Nowadays, people's reliance on electronic products is ever-growing, which is driving the demands for higher computational power and mobility of electronic devices. This increasing demand has led to quick updates on electronic devices in a very competitive market, which means time for design cycles in electronic devices can be very short. Shorter design cycles may help companies seize more opportunities. IC design takes up a portion of electronics device designs. Hence, the design process of IC needs to be accelerated to adapt to the rapid changes. The simulation of design plays an important role in the design process. It allows the designer to know how the design works in reality, so the designer is able to modify and verify their design. However, the simulation can take a huge amount of time, since circuits today consist of hundreds of thousands of transistors and the transistor model for running accurate circuit simulation is very complex. This means that a large amount of data are processed to simulate circuit behavior. Besides, during the design process, circuit design needs to be modified and verified several times before manufacturing. Therefore, long simulation time can cause a large increase in the design time. This master thesis proposes decreasing the simulation time by ignoring redundant parts of the circuit during the simulation. As a result, the simulation time of the reduced circuit can be cut significantly while the functionality of the circuit is still ensured.

# Table of Contents

# List of Figures

# List of Tables

# Listings

# List of Abbreviations

**BL** BitLine. ix, 6–8, 10, 22, 24, 33

**BLB** Bitline-inverse. ix, 6, 7

**BSIM** Berkeley Short-Channel insulated-gate field-effect transistor Model. 13

**C-V** Capacitance-Voltage. 13

**CMOS** Complementary Metal Oxide Semiconductors. 1

**HRC** High Replication Circuits. 7

**I-V** Current-Voltage. 13

**SoC** System on Chip. iii, 1

**SPICE** Simulation Program with Integrated Circuit Emphasis. 1

**SRAM** Static Random-Access Memory. iii, ix, xi, 1, 2, 5–7, 10, 14, 21, 26–28, 37, 38, 42–45, 49, 51

**WL** WordLine. ix, 6–8, 10, 22, 24, 33

# Chapter 1
# Introduction

Today's System on Chip (SoC) needs large amounts of memories to support various functionalities and features. On-chip memories can improve the functionality of SoC and in particular, Static Random-Access Memory (SRAM) is a popular choice due to its high access speed and relatively low power consumption.

SRAMs in many of today's SoCs are in the range of megabytes [1] which results in a very large design. Take the 6-T SRAM for example, every bit will be stored in an individual unit which contains two stable states of a latch composed of two Complementary Metal Oxide Semiconductors (CMOS) inverters in a feedback loop [2]. This would result in an extremely huge number of transistors, resistors, and capacitors. In other words, the netlist file that describes electronic components, nets, and connections in an SRAM schematic can be extremely large. If there is a need to run SPICE® (Simulation Program with Integrated Circuit Emphasis) simulations through all the circuit components to get accurate results, it can be very computationally intensive. For a large SRAM, these simulations can take time in orders of weeks or months even in a huge server. During the iteration of SRAM design, the simulation time reduction can improve the total design drastically.

## 1.1 Motivation

The increasing size of memories used in SoCs will lead to the increasing complexity of circuits, so verification of the memories can be a problem due to the long simulation time. The behavior of components in the circuit are simulated by analysis and computation based on device models. During the simulation, stimuli are given to activate a part of the circuit. Take the SRAM circuit as an example, only a very small part of the circuit (active bitcell rows and corresponding peripheral circuits) is operating. Therefore, there are different methods to speed up circuit simulation, such as netlist reduction or improving device models.

This thesis focuses on using circuit reduction techniques to shorten the simulation time. The advantages of this method are that it will not change the circuit

design itself and the simulator does not change.

## 1.2 Thesis specification and Main Challenge

In order to decrease the simulation time, an automatic netlist reduction engine is proposed. Figure. 1.1 shows the general design idea of the reduction engine. The idea is to lower the number of circuit components in the netlist so that the simulation speed improves. This is achieved by observing circuit components and nodes and checking those parts which influence the observation nodes. These components which influence the observation nodes are flagged. By removing the non-flagged parts of the circuit and equivalent simpler models are substituted for them, the simulation time can be significantly lowered. One important aspect to note here is that the functionality of the circuit should remain unaffected. In this project, the reduction engine has a particular pattern detection function which will enhance the reduction dramatically for a special structured circuit like SRAM. After the realization of the reduction aimed at 6-T SRAM, the program can also be extended to other kinds of SRAM and even to general circuits through a corresponding add-on.

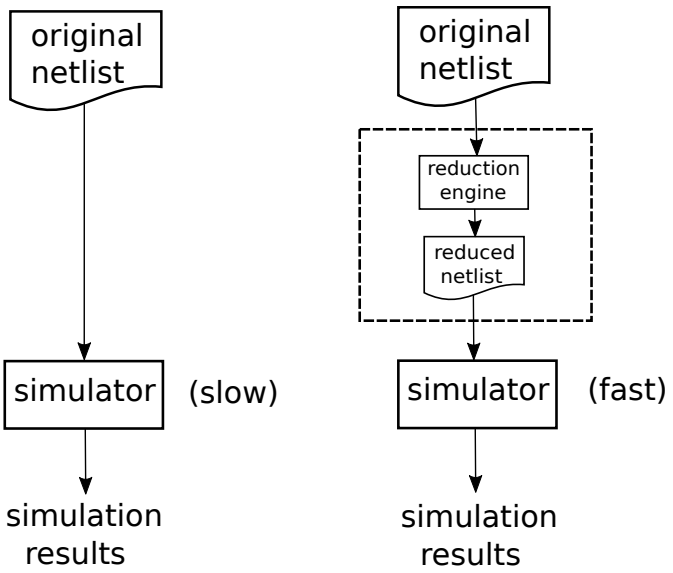



**Figure 1.1:** Simulation flow overview showing the application of the netlist reduction engine. The reduction process speeding up the simulation is shown inside a dashed square.

The main challenges in this project are described below.

- The pattern recognition of the bitcell logic.

- The method to detect the redundant part of the circuits based on the given simulation condition.
- How to establish a replacing model to maintain the accuracy of the simulation results.

## 1.3 Thesis Outline

In Chapter 2, basic concepts related to the netlist reduction algorithm and previous work about reducing circuit simulation time are presented. In Chapter 3, a hybrid circuit reduction algorithm is proposed first. The implementation process is described thoroughly after the algorithm description. The verification of the design and results comparison are discussed in Chapter 4. In Chapter 5, a conclusion is drawn for the thesis. Chapter 6 discusses the future work of this project.

Chapter 2

# Simulation Speeding Up Methodology

Circuit simulation accounts for a substantial portion of a whole circuit design process. In order to decrease the simulation time, various simulation speeding up methodologies have been proposed. In this Chapter, relative background concepts are explained first. Several methodologies from two aspects, improving device model and improving the simulation process, are studied after the background description.

## 2.1 Background

In this section, the relative background knowledge of the netlist reduction engine is introduced. Section 2.1.1 gives an introduction about the architecture of memory. Section 2.1.2 explains some concepts related to simulation and verification of designs.

### 2.1.1 Memory Architecture

The structure of memories is the reason why the netlist reduction is more efficient on SRAMs than general circuits. In this section, a short introduction of SRAM bitcell structure and SRAM architecture is present.

#### 2.1.1.1 SRAM bitcell structure

***Bistable Circuit:*** A bistable circuit is an electronic circuit that can hold two stable states (0 and 1). This structure is also in SRAM. As shown in Figure. 2.1, two inverters are connected in cascade along. The positive feedback in this structure is a basis of the storage capacity of SRAM [3].

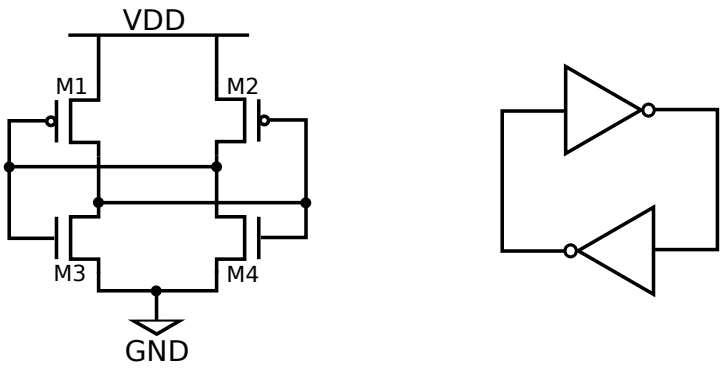



**Figure 2.1:** The bistable circuit schematics (right) and its symbol (left).Two inverters which consist of M1, M3 and M2, M4, are connected in cascade along, forming a positive feedback.

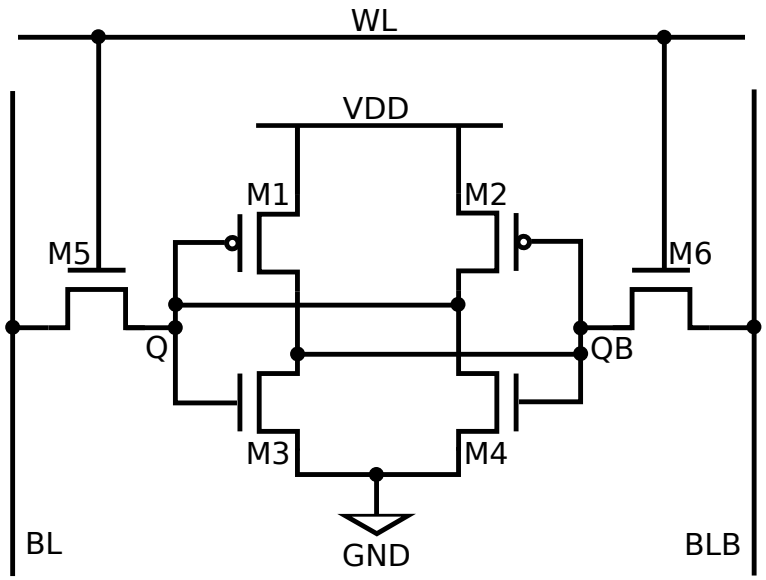



**Figure 2.2:** The single port SRAM bitcell. M1, M2, M3, and M4 comprise a bistable structure which can hold the value stored in the bitcell. M5 and M6 connect the BL/BLB with the bistable structure, allowing writing/reading value when WL is on.

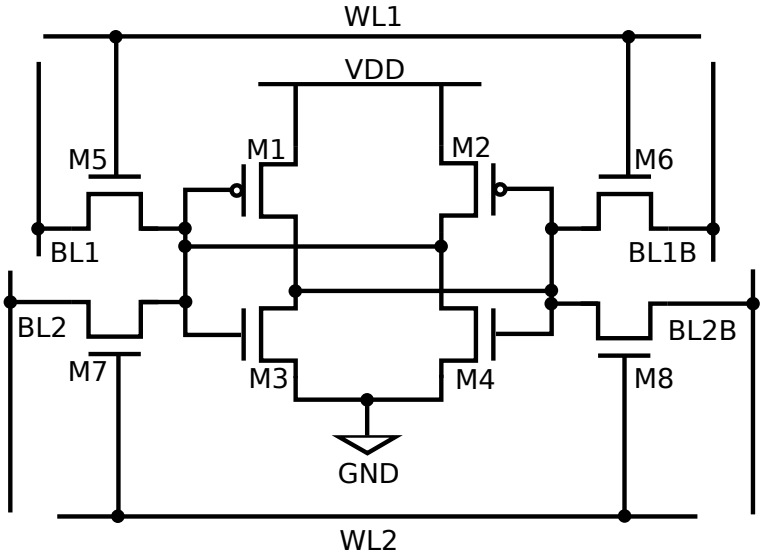



**Figure 2.3:** The dual port SRAM bitcell. M1, M2, M3, and M4 comprise a bistable structure which can hold the value stored in the bitcell. Four transistors (M5, M6, M7, M8) connect the bistable structure with the four BLs. Two parallel operations can be managed.

***Single port SRAM:*** A conventional 6-T single port SRAM bitcell is shown in Figure. 2.2. Six transistors are used to store one bit in this SRAM structure. By pre-charging/discharging Bitline (BL) and Bitline-inverse (BLB), the voltage of bitlines can be controlled. In write mode, BL will be set to 0/1 and BLB will be set to the opposite value, then by activating wordline (WL), the value can be stored in SRAM. When doing a read operation, two BLs need to be charged to 1 at first. The value stored in the bitcell can be known by detecting the voltage of these two BLs when WL is active.

***Dual port SRAM:*** A conventional 8-T dual port SRAM bitcell is shown in Figure. 2.3. The write operation and read operation of 8-T dual port SRAM bitcell is quite similar to the 6-T single port SRAM bitcell, but the 8-T dual port SRAM bitcell can manage two parallel operations due to the presence of two pairs of BL and two WL.

#### 2.1.1.2 SRAM architecture

SRAM is a High Replication Circuit (HRC), because bitcells are replicated in the whole circuit [4]. Netlist reduction can be very efficient on the SRAM circuit, due to this feature. It is also the reason why simulating reduced netlist can verify the functionality of the SRAM circuit.

***Memory Macro:*** A memory macro consists of a bitcell arrary and corresponding peripheral circuits. A memory macro is shown in Figure. 2.4. Bitcells in the same row[1] shares the same WL, and bitcells in one column[2] use the same BL. By selecting the WL and two BLs, a certain bitcell can be chosen for one read operation or write operation.

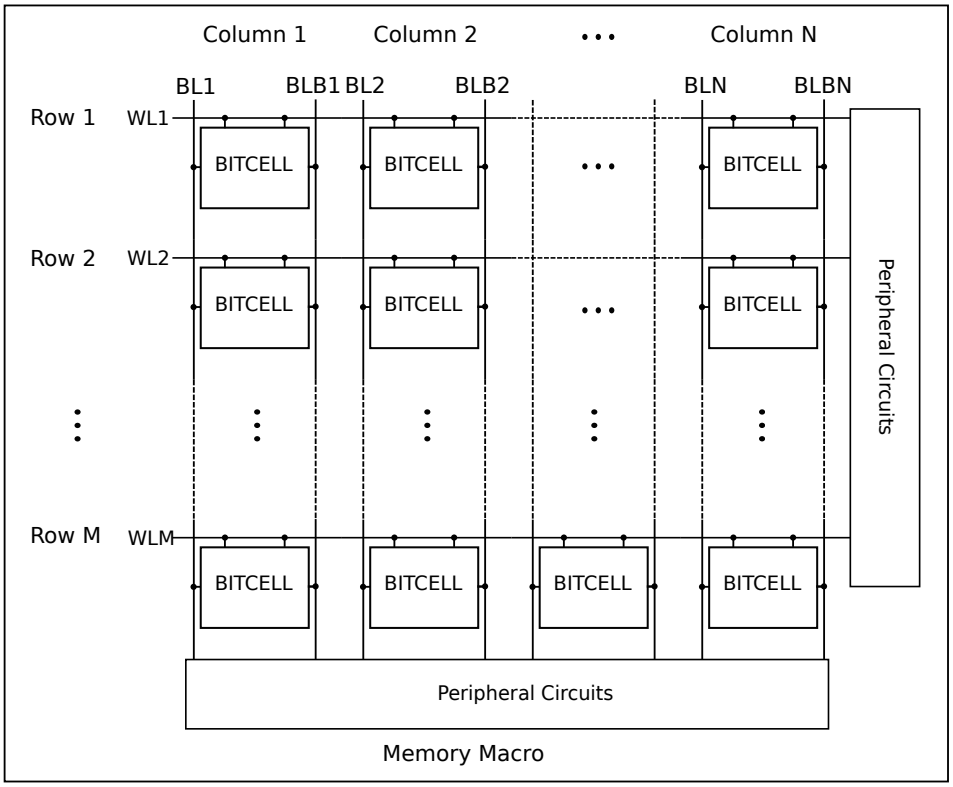



**Figure 2.4:** A memory macro. Bitcells in each row share one WL and bitcells in each column share the same BLs.

***Memory Bank:*** A memory bank is a unit of memory, which is commonly used in large-scale memory design. The capacitance on WL and BL can be very large if a large memory comprises one large bitcell array directly. This can lead to errors in write operation or read operation, such as writing a wrong value into the bitcell, and the speed of reading operation and writing operation can also be influenced.

Large memories normally consist of several memory banks and these memory banks are composed of small size bitcell arrays. A memory bank consists of several memory macros is shown in Figure. 2.5. The bitcells in one memory bank are se-

[1]In this paper, one row of bitcells means a row of bitcells on WL-direction
[2]In this paper, one column of bitcells means a column of bitcells on BL-direction.

lected by giving a certain address at the input during the write and read operation.

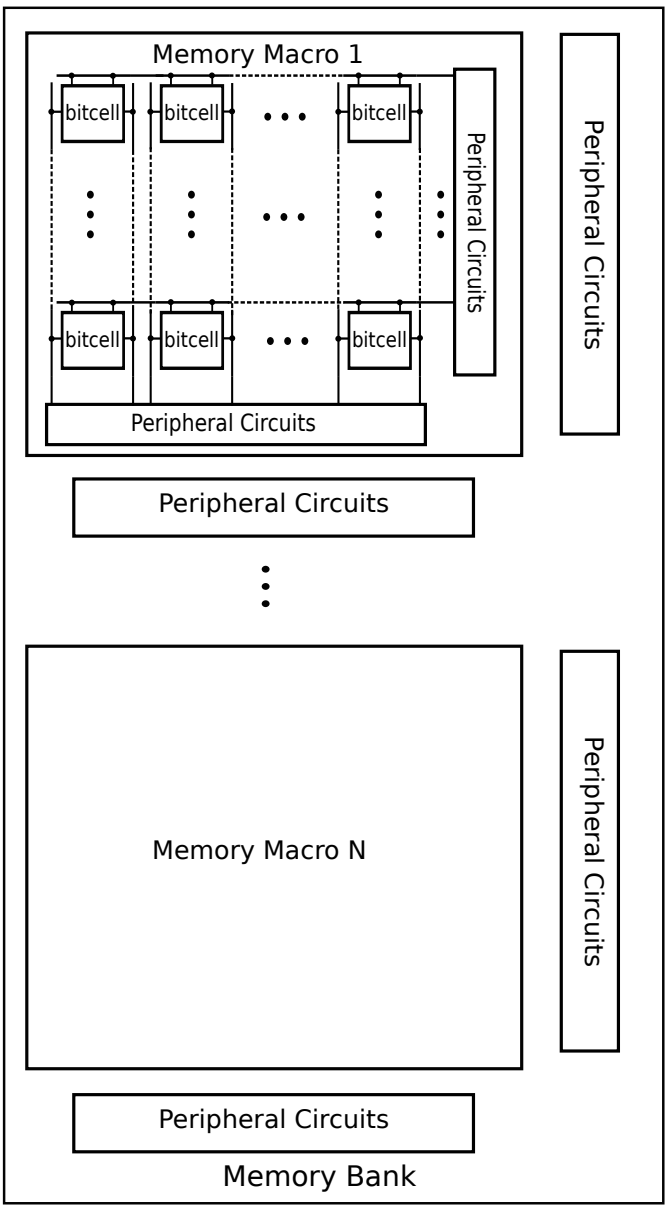



**Figure 2.5:** A memory bank consists of several memory macros.

The simulation time is very long due to a large number of transistors/components, although only one bitcell in one memory bank is required when doing the write or read operation. Therefore, the rest of the bitcells can be replaced with simpler models to reduce the total number of transistors. Take an 8 kb ($64 \times 64 \times 2$)

size memory as an example, which means there are 64 rows and 64 columns of bitcells in one memory bank and the memory consists of two memory banks. If one bitcell is the target bitcell of read/write operation, 8191 bitcells can be removed and only one bitcell remains during netlist reduction process. If the 6-T SRAM bitcell structure is used in this memory, the number of transistors in inactive bitcells are 49146. A large number of transistors in inactive bitcells can lead to a large amount of information to be processed during circuit simulation. The reduction will be more efficient when the size of SRAM is large, since bitcells takes the largest number of transistors in the whole SRAM circuit.

The SRAM architecture is also the reason why the functionality of the SRAM circuit can be verified by simulating the reduced circuit netlist. The peripheral circuit in the SRAM circuit takes a small number of transistors in the whole circuit, and some peripheral circuit blocks are shared by the HRC in the SRAM. For example, a row of bitcells can be activated by one WL and the WLs are controlled by the decoder, which means the decoder block can manage several rows of bitcells. It is able to verify that one row of bitcells can be activated by the corresponding WL by testing the behavior of one bitcell in this row. Therefore, the simulation results of the reduced circuit can be used to verify the feature of the whole SRAM circuit.

#### 2.1.1.3 Peripheral Circuits

The peripheral circuits are very important parts of the memory circuit. It consists of blocks with different functionalities, such as address decoders, sense amplifiers as well as timing and control blocks.

***Address decoder:*** Address decoder is a functional block which is responsible for WL selection. It is comprised of a lot of logic gates. Each input address pattern is referring to a certain WL of the memory. Normally, an X-input decoder is able to handle $2^X$ WLs [3].

***Sense amplifier:*** Sense amplifier is a functional block which outputs the value reading from the bitcell by sensing the difference on BLs. The sense amplifier can be used to reduce the operation delay because it can detect a small difference on the BLs and amplify it to large signal output. This means that the voltage of BLs does not need to be exactly 0, when 0 is reading out from the bitcell. Therefore, the time between WL activated and signal read out can be reduced [3].

***Timing and control:*** The operation of the memory consists of a series of sequential actions. For example, the bitlines need to be precharged before the WL is activated. Therefore, the timing and control circuit should be designed very carefully [3].

A large part of the peripheral circuit blocks can be removed during the netlist reduction process while reducing the memory to one bitcell or one row of bitcells.

In particular, some sensitive analog circuit blocks, such as the timing control block, are set as "do not touch" to avoid inside modification.

### 2.1.2 Simulation and Verification of Designs

The simulator is also one factor which lead to a long time simulation. In this section, concepts related to simulation and verification process is introduced, including the feature of netlist and simulators.

#### 2.1.2.1 Circuit Netlist

A netlist is a description of an electronic circuit. It contains the information about components, such as transistor models, and their connections. Netlists can be written in different formats, and the one we are dealing with is hierarchical netlist.

The circuit design is normally divided into small parts when the design is very large. These small parts, called subcircuits, are 'packaged' and used in the circuit as instances. In netlists, a subcircuit is declared only once and invoked as the circuit model of instance. Therefore, the subcircuit is in lower hierarchy and the circuits that invoke the subcircuit are in a higher hierarchy.

Listing. 2.1 and Listing. 2.2 shows an example of an inverter netlist for SPICE® simulator and Spectre® simulator respectively. The corresponding inverter schematic is shown in Figure. 2.6. The inverter is described as a subcircuit in the netlist. At the beginning of the subcircuit declaration, the subcircuit name and the ports of this subcircuit are mentioned. All the components in this subcircuit are described in the body. The description order of one component is the component name, the component connection, the component model and some related parameters. The main structures of these two netlists are very similar, although there is a small difference in the format. Netlists can be processed by a specific simulator. Therefore, the reduced netlist can be simulated directly.

**Listing 2.1:** An inverter netlist for SPICE® simulator

```
. SUBCKT Inverter out in vdd vdds gnd gnds
M0 out in vdd vdds transistor_model L=length W=width
M1 out in gnd gnds transistor_model L=length W=width
.ENDS
```

**Listing 2.2:** An inverter netlist for Spetre® simulator

```
subckt Inverter out in vdd vdds gnd gnds
M0 (out in vdd vdds) transistor_model L=length W=width
M1 (out in gnd gnds) transistor_model L=length W=width
end Inverter
```

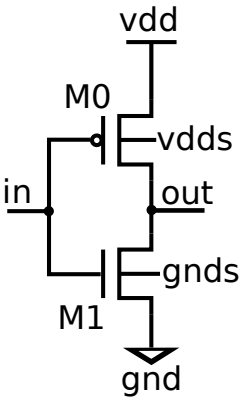



**Figure 2.6:** The schematic of an inverter.

#### 2.1.2.2 Simulator for processing netlist

There are different simulators that can be used to process circuit netlist files for different requirements.

***SPICE® simulator:*** SPICE® is a circuit simulator for verifying the integrated circuit design in detailed transistor-level. It can simulate the behavior of circuits by processing certain electronic component models. SPICE® simulation allows designers to verify if the circuit operation is exactly as it is expected before manufacturing [5].

***Spectre® simulator:*** Spectre® simulator is a SPICE-class circuit simulator that can analyze circuits in detailed transistor-level. It can provide high precision simulation results, but with long time and high memory occupancy at the same time [6]. For large-scale circuits with high complexity, the Spectre® simulator is not even able to process them in a general server, due to the huge amount of information to be calculated and to be stored.

***UltraSim® simulator:*** UltraSim® is a circuit simulator which can verify the functionality of large-scale hierarchical circuits. It is faster and with lower memory consumption compared with traditional simulators like Spectre®. The drawback is that the simulation results of UltraSim® are not as accurate as of the results of the Spectre® simulator [7]. Therefore, the UltraSim® can be used to check the functionality of huge circuits which are even not able to be run by the Spectre® simulator.

Simulators use complex device models to simulate the whole circuit, including active parts and inactive parts. Some inactive parts of the circuit do influence the active parts of the circuit somehow, and the influence comes from the resistance and capacitance characteristics of transistors. However, the simulator will perform

a very complicated calculation based on a corresponding transistor model library. This process is very time-consuming and memory-consuming since the intensive computation can generate a huge amount of data. This can be solved if the inactive part of the circuit is replaced by equivalent resistance and capacitance models.

## 2.2 Simulation Model Improving Method

Some researches focus on decreasing simulation time by improving the simulation principle itself. As explained in Section 2.1.2, a trade-off between speed and accuracy exits in different simulators. These methods are going to decrease the simulation time without sacrificing accuracy or increase the accuracy without sacrificing simulation time.

[8] proposed a new simplified transistor model, a finite-point-based transistor model, to handle the increasing transient simulation time. This model uses an analytical expression to describe the process impact and design variations for the finite key points in I-V (Current-Voltage) and C-V (Capacitance-Voltage) characteristics of a transistor. The finite data points, five critical data points specifically, are extracted based on physical meaning and their importance in circuit operation. As shown in Figure. 2.7, this model extrapolated the full I-V and C-V from limited data point to save the statistical simulation time. As a result, this model can reduce the simulation time up to nine times compared to Monte-Carlo simulations with BSIM (Berkeley Short-Channel insulated-gate field-effect transistor Model) [10, 11].

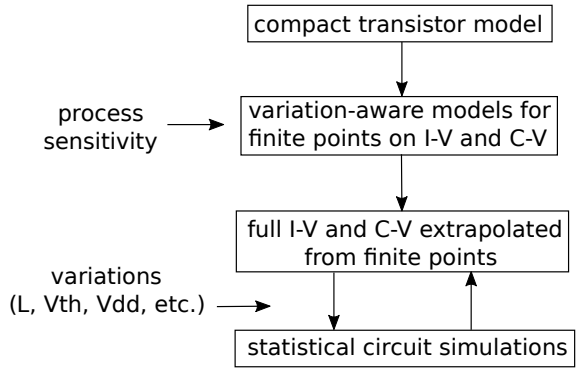



**Figure 2.7:** Application of the finite-point-based transistor model [8].

[9] proposed a new method to reduce the order of circuit equations in simulation processes for both linear and nonlinear circuits. In order to reduce the order of a circuit, a congruence transformation with the projection matrix is needed. This method speeds up the whole process by reducing the calculation cost in generating a projection matrix. It remodels the projection matrix at each time point in the

simulation. This means calculating a reduced Jacobian matrix directly in each Newton-Raphson iteration to avoid calculating the original size of the Jacobian matrix which will cost more time.

## 2.3 Netlist Reduction Method

The two methods mentioned in Section. 2.2 focus on device evaluation time, while some other methods focus on another factor of simulation runtime: circuit scale itself. These methods use different algorithms and different pre-configurations to achieve the same purpose: reduce the inactive parts of the circuit with fixed pre-confugurations. This is also the purpose of our engine.

### 2.3.1 Manual SRAM Reduction Method

The manual reduction methods always have an assumption: the architecture and details of the circuit is known before the reduction and modifications can be done manually. Two methods are studied in this section and they are both struggling with the complexity of the reduction process, the efficiency of the reduction and the accuracy of the reduction.

#### 2.3.1.1 Pure Reduction

[12] proposed a method based on a known specific structure. A critical path, the worst delay path, will be manually analyzed first. As shown in Figure. 2.8, the bitcell in the top right corner has the longest path to peripheral circuits (decoder) in this exact floorplanning. Then only the bitcells and peripheral circuit blocks on the critical path or neighboring paths will remain. This step is achieved by modifying and renaming basic circuit block modules. After function and Layout Versus Schematics (LVS) checking, the reduced circuit will be used for parasitic parameters extraction for post-layout simulation. For a $2K \times 32$ bit SRAM, this model can save 92% simulation time while keeping the tolerance below 5%. But this method has low flexibility for its fixed floorplanning requirement and critical path manually extraction. In addition, module based modifications can be time-consuming. And for a $n \times m$ cell array, (2n+2m-2) bitcells will remain which means $6 \times (2n + 2m - 1)$ more transistors will remain compared to some other methods like [13].

#### 2.3.1.2 Reduction with Equivalent Model

Some research reduced only the bitcells of the whole SRAM. [13] proposed a corresponding equivalent and simplified model of bitcell array, bitline, and wordline. The model will ignore peripheral circuits and do the substitution to the cell array

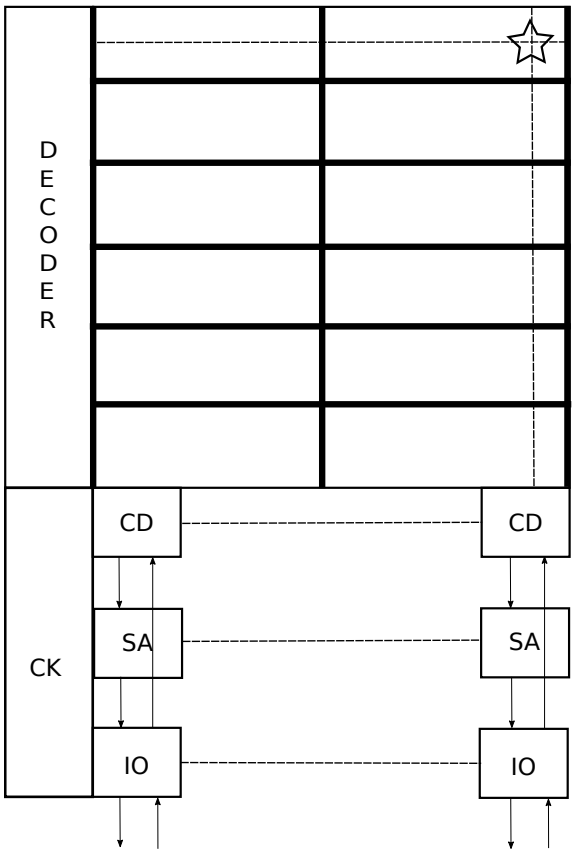



**Figure 2.8:** Redcution based on known structure and known critical path [12].

except for the positioned bitcell. The parasitical parameters of the layout will be extracted without cell arrays first, then the memory core will be replaced by the reduced memory core with corresponding parasitical parameters. As shown in Figure. 2.9, the reduced model of bitcell, bitline and wordline are based on layout netlist so the length and width of the line will also be considered.

The parameters can be calculated by the Equation 2.1 and Equation 2.2:

$$C_1 = C_2 = C_{ox} \times W_n \times L_n \times \varepsilon_0 \varepsilon_{ox} / t_{ox} \tag{2.1}$$

$$R = \rho L / W \tag{2.2}$$

The wordline model is shown in Figure. 2.10. The value of the resistance and capacitance shown in the above figure are calculated by the Equation 2.3 and Equation 2.4:

$$R_m = (R_1 + R_2) \times (C_1 + C_2) \times (m-1)/2 \tag{2.3}$$

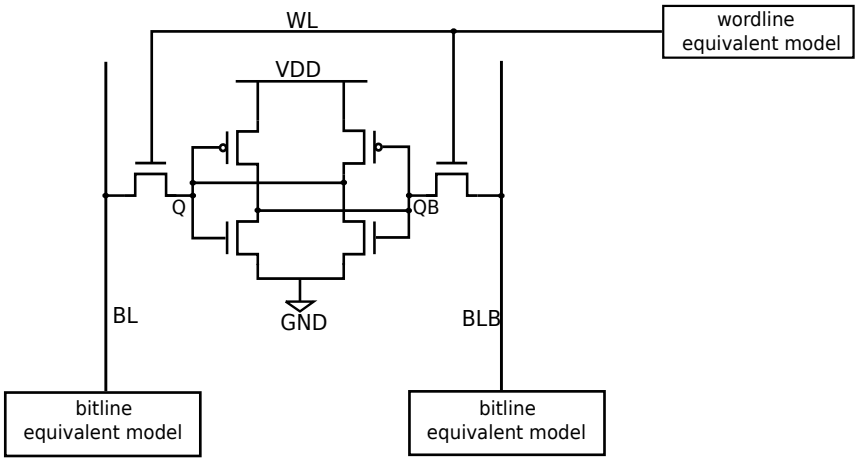



**Figure 2.9:** Simplified BL model and WL model of memory core [13].

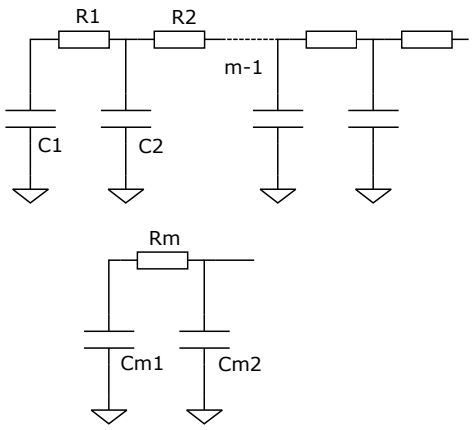



**Figure 2.10:** Distributed and lumped $\pi$ model of wordline [13].

$$C_{m1} = C_{m2} = (C_1 + C_2) \times (m-1)/2 \tag{2.4}$$

The bitline model is shown in Figure. 2.11. The value of the resistance and capacitance shown in the above figure are calculated by the Equation 2.5 and Equation 2.6:

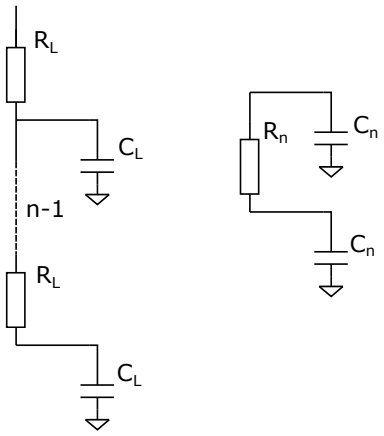



**Figure 2.11:** Distributed and lumped $\pi$ model of bitline [13].

$$C_L = C_J + C_{JSW} = C_j L_S W + C_{jsw}(2L_S + W) \tag{2.5}$$

$$R_n = R \times (n-1), C_n = C_L \times (n-1)/2 \tag{2.6}$$

This method focuses on layout and only cares about one cell array in the whole memory, while inactive peripheral circuits also take much unnecessary time to simulate. The advantage is that this model calculated parasitical parameters for memory core ahead to completely avoid extracting parasitical parameters from bitcells.

## 2.3.2 Automatic Circuit Reduction Method

The two methods discussed in Section 2.3.1 need to be done manually which is time-consuming in circuit design flow. The methods discussed below can reduce the circuit automatically with corresponding configurations before the reduction.

### 2.3.2.1 DRAM Reduction Based on Pre-defined Architecture

[14] presents a DRAM reduction tool which can reduce the DRAM circuit automatically based on pre-configurations. The reduced DRAM architecture will be built first, then the corresponding circuit block extracted from the original netlist will be filled in the architecture. Two architectures are available in this tool, one

is Block-Selection Type Memory Array Structure and another one is Parameter-Selection Type Memory Array Structure. Bitlines and wordlines in the architecture will be replaced by wire models chosen from six different wire models. As for the bitcells, the reduction method can be chosen by the user from two options. One is replacing the inactive cell by a simplified model, another one is keeping all the cells and redistributing the multiple factor parameter M. One disadvantage of this tool is that several DRAM architectures are predefined in the tool which is not flexible. Only one bitcell is fixed during each simulation and this need to be defined before simulation each time. The flow of the tool is shown in Figure. 2.12.

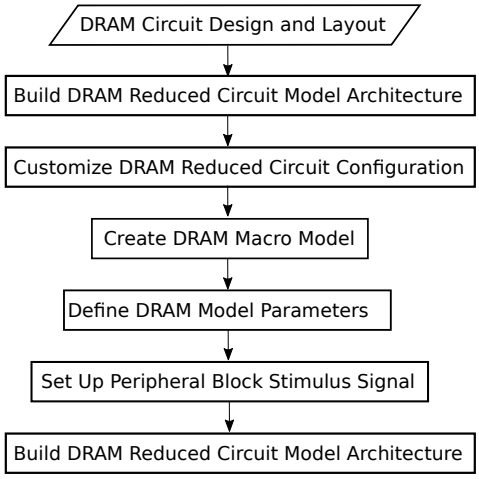



**Figure 2.12:** Auto DRAM process flow [14].

#### 2.3.2.2 Genral Circuit Reduction Based on Fixed Stimuli

[15] presents a recognition method to trace signal flows of a general circuit and reduce the circuit according to the signal flow. The flow of this method is shown in Figure. 2.13.

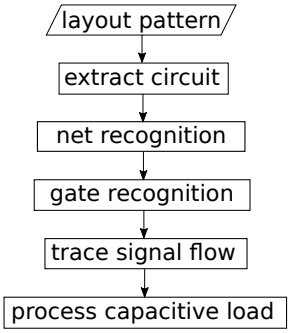



**Figure 2.13:** Gate recognition method flow for active path tracing.

In the first step, the circuit will be recognized from layout pattern data. With given stimuli, the signal flow can be traced by using their new presented function recognition algorithm. During the gate reduction and parasitics reduction based on a specified critical path and signal flow, the terminated net will be considered as a capacitive load.

As for the function recognition algorithm, logic blocks will be recognized first and the logic tree will be formed as shown in Figure. 2.14. This algorithm has two restrictions to avoid some special function blocks (clock) becoming a vertex in the logic tree. As for the capacitive load, one example model is that if a net connected to the gate terminal of a transistor, the source and drain terminals of the transistor will be connected together.

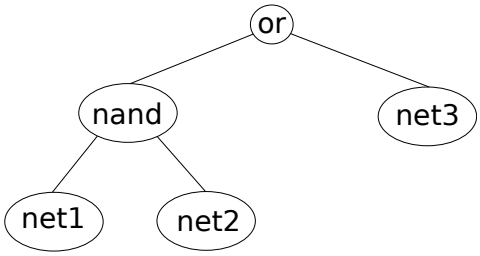



**Figure 2.14:** Logic tree of function recognition algorithm.

# Chapter 3
# Implementation of Reduction Engine

The whole flow of the design and details of the engine implementation are described in this chapter. After a brief description of the configuration, the major reduction process will be introduced. Adding RC models and tracing critical paths will also be discussed in this chapter as add-ons.

## 3.1 Algorithm Design

Although the methods discussed in Section. 2.3 are based mostly on layout, the circuit analysis and reduction process will always be done after filtering the parasitics data of the layout pattern first. An automatic hybrid netlist reduction focuses on schematic netlists is presented in this section.

### 3.1.1 Clustering

The first step of clustering is detecting special units and clustering each of them. Different special units will be labeled differently to be recognized. A special unit can be defined in a configuration file before the reduction. In the second step, the remaining transistors that are connected to each other by source terminals or drain terminals will be clustered together. One thing that needs to be mentioned is that all the power and ground connections will not be considered as normal nets. Considering the special structure of SRAM bitcell arrays, the bitcell needs to be recognized as a special unit before beginning normal clustering. This predetecting can ensure that the bitcells connected to the same bitline are considered as different clusters, which will increase the reduction efficiency.

Another function embedded in the engine is the "do not touch" circuit setting. In the configuration file, users can input the name of the module that they do not want to be modified. When the engine is flattening the circuit, the "do not touch" sign will be labeled on all the transistors in the specific module. After clustering, the clusters which contain "do not touch" labeled transistors will be considered as "do not touch" clusters. This is a trait to be detected in the following steps.

In the end, the structure of the circuit will look like in Figure. 3.1.

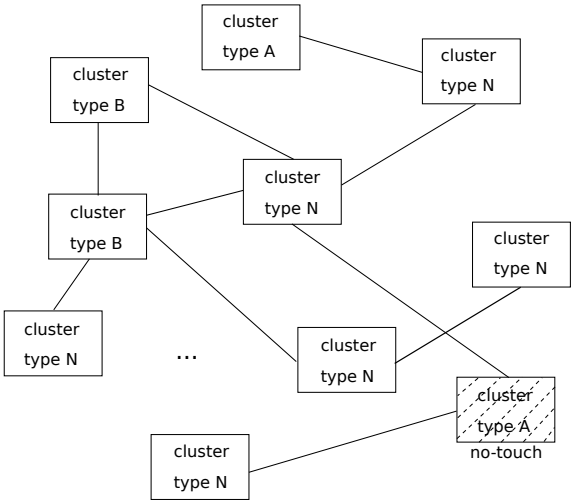



**Figure 3.1:** Circuit structure after clustering.

### 3.1.2 Active Path Tracing

At the beginning of the path tracing, the ports of the clusters received from the last step will be defined. If a port in a given cluster is connected to the gate of one of the transistors in it, it will be defined as a gate port. Similarly, if a port in a cluster is connected to the source or drainl of the transistors in it, it will be defined as a channel port. A port can be a gate port and a channel port at the same time.

In the first step, multiple target bitcells can be pre-configured by BL and WL information. This function makes different bitcell simulations after only one reduction process possible. Four net names "BL1, BL2, WL1, WL2" are considered as one input information group. The two WL net names can be the same if the bitcell is connected to only one WL. In order to find the target bitcells, bitcell clusters connected directly to any net of the start group will be correspondingly labeled. Those bitcells which are labeled by all four nets in one start group are the target bitcells. Except for these target bitcells, all the other bitcells will be reduced. For example, "BL3, BLB3, WL2, WL2" is the input information group1 and "BL4, BLB4, WL3, WL3" is the input information group2. As shown in Figure. 3.2, only the bitcells labeled by [1,1,1,1] will remain.

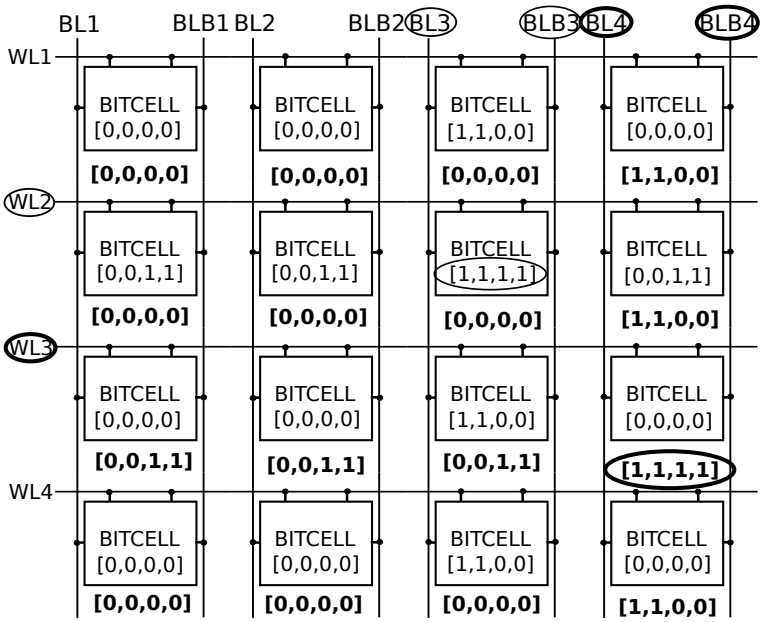



**Figure 3.2:** Fixed target bitcells. One type circles lead to one input information group and one target bitcell. Only bitcells labeled [1,1,1,1] remain.

In the second step, all the nets in an input information group and all the observation points will be considered as "start" nets. From the start nets, all the clusters that may affect the "start" nets will remain. As shown in Figure. 3.3, the cluster whose output is connected to the "start" net will be recorded and traced, except for the bitcell clusters. The input nets of the to-be-traced cluster will be considered as new "start" nets and be recorded and traced. The loop chain will be stopped and all the clusters in the chain will be labeled as active if the chain meets the input of the whole circuit or the target bitcell.

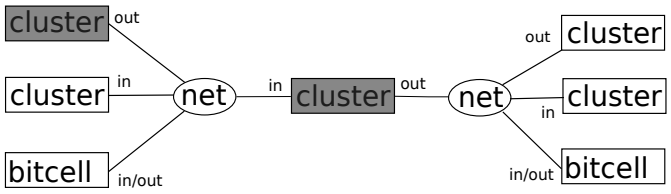



**Figure 3.3:** Active clusters tracing example. The right net is a "start" net and clusters in shadow squares will be recorded as possible active clusters.

### 3.1.3 Add Equivalent Model of Bitcell

For the reduced circuit, a huge amount of reduced bitcells on target bitlines and word lines will cause huge capacitance loss. Capacitance loss on general circuits nets can be ignored temporarily compared with those on target BL and WL. As shown in Figure. 3.4, 0 or 1 is stored in a bitcell when the wordline is not enabled. For BL, the 1-store-side-bitline will have a leakage current path to the power while the 0-store-side-bitline will have a leakage current path to the ground. For WL, the capacitance loss depends mostly on the capacitance between the gate and the body of the transistor.

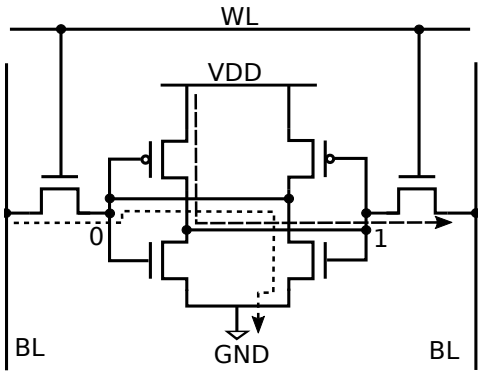



**Figure 3.4:** Leakage in inactive bitcell.

To compensate for the capacitance loss, a pure capacitance model shown in Figure. 3.6 is used. The capacitors which connect to one net will be combined together in the end. $C_W$ and $C_B$ are the capacitance of the corresponding single bitcell. And the capacitance value of a single bitcell can be pre-defined or extracted from single bitcell netlist.

In order to extract the capacitance more accurately and technology independently, an add-on function is embedded in the engine. All the bitcells which connect directly to the target WL and BL will not be reduced at first as shown in Figure. 3.5. This first edition of reduced netlist will be simulated first by Ocean® script [17] and the capacitance information of one bitcell will be extracted from the simulation results automatically. After this extraction, the first edition of the reduced netlist will be reduced again until only target bitcells remain and the sum of capacitance being added to the target BL and WL. One advantage of this method is that the first reduced edition can also be used as a final edition in a high accuracy required situation.

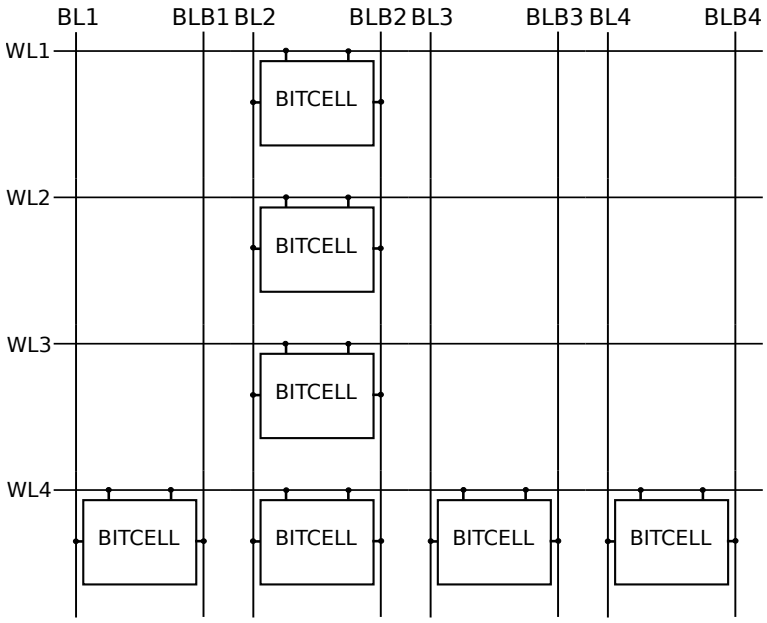



**Figure 3.5:** First step of reduction for accurate capacitance extraction.

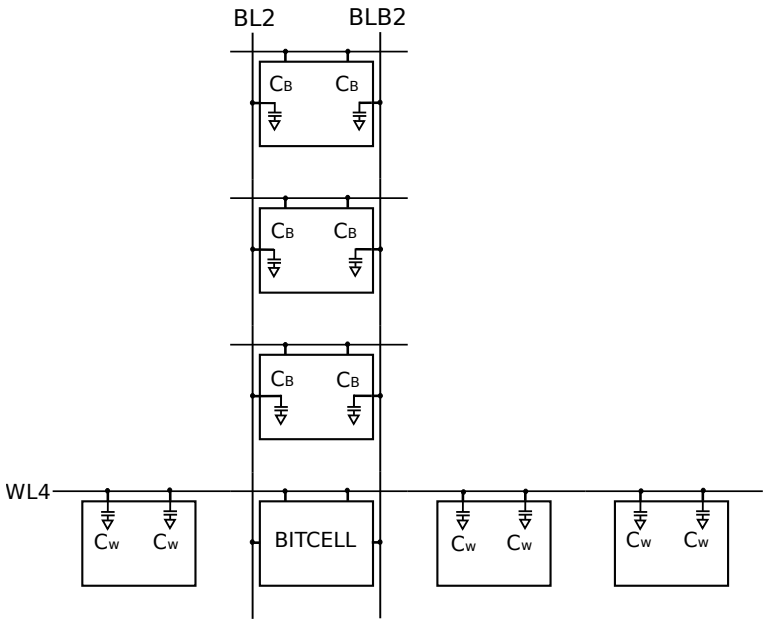



**Figure 3.6:** Capacitor model for reduced bitcells which connected directly to target BLs and WLs.

In Chapter 4, simulation results of different SRAM circuits will be listed in tables and be compared with each other. The efficiency, function, and accuracy will also be compared with some other methods published by others.

## 3.2 Configuration Before the Reduction

An information text document can be read in to give variables corresponding values after users input the information.

***Path information:*** The path of an original netlist is necessary for the engine to find the original netlist. The path of the output folder is necessary for saving all the output results files, including reduced netlist and other log files of the reduction process.

***Transistor model name list:*** All the used transistor model names should be included in this list. This configuration can help to identify transistors from instances when the engine read in the original netlist.

***Bitline and wordline information:*** The information of bitline and wordline is used to fix target bitcells. So for each target bitcell, information of four net names (two bitlines and two wordlines) will be provided. If one bitcell has only one wordline, same net names are acceptable. To avoid unnecessary bitcells being selected, this information must be given in a group. For example, if eight net names belong to two groups are given together, four bitcells will be fixed rather than two.

***Module names of no-touch blocks:*** For those circuit blocks that the users want to keep in a black box without any modification, their name should be given. All the transistors and instances in no-touch blocks will remain exactly the same after the reduction.

***Net name of power/input/output:*** The ports of top-level connections to power/input/output is informed before the reduction to save the time of identifying the port type.

***Input and output net name of critical path:*** The input and output net can be any net inside the circuit. The add-on function, critical path extraction, will start from the output net to the input net specified in this configuration.

All the net name information in the document above should have a complete form with hierarchy, like "I0.I3.net1".

## 3.3 Netlist Reduction

The process of the reduction is divided into four parts: preprocessing the data, classifying the circuit, reducing the circuit, and reconstructing the circuit to output. All of these steps are realized by Python, a high-level programming language.

### 3.3.1 Data Pre-processing

Figure. 3.7 shows the procedures of preprocessing the data read from an original netlist.

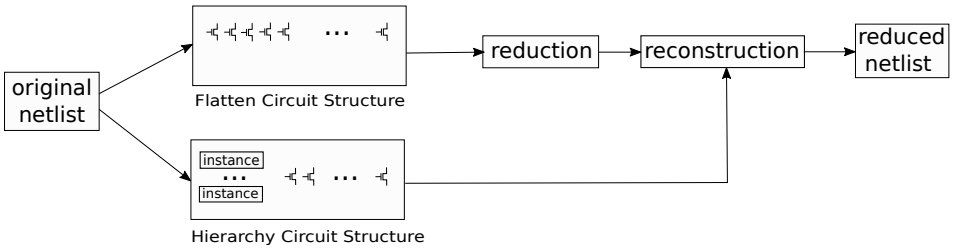



**Figure 3.7:** Two structures to store the netlist data. Flatten circuit structure for processing and hierarchy circuit structure for new netlist reconstruction.

The circuit information is described in a netlist with hierarchy. This means that the netlist consists of one or more circuit blocks and these circuit blocks can invoke each other. The circuit information is extracted from a netlist and stored by a particular data structure. In this project, two data structures are used. One structure is called "flatten circuit structure". All the devices in the circuit will be instantiated as single transistors or other basic electronic components with only information of connection. There is only one circuit block in this structure without sub-blocks. Data stored in this structure will be used for the major circuit analysis. Another structure is called "hierarchical circuit structure". In this structure, the invoking relationship between subcircuits in the original netlist will be stored. Data stored in this structure will be used as a reference when the engine reconstructs the reduced netlist. This will keep the reduced netlist well-structured.

### 3.3.2 Classification

To reduce the complexity of the circuit which will be reduced, a classification of components needs to be done. In this process, all the transistors are classified into different clusters. Figure. 3.8 shows the flow of this process.

Bitcell detection is run at the beginning of the classification process. By searching the bistable circuit structure which is a common structure in SRAM bitcells, all the components which can be SRAM bitcell are found. Then these components

are verified if they are SRAM bitcells by checking if only drain or source of two NMOS are connecting with the bistable circuit. This method can be further developed to detect different bitcell structures like an 8-T SRAM bitcell structure shown in Figure. 2.3. All the transistors in one bitcell will be labeled and can be easily recognized in the further process. Transistors, except those in bitcells which are connected together by drain or source, are considered as a cluster since they will influence each other during operation. After giving these clusters different labels, the classification process is finished.

Bitcells are considered as special clusters, due to the structure of the bitcell array. As shown in Figure. 3.9, bitcells which are sharing the same BL will be considered as one cluster if we do not do bitcell detection before clustering. This huge cluster will keep more inactive transistors in the reduced netlist and decrease the efficiency of the netlist reduction engine.

The reduction process can be messy and complicated without this step since it can reduce the types of port for one cluster. If the reduction process is implemented in transistor level, a port of transistors can be input, output or both. After classification, only the ports used for clusters connection need to be considered, and the direction of the path can be determined easier during reduction.

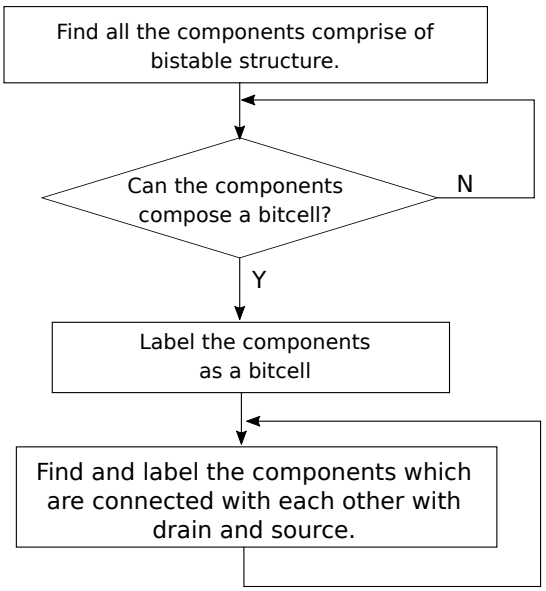



**Figure 3.8:** The flow of classification process.

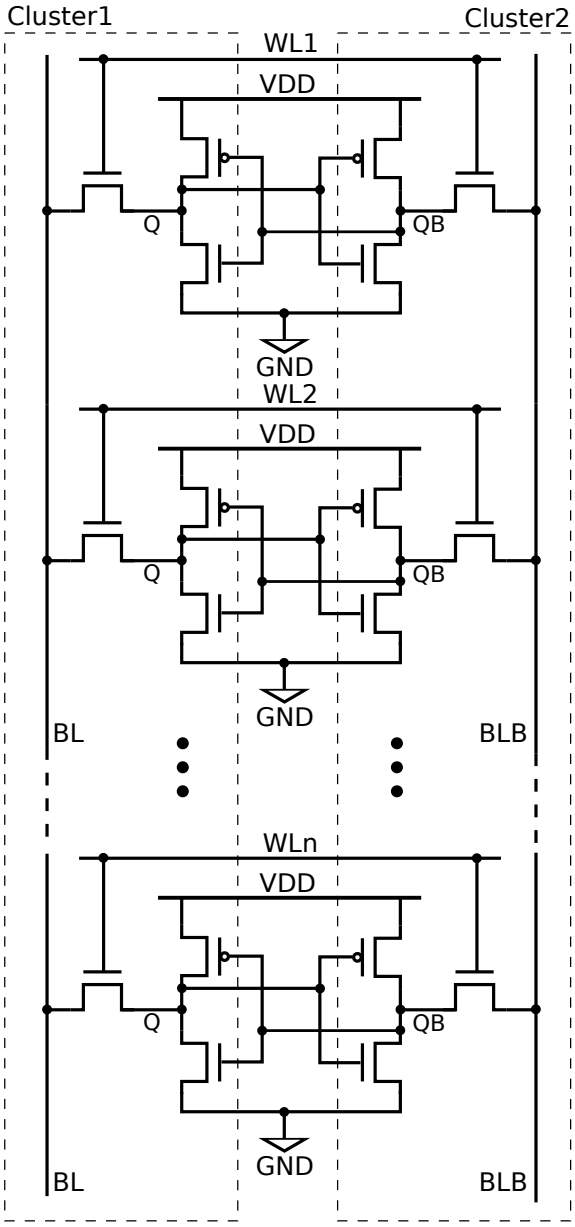



**Figure 3.9:** Bitcells sharing the same BLs.

### 3.3.3 Active Path Tracing

According to the configuration file, transistors in blocks which are not expected to be reduced will be labeled first. Figure. 3.10 shows the algorithm flow of the active path tracing.

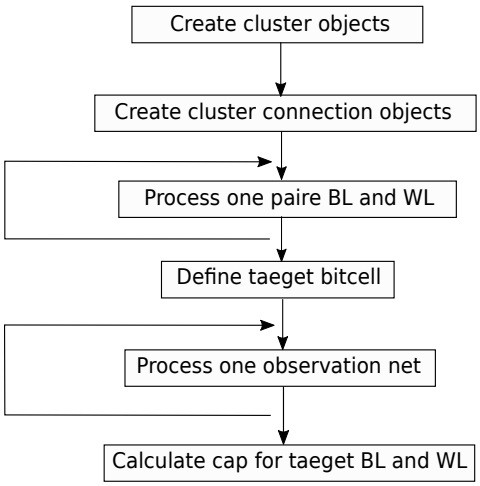



**Figure 3.10:** Active path tracing flow

The first step is creating cluster objects. One cluster object contains all the information of the transistors that have been classified into this cluster. As for the ports of these transistors, only the transistor ports that connect between this cluster and any other cluster will be defined as the ports of the cluster. These ports will be divided into two types. One type is the port that can only be the input of the cluster circuit, and another type is the port can be the input and also can be the output of the cluster circuit. All this information will be stored in the cluster objects. The cluster connection objects are simple net objects that only contain the name information of the clusters which connect to the net.

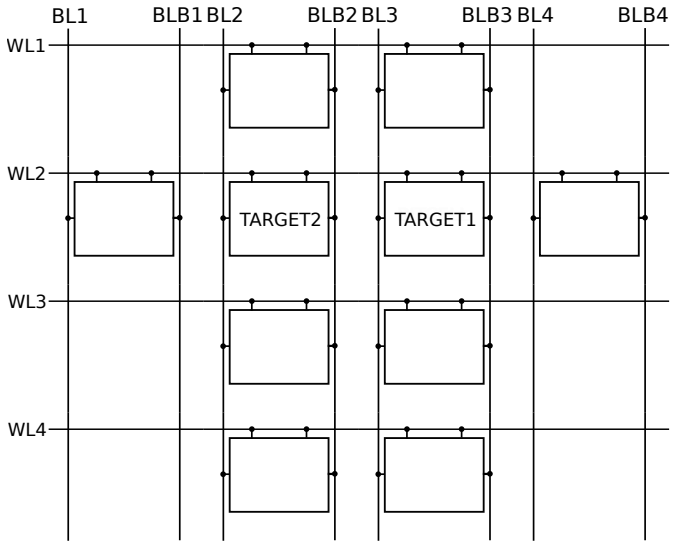



**Figure 3.11:** Two bitcell targets

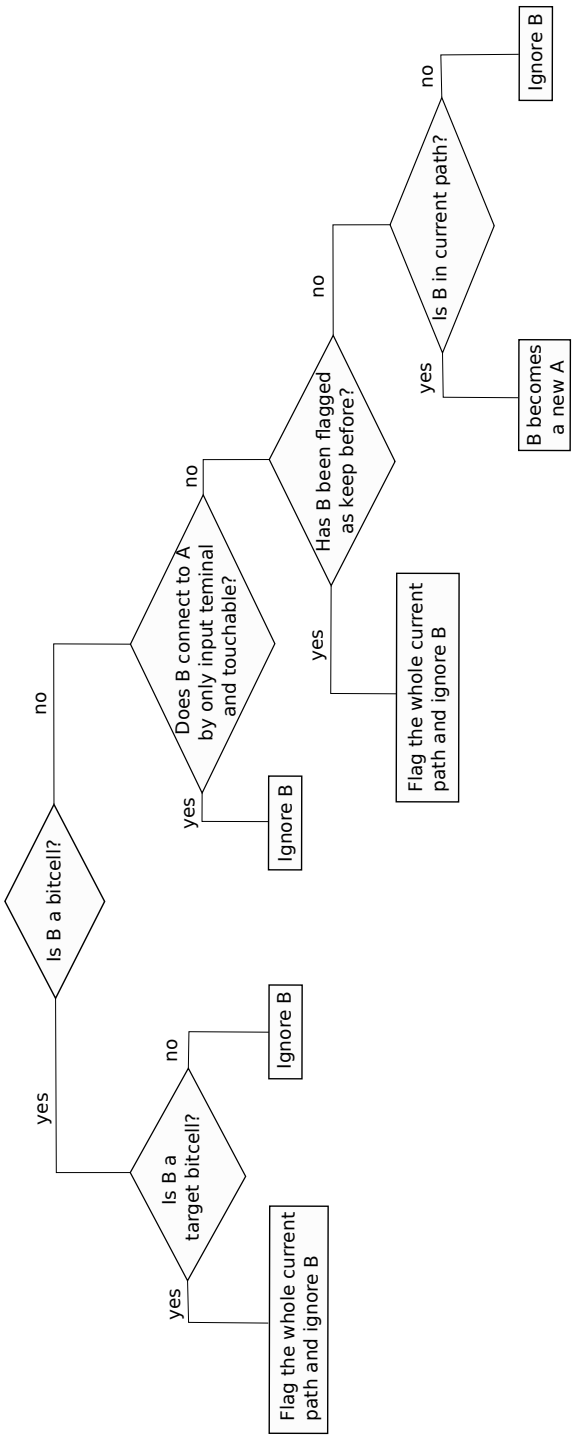



**Figure 3.12:** Active cluster tracing flow. A represents the latest testing cluster in one tracing path and B represents any cluster connect to A.

After creating objects, the possible active cluster path will be traced depending on several pairs of BL and WL information. And the bitcells which connect to the target BL and WL will be flagged correspondingly. Then the target bitcells can be fixed. In addition, the number of deleted bitcells in target BL or WL will be calculated with the capacitance value of one bitcell for further capacitance calculation. Take the cell array in Figure. 3.11 for example, the target bitcell 1 and 2 are fixed, the weight of BL2 and BLB2 is 3 and the weight of WL2 is 4. The last step is the possible active cluster path tracing from observation nets.

The main process of reduction is the possible active cluster tracing, and this process will be divided into two parts. One is tracing from the BL and WL information in the configuration file and another one is tracing from observation nets in the configuration file. The algorithm flow is shown in Figure. 3.12.

In the diagram, A represents the latest testing cluster in one tracing path, and B represents any cluster that connects to A. If B is a target bitcell cluster, the whole tracing path from start point to A will be kept and the engine will keep searching for other clusters which connect to A. If B is a normal bitcell cluster, it will be passed and the path will keep searching for the next cluster connected to A. When B is not a bitcell cluster, if it connects A only by its only-input-terminal and it is not configured as a "do not touch" cluster at the beginning, the B cluster will be ignored. Then if B has not been tested before, it will become the new A of the whole path and the steps above will be repeated until all the possible active clusters are found and flagged.

### 3.3.4 Reconstruction

In this process, the hierarchical structure of the reduced circuit is built. The flow is shown in Figure. 3.13.

After the reduction operation, all the transistors which are considered as active are labeled. The information about circuit hierarchy is recorded and shown via the name of devices in flatten circuit. Therefore, the tool can easily identify which hierarchy the transistors are in and relocat them in the hierarchical circuit. After this relocation, the reduced circuit can be reconstructed via labeled transistors and hierarchical circuit structure. This step rebuilds the structure of the reduced netlist and keeps the netlist structure hierarchical. Moreover, when checking the structure of the reduced netlist is necessary, the structure rebuilding can make this checking easier.

Due to the instantiation of all the components in the hierarchical circuit structure, repeated circuit model declaration will cause a huge size netlist for the new reconstrcuted reduced circuit. By comparing the information in the circuit models, those repeated circuit models will be replaced by one single model. This step helps reduce the size of the reduced netlist file and makes the verification of the netlist easier.

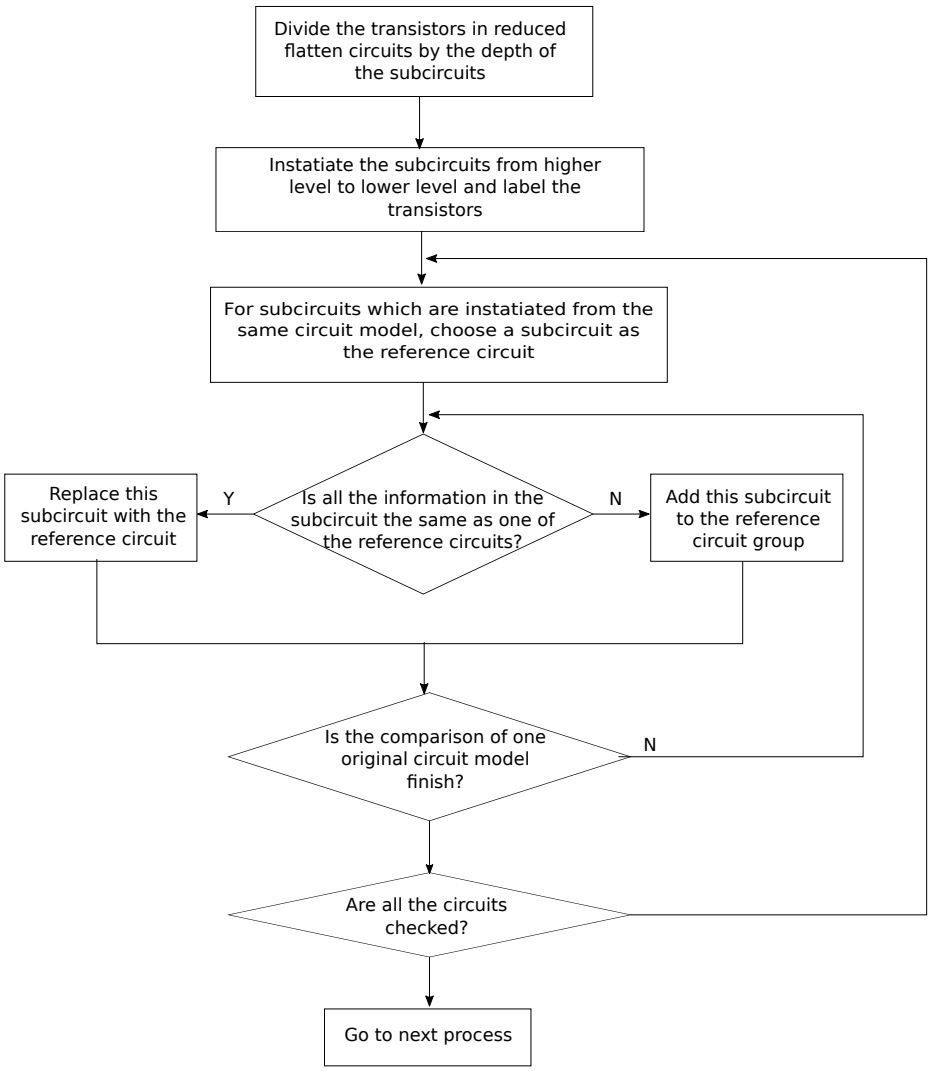



**Figure 3.13:** The flow of reconstruction process.

At this point, the whole netlist reduction process is finished and a new reduced netlist file can be output.

## 3.4 Build RC Models for Redundant Circuits

The capacitance extraction is necessary for building the capacitance model for deleted circuits. The BL and WL capacitance model is the most determining fac-

tor for the accuracy of the netlist reduction engine.

Only the capacitance of transistors is concerned when extracting the capacitance since the reduction aims at a schematic level and the wires in the schematic are considered as ideal wires. In Cadence Virtuoso®, there is a function called cap table extraction, which can extract the capacitance of all the circuit nodes at one moment during a simulation.

Capacitance is a dynamic characteristic of transistors. Therefore, the value of it changes depending on the working condition of the transistors. During reduction, a row and a column of bitcells are remained for this extraction, as shown in Figure. 3.14. A is the target bitcell. The bitcells labeled as D has been deleted in previous steps, and bitcells labeled with B and C will be deleted after extraction.

This process has been integrated into the reduction engine. The capacitance can be extracted and add back to the circuit automatically if it is required. The capacitance value can also be given previously and the extraction process will be skipped.

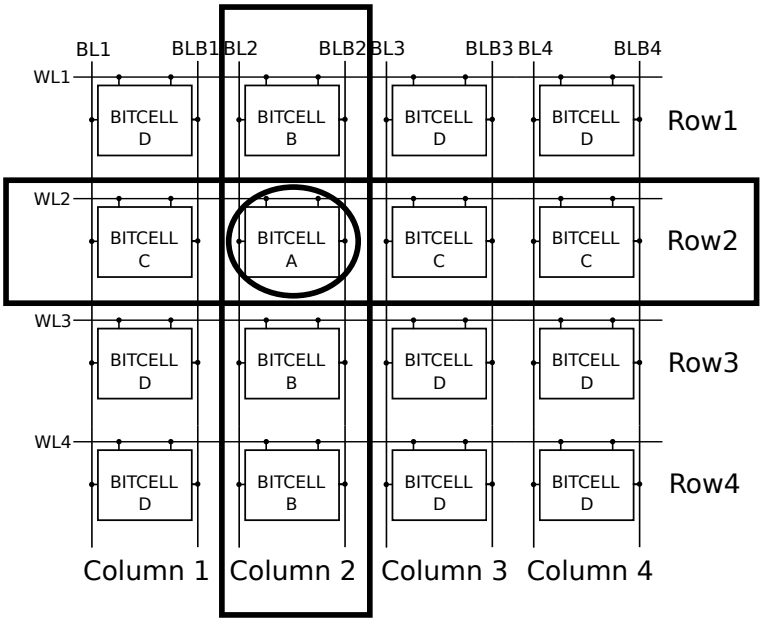



**Figure 3.14:** Different strategies for different bitcells.

## 3.5 Extract Critical Path for a Circuit

Extracting critical path for a circuit is implemented as an add-on in the netlist reduction engine. The critical path is the worst delay path of the circuit. The delay depends on the resistance and capacitance mostly. So the delay of the circuit depends on the parasitics on the critical path. The critical path extraction in this project is an add-on function and it is only an approximate critical path. The circuit is divided into small units. A unit can be a transistor or a cluster. We assume every unit have a weight. The weight represents the length of the critical path from input to the unit. The former unit who provide the largest weight to the recent unit will be recorded in the recent unit. Furthermore, the weight can be calculated differently according to different algorithms. Take the diagram in Figure. 3.15 for example, an input “I” and an output “O” will be known as the function input. The first step is comparing the weight of “b” and “e”. When the engine found “b” and “e” have not been defined a weight, it will keep comparing the weight of “a” and “c” for “b”. This step will repeat until tracing to the input. Then the units will have weight one by one from the input to output. For example “b” will be given a weight equal to 4 and the name of “c” will be stored in “b” because “c” can provide 3 to “b” while “a” can only provide 2 to “b”. At last, “O” will get 4 from “b” and the critical path will be “I”-“a”-“c”-“b”-“O”.

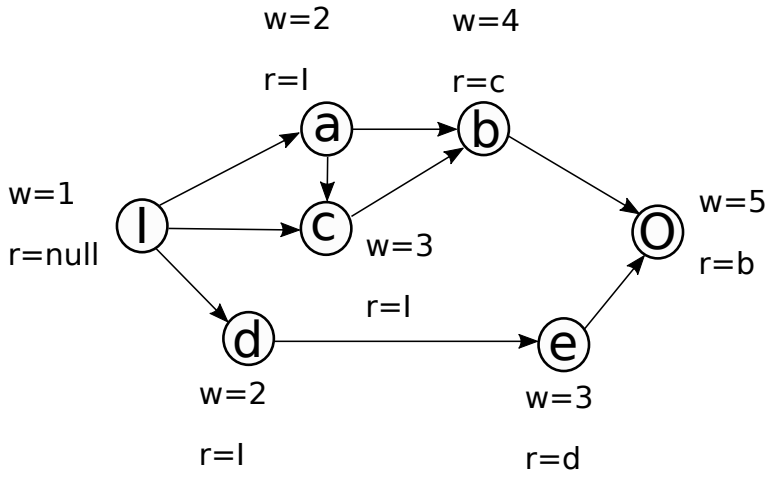



**Figure 3.15:** Critical path tracing example. "w" is the weight of current cluster and "r" is the former biggest weight cluster name. Weight can be calculated differently according to different algorithms.

# Chapter 4
# Verification of Reduction Engine

In this chapter, a verification flow and corresponding scripts are introduced first. Results of the verification are listed in tables and compared between different SRAM sizes and different reduction methodologies.

## 4.1 Flow of Verification

The verification flow is shown in Figure. 4.1. After inputting the original netlist and configuration to the reduction engine, the reduced netlist and the log file recording reduction time will be generated. After the original netlist and the reduced netlist being simulated by the same simulator, a raw file and simulation time for each netlist will be output as the results of a simulation. Based on the raw data, important constraints will be calculated by the constraint calculator. These constraints will be compared between an original netlist and a reduced netlist to show the accuracy of the reduction engine.

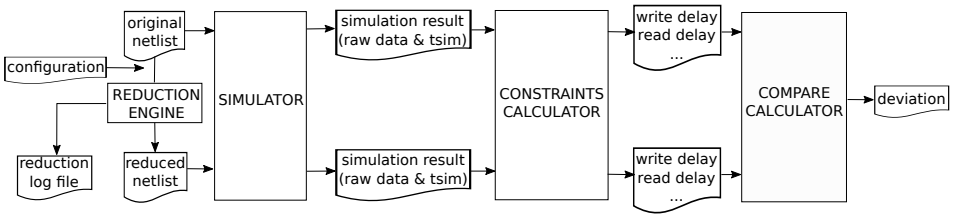



**Figure 4.1:** The flow of verification process.

In the verification phase, various size SRAM circuits are used to test the reduction engine as shown in Table. 4.1. As for the simulator, Spectre® is used for all the netlists except for the original netlist of 32 kb ($32 \times 32 \times 32$) size SRAM for its unfeasible simulation time. For this large netlist, UltraSim® is used as the simulator. As for the constraint calculator, the delay of the read and write operation, and the rise time and fall time of the switch of observed signals will be calculated. Figure 4.2 shows the calculation method of delay for the two signals and the fall time of a signal switch.

**Table 4.1:** Tested SRAM types

| Peripheral structure | Null | Type A(simple) | Type B(complex) |
|---|---|---|---|
| 256 b ($16 \times 16$) | ✓ | ✓ | |
| 512 b ($16 \times 32$) | ✓ | ✓ | |
| 1 kb ($32 \times 32$) | ✓ | ✓ | |
| 4 kb ($64 \times 64$) | ✓ | ✓ | |
| 8 kb ($64 \times 64 \times 2$) | ✓ | ✓ | |
| 32kb ($32 \times 32 \times 32$) | | | ✓ |

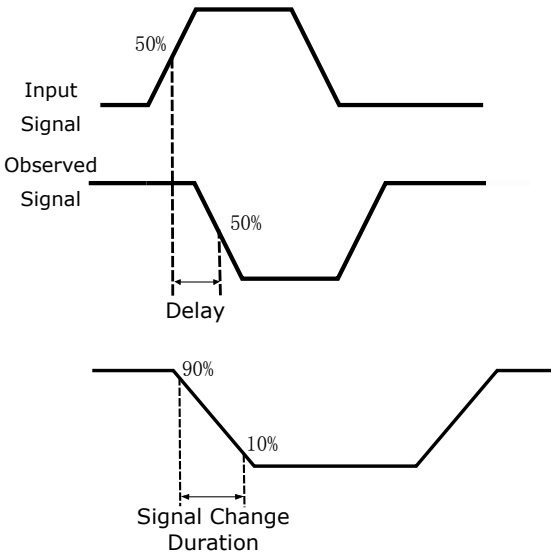



**Figure 4.2:** An example waveform for measurement.

As described in Section. 3.1.2, the active path tracing is based on the target bitcells rather than stimulus. When one or more target bitcells are fixed, the stimulus can change. In the verification flow, a circuit with a testbench in the netlist can also be read in and be separated to circuit and testbench. The circuit part will be sent to the reduction engine and the testbench part will be kept and added back to the reduced netlist after the reduction process as shown in Figure 4.3. The testbench of the netlist can be exactly the same because the ports of the highest level are not changed in the reduction.

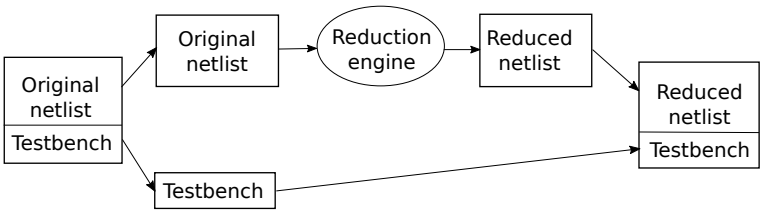



**Figure 4.3:** Testbench separation

## 4.2 General Function Verification

The first step of verification is the functional verification. The expected result of this step is fundamental. If the reduced circuit can read the data stored in target bitcells and write data into target bitcells correctly, the general function verification will be considered as correct. More detailed verification such as read delay tolerance and write delay tolerance will be calculated and compared in the next section. The result of function verification is shown in Figure 4.4 and Figure 4.5, the upper waveform is the original circuit signal and the lower waveform is the reduced circuit signal. Figure 4.4 is the simulation results of a reading operation and Figure 4.5 is the simulation results of a writing operation. All the coordinate values are being normalized but we can still see the basic function is fulfilled in this example.

## 4.3 Results Comparison and Analysis

In this section, the reduction engine performance and simulation results will be compared in different aspects.

### 4.3.1 Data Formation

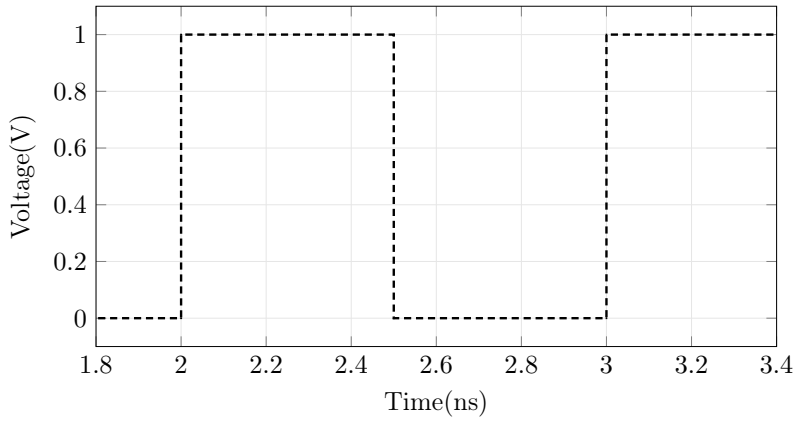
All the data presented in the table are normalized for better intuitive comparison. For example, initial raw data is listed in Table 4.2.

**Table 4.2:** Rate Calculation Example

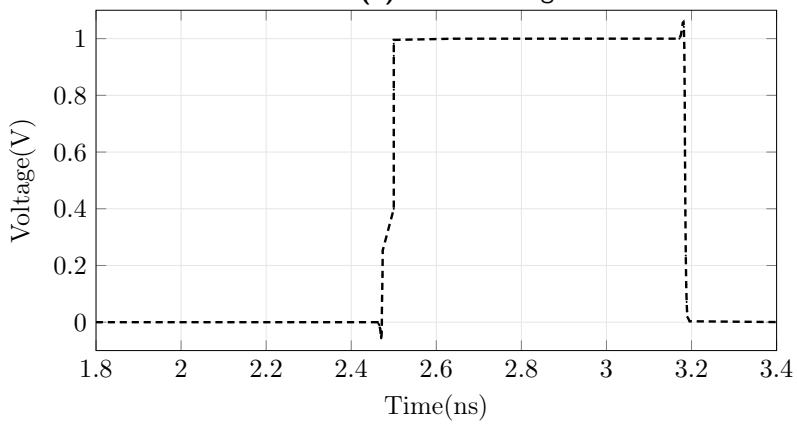
| Original netlist result | Reduced netlist result | Rate(%) |
|---|---|---|
| A | B | $(1 - B/A) \times 100$ |

Parameters are calculated from raw data using the formulas below:

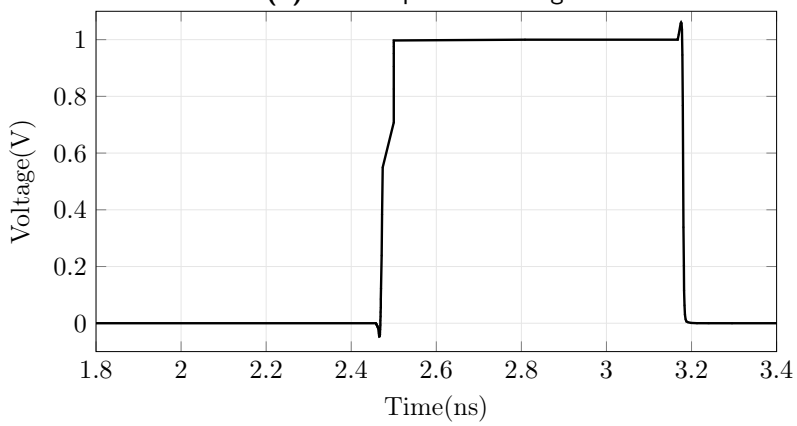
$$tolerance = \frac{original\ netlist\ result - reduced\ netlist\ result}{original\ netlist\ result} \tag{4.1}$$




**(a)** The clock signal

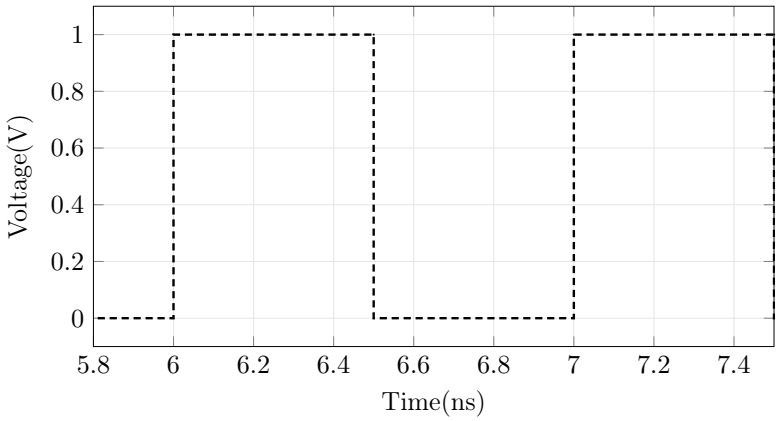



**(b)** The output of the original circuit

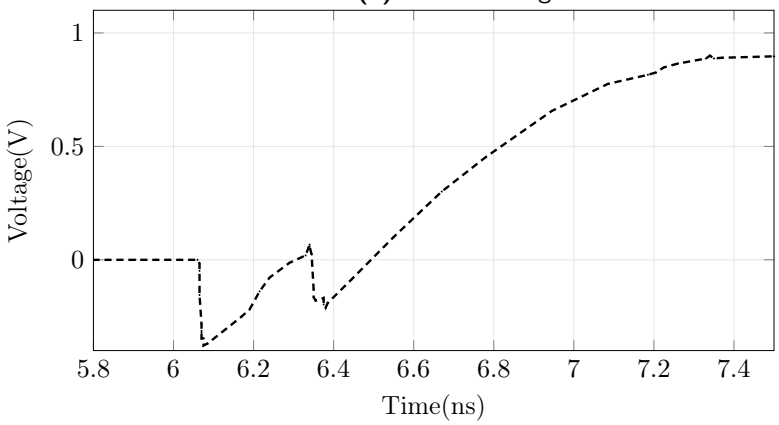



**(c)** The output of the reduced circuit

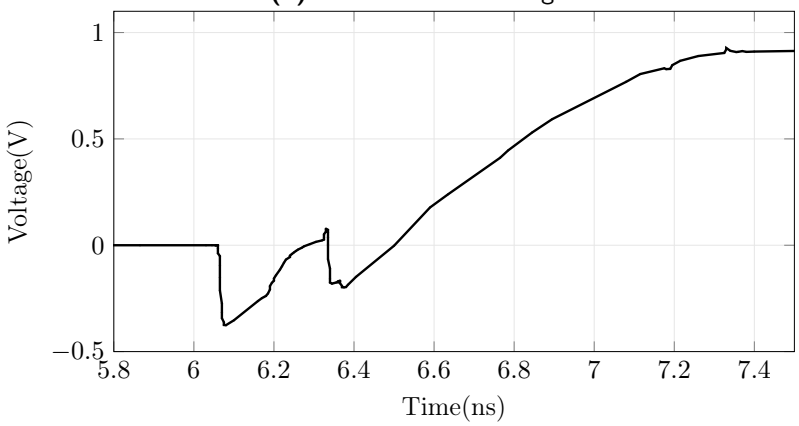
**Figure 4.4:** Comparison between simulation results of original circuit and reduced circuit during read operation. (a) is the reference clock signal, (b) is the output data reading out of the original circuit. (c) is the output data reading out of the reduced circuit.




**(a)** The clock signal




**(b)** The result of the original circuit




**(c)** The result of the reduced circuit

**Figure 4.5:** Comparison between the simulation results of the original circuit and the reduced circuit during the write operation. (a) is the reference clock signal, (b) the signal transition when writing a value into the bitcell of the original circuit, (c)the signal transition when writing a value into the bitcell of the reduced circuit.

$$reduced\ t_{sim} = \frac{original\ sim\ time - reduced\ sim\ time}{original\ sim\ time} \tag{4.2}$$

In order to measure differences between original netlists and reduced netlists in an equal way, one circuit (including an original netlist and a reduced netlist) must be simulated by one simulator, but different simulators can be used for different circuits. However, different circuits must be simulated by one simulator when it comes to the comparison between of different size SRAMs. As shown in Table 4.3, the reduced simulation time and tolerance will be influenced by using different simulators. In general, UltraSim® is faster and less accurate so the netlist reduction has less reduced simulation time when using this simulator. For example, after same reduction process, a 4 kb SRAM reduced 96% simulation time using Spectre® while only 52.31% reduced using UltraSim®. This difference comes from different simulation principles inside different simulators. So in the next sections, all the example SRAM circuits will be simulated using Spectre® except for a 32 kb SRAM because the original netlist of this 32 kb SRAM is unable to be simulated by Spectre® since it will cause a processor memory overflow. To make a fair results comparison between an original netlist and a reduced netlist, both the original netlist and the reduced netlist of this 32 kb SRAM will be simulated using UltraSim® and this is marked in the tables when the results are simulated from UltraSim®.

**Table 4.3:** Simulation results influenced by different simulators.

| Size | 4 kb($32 \times 32 \times 2 \times 2$) | | 2 kb($32 \times 32 \times 2$) | |
|---|---|---|---|---|
| Simulator | Spectre® | UltraSim® | Spectre® | UltraSim® |
| Reduced $t_{sim}$ | 99.65% | 81.06% | 96.31% | 52.31% |
| Tolerance | 1.62% | 2.56% | 1.24% | 3.87% |

### 4.3.2 Peripheral Circuit Structure and Processing Method Comparison

Normally, SRAM consists of bitcells and peripheral circuit blocks controlling the reading and writing operation. Most researches on SRAM netlist reduction focuses only on reduction of the bitcells while keeping all the peripheral circuit remained. In this section, we take a 4 kb SRAM as an example to compare different ways to process peripheral circuits.

Table 4.4 shows three methods to process the peripheral circuit of SRAM. The reduction method on bitcells are exactly the same and the reduction on a peripheral circuit is the only variable.

**Table 4.4:** Different mthods to process peripheral circuits

| | Algorithm |
|---|---|
| Method 1 | Keep the active clusters in the peripheral circuits |
| Method 2 | Keep both active clusters and inactive clusters which directly connects to active clusters |
| Method 3 | Keep the whole peripheral circuit |

Table 4.5 shows the simulation results using these three different methods for a 4 kb (64×64) SRAM. Write delay is used for tolerance calculation. The write delay in this situation is the time difference between decoder enable signal inversion and bitline signal inversion. The peripheral circuit transistors account for 6.2% of the whole circuit. The difference among different methods is slight in this table because of the small amount of peripheral circuit in this SRAM. But we can see that if we reduce the netlists for the peripheral circuits, the transistor number and simulation time will be reduced. If we use method2, the accuracy will improve but much more simulation time will be needed. According to these results, method1 is used in our engine.

**Table 4.5:** Different peripheral process methods comparison for 4 kb ($64 \times 64$) SRAM with a simple peripheral circuit (6.2%)

| | Method 1 | Method 2 | Method 3 |
|---|---|---|---|
| Reduced transistors | 95% | 94.16% | 93.76% |
| Reduced $t_{sim}$ | 96.8% | 96.5% | 96.1% |
| Tolerance | 3.35% | 3.30% | 3.30% |

Table 4.6 shows the simulation results for a complex structured 32 kb ($32 \times 32 \times 32$) SRAM. The peripheral circuit transistors account for 19.2% of the whole circuit. For a complex peripheral SRAM, the reduction of a peripheral circuit will bring more efficiency. The engine can reduce almost 10% transistors if reducing the inactive part of the peripheral circuit. But due to the huge amount of peripheral circuitry and simulator changing (for large scale netlist), the simulation time will reduce less than the former SRAM under the same condition.

### 4.3.3 Simulation Results Comparison for Different Size SRAMs

Applying the same reduction method on different size SRAMs will give us different results. In this section, the reduction time and reduced simulation time will be compared for different size SRAMs first and accuracy comparison follows.

**Table 4.6:** Different peripheral process methods comparison for 32 kb ($32 \times 32 \times 32$) SRAM with a complex peripheral circuit (19.2%)

| | Method 1 | Method 3 |
|---|---|---|
| Reduced transistors | 91.4% | 80.8% |
| Reduced peripheral transistors | 19.93% | 48.5% |
| Reduced $t_{sim}$ (UltraSim®) | 73.52% | 68.48% |
| Tolerance (UltraSim®) | 0.5% | 1.07% |

#### 4.3.3.1 Reduction Time and Simulation Time

The reduction time and simulation time for different size SRAMs are shown in Table 4.7. The first two SRAMs have the same simple structure, and the third SRAM has a complex structure. The upper four rows are the time spent on different reduction steps, and the fifth row is the total time spent on reduction.

**Table 4.7:** Reduction time and simulation time comparison

| | 4 kb | 8 kb | 32 kb |
|---|---|---|---|
| Memory bank structure | $64 \times 64$ | $64 \times 64 \times 2$ | $32 \times 32 \times 32$ |
| Flatten | <1s | <1s | 4s |
| Classification | 1s | 5s | 162s |
| Active path trace | 5s | 15s | 184s |
| Reconstruction | <1s | <1s | 6s |
| Reduction process time | 7s | 21s | 256s |
| Reduced $t_{sim}$ | 96.6% | 98.7% | 73.52%(Ultrasim®) |
| Reduced transistor | 94.2% | 95.9% | 91.4% |
| Peripheral/total | 6.3% | 5.2% | 19.2% |

From the table above, the engine has a better reduction efficiency on a large, regular SRAM structures, both in terms of reduced transistors and reduced simulation time. For different structures, the amount of peripheral circuitry will influence the reduction efficiency. Take the 32 kb ($32 \times 32 \times 32$) SRAM for example, the peripheral circuit accounts for 20% of the total transistor number, so the percentage of the reduced transistors of the whole circuit is less compared to other structures. Additionally, UltraSim® is used to simulate the 32 kb SRAM instead of Spectre® because the circuit is too large to run the Spectre® simulation. For a one-time simulation, the total reduced time can be calculated by Equation 4.3. If an iterative simulation is run, the reduced total time will be much closer to 1 according to the Equation 4.4.

$$reduced\ total\ time = 1 - \frac{reduction\ time + reduced\ sim\ time}{original\ sim\ time} \tag{4.3}$$

$$reduced\ total\ time = 1 - \frac{reduction\ time + reduced\ sim\ time \times N}{original\ sim\ time \times N} \tag{4.4}$$

As for the different reduction steps, the active path tracing is the most time consuming because of the iteration algorithm. The classification step will be slower if the structure is more complex or the size is larger, but it depends more on the structure complexity. The flatten and reconstruction costs less than 10% of total reduction time.

In a nutshell, the engine can reduce more simulation time for larger sized memory with the same peripheral structure under the same simulation condition (simulator and processor). When the size of memory is fixed, the engine will have lower efficiency if the peripheral structure is simpler. As for the reduction time, tracing the active path is the most time consuming and it depends on the scale and complexity of the circuit. But for iterative simulations, the reduction time can even be ignored compared with total simulation time.

#### 4.3.3.2 Accuracy

The tolerance of simulation results between the original netlist and the reduced netlist is considered as the criteria of the engine accuracy. As for the simulation results of an SRAM circuit, read and write delay are measured. Read delay is the time delay between reading enable inversion and read output signal inversion. Write delay is the time delay between write enable inversion and value inversion inside the target bitcell. The tolerance for different size SRAMs is shown in table 4.8.

**Table 4.8:** Tolerance when keeping one column and one row of bitcell

| | 1 kb | 4 kb | 8 kb |
|---|---|---|---|
| Memory bank structure | $32 \times 32$ | $64 \times 64$ | $64 \times 64 \times 2$ |
| Read delaytolerance | 0.8% | 1.91% | 1.9% |
| Write delay tolerance | 0.03% | 0.05% | 0.05% |

The results in the table above are from a reduction method that keeping one column and one row. The tolerance depends on the single cell array most, the bitcells number connected on one bitline or wordline. The more bitcells connected on one bitline or wordline, the larger the tolerance will become.

## 4.4 Comparison with Other Methodology

For the limited results of different reduction methodology being published, these results are compared based on different-size-memory in table 4.9. Except for simulation time and tolerance, the reduction is finished automatically or manually, and whether the method can be used in general circuit or not is also compared in the table below.

**Table 4.9:** Simulation results comparison with other methodology

| Method | Size | Reduced $t_{sim}$ | Tolerance | Auto | General |
|---|---|---|---|---|---|
| Our engine | 2 kb ($16 \times 16 \times 8$) | 93.2% | 4.3% | Yes | Yes |
| [12] | 64 kb ($2k \times 32$) | 90% | 5% | No | No |
| [13] | 2 kb ($16 \times 16 \times 8$) | 97.3% | 2.4% | No | No |
| [15] | Not mentioned | >80% | >10% | Yes | Yes |

Reducing circuit manually is a good option if the circuit is small, but when the circuit is in large-scale, manual reduction can be complex. It requires the person who is doing the reduction have a deep understanding of this circuit, so the person can know which part of the circuit should be reduced. Moreover, for those subcircuits which have been invoked repeatedly, instantiation of these subcircuits should be considered. This can be complicated and can take a long time. Therefore, a manual reduction is normally for reducing bitcells and adding equivalent RC models. Besides, there is a greater probability of tolerance in manual reduction compared with processing with automatic reduction engine.

During a design process, various data structures and methods have been tried and the one we presented in previous parts is the most efficient one. In this part, comparisons between these methods will be introduced.

The reduction process now is implemented on a flatten circuit. In the beginning, it was supposed to operate on a hierarchical circuit structure. The components are labeled and the instances are instantiated when they are found on the path.

The advantage of this method is that the data structure will be simple. The reconstruction function is not needed since the process is working on a hierarchical structure. On the other hand, the drawback of this method is obvious. It can be complicated when tracing a path from low hierarchy to high hierarchy, from example, from WL to input. The information of which hierarchy this instantiated subcircuit is invoked is needed when tracing a path from low hierarchy to high hierarchy, but it is hard for the tool to distinguish which hierarchy or which instance the target subcircuit is corresponding to if the subcircuit has been invoked in different hierarchies. To process components in a different hierarchy, the method of flattened circuit structure comes out.

# Chapter 5
# Conclusions

A netlist reduction engine was designed and tested in this project. The verification focuses on the efficiency of the netlist reduction engine and the functionality of the reduced circuit. To evaluate the efficiency, netlist reduction was run on different size circuits. The reduced simulation time and reduced transistor number were compared in table 4.7. The amount of remaining transistors number is less than 10% of the transistor number in the original circuit, while the simulation time is decreased to under 10% compared with the simulation time on the original circuit. Another factor we concerned is the tolerance between the simulation result of an original circuit and a reduced circuit. The comparison of an SRAM circuit with peripheral and the comparison of different size SRAM can be seen in table 4.5 and table 4.8. The tolerance will increase with size increasing.

In conclusion, the netlist reduction engine is able to reduce the simulation time efficiently by reducing the inactive circuit, and the differnence of simulation results between the original circuit and the reduced circuit can be very small depending on the accuracy of the replaced circuit model.

# Chapter 6
# Future Work

This project focuses on pre-layout netlist reduction. For future work, a netlist reduction aiming at post-layout simulation can be implemented. The post-layout netlist needs one more pre-processing step to recognize the circuit and remove the parasitic parameters. The RC model of the layout netlist will also be more complex to develop because the wire is not ideal in the post-layout netlist.

For the bitcell detection, a more bitcell type can be detected besides 6-T SRAM bitcell. For example, 8-T SRAM can be detected based on a similar detecting algorithm. And circuit which has a similar array structure as memory can also be reduced in an efficient way by detecting a configured pattern.

Additionally, a more accurate RC model for a general circuit can be developed to reduce more transistors other than keeping all the clusters that connect directly to active clusters.

# References


[1] V.G. Oklobdzija, *Digital Design and Fabrication*, Chapter 7.3 High-Performance Embedded SRAM2007, CRC Press.

[2] P.Girard , A.Bosio , L.Dilillo , S.Pravossoudovitch , A.Virazel, *(2010) Basics on SRAM Testing. In: Advanced Test Methods for SRAMs*, Springer, Boston, MA.

[3] J.M. Rabaey, A.Chandrakasan, B.Nikolic, *Digital Integrated Circuits: A Design Perspective*, 2nd Edition, Chapter 7.2, Page 330.

[4] A. Singhee, R. A. Rutenbar, *Extreme Statistics in Nanoscale Memory Design*, Charpter 1.2, Page 2, Springer.

[5] N. Laurence, and D. O. Pederson. *SPICE (Simulation Program with Integrated Circuit Emphasis)*, 1973.

[6] Cadence® *Spectre® Circuit Simulator Reference*, Version 18.1. Avalible: `https://support1.cadence.com/tech-pubs/Docs/spectreref/spectreref18.1/spectreref.pdf`

[7] Cadence® *Virtuoso UltraSIm Simulator User Guide*, Version 18.1. Avalible: `https://support1.cadence.com/tech-pubs/Docs/UltraSim_User/UltraSim_User18.1/UltraSim_User.pdf`

[8] M.Chen, W.Zhao, F.Liu, Y.Cao, *FinitPoint_ based Transistor Model: A New Approach to Fast Circuit Simulation*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 17, no. 10, pp. 1470-1480, Oct. 2009.

[9] T.Mine, H.Kubota, A.Kamo, T.Watanabe, H.Asai, *Hybrid Reduction Technique for Efficient Simulation of Linear/Nonlinear Mixed Circuits*, Proceedings Design, Automation and Test in Europe Conference and Exhibition, Paris, France, 2004, pp. 1327-1332 Vol.2.

[10] C. Z. Mooney, *Monte Carlo Simulation*, [Elektronisk Resurs]. SAGE, 1997.

[11] B.J. Sheu, D.L. Scharfetter, P.-K. Ko, M.-C.Jeng, *BSIM: Berkeley short-channel IGFET model for MOS transistors.*, IEEE Journal of Solid-State Circuits, vol. 22, no. 4, pp. 558-566, Aug. 1987.

[12] X.Jing, R.Yao, *A Fastsimulation Model for Post-layout SRAM*, 2007 7th International Conference on ASIC, Guilin, 2007, pp. 1197-1200.

[13] Z.Zhou, G.Zhang, *The Fast Simulation Model of SRAM*, 2006 8th International Conference on Solid-State and Integrated Circuit Technology Proceedings, Shanghai, 2006, pp. 1333-1335.

[14] W.H.Kao, X.C.Gao, R.Hamazaki, H.Kikuchi, *A Modeling and Circuit Reduction Methodology for Circuit Simulation of DRAM Circuit*, Records of the 1995 IEEE International Workshop on Memory Technology, Design and Testing, San Jose, CA, USA, 1995, pp. 15-20.

[15] G.Yokomizo, C.Yoshida, M.Miyama, Y.Motono, K.Nakajo, *A New Circuit Recognition and Reduction Method for Pattern Based Circuit Simulation*, IEEE Proceedings of the Custom Integrated Circuits Conference, Boston, MA, USA, 1990, pp. 9.4/1-9.4/4.

[16] A.Blotti, F.Mannozzi, R.Roncella, R.Saletti, F.Tinfena, *Gate Recognition and Netlist Reduction for Switch-level Simulation of Dynamic Bit-level Systolic Arrays*,2001 Southwest Symposium on Mixed-Signal Design (Cat. No.01EX475), Austin, TX, USA, 2001, pp. 56-60.

[17] Cadence® *OCEAN Reference*, Version ICADVM18.1. Available: `https://support1.cadence.com/tech-pubs/Docs/oceanref/oceanrefICADVM18.1/oceanref.pdf`

Printed by Tryckeriet i E-huset, Lund 2019


LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2019-698
http://www.eit.lth.se