UNIVERSITY OF CINCINNATI

Date: 10/28/2005

I, Wei Pei,

hereby submit this work as part of the requirements for the degree of:

Master of Science

in:

Computer Engineering

It is entitled:

"Fault Modeling and Detection for Drowsy SRAM Caches"

This work and its defense approved by:

Chair: Wen-Ben Jone

Ranga R. Vemuri

Karen A. Tomko

Fault Modeling and Detection for Drowsy SRAM Caches

A thesis submitted to the

Division of Research and Advanced Studies
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the Department of
Electrical and Computer Engineering and Computer Science
of the College of Engineering
Oct. 28

by

Wei Pei
B.S. (Computer Engineering), University of Science and
Technology of China, July 2003

Thesis Advisor and Committee Chair: Dr. Wen-ben Jone

Abstract

As CMOS transistor feature size shrinks, sub-threshold leakage power dissipation begins to dominate the total power consumption of a chip. A drowsy technique was introduced to reduce sub-threshold leakage power significantly. However, with the introduction of the drowsy cache design technique, new fault behaviors appear and more restrictive design rules must be concerned.

In this research, we implement a drowsy SRAM cache with peripheral circuits in layout level and simulate all possible spot defects (SDs) under normal mode and drowsy mode in different resistance regions. Six new fault models appear with the introduction of drowsy mode for memory arrays. We develop a march algorithm which can detect all SDs in either data caches or instruction caches. A built-in self-repair (BISR) scheme is developed. By utilizing BISR, the cache can still work even if some cache lines fail to work in drowsy mode.

To my dearest parents

Acknowledgement

I wish to express my sincere thanks to my advisor, Dr. Wen-ben Jone, for the guidance and constructive criticism that he provided throughout the whole work. He was always ready to provide guidance, and to encourage me to do better work. Thank you Dr. Jone, for all the great help and sincere concern.

I would like to thank the members of my committee, Dr. Ranga R. Vemuri and Dr. Karen A. Tomko, for spending their valuable time reviewing this work. Special thanks to my close friends in our lab, Fei, Yuan, Ming, Gaurav, and Sriram. Last, but not the least, I would like to express my gratitude to my parents and my brother, for their constant support and love.

Thank all my friends.

Contents

1 Introduction **1**

2 Background **6**

2.1 Sub-threshold Leakage 6

2.2 Drowsy Technique 8

2.2.1 Dynamic Voltage Scaling (DVS) 8

2.2.2 Drowsy Data Caches 10

2.2.3 Drowsy Instruction Caches 12

2.3 Memory Testing 13

2.3.1 Spot Defects 13

2.3.2 Definition and Location of Open Faults 15

2.3.3 Definition and Location of Short Faults 15

2.3.4 Definition and Location of Bridges 16

2.3.5 Faults Notation 17

3 Drowsy Cache Design and Simulation **19**

3.1 Drowsy Cache Circuit Implementation 20

3.2 Data Retention Voltage 25

3.3 Drowsy State 27
3.4 High-level Simulation Code 28

4 Fault Modeling **31**
4.1 Modeling Strategy 31
4.2 FFM1 Fault Class 33
4.3 FFM2 Fault Class 41

5 March Algorithm and Built-in Self Repair **54**
5.1 March DWOM 54
5.2 Fault Model Simplification 56
5.3 Fault Coverage of March DWOM 59
5.4 Buit-in Self Repair 61

6 Conclusions and Future Work **63**

A C++ code **69**

List of Figures

1.1 Leakage power increasing v.s. dynamic power increasing 2

2.1 Leakage currents in a NMOS transistor. 8
2.2 Sub-threshold leakage current in an SRAM cell. 8
2.3 Drowsy SRAM control and leakage power reduction 10
2.4 Drowsy cache line for data cache 11
2.5 Possible defect positions within an SRAM cell 14
2.6 Four-cell configuration 17

3.1 Drowsy SRAM cache architecture 21
3.2 Write and pre-charge control 22
3.3 Sense amplifier. 23
3.4 Address decoder. 23
3.5 SRAM layout and possible bridge faults 25
3.6 Deterioration of inverter VTC under low-Vdd 26
3.7 Drowsy procedure in an SRAM cell 29
3.8 Drowsy states of an SRAM cell 29

4.1 Fault modeling architecture 32

4.2 Fault behavior of BC1 in normal mode and drowsy mode 34
4.3 Simulation result of BC1 in drowsy mode 35
4.4 Drowsy data retention fault at BC2 37
4.5 Stuck-at fault at cBCC3 38
4.6 Bridge fault between two adjacent cells 45
4.7 Coupling drowsy transition fault 45
4.8 Deceptive read destructive coupling fault at BC2 47

5.1 Voltage window detector circuit 56
5.2 BISR solution of drowsy cache 62

List of Tables

2.1 List of opens 15

2.2 List of shorts 16

2.3 List of bridges within a cell 16

2.4 Bridges between adjacent cells 17

3.1 A high-level code for HSpice simulation 30

4.1 Bridge defects in a cell 42

4.2 Open defects in a cell 43

4.3 Short defects in a cell 44

4.4 Bridge defects between cells in the same row 51

4.5 Bridge defects between cells in the same column 52

4.6 Bridge defects between cells in the same diagonal 53

5.1 March DWOM 57

Chapter 1

Introduction

In the past, dynamic power dominated the total power consumption of CMOS transistors. When CMOS transistors are not switching, they are in OFF state and leakage power is negligible. However, as feature sizes shrink, leakage power increases much faster than dynamic power does. As shown in Fig. 1.1, in current $0.13\mu m$ - $0.09\mu m$ technologies, leakage power is already considerable when compared with the active power dissipation. When technology moves below $0.09\mu m$, leakage power consumption is approaching over 50% of the total power, which is not practical. Suppressing leakage current is hence critical.

On-chip memories, especially large cache memories, provided high performance with very low power-density than logic circuits before. As a result, larger and larger portion of the die area has been occupied by cache memories. For instance, 50% of Pentium®4 chip area and 60% of StrongARM chip area are allocated to the cache structure [4, 21]. On-chip memories had lower power-density, because typically only a small portion of the memories are needed to be accessed

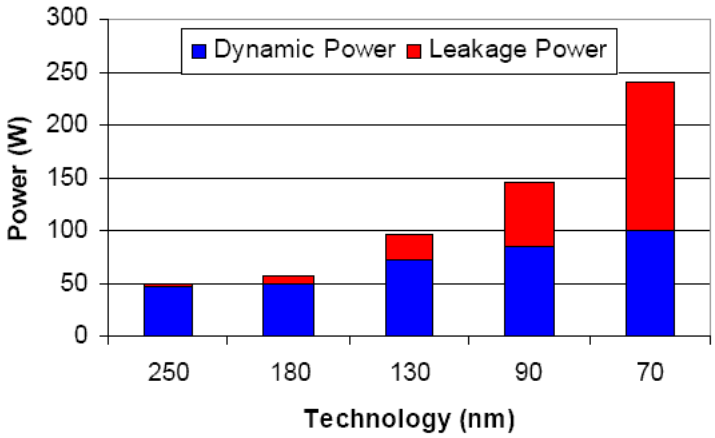

Figure 1.1: Leakage power increasing v.s. dynamic power increasing [3].

every clock cycle. It is no longer true when leakage has become a problem for transistors. Due to a large number of storage cells and lack of stacking effect [22] to reduce leakage current, leakage power will dominate the cache power consumption and thus the total power of a chip. According to the projection in [1], for 70-nm process, more than 60% of power can be consumed in L1 caches if left unchecked. Reducing leakage power of on-chip caches can decrease the total power consumption of a chip significantly.

Several techniques have been presented on leakage reduction. In [6], a dual-V_t technique uses transistors with high threshold voltage in non-critical part of memory cells, since sub-threshold leakage current reduces exponentially with the increase of V_t. But, high-V_t transistors have lower switching speed, and hence it is not suitable for caches. The gated-V_{dd} technique inserts a high-V_t transistor between the circuit and one of the power supply trails (V_{dd}/GND) [7]. The circuit will be detached from its power supply when it does not tend to be used, and the state of the circuit is lost. Thus, this technique is not appropriate for caches either.

A multi-threshold CMOS (MTCMOS) technique has also been presented to lower the threshold voltage and to reduce the leakage power [13]. However, MTCMOS will also detach SRAM cells from power supply trails in *stand-by* mode, and hence it cannot preserve the SRAM state. A simple but effective drowsy technique is proposed in [1]. This method implements caches with drowsy/standby mode and normal mode, where different supply voltages can be selected. The SRAM cells consume significantly less leakage power when placed into drowsy mode by supplying lower voltage. Due to the *spatial locality* and *temporal locality* of on-chip caches, a large portion of cache lines can be placed into drowsy mode to cut down power consumption.

Many faults in memory circuits are caused by spots of extra, missing or undesired material in a small area. These defects are called spot-defects (SDs) and are the primary testing target. In [2], a complete analysis of spot defects for industrial SRAMs is presented. *Functional fault models (FFMs)* are defined to describe the fault behaviors, and march tests are developed based on these FFMs. All electrical faults are transformed into functional fault models, which consist of nine single-cell faults (e.g., stuck-at fault) and five coupling faults (e.g., deceptive read destructive fault). A March SRD algorithm with test length 14n is developed to detect all FFMs with deterministic data outputs at sense amplifiers [2]. Recently, a similar defect injection and circuit simulation technique has also been used to derive the fault behaviors of embedded DRAMs [8]. Built-in self test (BIST) is a technique that integrated circuits can perform testing without an automatic test equipment (ATE) [9]. BIST methods based on patterns generated by march tests are dominant for testing memories nowadays [10]. The work in [11] has found

that the symmetrical structure of a march test will make it easier to implement the corresponding BIST technique. As a result, many march algorithms (e.g., March SRD in [2]) have been developed as symmetrical structures. As the complexity and the size of embedded caches/memories increase, built-in self repair (BISR) is used to improve the overall yield. BISR begins with applying memory test patterns and collecting the test response. Traditionally, the defective addresses are eliminated and substitued with redundant memory circuits [12], so the memory yield can be dramatically increased.

Unfortunately, new fault behaviors can appear with the introduction of drowsy mode caches or memories. In this research, we implement a drowsy SRAM cache with peripheral circuits like sense amplifier, address decoder, write circuit, etc. All possible spot-defects are simulated in both normal mode and standby/drowsy mode using HSpice. We find new fault behaviors in standby mode. These fault behaviors are transformed into functional fault models and a march algorithm is developed. We demonstrate that all drowsy faults can be detected by our proposed march algorithm. In this work, our march algorithm is divided into two parts, and each of them has a symmetrical structure. Further, if a cell functions properly in normal mode but manifests its defect in drowsy mode, the entire cache line that this cell locates will be marked as a non-drowsy cell (using a register), and will not be subject to drowsy operation. Thus, no redundant memory cells are required for the BISR circuit, and drowsy defects can be tolerated if the power budget is not exceeded.

The thesis is organized as follows:

Chapter 2 reviews the background of leakage currents, drowsy technique, mem-

ory testing terms and notations, and spot defects definitions.

Chapter 3 shows the design and implementation of a drowsy SRAM cache. Detailed analysis of drowsy state and minimum standby voltage are presented thereafter.

Chapter 4 performs simulation of all possible spot defects (SDs) in both normal mode and standby mode. Then, fault behaviors are transformed into functional fault models (FFMs).

Chapter 5 derives a march algorithm to detect the drowsy SRAM cache, and a built-in self-repair circuit is suggested to tolerate drowsy defects.

Chapter 6 concludes this thesis and discusses future work.

Chapter 2

Background

This chapter provides a brief introduction for the drowsy memory technique and memory testing. First, leakage currents existing in SRAM cells are presented, and sub-threshold leakage is identified as the dominant part. A simple voltage scaling method (drowsy technique) is then introduced, and memory fault definition and fault model notation are presented finally.

2.1 Sub-threshold Leakage

According to different physical mechanisms, leakage currents are categorized as follows:

- Sub-threshold leakage I_{sub}. When gate-to-source voltage V_{GS} is smaller than threshold voltage V_{TH}, I_{sub} exists from drain to source. I_{sub} increases exponentially with respect to threshold voltage reduction [14] and temperature increase [15].

- Gate direct-tunneling leakage I_{EDT}. According to quatum mechanism, charged carriers can pass through the gate oxide potential barrier into the gate [15], and this causes gate direct-tunneling leakage current. I_{EDT} will be a major issue for nanometric electronics.

- Gate-induced drain-leakage I_{GIDL} flows from drain-gate overlap to substrate of a transistor. This leakage current arises in the high electric field under gate/drain overlap region causing deep depletion. Both I_{EDT} and I_{GIDL} increases exponentially with the reduced gate oxide thickness [18, 19].

- Reverse-biased pn junction leakage I_{RBJL}. It consists of two components: one is the minority-carrier drift near the edge of the depletion region, and the other is due to electron-hole pair generation in the depletion region of the reversed junction. For present technology, leakage current induced by reverse-biased pn junction leakage is lower than I_{sub}, and thus can be neglected [15].

All these leakage currents are shown in Fig. 2.1. For submicron technologies below $0.5\mu m$, sub-threshold leakage is the dominant component of leakage power [4, 15], and can be modeled as [23, 24]:

$$I_{sub} = I_{s0} \exp^{V_{GS}-V_{TH}/(nkT/q)}(1 - \exp^{-V_{DS}/(nkT/q)})(1 + \lambda V_{DS}) \qquad (2.1)$$

where λ is a parameter modeling the pseudo-saturation region in the weak inversion region, I_{s0} is the process-specific current of a transistor when $V_{GS} = V_{TH}$, T is chip temperature, and n is process dependent, typically 1.4-1.5 [30].

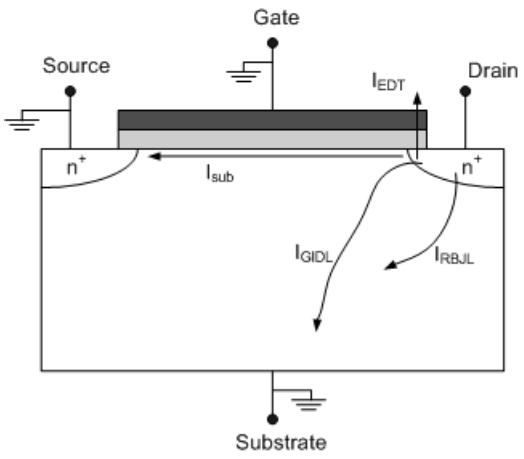

Figure 2.1: Leakage currents in a NMOS transistor.

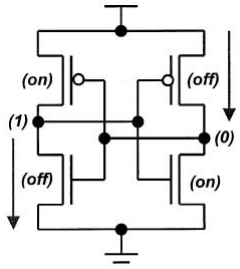

Figure 2.2: Sub-threshold leakage current in an SRAM cell.

2.2 Drowsy Technique

This section first presents the drowsy technique of a single SRAM cell. Then, a drowsy control architecture for data cache and instruction cache are introduced separately [1].

2.2.1 Dynamic Voltage Scaling (DVS)

Fig. 2.2 shows the relationship between supply voltage and leakage current in a 6T SRAM cell. The two pass transistors are not included, since they are turned off

when the cell is not accessed. As shown in Fig. 2.2, there are two off-state leakage current paths in a stand SRAM cell. The ON transistors are in strong inversion and have negligible resistance. Derived from Equation 2.1, the overall leakage of the SRAM cell can be modeled as [1]:

$$I_L = ((I_{SN} + I_{SP}) + (I_{SN}\lambda_N + I_{SP}\lambda_P)V_{DD})(1 - \exp^{-V_{DD}/(nkT/q)}) \qquad (2.2)$$

where I_{SN} and I_{SP} are nMOS and pMOS off-transistor current factors, which are independent of V_{DS} in Equation 2.1. Since the leakage current reduces *super linearly* with V_{DD}, the dynamic voltage scaling (DVS) technique is used in [1] to reduce leakage power significantly.

Fig. 2.3(a) illustrates the simple drowsy technique with a supply voltage control mechanism. By selecting $LowVolt/\overline{LowVolt}$, the SRAM cell can be placed into two modes: active/normal mode and standby/drowsy mode. In normal mode, the SRAM cell is supplied with standard voltage V_{DD} (1V, under the 70nm-technology [1]). In drowsy mode, the SRAM cell is only supplied with $V_{DD}Low$ (0.3V), while the logic value of the cell can still be retained. When a cell is not to be accessed for a period of time, it can be placed into drowsy mode and its leakage power can be reduced significantly. However, in drowsy mode, the cell is not allowed to be accessed, because the precharged bitline voltage (V_{DD}) is higher than the storage cell core voltage, which can destroy the state of the cell. In addition, during the read operation, the sense amplifier may not operate properly at the low storage cell voltage [1]. When the cell is to be accessed, $LowVolt$ is set to '0' and

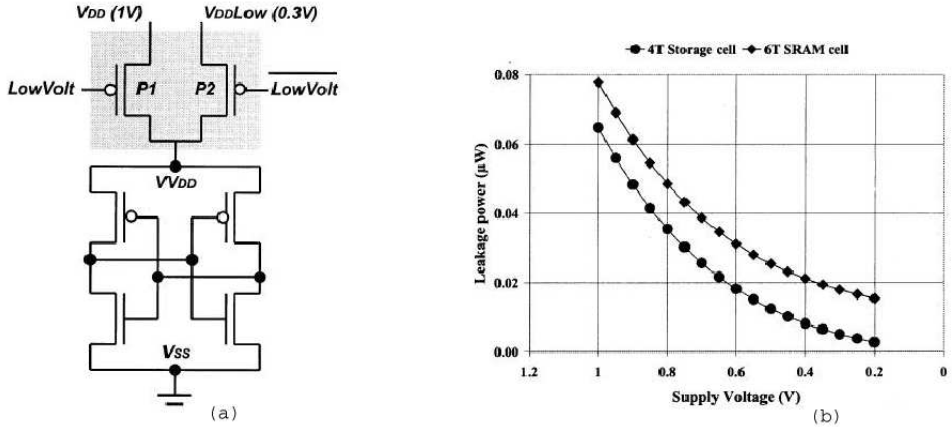

Figure 2.3: Drowsy SRAM control and leakage power reduction [1].

the cell is charged back by the standard supply voltage for read/write operations.

As shown in Fig. 2.3(b), the leakage power of 6T and 4T SRAM cells reduce significantly as we scale the supply voltage down. According to the result in [1], the leakage power of the 4T and 6T SRAM cells can be reduced by 92% and 77% respectively at 300mV standby voltage. However, the standby voltage cannot be reduced unlimitedly, and the reason will be presented in Chapter 3.2.

Since data caches tend to have better *temporal* locality while instruction caches tend to have better *spatial* locality, by using proper cache management policies, the drowsy cache technique can reduce the total leakage power significantly with trivial increase in runtime [1]. The drowsy control architecture and corresponding cache management policies are introduced in the following two sub-sections.

2.2.2 Drowsy Data Caches

Fig. 2.4 shows an implementation of drowsy cache line design for a data cache. The drowsy bit is used to control the supply voltage of cache lines. When *drowsy*

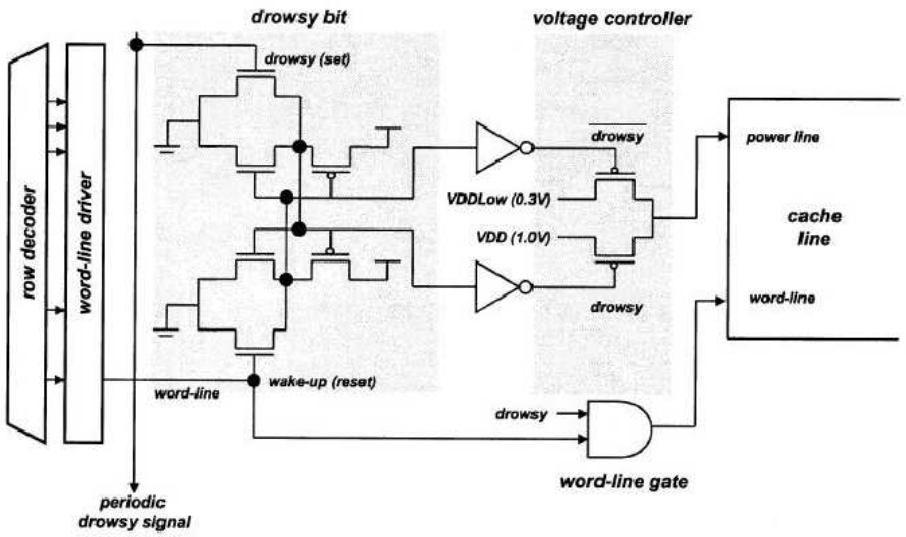

Figure 2.4: An implementation of drowsy cache line for data cache [1].

is set to be logic '1', the whole cache line is placed into drowsy state. The *word-line gate* is used to prevent access of the cache line when it is in drowsy mode. Due to the *temporal locality* of data caches, when a location is accessed, it is very likely that it will be accessed again soon [16]. Hence, a simple policy is that all the cache lines are placed into drowsy mode periodically, and a line is woke up only when it is accessed. In [1], a 2000-cycle update window is used to place all cache lines into drowsy mode every 2000 cycles. The impact of increased wake-up latency is negligible. By putting an average of 90% cache-lines into drowsy mode, roughly 85% of leakage power can be saved and runtime only increases by 0.64%.

2.2.3 Drowsy Instruction Caches

The simple policy works well for data caches but it is not effective for instruction caches, due to the *spatial locality* of instruction caches. It is found that for drowsy instruction caches, the worst case runtime increase is 10.3% and the average is 2.4% if it uses the simple policy of data caches [1]. This is much worse than the simple policy for drowsy data caches, which causes 1.2% worst case increase and only 0.6% average.

A subbank-based drowsy technique is adopted in [1] for instruction caches based on the work of [17]. The cache is divided into several subbanks, and only a limited number of subbanks are checked for their contents during each access. By using additional decoder logic to index the subbanks as shown in [17], the access latency increases slightly. In [1], a 16KB direct-mapped instruction cache is divided into four 4-kB subbanks, and only one subbank needs to be activated on each access. Each subbank consists 128 cache-lines, and all the lines together are controlled by a single drowsy bit. By setting the drowsy bit, the whole subbank can be placed into drowsy mode. In [1], several subbank prediction techniques (e.g., Next Subbank Prediction Buffer) are used to wake up a subbank before an instruction really use it. As a result, the drowsy instruction cache can save the total leakage power dissipation by more than 77% with trivial runtime increase (0.79% in average).

2.3 Memory Testing

2.3.1 Spot Defects

Defects in SRAM memory chips can be categorized as *global defects* and *local defects* [32]. *Global defects* affect a large part of the silicon; *Local defects* affect only a small (local) area of an IC, and are called spot defects (SDs). SDs can be modeled as spots of extra, missing or undesired material (resistance), and can cause undesired connections or disconnections in circuits. SDs can be introduced during any one of the many steps in the IC fabrication process, and are the primary test target since they are much harder to be detected than global defects [26]. In this study, only SDs will be considered. Depending on their conductivities in memory chips, they can be categorized to the following three groups [2]:

- *Open*: an extra resistance (R_{op}) within a connection, where $0 < R_{op} \leq \infty$.
- *Short*: an undesired resistive path (R_{sh}) between a node and V_{dd}/GND, where $0 < R_{sh} \leq \infty$.
- *Bridge*: an undesired resistive path (R_{br}) between two connections which are not V_{dd}/GND, where $0 < R_{br} \leq \infty$.

There will be more than 22 defects when considering defect locations between cells. But, due to the symmetric structure of the 6T SRAM cell, it has been demonstrated in [2] that only a subset of these defects needs to be simulated by introducing the following notations. Fig. 2.6 shows the arrangement of four adjacent cells.

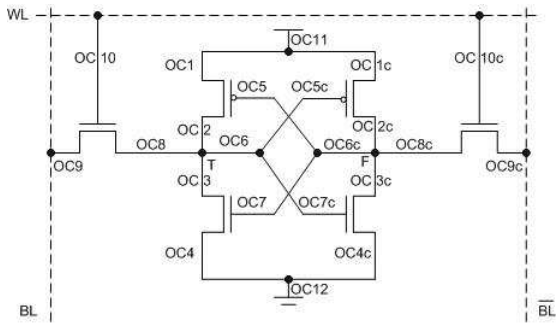

Figure 2.5: Possible defect positions within an SRAM cell [2].

- *Complementary behavior*: the locations of SD1 and SD2 in a SRAM cell are symmetrical, so the fault behavior of SD1 is similar to that of SD2. The only difference is that all 1's are replaced with 0's and vice versa. For example, a possible SD at location OC6 of Fig. 2.5 (SD1) can cause the cell stuck-at '0', then at the presence of a SD at location OC6c of Fig. 2.5 (SD2), the cell will be stuck-at '1'.

- *Interchanged behavior* (two cells involved): the fault behavior of SD1 is similar to that of SD2, except that the aggressor and the victim cells are interchanged. For example, we assume that SD1 is a bridge fault between T0 and BL1 in Fig. 2.6, where cell 0 is the aggressor and cell 2 is the victim. Then a SD between BL0 and T2 will have the similar fault behavior, except that cell 0 will be the aggressor and cell 2 will be the victim.

- *Interchanged Complementary behavior*: SD2 shows a complementary and interchanged behavior of SD1. Take the SD1 which has a bridge between T0 and BL1 in Fig. 2.6 as an example again. To make it clear, we denote SD1 as (T0-BL1). Assume SD2 has a bridge between $\overline{BL0}$ and F2. SD2 has

Table 2.1: List of opens [2]

Name	Description
OC1/OC2	Source/drain of pull-up at true side broken
OC3/OC4	Drain/source of pull-down at true side broken
OC5	Gate of pull-up at true side broken
OC6	Cross coupling at true side broken
OC7	Gate of pull-down at true side broken
OC8	Pass transistor connection to T broken
OC9	Pass transistor connection to bit line broken
OC10	Gate of pass transistor at true side broken
OC11/OC12	V_{cc}/V_{ss} path of the cell broken
OB_w	The bit line BL at the write side broken
OB_r	The bit line BL at the read side broken
OW	The word line WL broken

the similar fault behavior as SD1. To derive it, we first get the interchanged fault behavior of SD1, which is (BL0-T2), then get its complementary fault behavior ($\overline{BL0}$ - F2).

2.3.2 Definition and Location of Open Faults

Opens in an SRAM cell are categorized as opens within a cell (OC), opens at bit lines (OB) and word lines (OW). As shown in Fig. 2.5, opens at location OCx and OCxc show a *complementary* behavior, so only defects at OCx need to be simulated, and the fault behavior at opens at OCxc can be derived from that of OCx. Table 2.1 gives a detailed description of these open defects. Opens at bit lines and word lines affect many cells in same column/row of the memory. Thus, only the first cell affected by opens will be studied.

2.3.3 Definition and Location of Short Faults

Short defects can be classified as shorts within a cell (SC), shorts at bit lines (SB) and shorts at word lines (SC). As shown in Table 2.2, for example, a short at F

Table 2.2: List of shorts [2]

Name	Behav.	Comp. behav.
SC1	T-V_{dd}	F-V_{dd}
SC2	T-GND	F-GND
SB1	BL-V_{dd}	$\overline{BL}$-V_{dd}
SB2	BL-GND	$\overline{BL}$-GND
SW1	WL-V_{dd}	
SW2	WL-GND	

Table 2.3: List of bridges within a cell [2]

Name	Behav.	Comp. behav.
BC1	T-F	
BC2	T-BL	F-BL
BC3	T-$\overline{BL}$	F-$\overline{BL}$
BC4	T-WL	F-WL
BC5	BL-$\overline{BL}$	
BC6	BL-WL	$\overline{BL}$-WL

will show a complementary behavior to the short at T. SBs and SCs affect many cells, and, again, the first cell affected will be concerned.

2.3.4 Definition and Location of Bridges

Assume that bridges can exist between nodes located close to each other. Thus, all bridge faults can be classified as *bridges within a cell* and *bridges between cells*.

Table 2.3 shows all possible bridge defects within a cell (denoted as BCx), while Fig. 2.6 is used to illustrate relative cell locations in a memory. Depending on different layout implementations, all possible bridges between cells are listed in Table 2.4. Here, rBCCx denotes the bridges between cells in the same row, cBCCx denotes the bridges between cells in the same column, and dBCCx denotes the bridges between cells in near diagonal cells.

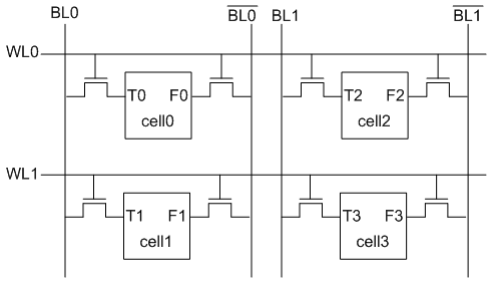

Figure 2.6: Four-cell configuration.

Table 2.4: Bridges between adjacent cells [2]

Name	Behav.	Comp. behav.	Inter. behav.	Inter. Comp. behav.
rBCC1	T0-T2	F0-F2		
rBCC2	T0-F2	F0-T2		
rBCC3	T0-BL1	F0-$\overline{BL1}$	BL0-T2	$\overline{BL0}$-F2
rBCC4	T0-$\overline{BL1}$	F0-BL1	$\overline{BL0}$-T2	BL0-F2
rBCC5	BL0-BL1	$\overline{BL0}$-$\overline{BL1}$		
rBCC6	BL0-$\overline{BL1}$		$\overline{BL0}$-BL1	
cBCC1	T0-T1	F0-F1		
cBCC2	T0-F1	F0-T1		
cBCC3	T0-WL1	F0-WL1	WL0-T1	WL0-F1
cBCC4	WL0-WL1			
dBCC1	T0-T3	F0-F3		
dBCC2	T0-F3	F0-T3		

2.3.5 Faults Notation

To describe the fault behaviors involving SRAM cells, *fault primitives* (FPs) with compact notation are introduced [2]. Each FP represents the fault behavior, and all FPs can be divided into following categories:

- $< S/F/R >$: This FP involves faults in a single cell. Here, S is the *sensitizing* operation; $S \in \{dr0, dr1, 0, 1, w0, w1, w\uparrow, w\downarrow, r0, r1, \forall\}$, where dr0 (dr1) describes the drowsy operation on the cell with logic value '0' ('1'). Further, 0/1 denotes logic value '0' and '1' separately; w0/w1/r0/r1 denotes write/read operation; $w\uparrow$ ($w\downarrow$) denotes an up (down) transition write operation. If the fault behavior of S appears after time T, the sensitizing operation is denoted as S_T. $\forall$ can be '0' or '1'. F describes the fault

behavior of the cell, $F \in \{0, 1, \uparrow, \downarrow, X\}$, where $\uparrow$ ($\downarrow$) denotes an up (down) transition; 'X' denotes an undefined logic value. R denotes the output value of an SRAM cell, if the sensitizing operation applied to the cell is *read*. We have $R \in \{0, 1, X, -\}$, where '-' means the output is not available. For example, when S is a write operation, R can be denoted as '-'. For the easiness of discussion, FPs involving in a single cell are called FP1s, and FPs involving two cells are called FP2s.

- $< S_a; S_v/F/R >$: This FP involves two cells. S_a denotes the sensitizing operation or state of the *aggressor* cell (a-cell), while S_v denotes the sensitizing operation or state of the *victim* cell. The a-cell sensitize a fault of v-cell. We have $S_a, S_v \in \{dr0, dr1, 0, 1, X, w0, w1, w\uparrow, w\downarrow, r0, r1, \forall\}$, whereby X is the *don't care* value, $X \in \{0, 1\}$. The definitions of 'F' and 'R' are the same as those of $< S/F/R >$ above.

- wF (weak fault): A fault is partially sensitized by a read/write operation [2]; e.g., if a defect can only cause a small disturbance within the noise margin, it can not be detected by an operation. In other words, in the presence of a wF, all operations pass correctly [2].

Chapter 3

Drowsy Cache Design and Simulation

This chapter first presents the detail of circuit implementation for the drowsy cache design. With the introduction of drowsy mode, two problems arise: one is how small the standby voltage can be; the other is how long (i.e., circuit delay) it needs to simulate for the drowsy state. The minimum standby voltage and the minimum simulation time for drowsy mode are then derived. Since the simulation is conducted using HSpice and it is complex to modify the HSpice file directly, an instruction-level model is established and a C++ program is implemented to convert the high-level code to the HSpice file. As a result, we need only to deal with the high-level code to perform testing algorithms, instead of digging into the details of HSPICE files.

3.1 Drowsy Cache Circuit Implementation

In this research, cache line is adopted as the major component of the cache architecture, and multiple bytes in a cache line are accessed simultaneously. The cache structure is implemented based on 6T SRAM cells, and includes the corresponding peripheral circuits like sense amplifier, address decoder, pre-charge circuit, etc. Magic is used to generate the layout with the TSMC 0.18 μm technology, which is the most up-to-date technology available in our department; HSpice is used for simulation. The diagram of the drowsy cache is shown in Fig. 3.1, and each component of the cache is introduced in the following discussion. For current 0.18um technology, Vdd is 1.8v. VddLow is derived from Chapter 3.2. The details of *write circuit*, *pre-charge circuit* and *cell* of Fig. 3.1 are shown in Fig. 3.2.

SRAM cell: Fig. 3.2 shows the design diagram of the drowsy cache. Each cell is a typical 6T SRAM. The only special feature of the SRAM cells is that they can be supplied by two different voltages. In Fig. 3.1, *M* is a mux-like module where its output can be Vdd or VddLow depending on the selecting bit. Its output (e.g., V_{dd0}) provides the power supply of SRAM cells (e.g., V_d in Fig. 3.2). For instance, cells in first row (row0) can be placed into drowsy mode by setting *drowsy0* to '1'. At this time, all these cells are supplied with VddLow (0.36V) and cannot be accessed; In normal mode, these cells are supplied with the standard voltage (1.8V) and hence can be accessed.

Write and pre-charge circuit: In Fig. 3.2, the circuit labeled as *A* is for write-control. The write-enable signal (WE) is set to '1' to connect the write circuit to bit lines. The one labeled as *B* is the pre-charge circuit. The pre-charge clock is set

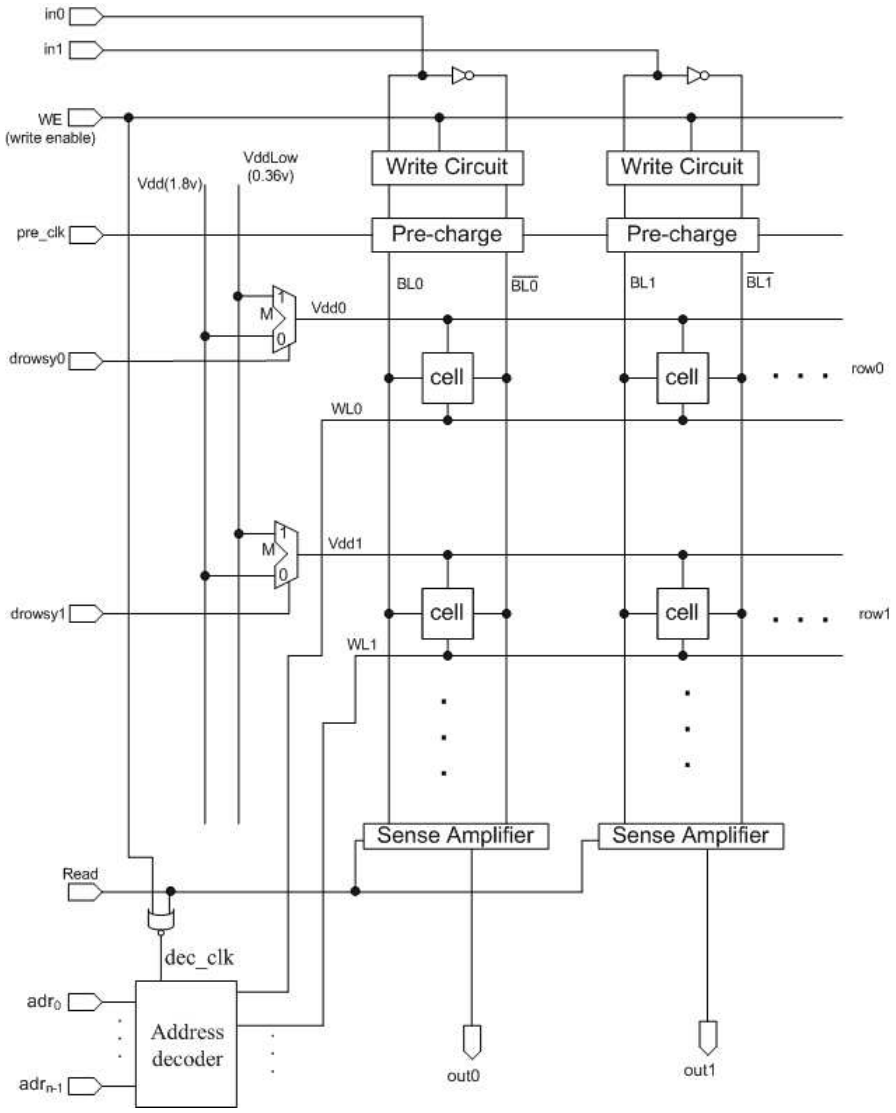

Figure 3.1: Drowsy SRAM cache architecture.

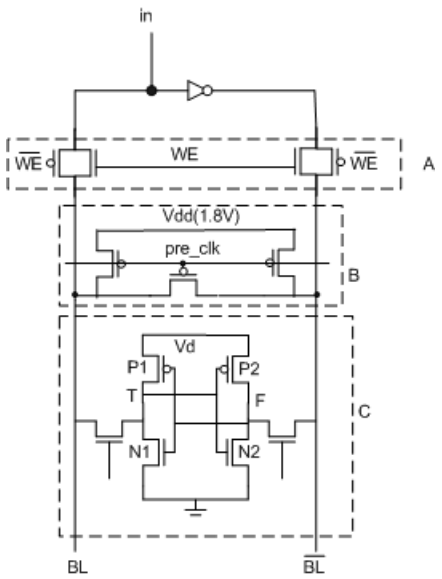

Figure 3.2: Write and pre-charge control.

to '0' when cells are not accessed, and hence each bit line is charged to standard voltage (1.8V). The P-transistor on the horizontal direction in B is for equalizing bitlines, which is needed by sense amplifier. For example, to write a value into cell *C*, the write-enable signal (WE) is set to '1' and pre-charging is disabled by setting pre-charge clock (pre_clk) to '1'. It is similar to read a cell, except that WE is also '0' and sense amplifier is enabled. When the cells in a column are not accessed, pre_clk is '0' and both bitlines are charged to V_{dd} (1.8V). Since cells are isolated by disabling the corresponding cache line, it is possible that one cache line is accessible when other cache lines are in drowsy mode, e.g., row1 can be accessed if it is in normal mode and row0 is in drowsy mode.

Sense amplifier: A double-ended current-mirror amplifier shown in Fig. 3.3 is used in this work as the differential sense amplifier. BL and $\overline{BL}$ are connected to the corresponding bit lines. A read operation will set signal $SAen$ ($\overline{SAen}$) to

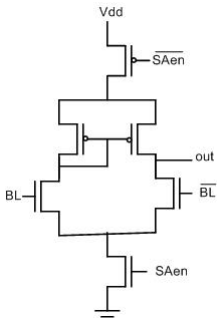

Figure 3.3: Sense amplifier.

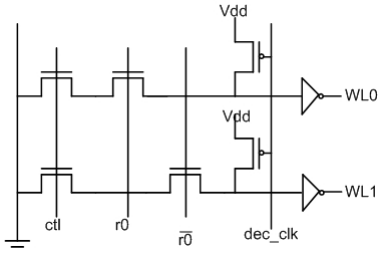

Figure 3.4: Address decoder.

'1' ('0'), and hence the corresponding SRAM cell is sensed and its value appears at *out*. Note that when there is no read operations, $SAen$ is set to '0', and sense amplifier will be disconnected with memory cells.

Address decoder: To speed up the accessing time, a dynamic NAND decoder is used. The decoder clock (dec_clk) periodically charge the address signals to Vdd. Signal *ctl* in Fig. 3.4 enables/disables the decoder block. Fig. 3.4 shows a 1-to-2 decoder design. To make word lines WL0, WL1 to logic '1' when selected, inverters are used. By setting *dec_clk* $= \overline{Read + Write}$, all the word lines are '0' when there is no Read/Write operation.

Drowsy operation control unit: Read, Write, drowsy0 and drowsy1 are the major control signals of the drowsy SRAM cell array in Fig3.1. They are generated by a finite state machine (FSM), where the FSM schedules which cache lines

are to be placed into drowsy state. The FSM also ensures the timing restriction between Read/Write signal and the drowsy signals (drowsy0, drowsy1, ...). In general, the FSM determines which cache lines are placed into drowsy mode according to a pre-defined strategy and keeps a record of them. When a cache line n is to be accessed, the FSM first checks if it is in drowsy mode. If so, the FSM first sets the drowsy control bit $drowsyn$ to '0', and performs the read/write operation right after the cache line has been waked. If the cache line is in normal mode, then the FSM directly read/write this cache line. The FSM implementation differs based on different drowsy prediction strategies mentioned in Chapter 2. However, since the FSM implementation does not affect the fault behavior of cell arrays, in this study, it is simplified by modifying the HSpice file directly to achieve the timing/logic restrictions between these control signals.

SRAM layout: Two close current paths can introduce SDs more easily. For example, given the cell layout (Fig. 3.5) used by this work, the possibility of BC2 (bridge fault between node *T* and bitline *BL*) is much higher than the possibility of BC3 (bridge fault between node *T* and bitline $\overline{BL}$). As a result, different layout implementation causes different probability distribution of spot defects (SDs). The probability distribution can be derived by the Inductive Fault Analysis (IFA) technique [31]. To get a 'general' testing algorithm for arbitrary SRAM layout, all possible SDs (BC1-BC6, OC1-OC12, etc) are considered in this research. Typically, the read/write time of the cache is set to 5ns in this work.

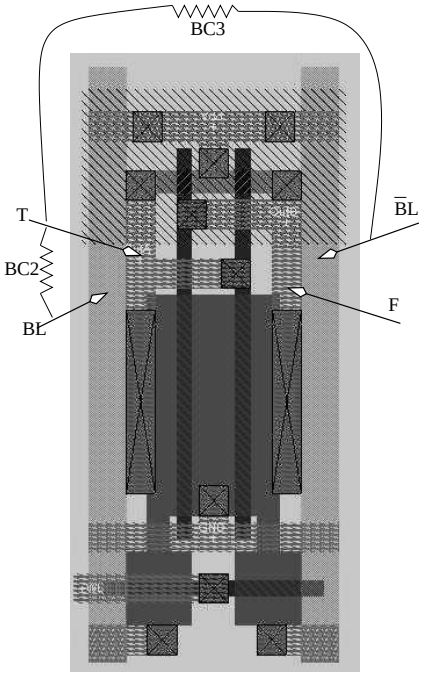

Figure 3.5: SRAM layout and possible bridge faults.

3.2 Data Retention Voltage

Since the leakage power reduces super-linearly with the reduced standby voltage (Equation 2.2), the minimum standby voltage (Date Retention Voltage, DRV) hence can achieve the minimum leakage power while preserving the data stored in an SRAM cell. This section exploits the limit of SRAM low voltage data preservation.

The cell stability is often characterized using *static noise margin* (SNM) where noises like mismatches and disturbances are modeled as DC offsets [27, 28, 29]. When these DC offsets exceeds the SNM of an SRAM cell, the cell is caused a false switch. SNM can be visualized by superimposing the voltage transfer curves (VTC) of both cross-coupled inverters within an SRAM cell. Its value is

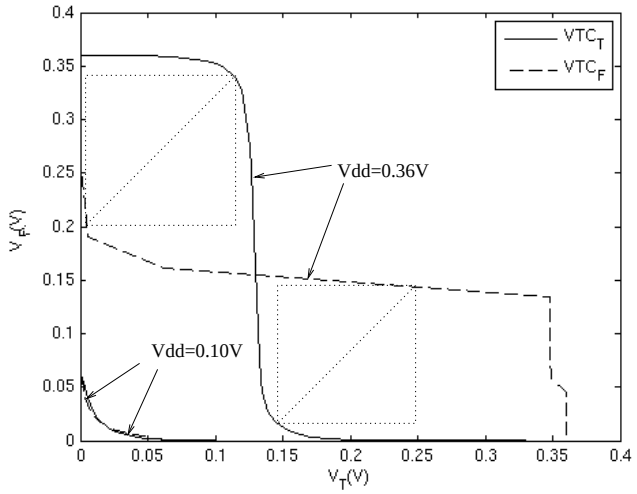

Figure 3.6: Deterioration of inverter VTC under low-Vdd.

defined as the edge of the maximum square that can fill into the two VTC curves [29]. In Fig. 3.6, V_T and V_F denote the voltages of nodes T and F in the SRAM cell mentioned in Fig. 3.1. VTC_T denotes the VTC resulting from the inverter whose input is T and output is F, while VTC_F denotes the VTC resulting from the inverter whose input is F and output is T. When Vdd is 0.36v, the resulting SNM is around 100mv. When V_{dd} reduces to 0.1v, the voltage transfer curves (VTC) degrades such that the noise margin degrade to 0. If V_{dd} reduces further, the SRAM cell can not retain the stored data any more. But, the real noise margin comes not only with reduced V_{dd}, but with temperature, process variation, etc. So, the standby voltage cannot be reduced all the way down to 0.1v. In [25], it is found that a guard band over 100mv of the minimum voltage (the one with zero SNM) is sufficient to overcome these noise effects. In this work, 0.36v is used as the DRV for our $0.18\mu m$ technology.

3.3 Drowsy State

To illustrate how the drowsy cache works, the SRAM cell in Fig. 3.7 is used as an example. The absolute value of both P-transistor and N-transistor threshold voltages is 0.53v, which is denoted in the technology file (TSMC SCN6M_SUBM). It is assumed that the cell contains logic '1' before being placed into drowsy state. Hence, V_T (V_F) equals to 1.8v (0v) in the beginning. The memory cell goes through three phases to enter into drowsy state, as illustrated in Fig. 3.7 and Fig. 3.8. Fig. 3.7 consists of four procedures: (a) shows the initial status of the cell, at this time, Vdd (1.8v) is supplied; (b) when Vdd is reduced to 0.53v, the voltage of T reduces all the way down to 0.53v immediately; (c) when Vdd is reduced to 0.36v, the voltage of T reduces very slowly because of the leakage current; (d) the voltage of T reduces to 0.36v, the cell enters into a stable state. In Fig. 3.8, V_T (V_F) denotes the voltage of node T (F) in an SRAM cell (Fig. 3.1), Vdd denotes that supply voltage to the cell. The regions denoted by number (1,2,3) shows different phases when cell is placed in drowsy state. Phases 1 and 2 are divided based on the Vdd value, which is denoted as point (2.18e-08,5.53e-01). Note that this point represents 21.8ns and 0.553 volt.

In *phase one* (region 1 in Fig. 3.8), the supply voltage V_{dd} is reduced but is still above the absolute value of threshold voltage (V_{TH}, 0.53v) of P1. During this period, the gate-to-source voltage V_{GS} of P1 (0-V_{dd}) is less than V_{TH} (-0.53v), transistor P1 is ON (P2, N1 are OFF, and N2 is ON), and V_T reduces immediately with V_{dd} (Fig. 3.7(b)). This phase can also be presented as (a)$\rightarrow$(b) in Fig. 3.7.

In *phase two* (region 2 in Fig. 3.8), we have $V_{dd} < |V_{TH}|$, hence $V_{GS} > V_{TH}$, and transistor P1 is OFF (P2, N1, and N2 are OFF). At this time, all the 6

transistors are in sub-threshold region. A leakage current exists from node T to node V_{dd}, which is shown in Fig. 3.7(c). During this time, V_T reduces slowly when compared to the change in V_{dd}. This phase can be denoted as (b)→(c) in Fig. 3.7.

In *phase three*, V_T equals to V_{dd} (shown in Fig. 3.7(d)). As derived from Equation 2.1, leakage current is around 0.

It can be seen that the drowsy time (the time needed for a cell to enter into drowsy mode) and the wakeup time (the time needed to charge the cell to standard voltage) depend on the slope of V_{dd}. But the time needed for a cell to enter into 'stable' drowsy state (the voltage of node with logic '1' reduces to standby voltage) is much larger. In Fig. 3.8, the cell enters into steady drowsy state only after around 280ns. In following discussion, we define region 2 in Fig. 3.8 as 'early' drowsy state, while region 3 as 'static' drowsy state. Fortunately, to detect all the faults in a drowsy cell, the cell only needs to enter into 'early' drowsy state which is only several nano-seconds. This will be presented in following chapters.

3.4 High-level Simulation Code

Since all possible SDs (39 SDs in this work) in different resistance ranges have to be simulated, a C++ program *tbt.cpp* is implemented to convert the high-level simulation code to all corresponding HSpice files. Herein, the high-level code consists of six operations: W(rite), R(ead), I(dle), D(rowsy), C(harge) and E(nd). The operation `W adr i0 i1 i2` ... writes $i0, i1, i2, \ldots$ to cache line adr; `R adr` reads the contents of cache line adr, and outputs can be observed from the

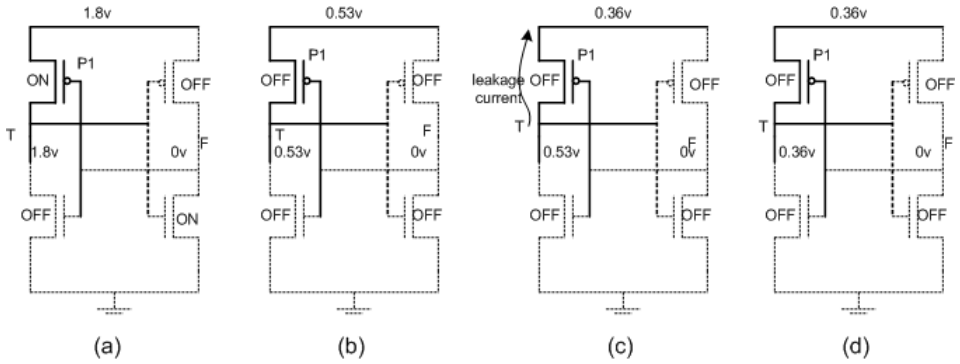

Figure 3.7: Drowsy procedure in an SRAM cell.

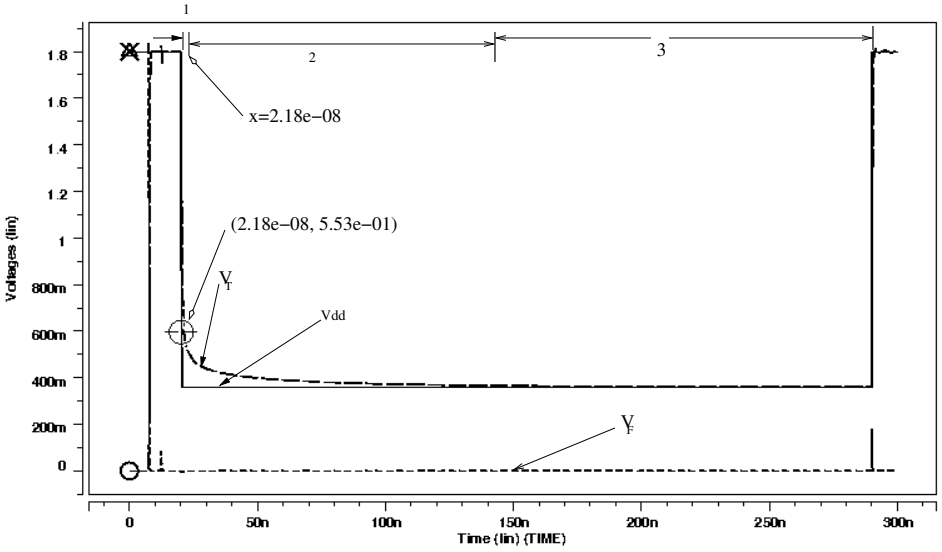

Figure 3.8: Drowsy state of an SRAM cell.

Table 3.1: A high-level code for HSpice simulation

W 0 0 1	(1)
W 1 1 1	(2)
R 1	(3)
D 0 -1	(4)
W 1 1 0	(5)
R 1	(6)
C 0 -1	(7)
I	(8)
E	(9)

simulation result; `I` stands for idle, and the whole cache is not accessed during this time; `D adr0 adr1 ... -1` places cache line $adr0$, $adr1$, ... to drowsy mode, and delimiter -1 denotes the end of this operation; `C adr0 adr1, ..., -1` wakes up cache line $adr0$, $adr1$, ..., and -1 is also used as delimiter. Finally, `E` ends the code translation and writes all results to the HSpice file. Table 3.1 is an example based on the assumption that each cache line is two-bit wide. After `D 0 -1` operation, cache line 0 (cell 0,1) is in standby mode until it is woke up by `C 0 -1`. During the drowsy time, cache line 1 can still be accessed (operation (5),(6) in Table 3.1). The C++ code for tbt.cpp is included in Appendix A.

Chapter 4

Fault Modeling

Based on the cache implementation of Chapter 3, we simulated all possible SDs in both standard mode and drowsy mode. FFM1 and FFM2 fault models are then derived from the simulation results.

4.1 Modeling Strategy

In this work, it is assumed that SDs can only exist either within a cell or between two adjacent cells in the same row/column/diagonal. Therefore, four single SRAM cells are scheduled as a 2-by-2 array to model all possible SDs. In Fig. 4.1, these four SRAM cells are labeled from 0 to 3. Further, In0 and In1 are the two bits of inputs; d0 and d1 are the corresponding outputs. Different rows of cells can be placed into drowsy mode separately. For example, cells 0, 2 and cells 1, 3 can be placed into drowsy mode separately. However, cells in the same row (e.g., cells 0, 2) must be placed into drowsy mode simultaneously.

To simulate the *short* and *bridge* faults, we manually insert an additional re-

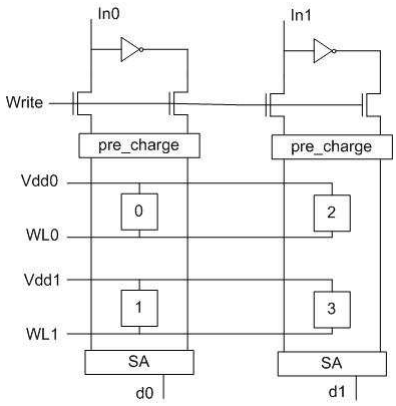

Figure 4.1: Fault modeling architecture
Fault modeling architecture.

sistance in the HSpice file between any two nodes where possible faults might occur. For instance, a BC1 fault can be simulated by a line `Rbc1 T F` *`resist_value`* in the HSpice file, where the *resist_value* ranges from 0 to ∞. However, the circuit layout has to be modified to simulate the *open* faults. Take the OC1 fault in Fig. 2.5 as an example, the connection at position OC1 in the layout needs to be broken into two sections labeled as a, b separately; the OC1 fault can then be represented by `Roc1 a b` *`resist_value`*, where $0 < resist\_value < \infty$. By changing the resistance value from 0Ω to $\infty\Omega$ gradually, all possible SDs are simulated and the corresponding fault behaviors can be observed.

As mentioned in Chapter 2, data caches and instruction caches have different drowsy control strategies and architectures. The subbanks introduced by instruction caches are relatively isolated with each other, and hence the possibility of SDs between two subbanks is much lower. All cache lines within a subbank of an instruction cache are placed into drowsy mode or normal mode all together, which is

not same as those in data caches. As a result, a smaller number of fault behaviors will be derived from instruction caches. In the following work, all fault behaviors but one can be derived from both data caches and instruction caches. As it will be mentioned in Chapter 4.3, fault CF_{wdtf} (drowsy coupling write destructive Fault) and some of CF_{dtf} and CF_{tdtf} only exist in data caches. It will be analyzed when we derive the testing algorithm for both data caches and instruction caches.

The fault primitives (FPs) introduced in Chapter 2 can be translated into *functional fault models* (FFMs). A FFM is defined as a non-empty set of fault primitives (FPs) [2]. The functional fault models (FFMs) are categorized as FFM1 and FFM2, where FFM1 consists all FP1s and FFM2 consists of all FP2s.

4.2 FFM1 Fault Class

The simulation results of FFM1 are listed in Tables 4.1, 4.2, 4.3, 4.4, and 4.5. By default, all SDs are simulated at cell 0 in Fig. 4.1. Hence, in these tables, FP1s without sub-script show the simulation result of cell 0, while those with sub-script x ($< S/F/R >_x$) show the observed fault behavior of cell x.

In the following tables, column 'Comp.behavior' shows the *complementary* behavior of a specific fault. Column 'Name' denotes each SD fault according to its type and position. For *bridge* and *short* faults, notation (A-B) within the 'Name' column shows that a *bridge* (*short*) exists between nodes *A* and *B*; for each *open* fault, the open position can be found in Fig. 2.5. 'Resistance' column denotes the different resistance regions (in increasing order, from 0Ω to $\infty\Omega$) where different fault behaviors occur. Note that different SDs have different number of resistance

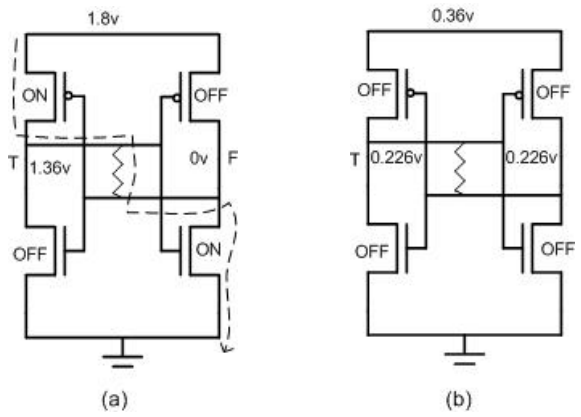

Figure 4.2: Fault behavior of BC1 in normal mode and drowsy mode.

regions. For example, $BC1$ has four resistance regions while $BC2$ has six of them. The values of different resistance regions might also be different. Take $BC1$ and $BC2$ as an example. The region II of $BC1$ is 40KΩ or above, while the region II of $BC2$ is 2kΩ or above. 'Fault behavior' shows the fault behavior of each SD, where '-' denotes there is no fault behavior for the current setting. Further, *wF* denotes that the defect can only cause a small disturbance and does not affect the cell function. Column 'FFM' shows the name of that functional fault model defined in this section. FP2s mentioned in Table 4.1 will be explained in next section.

The simulation time for drowsy operation (the time a cell being placed into drowsy mode) ranges from 5ns ('early' drowsy state in Fig. 3.8), to 300ns ('static' drowsy state in Fig. 3.8). We found that different drowsy operation time units can get the same simulation results, which means that 'early' drowsy state is enough for simulation. As a result, the simulation time for each drowsy operation can be 5ns, which will save much time for testing. To save space, the simulation results of different drowsy times will not be shown in following tables.

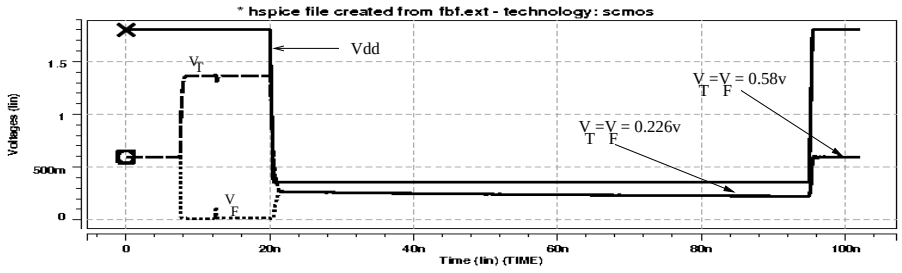

Figure 4.3: Simulation of BC1 in drowsy mode.

New fault behavior appears with the introduction of drowsy state. Take Fig. 4.2 as an example. Assume there is an extra resistance (25k) between T and F. When the cell is operated with standard voltage, current goes through from V_{dd} to T, F, GND (shown in Fig. 4.2(a)). Due to the large resistance, the voltage of T is 1.36v, and hence the cell can retain its value (logic '1'). When the cell goes into drowsy state, all six transistors are OFF, and the voltages of T and F become the same. At this time, no current path exists. The cell can no longer retain its value when waking up. This can be observed from the waveform of Fig. 4.3. As a result, when bridge defect BC1 (25k Ω) exists, the cell operates properly under the standard voltage. But, once it enters the drowsy mode, the voltages of both T and F nodes become the same (0.226v). Thus, when it is accessed after being woke up, the cell returns an undefined state (0.58v). This fault can be represented as $< dr1/X/- >$ in Table 4.1. This fault model is defined as *Drowsy Undefined Fault* (DUF) here.

Another new fault behavior introduced by the drowsy technique is *Drowsy Data Retention Fault* (DDRF), where a drowsy operation applied to a cell will change the cell value to its inversion. Take the bridge fault BC2 in Table 4.1 as an example. Assume that a resistance (bridge fault) of 40kΩ exists between nodes T

and BL0 in the cell C of Fig. 3.1. The cell C stores a '1' in the beginning state. The cell can retain its original state when it is woke up from the drowsy state. Then, the cell is written a logic '0'. When it enters into drowsy mode, because of the bridge between T and BL0, node T will be charged when BL0 is pre-charged to 1.8v. After a certain delay time (8ns as shown in Fig. 4.4), the cell value is inversed. When it is woke up, the cell contains a '1' now. This can be seen in Fig. 4.4. This fault can be modeled as $< dr0_T/1/- >$. We found that when the resistance of a bridge defect becomes larger, the corresponding simulation time for DDRF increases up to 2us. In region V of BC2 in Table 4.1, the dagger sign (†) after *DDRF* shows the drowsy operation time (i.e., test application time) needed to detect all possible *DDRF*s.

Note that FP1 can still exist even if there is a SD between two cells, as shown in Tables 4.4, 4.6. Take the stuck-at fault (SAF) in region II of cBCC3 as an example. Assume that a bridge fault (5000Ω) exists between the true side node (T0) of cell 0 and the write line (WL1) of cell 1 in Fig. 4.1. In Fig. 4.5, V_{T0}/V_{F0} and V_{T1}/V_{F1} denote the voltage of T/F node in cells 0, 1 separately. V_{WL0} and V_{WL1} are the write line (WL) signals of cells 0, 1. In *region 1*, a *write '1'* operation (input is not shown here) applies to cell 0, but cell 0 is still in logic '0' after this *write* operation. In other hand, cell 1 was written a '1' in *region 2* and then a '0' in *region 3*. As is shown by the curves of V_{T1}/V_{F1}, cell 1 works well. This fault is denoted as $< \forall/0/- >_0$, which is a FP1 fault.

Based on the fault simulation results of opens, shorts, bridges within a single cell, the following FP1s in FFM1 are derived.

- *Stuck-at fault (SAF)*: the logic value of a cell is always '0' or '1'. SAF

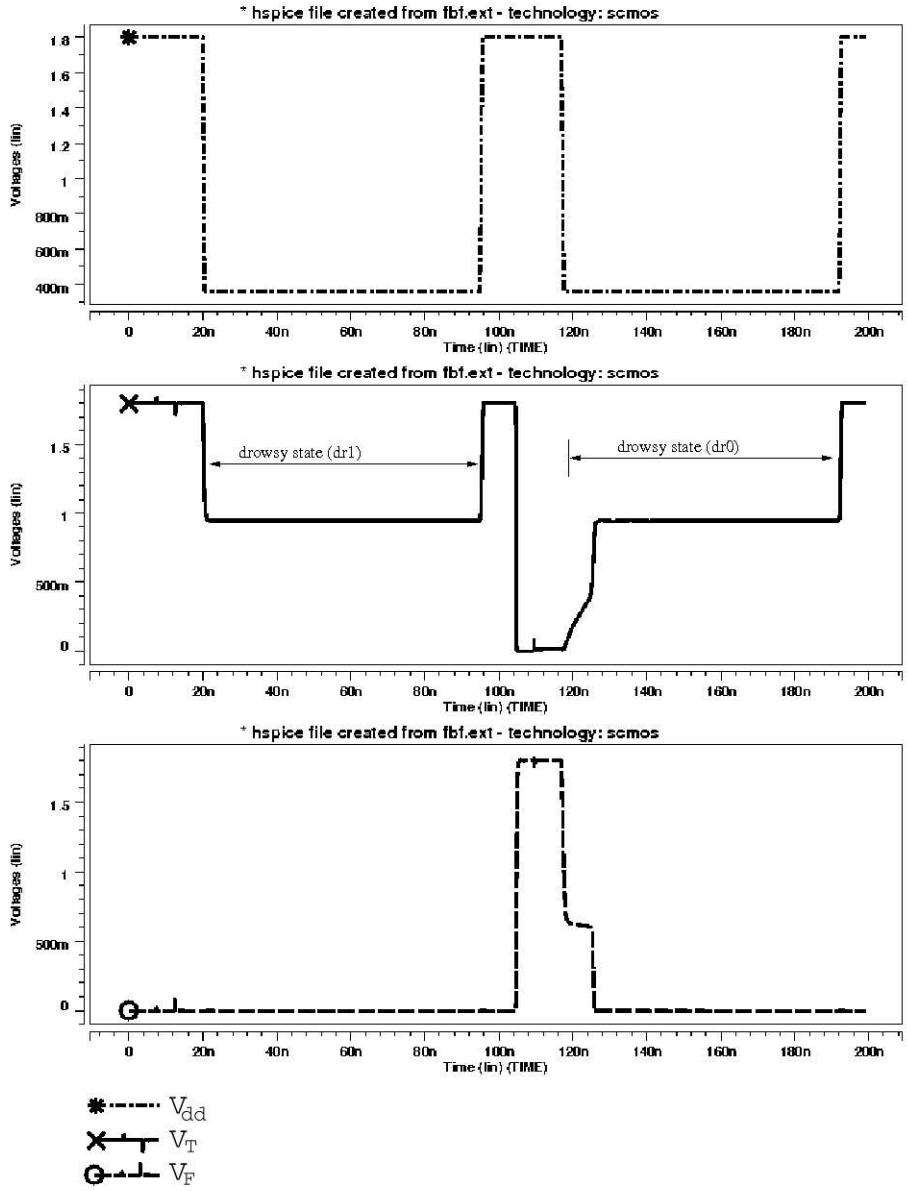

Figure 4.4: Drowsy data retention fault (DDRF) at BC2.

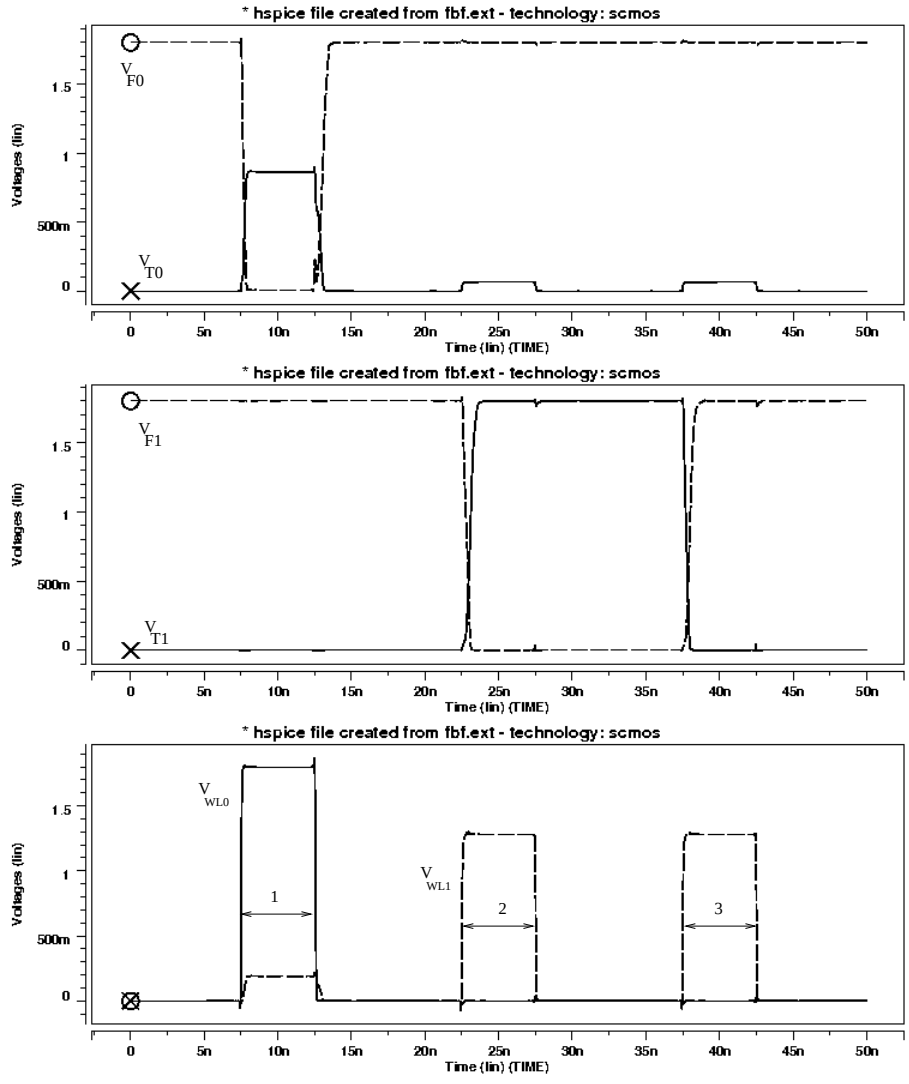

Figure 4.5: Stuck-at fault (SAF) at cBCC3 (5000Ω).

consists of two FPs: $< \forall/0/- >$ and $< \forall/1/- >$. It can be caused by: (a) opens within a cell (OC6, OC7, OBw), (b) shorts (SC1, SC2, SB1, SB2), and (c) bridges (BC4, BC6, cBCC3).

- *Stuck-open fault (SOF)*: $< w \uparrow /0/- >$, $< w \downarrow /1/- >$, $< rx/x/X >$ defines an unaccessible cell. It can be caused by: (a) bridge between a node of a cell and write line (BC4), and (b) short between write line and GND (SW2).

- *Undefined state fault (USF)*: a read/write operation performed to a cell brings the cell into an undefined state, and hence a rx operation returns a random value. The USF consists of four FPs: $< w0/X/- >$, $< w1/X/- >$, $< r/X/X >$. It can be caused by (a) bridge between T and F nodes of a cell (BC1), (b) bridge between bitlines within a cell (BC5), and (c) bridge between two adjacent write lines (cBCC4).

- *Transition fault (TF)*: the cell fails to undergo a transition ($0 \rightarrow 1$ or $1 \rightarrow 0$) when it is written: $< w \uparrow /0/- >$, $< w \downarrow /1/- >$. It can be caused by (a) pass transistor connection of the cell broken (OC8, OC9), and (b) Gate of pass transistor broken (OC10).

- *Data retention fault (DRF)*: the cell fails to retain its logic value after a period of time. DRF consists of $< 1_T/0/- >$, $< 0_T/1/- >$, $< 1_T/X/- >$, $< 0_T/X/- >$. It can be caused by (a) source/drain/gate of the pull-up transistor of a cell broken (OC1, OC2, OC5), and (b) V_{cc}/V_{ss} path of a cell broken (OC11, OC12).

- *Read destructive fault (RDF)*: a rx operation performed on a cell changes the cell value into $\overline{x}$ while returns $\overline{x}$. RDF consists of two FPs: $< r0/\uparrow/1 >$ and $< r1/\downarrow/0 >$. It can be caused by: (a) short between bitline and V_{ss} (SB2), (b) bridge between a bitline and write line within a cell (BC6), and (c) drain/source of the pull-down transistor of a cell broken (OC3, OC4).

- *Deceptive read destructive fault (DRDF)*: a rx operation performed to a cell changes the cell state to $\overline{x}$ while returns value x. DRDF consists of two FPs: $< r0/\uparrow/0 >$ and $< r1/\downarrow/1 >$. It can be caused by drain/source of the pull-down transistor of a cell broken (OC3, OC4).

- *Incorrect read fault (IRF)*: a rx operation applied to a cell returns $\overline{x}$ or an undefined value, while retains the state of the cell. IRF consists of two FPs: $< r0/0/1 >$ and $< r1/1/0 >$. Open at the read side bitline (OBr) can cause this fault.

- *Drowsy transition fault (DTF)*: a drowsy operation applied on a cell with value x changes the value to $\overline{x}$, when the cell is waken up. DTF consists of two FPs: $< dr0/1/- >$ and $< dr1/0/- >$. It can be caused by: (a) bridge between one node of a cell and bitline $BL/\overline{BL}$ or writeline WL within a cell (BC2, BC3, BC4), (b) gate of pull-up at true side broken (OC5), (c) short between node of cell and V_{ss}/V_{dd} (SC1, SC2), (d) bridge between a node of a cell and bitlines $BL/\overline{BL}$ (rBCC3, rBCC4), and (e) bridge between node of a cell and its adjacent write line (cBCC3).

- *Drowsy undefined state fault (DUF)*: a drowsy operation performed on a cell brings the cell into an undefined state ($< dr0/X/- >$, $< dr1/X/- >$). It

can be caused by bridge between the T and F nodes of a cell (BC1).

- *Drowsy data retention fault (DDRF)*: a cell fails to retain its value after a period of time under drowsy state. DDRF consists of four FPs: $< dr0_T/\uparrow /- >$, $< dr1_T/\downarrow /- >$, $< dr0_T/X/- >$ and $< dr1_T/X/- >$. It can be caused by (a) opens within a cell (OC1, OC2, OC5, OC11, OC12), (b) bridges within a cell (BC1, BC2, BC3, BC4), and (c) bridges between cells (rBCC3, rBCC4, cBCC3).

Three new fault models (DTF, DUF, DDRF) are introduced by bridging faults in drowsy state (Table 4.1). For opens within a cell, only the source/drain of pull-up open defects (OC1, OC2, OC5, OC11, OC12) introduce new fault behavior (DDRF in Table 4.2). The cell needs to be placed into drowsy mode for at least 2ms to observe the fault behavior (Table 4.2). For shorts, drowsy state does not introduce any new fault model (Table 4.3).

4.3 FFM2 Fault Class

Cells in same row/column/diagonal are simulated to derive FFM2. To save space, their *interchanged* (*complementary*) behavior will not be listed in Tables 4.4, 4.5, and 4.6. The fault notation $< Sa; Sv/F/R >_{i,j}$ indicates cells *i, j* are aggressor/victim to each other.

New coupling faults also exist with the introduction of drowsy operation. Take the rBCC2 defect (bridge between two cells in the same row, Table 4.4) shown in Fig. 4.6 as an example. The resistance between F0 and T2 is 15kΩ. Assume that

Table 4.1: Bridge defects in a cell

Name	Resistance	Fault behavior	Comp. behavior	Class	FFM
BC1 $(T-F)$	I	$<wx/X/->$ $<r/X/X>$	-	FP1	USF
	II	$<drx/X/->$	-	FP1	DUF
	III	$<drx_T/X/->$	-	FP1	$DDRF$ †
	IV	wF	-	-	-
BC2 $(T-BL)$	I	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
		$<w0;1/\downarrow/->_{1,0}$ $<r0;1/\downarrow/->_{1,0}$	$<w0;0/\uparrow/->_{1,0}$ $<r0;0/\uparrow/->_{1,0}$	FP2	CF_{ds}
		$<0;w\uparrow/0/->_{0,1}$	$<1;w\uparrow/0/->_{0,1}$	FP2	CF_{iw}
	II	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
		$<w0;1/\downarrow/->_{1,0}$ $<r0;1/\downarrow/->_{1,0}$	$<w0;0/\uparrow/->_{1,0}$ $<r0;0/\uparrow/->_{1,0}$	FP2	CF_{ds}
		$<0;r1/\downarrow/0>_{0,1}$	$<1;r0/\uparrow/1>_{0,1}$	FP2	CF_{rd}
	III	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
		$<w0;1/\downarrow/->_{1,0}$ $<r0;1/\downarrow/->_{1,0}$	$<w0;0/\uparrow/->_{1,0}$ $<r0;0/\uparrow/->_{1,0}$	FP2	CF_{ds}
		$<0;r1/\downarrow/1>_{0,1}$	$<1;r0/\uparrow/0>_{0,1}$	FP2	CF_{drd}
	IV	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
	V	$<dr0_T/1/->$	$<dr1_T/0/->$	FP1	$DDRF$ †
	VI	wF	wF	-	-
BC3 $(T-\overline{BL})$	I	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
		$<w1;1/\downarrow/->_{1,0}$ $<r1;1/\downarrow/->_{1,0}$	$<w1;0/\uparrow/->_{1,0}$ $<r1;0/\uparrow/->_{1,0}$	FP2	CF_{ds}
		$<0;w\downarrow/1/->_{0,1}$	$<1;w\downarrow/1/->_{0,1}$	FP2	CF_{iw}
	II	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
		$<w1;1/\downarrow/->_{1,0}$ $<r1;1/\downarrow/->_{1,0}$	$<w1;0/\uparrow/->_{1,0}$ $<r1;0/\uparrow/->_{1,0}$	FP2	CF_{ds}
		$<0;r0/\uparrow/1>_{0,1}$	$<1;r0/\uparrow/1>_{0,1}$	FP2	CF_{rd}
	III	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
		$<w1;1/\downarrow/->_{1,0}$ $<r1;1/\downarrow/->_{1,0}$	$<w1;0/\uparrow/->_{1,0}$ $<r1;0/\uparrow/->_{1,0}$	FP2	CF_{ds}
		$<0;r0/\uparrow/0>_{0,1}$	$<1;r0/\uparrow/0>_{0,1}$	FP2	CF_{drd}
	IV	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
	V	$<dr0_T/1/->$	$<dr1_T/0/->$	FP1	$DDRF$ †
	VI	wF	wF	-	-
BC4 $(T-WL)$	I	$<r0/0/X>$ $<w\uparrow/0/->$	$<r1/1/X>$ $<w\downarrow/1/->$	FP1	SOF
		$<r1/1/X>_2$ $<w\downarrow/1/->_2$	$<r0/0/X>_2$ $<w\uparrow/0/->_2$	FP1	SOF
	II	$<\forall/0/->$	$<\forall/1/->$	FP1	SAF
	III	$<dr1/0/->$	$<dr0/1/->$	FP1	DTF
	IV	$<dr1_T/0/->$	$<dr0_T/1/->$	FP1	$DDRF$ †
	V	wF	wF	-	-
BC5 $(BL-\overline{BL})$	I	$<rx/x/X>$	-	FP1	USF
	II	wF	-	-	-
BC6 $(BL-WL)$	I	$<\forall/1/->$	$<\forall/0/->$	FP1	SAF
		$<r1/\downarrow/0>_1$	$<r0/\uparrow/1>_1$	FP1	RDF
	II	wF	-	-	

†: T is at least 2us

Table 4.2: Open defects in a cell

Name	Resistance	Fault behavior	Comp. behavior	Class	FFM
OC1,OC2	I	wF	wF	-	-
	II	$<1_T/\downarrow/->$	$<0_T/\uparrow/->$	FP1	$DRF^{\dagger}$
		$<dr1_T/\downarrow/->$	$<dr0_T/\uparrow/->$	FP1	$DDRF^{\ddagger}$
OC3,OC4	I	wF	wF	-	-
	II	$<r0/\uparrow/0>$	$<r1/\downarrow/1>$	FP1	DRDF
	III	$<r0/\uparrow/1>$	$<r1/\downarrow/0>$	FP1	RDF
OC5	I	wF	wF	-	-
	II	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
		$<1_T/\downarrow/->$	$<0_T/\uparrow/->$	FP1	$DRF^{\dagger}$
		$<dr1_T/\downarrow/->$	$<dr0_T/\uparrow/->$	FP1	$DDRF^{\ddagger}$
OC6	I	wF	wF	-	-
	II	$<\forall/0/->$	$<\forall/1/->$	FP1	SAF
OC7	I	wF	wF	-	-
	II	$<\forall/1/->$	$<\forall/0/->$	FP1	SAF
OC8	I	wF	wF	-	-
	II	$<w\downarrow/1/->$	$<w\uparrow/0/->$	FP1	TF
OC9	I	wF	wF	-	-
	II	$<w\downarrow/1/->$	$<w\uparrow/0/->$	FP1	TF
OC10	I	wF	wF	-	-
	II	$<w\downarrow/1/->$	$<w\uparrow/0/->$	FP1	TF
OC11,OC12	I	wF	wF	-	-
	II	$<1_T/X/->$ $<0_T/X/->$	-	FP1	$DRF^{\ddagger}$
		$<dr1_T/X/->$ $<dr0_T/X/->$	-	FP1	$DDRF^{\ddagger}$
OB_w	I	wF	wF	-	-
	II	$<\forall/1/->$	$<\forall/0/->$	FP1	SAF
OB_r	I	wF	wF	-	-
	II	$<r1/1/0>$	$<r0/0/1>$	FP1	IRF
OW	I	wF	wF	-	-
	II	$<rx/x/X>$	$<rx/x/X>$	FP1	USF

†: T is at least 600us
‡: T is at least 2ms

Table 4.3: Short defects in a cell

Name	Resistance	Fault behavior	Comp. behavior	Class	FFM
SC1 ($T - V_{cc}$)	I	$< \forall/1/- >$	$< \forall/0/- >$	FP1	SAF
	II	$< dr0/1/- >$	$< dr1/0/- >$	FP1	DTF
	III	wF	wF	-	-
SC2 ($T - V_{ss}$)	I	$< \forall/0/- >$	$< \forall/1/- >$	FP1	SAF
	II	$< dr1/0/- >$	$< dr0/1/- >$	FP1	DTF
	III	wF	wF	-	-
SB1 ($BL - V_{cc}$)	I	$< \forall/1/- >$	$< \forall/0/- >$	FP1	SAF
	II	wF	wF	-	-
SB2 ($BL - V_{ss}$)	I	$< \forall/0/- >$	$< \forall/1/- >$	FP1	SAF
	II	$< r1/\downarrow/0 >$	$< r0/\uparrow/1 >$	FP1	RDF
	III	wF	wF	-	-
SW1 ($WL - V_{cc}$)	I	wF	-	-	-
SW2 ($WL - V_{ss}$)	I	$< w\downarrow/1/- >$ $< rx/x/X >$	-	FP1	SOF
	II	wF	-	-	-

originally both cells store logic '0'. When both cells enter into drowsy state, because of the resistance path between F0 and T2, the voltage of F0 will be reduced. When cell 0 is woke up, both nodes (T0, F0) are charged. But, at this time the voltage of T2 is 0 and there is a bridge between F0 and T2, so node F0 will be charged much slower. As a result, cell 2 will manifest its defect when woke up. This fault is denoted as $< dr0; dr0/1/- >_{2,0}$, and it is called *coupling drowsy transition fault* (CF_{dtf}). The simulation results are shown in Fig. 4.7, where V_{T0}/V_{F_0} and V_{T2}/V_{F2} denote the voltage of T/F node of cell 0 and cell 2 separately. Due to the bridge between node F0 and T2, V_{F0} is forced to around 0v when cells 0, 1 are placed into drowsy mode. When cell 0 is woke up, it is inversed.

Note that even when a SD exists within a cell, it might affect other cells. In this work, we found that a SD between either node (T or F) of a cell and either (BL or $\overline{BL}$) of its bit-lines can cause FP2 faults. These faults are listed in BC2 and BC3 section of Table 4.1. Take the *deceptive read destructive coupling fault*

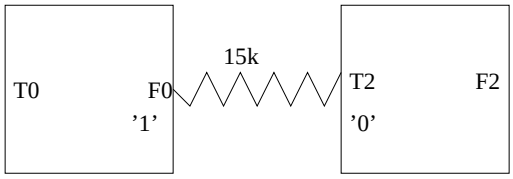

Figure 4.6: Bridge fault between two adjacent cells.

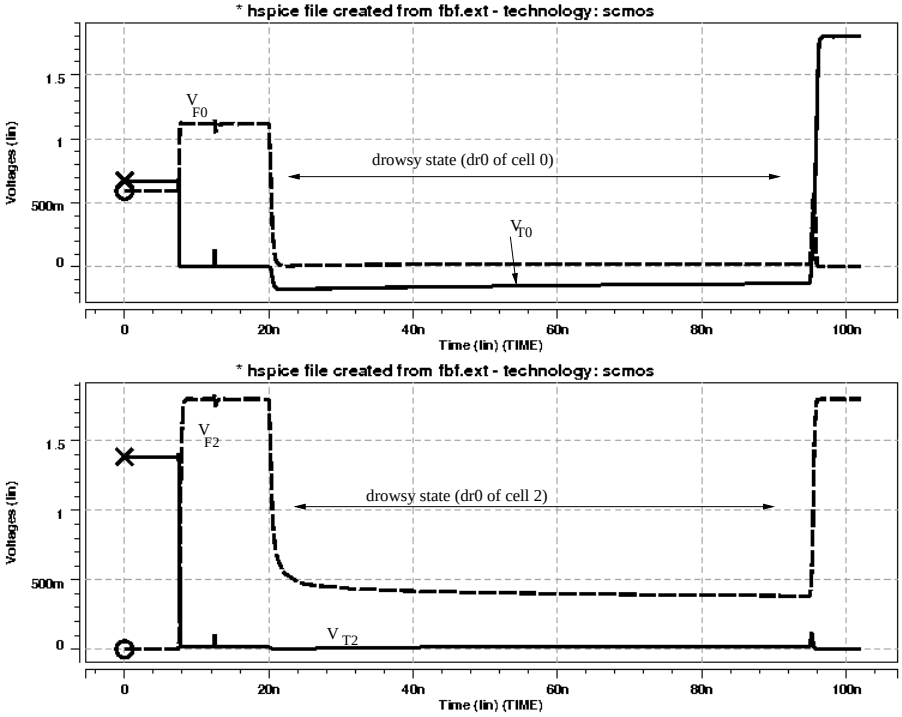

Figure 4.7: Coupling drowsy transition fault (CF_{dtf}) caused by the bridge fault in Fig. 4.6.

(CF_{drd}) in resistance region II of BC2 as an example. We assume that a bridge fault (5000Ω) exists between node T and bitline BL in cell 0 of Fig. 4.1. In Fig. 4.8, V_{T0}/V_{F0} and V_{T1}/V_{F1} denote the voltage of T/F node of cells 0, 1. *Write*/*Read* denotes the write/read operation applied to cells 0, 1. Curve *d0* denotes the output of the read operation of cell 0 or 1. As is shown in Fig. 4.8, cell 0 and cell 1 are successfully written a logic '0' and '1' separately after the *write cell 0* and *write cell 1* operations. Then a *read* operation is applied to cell 1. It can be seen from curves V_{T1}/V_{F1} that cell 1 manifest its defect during the *read* operation. The FP2 fault is then be abstracted as $< 0; r1/\downarrow/1 >_{0,1}$. This fault affects all other cells in the same column with cell 0. During testing procedure, we only consider its adjacent cell, since the detection of one can derive the detection of other cells.

Based on simulations of SDs between two neighbor cells in same row (or column, diagonal), the FFM2s have been derived as following:

- *Disturb coupling fault (CF_{ds})* : the v-cell undergoes a transition due to a write or read operation performed to the a-cell. The CF_{ds} consists of eight FPs: $< w0; 0/\uparrow/- >$, $< w0; 1/\downarrow/- >$, $< w1; 0/\uparrow/- >$, $< w1; 1\downarrow/- >$, $< r0; 0/\uparrow/- >$, $< r0; 1/\downarrow/- >$, $< r1; 0/\uparrow/- >$, $< r1; 1/\downarrow/- >$. It can be caused by bridge between a node of a cell and bitline within a cell (BC2, BC3).

- *State coupling fault (CF_{st})* : the v-cell is forced to a certain logic value ('0' or '1') when the aggressor has a specific logic value. CF_{st} consists of four FPs: $< 1; X/0/- >$, $< 1; X/1/- >$, $< 0; X/0/- >$, $< 0; X/1/- >$. It can be caused by defects like: (a) bridge between nodes of two adjacent cells in a row (rBCC1, rBCC2), (b) bridge between nodeds of two adjacent

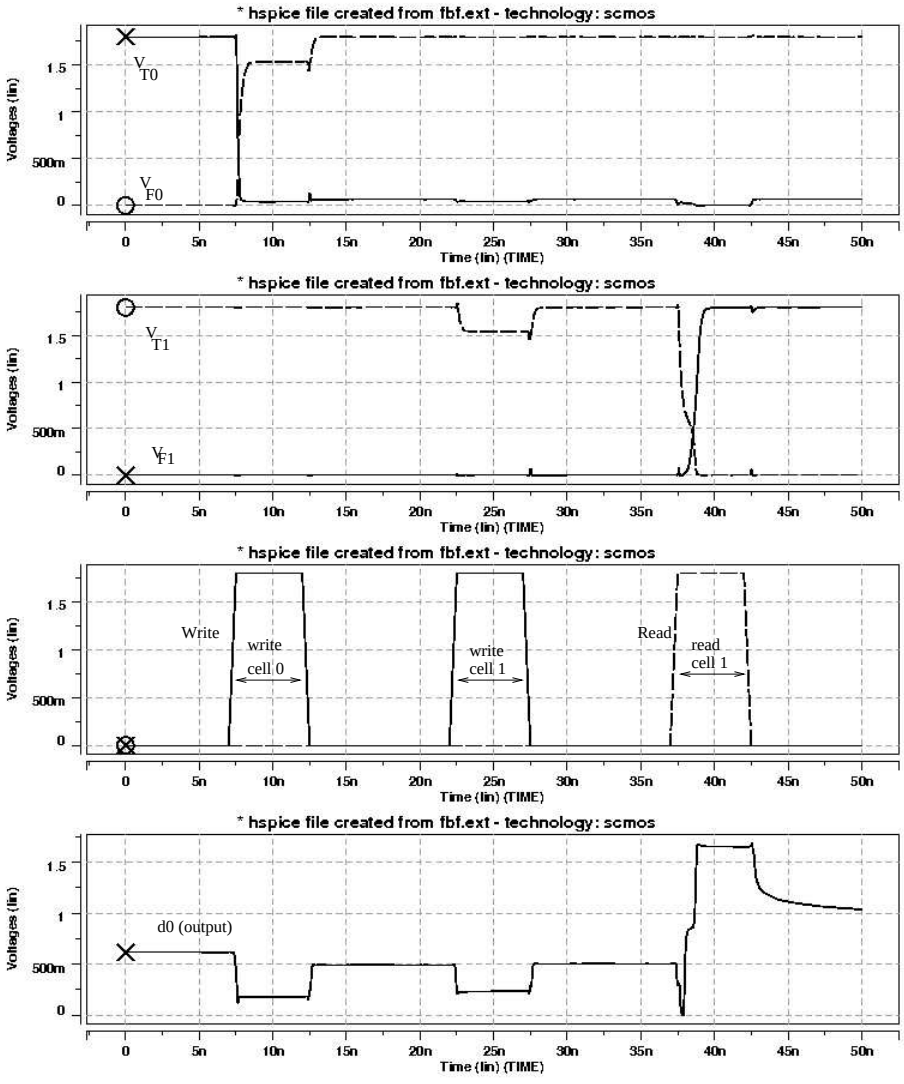

Figure 4.8: Deceptive read destructive coupling fault (CF_{drd}) at BC2 (5000Ω).

cells in a column (cBCC1, cBCC2), and (c) bridge between nodes of two adjacent cells within a diagonal (dBCC1, dBCC2).

- *Read destructive coupling fault (CF_{rd})* : when the aggressor cell is in a specific state, a rx operation applied to the v-cell causes a transition in the v-cell and returns an incorrect value $\overline{x}$. CF_{rd} consists of four FPs: $< 0; r0/\uparrow /1 >$, $< 0; r1/\downarrow /0 >$, $< 1; r0/\uparrow /1 >$, $< 1; r1/\downarrow /0 >$. It can be caused by: (a) bridge between a node of a cell and bitline within a cell (BC2, BC3), and (b) bridge involving a bitline (rBCC3, rBCC4, rBCC5, rBCC6).

- *Deceptive read destructive coupling fault (CF_{drd})* : when the a-cell has a specific logic value, a rx operation applied to the v-cell causes a transition in the cell while returns the correct value x. CF_{drd} consists of four FPs: $< 0; r0/\uparrow /0 >$, $< 0; r1/\downarrow /1 >$, $< 1; r1/\downarrow /1 >$, $< 1; r0/\uparrow /0 >$. It can be caused by: (a) bridges within a cell (BC2, BC3), and (b) bridge between cells (rBCC3, rBCC4).

- *Incorrect write coupling fault (CF_{iw})* : when a write operation is applied to the a-cell, the v-cell in same cache line will fail to undergo a transition. It consists of four FPs: $< w0; w\downarrow /1/- >$, $< w1; w\downarrow /1/- >$, $< w1; w\uparrow /0/- >$ and $< w0; w\uparrow /0/- >$. CF_{iw} can be caused by bridges involving bitlines (rBCC3, rBCC4, rBCC5, rBCC6).

- *Drowsy coupling transition fault (CF_{dtf})* : a drowsy operation performed on the v-cell causes a transition in the v-cell. CF_{dtf} consists of eight FPs: $< 0; dr0/1/- >$, $< 0; dr1/0/- >$, $< 1; dr0/1/- >$, $< 1; dr1/0/- >$, $< dr0; dr0/1/- >$, $< dr0; dr1/0/- >$, $< dr1; dr0/1/- >$, $< dr1; dr1/0/$

$->$. It can be caused by defects like: (a) bridge between two cells of same row (rBCC1, rBCC2), (b) bridge between two adjacent cells in same column (cBCC1, cBCC2), and (c) bridge between nodes of two adjacent cells in same diagonal (dBCC1, dBCC2). $<0;dr0/1/->$, $<0;dr1/0/->$, $<1;dr0/1/->$, and $<1;dr1/0/->$ can only exist in data caches, since in instruction caches, all the cache lines in a sub-bank enter into drowsy mode simultaneously.

- *Drowsy coupling write destructive fault (CF_{wdtf})*: a write operation applied to the a-cell caused a transition in the v-cell which is in drowsy mode. CF_{wdtf} consists of four FPs : $<w1;dr0/1/->$, $<w0;dr0/1/->$, $<w1;dr1/0/->$, $<w0;dr1/0/->$. It can be caused by bridges between nodes of two adjacent cells in the same column (cBCC1, cBCC2), or two adjacent cells in the same diagonal (dBCC1, dBCC2). Obviously, this fault can only exist in data caches, because all caches lines within an instruction cache enters into drowsy mode at the same time.

- *Drowsy coupling destructive transition fault (CF_{tdtf})* : the v-cell contains value x before changes to $\overline{x}$ after some period of time. CF_{tdtf} consists of six FPs: $<1;dr0_T/1/->$, $<0;dr1_T/0/->$, $<dr0;dr1_T/0/->$, $<dr1;dr1_T/0/->$, $<dr0;dr0_T/1/->$, $<dr1;dr0_T/1/->$. It can be caused by bridge between nodes of two adjacent cells (rBCC1, rBCC2, cBCC1, cBCC2, dBCC1, dBCC2). $<1;dr0_T/1/->$, $<0;dr1_T/0/->$, $<dr0;dr1_T/0/->$, and $<dr1;dr1_T/0/->$ can only exist in data caches, since in instruction caches, all the cache lines in a sub-bank enter

into drowsy mode simultaneously.

Table 4.4: Bridge defects between cells in the same row

Name	Resistance	Fault behavior	Comp. behavior	Class	FFM
rBCC1$(T0-T2)$	I	$<0;\forall/0/->_{i,j}$	$<1;\forall/0/->_{0,2}$ $<0;\forall/1/->_{2,0}$	FP2	CF_{st}
	II	$<dr0;dr1/0/->_{i,j}$	$<dr1;dr1/0/->_{0,2}$ $<dr0;dr0/1/->_{2,0}$	FP2	CF_{dtf}
	III	wF	wF	-	-
rBCC2$(T0-F2)$	I	$<0;\forall/1/->_{i,j}$	$<1;\forall/1/->_{0,2}$ $<0;\forall/0/->_{2,0}$	FP2	CF_{st}
	II	$<dr0;dr0/1/->_{0,2}$ $<dr1;dr1/0/->_{2,0}$	$<dr1;dr1/0/->_{0,2}$ $<dr0;dr0/1/->_{2,0}$	FP2	CF_{dtf}
	III	wF	wF	-	-
rBCC3 $(T0-BL1)$	I	$<dr0/1/->$	$<dr1/0/->$	FP2	DTF
		$<w0;w\uparrow/0/->_{0,2}$	$<w1;w\uparrow/0/->_{0,2}$	FP2	CF_{iw}
		$<r0;r1/\downarrow/0>_{2,0}$	$<r0;r0/\uparrow/1>_{2,0}$	FP2	CF_{rd}
	II	$<dr0/1/->$	$<dr1/0/->$	FP2	DTF
		$<r0;r1/\downarrow/0>_{2,0}$	$<r1;r0/\uparrow/1>_{2,0}$	FP2	CF_{rd}
	III	$<dr0/1/->$	$<dr1/0/->$	FP2	DTF
		$<r0;r1/\downarrow/1>_{2,0}$	$<r1;r0/\uparrow/0>_{2,0}$	FP2	CF_{drd}
	IV	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
	V	$<dr0_T/1/->$	$<dr1_T/0/->$	FP1	$DDRF$ †
	VI	wF	wF	-	-
rBCC4 $(T0-\overline{BL1})$	I	$<dr0/1/->$	$<dr1/0/->$	FP1	DTF
		$<w0;w\downarrow/1/->_{0,2}$	$<w1;w\downarrow/1/->_{0,2}$	FP2	CF_{iw}
		$<r1;r1/\downarrow/0>_{2,0}$	$<r0;r0/\uparrow/1>_{2,0}$	FP2	CF_{rd}
	II	$<dr0/1/->_0$	$<dr1/0/->_0$	FP1	DTF
		$<r1;r1/\downarrow/0>_{2,0}$	$<r1;r0/\uparrow/0>_{2,0}$	FP2	CF_{rd}
	III	$<dr0/1/->_0$	$<dr1/0/->_0$	FP1	DTF
		$<r1;r1/\downarrow/1>_{2,0}$	$<r1;r0/\uparrow/0>_{2,0}$	FP2	CF_{drd}
	IV	$<dr0/1/->_0$	$<dr1/0/->$	FP1	DTF
	V	$<dr0_T/1/->_0$	$<dr1_T/0/->$	FP1	$DDRF$ †
	VI	wF	wF	-	-
rBCC5 $(BL0-BL1)$	I	$<w0;w\uparrow/0/->_{0,2}$	$<w1;w\uparrow/0/->_{0,2}$	FP2	CF_{iw}
	II	$<r0;r1/\downarrow/0>_{0,2}$	$<r1;r1/\downarrow/0>_{0,2}$	FP2	CF_{rd}
	III	wF	wF	-	-
rBCC6 $(BL0-\overline{BL1})$	I	$<w1;w\uparrow/0/->_{0,2}$	$<w0;w\uparrow/0/->_{0,2}$	FP2	CF_{iw}
	II	$<r0;r0/\uparrow/1>_{0,2}$ $<r1;r1/\downarrow/0>_{2,0}$	$<r1;r0/\uparrow/1>_{0,2}$ $<r0;r1/\downarrow/0>_{2,0}$	FP2	CF_{rd}
	III	wF	wF	-	-

†: T is at least 2us

Table 4.5: Bridge defects between cells in the same column

Name	Resistance	Fault behavior	Comp. behavior	Class	FFM
cBCC1 $(T0-T1)$	I	$<0;\forall/0/->_{i,j}$	$<1;\forall/0/->_{0,1}$ $<0;\forall/1/->_{1,0}$	FP2	CF_{st}
		$<w1;dr0/1/->_{i,j}$	$<w0;dr0/1/->_{0,1}$ $<w1;dr1/0/->_{1,0}$	FP2	CF_{wdtf}
		$<w0;dr1/0/->_{i,j}$	$<w1;dr1/0/->_{0,1}$ $<w0;dr0/1/->_{1,0}$	FP2	CF_{wdtf}
	II	$<1;dr0/1/->_{i,j}$	$<0;dr0/1/->_{0,1}$ $<1;dr1/0/->_{1,0}$	FP2	CF_{dtf}
		$<0;dr1/0/->_{i,j}$ $<dr0;dr1/0/->_{i,j}$	$<1;dr1/0/->_{0,1}$ $<0;dr0/1/->_{1,0}$ $<dr1;dr1/0/->_{0,1}$ $<dr0;dr0/1/->_{1,0}$	FP2	CF_{dtf}
	III	$<1;dr0_T/1/->_{i,j}$	$<0;dr0_T/1/->_{0,1}$ $<1;dr1_T/0/->_{1,0}$	FP2	CF_{tdtf} †
		$<0;dr1_T/0/->_{i,j}$ $<dr0;dr1_T/0/->_{i,j}$	$<1;dr1_T/0/->_{0,1}$ $<0;dr0_T/1/->_{1,0}$ $<dr1;dr1_T/0/->_{0,1}$ $<dr0;dr0_T/1/->_{1,0}$	FP2	CF_{tdtf} †
	IV	wF	wF	-	-
cBCC2 $(T0-F1)$	I	$<0;\forall/1/->_{i,j}$	$<1;\forall/1/->_{0,1}$ $<0;\forall/0/->_{1,0}$	FP2	CF_{st}
		$<w0;dr0/1/->_{1,0}$ $<w1;dr1/0/->_{0,1}$	$<w0;dr1/0/->_{1,0}$ $<w0;dr1/0/->_{0,1}$	FP2	CF_{wdtf}
		$<w0;dr0/1/->_{0,1}$ $<w1;dr1/0->_{1,0}$	$<w1;dr0/1/->_{0,1}$ $<w1;dr0/1/->_{1,0}$	FP2	CF_{wdtf}
	II	$<0;dr0/1/->_{1,0}$ $<1;dr1/0/->_{0,1}$	$<0;dr1/0/->_{1,0}$ $<0;dr1/0/->_{0,1}$	FP2	CF_{dtf}
		$<0;dr0/1/->_{0,1}$ $<1;dr1/0/->_{1,0}$ $<dr0;dr0/1/->_{0,1}$ $<dr1;dr1/0/->_{1,0}$	$<1;dr0/1/->_{0,1}$ $<1;dr0/1/->_{1,0}$ $<dr1;dr0/1/->_{0,1}$ $<dr1;dr0/1/->_{1,0}$	FP2	CF_{dtf}
	III	$<0;dr1_T/0/->_{1,0}$ $<1;dr1_T/0/->_{0,1}$	$<0;dr0_T/1/->_{1,0}$ $<0;dr1_T/0/->_{0,1}$	FP2	CF_{tdtf} †
		$<0;dr0_T/1/->_{0,1}$ $<1;dr1_T/0/->_{1,0}$ $<dr0;dr0_T/1/->_{0,1}$ $<dr1;dr1_T/0/->_{1,0}$	$<1;dr0_T/1/->_{0,1}$ $<1;dr0_T/1/->_{1,0}$ $<dr1;dr0_T/1/->_{0,1}$ $<dr1;dr0_T/1/->_{1,0}$	FP2	CF_{tdtf} †
	IV	wF	wF	-	-
cBCC3 $(T0-WL1)$	I	$<\forall/0/->_0$ $<\forall/0/->_1$	$<\forall/1/->_0$ $<\forall/0/->_1$	FP1	SAF
	II	$<\forall/0/->_0$	$<\forall/1/->_0$	FP1	SAF
	III	$<dr1/0/->$	$<dr0/1/->$	FP1	DTF
	IV	$<dr1_T/0/->$	$<dr0_T/1/->$	FP1	$DDRF$ †
	V	wF	wF	-	-
cBCC4 $(WL0-WL1)$	I	$<wx/X/->$ $<rx/x/X>$	-	FP1	USF
	II	wF	-	-	-

†:T is at least 2us

Table 4.6: Bridge defects between cells in the same diagonal

Name	Resistance	Fault behavior	Comp. behavior	Class	FFM
dBCC1 $(T0-T3)$	I	$<0;\forall/0/->_{i,j}$	$<1;\forall/0/->_{0,3}$ $<0;\forall/1/->_{3,0}$	FP2	CF_{st}
		$<w1;dr0/1/->_{i,j}$	$<w0;dr0/1/->_{0,3}$ $<w1;dr1/0/->_{3,0}$	FP2	$CF_{w\,dtf}$
		$<w0;dr1/0/->_{i,j}$	$<w1;dr1/0/->_{0,3}$ $<w0;dr0/1/->_{3,0}$	FP2	$CF_{w\,dtf}$
	II	$<1;dr0/1/->_{i,j}$	$<0;dr0/1/->_{0,3}$ $<1;dr1/0/->_{3,0}$	FP2	CF_{dtf}
		$<0;dr1/0/->_{i,j}$ $<dr0;dr1/0/->_{i,j}$	$<1;dr1/0/->_{0,3}$ $<0;dr0/1/->_{3,0}$ $<dr1;dr1/0/->_{0,3}$ $<dr0;dr0/1/->_{3,0}$	FP2	CF_{dtf}
	III	$<1;dr0_T/1/->_{i,j}$	$<0;dr0_T/1/->_{0,3}$ $<1;dr1_T/0/->_{3,0}$	FP2	CF_{tdtf} †
		$<0;dr1_T/0/->_{i,j}$ $<dr0;dr1_T/0/->_{i,j}$	$<1;dr1_T/0/->_{0,3}$ $<0;dr0_T/1/->_{3,0}$ $<dr1;dr1_T/0/->_{0,3}$ $<dr0;dr0_T/1/->_{3,0}$	FP2	CF_{tdtf} †
	IV	wF	wF	-	-
dBCC2 $(T0-F3)$	I	$<0;\forall/1/->_{i,j}$	$<1;\forall/1/->_{0,3}$ $<0;\forall/0/->_{3,0}$	FP2	CF_{st}
		$<w0;dr0/1/->_{3,0}$ $<w1;dr1/0/->_{0,3}$	$<w0;dr1/0/->_{3,0}$ $<w0;dr1/0/->_{0,3}$	FP2	$CF_{w\,dtf}$
		$<w0;dr0/1/->_{0,3}$ $<w1;dr1/0/->_{3,0}$	$<w1;dr0/1/->_{0,3}$ $<w1;dr0/1/->_{3,0}$	FP2	$CF_{w\,dtf}$
	II	$<0;dr0/1/->_{3,0}$ $<1;dr1/0/->_{0,3}$	$<0;dr1/0/->_{3,0}$ $<0;dr1/0/->_{0,3}$	FP2	CF_{dtf}
		$<0;dr0/1/->_{0,3}$ $<1;dr1/0/->_{3,0}$ $<dr0;dr0/1/->_{0,3}$ $<dr1;dr1/0/->_{3,0}$	$<1;dr0/1/->_{0,3}$ $<1;dr0/1/->_{1,0}$ $<dr1;dr0/1/->_{0,1}$ $<dr1;dr0/1/->_{3,0}$	FP2	CF_{dtf}
	III	$<0;dr1_T/0/->_{3,0}$ $<1;dr1_T/0/->_{0,3}$	$<0;dr0_T/1/->_{3,0}$ $<0;dr1_T/0/->_{0,3}$	FP2	CF_{tdtf} †
		$<0;dr0_T/1/->_{0,3}$ $<1;dr1_T/0/->_{3,0}$ $<dr0;dr0_T/1/->_{0,3}$ $<dr1;dr1_T/0/->_{3,0}$	$<1;dr0_T/1/->_{0,3}$ $<1;dr0_T/1/->_{3,0}$ $<dr1;dr0_T/1/->_{0,3}$ $<dr1;dr0_T/1/->_{3,0}$	FP2	CF_{tdtf} †
	IV	wF	wF	-	-

†: T is at least 2us

Chapter 5

March Algorithm and Built-in Self Repair (BISR)

Based on the fault models, we first use a voltage window detector circuit to identify defects that result in undefined state. A few simplification rules are then developed to reduce the number of faults that must be dealt with. A drowsy march algorithm is proposed to detect all traditional faults and drowsy faults. Finally, a built-in self-repair circuit is designed to tolerate drowsy defects occurring in drowsy cache devices.

5.1 March DWOM

Since cache line design is used in the cache architecture, the cache we implemented can be treated as a *word-oriented memory (WOM)* for the testing algorithm. A word-oriented memory contains B bit per word, where B is greater than

2 and is usually a power of two. Many memory test algorithms are based on *bit-oriented* design, i.e., *read* and *write* operations access only one bit in the memory. It is mentioned that word-oriented memories can be tested by *repeated application* of a test for *bit-oriented memories*, whereby a different data background is used during each iteration [35]. The new march algorithm referred as Drowsy Word-Oriented Memory (DWOM) march algorithm is introduced in Table 5.1.

To describe March DWOM, the traditional march notation will be used. A complete *march test* consists of a finite sequence of *march elements* [33]. A *march test* can be delimited by a pair of parentheses '$\{\cdots\}$'. A *march element* is composed of a finite sequence of operations applied to every cell in memory before the next cell can be proceeded. A *march element* can be denoted by a pair of brackets '$(\cdots)$', and it can be done in two address orders: an increasing ($\Uparrow$) address order (from address 0 to address n-1), or a decreasing ($\Downarrow$) address order. The test shown in Table 5.1 is based on the assumption that SDs can only exists within one cell or between two adjacent cells. It detects all FFMs with a deterministic output at the sense amplifier.

For FFMs with an undefined state output (X), i.e., the output voltage is between HI and LOW, the proposed march algorithm can also detect these FFMs by using a *voltage window detector circuit* shown in Fig. 5.1. Note that *A* and *B* in Fig. 5.1 are both operational amplifiers (op-amps), and we have $V_{ref1} = \frac{R3}{R1+R2+R3}$, and $V_{ref2} = \frac{R2+R3}{R1+R2+R3}$. The output V_{out} is HIGH when we have $V_{ref1} < V_{in} < V_{ref2}$, and is LOW otherwise. As a result, by configuring R1, R2, and R3 properly, this circuit can detect the undefined state.

In the following discussions, SD_i is used to describe the SDs mentioned in

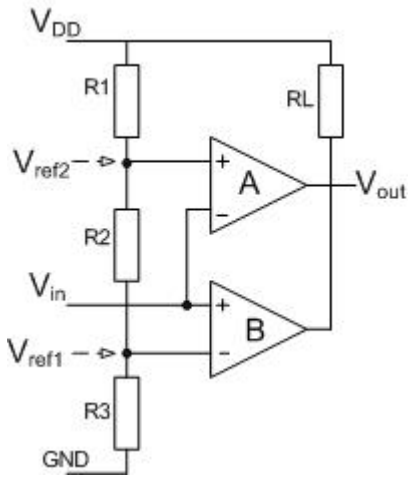

Figure 5.1: Voltage window detector circuit [34].

Chapter 4, where $SD \in \{BC1, \cdots, dBCC2\}$, and $i \in \{I, II, \cdots\}$ denotes the resistance region. For example, $BC1_I$ denotes the FFM of BC1 defect in resistance region I, which is an USF.

5.2 Fault Model Simplification

Chapter 4 gives a detailed description for FFMs of each SD in different resistance regions. However, the complexity of our march testing algorithm can be further reduced by the following *simplification rules*.

Rule (a) **SDs with both drowsy FPs and normal FPs**. We found that many SDs have both drowsy FPs and normal FPs within the same resistance region. Either of them (drowsy FPs and normal FPs) can be used to detect the SD. However, drowsy fault behavior can only be observed when a cell is placed into drowsy mode. Since each drowsy operation needs extra circuit operations (setting drowsy bits, switching supply voltages, *etc.*), in this

Table 5.1: March DWOM

$\{\Downarrow (w_0);$	M_1
$\Uparrow (r_0, w_1, r_1, w_0);$	M_2
$\Downarrow (r_0, r_0);$	M_3
$dr_T;$	M_4
$\Uparrow (r_0);$	M_5
$\Uparrow (w_1);$	M_6
$\Downarrow (r_1, w_0, r_0, w_1);$	M_7
$\Uparrow (r_1, r_1);$	M_8
$dr_T;$	M_9
$\Downarrow (r_1);$	M_{10}
$\Downarrow (w_0(odd), w_1(even));$	M_{11}
$dr_T;$	M_{12}
$\Downarrow (r_0(odd), r_1(even));$	M_{13}
$\Downarrow (w_1(odd), w_0(even));$	M_{14}
$dr_T;$	M_{15}
$\Downarrow (r_1(odd), r_0(even)); \}$	M_{16}

work, when a SD has both drowsy FPs and normal FPs within a specific resistance region, march test will be developed mostly based on the normal FPs. For example, $rBCC3_{II}$ in Table 4.5 has two FPs ($< dr0/1/- >$ and $< r1; r1/ \downarrow /0 >_{2,0}$), so our march algorithm is develped based on $< r1; r1/ \downarrow /0 >_{2,0}$. Further, CF_{wdtf} only comes with CF_{st} in all cases, so our march algorithm detects SDs with both CF_{wdtf} and CF_{st} (cBCC1, cBCC2, dBCC1, dBCC2) by testing CF_{st} only. Thus, CF_{wdtf} does not need to be considered when deriving the march test in this work.

But, it is different for SDs with data retention fault in normal state (DRFs) and those in drowsy mode (DDRFs). In Table 4.2, we found that if a defect causes DRF ($< x_T/\overline{x}/- >$), definitely it will also cause DDRF ($drx_T/\overline{x}/- >$). On the other hand, some SDs have only DDRF faults. In this work, therefore, a march algorithm will be developed based on DDRF

($drx_T/\overline{x}/->$) to detect these SDs within one march test step.

Rule (b) **SDs with different normal FPs**. Some SDs have more than one normal FP within a specific resistance region. The march can be developed based on either of them. For example, in Tables 4.1 and 4.4, if a SD has CF_{iw} (e.g., $<0;w\uparrow/0/->$) fault, it will definetly have CF_{ds} (e.g., $<w0;1/\downarrow/->$) fault or CF_{rd} (e.g., $<r0;r1/\downarrow/->$) fault. Thus, CF_{iw} is not required to derive our march algorithm, since either CF_{ds} or CF_{rd} will be used to detect this fault.

Rule (c) **SDs with drowsy and non-drowsy aggressors**. In this research, only CF_{dtf}s and CF_{tdtf}s belong to this category. For CF_{dtf}s, if a SD has $<x;drz/\overline{z}/->$, it will also have $<drx;drz/\overline{z}/->$. CF_{tdtf}s have the same fault behavior in that if a SD has $<x;drz_T/\overline{z}/->$, it will also have $<drx;drz_T/\overline{z}/->$. Since each drowsy operation needs extra circuit operations (setting drowsy bits, switching supply voltages, *etc.*), it is better to use as small number of drowsy steps as possible. In this work, march test is based on drowsy aggressor testing such as $<drx;drz/\overline{z}/->$ and $<drx;drz_T/\overline{z}/->$, since the whole memory can be placed into drowsy state to detect this fault, after specific logic values to x and z are assigned. This strategy is especially useful when a circuit delay is needed, i.e., $<x;drz_T/\overline{z}/->$ and $<drx;drz_T/\overline{z}/->$. If march test is based on $<x;drz_T/\overline{z}/->$, each line must be placed into drowsy state for T time units (T is usually 2us) with its aggressor in normal mode, and the testing time will be excessive. Nevertheless, if march test is based on $<$

$drx; drz_T/\overline{z}/- >$, the entire memory needs only to be placed into drowsy state for T time units, which can save a lot of testing time.

Rule (d) **SDs with or without data retention fault in drowsy mode**. In this research, we found that DTF ($< drx/\overline{x}/- >$) and DDRF ($< drx_T/\overline{x}/- >$ are similar, except that DDRF can only be observed after T delay time units. It is obvious that march element based on DDRF can also detect DTF, since the DTF fault can also be observed after T delay time units. The case of CF_{dtf} ($< drx; drz/\overline{z}/- >$) and CF_{tdtf} ($< drx; drz_T/\overline{z}/- >$) is similar. As a result, DTF (CF_{dtf}) can be treated as DDRF (CF_{tdtf}) when we develope our march algorithm.

5.3 Fault Coverage of March DWOM

In this research, we observed 13 kinds of fault behaviors in normal mode, and 6 fault behaviors in drowsy mode for all possible SDs.

- All SDs which have FPs in normal mode can be detected by March DWOM proposed in this research, because our DWOM includes all operations of the march algorithm in [2].

- All DTFs, DUFs and $DDRF$s are detected by March DWOM. Since for each single cell, a '0' and a '1' is read after the dr_T drowsy operation. Thanks to the introduction of the voltage window detector circuit, DUFs ($< drx/X/- >$) can be detected by M_5 and M_6. According to the simplification rule (d) of the previous section (Section 5.2), DTFs can be treated as

$DDRF$s ($< drx_T/\overline{x}/- >$) when developing a march element, and DDRFs can be detected by march element M_5 or M_{10}. As a result, all SDs with DTFs, DUFs or $DDRF$s can be detected by march operation M_5 or M_{10}.

- ALL CF_{dtf}s and CF_{tdtf}s are detected. Based on the simplification rule (c) of the previous section (Section 5.2), SDs with fault behavior $< x; dry/\overline{y}$ $/- >$ also have $< drx; dry/\overline{y}/- >$. Due to simplification rule (d), SDs with CF_{dtf}s can be detected by test operation for CF_{tdtf} ($< drx_T; dry_T/\overline{y}/$ $- >$). So, $< drx; dry_T/\overline{y}/- >$ can be used to represent all fault behaviors of both CF_{dtf}s and CF_{tdtf}s. As mentioned in Chapter 4, we assumed that all SDs can only exist either within a cell or between two *adjacent* cells. The march operations M_2, M_7, M_{11} and M_{14} can generate all required patterns to detect (observe) all CF_{dtf}s and CF_{tdtf}s by M_5, M_{10}, M_{13} and M_{16}.

- All CF_{wdtf}s ($< wx; dry/\overline{y}/- >$) are not required to be detected. From the simplification rule (a) of the previous section (Section 5.2), we can see that CF_{wdtf} faults always co-exist with CF_{st} faults together. Thus, a SD with CF_{wdtf} fault behavior can be detected by the test operation which detects the corresponding CF_{st}.

- All faults in data caches and instruction caches are detected. As mentioned before, CF_{wdtf}s ($< wx; dry/\overline{y}/- >$), some of CF_{dtf}s ($< x; dry/\overline{y}/- >$), and some of CF_{tdtf}s ($< x; dry_T/\overline{y}/- >$) are the difference between the fault behaviors of data caches and instruction caches. Fortunately, base on simplification rule (d), SDs with CF_{wdtf} are detected; based on simplification rule (c), all CF_{dtf}s and CF_{tdtf}s can be detected by $< drx; dry_T/\overline{y}/- >$.

In conclusion, March DWOM can detect all the faults in data caches and instruction caches.

5.4 Buit-in Self Repair

It is found that, for some defects, the cell manifests itself only under the drowsy mode. For example, in the resistance regions IV, and V of the BC3 defect in Table 4.1, the cell works properly under normal mode, but its value is inversed when it is placed into drowsy mode. Instead of discarding a chip when a fault is detected, we can still use it if it is only a drowsy-mode defect.

The basic idea is that during the *testing* mode, a `drowsy_mask[i]` register bit will be set to '0' if cache line i only manifests itself in drowsy mode. This can be done in march elements M_5, M_{10}, M_{13} and M_{16} in Table 5.1. During the *working* mode of the cache, the FSM checks `drowsy_mask[i]` when issuing the drowsy control signal to cache line i. This can be done by an AND gate, and the BISR architecture is shown in Fig. 5.2.

The BISR differs for data caches and instruction caches. For data caches, the drowsy control circuit has a register bit (i.e., `drowsy_mask[i]`) for each cache line. Instruction caches are divided into several sub-banks to implement the drowsy control. Each sub-bank needs only one *drowsy* signal. To utilize the BISR feature, each cache line within a subbank is also connected to a `drowsy_mask[i]` register as shown in Fig. 5.2. Hence, a subbank can still be placed into drowsy mode even when a cache-line within this subbank fails in drowsy mode. When the subbank is placed into drowsy mode, all it cache lines are in drowsy mode except

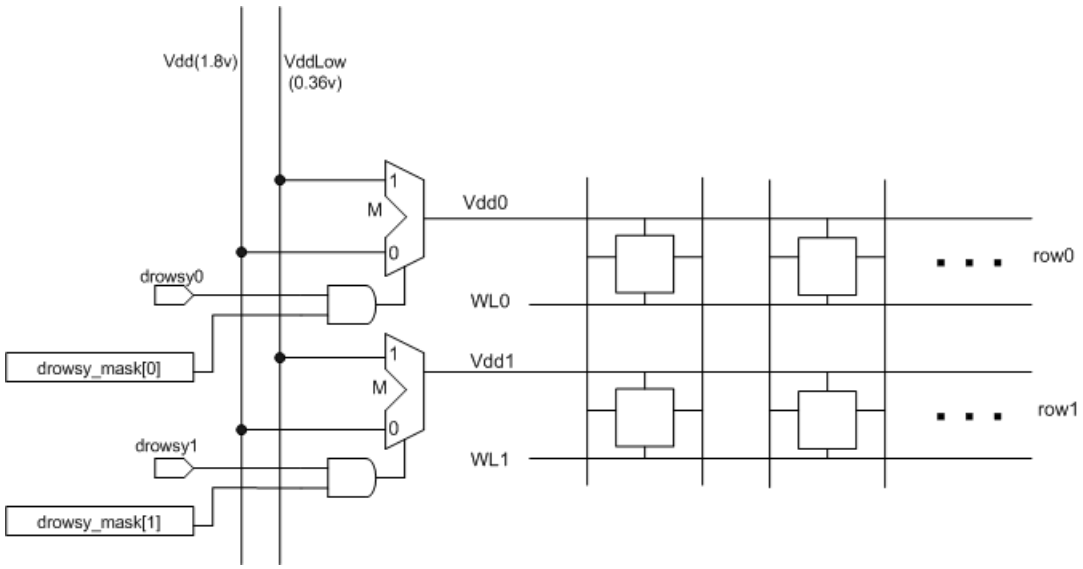

Figure 5.2: BISR solution of drowsy cache

the faulty one. As a conclusion, this BISR works well for both data caches and instruction caches.

Chapter 6

Conclusions and Future Work

In this research, we implemented a full-functional drowsy SRAM cache with peripheral circuits like address decoder, sense amplifier, etc. Since the sub-threahold leakage power decreases significantly with the decreasing supply voltage, we derived the minimum stand-by voltage which can still retain the state of each cell. Based on the assumption that spot-defects (SDs) can only exist either within a cell or between two adjacent cells in the same row/column/diagonal, we simulate all possible SDs with different resistance region (from 0 to ∞) in standard mode and drowsy mode separately. Six new faults (DTF, DUF, DDRF, CF_{dtf}, CF_{wdtf}, and CF_{tdtf}) appear with the introduction of drowsy operations. *Drowsy coupling write destructive fault* (CF_{wdtf}) can only exist in data caches, since data caches and instruction caches tend to have different architectures and scheduling strategies. However, during the simplification process of all fault behaviors, we found that CF_{wdtf} always comes with CF_{st} in all cases. As a result, CF_{wdtf} can be negligible, and hence the test algorithm we derived can detect SDs in both data

caches and instruction caches. A data-background based march algorithm March DWOM has been developed. A voltage window detector circuit is used to detect faults with undefined state. With the benefit of the detection of undefined state, March DWOM can detect all SDs in the drowsy cache we implemented, and thus it has 100% fault coverage.

Traditionally, march test for word-oriented memories can be derived from march test for bit-oriented memories, where a different data background is used in each iteration [35]. Because of this, March DWOM is based on the bit-oriented march test of [2] in this research. However, this method is not efficient and time consuming. The work of [35] already presented a method to reduce the number of iterations and number of background data. In the future, we will work on the simplification of March DWOM to reduce the testing time.

Bibliography

[1] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Circuit and microarchitectural techniques for reducing cache leakage power," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 2, pp. 167-184, Feb. 2004.

[2] S. Hamdioui, A. J. van de Goor, "An experimental analysis of spot defects in SRAMs: realistic fault models and tests," *Proceedings of Asian Test Symposium*, pp. 131-138, 2000.

[3] Johan de Gelas., *The quest for more processing power*, http://www.anandtech.com, Feb. 2005.

[4] S. Borkar, *Gigascale integration-challenges and opportunities*, http://www.intel.com/cd/ids/developer/asmo-na/eng/strategy/182440.htm.

[5] Z. Al-Ars, S. Hamdioui, and A. J. van de Goor, "A fault primitive based analysis of linked faults in RAMs," *Proceedings of Design and Testing of Memory Technology*, pp. 33-39, 2003.

[6] E. S. Muhammad, M. I. Elmasry, "Split-gate logic circuits for multi-threshold technologies," *Proceedings of International Symposium on Circuits and Systems*, pp. 798-801, 2001.

[7] M. Powell, S. H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-V_{dd}: a circuit technique to reduce leakage in deep-submicron cache memories," *Proceedings of International Symposium on Low-power Electronics and Design*, pp. 90-95, 2000.

[8] Z. Al-Ars and A. J. van de Goor, "Static and dynamic behavior of memory cell array spot defects in embedded DRAMs," *IEEE Transactions on Computers*, vol. 52, no. 3, pp. 293-309, March 2003.

[9] H. C. Tsai, K. T. Cheng, "On improving test quality of scan-based BIST," *IEEE Transactions on Computer-aided Design of Intergrated Circuits and Systems*, vol. 19, no. 8, pp. 928-938, Aug. 2000.

[10] S. Hamdioui, G. Gaydadjiev, *Future chanllenges in memory testing*, http://ce.et.tudelft.nl/publicationfiles/818_26_hamdioui1.pdf.

[11] D. R. Aadsen, *et al.*, "Automated BIST for regular structures embedded in ASIC devices," *AT&T Technical Journal*, vol. 69, no. 3, pp. 258-261, May. 1990.

[12] Y. Nagura, M. Mullins, Y. Fujiwara, *et al.*, "Test cost reduction by at-speed BISR for embedded DRAMs," *Proceedings of International Test Conference*, pp. 182-187, 2001.

[13] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada, "1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *IEEE J. Solid-State Circuits*, vol. 30, no. 8, pp. 847-857, Aug. 1995.

[14] M. J. Chen, H. T. Huang, C. S. Hou, and K. N. Yang, "Back-gate biasenhanced band-to-band tunneling leakage in scaled MOSFETs," *IEEE Electron Device Lett.*, vol. 19, pp. 134-136, Apr. 1998.

[15] C. Piguet, *Low-power electronics design*, CRC Press LLC, 2005

[16] J. Hennessey and D. Goldberg, *Computer architecture: a quantitative approach*, Morgan Kaufmann, San Fransisco, 2003.

[17] K. Ghose and M. Kamble, "Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation," *Proceedings of IEEE/ACM International Symposium on Low-power Electronics and Design*, pp. 70-75, 1999.

[18] K. N. Yang, H. T. Huang, M. J. Chen, *et al.*, "Characterization and modeling of edge direct tunneling (EDT) leakage in ultrathin gate oxide MOSFETs," *IEEE Transactions on Electron Devices*, vol. 48, pp. 1159-1164, June 2001.

[19] M. Rosar, B. Leroy, and G. Schweeger, "A new model for the description of gate voltage and temperature dependence of gate induced drain leakage (GIDL) in the low electric field region," *IEEE Transactions on Electron Devices*, vol. 47, pp. 154-159, Jan. 1999.

[20] Y. S. Lin, C. C. Wu, C. S. Chang, R. P. Yang, W. M. Chen, J. J. Liaw, and C. Diaz, "Leakage scaling in deep submicron CMOS from SOC," *IEEE Transactions on Electron Devices*, vol. 49, no. 6, pp. 1034-1041, June 2002.

[21] S. Manne, A. Klauser, and D. Grunwald, "Pipeline gating: speculation control for energy reduction," *Proceedings of IEEE/ACM International Symposium on Computer Architecture*, pp. 132-141, 1998.

[22] S. Narendra, S. Borkar, V. De, D. Antoniadis, and A. Chandrakasan, "Scaling of stack effect and its application for leakage reduction," *Proceedings of IEEE/ACM International Symposium on Low-power Electronics and Design*, pp.195-200, Aug. 2001.

[23] A. Keshavarzi, K. Roy, and C. Hawkins, "Intrinsic leakage in low power deep submicron CMOS ICs," *Proceedings of IEEE International Test Conference*, pp. 146-155, 1997.

[24] J. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital integrated circuits: a design perspective*, 2nd ed., NJ:Prentice-Hall, Englewood Cliffs, 2002.

[25] H. Qin, Y. Cao, D. Markovic, A. Vladimirescu, and J. Rabaey, "SRAM leakage suppression by minimizing standby supply voltage," *Proceedings of International Symposium on Quality Electronic Design*, pp. 55-60, 2004.

[26] R. Kapur and M. Chandramouli, *Shifting from functional to structured techniques improves test quality*, http://www.eetimes.com/story/OEG20030228S0046y, 2003.

[27] T. Douseki and S. Mutoh, "Static-noise margin analysis for a scaled-down CMOS memory cell," *Electronics & Communications in Japan, Part II: Electronics*, vol. 75, no. 7, pp. 102-115, 1992.

[28] M. Lee, W. I. Sze, and C. M. Wu, "Static noise margin and soft-error rate simulations for thin film transistor cell stability in a 4Mbit SRAM design," *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 937-940, 1995.

[29] E. Seevinck, F. J. List, and J. Lohstroh, "Static-noise margin analysis of MOS SRAM cells," *IEEE J. Solid-State Circuits*, vol.22, no. 5, pp. 748-754, Oct. 1987.

[30] A. Aziz, *CMOS VLSI design: MOS behavior in DSM*, http://www.ece.utexas.edu/ adnan/vlsi-05/lec15DSM.ppt, 2004.

[31] J. P. Shen, W. Maly, and F. J. Ferguson, ”Inductive fault analysis of CMOS integrated circuits,” *IEEE Design and Test of Computers*, vol. 2, no. 6, pp. 13-26, Dec. 1985.

[32] A.J. van de Goor, *Testing semiconductor memories: theory and practice*, John Wiley & Sons, New York, 1991.

[33] D. S. Suk and S. M. Reddy, ”A march test for functional faults in semiconductor random-access memories,” *IEEE Transactions on Computers*, vol. 30, pp. 982-985, Dec. 1981.

[34] R. Paisley, *Voltage comparator information and circuits*, http://home.cogeco.ca/ rpaisley4/Comparators.html, 2003.

[35] A. J. van de Goor, and B.S. Tlili, ”March tests for word-oriented memories,” *Proceedings of IEEE Design, Automation and Test*, pp. 501-508, 1998.

Appendix A

C++ code

```
/*
 * tbt.cpp -- 2x2 SRAM cell spice file (tbt.mag) generator
 *
 * by:  Wei Pei
 *  wei.pei@gmail.com
 * Last modified: 06/03/2005
 */


/*
 * Input file format (I,W,C,R,D,E)
 * Idle, Write, Charge, Read, Drowsy,End
 * -----------------------------------
 * I
 * W 0 0 1
 * I
 * R 1
 * I
 * D 0 -1
 * I /----------------------------------------
 * W 1 0 0 / When some cachelines in drowsy mode,
 * I / others can be accessed within drowsy time
 * R 0 /
 * I /----------------------------------------
 * C 0 -1
 * I
 * E
 * -----------------------------------
 * $>: tbt < input.ptn >> output.sp
 *
 */
#include <iostream>
#include <string>

#define HI 1.8
#define LO 0.36

#define U "n"
#define D_TIME 75
#define RW_TIME 5 // read/write time
```

```
#define STEP 0.5
#define C_TIME 2
#define I_TIME 5
#define DA_TIME 10 // data access time
#define RW_DELAY 2

using namespace std;


void op_print( char opr, float val, int counter, int i, float &time, float &t_tag )
{
if( i==0 ) {
cout << "0" << U << " " << val << ' ';
}
else {
if( opr!='D' ) {
cout << time+STEP << U << " " << val << ' ';
}
}

if( (opr == 'W') || (opr == 'R') ) {
cout << time+RW_TIME << U << ' ' << val << ' ';
cout << time+RW_TIME+STEP << U << " 0 ";
time += DA_TIME;
cout << time << U << " 0";
}
else {
if( opr == 'D' ) {
t_tag = time + D_TIME;
}
else if( opr == 'C' ) {
// finish drowsy phase
cout << t_tag << U << " 0 ";
time = t_tag+C_TIME;
cout << t_tag+STEP << U << ' ' << val << ' ';
t_tag = 0;
}
else if( opr == 'I' )
time += I_TIME;

if( opr!='D' ) {
cout << time << U << " " << val;
}
}

if( (i+1) % 6 == 0 ) cout << " \n+";
else if( opr!='D' ) cout << ' ' ;
}


void data_print( char opr, float val, int counter, int i, float &time, float &t_tag )
{
if( i==0 ) {
cout << "0" << U << " " << val << ' ';
}
else {
if( opr!='D' ) {
cout << time+STEP << U << " " << val << ' ';
}
}
```

```
if( (opr == 'W') || (opr == 'R') )
time += DA_TIME;
else if( opr == 'D' )
t_tag = time+D_TIME;
else if( opr == 'C' ) {
cout << t_tag << U << " 0 ";
cout << t_tag+STEP << U << ' ' << val << ' ';
time = t_tag+C_TIME;
t_tag = 0;
}
else if( opr == 'I' )
time += I_TIME;

if( opr!='D' ) {
cout << time << U << " " << val;
}

if( (i+1) % 6 == 0 ) cout << " \n+";
else if( opr!='D' ) cout << ' ' ;
}


void vd_print( char opr, float val, int counter, int i, float &time, float &t_tag )
{
if( i==0 ) {
cout << "0" << U << " " << val << ' ';
}
else {
if( t_tag==0 ) {
cout << time+STEP << U << " " << val << ' ';
}
}

if( (opr == 'W') || (opr == 'R') )
time += DA_TIME;
else if( opr == 'D' ) {
t_tag = time+D_TIME;
cout << t_tag << U << ' ' << val;
}
else if( opr == 'C' ) {
cout << t_tag+STEP << U << ' ' << val << ' ';
time = t_tag+C_TIME;
t_tag = 0;
}
else if( opr == 'I' )
time += I_TIME;

if( t_tag==0 ) {
cout << time << U << " " << val;
}

if( (i+1) % 6 == 0 ) cout << "\n+";
else cout << ' ' ;
}


int main()
{
string ro, In0, In1, Vd0, Vd1, s_opr;
```

```
float t_time = 0;
char c_tmp;
int i_tmp,counter=0;
int i;
int j;
int v[2];
float f_tmp = 0;
char c_bf;
float t_tag = 0;

cin >> c_tmp;
while( c_tmp != 'E' )
{
s_opr.append( &c_tmp );
counter++;
/* addr val0 val1
 * W(rite) 0/1 0/1 0/1
 * R(ead) 0/1
 * D(rowsy) 0 1 -1
 * C(harge) 0 1 -1
 * I(dle)
 * E(nd)
 */
if( c_tmp == 'W' ) // w 0 0/1 0/1
{
cin >> i_tmp;
ro.append( (i_tmp==0) ? "0" : "1" );
cin >> i_tmp;
In0.append(  (i_tmp==0) ? "0" : "1" );
cin >> i_tmp;
In1.append(  (i_tmp==0) ? "0" : "1" );
}
else if( c_tmp == 'R' ) // R 0
{
cin >> i_tmp;
ro.append(  (i_tmp==0) ? "0" : "1" );
}
else if( c_tmp == 'D' ) // D 0 1 -1
{
for( i=0; i<2; i++ ) {
v[i] = 1;
}

cin >> i_tmp;
while( i_tmp != -1 )
{
v[ i_tmp ] = 0;
cin >> i_tmp;
}

Vd0.append( ( v[0] == 0 ) ? "0" : "1" );
Vd1.append( ( v[1] == 0 ) ? "0" : "1" );
}
else if( c_tmp == 'C' ) // C 0 1 -1
{
cin >> i_tmp;
while( i_tmp != -1 )
{
v[ i_tmp ] = 1;
cin >> i_tmp;
```

```
}

Vd0.append( ( v[0] == 0 ) ? "0" : "1" );
Vd1.append( ( v[1] == 0 ) ? "0" : "1" );
}
cin >> c_tmp;
}

/* pre */
cout << "\n.include ../spice_para.def" << endl;
cout << "VDD Vdd Gnd " << HI << endl;

/* print out Write*/
cout << "\n* Write Signal" << endl;
cout << "VWrite W Gnd PWL(";
t_time = 0;
t_tag = 0;
for( i=0; i<counter; i++ ) {
c_tmp = s_opr.at(i);
i_tmp = ( c_tmp == 'W') ? 1 : 0 ;
op_print( c_tmp, i_tmp*HI, counter, i, t_time, t_tag );
if( i==(counter-1) ) {
cout << " TD=" << RW_DELAY << U << ")" << endl;
}
}

/* Read */
cout << "\n* Read Signal" << endl;
cout << "VRead R Gnd PWL(";
t_time = 0;
t_tag = 0;
for( i=0; i<counter; i++ ) {
c_tmp = s_opr.at(i);
i_tmp = ( c_tmp == 'R') ? 1 : 0 ;
op_print( c_tmp, i_tmp*HI, counter, i, t_time, t_tag );
if( i==(counter-1) ) {
cout << " TD=" << RW_DELAY <<  U << ")" << endl;
}
}

/* In0 */
cout << "\n* In0" << endl;
cout << "VIn0 In0 Gnd PWL(" ;
t_time = 0;
t_tag = 0;
j = 0;
for( i=0; i<counter; i++ )
{
i_tmp = 0;
c_tmp = s_opr.at(i);
if( c_tmp == 'W' )
{
if( In0.at(j) == '1' ) i_tmp = 1;
j++;
}
data_print( c_tmp, i_tmp*HI, counter, i, t_time, t_tag );
if( i==(counter-1) ) {
cout << ')' << endl;
}
}
```

```
/* In1 */
cout << "\n* In1" << endl;
cout << "VIn1 In1 Gnd PWL(" ;
t_time = 0;
t_tag = 0;
j = 0;
for( i=0; i<counter; i++ )
{
i_tmp = 0;
c_tmp = s_opr.at(i);
if( c_tmp == 'W' )
{
if( In1.at(j) == '1' ) i_tmp = 1;
j++;
}
data_print( c_tmp, i_tmp*HI, counter, i, t_time, t_tag );
if( i==(counter-1) ) {
cout << ')' << endl;
}
}


/* row */
cout << "\n* row" << endl;
cout << "Vro ro Gnd PWL(" ;
t_time = 0;
t_tag = 0;
j = 0;
for( i=0; i<counter; i++ )
{
i_tmp = 0;
c_tmp = s_opr.at(i);
if( (c_tmp == 'W') || (c_tmp == 'R') )
{
if( ro.at(j) == '1' ) i_tmp = 1;
j++;
}
data_print( c_tmp, i_tmp*HI, counter, i, t_time, t_tag );
if( i==(counter-1) ) {
cout << ')' << endl;
}
}

/* Vd0 */
cout << "\n* Vd0" << endl;
cout << "Vvd0 Vd0 Gnd PWL(" ;
t_time = 0;
t_tag = 0;
j = 0;
c_bf = 'N';
for( i=0; i<counter; i++)
{
c_tmp = s_opr.at(i);
f_tmp = HI;
if( (c_tmp == 'D') || (c_tmp == 'C') )
{
if(Vd0.at(j) == '0') f_tmp = LO;
j++;
}
```

```
vd_print( c_tmp, f_tmp, counter, i, t_time, t_tag );
c_bf = c_tmp;
if( i==(counter-1) ) {
cout << ')' << endl;
}
}

/* Vd1 */
cout << "\n* Vd1" << endl;
cout << "Vvd1 Vd1 Gnd PWL(" ;
t_time = 0;
t_tag = 0;
j = 0;
c_bf = 'N';
for( i=0; i<counter; i++)
{
c_tmp = s_opr.at(i);
f_tmp = HI;
if( (c_tmp == 'D') || (c_tmp == 'C') )
{
if(Vd1.at(j) == '0') f_tmp = LO;
j++;
}
vd_print( c_tmp, f_tmp, counter, i, t_time, t_tag );
c_bf = c_tmp;
if( i==(counter-1) ) {
cout << ')' << endl;
}
}


cout << "\n.option post" << endl;
cout << ".tran 1" << U << ' ' << t_time << U  << endl;
cout << ".end" << endl;

return(0);
}
```