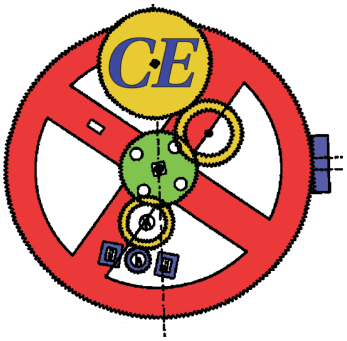


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

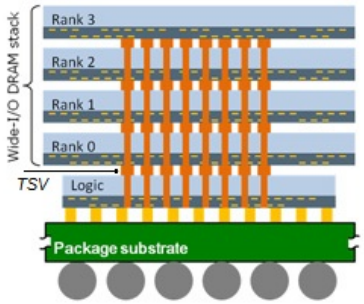
2013

MSc THESIS

Interconnect Testing for 3D Stacked Memories

Mahmoud Saleh Masadeh

Abstract



Three-dimensional stacked ICs (3D-SICs) technology based on Through-Silicon Vias (*TSVs*) provides numerous advantages as compared to traditional 2D-ICs. TSVs are holes going vertically through the chip silicon substrate filled with a conducting material. A potential application is a 3D-SIC where one or more memory dies are stacked on a logic die; thereby increasing the memory density, enhancing its throughput, and reducing its latency and power consumption as compared to planar ICs.
However, testing the TSV interconnects between the memory and logic die is challenging, as both memory dies and logic dies might come from different manufacturers. Currently, extended versions of two 2D standards might be applicable to test these interconnects. The first (*Boundary Scan Based*) method extends JTAG in which Boundary Scan Cells (BSCs) are placed on both TSVs ends providing full TSV controllability and observability to the TSVs. The second (*Logic Based*) method that can be applied is an extended form of the IEEE 1581 standard. In this standard, interconnects are tested by bypassing the memory. In the test mode, memory outputs are a direct logic function of the inputs. Both methods, however, result in extra area overhead, inflexible, and fail to address dynamic and time-critical faults (at speed testing). In addition, memory vendors have been reluctant to put JTAG or additional DfT structures on their memory devices.
In our *Memory Based Interconnect Testing* (MBIT) approach, we perform a post-bond memory based test where we test the interconnects by converting the developed test patterns to memory read and write operations. This method results in (1) zero area overhead, (2) the ability to detect both static fault and dynamic faults, (3) at speed testing, and (4) flexibility in applying the test patterns, as this can be executed by the CPU on the logic die.

CE-MS-2013-06

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Interconnect Testing for 3D Stacked Memories

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Mahmoud Saleh Masadeh
born in Kitim, Irbid, Jordan

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Interconnect Testing for 3D Stacked Memories

by Mahmoud Saleh Masadeh

Abstract

Three-dimensional stacked ICs (3D-SICs) technology based on Through-Silicon Vias (*TSVs*) provides numerous advantages as compared to traditional 2D-ICs. TSVs are holes going vertically through the chip silicon substrate filled with a conducting material. A potential application is a 3D-SIC where one or more memory dies are stacked on a logic die; thereby increasing the memory density, enhancing its throughput, and reducing its latency and power consumption as compared to planar ICs.

However, testing the TSV interconnects between the memory and logic die is challenging, as both memory dies and logic dies might come from different manufacturers. Currently, extended versions of two 2D standards might be applicable to test these interconnects. The first (*Boundary Scan Based*) method extends JTAG in which Boundary Scan Cells (BSCs) are placed on both TSVs ends providing full TSV controllability and observability to the TSVs. The second (*Logic Based*) method that can be applied is an extended form of the IEEE 1581 standard. In this standard, interconnects are tested by bypassing the memory. In the test mode, memory outputs are a direct logic function of the inputs. Both methods, however, result in extra area overhead, inflexible, and fail to address dynamic and time-critical faults (at speed testing). In addition, memory vendors have been reluctant to put JTAG or additional DfT structures on their memory devices.

In our *Memory Based Interconnect Testing* (MBIT) approach, we perform a post-bond memory based test where we test the interconnects by converting the developed test patterns to memory read and write operations. This method results in (1) zero area overhead, (2) the ability to detect both static fault and dynamic faults, (3) at speed testing, and (4) flexibility in applying the test patterns, as this can be executed by the CPU on the logic die.

Laboratory : Computer Engineering
Codenumber : CE-MS-2013-06

Committee Members :

Advisor:	Dr.ir. Said Hamdioui, CE, TU Delft
Chairperson:	Dr.ir. Koen Bertels, CE, TU Delft
Member:	Dr.ir. Nick van der Meijs, CAS, TU Delft
Member:	Dr.ir. Zaid Alars, CE, TU Delft
Member:	Ir. Mottaqiallah Taouil, CE, TU Delft

To my parents. To my wife Huda. To my cherished kids Yamin and Nada.

Contents

List of Figures

List of Acronyms

2D Two dimensional

3D Three dimensional

ATE Automatic Test Equipment

ATPG Automatic Test Pattern Generation

BEOL Back-end-of-line

BIST Built-In-Self-Test

BSC Boundary-Scan Cell

BSR Boundary-Scan Register

CMOS Complementary Metal Oxide Semiconductor

COB Chips-on-Board

CTE Coefficients of Thermal Expansion

DRAM Dynamic Random Access Memory

EDA Electronic Design Automation

EM Electromigration

FEOL Front-end-of-line

ITRS International Technology Roadmap for Semiconductors

KGD Known-Good-Die

MAF Maximum Aggressor Fault

MBIT Memory Based Interconnect Testing

MCM Multi-Chip-Module

MCP Multi-Chip-Package

MOS Metal Oxide Semiconductor

PCB Printed-Circuit-Board

SCITT Static Component Interconnect Test Technology

SI Signal Integrity

SIC Stack Integrated Circuit

SiP System-in-Package

SMT Surface Mount Technology

SoC System-on-Chip

SRAM Synchronous Random Access Memory

TAM Test Access Mechanism

TSOF Transistor Stuck Open Fault

TSSF Transistor Stuck Short Fault

TSV Through Silicon VIA

Acknowledgements

I am particularly grateful to *Mottaqiallah Taouil* for his precious advices during my master study, for the many and long discussions we had, and for his contributions to the thesis especially for endless revisions :) .

A special thank goes to my professor, *Dr.Said Hamdioui*, for giving me the opportunity to know and learn about the merging technology of 3D and Testing and his unique style of learning.

I would like to acknowledge my friends (*Emanuel*, *Anh*, *Amora*, and *Soran*) who companied me during my master study, special thank goes to Emanuel for all the courses we studied and passed together.

Last but not least, my wife deserves all the credit for this accomplishment, as she supported me every step of the way. Thank you for your unconditional love and endless support. *Huda*, you are the best wife one can ever wish for. Thank you for always being with me. It is your encouragement that makes me finish my master degree.

Mahmoud Saleh Masadeh
Delft, The Netherlands
August 20, 2013

1 Introduction

The successive generations of smaller technology nodes driven by Moore's law led to chips with increased transistor density. Scaling down into sub 28nm technologies resulted in several issues, such as [25, 26]: (1) the exponential lithography cost increase with scaling becomes economically impractical beyond a certain pitch, (2) power dissipation budget limits the clock frequency, and (3) increase in dynamic power dissipation as the interconnect length is increasing relatively to reduced feature sizes. 3D integration is one of the potential candidates that can be used to alleviate these problems [27]. Figure 1.1 depicts a general structure of a three-dimensional stacked ICs (3D-SICs) where a memory is stacked on logic (CPU). The connection between both dies go through TSVs.

3D-SICs based on TSVs are one of the promising solutions to alleviate the aforementioned problems. Through Silicon Vias (*TSVs*) are vertical electrical connections between stacked dies that go through the silicon substrate and are filled with a conducting material such as copper or tungsten. These short vertical interconnects (TSVs) have several advantages [27, 28, 3, 29, 30] such as:

- Reduced power consumption due to reduced wire length as the wires between components do not have to travel across large chip area's.
- Short global interconnect delay due to short vertical interconnects.
- High communication bandwidth is achievable, as the vertical interconnects between dies are perpendicular to the area surface and not only on the perimeter of the chip.
- Improved form factor.

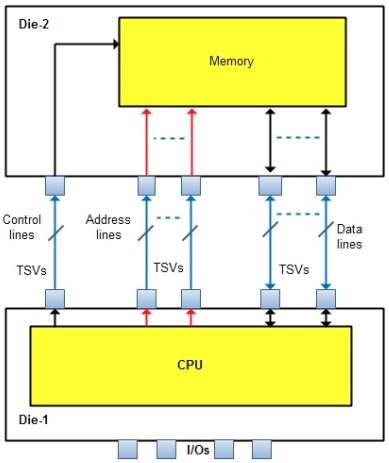


Figure 1.1: General structure for 3D stacked IC (Memory-on-Logic)

- Heterogeneous die integration, as each die can be manufactured in a different technologies and optimized for specific requirements such as area (DRAM) and speed (CPU).

3D-stacking is an emerging field with several challenges that require more research, such as (1) heat dissipation in stacked dies may negatively reduce the reliability of IC [31], (2) the TSV size is relatively huge compared to transistors and may lead to area overhead [2], and (3) testing 3D-SICs are challenging from a quality and cost point perspective [32, 33].

1.1 Project Goals

Figure 1.2 shows the scope of this thesis. It depicts different digital IC types for both 2D and 3D. In 2D ICs, usually dies are either optimized for speed (CPU+SRAM) or area (DRAM) as combing them is expensive [12, 50]. However, due to the possibility of heterogeneous stacking, a good candidate of 3D-SICs is to stack memories on logic which are among the first 3D-SICs to enter the market [30]. One of the main challenges is to test interconnects between such dies. The reason is that multiple dies, likely from different manufacturers, are stacked on each other.

The objective of this thesis is to develop a new memory based methodology for interconnect testing in 3D stacked memories which has overall similar or better capabilities as the state-or-the-art methodologies; for example it should: (1) be ignorant to the internal implementation details of the stacked dies due to Intellectual property (IP) constraints, (2) be able to detect both static faults and dynamic faults, (3) perform at-speed testing, (4) perform testing with minimum time and area overhead, , and (5) be scalable with the number of interconnects independent of the technology.

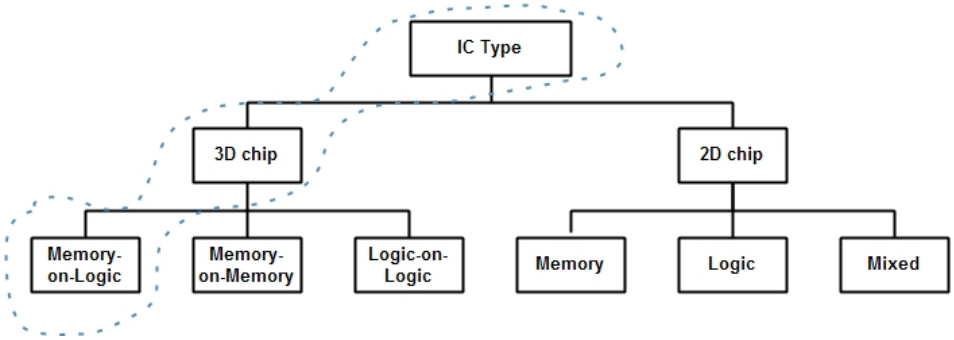


Figure 1.2: Thesis Scope

1.2 State-of-the-Art

Different research investigated the testability of 3D-SICs. For example, the authors in in [34, 35, 36, 37, 20] present different test approaches for 3D-SICs.

In [34], the authors studied the construction of a scan chain for 3D-SICs using different approaches. In [35, 36], the authors presented an optimized technique to minimize the test time for 3D core-based SOCs. In [20], the author(s) proposed a modular DfT test access architecture for 3D-SICs. In their approach, different components can be tested at several test phases including pre-bond, mid-bond, and post-bond test phase. The proposed approach allows testing several modules separately, such as separate dies, TSV-interconnects between dies, and external I/Os.

IEEE 1581 (Test Logic) [40, 23] is targeting the detection of static fault for interconnects in 2D-ICs. The primary focus is complex memories such as Flash, SDRAM, and DDR-SDRAM. In test mode, interconnects are tested by bypassing the memory. During test, the outputs of the memory are a logic function of its inputs. The standard is JTAG compliant, i.e., the test logic can function with a CPU that has JTAG.

Currently, standards for 3D-SICs testing do not exist yet. However, they are under development and many of these developed features of [20] are on-going activities in the IEEE P1838 working group [38, 22, 39].The state-of-the-art in 3D-SIC testing is based on the presence of JTAG in all dies. However, memory vendors are not in favor of integrating JTAG on their devices [23]. Recently, JEDEC which is the global leader in developing open standards for the microelectronics industry, has released a standard for Wide-I/O Mobile DRAMs which specifies the interface of memory stacked on logic [30]. They support JTAG. Moreover, IEEE 1581 approach may be extended for 3D-SICs where the bottom (logic) die will be JTAG compliant and where the top die (memory) will contain the test logic.

1.3 Main Thesis Contributions

The following contributions can be assigned to this thesis:

- An overview of the manufacturing steps of 3D-SICs is provided.
- An overview of the granularity partitioning of memory stacked on logic is provided.
- Research has been performed in identifying defects in 3D-SICs; these defects are classified based on their location in the stack.
- Classified fault models for 2D-ICs are extended to 3D-SICs.
- Test patterns, both for static and dynamic faults are introduced to test for defects in TSV arrays.
- The Development of a Memory Based Interconnect Test (MBIT) methodology that detects both static and dynamic faults in 3D-SICs interconnects for memory stacked on logic.
- The evaluation of the MBIT approach using MIPS simulator.
- Comparison of MBIT with state-of-the-art solutions.
- A paper will be submitted to the DATE 2013 conference.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 introduces the basics of 3D stacking. In this chapter the different 3D stacking steps are explained. In addition, partitioning granularities are provided that allow planar-memories to be mapped differently into the third dimensional. Chapter 3 introduces the failure mechanisms for both 2D and 3D ICs. It explains also terminology that is related to IC reliability and quality. The defects in 3D-SICs have been classified based on their locations, and the fault models in 2D-ICs have been extended for 3D-SICs. Chapter 4 explains the main idea of interconnects testing in memory stacked on logic by performing read and write operations. The detections conditions for the targeted fault models have been developed, and subsequently converted to test patterns. Chapter 5 discusses the experimental results of a study case where the MBIT is implemented in MIPS and compares them with the state-of-the-art approaches in this field. Each methodology is explained and examined on general defined interconnect test requirements. Chapter 6 summarizes this thesis and presents the future work.

2 3D Memory

This chapter addresses some basics in 3D-stacked ICs and focus mainly on memories. Section 2.1 introduces the 3D-stacking concept. Section 2.2 discusses different 3D ICs manufacturing processes. Section 2.3 explains the functionality and implementation of 2D memories. Section 2.4 describes how to extend the 2D memories into 3D, with different partitioning granularities. Finally, Section 2.5 summarizes the chapter.

2.1 3D ICs: The Concept

Figure 2.1 shows the main forms of 3D integration over time. Figure 2.1(a) shows a Printed Circuit Board (PCB) in which multiple heterogeneous ICs are integrated. Wires are used for the communication between ICs, relatively with low performance and high power consumption. In Figure 2.1(b) a Multi-Chip Package (MCP) is shown. Here multiple heterogeneous dies are placed close to each other in a single package.

Figure 2.1(c) shows a System-in-Package (SiP); the IC exploits the vertical dimension and multiple naked dies are stacked vertically in a single package. Therefore, it reduces the footprint. Communication between dies in both MCP and SiP is through wiring. SiP is widely used in mobile devices due to its small footprint compared to PCB and MCP. The last configuration, the 3D-SIC, shown in Figure 2.1(d) is similar to SiP in its concept. Moreover, here the interconnections go through the silicon of the dies instead of using external wires. TSVs are smaller in size as compared to wire bonding. Relatively speaking, wire bonding is slow, and energy-inefficient [1]. TSVs on the other hand have a much higher interconnection density, although at higher manufacturing cost.

3D integration is considered as *More than Moore's* with several advantages over planar ICs, such as high speed, less power consumption, small form factor, and heterogeneous integration [27, 28, 3, 29]. In addition interconnect delay is a limiting factor in today's IC's performance, and short interconnections in 3D-ICs can alleviate this problem.

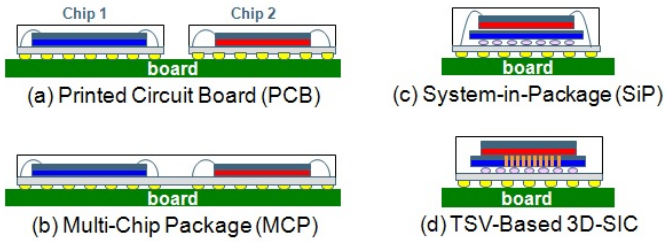


Figure 2.1: 3D integration forms [1]

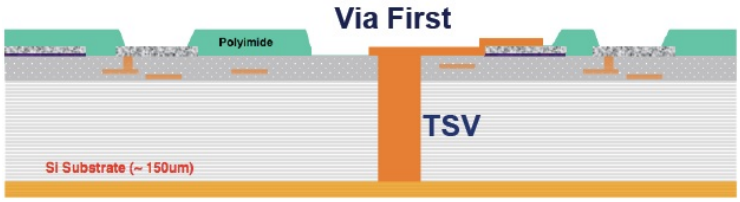


Figure 2.2: TSV-formation: Via-First [2]

2.2 3D ICs: Manufacturing Process

3D-SIC manufacturing process is performed by stacking of planar dies in the vertical dimension. Each die can be designed and manufactured simultaneously using different fabrication labs; therefore it can also be called *parallel process*. The 3D stacking process includes three main steps: (1) TSV formation, (2) wafer thinning, and (3) wafer or die bonding. 3D ICs manufacturing process has different approaches depending on TSV integration scheme, bonding types, and stacking process [29, 12, 3]. We will discuss each of these options.

2.2.1 TSV Formation

TSVs can be classified into three groups, based on their time of manufacturing, during IC fabrication process; (1) before FEOL (transistor) formation, and called via-first, (2) after FEOL (transistor) formation and before BEOL (metal layer) formation, and called via-middle, (3) after IC fabrication process and this is called via-last which has different options shown below. These three classes are described next, where they have different TSVs size, conductivity, and filling material.

1. **Via-First**

   Via-first approach has the TSVs fabricated along with the fabrication of active circuitry and before thinning, dicing, and assembly. The filling material is polysilicon, because the copper is not a good choice for high temperature used in front-end CMOS. Via-first allows the use of high thermal budget materials for high voltage applications [41]. TSV diameter is about 1μm-5μm [42] as shown in Figure 2.2.

   Via-first has many advantages. First, TSVs will be visible after thinning, and can be used to align the masks for additional backside process step. Moreover, Via-first approach will minimize the thin wafer handling and processing steps. Via-first has disadvantages such as; the manufacturing cost is high and requires adding constraints on design rules of transistor scaling. Also, it is better with wafer stacking only, because thin die handling is costly [3].

2. **Via-Middle**

   Via-middle approach has the TSVs fabricated after the fabrication of active circuitry and before metal layers deposition, thinning, dicing, and assembly. The filling material may be copper, or polysilicon.

Figure 2.3: TSV-formation: Via-Last [2]

3. **Via-Last**

   Via-last TSV formation approach has the TSVs fabricated after the fabrication of active circuitry and back-end layers and before dicing and assembly. The filling material is copper. This approach has many advantages. First, the cost of Via-last manufacturing is potentially lower than other methods, and can be used for both wafer and die stacking. Also, TSVs fabrication after fabricating the active circuitry will avoid conflict with standard process flow. Moreover, Via-last reduces overall manufacturing cost, because it does not require expensive equipments. On the other hand, TSV last goes through all metal layers in addition to device and substrate, and this is considered as a routing obstacle [42]. In addition, the TSV diameter is large, which is around 5μm-20μm as shown in Figure 2.3, where large TSV suffers from large pitch, large Keep-out-zone, low speed, and high power [42].

2.2.2 Wafer Thining

3D integration demands wafer thinning to a thickness around 100μm. Handling such thin wafer with high diameter is challenging. Therefore, wafers are mounted on temporary handle wafers (carrier wafers). On one hand, the IC wafer is mounted face-down into the handling wafer, and bonded to 3D IC stack after thinning as shown in Figure 2.4(b), on the other hand, the IC wafer can be bonded directly to the 3D IC stack, then thinned where the handling wafer is not needed as shown in Figure 2.4(a) [3]. Throughout this thesis, the terms chip and die are interchangeable.

2.2.3 Wafer or Die Bonding

Wafer or die bonding, has different options regarding bonding type and orientation, each explained below.

Bonding Types

There are three different stacking methods to manufacture 3D-SICs, as shown in Figure 2.5; (1) Wafer-to-Wafer, (2) Die-to-Wafer, and (3) Die-to-Die [3, 43]. Each bonding method has its advantages and disadvantages. Each method is explained next.

1. **Wafer-to-Wafer (W2W)**

   In W2W stacking, entire wafers of dies are stacked on top of each another. This approach has several advantages, such as high throughput as dies on the wafers are processed simultaneously. In addition, bonding a complete wafer makes it possible

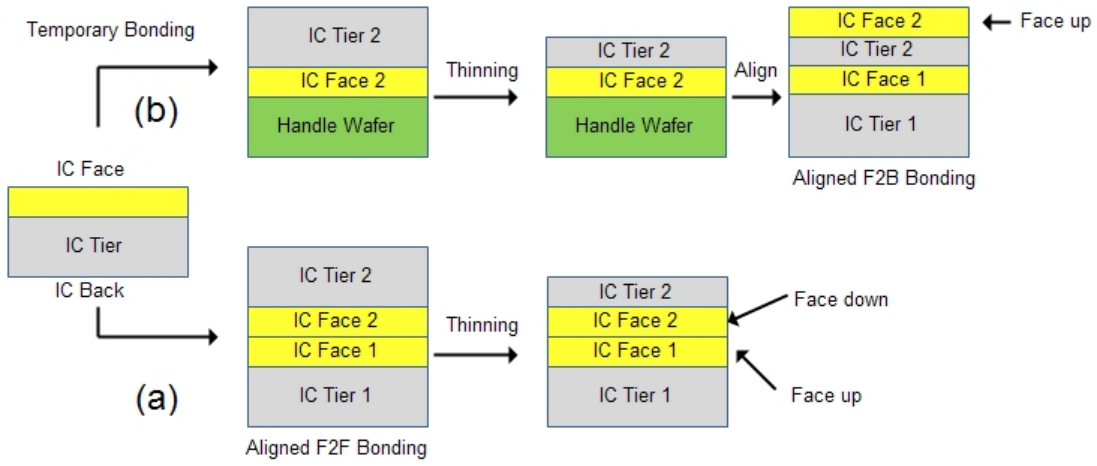


Figure 2.4: Wafer thinning (a) direct after bonding, (b) using 'handle wafer' before bonding [3]

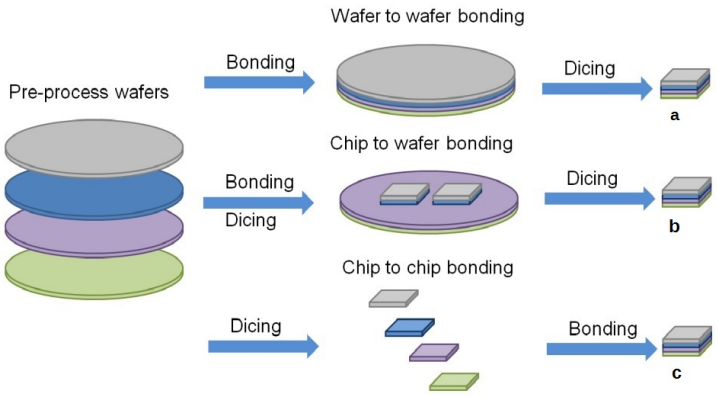


Figure 2.5: Bonding types [4]

to handle small dies [44]. This approach has also disadvantages. W2W stacking requires dies to be of the same size [3]. Nevertheless, candidates for this approach are memories due to their regularity. DRAM memory stacking is based on this method because it provides increased production speed and throughput [43]. A second drawback of this approach is the compound yield, as there is no flexibility to stack individual dies. The method compound yield reduces exponentially with stack size [3]. Figure 2.5(a) shows an example flow of W2W bonding for four wafers.

2. **Die-to-Wafer (D2W)**

   In D2W stacking, individual dies of the top wafer are stacked on a bottom wafer as depicted in Figure 2.5(b). This method has several advantages such as: the manufacturing throughput is not as good as in W2W stacking, but it allows Know-Good-Die (KGD) stacking. In addition, dies with different dimensions can be stacked, as long as the top die is smaller than or equal to the bottom die. Therefore,

Table 2.1: 3D Bonding Types

Criteria	W2W	D2W	D2D
Different Wafer size	Same size	No limitation	No limitation
Different chip size	Same size	No limitation	No limitation
Different Fabs	Less likely	Likely	Most likely
Number of Stacked layers	Many	Few	Few
Modular Design	Enabled	Enabled	Enabled
Yield	Lowest	High	Highest
Throughput	Highest	Moderate	Lowest

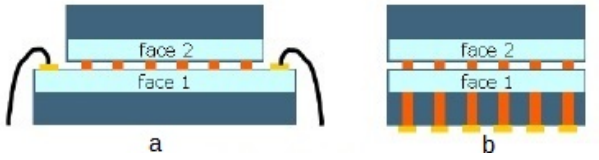


Figure 2.6: Face-to-Face [5]

this approach is the most favorable one from yield point of view where it has a yield similar to D2D [44]; and is the most relevant from an industry point of view.

3. **Die-to-Die (D2D)**

   In D2D bonding, dies are stacked individually after both top and bottom wafers are diced, as depicted in Figure 2.5(c). This bonding method is considered as the most flexible one. A high compound yield can be obtained in case Known-Good-Dies (KGD) are stacked. The throughput will be less than other methods [44], as aligning required for individual dies. Moreover, small dies are hard to handle. Therefore it is less preferred than other methods in industry [43].

Table 2.1 summarizes the main criteria's for different types of bonding.

Bonding Orientation

3D stacking can be implemented based on wafer and/or die stacking orientation. There are three different approaches. Each stacking orientation differs in its fabrication processes and its inter-die via requirements. These three types of stacking are distinguished based on the face and back side of the stacked dies. The face side is the side of die where the metal layers reside, and the back side is the side of the die where the substrate resides [45, 46]. The three possible orientations are: (1) F2F, where the face of one die is stacked on the face of the other die, (2) F2B, where the face of one die is stacked on the bottom of the other die, and (3) B2B, where the back side of one die is stacked on the back side of the other die. Each of them is described next.

1. **Face-to-Face (F2F)**

   In F2F, the face side of the bottom die is stacked with the face side of the top die. Since the active sides for stacked dies are connected directly, F2F is the sim-

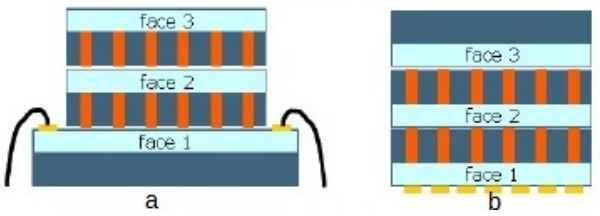


Figure 2.7: Face-to-Back [5]

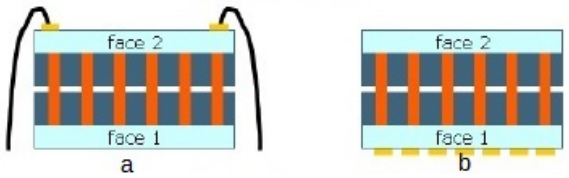


Figure 2.8: Back-to-Back [5]

plest process for 3D integration, and requires minimum changes in the fabrication process [47]. Figure 2.6 shows the concept of F2F stacking. The external communication to the outside occurs either through TSV or wire bonds.

Wire-bonding requires some extra space, therefore the bottom die have to be slightly larger than the top one as shown in Figure 2.6(a). If the external communication goes through flip-chip bumps, TSVs are needed as depicted in Figure 2.6(b). The TSVs are used to pass data through the silicon bulk to I/O's of the chip. Note that this configuration requires die thinning of the bottom die, to expose the TSVs to the I/O pads.

F2F is not scalable to stacks consisting of more than two dies [32, 5, 3]. Therefore, if additional layers need to be stacked, either F2B or B2B must be used. These schemes are explained next.

2. **Face-to-Back (F2B)**

   In F2B, the face side of bottom die is stacked with the bottom side of the top die. This is depicted in Figure 2.7. F2B stacking may have all dies in the stack faced-up, in which external communication is done through wire bonding. The bottom die must be slightly larger than to provider wiring space as shown in Figure 2.7(a). F2B stacking may have all dies in the stack faced-down. Here, the external communication goes through flip-chip bonds at the bottom die as shown in Figure 2.7(b). F2B is scalable, where an arbitrary number of die layers can be stacked [32, 3].

3. **Back-to-Back (B2B)**

   In B2B, the back sides of both dies are stacked together. Both stacked dies require TSVs to form the interconnects. This is depicted in Figure 2.8. Hence, being more expensive than F2F and F2B. The external communication goes either through wire bonds if connected from the top die as shown in Figure 2.8(a), or through

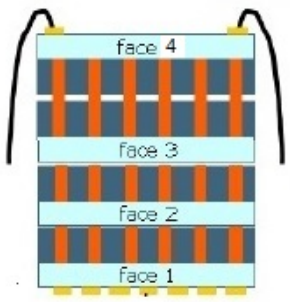


Figure 2.9: Mixed stacking orientation

Table 2.2: 3D integration processing sequence [3]

Process	IC Wafer	Step#1	Step#2	Step#3
A	FEOL TSV (vias first)	Wafer thinning (on handle)	Face-up bond (metal bonding)	—
B	FEOL TSV (vias first)	Face-down bond (metal bonding)	Wafer thinning (on 3D stack)	—
C	BEOL TSV (vias middle)	Wafer thinning (on handle)	Face-up bond (metal bonding)	—
D	BEOL TSV (vias middle)	Face-down bond (metal bonding)	Wafer thinning (on 3D stack)	—
E	No TSV	TSV from front (vias last)	Face-down bond (metal bonding)	Wafer thinning
F	No TSV	TSV from front (vias last)	Wafer thinning (on handle)	Face-up bond (metal bonding)
G	No TSV	Face-down bond (all methods)	Wafer thinning (on 3D stack)	TSV from back (vias last)
H	No TSV	Wafer thinning (on handle)	Face-up bond (all methods)	TSV from front (vias last)
I	No TSV	Wafer thinning (on handle)	TSV from back (vias last)	Face-up bond (metal bonding)

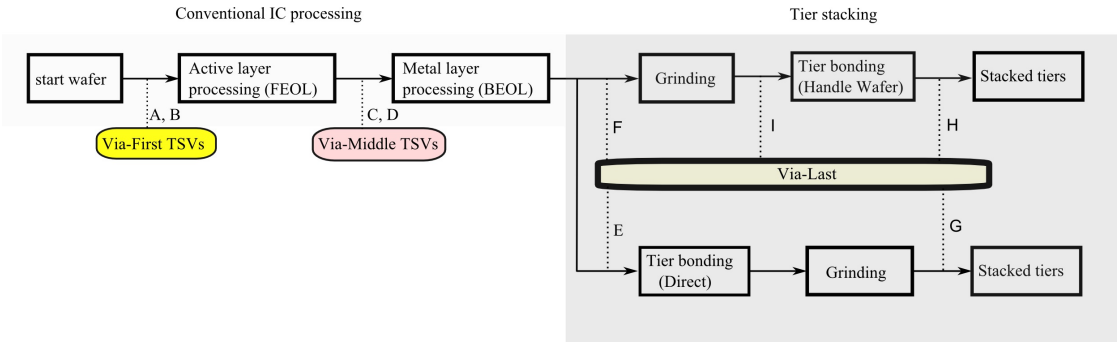


Figure 2.10: 3D integration processing sequence

flip-chip bumps connected at the bottom die as shown in Figure 2.8(b). Like in F2F, B2B is not scalable to stacks consisting of more than two dies [32, 5, 3].

4. **Other combinations**

Dies can be mixed with any combination of the previous orientation. For example, stacking depicted in Figure 2.9 have four dies; where dies 1 and 2 are stacked in B2F fashion, dies 2 and 3 are stacked in B2F fashion, while die 3 and 4 stacked in B2B fashion.

2.2.4 Process Sequence for 3D Integration

The main steps for 3D manufacturing process based on TSV-formation could be performed in different orders and thus it made trade-offs. Different flows for 3D integration process according to the sequence of executed steps are summarized in Table 2.2 and depicted in Figure 2.10.

Process A and B are considered via-first, where the first step is bulk etching and TSV-metal deposition then transistor formation (FEOL) and metal deposition (BEOL) through lithographic process. The second step for process A requires flipping the wafer and bonding it to handle wafer in order to thin down the silicon until TSV tips are visible from the backside. The third step is bonding the thinned dies. Process B has the bonding and thinning steps in reverse order compared to process A; the second steps is bonding the dies while the third is thinning the wafer on 3D stack. Process C and D and considered via-middle and are similar to process A and B respectively except that the TSV formation in the first step is performed between transistor formation (FEOL) and deposition of metal (BEOL) layers.

Process E and F are via last, where the first step is TSV formation after transistor formation (FEOL) and metal deposition (BEOL). Process E involves metal bonding then wafer thinning, while process F involves wafer thinning using handle wafer then bonding.

The first step in process G is die bonding, then wafer thinning on the 3D stack, and the last step is TSV-formation based on via-last. The first two steps are reversed for process H compared to process G; it involves wafer thinning using handle wafer then bonding and the last step is TSV-formation based on via-last. Both process G and H are *via-after-bonding* since via formation is after bonding. Process I involves wafer thinning using handle wafer, then TSV-formation based on via-last, and the last step is metal bonding.

2.3 2D Memory

In the previous section, we explained the different 3D ICs manufacturing processes. This section discusses the functionality and implementation of 2D memories. Section 2.3.1 introduces the general modeling hierarchy used to describe the various levels of memory abstraction. Subsequent, Sections 2.3.2, 2.3.3, and 2.3.4 shows the behavioral model, functional model, and electrical model for memories, respectively. Finally, Section 2.3.5 explains the layout model and process technology.

2.3.1 Modeling of Memory

Random access memories (RAMs) which are depicted in Figure 2.11, are able to access any piece of data independent to its physical location with constant access time. It can be divided into two types: (1) read-write memories (*RAM*), and (2) read-only memories (*ROM*). RAMs which will be our focus in this thesis, include static RAMs (SRAMs) and dynamic RAMs (DRAMs). ROMs have different types: (1) masked ROMs, (2) one time programmable ROMs (OTP ROMs), (3) *erasable programmable ROMs (EPROMs)*, (4)

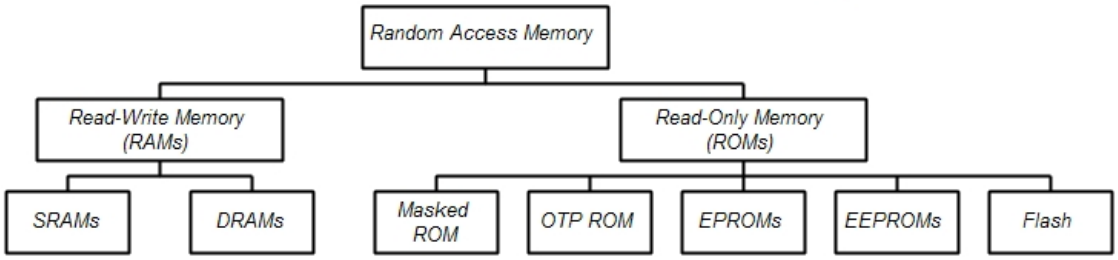


Figure 2.11: Memory classification

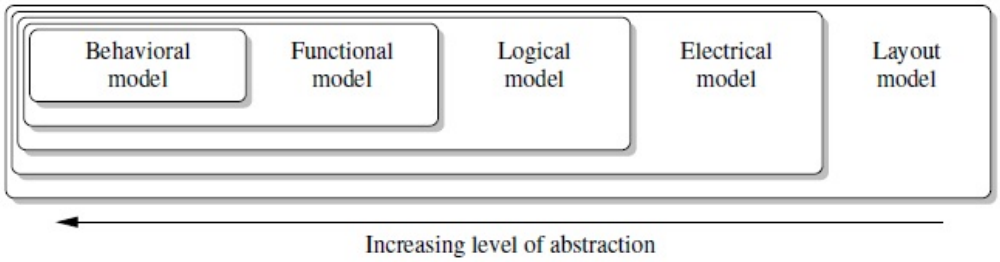


Figure 2.12: Models and levels for ICs representation

electrically erasable programmable ROMS (EEPROMs), and (5) flash memories. Next, different models for both SRAMs and DRAMs will be explained.

This section describes SRAM and DRAM memories using a layered modeling approach commonly used to describe modern and complex IC systems. The abstraction levels of this modeling approach are shown in Figure 2.12 [8, 6, 7].

Modeling is the simplification and structuring of an entity and its environment. *Models* are used to describe only the relevant phenomena such as; events, properties, and changes [7]. Each modeling level in Figure 2.12 is called an abstraction level. In the figure, a model represented by a larger block has a lower level of abstraction than a model represented by a smaller block. A higher level of abstraction contains more explicit information about the way a system is expected to function and less about its construction.

As we move from the layout model (lowest level of abstraction) toward the behavioral model (highest level of abstraction) in Figure 2.12, the models become less representative of the physical system and go toward a description of how system behaves. In other words, the models become less physical and more abstract. The lowest level is represented by the largest block in the figure and is the most closely related to the actual physical system.

It is possible to represent the system with a model that contains components from different levels of abstraction; this approach is referred to as *Mixed-Level Modeling*. With mixed-level modeling, one may focus on low-level details in certain areas of interest, while maintaining high-level models for the rest of the system. The modeling levels shown in Figure 2.12 will be described briefly next.

- *The Behavioral Model*: This model is the highest modeling level and is based on

the specifications of the system. The only information given is the relation between input and output signals while the internal system is considered as a black box. At this level, there is practically no information given about the internal structure of the system or possible implementations of the performed functions. A model at this level usually makes use of timing diagrams to convey information about the system behavior. The behavioral model specifically for memories is described in Section 2.3.2.

- *The Functional Model*: This model defines the system in functions. The system is divided into several interacting subsystems each with a specific function. Each subsystem is basically a black box called a functional block with its own behavioral model. The internal signals of the system are partially visible, and this model is referred to as gray-box Model. The collective operations of the functional blocks result in the proper operation of the system as a whole. The functional memory model is described in Section 2.3.3.

- *The logical Model*: This model presents the memory in terms of logic gates. At this level, simple boolean relations are used to describe the system. It is not very common to model memories exclusively using logic gates, although logic gates are often present in models of a higher or lower level of abstraction to serve special purposes. Therefore, no exclusive memory logical model is given in this chapter for this abstraction level.

- *The Electrical Model*: This model describes the memory system in terms of basic electrical components. The components are mostly transistors, resistors and capacitors. The internal structure of the system is completely visible, and this model is referred to as white-box/glass-box Model. At this level, we are not only concerned with the logical interpretation of an electrical signals but also with the actual electrical values of it (e.g., voltage or current levels). This memory model is presented in Section 2.3.4.

- *The Layout Model*: This model presents the actual physical implementation of the system. At this level, all aspects of the system are taken into consideration; even the geometrical configuration plays a role, such as the length and thickness of signal lines that is called design rule set. Section 2.3.5 briefly discusses this model and explains the flow to obtain the physical implementation from the layout model.

Taking a closer look at the behavioral and the functional models reveals that there is a strong correspondence between the two. In fact, the behavioral model can be treated as a special case of the functional model, with the condition that only one function is presented, namely the function of the system as a whole. Therefore, some authors prefer to classify both modeling schemes as special cases of a more general model called the structural model. The structural model describes a system as a number of connected functional blocks. According to this definition, a behavioral model is a structural model with only one function, while a functional model is a structural model with more than one function [8, 6].

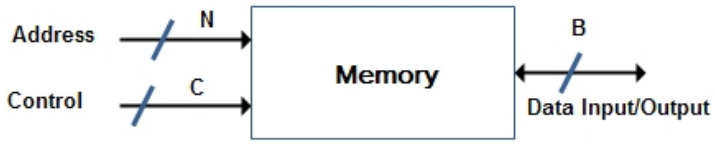


Figure 2.13: Memory block diagram

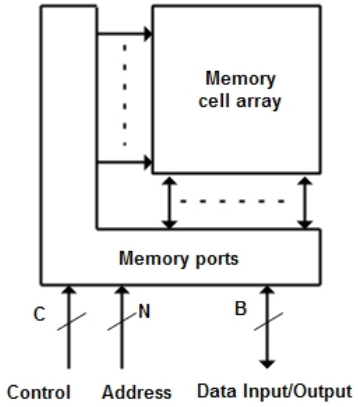


Figure 2.14: Memory main components

2.3.2 Behavioral Model

The behavioral model treats the memory as black box, and its focus lies on input and output signals and their timing specification. Therefore, it describes the external behavior of the memory; the internal implementation details and structure are not considered. Timing diagrams are used to describe the memory operations such as reading and writing to memory cells. Each operation adheres to a specific timing requirement.

The block diagram in Figure 2.13 shows an example memory, where the memory has two inputs and one bi-directional I/O bus. The input *Address* of width N specifies the address of each cell. The control signals *Control* of width C specify the operation type (e.g, read/write). The bi-directional line of the memory *Data Input/Output* of width B contains the cell values of the accessed cells for reading or writing.

To save pins memories usually have two types of multiplexing: (1) data multiplexing, where *Input-data* and *Output-data* are multiplexed to form bi-directional data lines as shown in Figure 2.13, and (2) address multiplexing.

When the memory includes a clock signal, read and write operation are synchronized according to this clock. The memory block diagram can be explained further by considering the main two components of memory as shown in Figure 2.14, these two components are: (1) the memory cell array used for data storage with a unique address for each storage element (memory cell), and (2) the memory ports which represents the real interface between the memory cell array and the external world that the memory is located in. SRAM and DRAM have some differences in the timing diagram for the read and write operations. Both are described next.

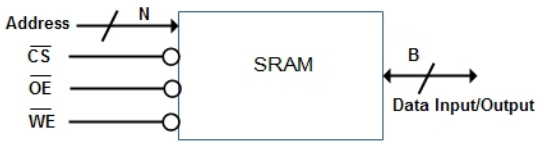


Figure 2.15: SRAM block diagram

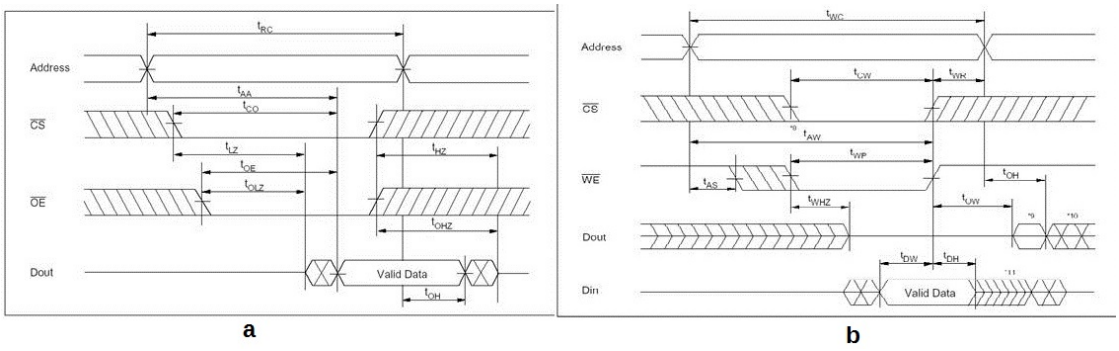


Figure 2.16: SRAM (a) Read operation, (b) Write operation timing diagram

Behavioral SRAM Model

An example of a behavioral SRAM model is shown in Figure 2.15. The memory has four inputs and one bi-directional I/O bus. The inputs are the (*Address*) with N bits, and three control signals; chip select ($\overline{CS}$), output enable ($\overline{OE}$), and write enable ($\overline{WE}$). $\overline{CS}$ selects the chip, and $\overline{OE}$ enables the output, and $\overline{WE}$ enable the chip to perform read or write operations. Note that these control signals are active low. The output is B bit bi-directional data bus.

Timing diagrams are used to depict the input and output signals of correct memory operation over time. The timing diagram shows the dependencies between memory signals, and the timing conditions for the operations to be performed properly. The read and write operation timing diagrams for SRAM are discussed next.

- Memory *read operations* are symbolized by **r0** and **r1**, where the value **0** or **1** represents the expected value to be read from memory. Figure 2.16(a) gives an example of an SRAM read operation. The read operation is initiated by its address; where the address must be valid for the period of t_{RC} (read cycle time). After that, the signals CS and OE must be defined. Valid data will appear on the data line after a period of t_{OE} (output enable access time), measured with respect to the high-to-low transition of the OE signal. The address t_{AA} (access time) is measured from the beginning of the valid address that appears on the address lines to the appearance of valid data on the data lines. The time, t_{CO}, measures the chip enable access time, which is the time for the valid data to appear after the high-to-low transition of the chip select signal CS.

- SRAM *write operation* is shown in Figure 2.16(b). Memory write cycle is initiated by it's address. The address needs to be stable for the specified duration of t_{WC}

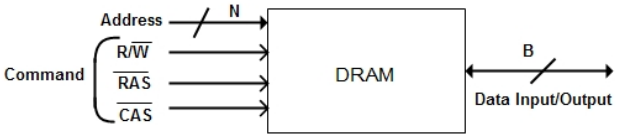


Figure 2.17: DRAM block diagram

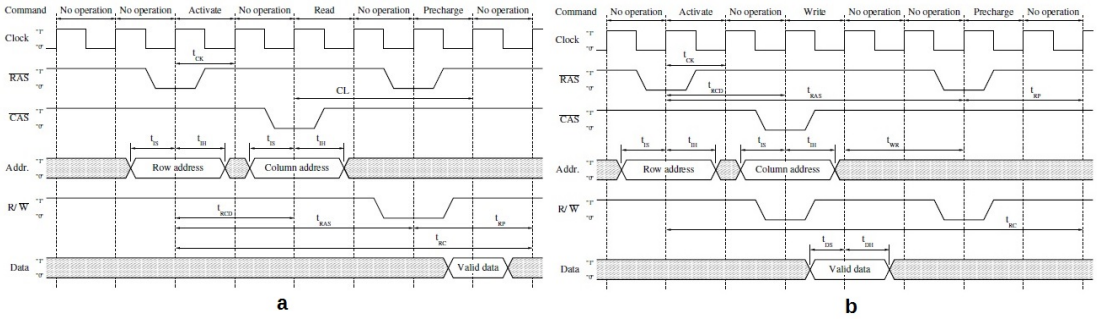


Figure 2.18: DRAM (a) Read operation, (b) Write operation timing diagram [6]

(write cycle time). After that, the CS and the WE signals are activated. The write enable signal WE is activated only after a minimum time t_{AS} (address setup time) measured from the beginning of the valid address. The time for which the WE signal remains active is known as the t_{WP} (write pulse width). After the WE signal becomes active, the data is written into the data lines. The WE signal must remain valid for a minimum time duration t_{DW} (data write) after data is applied at the data input lines. Once the WE signal is deactivated, the data must remain valid for a time, t_{DH} (data hold).

Behavioral DRAM Model

An example of a behavioral DRAM model is shown in Figure 2.17. The memory has four inputs and one bi-directional I/O bus. The command bus has three components: $\overline{RAS}$, $\overline{CAS}$, and R/$\overline{W}$ which are used to control the memory. The address is divided into two parts: row address and column address, and set up consecutively on the address pins.

The control signal row address strobe ($\overline{RAS}$)is used to indicate that the row address is ready on the address lines, and the column address strobe ($\overline{CAS}$) is used to indicate that the column address is ready on the column lines. The read and write operation timing diagrams for DRAM are discussed next.

- Memory *read operations* are symbolized by **r0** and **r1**, where the value **0** or **1** represents the expected value to be read from memory. Figure 2.18(a) gives an example for DRAM read operation with $t_{CK} = 10$ ns. On the top of the Figure the commands (Active, Read, No operation, and Precharge) represent the DRAM internal commands for each clock cycle.

  The first step in read operation is setting the row address (first half of the address) on the address bus for a time period with minimal value equal to t_{IS} (time input

setup) before the rising edge of the clock, so the address will be stable on the inputs. After the clock, the address should be held on the inputs for t_{IH} (input hold time) so the address on the address bus can be read properly. The user has to pull RAS down, and pull R/$\overline{W}$ up, so the memory will know that the address is ready on the inputs. A minimum period of time t_{RCD} (row-column delay time) should pass between providing the row address and providing the column address. $\overline{CAS}$ has to be pulled down to initiate the provided column address. To state that this is a read operation, R/$\overline{W}$ has to be pulled up at the same rising edge. After setting up the address (row address and column address) and issuing the read command, it needs a time period of CL ($\overline{CAS}$ latency) so that the data stored on the addressed cell will appear on data bus. Figure 2.18(a) have CL with a value equal $3 * t_{CK}$. The read operation ends by pulling down the signals that were pulled up at the beginning of read operation, both $\overline{RAS}$ and R/$\overline{W}$ must be pulled down.

The parameter t_{RAS} (row address strobe time) that is the maximum and minimum time between two RAS pulses for a single operation in memeory is used to determine the length of the whole read operation. Consecutive memory read operations have a minimum period of time t_{RP} (row precharge time) between them. Each read operation must have at least a period t_{RC} (row cycle time) to start another read operation.

- Memory *write operations* referred to as **w0** and **w1**, where the value **0** and **1** represents the value to be written by a successful write operation into specific memory cell. Figure 2.18(b) gives an example of an DRAM **write operation**. For memory write operation, the row address setup and cloumn address setup are similar to read operation with the same timing parameters. When the column address is on the address bus, memory write operation must be specified by pulling down the $R/\overline{W}$ signal. On the next clock cycle, the data to be written into memory cell should be on the data bus and left for t_{DS} (data setup time) to stabilize befor the rising edge of the clock, also should stay at the data bus for t_{DH} (data hold time) after the rising edge of the clock.
  Before ending the write operation (by pulling down $\overline{RAS}$ siganl), the memory needs enough time to write the required voltage into the cell and this minimum period is called t_{WR} (write recovery time). t_{WR} is also called write back window.

2.3.3 Functional Model

The memory functional model can be consisting of functions that interact with each other. The general memory functional model shown in Figure 2.19 includes SRAM and DRAM; for SRAM the refresh logic would be omitted. The main functional blocks are:

- *Memory cell array*: The memory cell array is arranged in an array of rows and columns, and used for data storage. It is considered the largest part in memory. For example, DRAM cell array occupies around 60% of the chip area [6, 48].

- *Address latch*: The address latch contains the input address, which will be divided into row and column part.

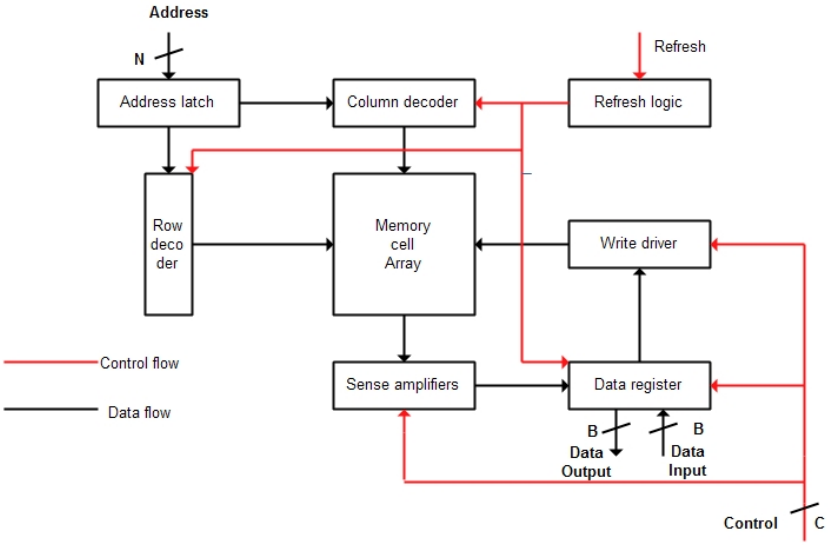


Figure 2.19: Memory functional block diagram [7]

- *Row address decoder*: The row address decoder selects an appropriate row (WL) in the memory cell array and takes the MSB bits of the address as input.

- *Column address decoder*: The column address decoder selects the required columns (BL) in the memory cell array and uses the LSB bits of the address as input. The number of columns selected depends on the chip data line width.

- *Sense amplifier*: The content of the selected cell during read operation are amplified by sense amplifier.

- *Data register*: The data register holds the read data during read operation from the sense amplifier, then present it on the data-output lines. During write operation, the data register holds the data from the data-input lines before written to the memory cell array through the write driver.

- *Refresh logic*: For DRAM memory, during refresh, the refresh logic disables the data register. The row decoder selects the row based on the content of the address latch. The column decoder selects all columns. All bits in the selected row are read and refreshed simultaneously.

2.3.4 Electrical Model

This section describes the electrical model of the memory. The basic building blocks of such a model include transistors, resistors and capacitors, etc. This model has many components similar in both SRAMs and DRAMs, which are explained below.

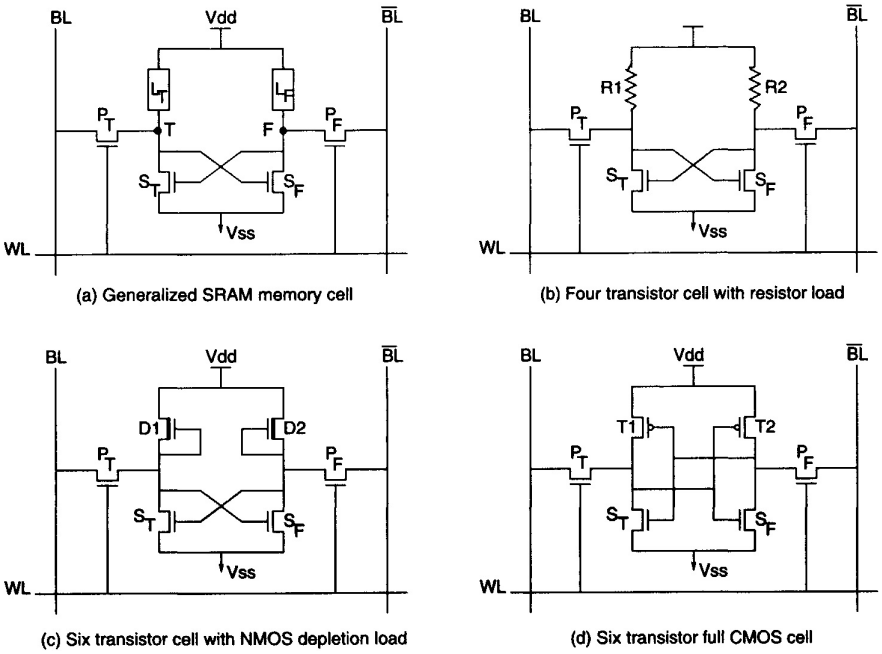


Figure 2.20: SRAM cell: (a) generalized cell, (b) resistive load, (c) NMOS load, and (d) PMOS load [8]

Electrical SRAM Model

The electrical properties of the functional model blocks will be explained. The main blocks are memory cells, address decoders, and Read/Write circuits.

- *SRAM memory cell*: The SRAM memory cell consists usually of a bi-stable flip-flop circuit that drives the cell into one of two possible states. Generally, the SRAM cell has three possible configurations shown in Figure 2.20.

  The generalized SRAM cell (Figure 2.20(a)) has two load elements (L_T and L_F), two pass transistors (P_T and P_F), and two storage elements (S_T and S_F). Transistor S_T and load element L_T together form an inverter. Similarly, S_F and load element L_F form also an inverter. Together, all four elements produce a latch. Transistors P_F and P_T are used to access the latch for read and write operation.

  The *read* operation is performed as follows: both BL and $\overline{BL}$ are pre-charged to a high level by the bit-line pre-charge, then the desired WL is selected, this allows the two access transistors (P_T and P_F) to pass the internal value (T) and its complement (F) on BL and $\overline{BL}$. The difference between the lines BL and $\overline{BL}$ is sensed and amplified. The SRAM read operation is non-destructive, so after the read operation is finished the memory cell content remains unchanged.

  To *write* a memory cell, a value must be placed on the BL, and its complement placed on $\overline{BL}$. By activating the word-line (WL), BL and $\overline{BL}$ form a direct connection with T and F. Applying the write value long enough will force the internal cell to the desired value.

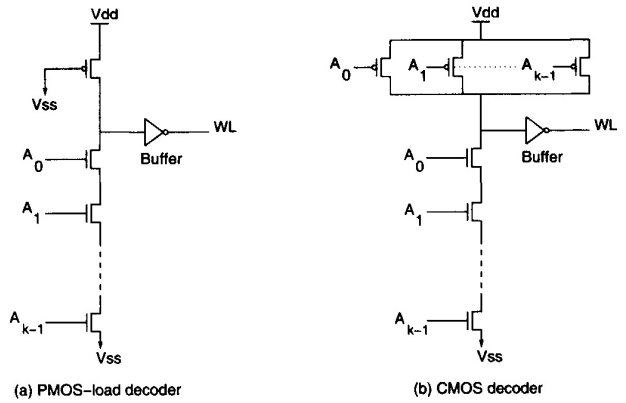


Figure 2.21: Static row decoders [8].

The generalized SRAM memory cell (Figure 2.20(a)) has two load elements, that can be implemented by NMOS transistors, or PMOS transistors. Figure 2.20(b) shows the implementation with resistors. This structure needs less area than the other designs but has a higher idle current because a small amount of current continuously flows through the resistors. Figure 2.20(c) uses an NMOS depletion load. This results in better switching performance, and less sensitive to variations in power supply. Figure 2.20(d) shows PMOS transistors as load devices. In this configuration, the static current is very low. However, it needs more processing steps compared to the other methods.

- *Row decoder*: The task of row decoder is to activate a single WL at a time. There are two types of static row decoders PMOS-load decoder and CMOS decoder. Both decoders have the address bits $A_0....A_{k-1}$ (or complement values) as inputs. In the PMOS-load decoder depicted in Figure 2.21(a), the address lines are connected to NMOS transistors only, while in the CMOS decoder depicted in Figure 2.21(b), the address lines are connected both to NMOS and PMOS transistors. The PMOS-load decoder has relatively a lower load delay as the load capacitance is lower, needs less area, but has a higher static current dissipation.

  To improve the performance of the static decoders, dynamic (clocked) row decoders are proposed as shown in Figure 2.22(a). It has almost zero static current consumption as the power is consumed only during the period of address transition (short period), also it has a compact layout.

- *Column decoder*: The column decoder is used to activate a single BL at a time. The column decoder selects B bit-lines out of the memory width. Figure 2.22(b) shows the column decoder based on PMOS load decoder. The decoder output goes to an inverter for amplification, and then to the column switches MOS transistors.

- *Write Circuitry*: Figure 2.23(a) shows a write circuit, where the Data-in value is forced on the BL and $\overline{Data-in}$ on $\overline{BL}$ in case Write is high. The circuit in Figure 2.23(b) uses a different structure, where two NAND gates are used.

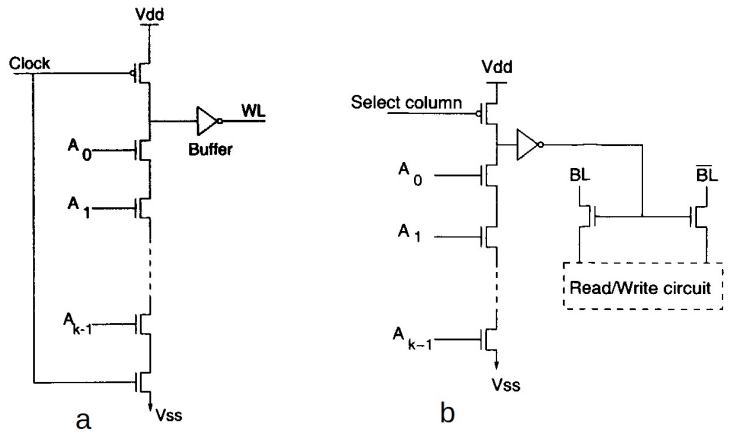


Figure 2.22: (a) Simple dynamic row decoder, and (b)PMOS based column decoder

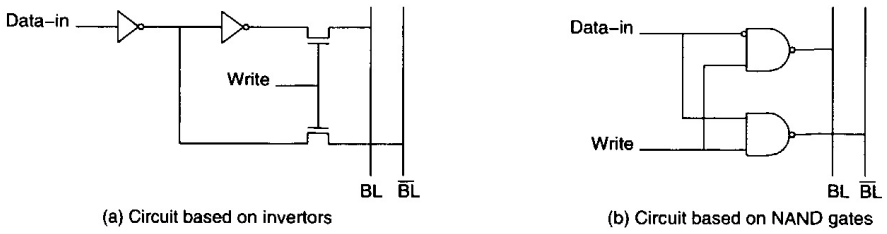


Figure 2.23: Memory writes circuites

- *Pre-charge circuit*: Before accessing the signal lines in the memory we have to set them to a given pre-defined voltage, and this is called pre-charging. The pre-charging process aims to ensure correct read operation. The pre-charge circuit is relatively easy to design. Figure 2.24 shows an implementation using three transistors. The *precharge* signal controls the three transistors and determines when the pre-charge action will happen. The signal 'Pre-charge' goes high before the read operation, so the three transistors work. Transistors T1 and T2 precharge both BL and BL to $V_{DD}/2$. Transistor T3 represents the balance (equalization transistor) and helps to speed up the pre charging process.

- *Sense amplifier*: The sense amplifier is a part of the read circuitry and used to amplify small signals on large capacitive bit-lines to a normal logic level. A simple inverter can be used, and in most cases a cross-coupled inverter is used for amplification. The sense amplifier can be voltage or current based, where the current mode sense amplifiers operate faster than the voltage mode sense amplifiers. When designing a sense amplifier, the aim is to have an amplifier that satisfy few requirements such as can fit in bit-line pitches, have high speed, highly stable, and can easily hold the data. Figure 2.25 depicts a voltage based sense amplifier. When the value of BL is 1, the transistor *M1* is conducting, therefore the transistor *Q2* drives the *Out* line to 1. When the value of BL is 0, the transistor *M2* is conducting and drives the *Out* line to.

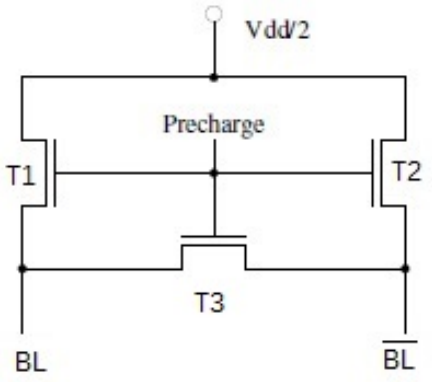


Figure 2.24: The precharge and equalization circuit

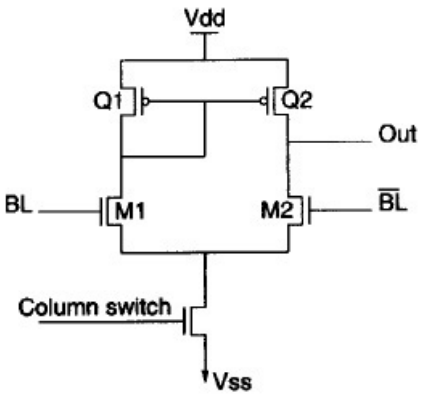


Figure 2.25: Sense amplifier

Electrical DRAM Model

The electrical DRAM memory model presents the memory in electrical schematic. The model contains several components listed below.

- *DRAM memory cell*: There are a different number of ways to construct a DRAM *memory cell*. The simplest way is by using one transistor DRAM cell, which consists of a pass transistor and capacitor, as shown in Figure 2.26(a). The transistor works as a switch that is controlled (turned ON or OFF) by the WL, and the cell capacitor is connected to BL. This capacitor stores the memory call value. The single pass-transistor (either PMOS or NMOS) has a disadvantage as it fails to provide a full voltage level (V_{dd} for NMOS and ground for PMOS) to BL. A transmission gate as replacement for the pass-transistor was proposed to solve this problem [8], but it is expensive in term of the silicon area needed.

- The *sense amplifier* depicted in Figure 2.26(b) is responsible for sensing the stored logic levels at the memory cell, then amplifying these voltages before forwarding to the output circuit. All types of sense amplifiers work according to the same

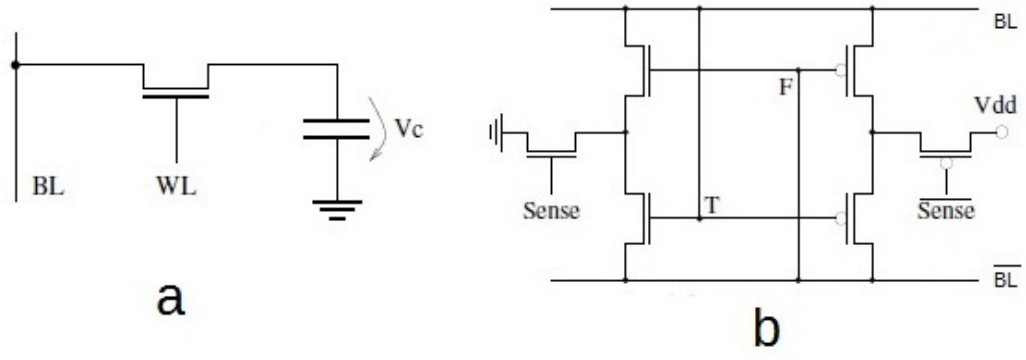


Figure 2.26: DRAM (a) cell schematic, (b) sense amplifier [6]

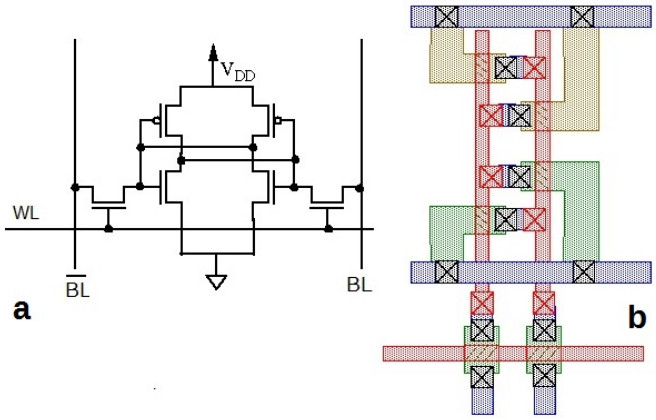


Figure 2.27: 6T-RAM cell (a) electrical model, (b) layout model [9]

principle; the sense amplifier will amplify a positive voltage differential to a full voltage high, and amplify a negative voltage differential to a full voltage low.

- *Pre-charge circuit*: As described in SARM.
- *Address decoder (Row and Column decoder)*: As described in SARM.

2.3.5 Process Technology

In this section we will briefly discuss the layout model, and the process steps to realize its physical implementation. Figure 2.27 shows the electrical representation of 6T-SRAM cell and its layout model. This layout model will be mapped on silicon during the manufacturing process. Similarly, a layout model exists for the remainder components of the memory. The manufacturing consists of several steps explained below.

The base material for semiconductor memories manufacturing process is called wafers. *Wafer* is obtained by cutting slices of lightly doped silicon, which is grown from a single crystalline seed. Wafers will have thickness around $400um$ and diameter around 30cm.

The process used to selectively pattern parts on a bulk of substrate is called UV

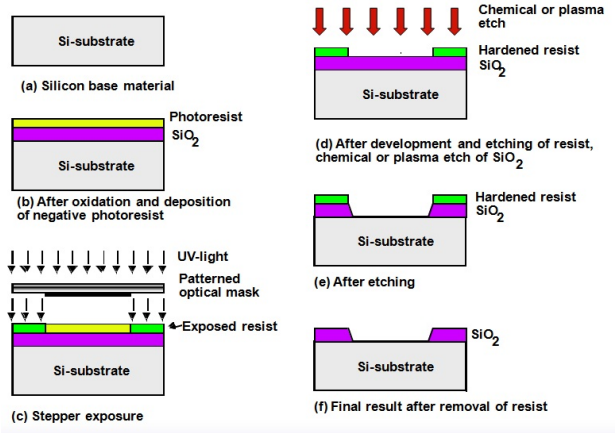


Figure 2.28: Process steps for patterning of SiO_2 [10]

lithography, where a light is used to transfer geometric patterns from opaque plate which has holes to allow light to pass through specified patterns to a light sensitive photoresist. Figure 2.28 depicts the main steps for patterning SiO2. The most interesting operations, during the UV lithography (photolithography) process, include:

- *Oxidation*: Exposing the wafer to a mixture of oxygen and hydrogen at 1000 Celsius, so a thin layer of SiO2 will form over the complete wafer, which used as an isolation layer and to form the transistor gates.

- *Photoresist coating*: A light sensitive polymer of a thickness of 1μm will cover the silicon oxide OSi2.

- *Stepper exposure*: A patterned mask containing the patterns to be transferred to the silicon is based over the wafer, then an UV light is exposed to the photoresist. Where the mask is opaque, the photoresist becomes soluble.

- *Photoresist development and bake*: Non-exposed areas of photoresist are removed by developing the wafer in either an acid or base solution. Then, soft-bake the wafer at low temperature to make the remaining photoresist harder.

- *Acid etching*: Selectively remove materials from the wafer, where areas are not covered by photoresist.

- *Spin, rinse, and dry*: The wafer is cleaned with deionized water then dried with nitrogen, this process is done in ultra-clean room to prevent dust from destroying the circuitry. The wafers are cleaned constantly after each process step to avoid contamination.

- *Photoresist Removal*: Selectively remove the remaining photoresis without damaging the device layers using a high-temperature plasma.

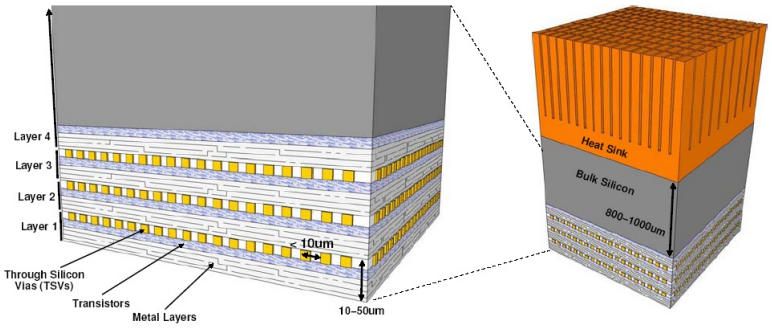


Figure 2.29: 3D-stacked dies [11]

2.4 3D Memory Stacked ICs

3D integrated memories have several advantages over 2D memories such as high speed, less power consumption, small form factor, and the ability to apply heterogeneous integration. 3D-stacked memory architecture based on TSV is a promising solution to Memory-Wall problem, because it can provide a wide and high frequency memory bus interface.

In [45], Gabriel et al. showed that a considerable speedups in performance can be gained if the system's main memory is placed on top of the processor, even though they considered the commodity DRAMs only. In [11], the researchers went further than commodity DRAMs in stacking and investigated the effect of highly parallel memory organization that was used in stacking over logic. Figure 2.29 shows 3D-ICs stacking with four layers, where each layer has a die produced in a typical 2D technology. The most promising vertical interconnect is Through-Silicon-Via (TSVs) which are able to combine different process technologies to stack memory on logic.

2.4.1 3D-Memory Classification

Partitioning memory across multiple layers can be implemented with different granularities [49]: (1) stacked banks that stack complete memory systems, (2) cell arrays stacked on logic, and (3) bit-partitioning. Each memory granularity has benefits and drawbacks [45]; they are explained next.

2.4.2 Stacked Banks

Stacking memory banks requires minimum changes in the overall memory design. Hence, several memory dies are directly stacked. Figure 2.30 shows an example of bank stacking, where the 3D-stacking consists of four layers of DRAM memory stacked on a traditional 2D processor core.

Each memory bank includes a complete memory system (memory cell array, row decoders, columns decoders, write derivers, and sense amplifiers). By stacking banks the global interconnect distance required is reduced [45]. As a result, power and delay will be reduced significantly. Samsung manufactured a 3D DRAM based on this configuration [50].

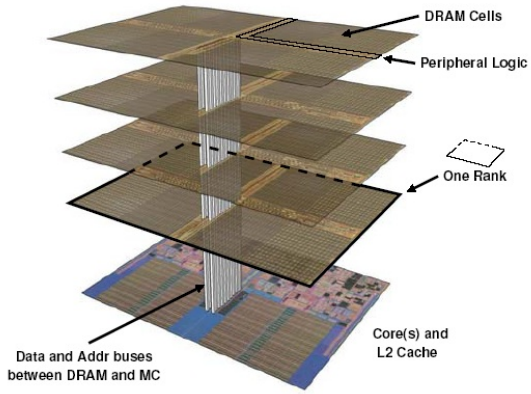


Figure 2.30: Memory banks stacked on logic [11]

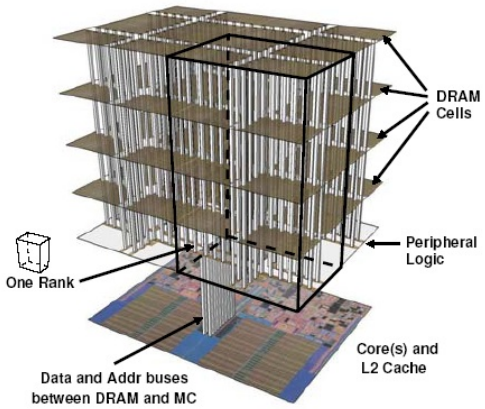


Figure 2.31: True 3D memory [11]

2.4.3 Cell Array Stacked-on-Logic

In this partitioning configuration, peripheral circuits (row decoders, columns decoders, write derivers, and sense amplifiers) are separated from the memory cell array, i.e, residing on different layer in the stack. This is referred to as *True 3D memory*, where the peripheral logic could be optimized for speed using a separate technology. The cell array can be customized and optimized for density, capacity, footprint, or thermal [49].

Tezzaron corporation has announced true 3D DRAM design strategy, where separate dies contain the DRAM cell arrays and DRAM peripheral circuits. Hence, the overall silicon area reduced and the speed is improved by implementing the peripheral circuit on a high performance logic die [28, 11, 51].

Figure 2.31 shows an example of this configuration. It shows that the various control and access circuits (row decoders, columns decoders, write derivers, and sense amplifiers) are implemented on the bottom layer, while the top layers contain stacked arrays. This organization reduces the length of word line, bit line, and internal bus to have reduced memory access latency. The splitting of bit cell arrays can further be classified into two

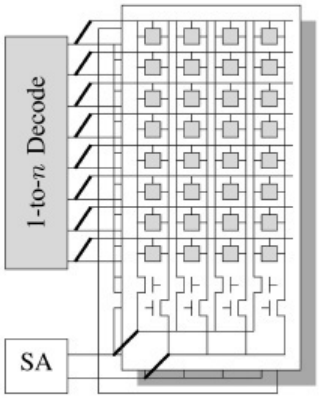


Figure 2.32: Column-stacking (3D-divided word-line) [12]

sub-types, to obtain further power and latency improvements [49]:

1. **Column-stacking or 3D-divided word-line**: Here, each word-line is splitted over two or more layers as shown in Figure 2.32. By reducing the length of word-line, latency and power will be reduced. In this configuration, the row decoder has to drive word-lines in both dies. Therefore each word line requires one TSV via [12]. Moreover, additional vias are needed, because the column select multiplexors have been split across multiple dies. The number of vias used in this configuration is greater than that of the bank-stack organization because each bit line and its complement must be routed to the sense amplifier.

2. **Row-stacking or 3D-divided bit-line**: Here, each bit-line is splitted over two or more layers as shown in the Figure 2.33. A direct consequence of this stacking approach is that row decoders can be split among multiple layers. Therefore, latency and power will be reduced. In this configuration, word-lines have similar lengths and loads as in the planar organization. However, bit-lines have reduced length [49]. Word-lines are divided and mapped on different layers, and each layer will has a word-line driver. However, duplication in number of drivers is compensated by resized drivers [52]. Bit-lines switch faster because the bit-line length and number of pass transistors for each bit-line are reduced.

   The sense amplifiers can be duplicated across different layers and this will reduce the access times, or the sense amplifiers can be shared among layers and this is suitable for reducing the number of transistors and leakage current. Sharing sense amplifiers will increase the number of 3D TSVs, because all bit-lines and $\overline{BL}$ require one via to connect the sense amplifier. Duplicated sense amplifiers reduced the number of TSVs by half, but require more transistors, and result in extra leakage [52].

Cell-array stacked on logic partitioning introduces a finer granularity than the *stack-banks* partitioning method, and provides a greater performance and power reduction.

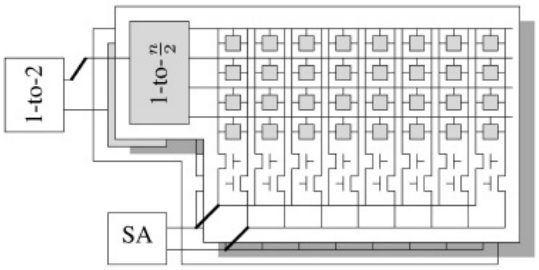


Figure 2.33: Row-stacking (3D-divided bit line) [12]

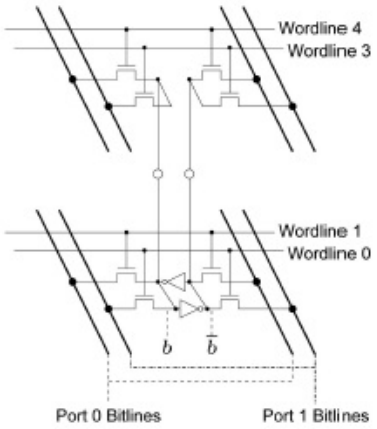


Figure 2.34: Bit partitioning 3D register file [12]

However, requires a redesign of the memory. The exact partitioning strategy (*column-stacking or row-stacking*) depends on the design objective; bit-line length reduction mainly minimizes the energy consumption, and word-line length reduction mainly reduces the latency [12, 49].

2.4.4 Intra-Cell (Bit) Partitioning

This memory partitioning technique is considered as the finest granularity level of partitioning, as it targets to split the cell over multiple layers. This can be the cell itself or its access port. For example, any of the six transistors of the 6T-SRAM cell can be assigned to different layers. The success of this partitioning depends on the TSV size and density, as compared to SRAM cell size. If the cell area is in the same order as the TSV dimensions, splitting cell across multiple layers in a beneficial way will be difficult or impossible.

In [12], this strategy is used in register files (as an example of multi-port SRAM), as shown in Figure 2.34, where the access transistors of the cells are partitioned across multiple layers. Bit partitioning may reduce the wire length and the gate load of word-lines. Thus, reducing energy and latency.

Bit partitioning may have other forms; for example, each inverter in the SRAM cell

Table 2.3: 6T-SRAM cell dimensions for various technologies

Technology[nm]	**90**	**65**	**45**	**32**	**22**	**16**
Area[um^2]	1	0.570	0.340	0.143	0.100	0.039
Height[um]	n.a.	0.460	n.a.	0.270	0.554	0.300
Width[um]	n.a.	1.240	n.a.	0.530	0.180	0.130
Reference	[53]	[54]	[55]	[56]	[57]	[58]

Table 2.4: TSV dimensions

	Manufactured			ITRS prediction/year	
	TSMC	**IMEC**	**LETI**	**2009-2012**	**2013-2015**
Diameter[um]	6.5	5.2	3	1 -2	0.8- 1.5
Pitch[um]	n.a.	10	n.a.	2-4	1.6-3.0
Reference	[59]	[60]	[61]	[62]	[62]

can be placed on a different layer, or NMOS and PMOS transistors could be splitted over different layers. However, such approach requires at least one additional TSV per 6T-SRAM cell; where the TSV size may exceeds the size of SRAM cell [45]. Table 2.3 shows the dimensions for 6T-SRAM cell for different technologies, and Table 2.4 shows TSV dimensions according to International Technology Roadmap for Semiconductors (ITRS) predictions and leading manufacturing companies. Since TSV size is greater than 6T-SRAM cell size, bit-partitioning seems impractical with current TSV technology.

2.5 Summary

This chapter introduced 3D memories. The main topics discussed are the following:

- Introduction to Three-Dimensional Stacked Integrated Circuits (3D-SICs), where we discussed it's definition, drivers, and its benefits over 2D-ICs.
- Explanation of the 3D ICs manufacturing process consisting of three main steps we described: (1) TSV manufacturing process, (2) wafer thinning, and (3) the bonding process.
- Description of the functionality and implementation of 2D memories, where memory models including behavioral, functional, and electrical were explained for SRAM and DRAM.
- Explained the various ways 2D memories can be extended in the third dimension according to different partitioning granularities such as: bank partitioning, and cell partitioning.

ICs Failure Mechanisms and Models

3

In this chapter, we discuss the failure mechanisms both in 2D and 3D ICs and their fault models. Section 3.1 defines key terminologies related to ICs quality and reliability, and the relationship among them. Section 3.2 classifies defects; either related to 2D or 3D processing. Section 3.3 presents a fault classification of defects according to their manifestation over time. Section 3.4 shows the fault models that can be applied to model defects in 2D-ICs, and how to extend them for 3D related defects. Finally, Section 3.5 summarize the chapter.

3.1 Key Terminologies

This section defines key terminology regarding IC failures in Section 3.1.1. Section 3.1.2 subsequently explains the difference between quality and reliability.

3.1.1 Defects, Faults, Fault Models, and Failures

Figure 3.1 summarizes the relation between key terminology related to IC failures. Each of them is explained next [14, 63].

- *Failure mechanism*: Failure mechanisms cause defects which depend on the process technology and complexity of the circuit layout. For example, warpage-induced stress in thinned dies and TSV pop-out due to Coefficient of Thermal Expansion (CTE) mismatch will cause defective ICs [64].

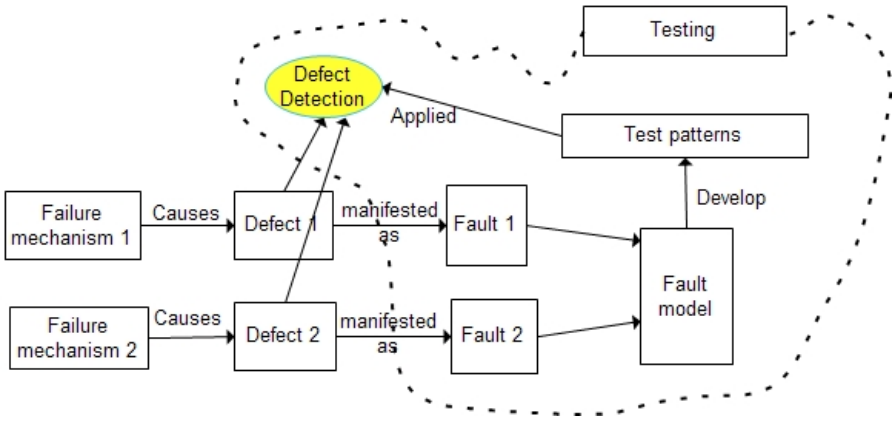


Figure 3.1: Key terminologies

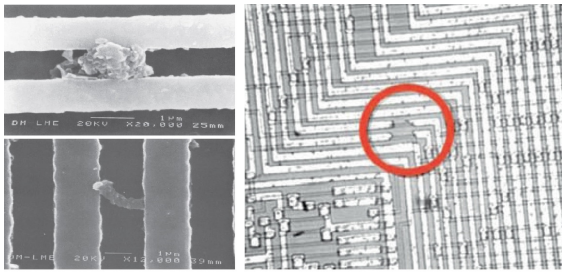


Figure 3.2: Defect example: (a) bridge defect, and (b) open defect [13]

- *Defect*: A defect is an unintended difference between the implemented hardware and the intended design [13, 14]. The authors in [65] define a defect as a physical anomalous emerged from the manufacturing process that was not originally defined in the design circuit. On one hand, physical anomalous due to unintended particles, such as an extra or missing material caused by contamination, are called *hard defects*. On the other hand, latent defects are hidden changes in IC material due to minor process variation, are becoming crucial in nanotechnology such as insufficient doping and material impurities.

  The probability of defects in ICs increases with decreased feature size [66, 67, 32]. These defects may occur during IC manufacturing (such as ion contamination and atom gradient) or operational usage (such as Hot Carrier Injection (HCI), Electromigration (EM), and Time Dependent Dielectric Breakdown (TDDB)) [14, 68]. Two examples of defects are depicted in Figure 3.2. Part (a) shows a bridge defect due to impurities, and part (b) shows open defects due to Electromigration phenomena.

- *Fault*: A fault is the way the defect manifest itself at electrical level; therefore it is an electrical representation of the behavior of the physical defect, such as *SA0*, and *SA1* fault. Fault abstraction reduces the complexity since as many defects have the same fault behavior.

- *Fault Model*: A fault model is a collection of faults with similar properties; such as Stuck-At-Fault *SAF* model that encompasses SA0 fault and SA1 faults. Fault models should accurately reflect the behavior of defects; as they are used for generating and evaluating test patterns [17]. Having a single fault model that accurately reflects the behavior of all possible defects is difficult. Therefore, various fault models are used.

- *Test pattern*: An input vector for the Circuit-Under-Test (CUT) that causes the present of a fault to be observed at a primary output.

- *Testing*: Testing is a process performed after manufacturing to ensure the correctness of the IC. The target is to distinguish between good, bad and weak ICs. Tests with high fault coverage are able to detect strong and weak defects, thus providing

good quality. Weak defects can be detected by testing under harsh environmental conditions such as high temperature [65].

- *Failure*: A failure occurs when the service provided by the IC differs from the expected service [13, 14, 7]. The failure can produce incorrect results, no results at all, or violations of design parameters (e.g., a correct but delayed result) [65]. Not all defects lead to failures; for example, partial opens, or defects that are masked by redundant hardware.

As depicted in Figure 3.1, different defects are originated because of different failure mechanisms. Each defect is manifested as a fault at electrical level where faults with similar properties are collected as one fault model that reflect the behavior of the original defects. Fault model is used for test patterns development. Then these test patterns are applied to the Circuit-Under-Test (CUT) to detect defective ICs. All defective ICs will be detected without any escapes if proper fault modeling was performed. Thus, high quality is guaranteed.

3.1.2 Quality vs. Reliability

The previous section explained key terminology related to IC quality. The quality of the chip is determined at the manufacturing test. However, defects may also occur during normal operation. A chip should work reliable enough during its prescribed life time, also referred to as reliability. Next we define both terms.

Quality: Quality is the metric used to quantify defective ICs that escape the manufacturing test measured in Defect Part Per Million (DPPM). Generally, a low number of customer returns indicates a high quality [65, 63].

Reliability: Reliability is the metric used to quantify the ability of a system in operating correctly for a specific period of time under specific conditions (e.g. temperature, voltage, etc). Reliability is measured by Failure In Time FIT (time=10^9 working hours). The time-frame a system functions correctly indicates the level of reliability [65, 63]. The failure rate of IC over the product life time typically follow the trend of the graph depicted in Figure 3.3, also known as *Reliability Bathtub Curve*. This curve is made up of three individual regions [14, 15]:

1. Infant mortality: This stage represents the failure rate for initial operation of the IC. It has relatively a high failure rate as weak chips (that passed the test) start to fail quickly. The number of failures reduces over time in this stage.

2. Normal life: In this stage, failures occur sporadically. The failure rate is very low as compared to the previous stage.

3. Wear-out: After the normal life time, more and more ICs start failing due to wear out, e.g., due to electromigration, etc. This stage is less important for ICs, as its life-time is reached.

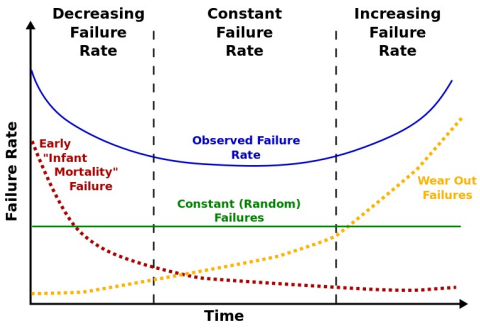


Figure 3.3: Reliability bathtub curve [14, 15]

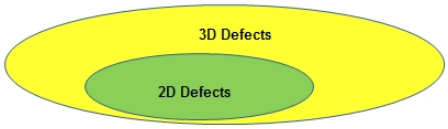


Figure 3.4: 3D defects nature

3.2 Defect Classification

Defects can be classified into two categories; either originating from wafer manufacturing (2D defects) or from the stacking process (3D defects). Here, 2D-defects are subset of 3D-defects as depicted in Figure 3.4.

3.2.1 Defects in 2D-ICs

Traditional defects that may happen in 2D-ICs are numerous; examples are [14, 15, 7]:

- Broken component; broken metal line or transistor.
- Incorrect mask.
- Incorrect connection.
- Error in functional design.
- Short between metal lines and V_{DD} or V_{SS}.
- Electromigration that may cause voids, or splinters between lines.
- Open defects due to missing conducting material or extra insulating material.
- Gate-oxide shorts due to electric field stress or Electro Static Discharge (ESD).

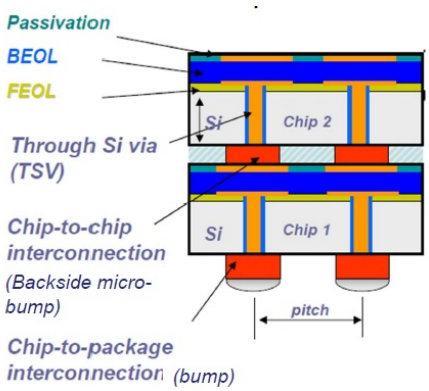


Figure 3.5: Defect location within 3D-SIC structure

3.2.2 Defects in 3D-ICs

3D integration includes new manufacturing steps that may cause additional 3D process-related defects [3, 32]. Figure 3.5 depicts a general 3D-SIC integration with possible defect locations. These defects can be classified according to their locations: (1) Intra-die defects within a die, and (2) Interconnect-based defects that form vertical interconnects between two dies. The defects are explained next.

Intra-Die Defects

Dies within 3D-SICs not only have the same traditional defects as in 2D-dies, but also new intra-die defects may be introduced by 3D processes such as wafer thinning, alignment, and bonding. For example, wafer thinning may cause degradation of transistor I-V characteristics [69]. Thermo-mechanical stress due to Coefficient of Thermal Expansion (CTE) mismatch may cause dies malfunction [69]. Defects inside the die, within a 3D-SIC can also be classified according to their location into three classes:

1. *Defects related to Back-End-Of-line (metal layer)*: Examples of such defects include:
   - stress induced by copper TSVs affect the BEOL interconnections [70].
   - Delamination and cracks [70].
   - Damage due to TSV copper filling [70].
2. *Defects related to Front-End-of-Line (transistor layer)*: Examples of such defects include:
   - Increased leakage copper from TSV [70].
   - Mechanical stress induced drift [70].
3. *Defects related to substrate*: Examples of such defects include:

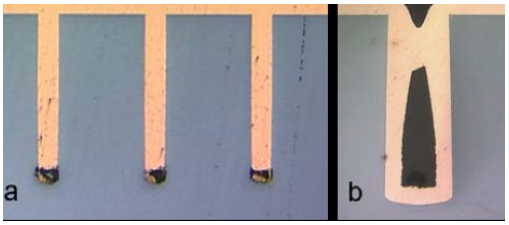


Figure 3.6: (a) TSV voids, and (b) TSVs pinch-off [16]

- CTE mismatch between copper, silicon, and silicon-oxide (SiO2) causes a local stress in Si that changes the MOS mobility. This might cause timing violations and increased critical path delay [16].
- Cracks and fractures [16].

Interconnect-Based Defects

Interconnects in 3D-stacking are considered as a potential source of defects. These defects may occur during manufacturing, bonding, or life-time of 3D-SICs. Interconnects defects can be classified into:

1. *Defects related to Through-Silicon-Via (TSV)*: Examples of such defect include:

   - An incomplete fill of TSVs (voids) that may originate from insufficient wetting of the vias during the plating as shown in Figure 3.6(a). Moreover, voids may originate also from electromigration. Voids cause partial opens and lead to increased TSV resistance [5, 71, 72, 16].
   - Pinch-off of TSVs due to plating, which also increase the TSV resistance. It may create a partial open as shown in Figure 3.6(b) [16].
   - TSV missing contact with transistors layer or metal layer [43, 16].
   - Ineffective removal of the seed layer during TSV formation that may cause a short between neighboring TSVs.
   - Pinhole defects that occur along the TSV wall and cause a short between TSV and substrate. Pinholes form a low resistance path between the TSV and the ground. Therefore, the signal quality between two dies will have either complete or partial degradation [5, 16, 71, 72].
   - TSV misalignment with μ-bumps during the bonding process increases the path resistance and causes TSV opens. Misalignment can be of two types; a misalignment shift that causes a high resistance path, and a complete misalignment that causes a complete open [16, 71, 72].
   - TSV cracks and sidewall delamination due to CTE mismatch between copper and substrate. Delamination occurs in the conducting material and continues to expand to the substrate and the insulation layer causing the path resistance to increase [73, 74, 71, 75, 72].

Table 3.1: Summary: TSV Defects and their electrical models

Defect	**Electrical model**
Void	Resistive Open
TSV-to-TSV short	Resistive Bridge
Pinch-off	Resistive Open
Delamination and Crack	Resistive Open
Pinhole in liner	Resistive Short
TSV missing contact with BEOL/FEOL	Resistive Open
TSV misalignment	Resistive Open
Small distance between TSVs	Capacitive Coupling

Table 3.2: Summary: μ-bump Defects and their electrical models

Defect	**Electrical Model**
μ-bump to μ-bump short	Resistive Bridge
Voids and cracks	Resistive Open
Damage in underlying BEOL	Resistive Open/Bridge
Weak bonding	Resistive Open

- Crosstalk between different TSVs due to the short distance in between [72, 76, 77].

Table 3.1 summarizes the TSV defects and their electrical models, i.e., how the defect behaves electrically.

2. *Defects related to μ-bumps*: Examples of such defects include:

   - Damage in underlying BEOL resulting in opens or bridges [70].
   - Weak bonding due to buckled (bend over) thinned Si chip resulting in resistive open [70].
   - TSVs height variation, which may cause tin to be squeezed out from μ-bump. This could cause shorts between μ-bumps [69].
   - misalignment with TSV during bonding causes TSV open that increase TSV resistance [72].
   - Electromigration can cause voids and cracks in the joints. The resulting voids and cracks can increase the resistance of μ-bump, and voids sometimes lead to complete opens [78].
   - μ-bump cracks due to CTE mismatch between copper, silicon, and silicon-oxide [16].

Table 3.2 summarizes the μ-bump defects and their electrical models.

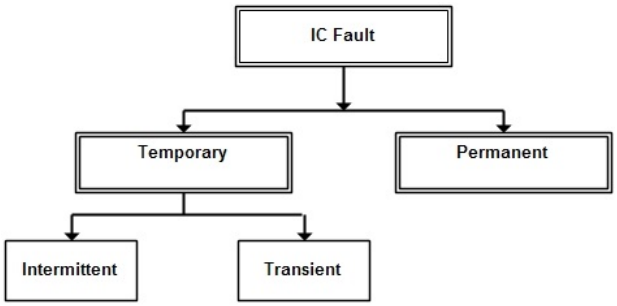


Figure 3.7: IC faults classification [14, 15]

3.3 Fault Classification

Faults are logical level representations of the behavior of physical defects. Faults can be classified according the way defects manifest themselves over time; either permanent or temporary faults [7, 15] as shown in Figure 3.7. This classification is applicable to both 2D-ICs and 3D-SICs. Each is described below.

3.3.1 Permanent Faults

Permanent faults are known as *Hard* or *Solid* faults. These faults are a consequence of irreversible physical processes. For example, they may occur due to hard defects and wear-out phenomenon. Permanent faults are permanent in nature and will not vanish; therefore testing such faults can be repeated with same results [13].

3.3.2 Temporary Faults

Temporary faults occur more frequent than permanent faults [15] and show up randomly for unknown but finite amount of time. This makes their detection difficult as the faults may not show up during the test. Temporary faults can be divided into two subtypes as shown in Figure 3.7 [14, 7]:

1. *Transient Faults*: Transient faults are caused by environmental conditions which originate from outside the IC such as radiations, temperature, humidity, pressure, power supply fluctuation, ground loops, and vibration [7]. Therefore, it is important to increase the IC's noise immunity. Transient faults have obscure and ambiguous influence on logical values; making it hard to model them. For example, soft errors might cause an undesired transition in wires and memory cells [7].

2. *Intermittent Faults*: Intermittent faults are caused by non-environmental conditions, such as signal interference, variation in resistances and capacitances, aged components, parameter degradations, and loose connections. Similarly as transient faults intermittent faults are temporary in nature and therefore hard to detect.

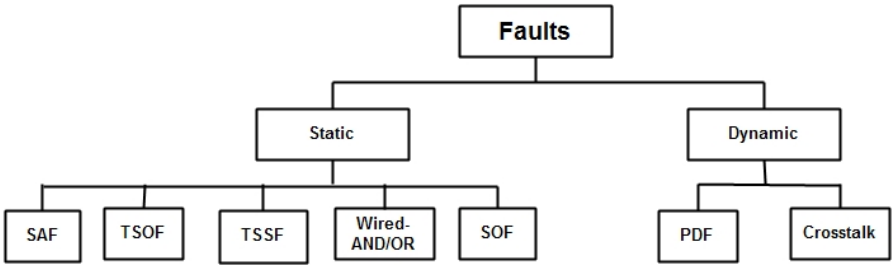


Figure 3.8: 2D-IC fault models classification

3.4 Fault Models

In general, models are used to simplify and reduce the complexity of physical systems. In ICs, many physical defects map to the same fault at higher abstraction level. Therefore, a test that targets a specific fault can be used to detect many defects without testing each defect separately. Faults with similar properties are collected in a fault model. Imperfect modeling may result in improper fault models that fail to include all defects or test for non-existing defects. This section introduces the most important fault models used in 2D-ICs and 3D-ICs.

3.4.1 2D Fault models

Figure 3.8 shows a general classification of 2D-IC fault models. The faults can be categorized in static and dynamic fault models. Each fault model is explained below.

Static fault models

Static fault models are time independent. Sensitizing such faults requires at the most one operation; i.e., single write operation sensitizes the fault. Static fault models include:

- Stuck-At-Fault (SAF) model

  The single SAF model is the simplest fault model used to represent different physical faults at gate level. In this model, a single line is assumed to be faulty at a time with permanent value 0 or 1 (stuck-at-0 or stuck-at-1 fault). A circuit with N-lines will have at most 2N single stuck-at faults; the gates are assumed to be defect-free. Besides the single stuck-at fault, also multiple coexisting faults can be considered (multiple SAF model). For example, a circuit with N-lines can have $3^N - 1$ multiple stuck-at faults. Practically, the single SAF model is nearly an effective as the multiple SAF model [63].

- Transistor Stuck-Open Fault (TSOF) model

  The transistor stuck-open is a fault model at transistor level where a missing or extra material will cause a non-conducting transistor which result in a disconnected branch. If open fault happened at fan-out, it will cause several open transistors.

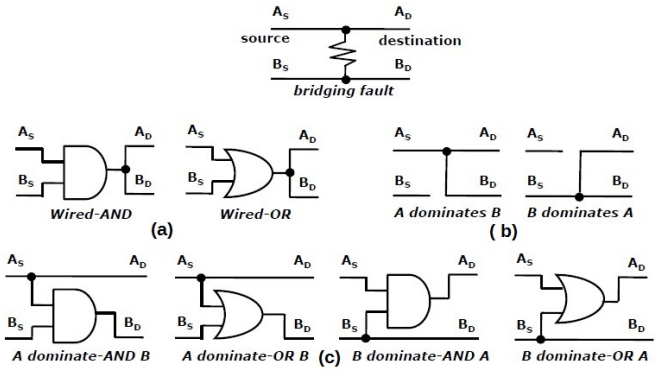


Figure 3.9: Bridging fault models [17]

- Transistor Stuck-Short Fault (TSSF) model

  The transistor stuck-short is a fault model at transistor level. Defects that cause the transistor to conduct permanently are modeled by a stuck-short fault model [17].

- Bridging fault model

  The bridging is a fault model at transistor level or gate level. A bridging fault represents an undesirable short between wires, for example as a result of extra unintended conducting material. Bridging faults may occur after fabrication due to electromigration, oxide surface conduction, and lateral charge spreading [15]. Bridges are technology-dependent, and based on that may be modeled with different fault models.

  Figure 3.9 illustrates commonly used bridging faults, where A_D and B_D represent the behavior of the two shorted signals A_S and B_S. The fault models are: (a) wired-AND/wired-OR bridging fault model; in this model both wires take the same value after the defect described by the AND/OR behavior of the defect free case, (b) dominant bridging fault model; here one driver is assumed to dominate the logic value on the shorted nets, therefore it is more difficult to detect because the faulty behavior shows up on one net only, and (c) dominant-AND/dominant-OR bridging fault model. Here one driver dominates the logic value of the shorted nets for one logic value only. When a wire is shorted to V_{DD} or V_{SS}, the bridging fault is equivalent to the SAF fault.

- Suck Open Fault (SOF) model

  A stuck open fault model assumes the wire to be completely broken, here the end side of the wire floats. The broken wire has two possibilities [17]: (1) the broken wire interconnecting gates (*inter-gate open*). Therefore, behaves like stuck-at fault(SAF) and modeled at gate level, and (2) the broken wire interconnecting transistors inside the gate (*intra-gate open*). Therefore, behaves like transistor stuck open fault (TSOF), and modeled at transistor level.

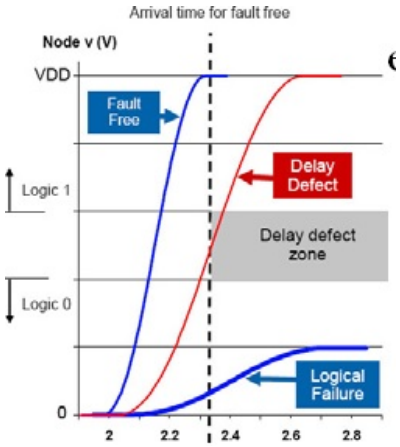


Figure 3.10: Delay fault model

Dynamic fault models

These fault models are time independent. Dynamic faults require more than one operation sequentially in order to be sensitized. It includes:

- Path Delay Fault (PDF) model

  The PDF model at gate level. Delay faults (DFs) affect the temporal behavior of the circuit while logically the circuit is fault-free as depicted in Figure 3.10; here the fault-free signal arrives after the determined time (circuit clock period) and therefore causes a delay fault. Two types of delay fault are usually used [14, 17]: (1) the gate-delay fault (GDF), where a single gate has a slow transition low-to-high (0-1) or high-to-low (1-0) from input to output, and (2) the path-delay fault (PDF), where a complete path from primary input to primary output has a slow low-to-high (0-1) or high-to-low (1-0) transition. The PDF model is more general than the GDF model as it models the cumulative delay along the path including gate and wire delay.

- Crosstalk fault model

  Crosstalk is a fault model at interconnects level. Crosstalk effect increases in nanometer technology [79]. In crosstalk, values of wires are affected by their neighbors. This crosstalk is modeled usually by the Maximum Aggressor Fault (MAF) Model; here lines may cause two main types of faults as shown in Figure 3.11:

  1. A crosstalk switch in which switching on the aggressor lines cause the victim to be affected due to coupling. Figure 3.11 (a) shows a positive glitch and (b) shows a negative glitch fault due to the up and down transitions on the aggressor lines.
  2. A crosstalk delay where transition on aggressor lines delays a simultaneous opposite transition in the victim line. Figure 3.11 (c) shows a falling delay in the victim line due to simultaneous switching of the aggressor lines in the opposite direction, and (d) shows a rising delay fault.

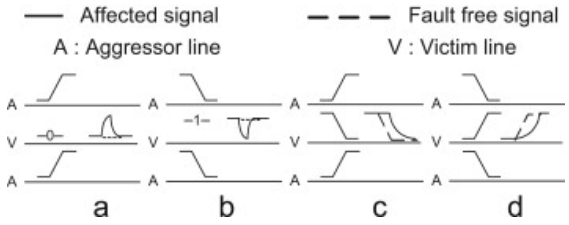


Figure 3.11: Crosstalk faults:(a) Positive glitch,(b) Negative glitch, (c) Falling delay fault, and(d) Rising delay fault [18, 19]

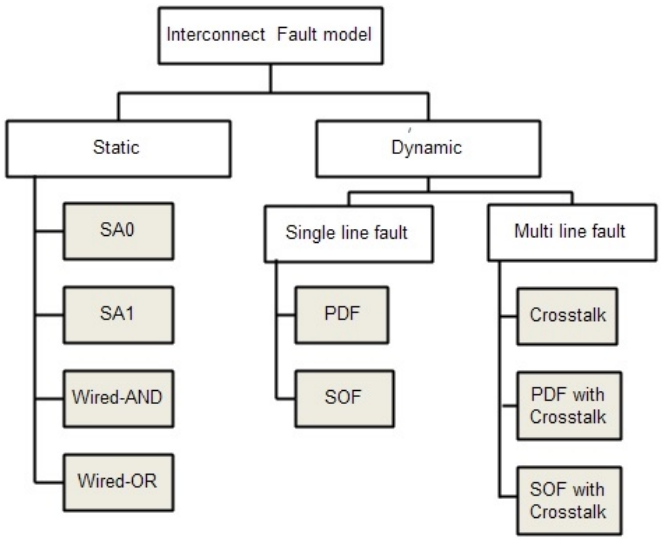


Figure 3.12: Interconnect fault models classification

3.4.2 3D Fault models

3D stacking requires extra processing steps that may add new defects; hence, potentially new faults, and fault models. Figure 3.4 showed that 2D defects are subset of 3D defects. Hence, 3D fault models include the classical 2D fault models in addition to possible new fault models. A 3D stack consists of vertically staked dies interconnected by TSVs. Defects in the die have already been discussed. Here we focus on new defects in the vertical interconnections. Figure 3.12 classifies the interconnect faults. They are divided into two types: static fault models and dynamic fault models. Both are explained next.

Static fault models

These fault models are time independent (applying a set of inputs at any instance of time will give the same output). Their sensitization may require at most one test pattern. They include:

- Stuck-at-1 (SA1) fault model: Regardless on the value applied on interconnect; it always transfers a 1.

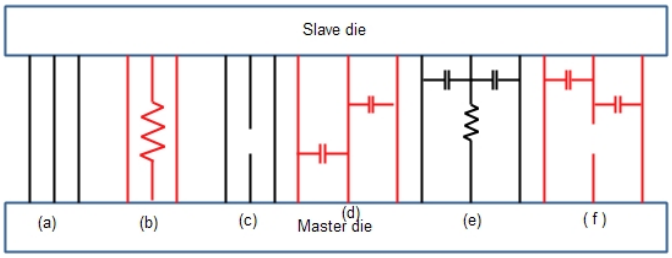


Figure 3.13: Dynamic faults; (a) golden case, (b) and (c) single line faults, and (d) (e) and (f) multi line faults

- Stuck-at-0 (SA0) fault model: Regardless on the value applied on interconnect; it always transfers a 0.
- Bridge wired-AND fault model: Here two lines or more are shorted together, and their behavior is 0-dominant; the value of all shorted lines will be 0 if at least one of them has value 0.
- Bridge wired-OR fault model: Here two lines or more are shorted together, and their behavior is 1-dominant; the value of all shorted lines will be 1 if at least one of them has value 1.

Dynamic fault models

These fault models are time dependent (i.e., defects impact the timing operation, such as delay). Their sensitization requires applying more than one operation sequentially as they are time dependent. Dynamic faults as shown in Figure 3.13 can be classified based on the number of lines and based on fault cause.

1. Single line faults: Where the fault involves only a single faulty line.
    - Path Delay Fault (PDF): The line has a high resistance that causes delay along the path.
    - Stuck Open Fault (SOF): The line is completely open due to infinite resistance.
2. Multi line faults: Where the fault involves more than one line. They consist of:
    - Crosstalk fault: Each fault-free wire can be affected by coupling of multiple neighbors simultaneously. Here, we can apply the MAF model used for crosstalk explained in the previous section. Each TSV could have crosstalk effect from eight surrounding neighbors at maximum, where the effect of non-neighbors TSVs is isolated by the direct neighbor TSVs.
    - Path Delay Fault with Crosstalk: The line has a high resistance that causes a delay, and it suffers from coupling with neighbors.
    - Stuck Open Fault with Crosstalk: The line is completely open and the voltage on the floating point of the open depends on the coupling with the neighboring interconnects .

3.5 Summary

This chapter explained the failure mechanisms in both 2D and 3D ICs and the fault models used for test patterns development. The main topics discussed are:

- Defining the key terminology related to ICs quality and reliability such as failure mechanisms, defects, faults, fault models, test patterns, testing, and failures.
- Classification of defects according to the IC technology; 2D or 3D and the defect location.
- Classification of faults according to how defects manifest themselves over time at system level; either permanent faults or non-permanent faults.
- Discussion of fault models used to model the faults in 2D-ICs, and the emerging models in 3D-SICs.
- Explanation of the main emerging fault models related to interconnects in 3D integration.

.

Testing Memory-on-Logic Interconnect

4

Our main idea to test interconnects between stacked memories and logic is by performing read and writes operations to the memory. These read and write operations must satisfy several requirements, such as being able detect the interconnect faults. Section 4.1 discusses the targeted fault models. Section 4.2 explains their detection conditions both for static and dynamic fault models for general stacked ICs. Section 4.3 discusses the applicable detection conditions specific for faults in memory stacked on logic. Section 4.4 describes how test patterns are derived from these detection conditions. Finally, Section 4.5 provides a chapter summary.

4.1 Targeted Fault Models

A set of test patterns that detect interconnect faults must satisfy several requirements such as: (1) small number of testing vectors to reduce cost, (2) high fault coverage, and (3) both detection and diagnosis capabilities to detect faults and identify their exact locations [80]. In this thesis, we focus on the detection of 3D-SICs interconnects faults; more specifically on their test optimization and test quality. As already shown in Figure 3.12, interconnect faults can be classified into two main types:

- Static faults: Which have four different types:
  - SA0: The interconnect transfers a 0.
  - SA1: The interconnect transfers a 1.
  - Wired-AND: The bridge behavior between two interconnects is 0-dominant.
  - Wired-OR: The bridge behavior between two interconnects is 1-dominant.
- Dynamic faults: This can be further categorized into two subclasses:
  - Single line faults: Where faults affect only a single line. They consist of:
    - Path Delay Fault (PDF): A fault due to a partial resistive open line.
    - Stuck Open Fault (SOF): A fault due to a complete open line.
  - Multi line faults: Where faults affect multiple lines. They consist of:
    - Crosstalk Fault: Faults on victim lines caused by crosstalk from aggressive neighbors.
    - Path Delay Fault with Crosstalk: Faults on partial resistive opens affected by crosstalk from neighbors.
    - Stuck Open Fault with Crosstalk: Faults on complete open lines affected by crosstalk from neighbors.

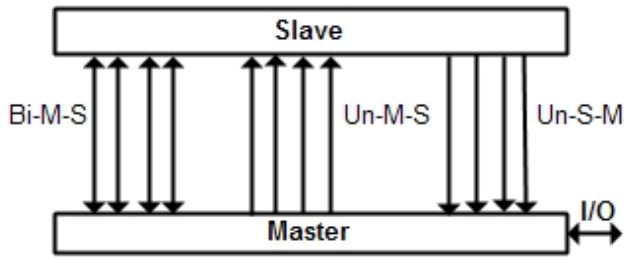


Figure 4.1: Master-slave stacking and possible interconnects types

Figure 4.1 depicts the general connection wires between two dies. The wires are of three kinds: Bi-directional master-slave (*Bi-M-S*) represented as $\updownarrow$, uni-directional master-slave (*Un-M-S*) as $\uparrow$, and uni-directional slave-master (*Un-S-M*) as $\downarrow$. Examples of communication that takes place on such wires are: (1) Bi-M-S $\updownarrow$ interconnects could be bidirectional data lines, (2) Un-M-S $\uparrow$ interconnects could be address lines, input data lines, or control lines, and (3) Un-S-M $\downarrow$ interconnects could represent output data lines or memory output control signals. Testing these wires is based on the following basic assumptions:

- Master (die with logic) and slave (memory die) are fault free.
- Each wire is (directly or indirectly via the memory) fully controllable and observable.
- We assume a single fault at a time for the interconnects.
- All faults must be collected at the master side; a fault must be detected on the master side or propagated back to the master in case detected on the slave side.

Throughout this thesis the following terminologies are interchangeable; *net, line, interconnect, and wire.*

4.2 General Detection Conditions

The detection conditions for each targeted fault, which depend on the wire type, are described next. In general, the detection conditions adhere to the following steps:

1. Fault sensitization (activation): Sensitize the fault in a particular wire to create a different behavior between the faulty and fault free circuit.
2. Fault propagation: Propagate the fault effect to the master (if needed).
3. Line justification: Back trace the values of the signals to the input of the circuit, such that the fault is sensitized.

Next, we describe the general detection conditions for each wire. Sensitizing the fault for Un-M-S interconnects happens directly by the master, after sensitization, the value

Table 4.1: Fault detection steps (fault propagation and line justification) for different interconnect wires

	Un-M-S $\uparrow$	**Un-S-M** $\downarrow$	**Bi-M-S** $\updownarrow$	
Fault propagation	Propagate the fault effect to the master using the line $\updownarrow$ or $\downarrow$	No action required	Tested in M-S direction: Propagate the fault effect to the master using the line $\updownarrow$ or $\downarrow$	Tested in S-M direction: No action required
Line justification	No action required	Justify the line $\updownarrow$ or $\uparrow$	No action required	Justify the line $\updownarrow$ or $\uparrow$

must be propagated back from the slave to the master by using either Un-S-M or Bi-M-S interconnects.

The detection condition for Un-S-M interconnects is as follows: Sensitize the fault on Un-S-M interconnect and propagate the sensitized value from slave to master, then justify the line by using a Bi-M-S or Un-M-S interconnect to reach the slave side.

The detection condition for Bi-M-S interconnects happens by using one of the approaches for Un-M-S or Un-S-M such that: (1) to test the Bi-M-S interconnect for master-to-slave direction, sensitize the value on the interconnect through the master then propagate back the value from slave to master by using Un-S-M , and (2) to test the Bi-M-S interconnect for slave-to-master direction, sensitize the fault on Bi-M-S interconnect and propagate back the sensitized value from slave to master, then make a line justification by using another Bi-M-S or Un-M-S interconnect to reach the slave side.

Fault sensitization is based on the fault model; however, fault propagation and line justification are common shared steps of the detection conditions independent of the fault model. These common steps are summarized in Table 4.1. In Section 4.2.1 and 4.2.2 we describe the fault sensitization part of the detection condition of static and dynamic faults, respectively.

4.2.1 Static Faults

Static faults are time independent and consist of four types (SA0, SA1, wired-AND/OR). Note that each fault could occur on the different interconnect types. The detection conditions for the static faults are described next.

SA0 (SA1)

A stuck at fault on a single wire forces that wire on a specific value; either 0 (SA0) or 1 (SA1). Therefore, to sensitize a stuck-at fault an opposite value must be applied to the wire, i.e., a '1' for SA0 and a '0' for SA1. Tables 4.2 show the fault sensitization for SA0 faults for the interconnect types, while fault propagation and line justification steps are similar as in Table 4.1.

For example, there are two detection conditions for SA0 fault at Bi-M-S $\updownarrow$ interconnect as shown in Table 4.1 and 4.2. In both cases the fault is sensitized on $\updownarrow$. In the first case, the sensitized line $\updownarrow$ is used to transfer data from master to slave. In this case $\downarrow$ or $\updownarrow$ must be used to propagate value back to the master. In the second case, the line $\updownarrow$ is used to transfer data from slave to master, to justify the line, a value of 1 must be set

Table 4.2: Fault sensitization for SA0 faults

	Un-M-S ↑	**Un-S-M ↓**	**Bi-M-S ↕**	
Fault sensitization	Set the line ↑ to 1	Set the line ↓ to 1	Tested in M-S direction: Set the line ↕ to 1	Tested in S-M direction: Set the line ↕ to 1

Table 4.3: Interconnect types and their fault models

Case	**Wired-AND**	**Wired-OR**
1	Bi-M-S/Bi-M-S	Bi-M-S/Bi-M-S
2	Bi-M-S/Uni-M-S	Bi-M-S/Uni-M-S
3	Bi-M-S/Uni-S-M	Bi-M-S/Uni-S-M
4	Uni-M-S/Uni-M-S	Uni-M-S/Uni-M-S
5	Uni-M-S/Uni-S-M	Uni-M-S/Uni-S-M
6	Uni-S-M/Uni-S-M	Uni-S-M/Uni-S-M

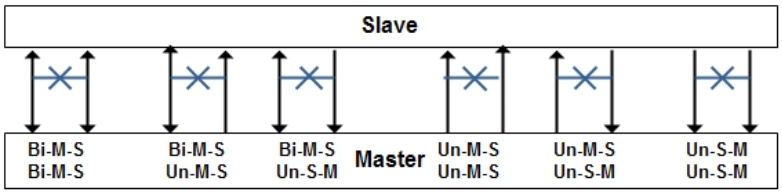


Figure 4.2: Wired-AND/OR for different interconnect types

on ↑ or ↕ from master to slave. Detection conditions for SA1 fault are similar to SA0 with minor changes; where fault sensitization requires forcing the line to have a value of 0 instead of 1.

Wired-AND (OR)

In wired-AND/OR faults, bridged interconnects may be of the same or different wire types. Tables 4.3 summarize the combinations between them which are also depicted in Figure 4.2. There are six distinguishable cases, which are short between (1) ↕ and ↕, (2) ↕ and ↑, (3) ↕ and ↓, (4) ↑ and ↑, (5) ↑ and ↓, and (6) ↓ and ↓. There are different fault models that represent bridging faults as discussed is section 3.4.1. Here, we are targeting the wired-AND/OR bridge fault models depicted in Figure 3.9(b).

To sensitize a bridge fault, two opposite values must be specified on each pair of lines. Fault sensitization for different bridged interconnects requires setting one line to a value of X that represent a logical value of (0 or 1) and the other line to a value of $\overline{X}$ that represent its complement (1 or 0). Fault propagation and line justification are similar to Table 4.1 where a propagated value of 0 indicates a wired-AND fault and a value of 1 indicates a wired-OR fault. For example, the detection of wired-AND/OR fault between two interconnects one of type ↕ and the other of type ↑, requires the following detection conditions steps illustrated in Figure 4.3:

1. Fault sensitization: Force both the lines ↕ and ↑ to have a value of X and $\overline{X}$ through the master, respectively.

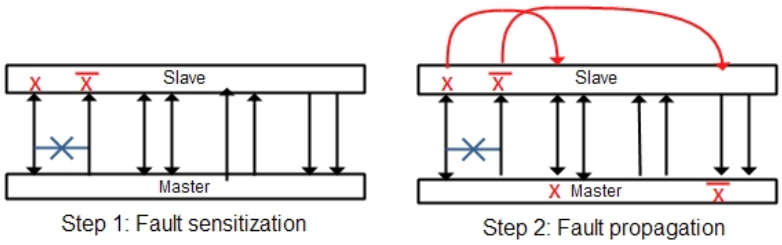


Figure 4.3: Wired-AND/OR detection conditions

Table 4.4: Fault sensitization for path delay faults

	Un-M-S $\uparrow$	**Un-S-M** $\downarrow$	**Bi-M-S** $\updownarrow$	
Fault sensitization	Set the line $\uparrow$ to have a transition from X to $\overline{X}$	Set the line $\downarrow$ to have a transition from X to $\overline{X}$	Tested in M-S direction: Set the line $\updownarrow$ to have a transition from X to $\overline{X}$	Tested in S-M direction: Set the line $\updownarrow$ to have a transition from X to $\overline{X}$

2. Fault propagation: Propagate back both values from slave to master; propagate the value on line $\updownarrow$ through $\updownarrow$ or $\downarrow$, and propagate the value on line $\uparrow$ using $\updownarrow$ or $\downarrow$.

3. Line justification: No line justification is needed since the sensitization occurred at the master.

4.2.2 Dynamic Faults

Dynamic faults are time dependent and are classified based on the number of interconnects that are involved. The detection conditions for each dynamic fault are explained below.

Single Line Faults

Here, faults are affected by a single line only, such as path delay and stuck open faults; they are described next.

Path Delay Fault

A path delay fault causes an additional unintended delay on a line without being impacted by neighbors. In this fault model, we assume that the delay exceeds the normal clock cycle with at most one clock cycle. To sensitize a path delay fault, a (0→1) and (1→0) transition must occur on each wire. Table 4.4 explains the fault sensitization for path delay faults for the different interconnects, where X represent a logical value of 0 or 1. Fault propagation and line justification go in a similar way as for the SA0/SA1 as shown in Table 4.1, but now with transitions.

Stuck Open Fault

This model is the same as the previous, except that the faulty interconnect is completely open. During short time intervals we assume that the non-driven part of the open does not change its value. The defect can be detected now by consecutively testing for a SA0 and SA1 or vice versa.

Multi Line Faults

Here, faults are affected by multiple lines, such as crosstalk faults, path delay faults with crosstalk, and stuck open faults with crosstalk; they are described next.

Crosstalk Fault

Crosstalk due to capacitive coupling between interconnects affects signal quality negatively. As described in Section 3.4.1 MAF model encompasses different faults to represent the crosstalk. Each has a different fault sensitization. They are:

- Positive glitch: Apply a (0→1) transition on aggressor(s) and keep the victim stable with 0.
- Negative glitch: Apply a (1→0) transition on aggressor(s) and keep the victim stable with 1.
- Falling delay: Apply a (0→1) transition on aggressor(s) and apply a (1→0) transition on the victim.
- Rising delay: Apply a (1→0) transition on aggressor(s) and apply (0→1) transition on the victim.

Victim line and aggressor line may be of any interconnect type; this gives 3^{1+K} possible combinations for a single victim with K neighbor aggressors. The fault sensitization in all cases requires a (X→$\overline{X}$) transition on the aggressor(s) and applying an opposite transition ($\overline{X}$→X) on the victim or keeping the victim stable with a value (X) similar to the initial value of aggressor. Fault propagation and line justification for the victim are in all cases similar as described in Table 4.1.

Path Delay Fault with Crosstalk

To increase the detection probability of path delay faults in the presence of crosstalk, aggressors must be set such that worst case stress conditions are applied on the victims. This is achieved by performing opposite transitions on the victim and its aggressors. Therefore, the fault sensitization is exactly the same as for the falling and rising delay of the MAF model.

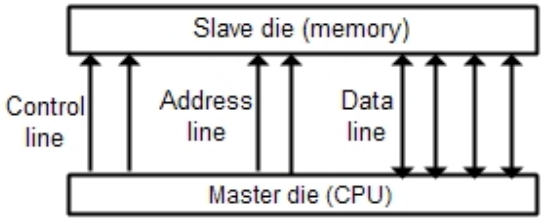


Figure 4.4: Memory stacked on logic

Stuck Open Fault with Crosstalk

The detection conditions for stuck open fault with crosstalk require maximum possible stress on the open line which can be applied either by keeping the value of the victim stable and creating a transition on the aggressors or by keeping the aggressors stable and creating a transition on the victim. Thus we have four possibilities of fault sensitization:

- Apply a (0→1) transition on the victim and keep the aggressor(s) stable with 0.
- Apply a (1→0) transition on the victim and keep the aggressor(s) stable with 1.
- Apply a (0→1) transition on aggressor(s) and keep the victim stable with 0.
- Apply a (1→0) transition on aggressor(s) and keep the victim stable with 1.

Fault propagation and line justification are similar to Table 4.1.

4.3 Specific Detection Conditions

In the previous section, we explained the detection conditions for different faults based on three different interconnect types; Bi-M-S, Un-M-S, and Un-S-M in general without restricting ourselves to memories. The restricted and more specific structure for memory stacked on logic is depicted in Figure 4.4. Here, Un-M-S represents control and address lines, and Bi-M-S interconnect represent bi-directional data lines. Note that in our considered memory architecture no Un-S-M lines are used. This simplifies the general detection conditions as Un-S-M lines are not used. Table 4.5 shows the fault propagation and line justification for the memory signals depicted in Figure 4.4. With respect to the controllability and observability of the TSVs for memory stacked on logic of Figure 4.4 we make the following assumptions:

- Data lines can be directly controlled and observed, either through master or through the output of the memory.
- Address lines can be directly controlled, and indirectly (through data lines) observed.
- Control lines are ignored and assumed to be tested implicitly.

In the next section, we convert the detection conditions into test operations.

Table 4.5: Fault detection steps (fault propagation and line justification) for applicable interconnects

	Un-M-S ↑	**Bi-M-S ↕**	
Fault propagation	Propagate the fault effect to the master using the line ↕	Tested in M-S direction: Propagate the fault effect to the master using the line ↕	Tested in S-M direction: No action required
Line justification	No action required	No action required	Justify the line ↕

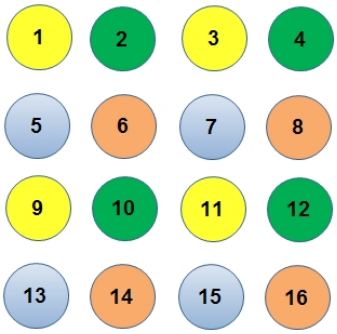


Figure 4.5: Interconnect structure in 3D-SICs for a 4x4 matrix of TSVs

4.4 Test Patterns

To identify defective ICs from fault-free ones each IC is tested for defects. During these tests, test patterns are applied that are derived from the detection conditions. By comparing the output of the applied test patterns with the golden reference (or fault-free output) defective ICs can be identified. The test patterns are derived from the fault models in such that they detect the faults of the fault model, i.e., the patterns sensitize the faults and propagate them to the output.

Each interconnect in a 3D-SIC is surrounded by multiple neighbors as depicted in Figure 4.5. We assume these interconnects to be grouped in an array to optimize the area. Each interconnect has three, five, or eight neighbors. Some faults, such as multi line dynamic faults are layout dependent; therefore, the test patterns change with different layout, but the approach remains the same.

In order to test the interconnects; test patterns are applied in the form of read write operations. Furthermore, in our examples we assume the following:

- We assume that the memory has a bi-directional 16-bit data bus and 16-bit address bus as depicted in Figure 4.6. Both the data bus data and address bus are assumed to have a 4 by 4 TSV layout (Figure 4.5).
- The Data Mask (DM) signal is a control signal which is used to mask bytes during a memory write operation.
- The memory is only index addressable and the DM signal can be used to write specific bytes of the index word.

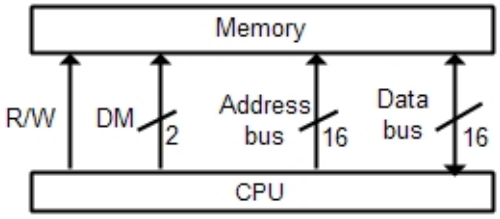


Figure 4.6: Interconnect for memory-on-logic

Table 4.6: Test patterns to detect SA0 fault at data lines

TP	Operation	Address	Data
TP1	W	AddressX	F F F F
TP2	R	AddressX	F F F F

- As interconnects are tested by applying functional memory operations, we ignore testing of control lines. They are assumed to be detected implicitly.

In general, we denote data width with L_d, address width with L_a, control width with L_c, read word operation with R, write word operation with W, read byte operation with R_b, write byte operation with W_b, and test pattern with TP.

4.4.1 Test Patterns for Static Faults

In this section, we are going to derive test patterns that are able to detect static faults SA0, SA1, wired-AND, and wired-OR. Moreover, a proof of correctness is given for each.

SA0 Fault

The test patterns to detect SA0 faults at data lines and address lines differs completely, both are discussed next.

Test patterns to detect SA0 fault at data line

Table 4.5 summarized fault propagation and line justification for SA0 fault at Bi-M-S $\updownarrow$, and Un-M-S $\uparrow$ lines. To generate proper test patterns it is required to sensitize the fault by forcing a 1 on all data lines from the slave side. This can be achieved by performing write and read operations as summarized in Table 4.6 where the address *AddressX* represents any valid memory address. First a pattern of all 1's (FFFF in hex) on the data line is written to a particular address (AddressX). Next, the same value is read from this address. SA0 fault will be detected as all 1's are expected at the output. This requires two memory instructions. The proof of the test patterns is shown in Table 4.7 where we assume that the Lowest Significant Bit (LSB) of data the line contains a SA0.

Table 4.7: Proof of test patterns to detect SA0 fault at data lines

TP	Operation	Address	Fault-free data	Faulty data
TP1	*W*	AddressX	F F F **F**	F F F **E**
TP2	*R*	AddressX	F F F **F**	F F F **E**

Table 4.8: Test patterns to detect SA0 fault at address lines

TP	Operation	Address	Data
TP1	*W*	0000 0000 0000 0000	Initial Data
TP2	*W*	0000 0000 0000 000**1**	$Data_1$
TP3	*W*	0000 0000 0000 00**1**0	$Data_2$
. . .	. . .	. . .	. . .
TP16	*W*	0**1**00 0000 0000 0000	$Data_{15}$
TP17	*W*	**1**000 0000 0000 0000	$Data_{16}$
TP18	*R*	0000 0000 0000 0000	Initial Data

Table 4.9: Proof test patterns to detect SA0 fault at address lines

TP	Operation	Fault-free address	Faulty address	Fault-free data	Faulty data
TP1	*W*	0000 0000 0000 0000	0000 0000 0000 0000	Initial Data	Initial Data
TP2	*W*	0000 0000 0000 000**1**	0000 0000 0000 000**0**	$Data_1$	$Data_1$
TP3	*W*	0000 0000 0000 00**1**0	0000 0000 0000 00**1**0	$Data_2$	$Data_2$
. . .	. . .	. . .	. . .	. . .	. . .
TP16	*W*	0**1**00 0000 0000 0000	0**1**00 0000 0000 0000	$Data_{15}$	$Data_{15}$
TP17	*W*	**1**000 0000 0000 0000	**1**000 0000 0000 0000	$Data_{16}$	$Data_{16}$
TP18	*R*	0000 0000 0000 0000	0000 0000 0000 0000	**Initial Data**	$Data_1$

Test patterns to detect SA0 fault at address line

Detecting SA0 faults at address lines is a bit more complex than detecting them at data lines. Faulty TSV address lines will affect the memory address. For example, writing the data pattern of all 1's to the address with all 1's and subsequently reading from the address with all 1's will not detect SA0's on the address lines, as the output in the fault free case is the same. This is because both read and write operations are affected in the same way by the SA0 fault. In order to test memory address lines each line should be tested separately. For example, by using a walking-1 sequence as depicted in Table 4.8. Detecting SA0 at address lines requires L_a+2 memory instructions. The proof of the test patterns are shown in Table 4.9 assuming that the LSB bit of the address line contains a SA0.

First, the memory is initialized by writing a particular data (*Initial Data*) to address cell 0 (all 0's). Note that the presence of a SA0 in the address line does not affect this initialization. By applying a walking-1 sequence on the address, a SA0 on the particular line where the bit is set to 1 of the walking sequence will make that address map to the initialization (all 0's address). In the example, data *Initial Data* will be overwritten by $Data_1$ during TP2. A read operation on the initialized address will detect any SA0 on the address lines at TP18. In order for this test to work successfully the following condition(s) must hold: $DataX \neq InitialData$, $1 \leq X \leq L_a$.

Table 4.10: Test patterns to detect SA1 fault at data lines

TP	Operation	Address	Data
TP1	W	AddressX	0 0 0 0
TP2	R	AddressX	0 0 0 0

Table 4.11: Proof test patterns to detect SA1 fault at data lines

TP	Operation	Address	Fault-free data	Faulty data
TP1	W	AddressX	0 0 0 **0**	0 0 0 **1**
TP2	R	AddressX	0 0 0 **0**	0 0 0 **1**

Table 4.12: Test patterns to detect SA1 fault at address lines

TP	Operation	Address	Data
TP1	W	1111 1111 1111 1111	Initia lData
TP2	W	1111 1111 1111 111**0**	$Data_1$
TP3	W	1111 1111 1111 11**0**1	$Data_2$
. . .	. . .	. . .	. . .
TP16	W	1**0**11 1111 1111 1111	$Data_{15}$
TP17	W	**0**111 1111 1111 1111	$Data_{16}$
TP18	R	1111 1111 1111 1111	Initial Data

SA1 Fault

To obtain the test patterns for SA1 fault a similar approach is used as for SA0. Again, a distinction is made between data and address lines.

Test patterns to detect SA1 faults at data line

Table 4.10 shows the test patterns required to detect SA1 fault at data lines, here all 1's of the SA0 fault on the data lines are replaced by all 0's. Detecting SA1 at data lines requires two memory instructions. Table 4.11 shows the proof of for this.

Test patterns to detect SA1 faults at address line

Table 4.12 shows the test patterns required to detect SA1 fault at address lines. Here, all address bits are flipped with respect to SA0 fault. Detecting SA1 faults at address lines requires also L_a+2 memory instructions. Table 4.13 shows the proof where we assume the LSB bit of the address line contains SA1 fault. In the Table, TP2 will cause $Data_1$ to overwrite *Initial Data*, therefore detectable at TP18.

Bridge Fault

A bridge fault may have a wired-AND or wired-OR behavior, and may happen between: (1) data lines, (2) between address lines, and (3) between data and address lines. Each category is described next.

Table 4.13: Proof test patterns to detect SA1 fault at address lines

TP	Operation	Fault-free address	Faulty address	Fault-free data	Faulty data
TP1	W	1111 1111 1111 1111	1111 1111 1111 1111	Initial Data	Initial Data
TP2	W	1111 1111 1111 111**0**	1111 1111 1111 111**1**	$Data_1$	$Data_1$
TP3	W	1111 1111 1111 11**0**1	1111 1111 1111 11**0**1	$Data_2$	$Data_2$
. . .	. . .	. . .	. . .	. . .	. . .
TP16	W	1**0**11 1111 1111 1111	1**0**11 1111 1111 1111	$Data_{15}$	$Data_{15}$
TP17	W	**0**111 1111 1111 1111	**0**111 1111 1111 1111	$Data_{16}$	$Data_{16}$
TP18	R	1111 1111 1111 1111	1111 1111 1111 1111	**Initial Data**	$Data_1$

Table 4.14: Test patterns to detect bridge fault at data lines

TP	Operation	Address	Data
TP1	W	AddressX	P_1
TP2	R	AddressX	P_1
TP3	W	AddressX	P_2
TP4	R	AddressX	P_2
. . .	. . .	. . .	. . .
$TP_{2*Log(Ld+2)-1}$	W	AddressX	$P_{Log(Ld+2)}$
$TP_{2*Log(Ld+2)}$	R	AddressX	$P_{Log(Ld+2)}$

(1) Test patterns to detect bridge faults between data lines

Bridge faults between data lines will cause one of the lines to flip based on the bridge type. Therefore, detecting bridge fault between data lines require that every pair of data lines must at least have either the combination 01 or 10 between the two lines. Modified counting sequence (MCS) satisfies the requirements of detecting bridge faults at data lines with Log(L_d+2) test patterns [81].

MCS has test vectors of the form P1=(1,0,1,0,1,0,1,...), P2=(0,1,1,0,0,1,1,...), P3= (0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,...) and so on. Therefore, Pj has the following form; a sequence of 0's of length $2^{(j-1)}$-1, followed by a sequence of 1's of length $2^{(j-1)}$, followed by a sequence of 0's of length $2^{(j-1)}$ and so on. Table 4.14 shows the test patterns for detecting bridge faults at data lines with 2* log(L_d+2) memory instructions where the address *AddressX* represents any memory address. A proof for such test patterns is not shown as it is proven in literature [81].

(2) Test patterns to detect bridge faults between address lines

Bridge faults between address lines cause one of the bit lines to flip, which may result in two behaviors:

1- Wired-AND bridge fault: Detecting bridge faulst between address lines with wired-AND behavior can be detected by applying a walking-1. This is similar to detecting SA0 fault at address lines we discussed previously. The test patterns are shown in Table 4.8. Any wired-AND bridge fault in the patterns TP2 till TP17 will overwrite the Initial Data of TP1. This fault will be detected by TP18, when the initialized value of index 0 is read.

Table 4.15: Test patterns to detect wired-AND bridge that flip data lines

TP	Operation	Address	Data
TP1	*W*	0000 0000 0000 0000	F F F F
TP2	*R*	0000 0000 0000 0000	F F F F

Table 4.16: Proof test patterns to detect wired-AND bridge that flip data lines

TP	Operation	Address	Fault-free data	Faulty data
TP1	*W*	0000 0000 0000 0000	F F F **F**	F F F **E**
TP2	*R*	0000 0000 0000 0000	F F F **F**	F F F **E**

2- Wired-OR bridge fault: Detecting bridge faults between address lines with wired-OR behavior can be detected by applying a walking-0. This is similar to detecting SA1 fault at address lines discussed previously in Table 4.12. Any wired-OR bridge fault in the patterns TP2 till TP17 will overwrite the Initial Data of TP1. This fault will be detected by TP18, when the initialized value of index 0 is read.

(3) Test patterns to detect bridge faults between data lines and address lines

Bridge faults between data lines and address lines may be wired-AND or wired-OR, and may flip data lines or address lines. In total, there are four different cases for which are described next.

1- Wired-AND bridge that flips a data line: Table 4.15 provides test patterns that detect wired-AND bridges that cause data lines to flip from 1 to 0. Any data bit that suffers from a wired-AND with address bits will cause the data to flip to zeros on the data side, which is easily detectable. These test patterns are similar to SA0 at data lines if the *AddressX* of Table 4.6 has the value 0. The proof of the test patterns is shown in Table 4.16 where it is assumed that the LSB bit of the data lines is bridged to the LSB bit of the address lines.

2- Wired-AND bridge that flips an address line: Table 4.17 provides the test patterns that are needed to detect wired-AND bridges that cause address bit lines to flip. A walking-1 pattern on the address lines ensures the detection of these types of defects. In TP1, the address consisting of all 0's is initialized with all 1's data (FFFF in hex). Note that for the initialization pattern (TP1) the address is not impacted in the presence of wired-AND defects. Any short on the walking-1 pattern (TP2 up to TP17) will overwrite the original initialization, i.e., in case a defect occurs. Therefore, the last read (TP18) from address 0 results in data all 1's data in the case of non-faulty interconnects and all 0's in case these types of faults are present. The test patterns in Table 4.17 are similar to the test patterns in Table 4.8 for detecting SA0 fault at address lines if *Initail Data* = FFFF and *DataX* = 0000, where $1 \leq X \leq L_a$. Table 4.18 shows the proof where it is assumed that the LSB bit of the data line and LSB bit of the address lines are bridged.

3-Wired-OR bridge that flips a data line: Table 4.19 provides the test patterns to detect wired-OR bridge that cause data lines to flip. Here, writing to the memory cell

Table 4.17: Test patterns to detect wired-AND bridge that flip address lines

TP	Operation	Address	Data
TP1	*W*	0000 0000 0000 0000	F F F F
TP2	*W*	0000 0000 0000 000**1**	0 0 0 0
TP3	*W*	0000 0000 0000 00**1**0	0 0 0 0
. . .	. . .	. . .	. . .
TP16	*W*	0**1**00 0000 0000 0000	0 0 0 0
TP17	*W*	**1**000 0000 0000 0000	0 0 0 0
TP18	*R*	0000 0000 0000 0000	F F F F

Table 4.18: Proof test patterns to detect wired-AND bridge that flip address lines

TP	Operation	Fault-free address	Faulty address	Fault-free data	Faulty data
TP1	*W*	0000 0000 0000 0000	0000 0000 0000 0000	Initial Data	Initial Data
TP2	*W*	0000 0000 0000 000**1**	0000 0000 0000 000**0**	0 0 0 0	0 0 0 0
TP3	*W*	0000 0000 0000 00**1**0	0000 0000 0000 00**1**0	0 0 0 0	0 0 0 0
. . .	. . .	. . .	. . .	. . .	. . .
TP16	*W*	0**1**00 0000 0000 0000	0**1**00 0000 0000 0000	0 0 0 0	0 0 0 0
TP17	*W*	**1**000 0000 0000 0000	**1**000 0000 0000 0000	0 0 0 0	0 0 0 0
TP18	*R*	0000 0000 0000 0000	0000 0000 0000 0000	**Initial Data**	**0 0 0 0**

Table 4.19: Test Patterns to detect wired-OR bridge that flip data lines

TP	Operation	Address	Data
TP1	*W*	1111 1111 1111 1111	0 0 0 0
TP2	*R*	1111 1111 1111 1111	0 0 0 0

Table 4.20: Proof test patterns to detect wired-OR bridge that flip data lines

TP	Operation	Address	Fault-free data	Faulty data
TP1	*W*	1111 1111 1111 1111	0 0 0 **0**	0 0 0 **1**
TP2	*R*	1111 1111 1111 1111	0 0 0 **0**	0 0 0 **1**

with address all 1's and setting the data lines to be all 0's ensures detecting these faults when we read from the same cell as bridges will cause data bits to flip to 1. These test patterns are similar to SA1 at data lines if the *AddressX* of Table 4.10 has the value of all 1's. Table 4.20 shows the proof where it is assumed that the LSB bit of the data line and LSB bit of the address lines are bridged.

4- Wired-OR bridge that flips an address line: Table 4.21 provides the test patterns needed to detect wired-OR bridges that cause the address bit lines to flip. A walking-0 pattern on the address lines ensures the detection of these types of defects. In TP1, the address of all 1's is initiated with all 0's data. Note that for the initialization pattern (TP1) the address of all 1's will not be impacted in the presence of these defects. Any short on the walking-0 pattern (TP2 up to TP17) will overwrite the original initialization, i.e., in case a defect occurs. Therefore, the last read (TP18) from address of all 1's result in data 0000 in case non-faulty interconnects and FFFF in case these types of faults are present. The test patterns in Table 4.21 are similar to the test patterns in Table 4.12 for detecting SA1 fault at address lines if *Initail Data* = 0000 and *DataX* =

Table 4.21: Test patterns to detect wired-OR bridge that flip address lines

TP	Operation	Address	Data
TP1	W	1111 1111 1111 1111	0000
TP2	W	1111 1111 1111 111**0**	F F F F
TP3	W	1111 1111 1111 11**0**1	F F F F
. . .	. . .	. . .	. . .
TP16	W	1**0**11 1111 1111 1111	F F F F
TP17	W	**0**111 1111 1111 1111	F F F F
TP18	R	1111 1111 1111 1111	0 0 0 0

Table 4.22: Proof test patterns to detect wired-OR bridge that flip address lines

TP	Operation	Fault-free address	Faulty address	Fault-free data	Faulty data
TP1	W	1111 1111 1111 1111	1111 1111 1111 1111	Initial Data	Initial Data
TP2	W	1111 1111 1111 111**0**	1111 1111 1111 111**1**	F F F F	F F F F
TP3	W	1111 1111 1111 11**0**1	1111 1111 1111 11**0**1	F F F F	F F F F
. . .	. . .	. . .	. . .	. . .	. . .
TP16	W	1**0**11 1111 1111 1111	1**0**11 1111 1111 1111	F F F F	F F F F
TP17	W	**0**111 1111 1111 1111	**0**111 1111 1111 1111	F F F F	F F F F
TP18	R	1111 1111 1111 1111	1111 1111 1111 1111	**Initial Data**	**F F F F**

FFFF, where 1$\leq$ X $\leq$ L_a. Table 4.22 shows the proof where it is assumed that the LSB bit of the data line and LSB bit of the address lines are bridged.

Static Faults Summary

The number of memory operations required for detecting all possible static faults are summarized in Table 4.23. However, these patterns can be optimized. Table 4.24 shows which test patterns lead to the same fault detection under specific conditions for certain addresses and data values. For example, the test patterns for SA0 at data lines in Table 4.6 equal the pattern set for wired-AND of Table 4.15 if $Address_X$ of Table 4.6 is set to all 0's. The total number of test patterns for detecting static faults is 2*Log(L_d+2)+2*L_d+8.

4.4.2 Test Patterns for Dynamic Faults

This section discusses the test patterns for detecting dynamic faults. Dynamic faults may require several test patterns for fault sensitization, as they are time dependent. Dynamic faults can be classified based on the number of interconnects: single line faults and multi line faults. Single line faults include path delay faults (PDF) and stuck open faults (SOF). Multi line faults include crosstalk faults, PDF with crosstalk, and SOF with crosstalk. The test patterns for detecting all dynamic faults at data lines and address lines are discussed next.

Single Line Faults

Single line faults that include path delay and stuck open faults will be discussed next.

Table 4.23: Summary of test patterns for static faults

Type of Static Fault	**# of memory operations**
SA1 data line	2
SA1 Address line	L_a+2
SA0 data line	2
SA0 Address line	L_a+2
Bridge Fault at Data line (Wired-AND/OR)	2*Log(L_d+2)
Bridge Fault at Address line (Wired-AND)	L_a+2
Bridge Fault at Address line (Wired-OR)	L_a+2
Wired-AND bridge between data and address lines that flip data line	2
Wired-AND bridge between data and address lines that flip address line	L_a+2
Wired-OR bridge between data and address lines that flip data line	2
Wired-OR bridge between data and address lines that flip address line	L_a+2

Table 4.24: Summary of optimized test patterns for static faults

Equivalent Faults	**Condition**	**# of operations**
SA0 at data lines/Wired-AND bridge between data and address lines that flips data line	Set $Address_X$ in Table 4.6 to all 1's as in Table 4.15	2
SA1 at data lines/Wired-OR bridge between data and address lines that flip data line	Set $Address_X$ in Table 4.10 to all 0's as in Table 4.19	2
SA0 Address line/Wired-AND bridge between data and address lines that flip address line/Bridge Fault at Address line (Wired-AND)	1- Set Initial$_D atainTable$ 4.8$toall1'sasinTable$ 4.17 2-Set $Data_X$ in Table 4.8 to all 0's for all $Data_X$ as in Table 4.17	L_a+2
SA1 Address line/Wired-OR bridge between data and address lines that flip address line/Bridge Fault at Address line (Wired-OR)	1-Set Initial$_D atainTable$ 4.12$toall0'sasinTable$ 4.21 2-Set $Data_X$ in Table 4.12 to all 1's for all $Data_X$ as in Table 4.21	L_a+2
Bridge Fault at Data line (Wired-AND/OR)		2*Log(L_d+2)

(1) Path Delay Fault (PDF)

This single line fault is due to a partial resistive open defect with low to moderate resistance value. Delays on a line with more than one clock cycle are tested as complete open lines, which are discussed later. The test patterns to detect PDF faults require both 0→1 and 1→0 transitions (see detection conditions in Section 4.2.2).

Test patterns to detect PDF faults at data lines

Interconnects between the master and slave do not only include TSVs, but they also need drivers in both dies as depicted in Figure 4.7. These drivers charge or discharge the lines based on the desired value. Therefore, testing must also consider these components as delays can also originate from defects in these drivers. This means that the bidirectional data lines must be tested in both directions; from master to slave and from slave to master. A defect in one of the pull-up networks will cause 0→1 transition fault, and a defect in the pull-down networks will cause 1→0 transition faults. Therefore, we have the following test cases:

- 0→1 transition from master to slave.

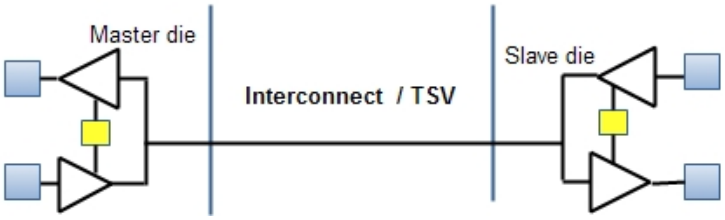


Figure 4.7: Data line with read and write drivers

Table 4.25: Test patterns for PDF faults at data lines

TP	Operation	Address	Data
TP1	Write	$Address_1$	0000 0000 0000 0000
TP2	Write	$Address_2$	1111 1111 1111 1111
TP3	Write	$Address_1$	0000 0000 0000 0000
TP4	Read	$Address_1$	0000 0000 0000 0000
TP5	Read	$Address_2$	1111 1111 1111 111
TP6	Read	$Address_1$	0000 0000 0000 0000

- 1→0 transition from master to slave.
- 0→1 transition from slave to master.
- 1→0 transition from slave to master.

Tests patterns as shown in Table 4.25 satisfy the previous requirements; the first part of the table (TP1-TP3) have both transitions from master to slave and the second part of the table (TP4-TP6) have both transitions from slave to master. The proof that these six test patterns detect PDF faults are shown in Table 4.26 for all four transitions (the six test patterns are replicated four times). For simplicity, we use only the 4 LSB bits of the data. The first time the patterns are used to demonstrate a 0→1 transition fault from master to slave. TP1 writes a value of all 0's to data lines then TP2 makes a 0→1 transition and writes a value of all 1's to $Address_2$. With a 0→1 PDF fault at the LSB bit of the data lines from master to slave, the value written will be all 1's except a 0 for the LSB bit. This will be detected during the read operation in TP4. The other transition faults work in a similar manner.

In case the read and write drivers in both dies are fault-free, for example when tested in the pre-bond phase, testing requires only a (0→1) transition from master to slave and 0→1 transition from slave to master, thus, skipping TP3 and TP6 in Table 4.25.

Test patterns to detect PDF faults at address lines

Detecting a path delay fault at address lines requires both 0→1 and 1→0 transitions on address lines. Here, we assume also a maximum delay of one clock cycle. A delay larger than one clock cycle requires the delay to be tested as stuck open fault. The test patterns are shown in Table 4.27, where the first part of the table (TP1-TP5) tests for 0→1 transition and the second part (TP6-TP10) tests for 1→0 transition.

Table 4.26: Proof test patterns for PDF faults at data lines

TP	Operation	**Address**	**Fault free Data**	**Faulty Data**	**Note**
TP1	Write	$Address_1$	0000	0000	0→1 transition
TP2	Write	$Address_2$	1111	111**0**	fault from master
TP3	Write	$Address_1$	0000	0000	to slave
TP4	Read	$Address_1$	0000	0000	
TP5	Read	$Address_2$	1111	111**0**	Fault detected
TP6	Read	$Address_1$	0000	0000	
TP1	Write	$Address_1$	0000	0000	1→0 transition
TP2	Write	$Address_2$	1111	1111	fault from master
TP3	Write	$Address_1$	0000	000**1**	to slave
TP4	Read	$Address_1$	0000	000**1**	Fault detected
TP5	Read	$Address_2$	1111	1111	
TP6	Read	$Address_1$	0000	000**1**	
TP1	Write	$Address_1$	0000	0000	0→1 transition
TP2	Write	$Address_2$	1111	1111	fault from slave
TP3	Write	$Address_1$	0000	0000	to master
TP4	Read	$Address_1$	0000	0000	
TP5	Read	$Address_2$	1111	111**0**	Fault detected
TP6	Read	$Address_1$	0000	0000	
TP1	Write	$Address_1$	0000	0000	1→0 transition
TP2	Write	$Address_2$	1111	1111	fault from slave
TP3	Write	$Address_1$	0000	0000	to master
TP4	Read	$Address_1$	0000	0000	
TP5	Read	$Address_2$	1111	1111	
TP6	Read	$Address_1$	0000	000**1**	Fault detected

With the presence of a PDF fault at the address lines, TP1 and TP2 ensure a fault-free initialization. The second write (TP2) will initialize address with all 1's to Data3, even if the previous operation failed due to the presence of a PDF fault. A 0→1 transition is subsequently created by TP3 and TP4. In case any bit fails to make the transition, TP5 will read a wrong value $Data_3$ as $Data_2$ is expected. Therefore, $Data_2$ and $Data_3$ must be different. Detecting a PDF with 1→0 transition fault using TP6-TP10 goes in a similar way as TP1-TP5, but with flipped address bits. The proof for such test patterns are shown in Table 4.28. For simplicity, again we use only the 4 LSB address bits.

(2) Stuck Open Fault (SOF)

The stuck open fault is considered as an extreme case of a partial resistive fault in which the line is completely open. This fault may happen at data lines or address lines; both are described below.

Test patterns to detect SOF faults at data lines

Detecting stuck open faults at data lines as shown in Figure 4.8 requires writing a value of all 0's and all 1's to data lines. The test patterns are shown in Table 4.29. It contains

Table 4.27: Test patterns for PDF faults at address lines

TP	Operation	Address	Data	Note
TP1	Write	1111 1111 1111 1111	Data3	
TP2	Write	1111 1111 1111 1111	Data3	Initialization
TP3	Write	0000 0000 0000 0000	Data1	0 to 1 transition
TP4	Write	1111 1111 1111 1111	Data2	
TP5	Read	1111 1111 1111 1111	Data2	
TP6	Write	0000 0000 0000 0000	Data3	
TP7	Write	0000 0000 0000 0000	Data3	Initialization
TP8	Write	1111 1111 1111 1111	Data1	1 to 0 transition
TP9	Write	0000 0000 0000 0000	Data2	
TP10	Read	0000 0000 0000 0000	Data2	

Table 4.28: Proof of test patterns for PDF faults at address lines

Operation	Data	Fault free address	Faulty address	Fault free data	Faulty data	Note
Write	Data3	1111	111x	—	—	Initialization
Write	Data3	1111	1111	—	—	
Write	Data1	0000	0000	—	—	Fault on 0→1
Write	Data2	1111	111**0**	—	—	transition on LSB
Read	—	1111	1111	**Data2**	**Data3**	Fault detected
Write	Data3	0000	000x	—	—	Initialization
Write	Data3	0000	0000	—	—	
Write	Data1	1111	1111	—	—	Fault on 1→0
Write	Data2	0000	000**1**	—	—	transition on LSB
Read	—	0000	0000	**Data2**	**Data3**	Fault detected

two write operations (write all 1's subsequently followed by write all 0's) and two read operations to verify them. During the two read operations (TP3 and TP4), we assume that the floating values could be one of the following three:

1. The line is stable with value **0** during both read operations. Therefore, TP4 detects this fault.

2. The line is stable with value **1** during both read operations. Therefore, TP3 detects this fault. This value 1 could actually be charge that was left from the write operation TP2.

3. The line has leaking value (initially 1 during TP3 which goes down to 0 during TP4). This fault is detected by both TP3 and TP4.

In case the line has a rising value (initially 0 during TP3 which goes up to 1 during TP4), the test cannot detect the defect; however, the probability that a floating line will make this transition is unlikely. The proof of test patterns for detecting SOF at data lines is shown in Table 4.30. The table shows three replications of the test patterns and shows the proofs for a stable float 0, stable float 1, and for a leaking float, for a fault at the LSB bit position.

Test patterns to detect SOF faults at address lines

A stuck open fault at address lines can be tested based on walking patterns. Similarly, as for the data lines, two 0→1 transitions are created. One with write values and one

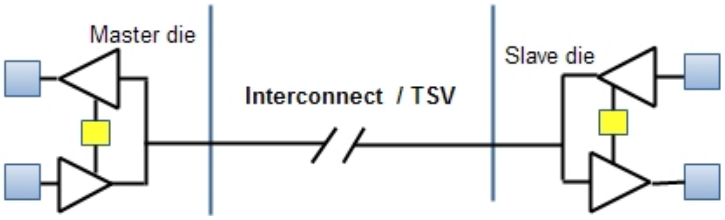


Figure 4.8: Data line with stuck open fault

Table 4.29: Test patterns for SOF faults at data lines

TP	Operation	Address	Data
TP1	Write	$Address_1$	0000 0000 0000 0000
TP2	Write	$Address_2$	1111 1111 1111 1111
TP3	Read	$Address_1$	0000 0000 0000 0000
TP4	Read	$Address_2$	1111 1111 1111 1111

Table 4.30: Proof patterns for SOF faults at data lines

		Fault-free writing		Faulty writing		Reading		
TP	**Operation**	**Address**	**Master/Slave**	**Master**	**Slave**	**Fault free data**	**Faulty data**	**Note**
TP1	Write	$Address_1$	0000	0000	000X			LSB is open and
TP2	Write	$Address_2$	1111	1111	111X			stable at 0
TP3	Read	$Address_1$	—	—	—	0000	0000	
TP4	Read	$Address_2$	—	—	—	111**1**	111**0**	Fault detected
TP1	Write	$Address_1$	0000	0000	000X			LSB is open and
TP2	Write	$Address_2$	1111	1111	111X			stable at 1
TP3	Read	$Address_1$	—	—	—	000**0**	000**1**	Fault detected
TP4	Read	$Address_2$	—	—	—	1111	1111	
TP1	Write	$Address_1$	0000	0000	000X			LSB is open and
TP2	Write	$Address_2$	1111	1111	111X			have 1→0
TP3	Read	$Address_1$	—	—	—	0000	000**1**	Fault detected
TP4	Read	$Address_2$	—	—	—	1111	111**0**	Fault detected

Table 4.31: Test patterns for SOF faults at address lines

TP	Operation	Address	Data
TP1	Write	0000 0000 0000 000**0**	Data1
TP2	Write	0000 0000 0000 000**1**	Data2
TP3	Read	0000 0000 0000 000**0**	Data1
TP4	Read	0000 0000 0000 000**1**	Data2

with read values. However, here each bit line must be tested separately as shown in Table 4.31. Any float on an address lines will be detected due to the quick transition made on them. The proof for an open fault at the LSB bit of the address lines is shown in Table 4.32 for a stable float 0 , stable float 1, and a leaking float (three possible cases).

Multi Line Faults

Based on the TSV array structure of 3D-SICs depicted in Figure 4.5, each interconnect may have coupling with its direct neighbors. Interconnects that are far away from each other are assumed to have no coupling and thus can be tested simultaneously in a group. We assume that only direct neighboring TSVs impact each other. Therefore,

Table 4.32: Proof test patterns for (SOF) faults at address lines

TP	Operation	Data	Fault free address	Faulty address	Fault free data	Faulty data	Note
TP1	Write	Data1	0000	0000	—	—	open line fault
TP2	Write	Data2	000**1**	0000	—	—	with stable 0
TP2	Read	—	0000	0000	**Data1**	**Data2**	Fault detected
TP4	Read	—	0001	0000	Data2	Data2	—
TP1	Write	Data1	0000	0001	—	—	open line fault
TP2	Write	Data2	000**1**	000**1**	—	—	with stable 1
TP3	Read		000**0**	000**1**	**Data1**	**Data2**	Fault detected
TP4	Read		0001	0001	Data2	Data2	—
TP1	Write	Data1	0000	0001	—	—	open line fault
TP2	Write	Data2	000**1**	000**1**	—	—	1→0 leaking at TP3→TP4
TP3	Read		000**0**	000**1**	**Data1**	**Data2**	Fault detected
TP4	Read		000**1**	000**0**	**Data2**	**XXXX**	—
TP1	Write	Data1	0000	0001	—	—	open line fault
TP2	Write	Data2	000**1**	000**1**	—	—	1→0 leaking at TP2→TP3
TP3	Read		0000	0000	**Data1**	**XXXX**	Fault detected
TP4	Read		000**1**	000**0**	**Data2**	**XXXX**	—
TP1	Write	Data1	0000	0001	—	—	open line fault
TP2	Write	Data2	000**1**	0000	—	—	1→0 leaking at TP1→TP2
TP3	Read		0000	0000	**Data1**	**Data2**	Fault detected
TP4	Read		000**1**	000**0**	Data2	Data2	—

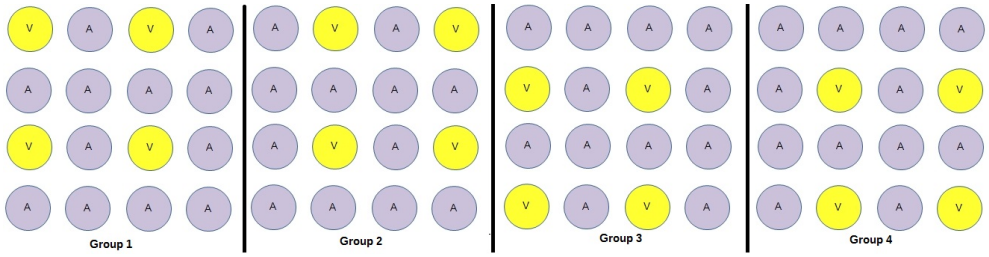


Figure 4.9: Interconnect groups in 3D-SICs for a 4x4 matrix of TSVs

interconnects may be divided into four groups as depicted in Figure 4.9. All victims that are part of the same group can be tested in parallel. Next, the three multi line faults will be described, which are (1) crosstalk fault, (2) path delay fault with crosstalk, and (3) stuck open fault with crosstalk. We assume that data lines and address lines are separated without coupling between them. Therefore, each have a TSV array as depicted in Figure 4.5.

(1) Crosstalk Fault

The crosstalk fault shown in Figure 3.13(d) can be modeled by the Maximum Aggressor Fault (MAF) model. The MAF model, shown in Figure 4.10, has different faults: (1) glitch-up, (2) glitch-down, (3) falling delay, and (4) rising delay. The required transitions on victim and aggressor lines to activate the faults are given in Table 4.33. Each fault has a specific fault behavior, while all of them represent the same phenomena. Therefore, selecting the minimum number of faults that reflect the problem will optimize the test. As detecting delay faults is easier than detecting glitch faults. Both the rising and the falling delay will be discusses next based on test patterns shown in Table 4.34. This Table shows for each of the 16 TSVs the transitions that must occur for victims of the

first group of Figure 4.9 .

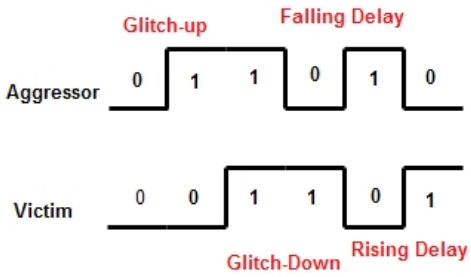


Figure 4.10: Maximum Aggressor Fault (MAF) model

Table 4.33: Crosstalk fault models and the transition on victim and aggressors

Victim		Aggressor		
From	**To**	**From**	**To**	**Fault**
0	0	0	1	Glitch-Up
0	1	1	1	—
1	1	1	0	Glitch-Down
1	0	0	1	Falling Delay
0	1	1	0	Rising Delay

Test patterns to detect falling delay at data lines

Test patterns to detect a falling delay at the data lines are shown in Table 4.35. The test patterns belong to the first group. Similarly, patterns can be created for the other three groups. TP1 and TP2 create a 1→0 transition on the victim data lines and 0→1 transitions on the aggressor data lines. We demonstrate the detection by assuming a fault in one of the lines. For example, in case the LSB bit of the data lines fails to make the transition due to the neighbors, TP4 will read a wrong value. The proof of this is shown in Table 4.36. For simplicity, we agian use only the 4 LSB bite of the data. Note that the transitions are made in both directions from master to slave and vice versa inspired by Figure 4.8.

Test patterns to detect falling delay fault at address lines

Test patterns to detect falling delay fault at the LSB bit of the address lines are show in Table 4.37. TP1 initialize an address of all 1's with a known data (*Data3*). TP1 and TP2 create a (1→0) transition on the victim address line and (0→1) transitions on the aggressor data lines. In case LSB of address lines fails to make the transition because of the opposite transition of the neighbors, TP3 overwrites the original initialization, therefore, the read operation (TP4) from an address of all 1's result in data value of *Data3* in case of non-faulty interconnects and data value of *Data2* in case this type of faults present. The proof of these test patterns is shown in Table 4.38. For simplicity, we use only the 4 LSB of the data.

Table 4.34: Test patterns for crosstalk fault/1^{st} group

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}
1	0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0	0
0	1	**0**	1	1	1	1	1	**0**	1	**0**	1	1	1	1	1
1	0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0	0

Table 4.35: Test patterns to detect falling faults at data lines

TP	Operation	Address	Data
TP1	Write	$Address_1$	**101**0 0000 **101**0 0000
TP2	Write	$Address_2$	**010**1 1111 **010**1 1111
TP3	Read	$Address_1$	**101**0 0000 **101**0 0000
TP4	Read	$Address_2$	**010**1 1111 **010**1 1111
. . .	. . .	. . .	. . .
TP1	Write	$Address_1$	**101**0 0000 **101**0 0000
TP2	Write	$Address_2$	**010**1 1111 **010**1 1111
TP3	Read	$Address_1$	**101**0 0000 **101**0 0000
TP4	Read	$Address_2$	**010**1 1111 **010**1 1111

Table 4.36: Proof of test patterns for falling delay faults at data lines

TP	Operation	Address	Fault free Data	Faulty Data	Note
TP1	Write	$Address_1$	0001	0001	Falling delay
TP2	Write	$Address_2$	111**0**	111**1**	
TP3	Read	$Address_2$	111**0**	111**1**	Fault detected

Table 4.37: Test patterns to detect falling delay fault faults at address lines

TP	Operation	Address	Data	Note
TP1	Write	1111 1111 1111 1111	Data3	Initialization
TP2	Write	0000 0000 0000 000**1**	Data1	—
TP3	Write	1111 1111 1111 111**0**	Data2	
TP4	Read	1111 1111 1111 1111	Data3	Data2 if faulty

Table 4.38: Proof of test patterns for falling delay faults at address lines

TP	Operation	Address	Fault free address	Faulty address	Fault free data	Faulty data	Note
TP1	Write	Data3	1111	1111	—	—	Initialization
TP2	Write	Data1	0001	0001	—	—	—
TP3	Write	Data2	111**0**	111**1**	—	—	—
TP4	Read	—	1111	1111	**Data3**	**Data2**	Fault detected

Test patterns to detect rising delay fault at data lines

Table 4.39 shows the test patterns required to detect rising delay fault at the LSB bit of the data lines, here all the transitions of the victim line and aggressor lines are flipped compared to the falling delay fault on the data lines. Table 4.40 shows the proof of the test patterns.

Table 4.39: Test patterns to detect rising delay faults at data lines

TP	Operation	Address	Data
TP1	Write	Address1	1111 1111 1111 111**0**
TP2	Write	Address2	0000 0000 0000 000**1**
TP3	Read	Address2	0000 0000 0000 0001

Table 4.40: Proof of test patterns for rising delay faults at data lines

TP	Operation	Address	Fault free Data	Faulty Data	Note
TP1	Write	Address1	1110	1110	Rising delay
TP2	Write	Address2	0001	0000	
TP3	Read	Address2	000**1**	000**0**	Fault detected

Table 4.41: Test patterns for rising delay faults at address lines

TP	Operation	Address	Data	Note
TP1	Write	0000 0000 0000 0000	Data3	Initialization
TP2	Write	1111 1111 1111 111**0**	Data1	—
TP3	Write	0000 0000 0000 000**1**	Data2	
TP3	Read	0000 0000 0000 0000	Data3	Data2 if faulty

Table 4.42: Proof of test patterns for rising delay faults at address lines

TP	Operation	Address	Fault free address	Faulty address	Fault free data	Faulty data	Note
TP1	Write	Data3	0000	0000	—	—	Initialization
TP2	Write	Data1	1110	1110	—	—	—
TP3	Write	Data2	000**1**	000**0**	—	—	—
TP4	Read	—	0000	0000	**Data3**	**Data2**	Fault detected

Test patterns to detect rising delay fault at address lines

Table 4.41 shows the test patterns required to detect rising delay fault at the LSB bit of the address lines, here all the transitions of the victim line and aggressor lines are flipped compared to the falling delay fault on the address lines. Table 4.42 shows the proof of the test patterns.

Multiple data lines can be tested together as one group because a defected data line will be detected directly through the read value and will not mask another data line. In 3D-SICs, as shown in Figure 4.5, data lines are testable in four groups. The test patterns for crosstalk fault detection at one group of data lines are shown in Table 4.43. The write operations are not directly verified by a read operation, for example, after writing Address2 (TP2), we continue first with writing Address3 (TP3). By start reading after finish writing, we test the data lines in both directions using the same test patterns. Testing multiple address lines at a time as one group is not possible because each possible faulty address line will map to different address, therefore each address line must be tested separately based on walking-pattern. The number of memory operations to detect crosstalk at data lines is 20 operations, and $8*L_a$ memory operations for detecting address line faults.

Table 4.43: Optimized test patterns for crosstalk faults at data lines

TP	**Operation**	**Address**	**Data**
TP1	Write	Address1	**101**0 0000 **101**0 0000
TP2	Write	Address2	**010**1 1111 **010**1 1111
TP3	Write	Address3	**101**0 0000 **101**0 0000
TP4	Read	Address2	0101 1111 0101 1111
TP5	Read	Address3	1010 0000 1010 0000

(2) Path Delay Fault (PDF) with Crosstalk

Test patterns for detecting path delay fault with crosstalk require maximum possible stress. The maximum possible stress can be achieved through forcing an (X→$\overline{X}$) transition on the victim line and an opposite transition ($\overline{X}$→ X) on the aggressor lines, where X represents a logical value of 0 or 1. The test patterns to detect this fault are similar to the test patterns to detect falling delay and rising delay faults which we discussed.

(3) Stuck Open Fault (SOF) with Crosstalk

Testing interconnects for stuck open fault with crosstalk requires applying maximum possible stress which can be tested either be keeping the victim line stable and make a transition on aggressor lines or by keeping the aggressor lines stable and making a transition on the victim line. The different choices for applying transition on lines are depicted in Figure 4.11. Figure 4.11(A) shows the victim line stable with a value of 0 and the aggressor lines have a (0→1) transition. Figure 4.11(B) shows the victim line stable with a value of 1 and the aggressor lines have a (1→0) transition. Figure 4.11(C) shows the victim with a (0→1) transition while the aggressor lines are stable with a value of 0. Figure 4.11(D) shows the victim with a (1→0) transition while the aggressor lines are stable with a value of 1.

The test patterns for the different possibilities for applying transitions are shown in Tables A.1, A.2, A.3, and A.4. For the test patterns to detect SOF fault with crosstalk at data lines and address lines, we will consider the test patterns in Table A.4 that make transition on victim while keeping the aggressors stable because this represent the case with the least changing in aggressors between consecutive test patterns to make the lines stable as long as possible in order to increase the possibility of fault detection.

Test patterns to detect (SOF) with crosstalk at data lines

As depicted in Figure 4.12 the open data lines have a crosstalk capacitance during the write operations different than the crosstalk capacitance during the read operation. Therefore, both cases must be tested. Table 4.44 shows such test patterns. Table 4.45 show the proof of the test patterns.

TP1-TP4 have a (0→1) transition on the victim line and the aggressor lines are stable at 0. Write operation (TP1 and TP2) make the transition from master to slave while read operation (TP3 and TP4) make the transition from slave to master. TP1 will cause the victim data line to have a value of 0 at master side, and at the slave side the data value will be 0 due to coupling with neighbors. TP2 will cause the victim data line to

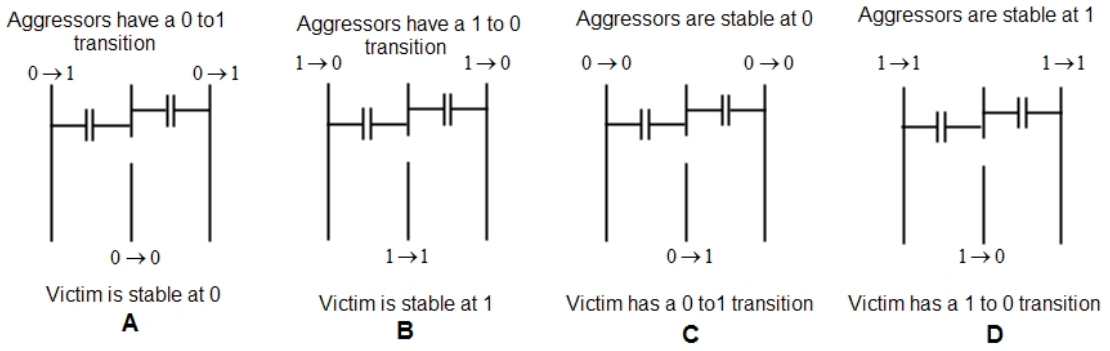


Figure 4.11: Stuck open fault with crosstalk

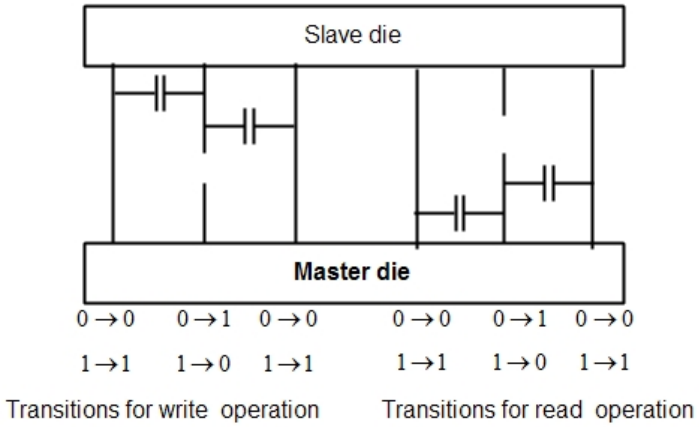


Figure 4.12: Stuck open data line with crosstalk

have a value of 1 at master side while the slave side will have a value of 0 due to coupling with neighbors. Reading from memory (TP3) will give a value of 0 at the victim line due to the coupling with neighbors. Reading from memory (TP4) will give a value of 0 at the victim which is due to coupling with neighbors and differs from the expected value. Thus the fault is detected.

TP5-TP8 have a (1→0) transition on the victim line and the aggressor lines are stable at 1. Write operation (TP5 and TP6) make the transition from master to slave while read operation (TP7 and TP8) make the transition from slave to master. TP5 will cause the victim data line to have a value of 1 at master side, and at the slave side the data value will be 1 due to coupling with neighbors. TP6 will cause the victim data line to value of 0 at master side but the slave side will have a value of 1 due to coupling with neighbors. Reading from memory (TP7) will give a value of 1 at the victim line which is due to the coupling with neighbors. Reading from memory (TP8) will give a value of 1 at the victim which is due to coupling with neighbors and differs from the expected value. Thus the fault is detected. Multiple data lines can be tested together as one group, and Table 4.46 shows the test patterns for one group.

Table 4.44: Test patterns for (SOF) with crosstalk at data lines

TP	Operation	Address	Data
TP1	Write	Address1	00 0 00
TP2	Write	Address2	00 1 00
TP3	Read	Address1	00 0 00
TP4	Read	Address2	00 1 00
TP5	Write	Address3	11 1 11
TP6	Write	Address4	11 0 11
TP7	Read	Address3	11 1 11
TP8	Read	Address4	11 0 11

Table 4.45: Proof of test patterns for (SOF) with crosstalk at data lines

TP	Operation	Address	Fault free Data	Faulty Data	Note
TP1	Write	Address1	00 0 00	00 0 00	
TP2	Write	Address2	00 1 00	00 0 00	
TP3	Read	Address1	00 0 00	00 0 00	
TP4	Read	Address2	00 **1** 00	00 **0** 00	Fault detected
TP5	Write	Address3	11 1 11	11 1 11	
TP6	Write	Address4	11 0 11	11 1 11	
TP7	Read	Address3	11 1 11	11 1 11	
TP8	Read	Address4	11 **0** 11	11 **1** 11	Fault detected

Table 4.46: Test patterns for (SOF) with crosstalk at data lines/ 1st group

TP	Operation	Address	Data
TP1	Write	Address1	**000**0 0000 **000**0 0000
TP2	Write	Address2	**101**0 0000 **101**0 0000
TP3	Read	Address1	0000 0000 0000 0000
TP4	Read	Address2	1010 0000 1010 0000
TP5	Write	Address1	**111**1 1111 **111**1 1111
TP6	Write	Address2	**010**1 1111 **010**1 1111
TP7	Read	Address1	1111 1111 1111 1111
TP8	Read	Address2	0101 1111 0101 1111

Test patterns to detect (SOF) with crosstalk at address lines

Testing multiple address lines at the same time is not possible which imposes testing each address line individually based on walking-patterns. Table 4.47 show the test pattern to detect SOF with crosstalk at single address line. TP1 initialize the memory by writing a specific data value (*Data1*) to a fault-free address (all 0's). TP2 after TP1 makes a (0→1) transition on the victim address line while the aggressor lines are stable at 0. TP3 will read a value of *Data1* for fault-free case, and *Data2* if the victim line failed to make (0→1) transition. TP4 initialize the memory by writing a specific data value (*Data3*) to a fault-free address (all 1's). TP5 after TP4 makes a (1→0) transition on the victim address line. TP6 will read a value of *Data3* for fault-free case, and *Data4* if the victim line failed to make (1→0) transition. Table 4.48 shows a proof of the test patterns.

Table 4.47: Test patterns for stuck open fault with crosstalk faults at address lines

TP	Operation	Address	Data
TP1	Write	00 0 00	Data1
TP2	Write	00 1 00	Data2
TP3	Read	00 0 00	Data1
TP4	Write	11 1 11	Data3
TP5	Write	11 0 11	Data4
TP6	Read	11 1 11	Data3

Table 4.48: Proof of test patterns for (SOF) with crosstalk at address lines

TP	Operation	Writing data	Fault free address	Faulty address	Fault free data	Faulty data	Note
TP1	Write	Data1	00 0 00	00 0 00	–	–	
TP2	Write	Data2	00 **1** 00	00 **0** 00	–	–	
TP3	Read	—	00 0 00	00 0 00	**Data1**	**Data2**	Fault detected
TP4	Write	Data3	11 1 11	11 1 11	–	–	
TP5	Write	Data4	11 **0** 11	11 **1** 11	–	–	
TP6	Read	—	11 0 11	11 1 11	**Data3**	**Data4**	Fault detected

Byte Addressable Memory

During the previous discussions, the test patterns were based on the assumption of indexed memory. In indexed memory, memory operations can read and write complete words. Moreover, the Data Mask control signal is used to mask data bytes during memory write operation. However, some memories may be byte addressable where a memory read and write operations can handle individual bytes instead of complete words. The test patterns based on byte addressable memory does not differ much from the test patterns of the indexed memory, following is an explanation:

1. Test patterns to detect faults at data lines such as Table 4.6, 4.10, 4.14 and others have the freedom in writing the specified data patterns to any valid memory address. These memory addresses must have a byte offset 0 if the memory operations are byte addressable in order to write the specified data into a whole word.

2. Test patterns to detect faults at address lines based on walking-patterns such as Table 4.8, 4.12, 4.17, 4.21 and others can be used in byte addressable memory with the following changes:

   - Test operation for memory initialization and memory read must be word operations.
   - Convert the read and write word operations for waking-pattern into read and write byte operations.
   - The written data in TP1 (fault-free initialization) must consist of eight different bytes.
   - Data value written in TP2 $\neq$ data value written in TP3, data value written in TP3 $\neq$ data value written in TP4, data value written in TP4 $\neq$ Data written in TP2.
   - The data written for TP5-TP17 must be different from each other.

3. Test patterns in Table 4.19 used for detecting a bridge fault between data lines and address lines that flip the data needs some modifications: (1) make the two instruction a write word and read word, and (2) add two instructions similar to the original ones but they write and read to byte 7.

Summarized Test Patterns

For fixed interconnect layout where the exact location for each TSV/interconnect is known, i.e., wide I/O mobile DRAM memory, data lines can be tested for bridge faults based on four groups, therefore, eight memory operations are sufficient. Table 4.49 show the number of optimized test patterns for static faults with fixed layout assuming the conditions satisfied in Table 4.24 still satisfied. Table 4.50 show a summary for the number of test patterns for dynamic faults detection which are optimized in Table 4.51. Table 4.52 gives the number of test patterns for detecting any possible fault.

Table 4.49: Summary: Optimized test patterns for static faults with fixed layout

Type of Static Fault	**Number of operations**
SA0 at data lines/Wired-AND bridge between data and address lines that flips data line	2
SA1 at data lines/Wired-OR bridge between data and address lines that flip data line	2
SA0 Address line/Wired-AND bridge between data and address lines that flip address line/Bridge Fault at Address line (Wired-AND)	L_a+2
SA1 Address line/Wired-OR bridge between data and address lines that flip address line/Bridge Fault at Address line (Wired-OR)	L_a+2
Bridge Fault at Data line (Wired-AND/OR)	8

Table 4.50: Summary: Test patterns for dynamic faults with interconnect fixed layout

Type of Dynamic Fault	**Number of operations**
Path delay fault (PDF) at data lines	6
Path delay fault (PDF) at address lines	10
Stuck open fault (SOF) at data lines	4
Stuck open fault (SOF) at address lines	4*L_a
Crosstalk fault at data lines	5*4=20
Crosstalk fault at address lines	8*L_a
Path delay fault (PDF) with crosstalk at data lines	5*4=20
Path delay fault (PDF) with crosstalk at address lines	8*L_a
Stuck open fault (SOF) with crosstalk at data lines	8*4=32
Stuck open fault (SOF) with crosstalk at address lines	6*L_a

Table 4.51: Summary: Optimized test patterns for dynamic faults with interconnect fixed layout

Type of Dynamic Fault	**Number of operations**
Path delay fault (PDF) at data lines	6
Path delay fault (PDF) at address lines	10
Stuck open fault (SOF) at data lines	4
Stuck open fault (SOF) at address lines	4*L_a
Crosstalk fault/ Path delay fault (PDF) with crosstalk at data lines	5*4=20
Crosstalk fault/ Path delay fault (PDF) with crosstalk at address lines	8*L_a
Stuck open fault (SOF) with crosstalk at data lines	8*4=32
Stuck open fault (SOF) with crosstalk at address lines	6*L_a

Table 4.52: Final test patterns for static and dynamic faults

Type of Dynamic Fault	**Number of operations**
SA0 at data lines/Wired-AND bridge between data and address lines that flips data line	2
SA1 at data lines/Wired-OR bridge between data and address lines that flip data line	2
SA0 Address line/Wired-AND bridge between data and address lines that flip address line/Bridge Fault at Address line (Wired-AND)	L_a+2
SA1 Address line/Wired-OR bridge between data and address lines that flip address line/Bridge Fault at Address line (Wired-OR)	L_a+2
Bridge Fault at Data line (Wired-AND/OR)	8
Path delay fault (PDF) at data lines	6
Path delay fault (PDF) at address lines	10
Stuck open fault (SOF) at data lines	4
Single line fault due to complete open at address lines	4*L_a
Crosstalk fault/Path delay fault (PDF) with crosstalk at data lines	5*4=20
Crosstalk fault/Path delay fault (PDF) with crosstalk at address lines	8*L_a
Stuck open fault (SOF) with crosstalk at data lines	8*4=32
Stuck open fault (SOF) with crosstalk at address lines	6*L_a

4.5 Summary

This chapter discussed a new algorithm called Memory Based Interconnect Testing (MBIT) to test interconnects in memory stacked on logic. The main topics are:

- Describing the fault models targeted in this thesis based on few assumptions and conditions related to memory-on-logic.
- Explaining the detection conditions for different interconnect types to detect interconnect static and dynamic faults.
- Discussing the applicable detection conditions, based on interconnect types in memory-on-logic, for detecting interconnect faults.
- Extending the applicable detection conditions to identify the exact test patterns, which represent memory write and read operations, to test data and address lines for both static and dynamic faults.

Experimental Results and Comparison

5

This chapter provides the experimental results of a case study and compares them with the state-of-the-art approaches. Section 5.1 discusses the DfT general requirements and constraints for interconnect testing in Memory-on-Logic. Section 5.2 presents an overview of the state-of-the-art regarding 3D interconnect testing. Section 5.3 discusses the newly proposed approach called Memory Based Interconnect Testing (MBIT). Section 5.4 compares MBIT methodology with the previous work. Finally, Section 5.5 concludes this chapter with a summary.

5.1 DfT Requirement for Memory-on-Logic Interconnect

In this section, we will explain the requirements, constraints, and conditions to test interconnect in Memory-on-Logic. As each chip must be tested, usually additional hardware is augmented during design time specifically to reduce the complexity of testing. This is also referred to as design-for-testability (DfT). It adds or modifies the hardware and usually introduces new test modes. In *normal mode*, the circuit behaves as initially intended; the test logic is completely transparent. In *test mode*, test patterns are applied to verify the correctness of the chip.

Testing 3D-SICs is more challenging as compared to planer dies, and has several test phases: pre-bond, mid-bond, post-bond, and final test phase [69, 82]. A pre-bond test takes place prior stacking and filters defective dies from entering the stack. Here, TSVs could be tested as well although interconnects are not formed yet as dies still have to be stacked. Pre-bond TSV testing is difficult because TSVs with small dimensions are hard to probe [83]. Mid-bond and post-bond test both take place after stacking; mid-bond testing is used to denote tests for partially created stacks, while post-bond testing takes place right before packaging which might result in the avoidance of unnecessary assembly and packaging costs. Note that during mid-bond and post-bond testing, dies can be retested. More importantly, however, is to test for the new component in the stack, i.e., the formed interconnections. The final test phase is after assembly and packaging, and controls the quality of the outgoing product. In this thesis, we focus on *post-bond testing*, of interconnections between memory and logic dies.

The test must satisfy requirements related to the memory and 3D-SIC interface, test quality and cost, compatibility with previous standards and test modularity [84]. Each is described next.

- Memory and 3D-SIC interface

  - The test architecture should support various types of memories such as SRAM, DRAM, Flash, etc. The interface between the memory and logic die goes over TSVs.
  - The I/Os of the 3D-SIC are connected through the logic layer, here assumed to be the bottom layer. Thus any test responses have to be propagated back to the bottom layer.

- Test quality and test cost
  - The test architecture must have full controllability and observability. Each TSV between logic die and memory die should be tested.
  - Test must be cost-effective; i.e., the test time should be reasonable and scalable with the number of TSVs/interconnects.
  - At-speed testing; test can be executed at application speed.
  - Test both static and dynamic faults.
  - Able to perform fault detection and diagnosis.
  - Flexibility in test patterns; able to add test patterns for new faults.

- Compatibility
  - The test architecture should be compliant with previous standards or an extension of an existing standard.

- Modular testing
  - Testing should be modular, i.e., dies and interconnects can be tested separately. Modular testing has several advantages such as [5]:
    * 3D-SICs with heterogeneous dies, each die has different defects, test patterns and fault models. Therefore, each die requires different test for fault detection.
    * Easy reuse of tests for different modules in the same 3D-SIC or for different 3D-SICs.
    * Divide-and-Conquer approach, where a design is divided into simple components that make test pattern generation more tractable, and save development time and cost.
    * First-order diagnosis, where the exact module containing the fault is determined.
    * Stacked dies sometimes constitute a protected *intellectual property* (IP), where the implementation details are withheld.

5.2 State-of-the-art in 3D Interconnect Testing

Currently, the state-of-the-art in 3D interconnect testing, primarily focused on the Boundary Scan [85]. However, other methods such as the IEEE 1581 [22] can be used for interconnect testing in 3D stacked memories. Both methods are described next.

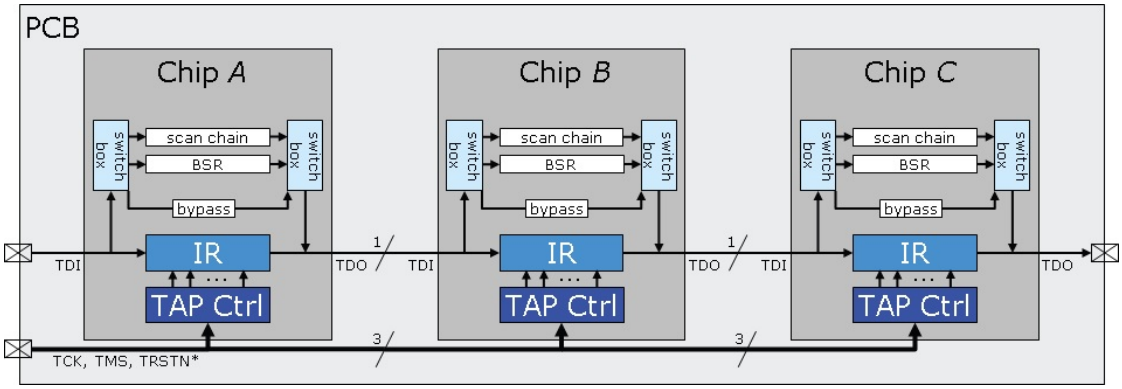


Figure 5.1: Boundary Scan based test architecture based on IEEE 1149.1 [20]

5.2.1 Boundary Scan Based Interconnect Testing

There are several standards based on the boundary scan concept such as IEEE 1149.1 [85] for 2D-ICs, IEEE P1838 [22, 39] which is a new standard for 3D-SICs, and IEEE 1500 [86] for Core-based SOCs. Next, we will consider IEEE 1149.1 and IEEE P1838.

IEEE 1149.1 design-for-test standard known as Joint Test Action Group (JTAG) is primarily designed for interconnect testing (EXTEST) between ICs on a Printed Circuit Board (PCB). It facilitate testing by inserting virtual probes replacing the need for physical probes [17]. As depicted in Figure 5.1, which represents an example for three chips placed on a PCB, IEEE 1149.1 consists of the following components:

- Test Access Ports (TAP), which consists of:
  - Test Data Input (TDI) and Test Data Output (TDO) input signals which are *concatenated* to form a sequential chain through the chips.
  - Test Clock (TCK), Test Mode Select (TMS), and Test Reset (TRSTN*) control signals which are *broadcasted* to all dies [20].
- TAP Controller: It controls the mode operation of each chip. The TAP controller is a finite state machine (FSM) with 16 different states, used to enable applying test patterns to data and instruction registers and observe the test response.
- Instruction Registers, used to specify the particular test mode that should be run.
- Boundary Scan Registers (BSR); registers around I/Os placed for observability and controllability.

Boundary Scan has four compulsory instructions in addition to six optional instructions. The compulsory instructions are:

1. *BYPASS*: This instruction is used to bypass the boundary scan register (BSR) on unused chips to prevent long shift operation. For example, Chip A and Chip B can be bypassed if the test targeted Chip C only.

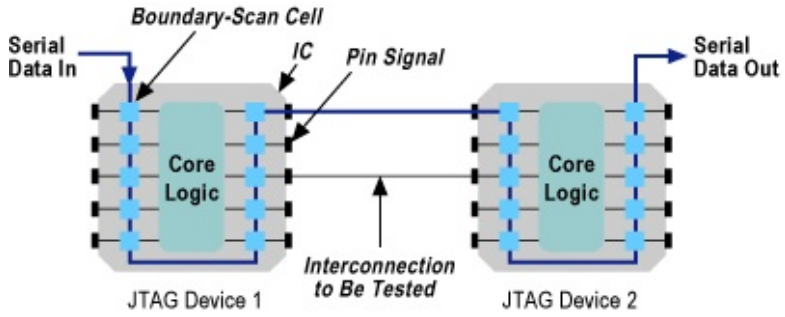


Figure 5.2: JTAG (IEEE 1149.1) based interconnect testing [21]

2. *SAMPLE*: This instruction is used to take and examine a snapshot of the normal operation of the chip. The captured data can be shifted out for examination. Also capture the signals applied to the primary inputs and the response appearing in the primary output of the internal chip logic.

3. *PRELOAD*: This instruction targets the Boundary Scan Register between TDI and TDO. It is used to place an initial data pattern at latched parallel outputs of the boundary scan cells without disconnecting the internal logic from the IC pins.

4. *EXTEST*: This instruction is used to test the interconnects between chips, where a test pattern is preloaded into output Boundary Scan Registers of the involved chips. The responses are latched on the input Boundary Scan Registers cells and shifted out using TDO for verification. For example, for interconnect testing using *EXTEST* instruction based on Figure 5.2, output boundary scan register (BSR) of JTAG Device 1 is pre-loaded with a test pattern, then propagate the test pattern to JTAG Device 2. After that, latch the response on the input cells and shift it to the TDO for examination and fault detection.

IEEE 1149.1 has only serial test mechanism and lacks a high-bandwidth parallel test access mechanism. Its major feature is supporting interconnect testing. Using the serial path between TDI and TDO to move megabytes of data around the guts of ICs is very slow. Moreover, it is not designed for delay testing, at-speed testing, or signal-integrity testing [79, 87]. Memories usually do not conform to JTAG standards; they do not have TAP or dedicated BSRs, even though, they can be tested from a connected device such as microprocessors that have a BS facility.

JTAG is not suitable for at-speed testing because the time between update of test stimuli and capture of response vector (the time interval between Update-DR state and Capture-DR state) is 2.5 clock cycle, also, it has a long shift operation between two consecutive test vectors. Moreover, JTAG is unable to detect dynamic faults such as crosstalk. Several modifications and extension targeted these drawbacks which add new hardware to BSCs or completely modifies them such as [79, 87], but none is suitable for practical adaptation.

Delay testing may be through some IEEE 1149.1 extensions; for example testing a delay at a single TSV interconnect cannot be tested at-speed since that demands too

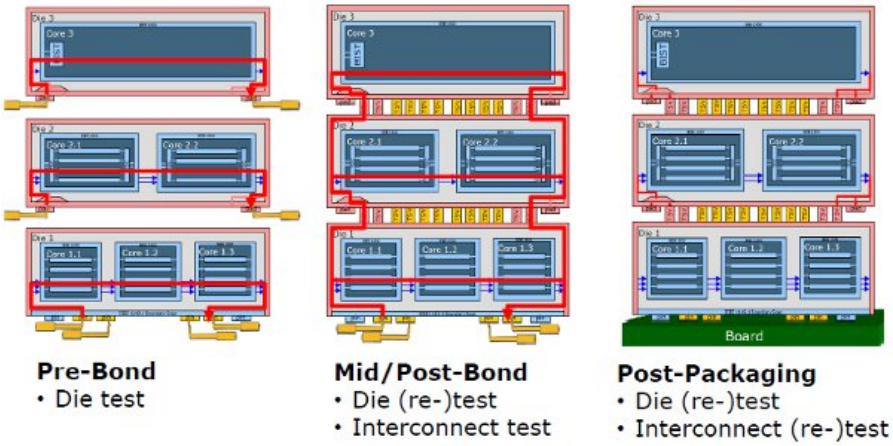


Figure 5.3: IEEE P1838 [22]

much speed. Therefore, performing an effective delay test can performed based on TSVs concatenations (testing a path of several TSVs). But, using this approach, identifying a single failure is difficult because the test performed for the whole chain of TSVs and not one TSV.

IEEE P1838 is an on-going standard for test access architecture for TSV-based 3D-SICs. As depicted in Figure 5.3 this architecture used for both intra-die (INTEST) testing and inter-die (EXTEST) testing during the following test phases [22, 39]:

- Pre-bond phase: Individual dies are tested using dedicated pins.
- Mid/Post-bond phase: Stacked dies in partial and complete stack are re-tested (INTEST) and the interconnects between dies are tested (EXTEST) based on JTAG.
- Post-packaging phase: Both dies and interconnects are re-tested.

The main characteristics of Boundary Scan Based Interconnect Testing can be summarized as follows:

- Not all the memory devices specially the complex ones support the JTAG (IEEE 1149.1) because the memory manufactures still reluctant to add new hardware to their products.
- For all 3D-SICs, the bottom die with I/Os that form the interface to the board usually is IEEE 1149.1 compliant.
- Test time is relatively long; each test pattern must go serially through all BSCs, i.e., the test time for testing interconnects between the dies in memory-on-logic is $2*R_L*\lceil Log(N+2)\rceil$ clock cycle, where N represent the number of TSVs/interconnects between the stacked dies which is $L_a+L_c+L_d$. R_L is the length of BSR which is $L_a+L_c+3*L_d$ plus the number of external I/Os signals where we can ignore it compared to number of TSVs.

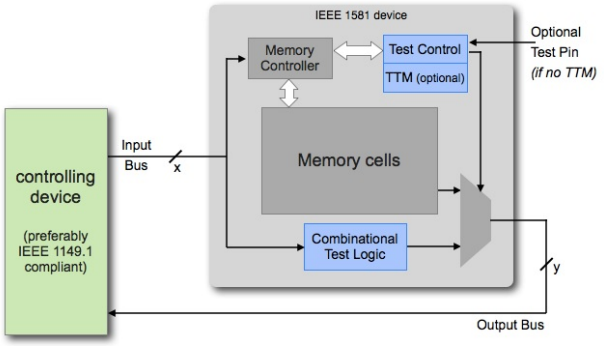


Figure 5.4: Logic based (IEEE 1581) interconnect testing [23]

- JTAG is unable to detect dynamic faults and perform at-speed testing because the time interval between Update-DR and Capture-DR is more than one clock cycle, and the length of BSR is too much.
- JTAG was designed to detect static faults such as SAF, open and short fault with good diagnosis capabilities.
- The test patterns are flexible, where the desired values on all lines/TSVs can predetermined then shifted serially.
- JTAG forms the basic Boundary Scan standard that has few extensions and modifications based on it.

5.2.2 Logic Based Interconnect Testing

IEEE 1581 which represent an alternative to Boundary Scan in memory devices was approved and published in 2011. It is based on Static Component Interconnect Test Technology (*SCITT*), which was proposed by philips and Fujitsu. SCITT is a simple and effective method for assembly test. Its device has a set of input and output pins, where the output is an XOR/XNOR of some of the inputs, and each input may be used in multiple gates. Each output pin has a unique mapping of the circuit inputs. SCITT has two main aspects: (1) it is about *static* testing, (2) specifically for component *interconnection* testing [88, 24].

Logic testing (IEEE 1581) can be used in memories, 3D-SICs, multi-chip packages, and in slave-type components for testing connectivity of pins. It is considered a complementary to JTAG and not a general replacement. IEEE 1581 is depicted in Figure 5.4 where the IEEE 1581 device (memory device) is connected to the controlling device which is JTAG compliant such as CPU. The IEEE 1581 standard applies to memory devices with the following characteristics [40]:

- The IEEE 1149.1 is not applicable for whatever reason, for example JTAG requires extra pins.

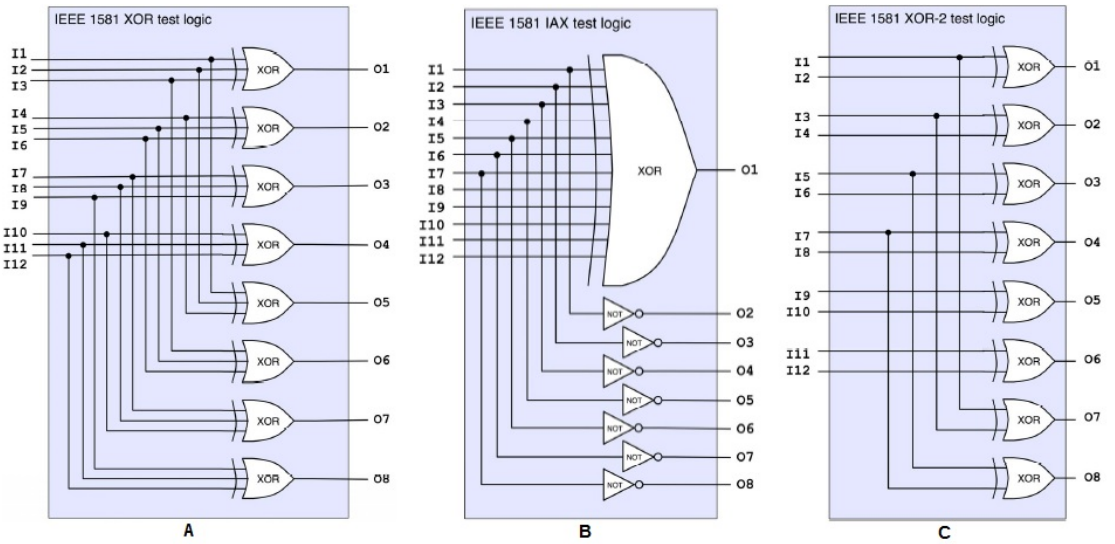


Figure 5.5: IEEE 1581 test logic (a) XOR, (b) IAX, (c) XOR-2[24]

- Typically, the device is complex memory; such as Flash, SDRAM, DDRRAM and others. Complex means the device requires setting several pins for proper device initialization, and any pin failure means that the memory cannot be accessed by the processor for testing.

- The device has a high functional speed.

The IEEE 1581 device has two modes of operation; normal mode and testing mode. In *normal mode*, the combinational test logic is disabled, and the device operates normally. In *test mode*, the device functionality is bypassed, and the combinational test logic is used for interconnect testing. Master device apply the inputs and observe the outputs to detect faults [23]. With defected interconnects, the test is still executable and this is a major benefit compared to other methods [23]. The internal test logic may have different combinational implementations options which will give the designer some controllability. The different options for combinational test logic implementation have different gates used with different number of inputs shown in Figure 5.5 such as:

1. XOR (3-input XOR or XNOR gates): This logic implementation is based on XOR test logic as shown in Figure 5.5(a) which represent an implementation with 12 inputs and 8 outputs. Any output depends on XOR gate with odd number of inputs, and three inputs as minimal requirement. There are no two outputs with the same inputs, and any input must participate in two XOR gates at least.

2. IAX (XOR, inverters, and AND gates): This logic implementation as shown in Figure 5.5(b) has a single XOR gate, inverters, and any number of AND gates. Each output must have set of inputs and logical function different than other outputs.

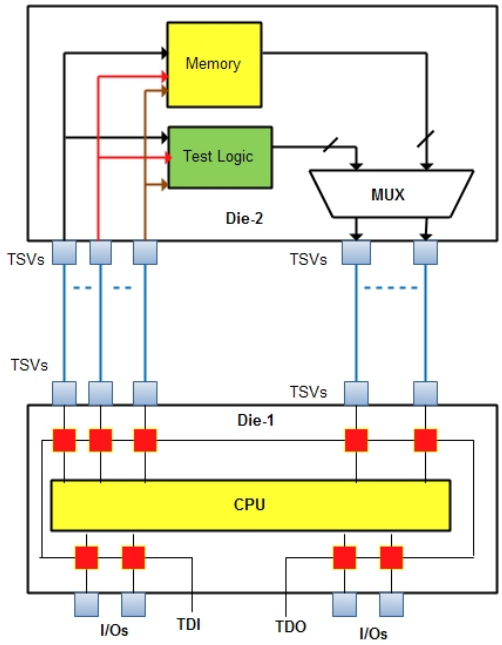


Figure 5.6: IEEE 1581

3. XOR-2 (2-input XOR or XNOR gates): This logic implementation is based on 2-input XOR gate, where each output has unique inputs. Figure 5.5(c) show such implementation with 12 inputs and 8 outputs.

Figure 5.6 depicts interconnect testing in 3D-SICs based on test logic. The bottom die (CPU) contains a Boundary Scan while the top die (memory) contains test logic. The multiplexer in the top die is used to choose between the output of the memory during operational mode and the output of the test logic during test mode.

Test time for static faults based on this approach is relatively long; each test pattern must go serially through all BSCs in the bottom die, i.e., the test time for testing interconnects between the dies in memory-on-logic is $(R_L+1)*\lceil Log(N+2)\rceil$ clock cycle, where N represent the number of TSVs/interconnects between the stacked dies which is $L_a+L_c+L_d$. R_L is the length of BSR in master die which is $L_a+L_c+3*L_d$ plus the number of external I/Os signals where we can ignore it compared to number of TSVs.

IEEE 1581 has few advantages compared to JTAG, such as: (1) have no extra pins/TSVs, (2) eliminate the need for memory initialization for proper interconnect testing, (3) faster execution time with small set of test vector, (4) suitable for complex memories, i.e., SDRAM and DDRRAM, and (5) solve memory cluster problems, i.e., shorter execution time. The main characteristics of Test Logic Based Interconnect Testing can be summarized as follows:

- Test logic can be implemented in all memory types, including the complex ones such as SDRAM, DDRRAM.
- For 3D-SICs with test logic implemented on the top die, the bottom die that forms the interface to the board with I/Os is IEEE 1149.1 compliant.

- Test time for static faults requires $R_L * \lceil Log(N+2) \rceil$ clock cycles.
- Test logic is unable to detect dynamic faults and perform at-speed testing because the logic die is Boundary Scan based.
- IEEE 1581 is designed to provide full detection and diagnosis of possible SAF, open, and short faults.
- Test patterns are not flexible because the values of memory output lines are generated according to the logical implementation of test logic based on the values of memory input lines.
- IEEE 1581 standard is based on SCITT methodology, and must be connected to JTAG compliant device. It may form a base for future extensions.

5.3 Memory Based Interconnect Test (MBIT)- A Case Study

The state-of-the-art in 3D interconnect testing have some drawbacks such as additional pins/TSVs, and the extra hardware. In addition, they are unable to provide at-speed and a full dynamic testing. DfT for CMOS logic can be defined and inserted with negligible area cost. While DfT definition and insertion into DRAM memory is more difficult due to some constraints, which makes interconnect testing more complicated. Further, some memory manufacturers are reluctant to provide an access to their memory BIST for intellectual property (IP) considerations. Therefore, Memory Based Interconnect Testing (MBIT) was proposed to overcome the previous drawbacks. The new approach was proposed and investigated throughout the thesis. It is based on performing read and write operations to the memory. MBIT satisfies the general requirements for interconnect testing, for example:

- The proposed methodology is memory type and interface independent; it performs read and write operations to any possible memory.
- The 3D-SIC consist of a memory die stacked on top of logic (CPU) die. The bottom logic die is JTAG compliant and has I/Os pins which is connected to the board.
- It provides a full controllability and observability of all the TSVs in the 3D-SIC where the read and write memory operations generated by the CPU are based on predetermined test patterns to detect faults on all the TSVs between the logic and memory dies.
- The test patterns are cost-effective where memory read and writes operations are optimized as much as possible to be executed in the shortest time.
- MBIT is the first methodology to run the test at application speed since it relies on the ordinary operations of the memory.
- In addition to detecting static faults, dealing with the memory is able to detect dynamic faults too.

- The memory operations used for faults detection can be extended easily to provide diagnosis capability.
- MBIT is very flexible; the detection of new possible faults requires specifying the test patterns based on fault nature, and then convert such patterns into memory operations.
- Memory Based Interconnect Testing does not demand adding new pins, TSVs, or hardware. Therefore, it is fully compliant with other standards and their extensions.
- The proposed methodology is developed for testing interconnects between dies, but faulty dies can be detected easily.

Based on memory read and write requirements, the case study covered in this chapter is written in assembler language. The configuration for MIPS64 assembler can handle maximum of 64-bit data lines and 12-bit address lines. Most readers may not be familiar with this, therefore a minimal subset of the MIPS64 assembler language is introduced at this point. There are 32 registers (numbered $0-$31) and each can hold 64 bits.

- Temporary Registers - $t0-$t9 - Used to hold *temporary* values.
- Saved Registers - $s0-$s7 - Used to hold *saved* values.
- Zero Register - $0 or $zero - Always 0, even if written to.

Table 5.1: MIPS Instructions

Opcode	**Instruction**	**Format**	**Purpose**
ADD	Add Word	ADD rd, rs, rt)	To add 32-bit integers. If an overflow occurs, then trap
ADDI	Add Immediate Word	ADDI rt, rs, immediate	To add a constant to a 32-bit integer. If overflow occurs, then trap
AND	Bitwise and	AND rd, rs, rt	To do a bitwise logical AND
ANDI	And Immediate	ANDI rt, rs, immediate	To do a bitwise logical AND with a constant
BEQ	Branch on Equal	BEQ rs, rt, offset	To compare GPRs then do a PC-relative conditional branch
BNE	Branch on Not Equal	BNE rs, rt, offset	To compare GPRs then do a PC-relative conditional branch
LB	Load Byte	LB rt, offset(base)	To load a byte from memory as a signed value
LD	Load Doubleword	LD rt, offset(base)	To load a doubleword from memory
SB	Store Byte	SB rt, offset(base)	To store a byte to memory
SD	Store Doubleword	SD rt, offset(base)	To store a doubleword to memory

Table 5.1 shows the instructions of interest to this thesis. We converted the test patterns implemented previously in Chapter 4 for every fault into a MIPS code. Table 5.2 and Table 5.3 shows the number of memory operations and clock cycles for static faults and dynamic faults respectively.

Table 5.2: Number of memory operations and clock cycles (CC) required using MIPS64 assembler for memory with 12-bit address line and 64-bit data line for each static fault

Fault Name	**# Memory operations**	**# CC**
SA0 at Data line	11	17
SA1 at Data line	4	10
SA0 at Address line	57	65
SA1 at Address line	48	56
Bridge between Address lines with Wired-AND behavior	57	65
Bridge between Address lines with Wired-OR behavior	48	56
Wired-OR bridge between Address and Data line, Flip Data line	4	10
Wired-AND bridge between Address and Data line, Flip Data line	11	17
Wired-OR bridge between Address and Data line, Flip Address line	11	17
Wired-AND bridge between Address and Data line, Flip Address line	12	18
Bridge between data lines (Wired-AND/OR)	63	81

Table 5.3: Number of memory operations and clock cycles (CC) required using MIPS64 assembler for memory with 12-bit address line and 64-bit data line for each dynamic fault

Fault Name	**# Memory operations**	**# CC**
PDF at data lines	17	21
PDF at address lines	16	24
SOF at data lines	14	19
SOF at address lines	39*2=78	61*2=122
Crosstalk at data lines	41	49
Crosstalk at address lines	18*L_a=216	26*L_a=312
PDF with crosstalk at data lines	41	49
PDF with crosstalk at address lines	18*L_a=216	26*L_a=312
SOF with crosstalk at data lines	17*4=68	23*4=92
SOF with crosstalk at address lines	39*2=78	61*2=122

5.4 Comparison and discussion

During the previous sections, the general requirements for 3D-SIC interconnect testing were explained. Then, these requirements were explained for the state-of-the-art methodologies including Boundary Scan and logic Based. Our Memory Based proposed methodology showed clear advantages and improvements over the state-of-the-art in various test requirements, such as:

- Able to detect interconnect faults for all memory types stacked on logic.
- MBIT assume the bottom logic die that has the I/Os pins is responsible for generating test patterns and analyzing its response.
- Able to test all TSVs in the stacking through the full controllability and observability.
- The fasted testing methodology because it executes at-speed an optimized memory operations that detect both static and dynamic faults and these memory operations can be modified to include new faults.

- There is no need to add any pins or hardware to the memory. Also, new TSVs between the dies is not needed which make it compatible with previous standards such as JTAG.

- It is a modular solution that can test interconnects between stacked dies.

Table 5.4 summarizes the requirements for testing interconnects in memory-on-logic for different methodologies where our proposed solution shows clear advancement.

Table 5.4: Comparison between Interconnect Testing methodologies

Test requirement	Boundary Scan Based	Logic Based	Memory Based
Support various types of memories	Some memories specially the complex are not supported	Support all memory types including the complex ones	Support all memory types regardless of complexity degree
3D-SIC I/Os location	Bottom logic die which is JTAG compliant	Bottom logic die which is JTAG compliant	Bottom logic die which is JTAG compliant
Degree of controllability and observability	Can control and observe every TSV/interconnect	Not full controllability and observability because data line values depends on the values of address lines and logic implementation	Can control and observe every TSV/interconnect
Static faults testing	Applicable	Applicable	Applicable
Degree of effectiveness	Requires $2*(L_a+L_c+3*L_d)*\text{Log}(N+2)$ test clock cycle for static faults	Requires $(L_a+L_c+3*L_d+1)*\text{Log}(N+2)$ test clock cycle for static faults	Requires $2*\text{Log}(L_d+2)+2*L_d+8$ memory operations for static faults
Dynamic faults testing	Not applicable	Not applicable	Applicable with $14*L_a+72$ memory operations
At-speed testing	Not applicable and requires modifications	Not applicable and requires modifications	Applicable
Diagnosis capability	JTAG was designed to detect static faults such as SAF, open and short fault with good diagnosis capabilities	Provide full detection and diagnosis of possible SAF, open, and short faults	Not applicable, can be achieved easily
Flexible test patterns	Unable to detect new faults or may requires modifications	Unable to detect new faults or may requires modifications	Able to detect new faults by developing new test patterns
Compatibility	JTAG is a standard and base for many extensions	Compatible with JTAG devices	Logic die is JTAG compliant and memory die has no extra hardware
Modular testing	Can test different dies separately	Can test different dies and interconnect separately	Can test different dies and interconnect separately
Extra area	Requires $2*(L_a+L_c+3*L_d)$ Boundary Scan Cells	Requires $L_a+L_c+3*L_d$ Boundary Scan Cells	Usually no area overhead, but may need ROM for storing patterns

5.5 Summary

This chapter discussed the experimental results of the proposed MBIT methodology with a study case and provides a comparison with the state-of-the-art methodologies. The main topics are:

- Explaining the general requirements for testing interconnects in memory-on-logic.

- Describing the widely used state-of-the-art methodologies for interconnects testing in 3D-SICs and to what degree each method support the general requirements for testing interconnects in memory-on-logic.
- Introducing our new methodology based on memory read and write operations and describing to what degree it supports the general requirements for testing interconnects in memory-on-logic.
- Providing a comprehensive comparison between the state-of-the-art and our new methodology, and highlighting the advantages of our methodology.

Thesis Summary and Future Work

6

This chapter presents the conclusion of this thesis. Section 6.1 provides a summary of this thesis organized per chapter. Section 6.2 presents recommendation for future work.

6.1 Thesis Summary

In this thesis, a Memory Based Interconnect Testing (MBIT) approach has been proposed targeting TSVs/interconnects on memory-on-logic stacked ICs. MBIT is based on memory read and write operations without any area overhead, and is able to detect both static and dynamic faults (at-speed testing).

In chapter 2, the concept of 3D-SICs was introduced. The definition, drivers, and benefits of Three-Dimensional Stacked Integrated Circuits (3D-SICs) over 2D-ICs were discussed in detail. The manufacturing process of 3D-SICs consists of three main steps: (1) TSV manufacturing process, (2) wafer thinning, and (3) the bonding process. All of them were covered. Then, different abstraction levels of 2D memories were described; they include the behavioral, functional, and electrical model of both SRAM and DRAM. Finally, these planar memories have been mapped to 3D-SICs using different partitioning granularities such as: bank partitioning, and cell partitioning.

In chapter 3, failure mechanisms in both 2D and 3D ICs and their fault models were discussed. It first explained the key terminology related to ICs quality and reliability such as failure mechanisms, defects, faults, fault models, test patterns, testing, and failures. Next, classification overview based on defect location was provided. Finally, fault models were discussed that contain faults (abstraction of defects). The fault models primarily focused on 3D-SIC interconnects.

In chapter 4, we introduced our Memory Based Interconnect Testing (MBIT) methodology to test interconnects in memory stacked on logic; By performing memory read and write operations interconnects are tested indirectly through the memory. The read and write instructions embed the detection of all faults both static and dynamic. To derive this patterns, first we explained the general detection conditions for different interconnect wires for static and dynamic interconnect faults. The applicable detection conditions for specific interconnects in memory-on-logic were discussed next. In the last step, detection conditions were mapped into memory instructions. They test both address lines and data lines. Control lines are assumed to be tested implicitly.

Chapter 5 discussed the experimental results of the proposed MBIT methodology with a study case implemented in MIPS and compared the results with state-of-the-art. The chapter explained general requirements for testing interconnects in memory-on-logic. Then it described state-of-the-art methodologies for interconnects testing in 3D-SICs. After that, it described our new MBIT methodology based on memory read and write operations. All schemes are compared and tested against several requirements, such

as ability to test faults, area overhead, etc. At the end, a comprehensive comparison between the state-of-the-art and our new methodology are provided where advantages and disadvantages of our methodology were highlighted.

6.2 Future Work

We recommend several topics for future work in different areas. They are:

- Testing for defects
  - Interconnect testing under the assumption of multiple faults at a time, where this assumption is more realistic. During the thesis we developed test patterns for fault detection assuming there is a single fault at a time.
  - The control lines were tested implicitly without special test patterns for them, under the assumption that defected control lines will cause Non-operating memory. The control lines can be tested explicitly by developing special test patterns for them to have faster and more accurate testing.
- Testing for diagnosis
  - During the thesis we tested for fault detection only. We can target fault diagnosis in addition to fault detection where diagnosis locates the exact failure location in the chip to perform failure analysis to examine the physical defect. Diagnosis is necessary to fix the process problem and improve the yield. Moreover, the diagnosis should also distinguish between memories and interconnect defects.
- Memory granularity
  - Testing the interconnects considering multiple memory stacked on logic. In this thesis, the focus was on a single die.
  - Investigation of the testability of different memory granularities (stacked banks, cell arrays stacked on logic etc).

Bibliography

[1] J. Verbree, "On 3d stacked ic yield improvement and 3d-dft test architecture," Master's thesis, Delft University of Technology, 2010.

[2] (2013, July). [Online]. Available: http://wenku.baidu.com/view/5c97a4d9ad51f01dc281f1f7.html

[3] P. Garrou, C. Bower, and P. Ramm, *Handbook of 3D Integration: Volume 1 - Technology and Applications of 3D Integrated Circuits*, ser. Handbook of 3D Integration: Technology and Applications of 3D Integrated Circuits. Wiley, 2008. [Online]. Available: http://books.google.nl/books?id=LtbakwQlNs0C

[4] J. Fan and C. S. Tan, *Low Temperature Wafer-Level Metal Thermo-Compression Bonding Technology for 3D Integration, Metallurgy - Advances in Materials and Processes, Dr. Yogiraj Pardhi (Ed.)*, 2012.

[5] E. Marinissen and Y. Zorian, "Testing 3d chips containing through-silicon vias," in *Test Conference, 2009. ITC 2009. International*, Nov., pp. 1–11.

[6] Z. Al-Ars, "Dram fault analysis and test generation," Ph.D. dissertation, Delft University of Technology, Delft, Netherlands, June 2005.

[7] A. J. van de Goor, *Testing semiconductor memories: theory and practice.* Gouda, The Netherlands: A.J. van de Goor, 2001.

[8] S. Hamdioui, "Testing multi-port memories: Theory and practice," Ph.D. dissertation, Delft University of Technology, Delft, Netherlands, April 2001.

[9] (2013) layout model for 6t-sram cell. [Online]. Available: http://www.ece.unm.edu/~jimp/vlsi/slides/chap8_2.html

[10] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital integrated circuits- A design perspective*, 2nd ed. Prentice Hall, 2004.

[11] G. H. Loh, "3d-stacked memory architectures for multi-core processors," *SIGARCH Comput. Archit. News*, vol. 36, no. 3, pp. 453–464, Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1394608.1382159

[12] K. Puttaswamy and G. Loh, "3d-integrated sram components for high-performance microprocessors," *Computers, IEEE Transactions on*, vol. 58, no. 10, pp. 1369 – 1381, oct. 2009.

[13] S. A. Stanisavljevic Milos and L. Yusuf, *Reliability of Nanoscale Circuits and Systems: Methodologies and Circuit Architectures.* Springer New York, 2011.

[14] N.K.Jha and S. Gupta, *Testing of Digital Systems.* Cambridge,United Kingdom: Cambridge University Press, 2003.

[15] Y. S.Mourad, *Principles of Testing Electronic Systems.* John Wiley & Sons,INC, 2000.

[16] K. Chakrabarty, S. Deutsch, H. Thapliyal, and F. Ye, "Tsv defects and tsv-induced circuit failures: The third dimension in test and design-for-test," in *Reliability Physics Symposium (IRPS), 2012 IEEE International*, april 2012, pp. 5F.1.1 – 5F.1.12.

[17] X. W. laung Terng Wang, Cheng-Wen Wu, *VLSI Test Principles and Architectures Design For Testability.* San Francisco ,USA: Morgan Kaufmann Publishers, 2006.

[18] L. Chen, X. Bai, and S. Dey, "Testing for interconnect crosstalk defects using on-chip embedded processor cores," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 317–322.

[19] X. Bai, S. Dey, and J. Rajski, "Self-test methodology for at-speed test of crosstalk in chip interconnects," in *Proceedings of the 37th Annual Design Automation Conference*, ser. DAC '00. New York, NY, USA: ACM, 2000, pp. 619–624. [Online]. Available: http://doi.acm.org/10.1145/337292.337597

[20] E. Marinissen, J. Verbree, and M. Konijnenburg, "A structured and scalable test access architecture for tsv-based 3d stacked ics," in *VLSI Test Symposium (VTS), 2010 28th*, april 2010, pp. 269 –274.

[21] (2013, July) Boundary-scan tool. Accessed: 2013-07-30. [Online]. Available: http://www.altera.com/support/devices/tools/boundary-scan/tls-boundary-scan.html

[22] (2013) Iieee 3d-test working group (3dt-wg). [Online]. Available: http://grouper.ieee.org/groups/3Dtest/

[23] H. Ehrenberg and B. Russell, "Ieee std 1581- a standardized test access methodology for memory devices," in *Test Conference (ITC), 2011 IEEE International*, 2011, pp. 1–9.

[24] F. de Jong and R. Raaijmakers, "Static component interconnection test technology in practice," in *Test Conference, 1999. Proceedings. International*, 1999, pp. 556–565.

[25] J. Lau, "Evolution, challenge, and outlook of tsv, 3d ic integration and 3d silicon integration," in *Advanced Packaging Materials (APM), 2011 International Symposium on*, oct. 2011, pp. 462 –488.

[26] L. YU, "A study of through-silicon-via (tsv) induced transistor variation," Master's thesis, Massachusetts Institute of Technology, 2011.

[27] T. Jiang and S. Luo, "3d integration-present and future," in *Electronics Packaging Technology Conference, 2008. EPTC 2008. 10th*, dec. 2008, pp. 373 –378.

[28] R. Anigundi, H. Sun, J.-Q. Lu, K. Rose, and T. Zhang, "Architecture design exploration of three-dimensional (3d) integrated dram," in *Quality of Electronic Design, 2009. ISQED 2009. Quality Electronic Design*, march 2009, pp. 86 –90.

[29] R. Patti, "Three-dimensional integrated circuits and the future of system-on-chip designs," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1214 –1224, june 2006.

[30] S. Deutsch, B. Keller, V. Chickermane, S. Mukherjee, N. Sood, S. Goel, J. Chen, A. Mehta, F. Lee, and E. Marinissen, "Dft architecture and atpg for interconnect tests of jedec wide-i/o memory-on-logic die stacks," in *Test Conference (ITC), 2012 IEEE International*, 2012, pp. 1–10.

[31] C. M. Tan and F. He, "3d circuit model for 3d ic reliability study," in *Thermal, Mechanical and Multi-Physics simulation and Experiments in Microelectronics and Microsystems, 2009. EuroSimE 2009. 10th International Conference on*, 2009, pp. 1–7.

[32] H.-H. S. Lee and K. Chakrabarty, "Test challenges for 3d integrated circuits," *IEEE Design and amp; Test of Computers*, vol. 26, no. 5, pp. 26–35, 2009.

[33] E. Marinissen, "Challenges in testing tsv-based 3d stacked ics: Test flows, test contents, and test access," in *Circuits and Systems (APCCAS), 2010 IEEE Asia Pacific Conference on*, 2010, pp. 544–547.

[34] X. Wu, P. Falkenstern, and Y. Xie, "Scan chain design for three-dimensional integrated circuits (3d ics)," in *Computer Design, 2007. ICCD 2007. 25th International Conference on*, 2007, pp. 208–214.

[35] X. Wu, Y. Chen, K. Chakrabarty, and Y. Xie, "Test-access mechanism optimization for core-based three-dimensional socs," in *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, 2008, pp. 212–218.

[36] L. Jiang, L. Huang, and Q. Xu, "Test architecture design and optimization for three-dimensional socs," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, 2009, pp. 220–225.

[37] L. Jiang, Q. Xu, K. Chakrabarty, and T. M. Mak, "Layout-driven test-architecture design and optimization for 3d socs under pre-bond test-pin-count constraint," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, 2009, pp. 191–196.

[38] E. J. Marinissen. (2011, October) Ieee 3d-test working group (3dt-wg). [Online]. Available: http://grouper.ieee.org/groups/3Dtest/

[39] (2013) P1838 - standard for test access architecture for three-dimensional stacked integrated circuits. [Online]. Available: http://standards.ieee.org/develop/project/1838.html

[40] (2013) Ieee std 1581 - ieee standard for static component interconnection test protocol and architecture. [Online]. Available: http://grouper.ieee.org/groups/1581/

[41] C. Laviron, B. Dunne, V. Lapras, P. Galbiati, D. Henry, F. Toia, S. Moreau, R. Anciant, C. Brunet-Manquat, and N. Sillon, "Via first approach optimisation for through silicon via applications," in *Electronic Components and Technology Conference, 2009. ECTC 2009. 59th*, May, pp. 14–19.

[42] D. H. Kim, K. Athikulwongse, and S.-K. Lim, "A study of through-silicon-via impact on the 3d stacked ic layout," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, 2009, pp. 674–680.

[43] A. NPapanikolaou, D. Soudris, and R. Radojcic, *Three Dimensional System Integration.* Springer US, 2011.

[44] M. Taouil, S. Hamdioui, K. Beenakker, and E. Marinissen, "Test cost analysis for 3d die-to-wafer stacking," in *Test Symposium (ATS), 2010 19th IEEE Asian*, Dec., pp. 435–441.

[45] G. H. Loh, Y. Xie, and B. Black, "Procegabrielssor design in 3d die-stacking technologies," *Micro, IEEE*, vol. 27, no. 3, pp. 31–48, May-June.

[46] R. Weerasekera, M. Grange, D. Pamunuwa, H. Tenhunen, and L.-R. Zheng, "Compact modelling of through-silicon vias (tsvs) in three-dimensional (3-d) integrated circuits," in *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*, 2009, pp. 1–8.

[47] A.-C. Hsieh and T. Hwang, "Tsv redundancy: Architecture and design issues in 3-d ic," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 4, pp. 711–722, 2012.

[48] R. W. (Ed), *Nanoelectronics and Information Technology: Advanced Electronic Materials and Novel Devices.* Wiley-VCH, 2003.

[49] S. Hamdioui and M. Taouil, "Yield improvement and test cost optimization for 3d stacked ics," in *Test Symposium (ATS), 2011 20th Asian*, nov. 2011, pp. 480 –485.

[50] U. Kang, H.-J. Chung, S. Heo, D.-H. Park, H. Lee, J. H. Kim, S.-H. Ahn, S.-H. Cha, J. Ahn, D. Kwon, J.-W. Lee, H.-S. Joo, W.-S. Kim, D. H. Jang, N. S. Kim, J.-H. Choi, T.-G. Chung, J.-H. Yoo, J. S. Choi, C. Kim, and Y.-H. Jun, "8 gb 3-d ddr3 dram using through-silicon-via technology," *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 1, pp. 111 –119, jan. 2010.

[51] L. Jiang, Y. Liu, L. Duan, Y. Xie, and Q. Xu, "Modeling tsv open defects in 3d-stacked dram," in *Test Conference (ITC), 2010 IEEE International*, 2010, pp. 1–9.

[52] Y.-F. Tsai, F. Wang, Y. Xie, N. Vijaykrishnan, and M. Irwin, "Design space exploration for 3-d cache," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 4, pp. 444 –455, april 2008.

[53] S. Thompson, M. Armstrong, C. Auth, M. Alavi, M. Buehler, R. Chau, S. Cea, T. Ghani, G. Glass, T. Hoffman, C.-H. Jan, C. Kenyon, J. Klaus, K. Kuhn,

Z. Ma, B. Mcintyre, K. Mistry, A. Murthy, B. Obradovic, R. Nagisetty, P. Nguyen, S. Sivakumar, R. Shaheed, L. Shifren, B. Tufts, S. Tyagi, M. Bohr, and Y. El-Mansy, "A 90-nm logic technology featuring strained-silicon," *Electron Devices, IEEE Transactions on*, vol. 51, no. 11, pp. 1790–1797, 2004.

[54] K. Zhang, U. Bhattacharya, Z. Chen, F. Hamzaoglu, D. Murray, N. Vallepalli, Y. Wang, B. Zheng, and M. Bohr, "Sram design on 65-nm cmos technology with dynamic sleep transistor for leakage reduction," *Solid-State Circuits, IEEE Journal of*, vol. 40, no. 4, pp. 895–901, 2005.

[55] U. Bhattacharya and K. Zhang, "45nm sram technology developmentand technology lead vehicle," *Intel Technology Journal*, vol. 12, 2008.

[56] D. Fried, J. Hergenrother, A. W. Topol, L. Chang, L. Sekaric, J. W. Sleight, S. Mcnab, J. Newbury, S. E. Steen, G. Gibson, Y. Zhang, N. C. M. Fuller, J. Bucchignano, C. Lavoie, C. Cabral, D. Canaperi, O. Dokumaci, D. Frank, E. A. Duch, I. Babich, K. Wong, J. Ott, C. Adams, T. Dalton, R. Nunes, D. Medeiros, R. Viswanathan, M. Ketchen, M. Ieong, W. Haensch, and K. Guarini, "Aggressively scaled (0.143 mu;m2) 6t-sram cell for the 32 nm node and beyond," in *Electron Devices Meeting, 2004. IEDM Technical Digest. IEEE International*, 2004, pp. 261–264.

[57] B. Haran, A. Kumar, L. Adam, J. Chang, V. Basker, S. Kanakasabapathy, D. Horak, S. Fan, J. Chen, J. Faltermeier, S. Seo, M. Burkhardt, S. Burns, S. Halle, S. Holmes, R. Johnson, E. Mclellan, T. M. Levin, Y. Zhu, J. Kuss, A. Ebert, J. Cummings, D. Canaperi, S. Paparao, J. Arnold, T. Sparks, C. S. Koay, T. Kanarsky, S. Schmitz, K. Petrillo, R. H. Kim, J. Demarest, L. Edge, H. Jagannathan, M. Smalley, N. Berliner, K. Cheng, D. LaTulipe, C. Koburger, S. Mehta, M. Raymond, M. Colburn, T. Spooner, V. Paruchuri, W. Haensch, D. Mcherron, and B. Doris, "22 nm technology compatible fully functional 0.1 um2 6t-sram cell," in *Electron Devices Meeting, 2008. IEDM 2008. IEEE International*, 2008, pp. 1–4.

[58] H.-Y. Chen, C.-C. Chen, F.-K. Hsueh, J.-T. Liu, C.-Y. Shen, C.-C. Hsu, S.-L. Shy, B.-T. Lin, H.-T. Chuang, C.-S. Wu, C. Hu, C.-C. Huang, and F.-L. Yang, "16nm functional 0.039 um2 6t-sram cell with nano injection lithography, nanowire channel, and full tin gate," in *Electron Devices Meeting (IEDM), 2009 IEEE International*, 2009, pp. 1–3.

[59] C. Yu, C. Chang, H. Y. Wang, J. H. Chang, L. H. Huang, C. Kuo, S. P. Tai, S. Y. Hou, W. Lin, E. Liao, K. F. Yang, T. J. Wu, W. C. Chiou, C. Tung, S. Jeng, and C. Yu, "Tsv process optimization for reduced device impact on 28nm cmos," in *VLSI Technology (VLSIT), 2011 Symposium on*, 2011, pp. 138–139.

[60] G. Katti, A. Mercha, J. Van Olmen, C. Huyghebaert, A. Jourdain, M. Stucchi, M. Rakowski, I. Debusschere, P. Soussan, W. Dehaene, K. De Meyer, Y. Travaly, E. Beyne, S. Biesemans, and B. Swinnen, "3d stacked ics using cu tsvs and die to wafer hybrid collective bonding," in *Electron Devices Meeting (IEDM), 2009 IEEE International*, 2009, pp. 1–4.

[61] H. Chaabouni, M. Rousseau, P. Leduc, A. Farcy, R. El-Farhane, A. Thuaire, G. Haury, A. Valentian, G. Billiot, M. Assous, F. De Crecy, J. Cluzel, A. Toffoli, D. Bouchu, L. Cadix, T. Lacrevaz, P. Ancey, N. Sillon, and B. Flechet, "Investigation on tsv impact on 65nm cmos devices and circuits," in *Electron Devices Meeting (IEDM), 2010 IEEE International*, 2010, pp. 35.1.1–35.1.4.

[62] (2013, July) International technology roadmap for semiconductors. Accessed: 2013-07-30. [Online]. Available: http://www.itrs.net

[63] M. L.Bushnell and V. D.Agrawal, *Essentials of Electronic Testing For Digital,Memory and Mixed-Signal VLSI Circuits.* Kluwer Academic Publishers, 2003.

[64] (2013, July). [Online]. Available: http://www.sematech.org/meetings/archives/3d/10124/pres/Reliability_tables.pdf

[65] N. Z. bin HARON, "Testability and fault tolerance for emerging nanoelectronic memories," Ph.D. dissertation, Delft University of Technology, Delft, Netherlands, 2012.

[66] S. Hamdioui, Z. Al-Ars, and A. J. Van de Goor, "Opens and delay faults in cmos ram address decoders," *Computers, IEEE Transactions on*, vol. 55, no. 12, pp. 1630–1639, 2006.

[67] A. J. Van de Goor, S. Hamdioui, and Z. Al-Ars, "Tests for address decoder delay faults in rams due to inter-gate opens," in *Test Symposium, 2004. ETS 2004. Proceedings. Ninth IEEE European*, 2004, pp. 146–151.

[68] JEDEC, *Failure Mechanisms and Models for Semiconductor Devices.* JEDEC Publication No. 122B 2003.

[69] E. Marinissen, "Testing tsv-based three-dimensional stacked ics," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, pp. 1689 –1694.

[70] (2013, July). [Online]. Available: http://www.sematech.org/meetings/archives/3d/10124/pres/Beyne.pdf

[71] S. Kannan, B. C. Kim, and B. Ahn, "Fault modeling and multi-tone dither scheme for testing 3d tsv defects," *J. Electronic Testing*, vol. 28, no. 1, pp. 39–51, 2012.

[72] F. Ye and K. Chakrabarty, "Tsv open defects in 3d integrated circuits: Characterization, test, and optimal spare allocation," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, June, pp. 1024–1030.

[73] C.-W. Kuo and H.-Y. Tsai, "Thermal stress analysis and failure mechanisms for through silicon via array," in *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2012 13th IEEE Intersociety Conference on*, 30 2012-June 1, pp. 202–206.

[74] X. Liu, Q. Chen, P. Dixit, R. Chatterjee, R. Tummala, and S. Sitaraman, "Failure mechanisms and optimum design for electroplated copper through-silicon vias (tsv)," in *Electronic Components and Technology Conference, 2009. ECTC 2009. 59th*, may 2009, pp. 624 –629.

[75] M. Jung, X. Liu, S. K. Sitaraman, D. Z. Pan, and S. K. Lim, "Full-chip through-silicon-via interfacial crack analysis and optimization for 3d ic," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '11. Piscataway, NJ, USA: IEEE Press, 2011, pp. 563–570. [Online]. Available: http://dl.acm.org/citation.cfm?id=2132325.2132455

[76] A. E. Engin and S. R. Narasimhan, "Modeling of crosstalk in through silicon vias," *Electromagnetic Compatibility, IEEE Transactions on*, vol. PP, no. 99, pp. 1 –10, 2012.

[77] ——, "Modeling of crosstalk in through silicon vias," *Electromagnetic Compatibility, IEEE Transactions on*, vol. PP, no. 99, pp. 1 –10, 2012.

[78] D. H. Jung, J. Kim, H. Kim, J. J. Kim, J. Kim, and J. S. Pak, "Disconnection failure model and analysis of tsv-based 3d ics," in *Electrical Design of Advanced Packaging and Systems Symposium (EDAPS), 2012 IEEE*, Dec., pp. 164–167.

[79] M. Tehranipour, N. Ahmed, and M. Nourani, "Testing soc interconnects for signal integrity using boundary scan," in *VLSI Test Symposium, 2003. Proceedings. 21st*, 2003, pp. 158–163.

[80] J. Zhao, F. Meyer, null, and N. Park, "Maximal diagnosis of interconnects of random access memories," *Reliability, IEEE Transactions on*, vol. 52, no. 4, pp. 423–434, Dec.

[81] P. Goel and M. T. McMahon, "Electronic chip-in-place test," in *Proceedings of the 19th Design Automation Conference*, ser. DAC '82. Piscataway, NJ, USA: IEEE Press, 1982, pp. 482–488. [Online]. Available: http://dl.acm.org/citation.cfm?id=800263.809248

[82] M. Taouil, S. Hamdioui, and E. Marinissen, "How significant will be the test cost share for 3d die-to-wafer stacked-ics?" in *Design Technology of Integrated Systems in Nanoscale Era (DTIS), 2011 6th International Conference on*, 2011, pp. 1–6.

[83] K. Smith, P. Hanaway, M. Jolley, R. Gleason, E. Strid, T. Daenen, L. Dupas, B. Knuts, E. Marinissen, and M. Van Dievel, "Evaluation of tsv and micro-bump probing for wide i/o testing," in *Test Conference (ITC), 2011 IEEE International*, 2011, pp. 1–10.

[84] M. Taouil, M. Lefter, and S. Hamdioui, "Exploring test opportunities for memory and interconnects in 3d ics," in *Proc. International Design and Test Symposium*, Doha, Qatar, December 2012.

[85] "Ieee standard test access port and boundary scan architecture," *IEEE Std 1149.1-2001*, pp. 1–212, 2001.

[86] (2013) Ieee std 1500 - standard for embedded core test. [Online]. Available: http://grouper.ieee.org/groups/1500/index.html

[87] S. Park and T. Kim, "A new ieee 1149.1 boundary scan design for the detection of delay defects," in *Proceedings of the conference on Design, automation and test in Europe*, ser. DATE '00. New York, NY, USA: ACM, 2000, pp. 458–462. [Online]. Available: http://doi.acm.org/10.1145/343647.343822

[88] A. Biewenga, H. HOLLMANN, F. De Jong, and M. Lousberg, "Static component interconnect test technology (scitt) a new technology for assembly testing," in *Test Conference, 1999. Proceedings. International*, 1999, pp. 439–448.

Table A.1: Test patterns for open line fault detection with (1) transition on neighbors, and (2) both transitions for each group

$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	$T_{1,4}$	$T_{2,1}$	$T_{2,2}$	$T_{2,3}$	$T_{2,4}$	$T_{3,1}$	$T_{3,2}$	$T_{3,3}$	$T_{3,4}$	$T_{4,1}$	$T_{4,2}$	$T_{4,3}$	$T_{4,4}$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	**0**	1	1	1	1	1	**0**	1	**0**	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	**1**	0	**1**	0	0	0	0	0	**1**	1	**1**

A Auxiliary Test Patterns

A.1 Multi line faults due to complete open and crosstalk coupling

Table A.2: Test patterns for open line fault detection with (1) transition on neighbors, and (2) first transition for all groups then second transition

$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	$T_{1,4}$	$T_{2,1}$	$T_{2,2}$	$T_{2,3}$	$T_{2,4}$	$T_{3,1}$	$T_{3,2}$	$T_{3,3}$	$T_{3,4}$	$T_{4,1}$	$T_{4,2}$	$T_{4,3}$	$T_{4,4}$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	**0**	1	1	1	1	1	**0**	1	**0**	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**

Table A.3: Test patterns for open line fault detection with (1) transition on victim, and (2) both transitions for each group

$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	$T_{1,4}$	$T_{2,1}$	$T_{2,2}$	$T_{2,3}$	$T_{2,4}$	$T_{3,1}$	$T_{3,2}$	$T_{3,3}$	$T_{3,4}$	$T_{4,1}$	$T_{4,2}$	$T_{4,3}$	$T_{4,4}$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	**0**	1	1	1	1	1	**0**	1	**0**	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**

Table A.4: Test patterns for open line fault detection with (1) transition on victim, and (2) first transition for all groups then second transition

$T_{1,1}$	$T_{1,2}$	$T_{1,3}$	$T_{1,4}$	$T_{2,1}$	$T_{2,2}$	$T_{2,3}$	$T_{2,4}$	$T_{3,1}$	$T_{3,2}$	$T_{3,3}$	$T_{3,4}$	$T_{4,1}$	$T_{4,2}$	$T_{4,3}$	$T_{4,4}$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	**1**	0	**1**	0	0	0	0	0	**1**	0	**1**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	**0**	1	1	1	1	1	**0**	1	**0**	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	**0**	1	**0**	1	1	1	1	1	**0**	1	**0**