Interconnect Test for 3D Stacked Memory-on-Logic

Mottaqiallah Taouil, Mahmoud Masadeh, Said Hamdioui
Delft University of Technology
Faculty of EE, Mathematics and CS
Mekelweg 4, 2628 CD Delft, The Netherlands
{m.taouil, s.hamdioui}@tudelft.nl

Erik Jan Marinissen
IMEC vzw
Kapeldreef 75, 3001 Leuven, Belgium
erik.jan.marinissen@imec.be

***Abstract*—Three-dimensional stacked IC (3D-SIC) technology based on Through-Silicon Vias (TSVs) provides numerous advantages as compared to traditional 2D-ICs. A potential application is memory stacked on logic, providing enhanced throughput, and reduced latency and power consumption. However, testing the TSV interconnects between the two dies is challenging, as both the memory and the logic die might come from different manufacturers. Currently, no standard exists and the proposed solutions fail to address dynamic and time-critical faults (at speed testing). In addition, memory vendors have not been in favor to put additional DfT structures such as JTAG for interconnect testing on their memory devices. This paper proposes a new Memory Based Interconnect Test (MBIT) approach for 3D stacked memories. Our test patterns are applied by read and write instructions to the memory and are validated by a case study where a 3D memory is assumed to be stacked on a MIPS64 processor. The main benefits of the MBIT approach are: (1) zero area overhead, (2) the ability to detect both static and dynamic faults and perform at speed testing, (3) flexibility in applying any test pattern, as this can be executed by the CPU on the logic die and (4) extreme short test execution time.**

Keywords: *interconnect testing, 3D-SIC, memory-on-logic*

I. Introduction

The popularity of 3D Stacked ICs (3D-SICs) is rising among industry and research groups [1]. 3D-SICs based on Through Silicon Vias (TSVs) are emerging as one of the main competitors to continue the trend of Moore's Law [2]. Stacking dies with vertical interconnects possess many benefits [1], such as (a) low latency between adjacent dies, (b) reduced power consumption, (c) high bandwidth communication, (d) improved form factor and package volume density, and (e) heterogeneous integration.

One of the main applications that utilizes the mentioned benefits is the stacking of memory (DRAM) on logic (CPU). After stacking, a post-bond interconnect test is required to test interconnects (TSVs + μ-bumps) between the memory and logic dies. This is not straightforward as (1) stacked dies may come from different providers (IP confidentiality), (2) memory providers are reluctant to integrate DfT such as JTAG for interconnect testing, and (3) even with DfT support, obtaining high coverage for dynamic faults is still challenging.

Currently, no standard exists to test interconnects in memories stacked on logic. However, some test approaches are being under development. IEEE P1838 [3] is currently an ongoing standard that develops DfT for general stacked ICs; it is based on the presence of Boundary Scan (BS) cells in all dies. Wide I/O [4] also supports interconnect testing using BS. However, (DRAM) memory vendors are not always in favor of integrating JTAG on their devices [5]. Other approaches such as the IEEE P1581 [5], originally for 2D ICs, can be extended in the third dimension. In test mode, the memory is bypassed and interconnects are tested by creating a direct logic function between the inputs and outputs of the memory. IEEE P1581 prefers a JTAG compliant logic chip, i.e., the test logic on the memory chip can function with a logic chip that supports JTAG. This standard can be mapped to 3D-SICs by having the bottom die (logic) JTAG compliant and where the test logic has to reside on the top die (memory). This approach, referred to as Test Logic (TL) based interconnect testing, also requires additional DfT test logic on the memory die. In addition to the undesired DfT on the memory die, both the BS and TL based test methods are unable to provide at speed testing required to target dynamic faults. Testing for dynamic faults is crucial, as 3D interconnects are expected to suffer from speed and timing related faults [6–11].

In [12] and [13] authors present hardwired BISTs with at-speed testing capability for crosstalk faults. Both methods are not flexible in altering test patterns and require additional DfT area. In [13] its reported that the area overhead of the method in [12] approximates 9.8% while their own equals 7%. They evaluate this in 90 nm technology using 15 μm TSV diameters.

This paper proposes a post-bond Memory Based Interconnect Test (MBIT) methodology being able to test interconnects between memory and logic dies by performing read and write operations from the logic die (CPU) to the memory dies. A similar approach is taken in [14], but it is inapplicable for TSV arrays. This paper also provides a classification of interconnect defects, and compiles them into fault models. In addition, it discusses the test pattern generation for these faults and uses the proposed MBIST to implement them. MBIT does not require any DfT area as it reuses existing components in the stack. It supports at-speed testing and detects static and dynamic faults. Moreover, it is very flexible in altering test patterns simply by modifying software instructions and has a extreme short test execution time.

The remainder of the paper is organized as follows. Section II presents defects, fault models and detection conditions for 3D interconnects. Section III describes thereafter the test pattern generation for the targeted fault models. Section IV provides the simulation results and compares our methodology with the state-of-the-art. Finally, Section V concludes this paper.

978-3-9815370-2-4/DATE14/©2014 EDAA

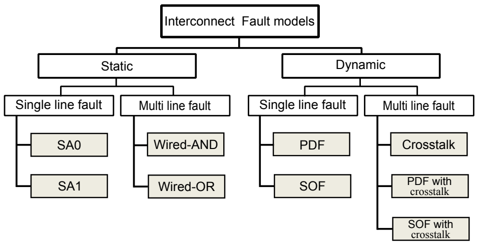

Fig. 1. Fault Model Classification for interconnects

II. Defect, Fault Models and Detection

A. Defects in Interconnects

Interconnects in 3D-stacking are a potential source of defects inherent to the manufacturing steps such as TSV fabrication/filling, bonding etc. Defects transpire both in TSVs and micro-bumps and examples of defects are given next.

Defects related to Through-Silicon-Via (TSV):

D1 Pinhole defects occur along TSV walls and cause a short (low resistance path) between TSVs and the substrate; This may cause degradation of the signal quality in terms of strength and speed [6,7,9,15].

D2 An incomplete fill of TSVs (voids) may originate from insufficient wetting during plating. Voids cause partial opens and lead to higher TSV resistance [6,7,9,15].

D3 Coefficient of thermal expansion (CTE) mismatch between TSV metal (most likely copper) and substrate may lead to TSV cracks and sidewall delamination. Both lead to increased path resistance [9,15–18].

D4 Pinch-off of TSVs during plating could lead to increased TSV resistance or partial opens [7].

D5 Missing contacts between TSVs and the transistors or metal layer cause opens [7,8].

D6 TSV misalignment with μ-bumps increases the resistance and causes (partial) opens [7,9,15].

D7 Crosstalk between different TSVs [9,10].

Defects related to μ-bumps:

D8 Damage in underlying BEOL [19].

D9 Weak bonding due to buckled thinned Si chip [19].

D10 Variation in TSV heights may cause tin to be squeezed out from μ-bump causing shorts between μ-bumps [19,20].

D11 Electromigration may cause voids and cracks in the joints, resulting in higher resistive μ-bumps, or opens [21].

D12 μ-bump cracks due to CTE mismatch between copper, silicon, and silicon-oxide [7].

B. Faults and Fault Models

Interconnect fault models can be classified into static and dynamic faults. Fig. 1 shows a classification of the faults. A defect can cause a single line or a multi line fault. Each fault is depicted in Figure 2 and explained next. Static faults include:

- Stuck-at-Fault (SAF). There are two types of SAF faults: stuck-at-0 (SA0) and stuck-at-1 (SA1) as depicted in Fig. 2(b). A SAF fault can be caused by defect D1.
- Bridge fault. *Simple* bridge faults include wired-AND (Fig. 2(c)) and wired-OR (Fig. 2(d)) faults. *Complex* bridge faults also exists, such as A dominate-AND B in which wire A is fault-free and where B takes the value A ∩ B. A bridge fault can be caused by defect D10.

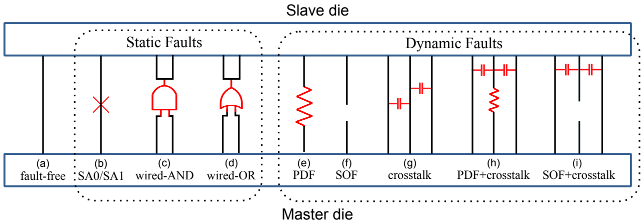

Fig. 2. Static and dynamic faults

Dynamic faults include:

- Path Delay Fault (PDF): A partial open line defect increases the line delay (Fig. 2(e)). It can affect both rising or falling delay time. PDF faults can be caused by defects D2, D3, D6, D8, D11, D12.
- Stuck Open Fault (SOF): This is caused by a complete open line defect (Fig. 2(f)). SOFs can be caused by defects D5, D6, D8, D9, D11, D12.
- Crosstalk Fault: Faults on victim lines are caused by crosstalk from aggressive neighbors (Fig. 2(g)). Several crosstalk faults exists as described by the Maximum Aggressor (MA) fault model [22] such as (1) glitch-up, (2) glitch-down, (3) falling delay, and (4) rising delay. Each fault has a specific behavior, while it represents the same phenomena. Defect D7 may cause crosstalk faults.
- PDF with Crosstalk: Faults due to partial resistive opens (victims) are affected by crosstalk from neighbors (Fig. 2(h)). PDF with Crosstalk faults can be caused by combinations of crosstalk and PDF faults.
- SOF with Crosstalk: Faults due to complete open lines (victims) are affected by crosstalk from neighbors (Fig. 2(i)). SOF with Crosstalk faults can be caused by combinations of crosstalk and SOF faults.

The dynamic faults embody most of the defects and therefore it is essential to test for them.

C. Detection Conditions

The detection conditions of each fault are described next. In general, the detection process adheres to the following steps:

1) Fault sensitization (activation): create a different behavior between the faulty and fault-free circuit.
2) Fault propagation: make the fault visible at the outputs.
3) Line justification: backtrack the values to the input of the circuit, such that the inputs sensitize the fault.

Fault propagation and line justification for address and data lines are dissimilar. Data lines can be controlled and observed directly through writing and reading. Therefore, fault propagation and line justification are straightforward. However, address lines are uni-directional and fault propagation must be performed indirectly by utilizing data lines (e.g., by writing and reading different values to different addresses). Control lines, such as write or read signals, are tested implicitly. For fault sensitization, special sequences and/or transitions are required for each fault.

TABLE I

SAF TEST PATTERNS FOR DATA LINES

OP	Op.	Address	Data
OP1	W	$Addr_x$	F F F F
OP2	R	$Addr_x$	F F F F

OP	Op.	Address	Data
OP1	W	$Addr_y$	0 0 0 0
OP2	R	$Addr_y$	0 0 0 0

SAF (SA0/SA1): A stuck-at-fault forces a wire to a specific value; either 0 (SA0) or 1 (SA1). Therefore, to sensitize a SAF fault an opposite value must be applied to the wire.

Bridge fault (Wired-AND/Wired-OR): To sensitize a bridge fault, two opposite values must be specified on each pair of lines. Simple bridge faults such as the ones depicted in Fig. 2(c) and (d) require at least one of the two patterns 0-1 or 1-0 as inputs. More complex bridges such as A dominate-AND B require both 0-1 and 1-0 inputs on each pair of wires for fault sensitization.

PDF: We assume that the path delay fault consists of a low to moderate resistance value, violating the normal operation with at most one additional clock cycle. Faults that lead to larger delays, i.e., more than one extra cycle, can be considered as SOFs. To sensitize PDF faults, both 0→1 and 1→0 transitions should be applied on each line.

SOF: For SOFs we assume that during short time intervals the non-driven part of the floats remain stable. Therefore, to sensitize SOF either a 0→1 or 1→0 transition is needed.

Crosstalk: We consider only the most relevant crosstalk faults. To sensitize a rising crosstalk fault, a victim must undergo a 0→1 transition, while the aggressors simultaneously make a 1→0 transition. The reverse applies for falling delay faults.

PDF with Crosstalk: The fault sensitization for PDF faults with crosstalk is the same as falling and rising delay faults, as this maximizes the applied stress from the aggressors.

SOF with Crosstalk: The fault sensitization for this fault requires both a 0→1 transition on the victim while keeping the aggressors stable at 0 and a 1→0 transition on the victim while keeping the aggressors stable at 1. Keeping the aggressors stable reduces the coupling with the floating part of the SOF, hence it minimizes the contribution of the aggressors to the transition on the floating part of the victim.

III. Test Pattern Generation

Due to space limitation, we discuss only a subset of fault models. We restrict ourselves to static faults and one dynamic fault (SOF with Crosstalk). Nevertheless, the experiment results in Section IV will be presented for all faults. Next, a single fault is assumed to occur at a time. In addition, during explanation we assume L_d=16 bit data lines (presented in *hexadecimal* value) and L_a=16 bit address lines (presented in *binary* value).

A. Static Faults

In this section we will present the test patterns of static faults. A SAF fault may happen in data lines or address lines. A bridge fault may happen: (1) between data lines, (2) between address lines, and (3) between data and address lines.

SAF at data lines: Table I shows the memory operations required to detect SA0 (left table) and SA1 faults (right table) on data lines. The tables consist of four columns; the first column shows the index of the operation; the second column the type of operation, i.e., read (R) or write (W); the third and fourth columns show the address and data values, respectively. Both tables contain a write followed by a read operation. SAF faults will be detected during read OP2.

TABLE II

SAF TEST PATTERNS FOR ADDRESS LINES

OP	Op.	Address	Data
OP1	W	0000 0000 0000 0000	Init_Data
OP2	W	0000 0000 0000 000**1**	$Data_1$
OP3	W	0000 0000 0000 00**1**0	$Data_2$
. . .	. . .	. . .	. . .
OP16	W	0**1**00 0000 0000 0000	$Data_{15}$
OP17	W	**1**000 0000 0000 0000	$Data_{16}$
OP18	R	0000 0000 0000 0000	Init_Data

OP	Op.	Address	Data
OP1	W	1111 1111 1111 1111	Init_Data
OP2	W	1111 1111 1111 111**0**	$Data_1$
OP3	W	1111 1111 1111 11**0**1	$Data_2$
. . .	. . .	. . .	. . .
OP16	W	1**0**11 1111 1111 1111	$Data_{15}$
OP17	W	**0**111 1111 1111 1111	$Data_{16}$
OP18	R	1111 1111 1111 1111	Init_Data

SAF at address lines: Table II shows the test patterns to detect both SA0 (left table) and SA1 faults (right table) in address lines. Detecting SA0/SA1 faults at address lines is more complex as they affect the memory address. For example, writing a value to the address all 1's and subsequently reading from this address will not detect any SA0 fault in the address lines; this is because both the write and read operation are affected in the same way and the fault is not sensitized. To test memory address lines, each address line should be tested separately. For example, by using a walking-1 sequence for SA0 as depicted in the left table. Address 0 of the memory is first initialized to *Init_Data* during OP1. During write operation OP2 to OP17 (with different data than *Init_Data*), any SA0 in the address lines will overwrite *Init_Data* of address 0. Therefore, read operation OP18 is able to detect any SA0 fault. The same applies for SA1 faults, but with complement addresses.

Bridges between data lines: The detection of wired-AND or wired-OR bridges between data lines requires that each pair of data lines must fulfill at least the combination 0-1 or 1-0. Modified Counting Sequence (MCS) satisfies this requirement at a cost of $log_2(L_d + 2)$ test patterns [23]. The total number of memory operations required to execute such test patterns equals $2 \cdot log_2(L_d + 2)$ memory operations; for each pattern there is a write and read operation (to any address). The effectiveness of these patterns is proven in literature [23]. Complex bridge faults, such as A-dominant AND B, require both 0-1 and 1-0 inputs on each pair of wires. The True/Complement Algorithm [24] can be used for this; it consists of $2 \cdot log_2(L_d + 2)$ test patterns resulting into $4 \cdot log_2(L_d + 2)$ memory operations.

Bridges between address lines: Wired-And and wired-OR faults between address lines must be considered separately.

Wired-AND bridge fault: Wired-AND bridge faults can be detected by a walking-1 pattern, similar to the detection of SA0 faults in address lines (left side of Table II); due to wired-AND fault operations OP2 till OP17 will overwrite *Init_Data* of OP1. This is detected by OP18.

Wired-OR bridge fault: Wired-OR bridge faults can be detected by a walking-0 pattern, similar to the detection of SA1 faults in address lines (right side of Table II). The walking-1 sequence for wired-AND faults and walking-0 for wired-OR detect both simple and complex bridge faults.

TABLE III
BRIDGE FAULTS TEST PATTERNS THAT FLIP DATA LINES

OP	Op.	Address	Data
OP1	*W*	0000 0000 0000 0000	F F F F
OP2	*R*	0000 0000 0000 0000	F F F F

OP	Op.	Address	Data
OP1	*W*	1111 1111 1111 1111	0 0 0 0
OP2	*R*	1111 1111 1111 1111	0 0 0 0

TABLE IV
BRIDGE FAULTS TEST PATTERNS THAT FLIP ADDRESS LINES

OP	Op.	Address	Data
OP1	*W*	0000 0000 0000 0000	F F F F
OP2	*W*	0000 0000 0000 000**1**	0 0 0 0
OP3	*W*	0000 0000 0000 00**1**0	0 0 0 0
. . .	. . .	. . .	. . .
OP16	*W*	0**1**00 0000 0000 0000	0 0 0 0
OP17	*W*	**1**000 0000 0000 0000	0 0 0 0
OP18	*R*	0000 0000 0000 0000	F F F F

OP	Op.	Address	Data
OP1	*W*	1111 1111 1111 1111	0 0 0 0
OP2	*W*	1111 1111 1111 111**0**	F F F F
OP3	*W*	1111 1111 1111 11**0**1	F F F F
. . .	. . .	. . .	. . .
OP16	*W*	1**0**11 1111 1111 1111	F F F F
OP17	*W*	**0**111 1111 1111 1111	F F F F
OP18	*R*	1111 1111 1111 1111	0 0 0 0

Bridges between data and address lines: Bridge faults behave as wired-AND or wired-OR, and may cause data or address lines to flip.

Bridge faults that flip data lines: The left side of Table III provides the memory operations that detect wired-AND bridge faults that lead to faulty data lines. Any data line that suffers from a wired-AND with an address line will cause the data line to flip to zero (on the data side), which is easily detectable. These test patterns are similar to those of SA0 in data lines when $Addr\_x$ of Table I is set to value 0. The right side of Table III provides the test patterns that detect wired-OR bridge faults that lead to faulty data lines. Here, the operations take the complement values of those of wired-AND. Any data line suffering from a wired-OR with an address line will cause the data line to flip to one, which is easily detectable. These test patterns are similar to those of SA1 in data lines when $Addr\_y$ of Table I is set to a value of all 1's.

Bridges that flip address lines: The left part of Table IV provides the test patterns needed for the detection of wired-AND bridges that cause address lines to flip. A walking-1 pattern on the address lines ensures the detection of these types of faults. In OP1, the address consisting of all 0's is initialized with all 1's data (FFFF in hex). Note that for the initialization pattern (OP1) the address is not impacted in the presence of wired-AND faults. Any address line that suffers from a wired-AND with a data line will cause the address line to flip to zero during the walking-1 sequence (OP2 up to OP17). This will overwrite the original initialization. Therefore, the last read (OP18) results in a data value of FFFF for non-faulty interconnects and 0000 in case a fault is present. These test patterns are similar to those in the left side of Table II used to detect SA0 faults in address lines when *Init_Data* = FFFF and $Data_x$ = 0000. The right part of Table IV provides the test patterns needed to detect wired-OR bridges that cause address lines to flip. Here, all address and data values are the complements of the wired-AND patterns. The memory operations are the same as to test for SA1 faults in address lines (right part of Table II) under the condition that *Init_Data* = 0000 and $Data_x$ = FFFF.

B. Dynamic Faults

Dynamic faults consist of single and multi line faults. For single line faults the same general approach as static faults can be used in which data lines are tested in parallel and address lines individually. However, for multi-line faults the layout of the address and data lines becomes important. For simplicity, we assume a regular TSV array of size 4×4 to demonstrate how to generate test patterns for SOF with Crosstalk. Furthermore, we assume a 1^{st} aggressor model, i.e., victims can only be affected by closest neighbor aggressors. Grouping the 4×4 matrix in four groups as shown in Fig. 3 allows us to test each group simultaneously. For example, when TSVs of group 1 are tested as victims it is assumed that the remaining TSVs act as aggressors. The same applies for the other three TSV groups. In general any k^{th} aggressor model can be used, where k the maximum TSV distance between victims and aggressors. Results reported in [25] show that restricting to k=1 is sufficient.

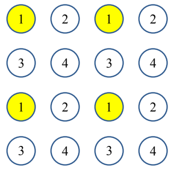

Fig. 3. TSV groups

TABLE V
SOF WITH CROSSTALK TEST PATTERNS FOR DATA LINES

OP	Operation	Address	Data
OP1	W	$Addr_1$	**0**0**0**0 0000 **0**0**0**0 0000
OP2	W	$Addr_2$	**1**0**1**0 0000 **1**0**1**0 0000
OP3	R	$Addr_1$	0000 0000 0000 0000
OP4	R	$Addr_2$	1010 0000 1010 0000
OP5	W	$Addr_1$	**1**1**1**1 1111 **1**1**1**1 1111
OP6	W	$Addr_2$	**0**1**0**1 1111 **0**1**0**1 1111
OP7	R	$Addr_1$	1111 1111 1111 1111
OP8	R	$Addr_2$	0101 1111 0101 1111

TABLE VI
SOF WITH CROSSTALK TEST PATTERNS FOR ADDRESS LINES

OP	Operation	Address	Data
OP1	W	00000000 0 0000000	Init_Data
OP2	W	00000000 1 0000000	$Data_1$
OP3	R	00000000 0 0000000	Init_Data
OP4	W	11111111 1 1111111	Init_Data
OP5	W	11111111 0 1111111	$Data_1$
OP6	R	11111111 1 1111111	Init_Data

SOF with Crosstalk at data lines: Table V shows the memory operations required to detect SOF with Crosstalk for TSV group 1. To sensitize such a fault, a transition must be created on the victim while keeping the aggressors stable. OP1-OP2 create a 0→1 transition from master to slave on the victim data lines, while keeping aggressors stable at 0. OP3 and OP4 make a similar transition, but from slave to master. In case the transition fails (during write or read) it will be detected during reading (OP3-OP4). In a similar manner, but with all data lines complemented, OP5-OP8 can be applied to detect the 1→0 transition fault on the victim. Similar patterns can be developed for the other remaining three groups.

SOF with Crosstalk at address lines: Table VI shows the test pattern to detect SOF with crosstalk for a single address line. OP1 initializes the memory by writing *Init_Data* to address 0. OP1 and OP2 create a 0→1 transition on the victim address line while the aggressors are kept stable at 0. OP3 expects *Init_Data* in case fault-free, and $Data_1$ if the victim line failed to make the 0→1 transition. Similarly, OP4-OP6 detect the reverse transition. These group of test patterns have to be repeated for each address line individually.

It is worth noting that the minimum set required to detect all static and dynamic faults targeted in this paper consist of only two test: (1) PDF with Crosstalk and (2) SOF with Crosstalk.

TABLE VII
TEST COST FOR STATIC FAULTS

Fault (set)	**#mem ops.**	**# MIPS instr.**	**#MIPS cycles**
Optimized SAF	32	45	57
Optimized static / Optimized Bridge (simple bridge)	48	109	137
Optimized static / Optimized Bridge (complex bridge)	62	137	179

TABLE VIII
TEST COST FOR DYNAMIC FAULTS

Fault (set)	**#mem ops.**	**# MIPS instr.**	**#MIPS cycles**
PDF at data lines	6	16	21
PDF at address lines	10	15	23
SOF at data lines	4	14	19
SOF at address lines	48	75	91
Crosstalk / (PDF + crosstalk) at data lines	24	58	66
Crosstalk / (PDF + crosstalk) at address lines	96	123	175
Stuck open fault (SOF) with Crosstalk at data lines	32	61	73
Stuck open fault (SOF) with Crosstalk at address lines	72	92	104

IV. EXPERIMENTAL RESULTS

A. Case Study

We simulate memory test patterns, for a memory die stacked on a logic die that consists of a MIPS64 processor, by using the MIPS64 simulator in [26]. The simulator can handle a maximum of L_d=64-bit data lines and L_a=12-bit address lines (lowest 3 bits are byte offset). The simulator supports three types of instructions: (1) ALU instructions such as add, subtract and shift, (2) Branch instructions such as branch if equal, and (3) Memory instructions such as load, store, etc; a complete reference can be found in [27].

The memory operations, which represent the test patterns, need to be translated into real MIPS instructions. An example for the SAF at data lines is provided in the code fragment below.

```
1. ori  r1,r0,0xFFFF   8. SD  R1, 0xFF8(R0)
2. dsll r1,r1,16       9. LD  R10,0XFF8(R0)
3. ori  r1,r1,0xFFFF  10. BNE R1,R10,SA0_DATA
4. dsll r1,r1,16      11. HALT
5. ori  r1,r1,0xFFFF
6. dsll r1,r1,16         SA0_DATA:
7. ori  r1,r1,0xFFFF     ;handle fault here
```

The test consists of 11 instructions. The first 7 instructions create the desired pattern FFFF FFFF FFFF FFFF in register R1 (similarly as in the left side of Table II). Instructions 8 and 9 contain the two memory operations in which R1 is written (SD) and read (LD) from memory. In case a stuck at fault is present a branch will be taken (instruction 10) to SA0_DATA.

Tables VII and VIII summarize the number of memory operations and clock cycles for all static and dynamic faults respectively. The tables provide for each fault the required number of memory operations, the number of MIPS instructions to execute those memory operations and finally, the number of MIPS cycles. For example, to test for all static faults requires only 179 MIPS cycles. The memory latency is 1 clock cycle.

B. Comparison with Prior Related Work

We compare our MBIT approach with BS, TL and the BIST methods [12,13] for several DfT requirements related to test quality (T1) and cost (T2).

T1 Test quality: The test methodology must support full controllability and observability and test for static and dynamic faults. In addition, diagnosis should identify faulty locations. Modifying test patterns for extra diagnosis or to target different faults is needed.

TABLE IX
COMPARISON BETWEEN INTERCONNECT TEST APPROACHES

Test Requirement	**Boundary Scan**	**Test Logic**	BIST [12]	BIST [13]	**MBIT**
T1 controllability/observability	Both	Memory outputs are only observable	both	both	Address lines are tested indirectly
T1 static/dynamic	Only static	Only static	crosstalk only	crosstalk only	Static + Dynamic
T1 detection/diagnosis	Support for both	Support for both	Support for both	Support for both	Support for both
T1 flexible test patterns	yes	yes, limited output controllability	no	no	yes
T2 area overhead	$2 \cdot (L_a + L_c + 2 \cdot L_d)$ BS cells (bottom/top die) + JTAG (top die)	$L_a + L_c + 2 \cdot L_d$ BS cells (bottom die) and test logic (top die)	9.8% with respect to TSV array	7% with respect to TSV array	No area overhead
T2 test cost (simple bridges)	$2 \cdot (L_a + L_c + 2 \cdot L_d) \cdot log_2(L_a + L_c + L_d + 2)$ test clock cycles	$(L_a + L_c + 2 \cdot L_d) \cdot log_2(L_a + L_c + L_d + 2)$ test clock cycles	not applicable	not applicable	$2 \cdot log_2(L_d + 2) + 2 \cdot L_a + 8$ at speed memory operations
T2b test cost (complex bridges)	$4 \cdot (L_a + L_c + 2 \cdot L_d) \cdot log_2(L_a + L_c + L_d + 2)$ test clock cycles	$2 \cdot (L_a + L_c + 2 \cdot L_d) \cdot log_2(L_a + L_c + L_d + 2)$ test clock cycles	not applicable	not applicable	$4 \cdot log_2(L_d + 2) + 2 \cdot L_a + 8$ at speed memory operations

T2 Test cost: The DfT overhead should be as low as possible and preferably without DfT on the memory die. The test time should be cost-effective; i.e., the test time should be reasonable and scalable with the number of TSVs.

Test quality comparison

Table IX summarizes the comparison between the five test methods. All approaches are in general able to control and observe the interconnects. TL has a limited controllability of memory outputs and MBIT propagates faults in address lines indirectly. BS and TL can be used for static faults only, while the approaches in [12] and [13] perform testing by hardwired state machines and target crosstalk faults only. MBIT is flexible enough to test for any fault. BS and TL can be modified for dynamic fault testing, but require extra hardware or complete cell modification [28,29]. BS interconnect testing has an additional limitation for the case where drivers and receiver cells cannot be tested simultaneously; in this case, approximately 75% of the drivers and receivers can be covered [4]. A similar problem exists in [12] and [13] as both solutions only can handle uni-directional lines. MBIT is able to test for both TSV drivers and receivers as patterns are applied in both directions. Diagnosis is possible for all cases, however, the schemes in [12,13] cannot apply flexible patterns as the BISTs are hardwired, while in TL some test patterns might not be applicable due to memory input output dependancy during test.

Test cost comparison

For a fair area overhead comparison, we assume a bottom die with default JTAG. In that case, the overhead for each method will be the following:

- BS: the overhead consists of the additional BS cells on both the bottom die and top die assigned to the interconnects, in total equal to $2 \cdot (L_a$+L_c+$2 \cdot L_d)$. Here L_a presents the number of address line, L_c the number of control lines, L_d the number of data lines. Control and address lines require a single BS cell per wire, while bi-directional data lines are assumed to have two BS cells [30]. In addition to BS cells, the JTAG infrastructure on the top die is also part of the overhead.

- TL: the overhead includes the BS-cells on the bottom die of length L_a+L_c+$2 \cdot L_d$ and the test logic on top die.
- BIST [12,13]: the overhead consists in both methods of a state-machine, several flip-flops and other control logic such as muxes. In [13] its reported that the area overhead of the method in [12] approximates 9.8%, while their own equals 7%; both are measured with respect to the total TSV area. It is evaluated in 90 nm technology using 15 μm TSV diameters using a 64×16 TSV matrix.
- MBIT: no area overhead.

The test time for each of the approaches is as follows:

- BS: the total test time for BS depends on the number of test patterns and the length of the BS cells. For the True/Complement Algorithm, the number of test patterns equal $2 \cdot \lceil log_2(L_a + L_c + L_d + 2)\rceil$ to detect all static faults. The length of the BS cells equals $2 \cdot (L_a + L_c + 2 \cdot L_d)$. Therefore, the test time equals $4 \cdot (L_a + L_c + 2 \cdot L_d) \cdot \lceil log_2(L_a + L_c + L_d + 2)\rceil$ test clock cycles.
- TL: The test time reduces by a factor of two when compared to BS, due to half the number of BS cells.
- BIST [12,13]: The test time of the hardwired BISTs in [12,13] is much lower than other approaches. For example, the method in [13] requires 122 cycles (assuming 1 cycle per TSV row pattern) to detect all targeted faults in this paper (i.e., the test set PDF with crosstalk and SOF with crosstalk faults).
- MBIT: To detect all static faults 179 MIPS cycles are required (assuming complex bridges). To detect all static and dynamic faults (PDF with crosstalk and SOF with crosstalk faults), MBIT requires 66+175+73+104=418 at speed cycles (see Table VIII).

In conclusion, with respect to the area overhead MBIT performs best followed by BIST [12], BIST [13], TL and BS. If we compare MBIT with BS and TL with respect to test time considering the same MIPS memory (L_a=12, L_d=64 and for simplicity ignore control lines L_c=0), BS based testing would require 3920 test clock cycles and Test Logic based testing 1960 test clock cycles for True/Complement Algorithm. Moreover, if we assume an operational clock frequency of 500 MHz and test clock speed of 100 MHz the differences between the methods becomes more apparent. The total test time would be 0.36μs, 39.20μs and 19.6μs for MBIT, BS and TL respectively. If we compare MBIT with the hardwired BIST solutions for both dynamic and static faults, we see that MBIT is slower in test time (418 cycles for MBIT versus 122 cycles for BIST [13]), but has the flexibility of applying different test patterns and does not require additional DfT.

V. Conclusion

This paper proposed a new Memory Based Interconnect Test (MBIT) approach for 3D-SICs where memory is stacked on logic by testing interconnects through memory read and write operations. Our MBIT solution is able to perform at-speed testing and detect all static and dynamic faults. It has zero area overhead and allows flexible patterns to be applied. In addition, the required test time is much lower than traditional based solutions such as Boundary Scan, but is three times slower than hardwired BIST solutions. However BIST solutions have a large area overhead and cannot apply flexible patterns.

References

[1] P. Garrou *et al.*, *Handbook of 3D Integration*. John Wiley & Sons, 2008.

[2] G. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[3] (2013) P1838 - standard for test access architecture for three-dimensional stacked integrated circuits. [Online]. Available: http://standards.ieee.org/develop/project/1838.html

[4] S. Deutsch *et al.*, "Dft architecture and atpg for interconnect tests of jedec wide-i/o memory-on-logic die stacks," in *ITC*, 2012, pp. 1–10.

[5] H. Ehrenberg and B. Russell, "IEEE Std 1581- A standardized test access methodology for memory devices," in *ITC*, 2011, pp. 1–9.

[6] E.J. Marinissen and Y. Zorian, "Testing 3D chips containing through-silicon vias," in *ITC*, Nov. 2009, pp. 1–11.

[7] K. Chakrabarty *et al.*, "TSV defects and TSV-induced circuit failures: The third dimension in test and design-for-test," in *IRPS*, April 2012, pp. 5F.1.1 –5F.1.12.

[8] A. Papanikolaou *et al.*, *Three Dimensional System Integration*. Springer US, 2011.

[9] F. Ye and K. Chakrabarty, "TSV open defects in 3D integrated circuits: Characterization, test, and optimal spare allocation," in *49th ACM/EDAC/IEEE DAC*, June 2012, pp. 1024–1030.

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

[11] C. Liu *et al.*, "Full-chip tsv-to-tsv coupling analysis and optimization in 3d ic," in *48th ACM/EDAC/IEEE DAC*, 2011, pp. 783–788.

[12] V. Pasca *et al.*, "Configurable Thru-Silicon-Via interconnect Built-In Self-Test and diagnosis," in *12th LATW*, 2011, pp. 1–6.

[13] Y.J. Huang *et al.*, "Post-bond test techniques for TSVs with crosstalk faults in 3D ICs," in *VLSI-DAT*, 2012, pp. 1–4.

[14] L. Chen *et al.*, "Testing for interconnect crosstalk defects using on-chip embedded processor cores," in *Design Automation Conference*, 2001, pp. 317–322.

[15] S. Kannan *et al.*, "Fault Modeling and Multi-Tone Dither Scheme for Testing 3D TSV Defects," *J. Electronic Testing*, vol. 28, no. 1, 2012.

[16] C.W. Kuo and H.Y. Tsai, "Thermal stress analysis and failure mechanisms for through silicon via array," in *ITherm*, June 2012, pp. 202–206.

[17] X. Liu *et al.*, "Failure mechanisms and optimum design for electroplated copper Through-Silicon Vias," in *ECTC*, May 2009, pp. 624 –629.

[18] M. Jung *et al.*, "Full-chip through-silicon-via interfacial crack analysis and optimization for 3D IC," in *ICCAD*. Piscataway, NJ, USA: IEEE Press, 2011, pp. 563–570.

[19] E. Beyne and I. Dewolf. (2013, July) Failure analysis for 3d tsv systems. [Online]. Available: http://www.sematech.org/meetings/archives/3d/10124/pres/Beyne.pdf

[20] E.J. Marinissen, "Testing tsv-based three-dimensional stacked ics," in *DATE*, march 2010, pp. 1689 –1694.

[21] D. Jung *et al.*, "Disconnection failure model and analysis of tsv-based 3d ics," in *EDAPS*, Dec 2012, pp. 164–167.

[22] M. Cuviello *et al.*, "Fault modeling and simulation for crosstalk in system-on-chip interconnects," in *ICCAD*, 1999, pp. 297–303.

[23] P. Goel and M.T. McMahon, "Electronic chip-in-place test," in *DAC*. Piscataway, NJ, USA: IEEE Press, 1982, pp. 482–488. [Online]. Available: http://dl.acm.org/citation.cfm?id=800263.809248

[24] P.T. Wagner, "Interconnect testing with boundary scan," in *International Test Conference (ITC '87)*, Sep. 1987, pp. 52–57.

[25] R. Weerasekera *et al.*, "Compact modelling of through-silicon vias (tsvs) in three-dimensional (3-d) integrated circuits," in *3D-IC*, 2009, pp. 1–8.

[26] (2013) Winmips64. [Online]. Available: http://indigo.ie/ mscott/

[27] (2013) Mips64 architecture for programmers volume ii: The mips64 instruction set. [Online]. Available: http://scc.ustc.edu.cn/zlsc/lxwycj/200910/W020100308600769158777.pdf

[28] M. Tehranipour *et al.*, "Testing soc interconnects for signal integrity using boundary scan," in *VLSI Test Symposium*, 2003, pp. 158–163.

[29] S. Park and T. Kim, "A new ieee 1149.1 boundary scan design for the detection of delay defects," in *DATE*, 2000, pp. 458–462.

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