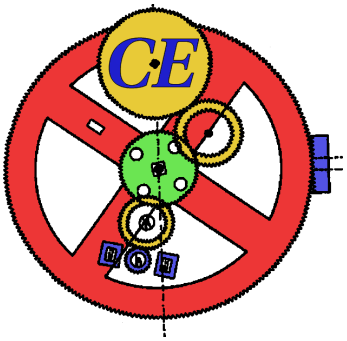


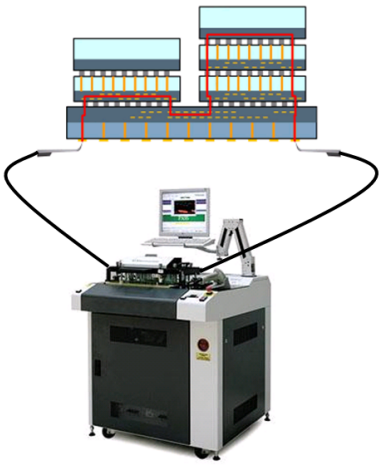
Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2012

MSc THESIS

Design and Automated Implementation of a DfT Architecture for 3D-SICs

Christos Papameletis



CE-MS-2012-09

Abstract

Three-dimensional stacked integrated circuits (3D-SICs) implemented with through silicon vias (TSVs) and micro-bumps open new horizons for faster, smaller and more energy-efficient chips. As all microelectronic structures, these 3D chips and their interconnects need to be tested for manufacturing defects. This thesis was executed in the context of the joint development program (JDP) on 3D Design-for-Test (DfT) between Cadence and IMEC. Extensions for an existing 3D-DfT architecture based on die-level wrappers were designed, and their implementation was automated in Cadence' electronic design automation (EDA) tools. The 3D-DfT architecture and the corresponding tools were augmented with support for (1) *test data compression* (TDC) and (2) *embedded-core wrappers*, as most modern designs employ these DfT techniques. Support for (3) *multi-tower* 3D-SICs was added as well, as they seem to gain ground in the 3D integration technology. Finally, (4) *automated tool flows* were designed and implemented for inserting the 3D-DfT to a design, and for performing ATPG at the die-level and at the stack-level in order to verify the correct operation of the 3D-DfT. These flows were applied to a test case and the results were verified through ATPG and simulation of the application of the generated test patterns.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Design and Automated Implementation of a DfT Architecture for 3D-SICs

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Christos Papameletis
born in Athens, Greece

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

Design and Automated Implementation of a DfT Architecture for 3D-SICs

by Christos Papameletis

Abstract

Three-dimensional stacked integrated circuits (3D-SICs) implemented with through silicon vias (TSVs) and micro-bumps open new horizons for faster, smaller and more energy-efficient chips. As all microelectronic structures, these 3D chips and their interconnects need to be tested for manufacturing defects. This thesis was executed in the context of the joint development program (JDP) on 3D Design-for-Test (DfT) between Cadence and IMEC. Extensions for an existing 3D-DfT architecture based on die-level wrappers were designed, and their implementation was automated in Cadence' electronic design automation (EDA) tools. The 3D-DfT architecture and the corresponding tools were augmented with support for (1) *test data compression* (TDC) and (2) *embedded-core wrappers*, as most modern designs employ these DfT techniques. Support for (3) *multi-tower* 3D-SICs was added as well, as they seem to gain ground in the 3D integration technology. Finally, (4) *automated tool flows* were designed and implemented for inserting the 3D-DfT to a design, and for performing ATPG at the die-level and at the stack-level in order to verify the correct operation of the 3D-DfT. These flows were applied to a test case and the results were verified through ATPG and simulation of the application of the generated test patterns.

Laboratory : Computer Engineering
Codenumber : CE-MS-2012-09

Committee Members :

Advisor:	Said Hamdioui, CE, TU Delft
Member:	Georgi Gaydadjiev, CE, TU Delft
Member:	René van Leuken, CAS, TU Delft
Member:	Erik Jan Marinissen, IMEC
Member:	Stephan Wong, CE, TU Delft

Οὐ στηλῶν μόνον ἐν τῇ οἰκείᾳ σημαίνει ἐπιγραφή, ἀλλὰ καὶ ἐν τῇ μὴ προσηκούσῃ ἄγραφος μνήμη παρ᾽ ἑκάστῳ τῆς γνώμης μᾶλλον.

Περικλῆς

What you leave behind is not what is engraved in stone monuments, but what is woven into the lives of others.

Pericles

Contents

List of Figures

List of Tables

Acknowledgements

I am particularly grateful to Erik Jan Marinissen (IMEC) for the many, long, horizon-extending discussions we had and his significant contributions to the project. A special thank you goes to Brion Keller (Cadence) and Dr. Vivek Chickermane (Cadence) for the industrial character their contributions imparted to the project. I would also like to thank Dr. Sandeep Goel (TSMC) for providing the testcase designs for the verification of the implementation. Furthermore, I would like to acknowledge my professor, Dr. Said Hamdioui (TU Delft), for initiating me into the world of testing and providing me with the oppurtunity to carry-out this project. Finally, I would like to acknowledge Cadence Design Systems for sponsoring this project and IMEC for hosting me as a resident researcher.

Christos Papameletis
Delft, The Netherlands
August 31, 2012

1 Introduction

Integrated circuits (ICs) have been around for more than half a century and they have changed many aspects of peoples' lives. With their advent, many tasks have been automated using small, smart, powerful, reliable and cheap electronic components. The evolution of microelectronics has been following Moore's law [21] and has thus been tremendous. However, during the last decade a gradual slowdown of this evolution has been observed, which pushes researchers to explore alternatives to technology downscaling in order to increase performance while maintaining a low cost [9].

One of the alternatives that are being explored, are three-dimensional stacked ICs (3D-SICs). This technique allows the integration of heterogeneous circuits (dies), each of which is manufactured in an optimal technology. So, products that combine the best characteristics of various technologies can be manufactured. Another advantage of stacking circuits is that the resulting topology keeps interconnect length and therefore also delay and power low, as wires connecting components do not have to run across a single, large chip [5]. Apart from interconnects, this improved topology can also benefit the product's form factor which is an important concern in mobile devices.

Meanwhile, chip testing is the means of keeping the promise of reliable and cheap electronic products. After manufacturing, all chips are tested at various stages of processing and only specification-compliant chips pass the tests. The ratio of passing chips over the number of manufactured chips is called yield and is typically such, that skipping testing is not a viable option. The cost of packaging a faulty chip, integrating it into a product, shipping, selling and then replacing it to the customer is prohibitive. The testing process incurs a cost of its own, but especially when testing is carefully planned, its cost is greatly outweighed by the savings it brings.

The motivation for this project lies in the increased importance of chip testing in the context of 3D-SICs. The reason is that multiple individual, and otherwise independent, circuits are bonded together to form a stack. Bonding a good die or even a good partial stack with one faulty die impairs all of these dies, thus multiplying the losses because of the faulty die. Therefore, multiple test application moments are often proposed as a preventive measure. Die or partial stack tests (pre-bond and mid-bond respectively) can be performed in addition to the final post-bond test of the complete stack to decrease the overall cost [20]. Of course, 3D-SIC testing comes with challenges of its own, as test accessibility of die components in the stack is reduced and the novel inter-die interconnects need to be tested as well. A generic design for test (DfT) architecture that enables modular testing at any stage in the stack manufacturing process is thus a requirement.

This project was based on an existing 3D-DfT architecture employing die-level wrappers, that was originally conceived and specified by IMEC [18, 19, 10]. In the context of the project, the original architecture was modified and extended to add compatibility with:

Test data compression (TDC): The support of TDC is crucial, as there are hardly any modern chips that do not include TDC; without it, their testing would be a far more time-consuming and expensive process. TDC makes a design appear in test mode smaller than it really is. However, equation-solving is required to determine the test patterns that correspond to the desired test stimuli and responses.

Embedded-core wrappers: Modular system-on-chip (SoC) designs including embedded cores are also very common, since the reuse of embedded cores in various designs saves significant development time. The extended 3D-DfT architecture is compatible with the standardized test interface of IEEE Std 1500 embedded-core wrappers.

Multi-tower 3D-SICs: As 3D-SICs become larger and more complex, having the stack branch-off into multiple towers seems like a natural evolution. This way, the functionality can be broken down with finer granularity into heterogeneous, more densely interconnected dies, which can increase the positive effects of the 3D integration technology on performance and power dissipation.

A *3D-DfT insertion flow* was designed and implemented for the application of the extended 3D-DfT architecture to any given design using Cadence' electronic design automation (EDA) tools. The verification of the inserted 3D-DfT per-die as well as across the stack was also automated using Cadence' EDA tools. In the *verification ATPG flow*, automatic test pattern generation (ATPG) and simulation of the generated test patterns is used for the verification of the correct operation of the DfT. This flow can be extended with migration of die-level test patterns to the stack boundary to become suitable for production-testing ATPG.

The remainder of the thesis is organized in five chapters. Chapter 2 provides the necessary background knowledge to follow the rest of the thesis. Chapter 3 gives an overview of the prior work on 3D-DfT. Chapter 4 presents the extended 3D-DfT architecture and the rationale behind the decisions taken. The automated implementation of the extended 3D-DfT architecture, and its verification through ATPG and simulation, are described in Chapter 5. Finally, Chapter 6 concludes this thesis.

2 Background Knowledge

3D-SIC testing is highly specialized as a field and as such it is based on significant pre-existing knowledge regarding both chip testing and the 3D integration technology. The necessary background knowledge is presented and explained in this chapter to arm the reader with the concepts that will allow him to understand the rest of the thesis.

2.1 Chip testing in general

Defects can and do occur in all semiconductor circuit manufacturing processes. Letting defective chips out of the production line has three main consequences. First, it leads to an increase in the cost of the final product, as one defective chip can ruin a whole product, the cost of which is often a lot higher compared to the chip itself. Second, it raises concerns regarding safety which is an important aspect in life-critical or mission-critical products. Third, especially if it continues for a long time, it can harm a manufacturer's reputation. The above make chip testing a necessary part of all production lines. The purpose of tests is to detect as many defects as possible, ideally all of them. However, if tests are not exhaustive, defective chips might pass them resulting in so-called *test escapes*. On the other hand, tests might incorrectly identify a non-defective chip as defective, which leads to so-called *false rejects*. The ratio of chips passing the tests over all manufactured chips is called *yield* and it is obviously negatively impacted by false rejects.

Chip testing incurs a cost of its own as it requires specialized infrastructure. This includes resources like for instance dedicated automatic test equipment (ATE), as well as people operating the equipment or designing the tests. The longer the testing procedure, the higher its cost, as it keeps the ATE busy for longer, preventing other chips from using it. This also means that achieving a higher throughput of chips leaving the production line requires purchasing additional ATE. However, the cost of chip testing is fully covered and exceeded by the savings it brings in terms of resources not wasted on a defective chip. Tests are carefully tuned in order to balance their cost according to the savings they bring while still attempting to meet any requirements set on test escapes.

Chip testing consists of two discrete phases, the test generation phase and the test application phase. The test generation phase will be elaborated in the section about *automatic test pattern generation* (ATPG). The test application phase essentially involves applying test stimuli to the *circuit under test* (CUT), capturing its responses and comparing them with expected responses. If they match, the circuit passes the test, otherwise it fails. Testing can be either functional or structural. *Functional testing* verifies the CUT in its functional mode. That means trying out possible combinations of its input values and monitoring its outputs. For realistic circuits these combinations amount to very large numbers, making tests prohibitively long, expensive and impractical if not

infeasible. *Structural testing* on the other hand abstracts from the functionality of the circuit and verifies the correct operation and interconnection of the various nodes of the circuit according to the logic models of its building blocks and its netlist. This can be achieved by testing far fewer combinations compared to functional tests.

2.1.1 Defects and faults

Various physical defects can occur in the manufacturing process of a chip. These include unintended short-circuits between wires, discontinuities in wires that result in opens, and transistors that malfunction. Defects can either cause a circuit to not function at all or to function outside of its specifications. In order to be able to reason about the correct or incorrect operation of a circuit in a scientific way and establish criteria according to which a circuit can pass or fail a test, *fault models* have been developed. These models abstract from the multitude of physical defects and represent their functional consequences as faults.

The most popular fault model due to its simplicity and effectiveness in detecting defects is that of *stuck-at* faults [7]. It considers every logic gate input or output to be potentially stuck either at logic high (stuck-at-1) or low (stuck-at-0), which can of course be the result of various manufacturing defects. In order to verify that, the stuck-at fault model attempts to drive every circuit interconnect to both possible logic values and checks whether the attempt succeeded or failed. Another somewhat similar in philosophy but less widespread fault model assumes that a transistor can be stuck-open or stuck-closed. Finally, in recent technologies, delay faults are becoming increasingly important, because it is often the case that circuits produce correct results but outside of their performance specifications [7]. *Delay fault* models consider delays to either occur at a single gate output or along a logic path. In order to detect a slow-to-rise delay fault a $0 \rightarrow 1$ transition needs to be generated, and similarly a $1 \rightarrow 0$ transition is used to detect a slow-to-fall delay fault. Generating a transition requires two appropriate test stimuli to be applied in succession. After loading the second test stimulus that is supposed to cause the transition, a clock pulse has to be applied at-speed to capture the response. A signal that takes longer than one cycle of the functional clock to propagate through a logic path constitutes a fault.

2.1.2 Automatic Test Pattern Generation (ATPG)

The test application phase previously described, is preceded by the test generation phase. *Automatic test pattern generation* (ATPG) is the automated process of generating the sets of test stimuli and expected responses needed to test a circuit. Various ATPG algorithms exist, but the principle behind all of them is common. First the faults that need to be tested have to be identified. Afterwards, a specific procedure that exposes each particular fault has to be followed. This procedure is illustrated here for the stuck-at fault model. Let's assume that the simple circuit depicted in Figure 2.1a needs to be tested for the stuck-at-0 fault shown in red in the upper left corner. In the figure the expected values of each node are shown next to it and the faulty values are shown next to them in parentheses. The steps that need to be followed are described subsequently.

Fault sensitization:
The opposite value from the value assumed in the fault has to be forced onto the circuit node under test. This is shown in Figure 2.1b.

Fault propagation:
The values of other nodes in the circuit have to be set to so-called non-controlling values, so that the value of the node under test is propagated to a primary output (PO) of the circuit where it can be observed. This is shown in Figure 2.1c.

Line Justification:
The values assigned to nodes in the steps of fault sensitization and fault propagation need to be mapped to values of the primary inputs (PIs) of the circuit under test so that they can stimulated externally. This is shown in Figure 2.1d.

The test stimuli that have to be applied on inputs A, B, C, D as well as the response expected on output E in the case of a fault-free circuit and in the case of a faulty circuit are shown in Table 2.1.

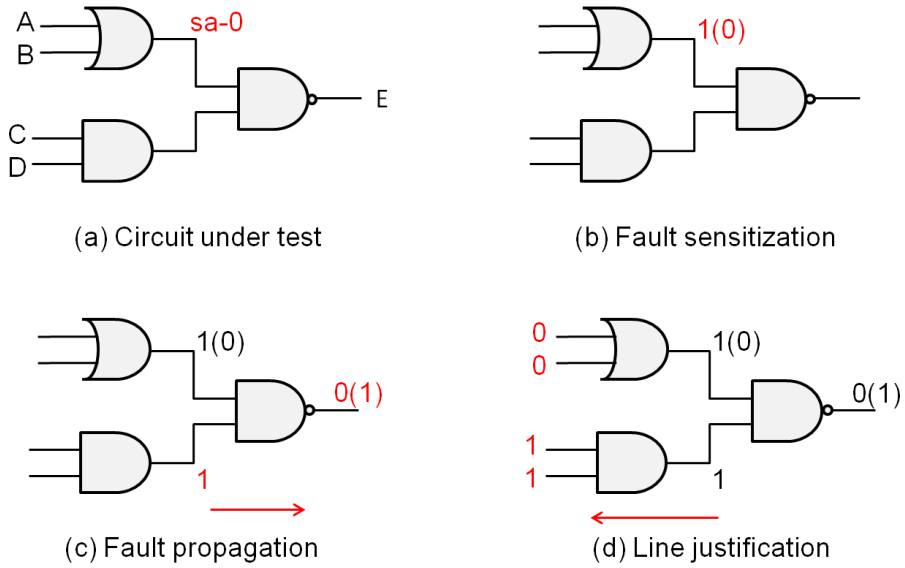


Figure 2.1: ATPG on a simple circuit detecting a single stuck-at fault.

Input				Good output	Faulty output
A	B	C	D	E	
0	0	1	1	0	1

Table 2.1: Test stimuli and responses for the circuit and fault of Figure 2.1.

The most well-known ATPG algorithms, the D-Algorithm, the Path-Oriented Decision Making (PODEM) and the Fan-Out Oriented (FAN) algorithm [7], are listed here as reference for the interested reader. The algorithms are implemented in automated software tools that primarily target high fault detection rate and low pattern count which

translates to low test time. A secondary target is low ATPG runtime as in realistically-sized circuits, runtime can reach several days or even weeks.

2.1.3 Fault coverage

The fault detection rate is officially called fault coverage and is defined as the ratio of faults a set of test patterns can detect over all potential faults in the circuit under test. It has to be noted though, that a fault coverage of 100% does not guarantee a 100% manufacturing-defect-free circuit, as a fault model does not capture all manufacturing defects. As mentioned before, high fault coverage is a primary objective of ATPG tools. There are two reasons that limit the fault coverage that is targeted in practice. First, while the first patterns might cover multiple faults, the last few faults are covered by a single dedicated pattern each, contributing disproportionally to the test time and cost. This saturating behavior is visible in Figure 2.2. If for example 100 patterns are required to achieve 100% fault coverage in a sample circuit under test, the first 30 patterns already provide a fault coverage of 95%. This is where test economics kick-in and limit the targeted fault coverage to a value that minimizes the summed costs of test and test escapes. Second, some faults might need multiple test patterns in order to be detected if the fault sites are buried in deep sequential logic, or they might even not be detectable at all. This is another reason why the fault coverage might drop and the reason why *design for test* (DfT), which is presented in the following section, was invented in the first place.

2.2 Design for Test (DfT)

The design practices, techniques, and hardware structures that target facilitating circuit testing are summarized under the umbrella-term *design for test* (DfT). Increasing a circuit's testability translates in turn into an increase in fault coverage and/or a reduction in test time. Initially DfT was only comprised of ad-hoc solutions that were applied to each individual case in a completely custom way in order to increase the testability of the problematic hardware structure. But as designs grew more complex, standard DfT practices were required to keep testability high while maintaining compatibility between different vendors at the same time should their designs be integrated into other, larger designs.

2.2.1 Scan design

Scan design is the most common DfT technique. It increases the testability of the circuit under test by essentially turning all flip-flop (FF) inputs into pseudo-primary-outputs and all flip-flop outputs into pseudo-primary-inputs. This is accomplished by connecting all flip-flops into one or more shift registers called scan chains, through which test stimuli can be scanned-in and test responses can be scanned-out. Of course the connection of flip-flops into scan chains should not affect their functional-mode operation, which requires their modification by adding extra hardware to convert them into scan flip-flops (SFFs). The procedure of applying scan design to a sample circuit is demonstrated in Figure 2.3.

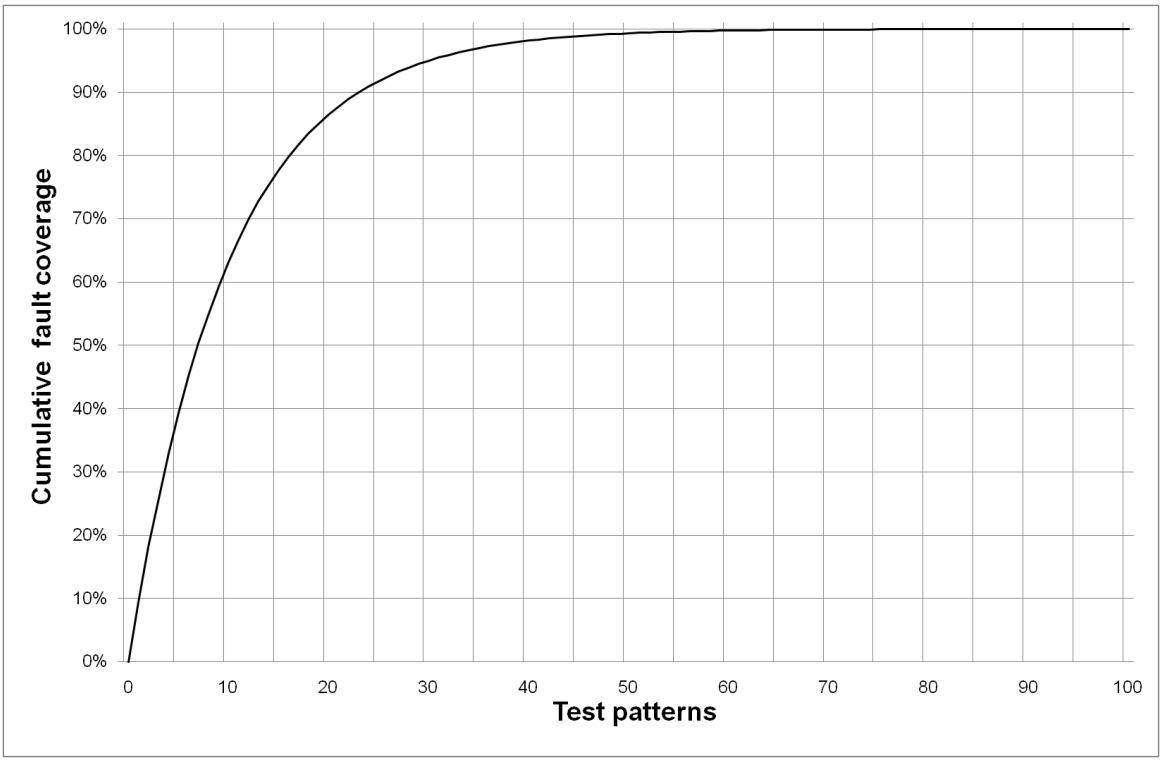


Figure 2.2: Cumulative fault coverage as a function of the number of test patterns.

The initial circuit is shown in Figure 2.3a, while the scan-enabled circuit is shown in Figure 2.3b. The extra components required for scan design are shown in red and they consist of the three following components each of which contributes to the circuit's total area.

Logic gates:

A multiplexer (MUX) is connected in front of every flip-flop input to convert it into SFF. Next to the functional data input of the flip-flop, this adds a test data input to the flip-flop so that one or the other can be selected by a so-called *scan enable* (SE) signal. So, while in the functional mode, the functional data input can be selected to connect the circuit in its initial topology, in the test mode the test data input is selected to connect the SFFs into scan chain(s).

Interconnects:

Extra wires need to be routed for realizing the scan chains as well as for the SE signal that needs to be connected to every single SFF. The SE is in that sense similar to the clock signal, only less sensitive in terms of delay and skew.

I/Os:

One extra I/O is required for the SE signal that controls whether the SFFs capture their functional or their test input. In addition, two extra I/O are needed per scan chain, one for scan-in and one for scan-out. There are cases though, where these

scan-ins and scan-outs can be shared with functional I/Os either natively, as it is possible in the example of Figure 2.3, or with the addition of multiplexers between functional and test output signals.

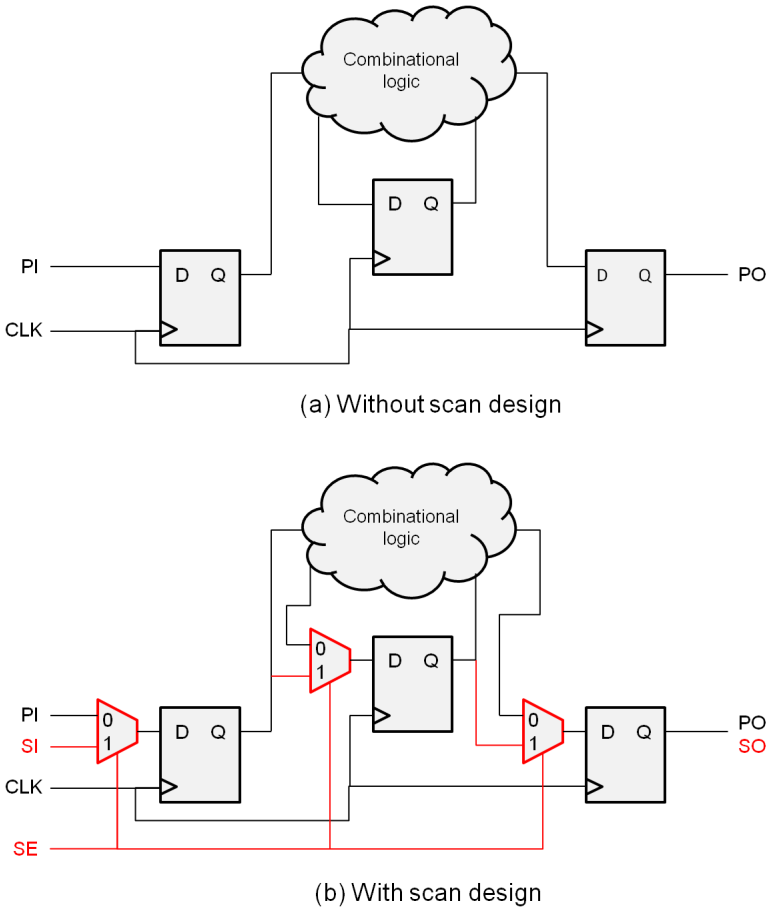


Figure 2.3: Application of scan design to a sample circuit.

The procedure of testing a scan-enabled design is comprised of three phases, the scan-in, the test application, and the scan-out. First, the test stimuli are scanned-in by applying clock pulses while the scan enable signal is asserted so that the SFFs capture their test inputs. Then, the scan enable signal is de-asserted and a clock pulse is applied so that the SFFs capture the test responses through their functional inputs. Finally, the scan enable signal is re-asserted and clock pulses are applied in order to scan-out the test responses for comparison. Figure 2.4 illustrates the signal waveforms needed to apply two test patterns to the scan-enabled circuit of Figure 2.3b. During the steady-state phase, the scan-in of one test pattern's stimuli can be overlapped with the scan-out of the previous test pattern's responses in order to save test time.

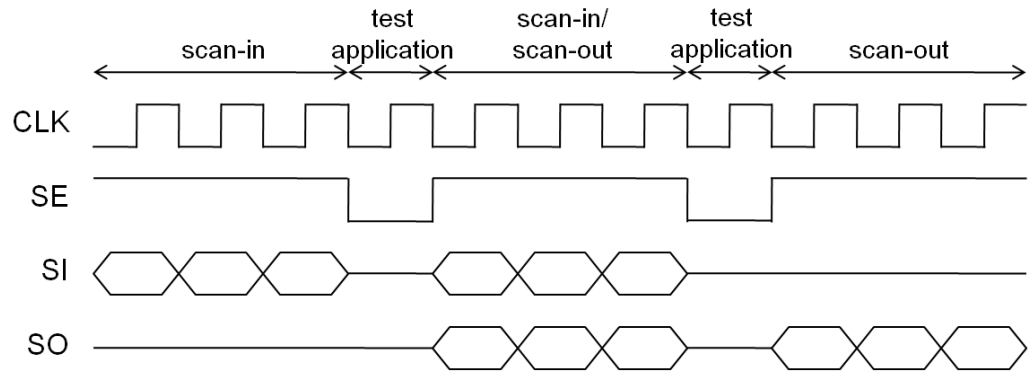


Figure 2.4: Application of two test patterns to the scan-enabled circuit of Figure 2.3b.

2.2.2 Test Data Compression (TDC)

As designs are getting increasingly more complex, the number of flip-flops they include also rises tremendously. In large designs this fact makes testing uneconomic even after the application of scan design, as the resulting scan chains are several tens of thousands flip-flops long. As a consequence, loading and unloading a scan chain takes an unacceptably long time. Another negative effect of the sheer increase in the volume of test data is the cost of the ATE, which needs exceptionally large and fast memories. One solution that tackles both problems is the compression of test data. Compressed test stimuli are generated and stored on the ATE. After they are loaded on the chip they are decompressed in order to load the scan chains. Subsequently the test is applied and the test responses are compressed as the scan chains are unloaded. Of course, the use of compressed test data comes with a tradeoff. While test time and test data volume are reduced, the hardware necessary to decompress test stimuli and compress test responses takes up precious chip area which is only useful at the moment the chip is being tested. Moreover, the compression and decompression are usually not lossless, which can potentially have a negative impact on the fault coverage, as it might not be possible to sensitize certain faults, or certain other faults might be masked during the process of test response compression. The above consequences become more prominent at high compression ratios.

The most basic form of *test data compression* (TDC) exploits the properties of exclusive-OR (XOR) gates. The organization of such a TDC module is shown in Figure 2.5. Externally, two scan chains are visible, while in reality there are four scan channels internally, which suggests a 2x compression ratio. As a result, the same number of flip-flops can be loaded and unloaded through a narrower interface and in a shorter time, as it is distributed over a greater number of scan chains. The decompressor can be as simple as fanout, which is also known as *Illinois Scan* [14]. Unfortunately, this creates undesired correlations between the test stimuli feeding different scan chains. These correlations can be mitigated if a more complex XOR network is used instead. The compressor is implemented as an XOR network in order to combine the outputs of two scan channels in a single scan chain. Figure 2.6 depicts examples of implementations for the decompressor (a, b) and the compressor (c) of Figure 2.5.

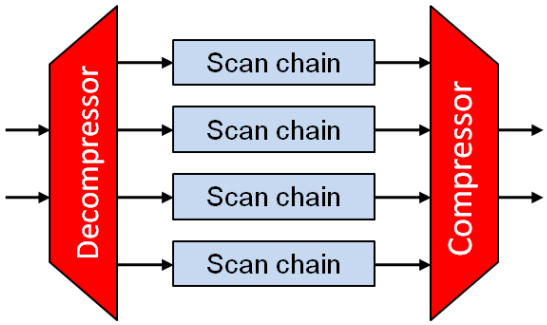


Figure 2.5: XOR-based test data compression organization.

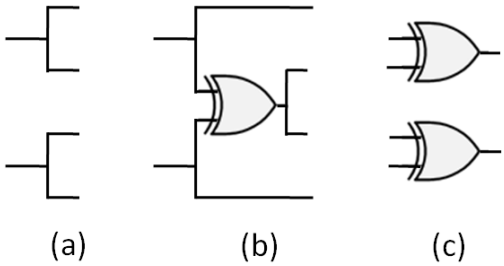


Figure 2.6: Implementation examples of the decompressor (a, b) and compressor (c) of Figure 2.5.

Apart from the XOR-based TDC, more sophisticated methods exist like the one employing a so-called *multiple input shift register* (MISR) for the compaction of the test responses. The principle behind the operation of the MISR is the creation of a test signature out of superimposed test responses. This allows the conversion of all scan outputs to scan inputs, an action that essentially doubles the test data bandwidth and in a sense also the achieved compression ratio. The above is made possible because the test response volume is reduced to mere signatures that need to be occasionally retrieved from the MISR for comparison to the expected signatures. A single scan output or bidirectional scan inputs that can also temporarily function as scan outputs serve that purpose adequately.

A complication introduced by the MISR is that its state has to be always well-defined, otherwise the signatures get corrupted. In order to accomplish that, the MISR has to be initialized with a known seed value and subsequently always fed with meaningful data, which means that any potentially existing *don't care* (X) bits have to be masked. Masking, which can also be optionally applied to XOR compression, requires the addition of masking logic, as simple as an AND or OR gate, at the output of each scan channel. A signal can then be used to determine when the masks are enabled and when they are transparent. This simple scheme has the obvious drawback that whenever a single channel needs to be masked, all other channels that carry potentially interesting data are masked as well, which might lead to fault coverage problems. Adding mask registers that are pre-loaded with data determining which masks will be applied at the assertion of the mask enable signal solves the problem at the expense of some extra area. An example

implementation of a MISR together with the channel masks is shown in Figure 2.7. As explained before, the signature contained in the MISR flip-flops after the completion of a series of tests can be retrieved either serially through a scan output or in a parallel fashion by temporarily using the scan inputs for scanning-out.

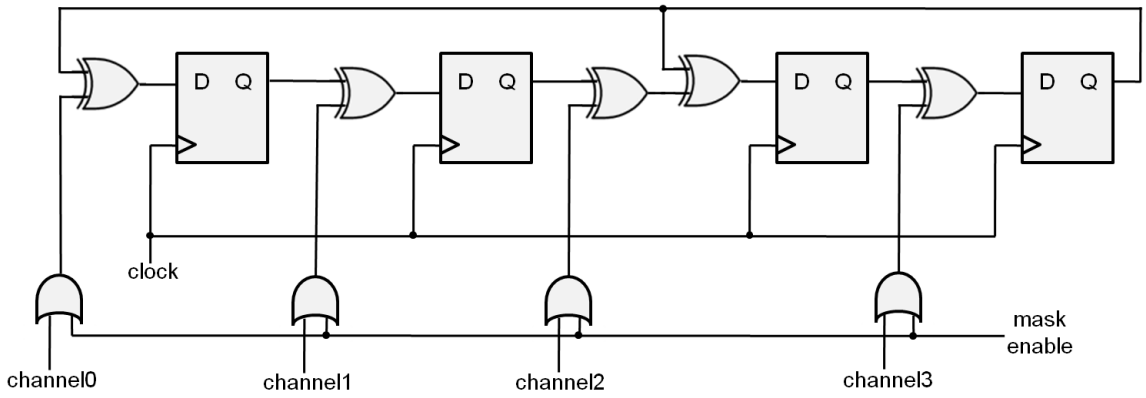


Figure 2.7: Example MISR implementation with channel masks.

2.2.3 2D-SoC testing: IEEE Std 1500

As designs are getting increasingly more complex, the concept of embedded cores has been widely adopted by the industry so that components can be reused in different designs in a modular way in order to save on development time and cost. In this fashion, even full-fledged electronic systems can be integrated on a single chip composing a so-called *system on chip* (SoC). There are a number of good reasons for applying this concept of modularity not only to the functional aspect of embedded cores, but also to their testing. First, testing smaller components in a modular way helps dealing with the size and complexity of modern designs. That holds both for the test generation phase, where the runtime can be constrained to reasonable figures, as well as for the test application phase, where only certain components can be individually targeted for testing. Second, embedded cores sometimes constitute protected *intellectual property* (IP), the implementation details of which are withheld. In such cases, the test patterns for the core are provided as-is and have to be applied to the boundary of the core, even though it is integrated inside a larger chip.

In order to overcome these challenges, IEEE Std 1500 establishes the insertion of a wrapper with a standardized test interface around every core [2]. Figure 2.8 presents an abstract view of the wrapper where its main features are visible. In terms of hardware components, the *wrapper boundary register* (WBR) is comprised of wrapper cells intercepting each functional I/O and thus isolating the core from its surrounding environment and providing controllability of its inputs and observability of its outputs. The I/O interface of such a wrapper cell is visible in Figure 2.9a, while a potential implementation is shown in Figure 2.9b. The wrapper cells are connected in one or more scan chains through their test inputs (CTI) and outputs (CTO) in order to form the WBR. Apart from the wrapper cells, the wrapper includes a shift and an update register

that together form the *wrapper instruction register* (WIR) that controls the state of the wrapper. Moreover, a *wrapper bypass* (WBY) register provides a minimum scan length path through the core, while maintaining a clean timing interface. A set of multiplexers reconfigure the scan chains as needed for the various test modes. In terms of I/Os, the wrapper mandates a serial scan-in (WSI) and serial scan-out (WSO) that are used both for test data and test instructions that configure the wrapper itself. In addition, an optional and scalable parallel scan-in (WPI) and parallel scan-out (WPO) bus can be implemented to increase the test bandwidth. Finally, a wrapper control signal bus (WSC) is also mandatory and includes the following signals:

- WRCK is the wrapper clock that is used to trigger shifting and updating the various flip-flops while in test mode.
- WRSTN is the wrapper reset signal which resets the wrapper in a default/safe mode.
- SelectWIR is the signal that controls whether the serial interface currently carries test data or test instructions.
- ShiftWR is the scan enable signal. When asserted, a shift operation is performed at the rising edge of WRCK.
- UpdateWR is the signal that defines that the WIR state has to be updated upon the rising edge of WRCK according to the instruction that has been scanned-in.
- CaptureWR is a signal that defines that a capture operation has to be performed by the WIR upon the rising edge of WRCK.

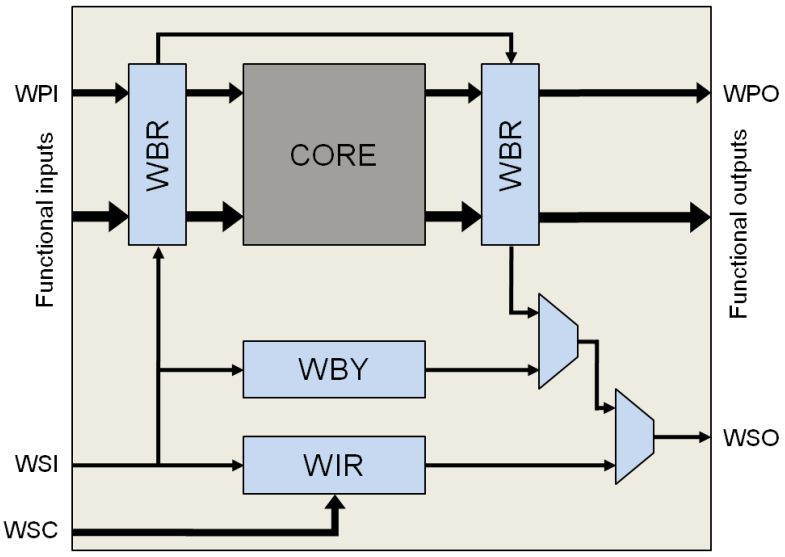


Figure 2.8: IEEE Std 1500 wrapper overview.

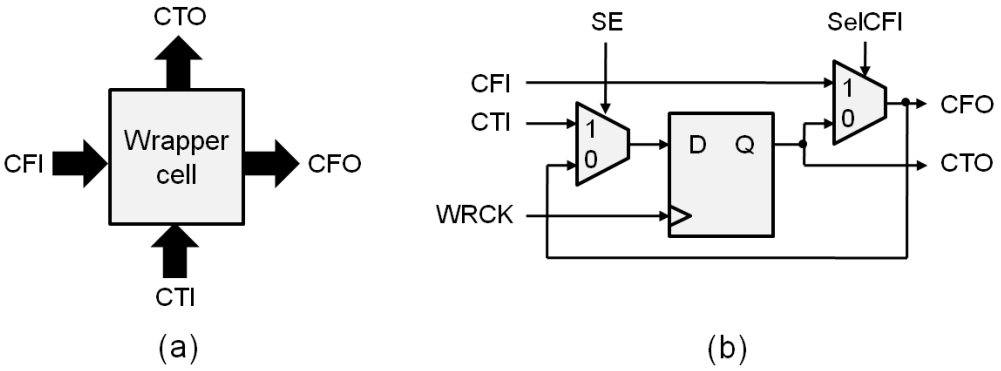


Figure 2.9: IEEE Std 1500 wrapper cell interface and implementation.

The wrapper enables two types of tests to be performed. The *internal test* (INTEST) tests the internals of the wrapped core by concatenating the WBR and the core-internal scan chains into one or more scan chains depending on whether the serial or the parallel test interface is used. In this mode, the input wrapper cells supply test stimuli to the primary inputs of the core, while the output wrapper cells capture test responses out of its primary outputs. The *external test* (EXTEST) tests the interconnects between the core and its surrounding circuitry. In this mode, only the WBR is included in the test path, with the input wrapper cells capturing test responses from the surrounding environment and the output wrapper cells supplying test stimuli to it. Finally, a bypass mode (BYPASS) allows the embedded core to be bypassed while other cores are tested, so that its contribution to the test path length is minimized.

2.2.4 PCB testing: IEEE Std 1149.1

IEEE Std 1149.1, often also referred to as JTAG, is a standardized wrapper for testing and debugging chips on a *printed circuit board* (PCB) [1]. Its purpose is similar to that of IEEE Std 1500 for embedded cores, but there is one key difference. Chip pins are a lot more expensive in terms of area compared to core I/Os, so minimizing their number is an important objective of this standard. In order to accomplish that, the test interface consists of a *test access port* (TAP) that is only 4 or 5 bits wide and includes a serial test interface and some signals that control a finite state machine (TAP controller), used to set the test state of the wrapper. The signals of the TAP are the following:

- TCK is the test clock that triggers the wrapper's flip-flops.
- TRST is an optional reset signal that resets the TAP controller to its default state (functional mode).
- TMS, Test Mode Select in full, is the signal used to traverse the various states of the TAP controller. Depending on its logic value upon the rising edge of TCK, a transition to one state of the finite state machine or another is performed.
- TDI is the scan-in port for test data and test instructions.

- TDO is the scan-out port for test data and test instructions.

An overview of the Boundary Scan Architecture is given in Figure 2.10, where the TAP signals, TAP controller, *boundary register*, *instruction register* (IR), *bypass register*, as well as the various test path reconfiguration multiplexers are shown. The IR test instructions are divided in public and private. The public instructions BYPASS, SAMPLE, PRELOAD and EXTEST are mandatory, while the INTEST, RUNBIST and HIGHZ instructions are optional. Private instructions can be defined by the user to control custom chip DfT structures.

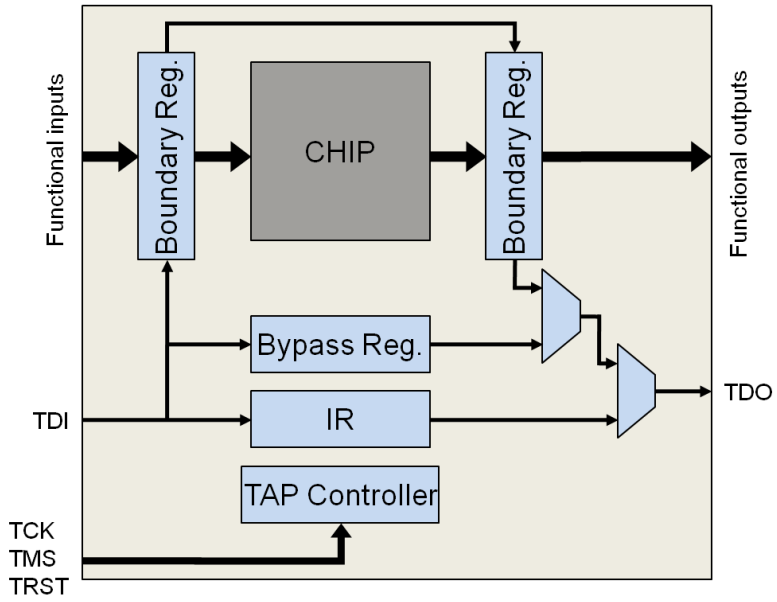


Figure 2.10: IEEE Std 1149.1 wrapper overview.

The finite state machine of the TAP controller is illustrated in Figure 2.11. The state diagram can be conceptually divided in two halves. The transitions and states in the left half of the diagram are used to control the operation of test *data registers* (DR), while those in the right half control the operation of the test *instruction register* (IR).

Finally, IEEE Std 1149.1 also requires wrapper cells that intercept the chip I/Os and allow their full controllability and observability by loading and unloading the Boundary register. These are slightly more complex than those of IEEE Std 1500 because of the extra functionality they incorporate. Namely, they include a second flip-flop, which is clocked independently from the first flip-flop, and allows the simultaneous updating of the output values of all wrapper cells. This feature is particularly useful in the case of delay fault testing. Figure 2.12 depicts such a boundary register cell.

2.3 3D-SIC technology with TSVs

In traditional 2D-IC technologies, there is no I/O interconnect access to the chip from its back-side, as I/O interconnects can only be placed on its front-side, right above

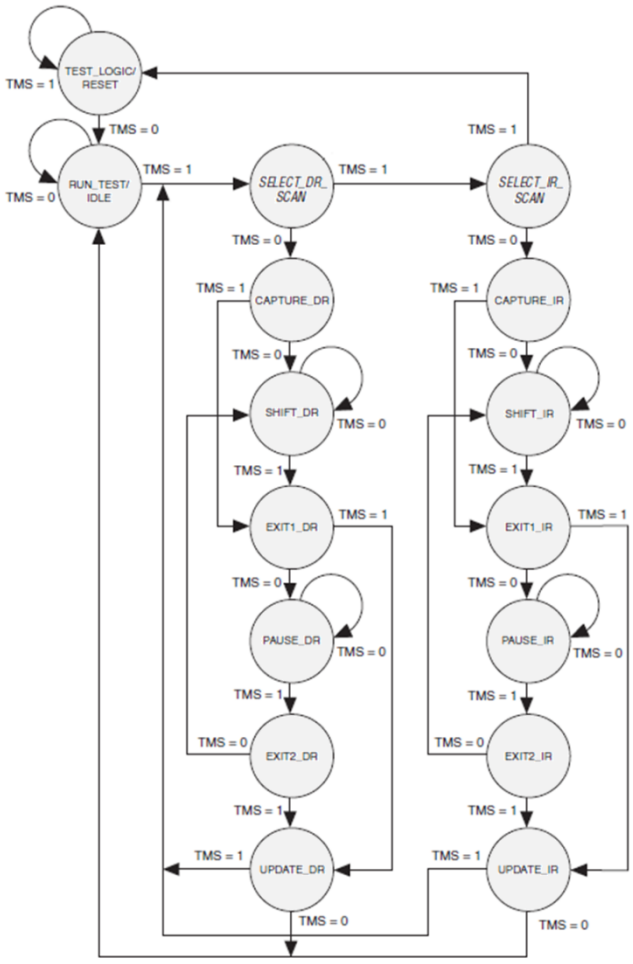


Figure 2.11: IEEE Std 1149.1 TAP controller FSM.

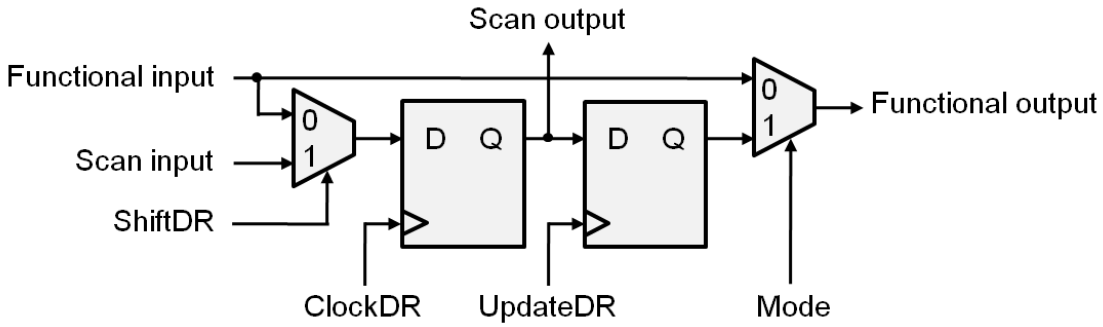


Figure 2.12: IEEE Std 1149.1 boundary register cell.

the top metal layer. However, vertical IC stacking, with the exception of face-to-face stacking of two dies, requires the presence of I/Os on both the front and the back-side

of a die. This requirement introduces various technological challenges. First of all, it is virtually impossible to create interconnects that penetrate the full depth of the silicon substrate of a full-thickness wafer. Therefore, wafers need to be thinned down, a procedure that makes them fragile and also affects the electrical properties of transistors if performed to an excessive degree. Once the problem of wafer thickness is overcome, copper or tungsten nails called through silicon vias (TSVs) can be used to connect the base transistor layer to the back-side of the wafer. Finally, so-called micro-bumps are placed on top of the exposed TSV tips to facilitate the bonding of neighboring dies with each other [3]. Figure 2.13 illustrates the necessary processing steps in order to manufacture the above microelectronics structures and create die stacks as they are specified by and performed at IMEC.

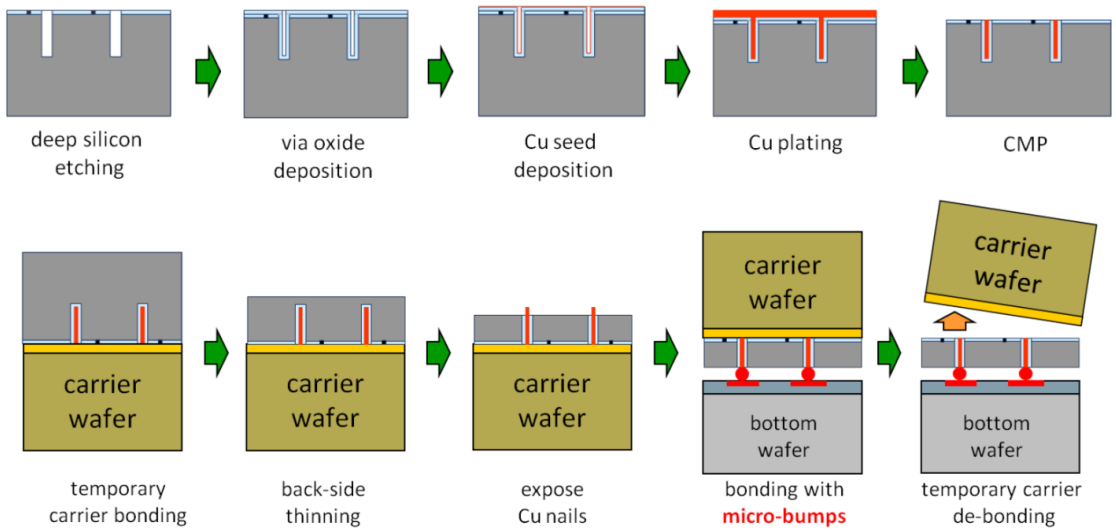


Figure 2.13: IMEC processing steps for manufacturing of TSVs and die bonding.

TSV features include high interconnect density and performance and low power dissipation thanks to their small dimensions and thus capacitance. These characteristics make inter-die connections within a stack more similar to on-chip connections rather than off-chip connections. The dimensions of TSVs in terms of pitch, diameter, and height are in the range of a few tens micrometers and are continuously being scaled down even further as soon as problems associated with the filling of the high-ratio TSV drill-holes with metal are resolved.

Stacked ICs with TSV interconnects enable novel applications. Dies manufactured in heterogeneous technologies can be integrated together in a stack, combining the advantages of each technology so that each component, may it be analog circuitry, digital logic or memory, is implemented in an optimal way. Moreover, the stacking possibilities are virtually endless. Examples of die stacks are shown in Figure 2.14 in order of increasing complexity from left to right. The leftmost 2.5D stack (a) contains three dies placed side-by-side on top of and connected through a so-called silicon interposer. The middle 3D stack (b) contains three vertically-stacked dies, while the rightmost 5.5D stack (c) combines both technologies and contains five dies organized in two towers.

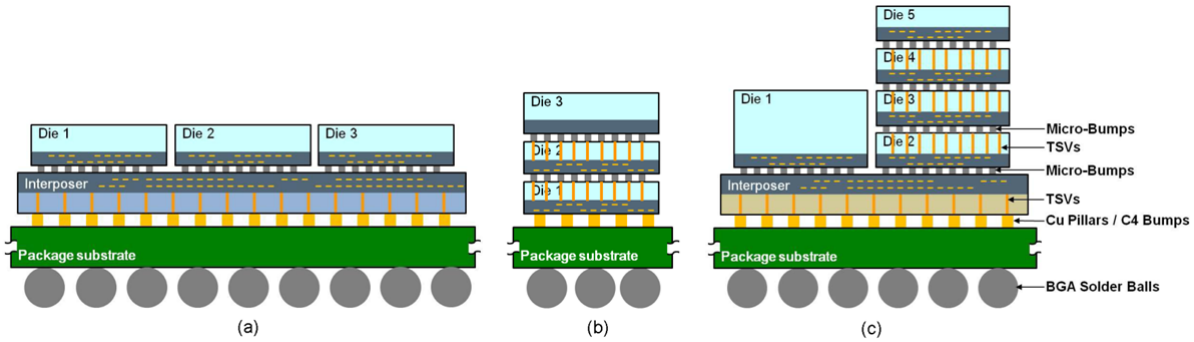


Figure 2.14: Examples of 2.5D (a), 3D (b) and 5.5D (c) stacked ICs.

3 Prior Work on 3D-DfT

Three-dimensional die stacking introduces various new test requirements and challenges which can be divided in three large categories: test flows, test content, and test access [20].

Regarding *test flows*, the natural test moments in the process of 3D stacking are pre-bond testing of a standalone die, mid-bond testing of an incomplete stack, and post-bond testing of a complete stack. Pre-bond testing is necessary because randomly stacking good and faulty dies together results in an exponential decline in the compound yield. Furthermore, a single large monolithic die can be partitioned into multiple smaller dies. As a result, the compound yield can increase, as each one of these smaller dies has reduced probabilities of containing a fault and can be tested individually to prevent stacking faulty dies.

The *contents of a test* targeting a single die match the stimuli and responses that would be generated for the equivalent 2D-chip. However, TSVs and micro-bumps are novel microelectronic structures that also need to be tested for manufacturing defects. Research regarding the potential need for specialized fault models has been and is still being conducted. The results are beyond the scope of this text, but [13] provides a comprehensive summary of TSV/micro-bump faults and associated tests for the interested reader. According to it, TSV and micro-bump faults might occur due to voids, opens, shorts, or incomplete metal-fills, misalignment, impurities, or leakage to the substrate. Interconnect continuity, resistance, capacitance, leakage, and AC (delay) tests are mentioned as test methods to detect them. Another aspect of test contents is their automated generation. Modern interconnect ATPG tools are already capable of generating efficient open/short test pattern sets for 3D-SIC interconnects in a short runtime.

Test access entails two challenges. First, pre-bond tests require electrical contact to be established between the ATE and the die-under-test. However, any die destined to be integrated into a 3D-SIC as a non-bottom die, only uses TSVs and micro-bumps for its I/Os. This is where the advantageously small dimensions of TSVs become a problem, as modern probe equipment cannot probe on them. Attempts are being made lately in order to overcome this problem through a leap in the probe technology [22], but a fully-working solution is not yet available. As a result, dedicated probe pads, compatible to the existing probe equipment, have to be included in every die to enable its pre-bond testing. However, pads consume significant silicon area, so their number has to be restricted, which in turn creates a tradeoff between the area overhead added by probe pads and the pre-bond test time.

The second *test access* challenge concerns access of a die up in a stack during mid-bond and post-bond testing. Of course, physical access is only possible through the bottom die as its I/Os are exposed to the outside world. Therefore, DfT structures are needed to enable the dies forming a stack to collaborate so that test stimuli can be

passed-on through them and provided to any component of any die in the stack, and its test responses can be captured and scanned-out in the same way for comparison. Furthermore, inter-die extest targeting the TSVs and the micro-bumps that interconnect neighboring dies requires even closer collaboration between them, as one die needs to supply a stimulus to an interconnect, while simultaneously the other captures the corresponding response. The definition of a standardized DfT architecture for the DfT that needs to be added to 3D-SIC dies is an important step as it ensures test compatibility, especially when the dies are provided by different manufacturers. Finally, another important aspect directly related to test access, is the migration of a die's tests to the corresponding boundary for pre-bond, mid-bond, and post-bond testing respectively.

The above are the main challenges of 3D testing as perceived by the author. Some of these needs and problems were identified and/or resolved in prior work on 3D testing which will be presented in the rest of this chapter. Meanwhile, some others were identified and/or resolved in the context of this thesis and will be discussed in Chapter 4.

3.1 Overview of prior work

The first paragraphs of this section present prior work on 3D-DfT performed in academia. The presented researchers and their work are not the only ones having attempted to tackle issues related to 3D-DfT, but they are the ones that have contributed the most to the field.

The first attempt to resolve problems related to 3D testing came from Georgia Institute of Technology and is described in [16]. In their paper, Lewis and Lee suggested using "scan-islands", essentially wrappers, as a means to isolate dies from each other and obtain controllability and observability of their I/Os. Scan design was used to increase testability and a so called *layer test controller* (LTC) in each die controlled the organization of flip-flops into one or more scan chains. Moreover, IEEE-1149.1 was included in one of the dies to facilitate testing of the complete chip once mounted on a PCB. The authors also proposed the use of dedicated probe pads for pre-bond testing. However, their DfT architecture was abstract and only addressed pre-bond testing, not taking mid- and post-bond testing into account. Finally, being a custom solution, it was not compatible with already standardized DfT solutions.

Subsequently, researchers from Pennsylvania State University and Duke University attempted to provide an answer to the problems of post-bond stack testing and *test-access mechanism* (TAM) optimization through their work presented in [24]. Wu et al. adopted Lewis's and Lee's solution of a die-level wrapper as a method to modularize the testing procedure. Instead of on pre-bond testing, their focus was on minimizing the number and length of scan chains running through several dies by means of *integer linear programming* (ILP). However, the fact that they only addressed post-bond testing, ignoring the challenges associated with pre- and mid-bond testing, limited the applicability of their solution to real designs. Finally, ad-hoc optimization of the number of inter-die test I/Os might be beneficial in terms of area, but it certainly complicates the process of creating several compatible dies that can be combined together to form a consistent stack-wide TAM, especially if they are provided by different manufacturers.

The most complete and overall academic solution on 3D test was provided by the

Chinese University of Hong Kong in collaboration with Duke University. The work presented by Jiang et al. in [15] was also based on die-level wrappers. The proposed wrapper architecture leverages the pre-bond solution given by Lewis and Lee by taking the increased area cost of pre-bond probe pads into account. The authors realized the benefits of allowing different pre-bond and post-bond TAM widths, each optimized according to the available area and test needs. Their test architecture is therefore able to handle different TAM widths, while allowing the sharing of routing resources among them. This work was the first to take both pre-bond and post-bond testing into account, and also optimize pre-bond test resources. However, it neither tackled mid-bond testing, nor did it focus on the actual architectural decisions in terms of organization and features.

While the academic work presented above was the first to point out and/or resolve several 3D-DfT related needs and problems, it failed to provide an overall solution. None of the above considers test control and instructions, or compatibility with existing wrappers. Furthermore, none of them is an overall, standardizable solution that can be applied to many different designs in a straightforward way. Last but not least, none touches upon the very important problem of inter-die interconnect Extest in order to test the TSVs and micro-bumps.

Industry also has a strong interest in 3D-SICs and the advantages they offer. Therefore companies conduct their own research on the topic of 3D integration which includes 3D-SIC testing and DfT. Through their work they attempt to solve the problems left unanswered by the academic research, so that they eventually end up with a fully functional product for their customers. The main contributors to this effort are major *electronic design automation* (EDA) companies, often in collaboration with microelectronics research institutes. These can currently be identified and listed as Cadence in collaboration with IMEC, Synopsys in collaboration with ITRI, and Mentor Graphics.

Cadence and IMEC work on a 3D-DfT architecture originally conceived and specified by IMEC [18, 19, 10]. The IMEC 3D-DfT architecture was the first to provide an overall solution to the needs and problems of 3D testing and has therefore been the foundation of the work conducted in the context of this thesis. Its detailed presentation will follow in the coming sections. As for the other companies and organizations, Mentor Graphics also base their work on the IMEC architecture [12], but are working on it independently. However, their efforts appear to be mainly on its automation and incorporation in their suite of test-related applications, rather than its extension and/or improvement on an architectural level. Finally, Synopsys and ITRI are working on a somewhat similar architecture described in [8] and which is based on core-level IEEE-1149.1-inspired wrappers. This architecture has several shortcomings. First, it is missing a parallel TAM for high-bandwidth production testing. Second, the fact that it is core-based complicates handling simple dies without embedded cores. The third and final issue is that of scalability. The TAMs of the individual cores are multiplexed on the wrapper TAM I/Os, rather than e.g. being concatenated in a daisy-chain architecture with bypass paths. Such centralized schemes often scale poorly, unlike distributed architectures.

3.2 Overview of the IMEC 3D-DfT architecture

The IMEC 3D-DfT architecture employs die-level wrappers based on IEEE-1500 and is a daisy-chain architecture. A conceptual schematic providing an overview of the architecture is shown in Figure 3.1. The wrapped die is depicted as a grey box and its flip-flops are organized into one or more scan chains. Wrapper cells intercept the die's I/Os and form the WBR that provides controllability and observability to the I/Os while in test mode. The test interface comprises a serial test interface (WSI/WSO) carrying both test instructions and test data, which is mandatory, and a scalable parallel test interface (WPI/WPO), which is optional. The test interface is completed by a test control bus, the WSC, which contains the same signals as the IEEE-1500 WSC. The primary test interface on the bottom side of the die is replicated on the top side of a non-top die forming the so called secondary test interface (WSIs/WSOs, WPIs/WPOs, WSCs). This allows transferring test data and instructions from and to a die up in a stack. Furthermore, a scalable number of dedicated probe pads can optionally be included on the bottom side of a die to enable its pre-bond testing. The TAM width of these pads can be different from the TAM width of the test TSVs used for post-bond testing. This gives the designer the necessary flexibility to optimize the test time as well as the area resources he is willing to spend on each test. Switch boxes containing a number of multiplexers are included for the configuration of the die-internal and boundary scan chains in the three different TAMs: serial, parallel pre-bond, and parallel post-bond. A WIR, controlled by the WSC and the scanned-in test instructions, is responsible for controlling the wrapper state and thus the test mode the wrapped die is currently in. The wrapper has a *turn* mode that sends test data downwards in the stack, and an *elevator* mode that sends data to and receives data from the die above. This turn/elevator mechanism enables the transfer of test data up and down the stack. Furthermore, each die implements an intest mode for testing the die-internal circuitry, an extest mode for testing the TSVs interconnecting neighboring dies in the stack, and finally a bypass mode that allows bypassing the die to reach other dies. Taking into account that stacks might get quite high in the future, the architecture includes a register on both the up-going (bypass) test path and the down-going path in order to provide a clean timing interface with the die(s) located directly above and below it.

The IMEC architecture enables modular pre-bond testing of individual dies, mid-bond testing of incomplete stacks, as well as post-bond testing of a complete stack. Following any path of the railroad diagram of Figure 3.2 from left to right generates a valid test mode supported by the architecture. Another advantage of the IMEC architecture stems from its origin. Being based on IEEE-1500, compatibility with embedded cores is almost native, while PCB interconnect testing is also made possible via the implementation of IEEE-1149.1 by the bottom die. Meanwhile, the fact that the architecture is a daisy-chain architecture allows any individual component of any die to be tested at full test bandwidth to achieve the minimum test time [23]. The JTAG of the bottom die as well as the full-width TAM running throughout the stack are illustrated in Figure 3.3.

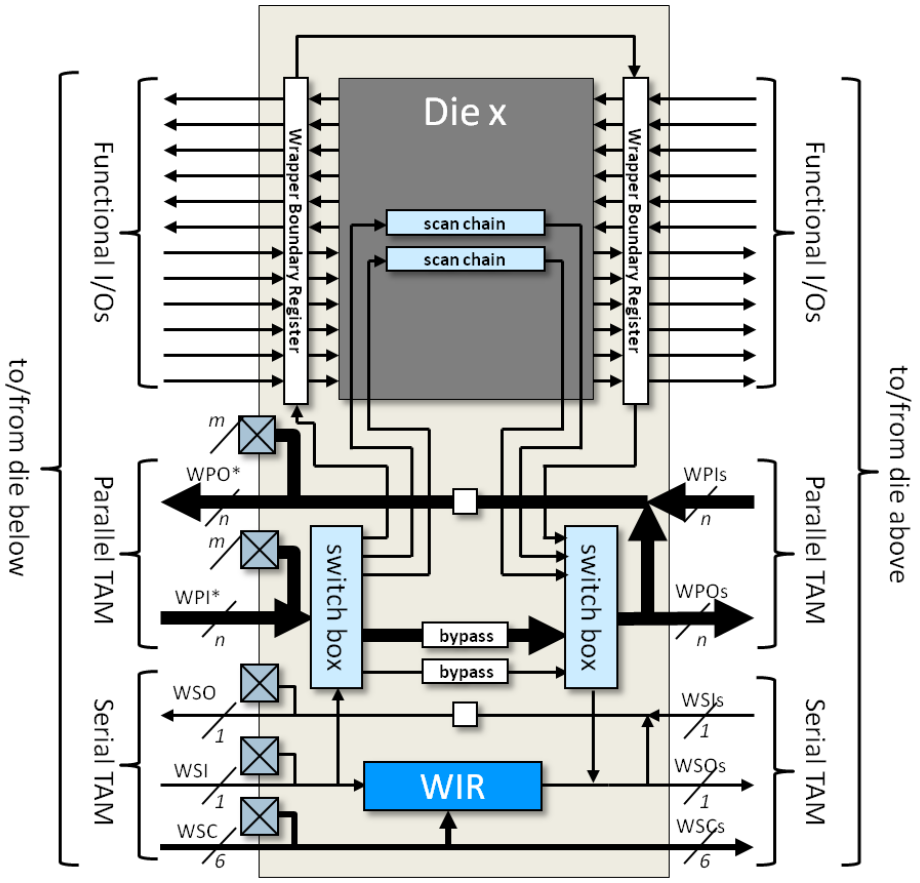


Figure 3.1: IMEC 3D-DfT architecture overview.

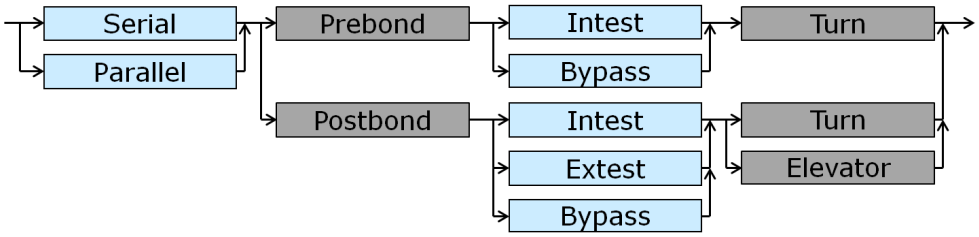


Figure 3.2: IMEC 3D-DfT architecture test modes railroad diagram.

3.2.1 Test path reconfiguration and width adaptation

A very important component of the 3D-DfT architecture is the test path reconfiguration mechanism that is essentially implementing the various test modes supported by the wrapper. The implementing components are a number of multiplexers which can be seen in Figure 3.4 and are explained below. The multiplexers located on the left side of the figure and labeled as prebond are responsible for selecting between probe pads and TSVs as the primary test interface I/Os. Another set of multiplexers, labeled as extest,

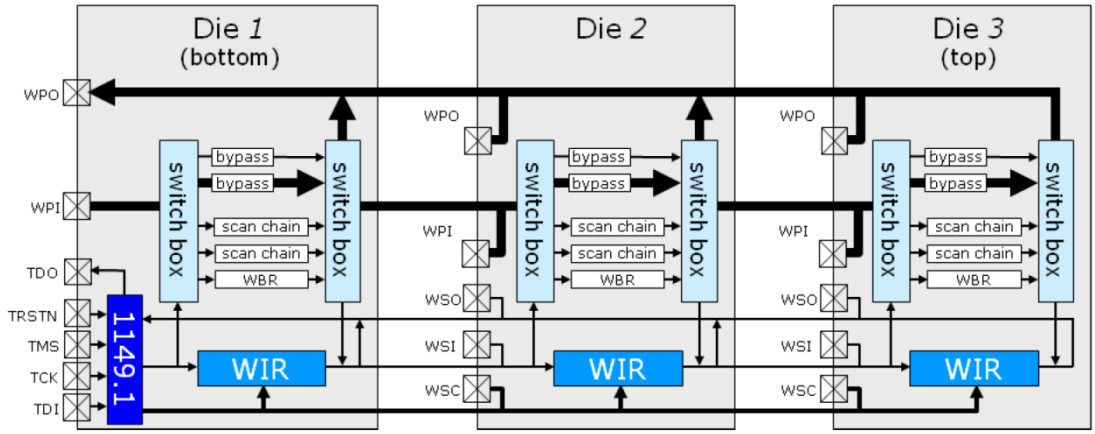


Figure 3.3: A three-die stack implementing the IMEC architecture.

select between the WBR alone for extest and the die-internal scan chains concatenated with the WBR for intest. There are three of those multiplexers, as there are that many separate TAMs. A serial bypass multiplexer and a parallel bypass multiplexer select between the die's scan chains and the bypass register that provides a minimum length scan path to the die on top. The serial TAM contains an extra multiplexer after the serial bypass multiplexer in order to select between the test data path and the test instruction path going through the WIR. Finally, a serial turn multiplexer and a parallel turn multiplexer enable the inclusion or exclusion of the die(s) on top from the test path. All of the above multiplexers are controlled by WIR output signals, which are asserted or de-asserted according to the test instruction that has been scanned in. An exhaustive enumeration of the supported test modes exactly matches those generated by the railroad diagram of Figure 3.2, which proves that the architecture is complete and covers all desired test modes.

An important feature of the 3D-DfT architecture is the option to have different parallel TAM widths for pre-bond testing through pads and for post-bond testing through TSVs. Of course the die-internal scan chains also need to be able to adapt to all different supported TAM widths, namely serial, parallel pre-bond, and parallel post-bond. Concatenation of scan chains is used as a means of width adaptation. An example with a parallel pre-bond TAM width of 2 and a parallel post-bond TAM width of 4 is shown in Figure 3.5. The required multiplexers are considered to be part of the switch box seen in Figure 3.1. If the number of die-internal scan chains happened to be more than 4, some scan chains would first be permanently concatenated with each other to scale the die-internal TAM width down to 4. A negative consequence of scan chain concatenation is that it might create imbalance in scan chains that were initially balanced.

3.2.2 Extensions for JEDEC Wide I/O DRAM

The original IMEC 3D-DfT architecture has also been extended in order to support 3D-SICs containing Wide I/O DRAM [11]. This type of DRAM [4] has a particularly wide

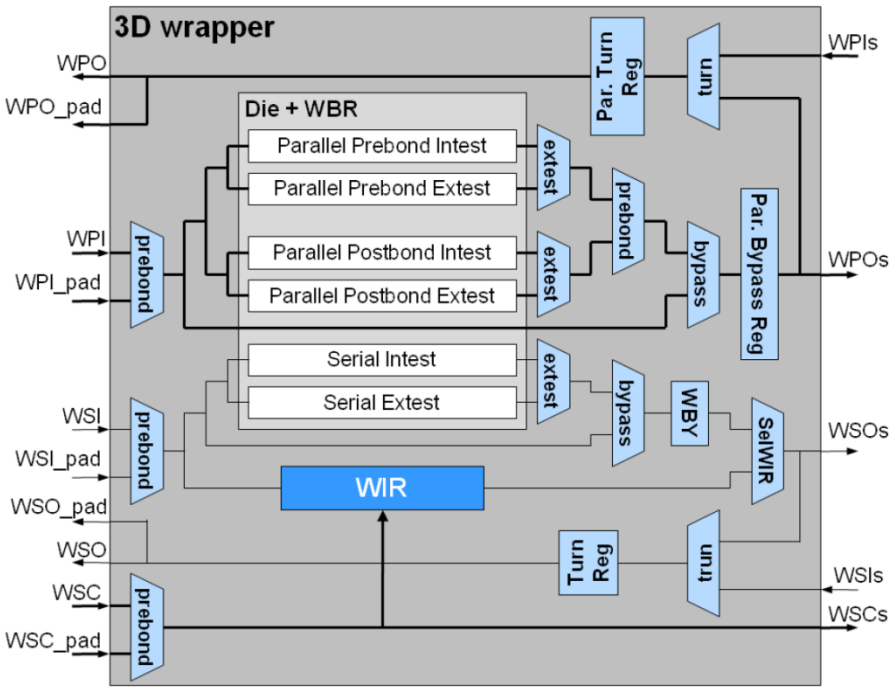


Figure 3.4: IMEC 3D-DfT architecture test path reconfiguration components.

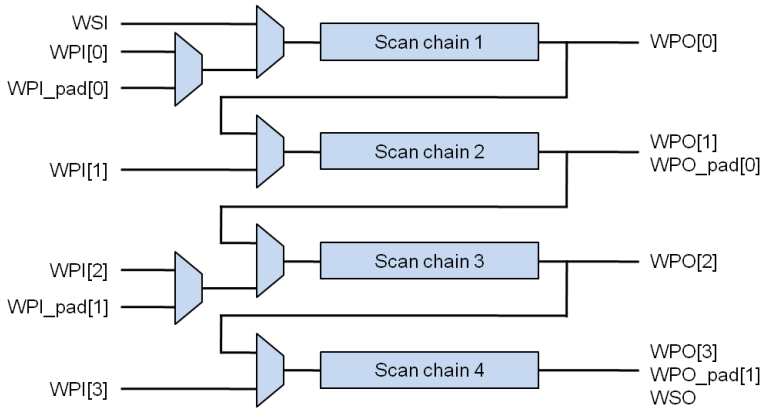


Figure 3.5: TAM width adaptation through scan chain concatenation.

data interface of 128 bits per channel, allowing for higher bandwidth with lower power dissipation. Each memory rank has four independent channels (a-d) and each memory can have up to four ranks (0-3) in total. A die containing the required DfT is illustrated in Figure 3.6 together with a Wide I/O DRAM with four ranks. The responsibility for the Intest of the memory remains to the die designer who has to include a memory built-in self test (MBIST) engine in his design. On the other hand, the Wide I/O DRAM standard includes a boundary scan implementation to enable testing its interconnects.

So, the boundary register of the DRAM participates in the Extest of the die beneath

it, and consequently the 3D-DfT architecture needs to control the corresponding test signals of the DRAM. Therefore, the WIR is extended with 13 bits controlling whether the DRAM is tested and if yes, which channels of which ranks should be included in the test. In addition, a basic memory controller is added to generate the DRAM control signals during test according to the instruction loaded in the WIR. This controller includes a number of multiplexers between the DRAM functional and test control signals, wrapper cells that provide observability of the functional control signals for chip select (CS_n), and registers controlling whether data are captured in the logic die or in the DRAM die during Extest. The boundary register of each rank, which is accessible through a serial interface, is included into the test path by concatenating it to the die WBR, so that test stimuli can be scanned into it, and test responses out of it. The included DRAM controller allows independently testing any rank-channel combination.

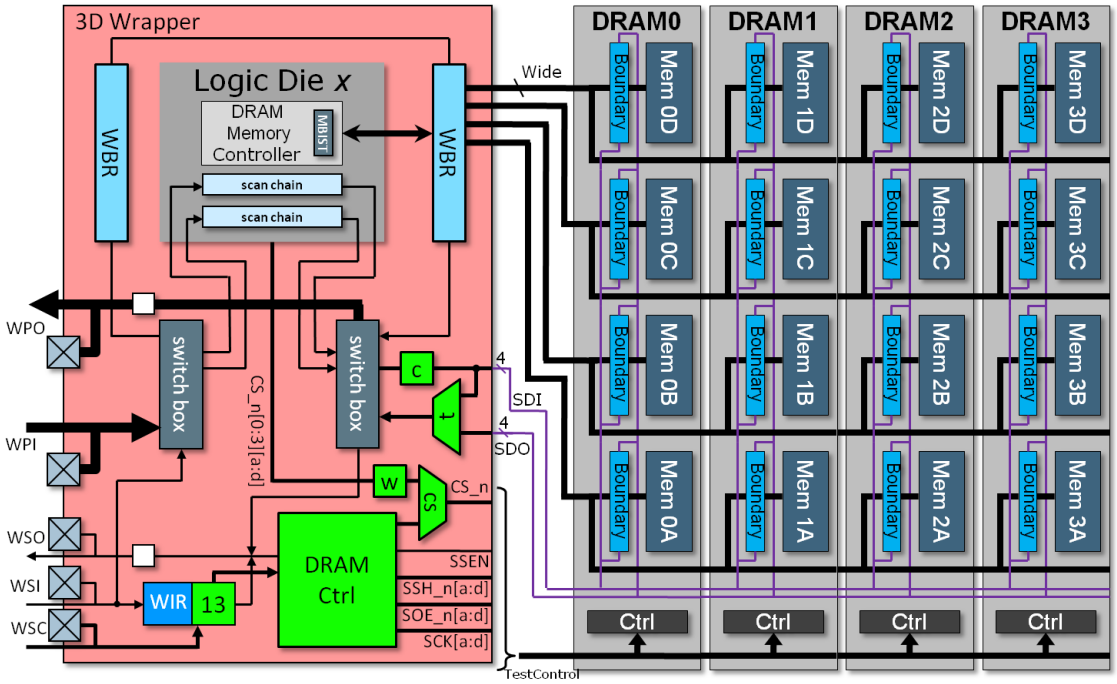


Figure 3.6: 3D-DfT architecture extended for Wide I/O DRAM.

3.2.3 Shortcomings

Despite being the first standardizable 3D-DfT architecture that provides an overall solution to the problems and needs of 3D testing, the original IMEC architecture also has some shortcomings that were addressed in the work conducted in the context of this thesis. One disadvantage was that it did not support mid-bond testing of incomplete stacks missing the bottom die. However, it is very important that all potential tests are possible in order to maximize the compound stack yield. Furthermore, the first commercial applications of 3D-SICs are memory-on-logic stacks, containing one or more logic dies and a memory die on top. In such cases, it is crucial to be able to test the

memory once it is bonded with the logic die directly beneath it, and before continuing with the rest of the stacking process. Another shortcoming of the original IMEC 3D-DfT architecture concerns the width adaptation mechanism implementation incorporated in its switch box. It relies on concatenation for width adaptation, which is incompatible with test data compression as it will be explained more thoroughly in Chapter 4. Also, despite the fact that an extension for embedded cores was already proposed in [17], a shift in methodology is required as embedded cores often include test data compression themselves. Finally, the original IMEC architecture did not support multi-tower 3D-SICs, which are gradually moving closer from fiction to reality.

3D-DfT Architecture Extensions

4

This thesis has three aspects: (1) the definition and design of 3D-DfT architecture extensions, (2) their implementation, and (3) the automation of the implementation in industrial electronic design automation (EDA) tools. From an architectural aspect, the original IMEC 3D-DfT architecture is extended so that it is compatible with *test data compression* (TDC), *embedded cores*, and *multiple towers*. The support of TDC is crucial as there are hardly any chips nowadays that do not include TDC; without it, their testing would be a far more time-consuming and expensive process. Modular systems-on-chip (SoCs) including embedded cores are also very common since the reuse of embedded cores in various designs saves significant development time. Finally, a trend towards multi-tower 3D-SICs seems to be evolving with stacks branching off in e.g. one logic and one memory tower. Such physical structures require special support in the 3D-DfT architecture.

4.1 Extended 3D-DfT architecture requirements

The presented 3D-DfT architecture is an extension of the IMEC architecture. As a result, the requirements of the original architecture also hold for the extended architecture and are therefore briefly listed below. First, modular testing of any component of any die in the stack has to be possible. Reconfiguration of the test access path enables reaching specific components no matter how high they are located in the stack or how deep in the design hierarchy. Furthermore, the architecture has to include a mandatory serial and an optional parallel TAM. All three of pre-bond, mid-bond, and post-bond testing have to be supported. Finally, TSVs/micro-bumps consume less area compared to probe pads. Therefore, the parallel TAM widths for pre-bond and post-bond tests should be allowed to be different. This enables individually tuning the two TAM widths according to the available area resources and the test time constraints.

The first major additional requirement of the extended 3D-DfT architecture is compatibility with *TDC*. TDC is a DfT technique that is part of most modern SoC designs. Its purpose is to make a design appear in test mode smaller than it really is. This is accomplished by scanning-in test stimuli that have been compressed off-chip after ATPG. These are then decompressed on-chip and used to drive a greater number of internal scan chains (scan channels). Test responses are compressed on-chip before scanning them out, thus combining the responses of different scan channels together into a smaller number of scan chains. This way, a design with many, long scan chains appears to have fewer, shorter scan chains externally. As a result, the test time as well as the number of required scan-in/scan-out I/Os are both reduced.

Compatibility with *embedded-core-based designs* is the second major requirement of the extended 3D-DfT architecture. The reason is that the use of embedded cores is

common practice in modern designs, as their reuse saves significant development time. Embedded cores are wrapped according to IEEE Std 1500 in order to provide a standardized test interface. The 3D-DfT architecture needs to be extended in order to provide an interface that is compatible with those wrappers. The support for embedded cores also needs to be closely coordinated with support for TDC, as many cores are quite sizable and therefore implement TDC themselves nowadays.

Supporting *multi-tower* 3D-SICs is the third major requirement of the extended 3D-DfT architecture. As 3D stacks get larger and more elaborate, they may branch-off into multiple towers and sub-towers, creating complex physical structures with dies containing more than a single die directly above them. This way, the functionality can be broken down with finer granularity into heterogeneous, more densely interconnected dies, which can increase the positive effects of the 3D integration technology on performance and power dissipation. The extended 3D-DfT architecture has to support a scalable number of towers on top of a die and solve the problems of test path configuration, and test control and instructions for them.

Finally, the extended 3D-DfT architecture has an additional two requirements. The first one is the *sharing of test I/Os* with functional I/Os and of serial TAM I/Os with parallel TAM I/Os in order to minimize the number of extra I/Os added to a die by the 3D-DfT architecture. The second feature is the support for *mid-bond testing* of an incomplete stack missing the bottom die in order to enable all possible tests.

4.2 Test data compression

The following three sections describe the complications introduced by TDC and the required modifications/extensions to the 3D-DfT architecture regarding width adaptation, test scheduling, and WBR organization in order to overcome them.

4.2.1 Scan chain concatenation and the scrambling function of TDC

TDC is incompatible with concatenation due to the scrambling of data as they go through the decompression and the compression logic. Scan chain concatenation can be used for two purposes. The first one is width adaptation between different TAMs, and the second one is parallel test schedules, where multiple components are concatenated in the scan path and tested simultaneously. The scrambling function of TDC is illustrated in Figure 4.1. A single bit value on the output of the compressor (first bit of scan chain SC2) might depend on a number of bit values on multiple scan channels (last bits of channels CHAN0 and CHAN1). These bits can be propagated from the beginning of channels CHAN0 and CHAN1 while scanning. These bits (first bits of channels CHAN0 and CHAN1) might in turn depend on a number of bit values on multiple inputs of the decompressor (last bits of scan chains SC0 and SC1). The required bit-mapping all the way to the scan inputs is a complex operation. In case of concatenation of TDC engines one after the other, or of feedback loops feeding TDC engine outputs to TDC engine inputs for width adaptation, the problem of bit-mapping becomes even more complex due to the increased number of correlations. The scrambling function of TDC obstructs both scanning-in stimuli

through a TDC engine to a post-pended scan chain, and scanning-out responses through a TDC engine from a pre-pended scan chain.

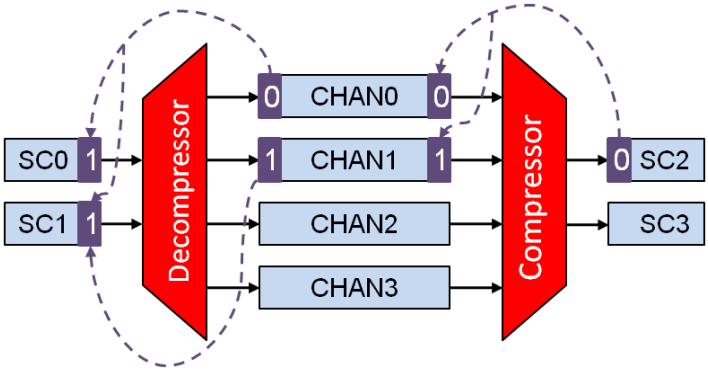


Figure 4.1: Mapping of test data across a TDC engine.

In certain cases solving the problem of bit-mapping the test data all the way to the scan inputs might be feasible, but in the general case it is not. There are mainly two reasons for that. First, the principle behind the operation of TDC is exploiting the low density of care-bits during testing. In fact, most test patterns contain a lot of don't care bits (X) that can be filled with any value as they do not affect the sensitization of the targeted faults or their propagation to the output where they can be observed. Typical don't care bit densities for modern designs and ATPG engines exceed 90%. This fact makes the bit-mapping of test data feasible, as only a small number of bits really matters. The remaining bits can be arbitrarily filled, only paying attention that their values do not obstruct the generation of the care bits values. However, if data has to be scanned through a TDC engine, the care-bit density is essentially 100% as the data needs to propagate through the TDC engine intact. This makes solving the problem of bit-mapping infeasible due to the many, often contradicting correlations and dependencies. Second, various implementations of TDC exist and the way some of them work makes scanning through them impossible by definition. Such an example is a TDC engine employing a MISR. In that case, signatures are scanned-out only occasionally, thus disrupting the continuous stream of test data to the next component on the scan path. Even if someone decided to use the signature scan-out function to propagate test data through the TDC engine, it would be very time consuming and sometimes impossible to generate the desired signature in the MISR. Concatenating compressed scan chains is then out of the question.

4.2.2 Width adaptation

In order to overcome the inability to concatenate in the presence of TDC, the width adaptation has to be performed by dedicated deserializers and serializers instead of multiplexers. These can up-convert or down-convert between the various TAM widths and "bridge" them in a way independent from the internal circuitry and DfT components. First, the serial TAM (STAM) and the parallel TAMs (PTAMs), and their widths are defined:

STAM (1):
The STAM (WSI/WSO) might use either dedicated probe pads or TSVs/micro-bumps for its I/Os and its width is always one throughout the stack. The STAM is the TAM where test instructions are multiplexed together with test data.

Stack PTAM (n):
The stack PTAM (WPI/WPO) uses TSVs/micro-bumps for its I/Os with the exception of the bottom-side of the bottom die where pads are used to interface with the surrounding environment. This TAM width has to be globally agreed between all the dies in a stack in order to create a consistent stack-wide TAM.

Pads PTAM (m):
The pads PTAM (WPI_pad/WPO_pad) uses dedicated probe pads for its I/Os. Its width is a local decision that can be different per die. Pads consume significantly more area than TSVs/micro-bumps, therefore it makes sense to allow the designer to efficiently balance the area resources and test time requirements for pre-bond testing. A conversion between the pads PTAM width and the stack PTAM width is necessary in order to enable mid-bond testing of incomplete stacks missing the bottom die.

Internal PTAM (k):
The internal PTAM width could be defined to always be equal to the stack-wide parallel TAM width. However, additional flexibility can be provided to the die designer by defining the die-internal parallel TAM width to be a local decision per die and convert from/to the stack-wide parallel TAM width.

The above widths are expected to obey the following ordering $1 \leq m \leq n \leq k$ in any given die. Careful planning of the organization of the de-/serializers (*SerDes*) provides the opportunity to also enable pre-bond testing of incomplete stacks missing the bottom die. This is the reason the deserialization and serialization process is split into two discrete stages. One SerDes pair can convert between the stack PTAM width (n) and the pads PTAM width (m) to always adapt the TAM width of a die to the stack PTAM width if needed. A second SerDes pair can then be used to convert between the stack PTAM width (n) and the internal PTAM width (k) if needed. Furthermore, extension of this second SerDes pair to also bridge the STAM (1) and the internal PTAM (k) enables testing compressed scan chains through the STAM. This results in significant test time reduction when limited resources prohibit the implementation of a PTAM. A dedicated multiplexer is also required to multiplex the compressed and the uncompressed STAM. The various TAMs and their widths as well as the SerDes organization previously described are depicted in Figure 4.2.

A complication of de-/serialization is the need for clock control. The reason is that the available TAM bandwidth remains the same, while the TAM width changes, therefore mandating an adaptation in the timing. Essentially, all scan chains before the deserializer and after the serializer (external) belong to one clock domain, while all scan chains between the deserializer and the serializer (internal) belong to another clock domain. The modified clock should only trigger the internal scan chains once the deserializer has been fully loaded and the serializer fully unloaded. More details on the functionality and

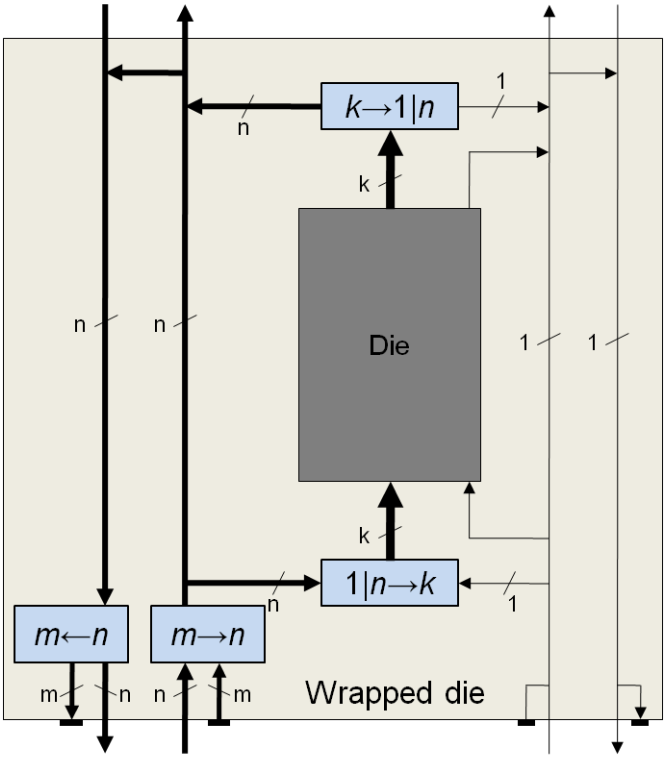


Figure 4.2: 3D-DfT architecture TAMs, TAM widths and SerDes organization.

the implementation of the two SerDes pairs and their clock control units are given in Section 5.3.

4.2.3 Test scheduling

In general, test schedules are divided in sequential and parallel. Figure 4.3 illustrates the differences between (a) a sequential and (b) a parallel test schedule on a simple example with two bypass-able scan chains. The active test path that is scanned in each case is shown in red color. In sequential test schedules, one component is tested at a time by only loading and unloading its scan chains, while the other components are bypassed. In parallel test schedules on the other hand, the scan chains of multiple components are concatenated so that all of them can be loaded and unloaded in a single scan-in/scan-out sequence. Parallel test schedules require spending fewer clock cycles scanning through bypass registers in order to reach the component(s) to be tested. Moreover, they allow overlapping the test application cycles of several components that would otherwise be tested separately, thus further saving on test time.

Despite the fact that parallel test schedules offer shorter test times, sequential test schedules are often preferred in practice for two reasons. First, they make diagnosis easier. When a test fails, identifying the faulty component is a trivial task since only one component is being tested at a time. Second, the independence of tests from each

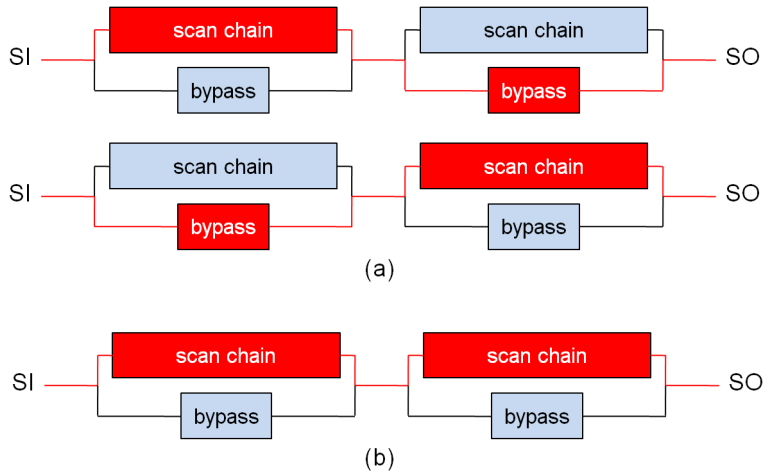


Figure 4.3: An example of a sequential test schedule (a), and parallel test schedule (b)

other, allows for easy test rescheduling. This property is particularly useful in case the yield of a specific component increases or decreases, as its tests can then be moved later or earlier in the test flow respectively.

The original 3D-DfT architecture supports both sequential and parallel test schedules. However, as explained in Section 4.2.1, concatenating TDC engines is infeasible and therefore parallel test schedule support has to be dropped in the extended architecture. It is interesting to quantify the impact of the loss of parallel test schedules on test time for realistic designs. In general, a test scheduling problem can be defined in the format of the example of Figure 4.3, by specifying three parameters: (1) the number of bypass-able scan chains (n) , (2) the length of each scan chain ($l_i : 1 \leq i \leq n$), and (3) the number of test patterns that need to be applied to each chain ($p_i : 1 \leq i \leq n$). The test time overhead due to sequential test scheduling is maximized when all the tests can be fully overlapped, i.e. when all p_i are equal. Otherwise, once all test patterns of a component have been applied, it has to be bypassed for the remaining test patterns, thus reducing the parallelism. Hence, calculating the test time overhead in such a case provides an upper bound for it and therefore allows drawing safe conclusions about the worst-case impact of dropping parallel test schedules. If all scan chains require p patterns each, the test time when using a sequential and a parallel test schedule is given by formulas (4.1) and (4.2) respectively.

$$T_{SEQ} = \sum_{i=1}^{n} (p \cdot (max(i-1, n-i) + l_i) + p + (min(i-1, n-i) + l_i)) \tag{4.1}$$

$$T_{PAR} = p \cdot \sum_{i=1}^{n} l_i + p + \sum_{i=1}^{n} l_i \tag{4.2}$$

From the formulas it can be concluded, that only the total scan length of all the scan chains ($\sum_{i=1}^{n} l_i$), and not the distribution of the total length over the individual scan

chains (l_i), is important. So, the length of each scan chain can be assumed to be equal to l. Table 4.1 contains calculations of the test time overhead of sequential over parallel test schedules in various cases that are representative of realistic designs. The most influential factor is the total scan chain length. The reason is that the longer the scan chains, the longer the scan-in/scan-out time, and therefore the smaller the overhead added by the bypass registers. The second most influential factor is the number of bypass-able scan chains. The greater their number, the greater the benefit of parallelism. Finally, the number of test patterns per scan chain contributes to the test time overhead only slightly. It has to be noted, that these numbers are an upper bound of the test time overhead, as they assume full overlap of all tests in a parallel test schedule. Hence, it can be concluded that the consequences of the loss of parallel test schedules in the extended 3D-DfT architecture are limited, unless the modules that are tested separately are great in number and small in size.

4.2.4 WBR organization

An essential part of the wrapper is the boundary register, which intercepts the die I/Os and provides controllability and observability to them. As suggested by its name, a wrapper, including the WBR, should ideally surround a die. In that case, the WBR would simply be pre-pended and/or post-pended to the die-internal scan chains. In the presence of TDC, this practice is impossible. Providing test stimuli to a wrapper cell that is post-pended to a TDC engine and propagating captured test responses from a wrapper cell that is pre-pended to a TDC engine, would require scanning test data through the TDC engine without corrupting them. As this is impossible, the WBR needs to be compressed for the Intest mode. An advantage of having a compressed WBR for Intest is that significant test time might be saved. The reason behind this, is the fact that both the I/O count of chips and the number of test patterns required to test them, keep increasing with their size. The test patterns for Intest involve the WBR, as controllability of inputs and observability of outputs is required. As a result, loading the WBR using compressed test patterns becomes an important feature with increasing chip size. On the other hand, inter-die Extest mandates concatenation of the WBRs of two or more neighboring dies in a stack. This is necessary because of the nature of inter-die interconnect Extest. To test an interconnect between two neighboring dies, a wrapper cell on one die needs to provide a stimulus to the interconnect under test, while a wrapper cell on the other die needs to simultaneously capture its response. The two constraints on the WBR that were described previously, lead to the requirement that the WBR needs to be compressed for Intest and uncompressed for Extest.

A necessary change in the conceptual view of the architecture in order to accommodate this requirement is that the WBR and the Intest/Extest functionality of the wrapper are pushed inside the die. This way, the WBR should be inserted before the TDC so that it can be organized in a compressed Intest mode as well as an uncompressed Extest mode. An example application of the extended 3D-DfT architecture on an abstract die without TDC is shown in Figure 4.4 and on an abstract die with TDC in Figure 4.5. The dimensions of the boxes representing the scan chains are roughly proportional to the TAM width and the scan length. Therefore, the boxes correspond-

n	l	p	$(T_{SEQ} - T_{PAR})/T_{PAR}$
2	100	100	1.478%
		1000	1.491%
		10000	1.492%
	1000	100	0.148%
		1000	0.150%
		10000	0.150%
	10000	100	0.015%
		1000	0.015%
		10000	0.015%
5	100	100	3.960%
		1000	3.989%
		10000	3.992%
	1000	100	0.397%
		1000	0.400%
		10000	0.400%
	10000	100	0.040%
		1000	0.040%
		10000	0.040%
10	100	100	7.834%
		1000	7.886%
		10000	7.892%
	1000	100	0.784%
		1000	0.789%
		10000	0.790%
	10000	100	0.078%
		1000	0.079%
		10000	0.079%
50	100	100	37.715%
		1000	37.946%
		10000	37.970%
	1000	100	3.772%
		1000	3.795%
		10000	3.798%
	10000	100	0.377%
		1000	0.380%
		10000	0.380%
100	100	100	74.978%
		1000	75.432%
		10000	75.477%
	1000	100	7.498%
		1000	7.544%
		10000	7.548%
	10000	100	0.750%
		1000	0.754%
		10000	0.755%

Table 4.1: Test time overhead of sequential over parallel test schedules.

ing to the parallel TAM are wider and their scan length is shorter compared those of the serial TAM. Similarly, the boxes corresponding to Intest chains, which contain the die-internal scan chains and the WBR, are longer than those of the Extest chains, which only contain the WBR. Finally, the compressed scan chains are shorter and greater in number compared to the uncompressed scan chains containing the same scan elements, and this is also reflected by the dimensions of the corresponding boxes.

A consequence of pushing part of the wrapper functionality inside the die is that the die needs to comply with certain requirements. Two fundamental requirements concern the test infrastructure, and are that the WBR and the Intest/Extest functionality need to be provided by the *DfT-Ready* die. Additional requirements concern the test control interface between the die and its wrapper. The minimum required set of test control signals has to define whether the die is accessed through the serial or the parallel TAM and whether it is in its Intest or Extest mode. Of course, any test signals controlling additional on-die DfT components such as a TDC engine need to be included in that test control interface as well.

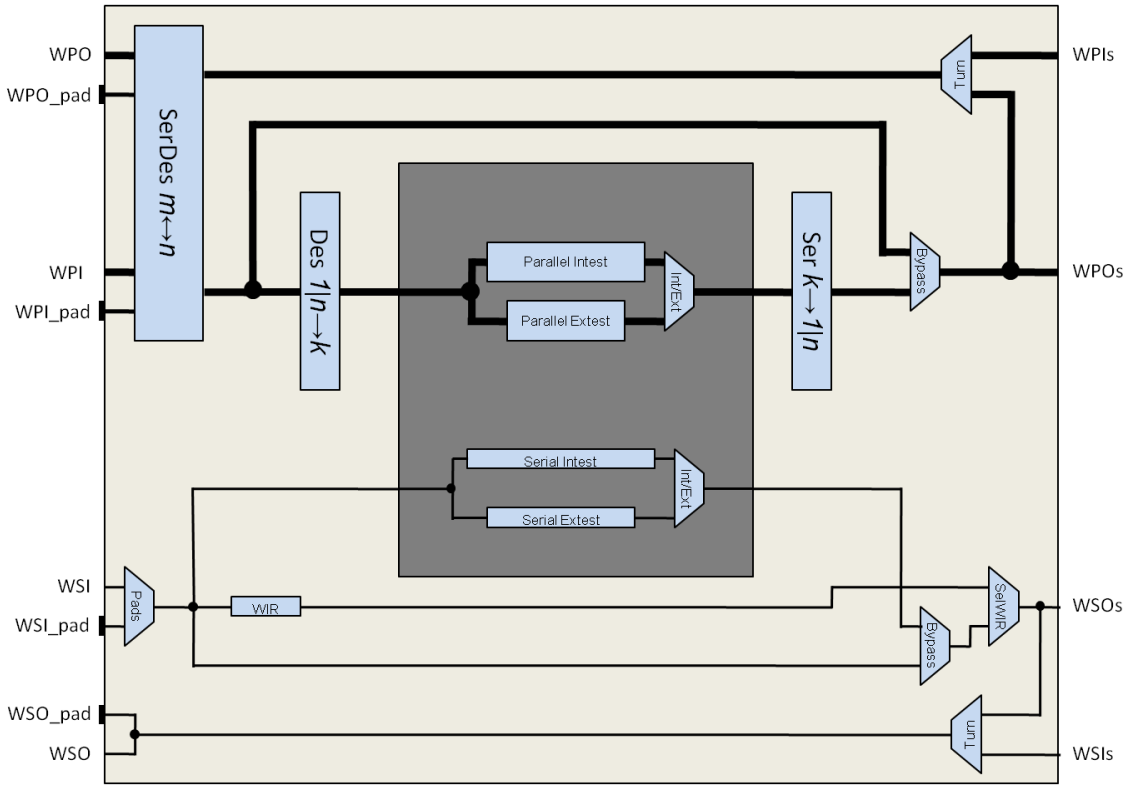


Figure 4.4: Wrapped die with plain scan chains.

4.3 Embedded cores

The extended 3D-DfT architecture needs to support embedded cores as they are standard practice in modern designs. Embedded cores of a *system-on-chip* (SoC) are similar to

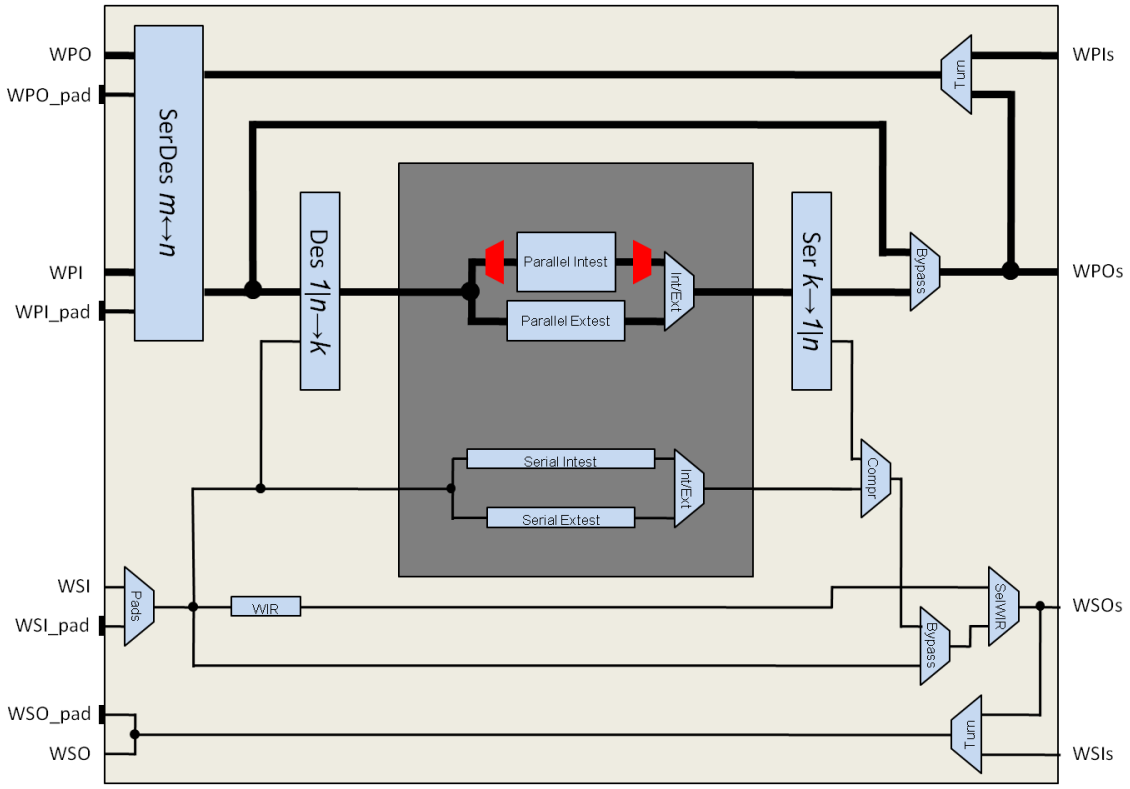


Figure 4.5: Wrapped die with TDC.

dies of a 3D-SIC in the sense that both exploit the concept of modularity to enable the reuse of designs. Supporting embedded cores in the extended 3D-DfT architecture requires solving two problems. First, a consistent test data interface between the die and the cores needs to be created, so that the embedded cores can be included into the test path. Regarding this, special care in the organization/interconnection of the embedded cores and the top-level is needed when embedded cores are combined with TDC. The root of the problem lies in the fact that performing a die Intest requires the die's embedded cores to participate to the test in their Extest modes. The reason is that the cores need to provide stimuli to the die and capture responses from the die, as illustrated in Figure 4.6. Nevertheless, the scrambling of stimuli by the TDC engine makes the loading of the core WBR chain(s) impossible if these are concatenated after the die's internal chains and WBR. Therefore, a different connection topology would be required in a case where both TDC and embedded cores coexist in a die.

One potential interconnection topology that solves the problem is the distribution of the test bandwidth over the various components to be tested. For example, the die-level scan chains could be assigned to a subset of the TAM wires, while each core would be assigned to a different dedicated subset of TAM wires. However, such a topology, which implements a so-called distribution architecture, increases the test time of each component. The only case when the total test time is not increased, is if all existing components are tested in parallel, thus exploiting the full available bandwidth. Another potential

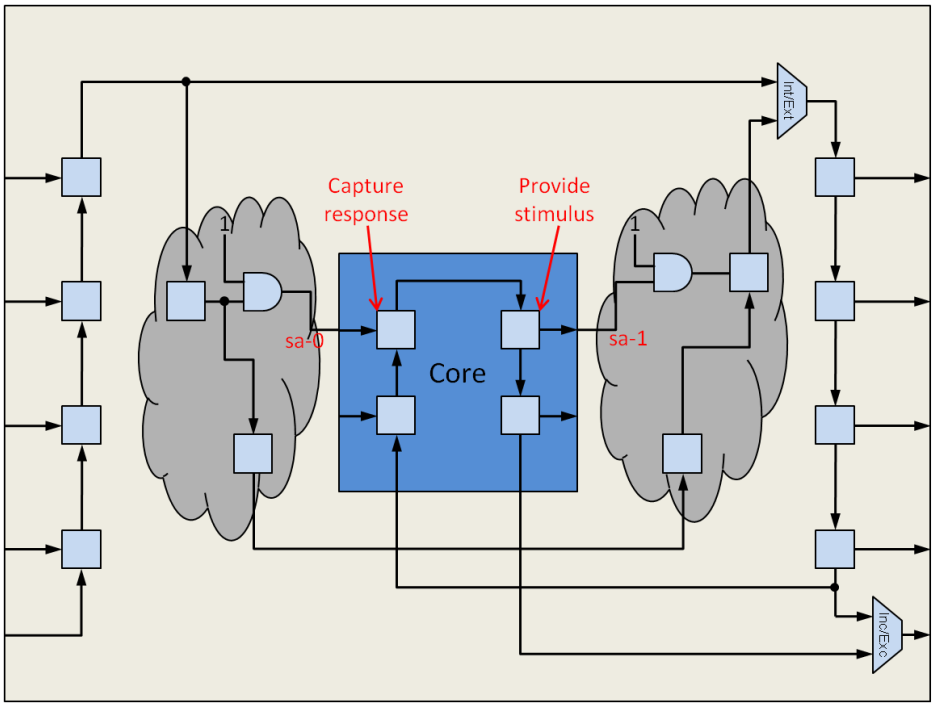


Figure 4.6: Complication of TDC when performing a die Intest and core Extest.

topology that avoids the above problem would be connecting the various components in a broadcast network and taking advantage of the scrambling properties of TDC. In such a broadcast architecture, the decompressors of the die and each embedded core, all receive identical test stimuli, but their scan channels receive different test stimuli depending on the implementation of each decompressor. On the responses side, all responses are combined in order to match the available wrapper TAM width by introducing a second compression level. Of course, the created test stimuli correlations and the increased test response compression ratio might result in a fault coverage reduction, which is negligible in most realistic cases. In cases where the achieved fault coverage is not satisfactory, an uncompressed mode can be provided as well to complement the initial fault coverage through additional, uncompressed patterns. An additional advantage of the broadcast architecture is in testing identical cores which need the same test stimuli and can thus be tested simultaneously in a natural way. An example of a die containing one embedded core and TDC, and implementing the broadcast architecture is shown in Figure 4.7.

The above lead to the realization that the design space of solutions that are available to a designer is quite extensive, and consequently so are the die types that might need to be wrapped. An easy way to support a multitude of different dies is by extending the concept of abstraction first introduced in the previous section, by pushing the task of the Intest mode(s) configuration to the die designer. By abstracting from the details of the wrapped die and requiring the designer to provide an interface of the test control signals, significant extra flexibility is gained in the die types that can be wrapped. This flexibility even includes dies with custom DfT components, custom interconnection

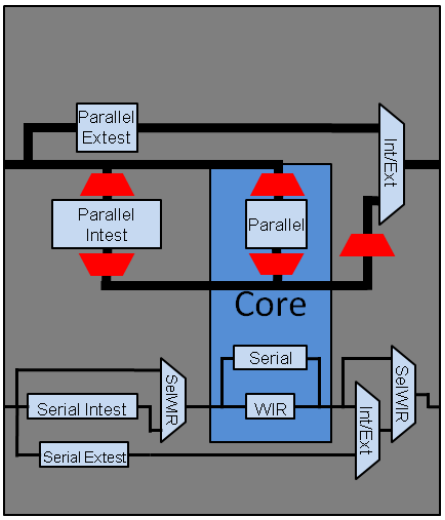


Figure 4.7: Broadcast architecture on a die with an embedded core and TDC.

topologies and/or custom test mode configurations.

The second problem that needs to be solved to support embedded cores in the extended 3D-DfT architecture, concerns test control and instructions. A mechanism that allows programming the core WIRs needs to be put into place. However, caution has to be taken while establishing the WIR path, as it can grow quite quickly in length. In order to limit the WIR path's scan length, the established mechanism needs to allow the inclusion of the core WIRs in the path or their exclusion from it as needed. Perhaps the simplest way to achieve the described behavior and functionality is by including two extra multiplexers to the 3D-wrapper. One multiplexes the test data and the test instruction paths that go into the die, so that the die WIR can be concatenated to the core WIR(s). The other selects between the path that includes the core WIR(s) and the die WIR, or only the latter. A consequence of this mechanism is that the WIR programming has to be performed in an incremental fashion. During the first WIR programming iteration the die WIR needs to be set to include the core WIR(s) in the test path. During the second iteration, the core WIR(s) can be programmed together with the die WIR. An additional provision that has to be taken is the gating-off of the embedded cores' clocks when they are not included into the test path. The WIR bit that controls the inclusion/exclusion of the core WIR(s) into/from the WIR path, can be used to gate-off their update signal (UpdateWR) as well. As a result, the test mode of the cores will only be updated if their WIRs are included in the test path, otherwise the default mode activated upon reset (WRSTN) is preserved. Also, the embedded cores can remain in a low-power test mode when inactive, in order to reduce the overall power dissipation during test and prevent potential damage to the chip due to overheating.

The 3D-DfT architecture wrapping a die containing an embedded core without TDC is shown in Figure 4.8, while a wrapped die with both an embedded core and TDC is illustrated in Figure 4.9. In Figure 4.8, performing a core Intest can be accomplished by bypassing the top-level scan chains, while in Figure 4.9, the same test would need these

chains to always output zeros so that they have no impact on the output. It has to be noted that the number of embedded cores is scalable. Furthermore, the interconnection topology and test mode configuration of the die top-level and the embedded cores are up to the die designer, and Figures 4.8 and 4.9 merely provide implementation examples. This flexibility is the result of the abstraction from the die details that was described before.

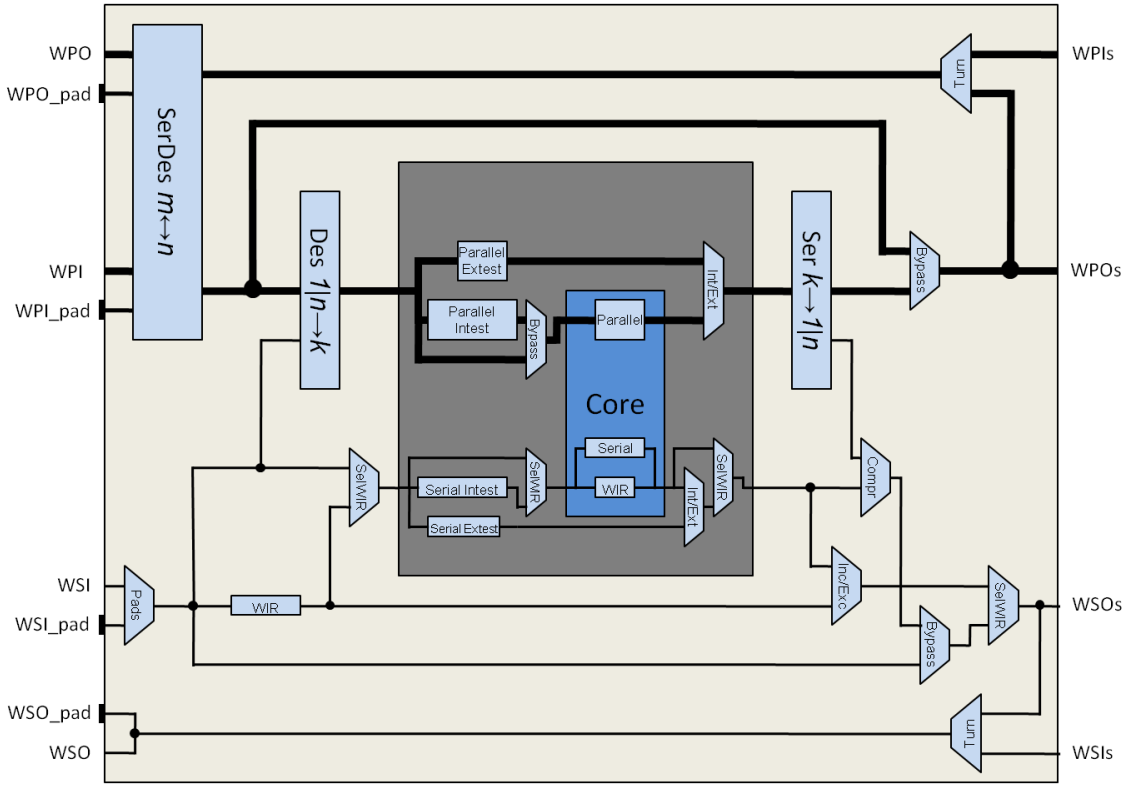


Figure 4.8: Wrapped die with an embedded core.

4.4 Multiple towers

As 3D-SICs become larger and more complex, having the stack branch-off into multiple towers seems like a natural evolution. This way, the functionality can be broken down with finer granularity into heterogeneous, more densely interconnected dies, which can increase the positive effects of the 3D integration technology on performance and power dissipation. The extended 3D-DfT architecture has to support a scalable number of towers on top of a die and solve the problems of test path configuration, and test control and instructions for them.

The problem of test path configuration consists in elevating test data up and down in the various towers of the 3D-stack. This requires an elevator/turn functionality similar to that present in the original 3D-DfT architecture. However, this mechanism now has to be able to provide the same functionality for multiple towers. So, first of all, each

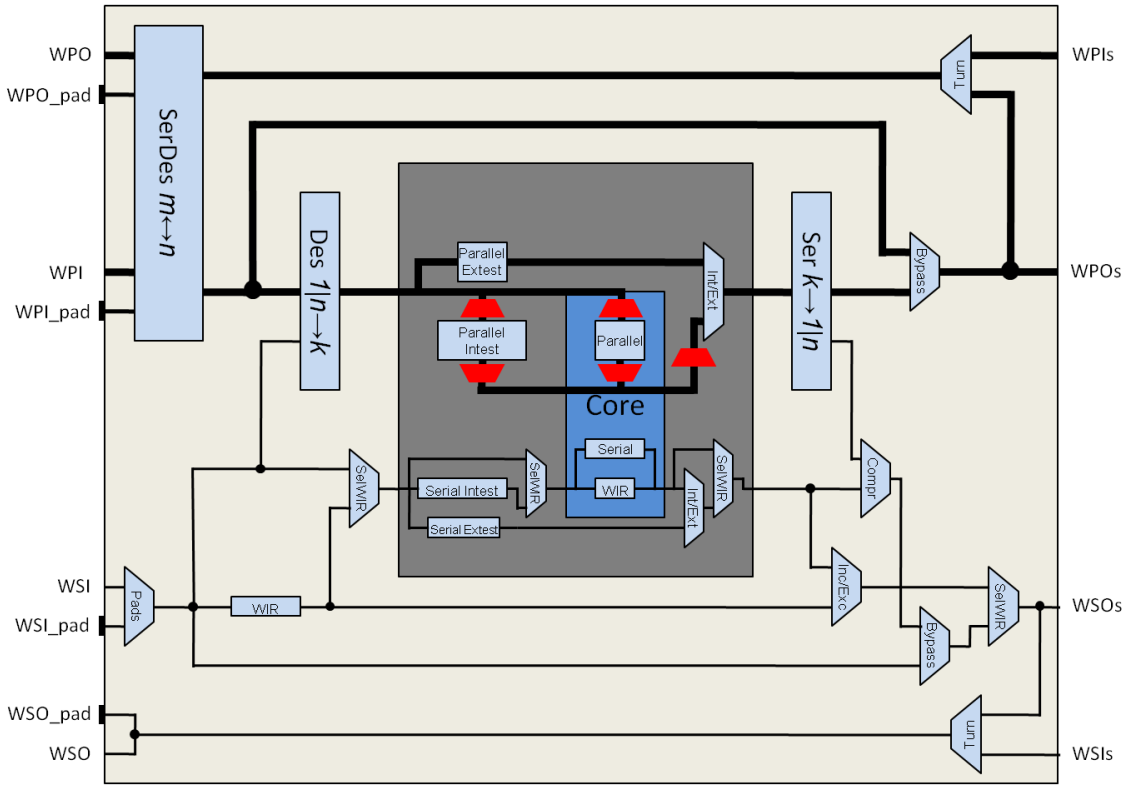


Figure 4.9: Wrapped die with an embedded core and TDC.

die destined to be bonded with n dies directly above it, has to implement n test ports on its top side in order to interface with them. Each such test port has to include the WSCs, WSIs, and WSOs signals, and in addition to them, the WPIs, and WPOs signals if a parallel TAM also exists. Regarding the test path configuration, two different granularities exist: (1) *per-tower*, and (2) *per-level*. A stack example implementing per-level test path configuration is shown in Figure 4.10, which only depicts the WIR path of the serial TAM. The same example would only include elevator/turn multiplexers in dies 1 and 2 if it implemented per-tower test path configuration. Those dies are namely the branch-off points where the stack is split into multiple towers. While both are valid solutions, the requirement for support of mid-bond testing rules out per-tower configuration. That is because a tower might still be incomplete or missing altogether at a certain testing moment. Then, maintaining the continuity of the test path requires being able to turn the test path at any given die. Per-level test path configuration is thus the preferred solution.

Test control and instructions are an equally important part of multiple towers. In order to prevent uncontrolled growth of the WIR path's scan length, an incremental mechanism for including only the necessary WIRs into the WIR path is required. To accomplish that, one additional WIR bit is introduced for every tower. That WIR bit serves two purposes. First, it controls the elevator/turn multiplexers that include/exclude that specific tower in/from the test path. This functionality is shown in Figure

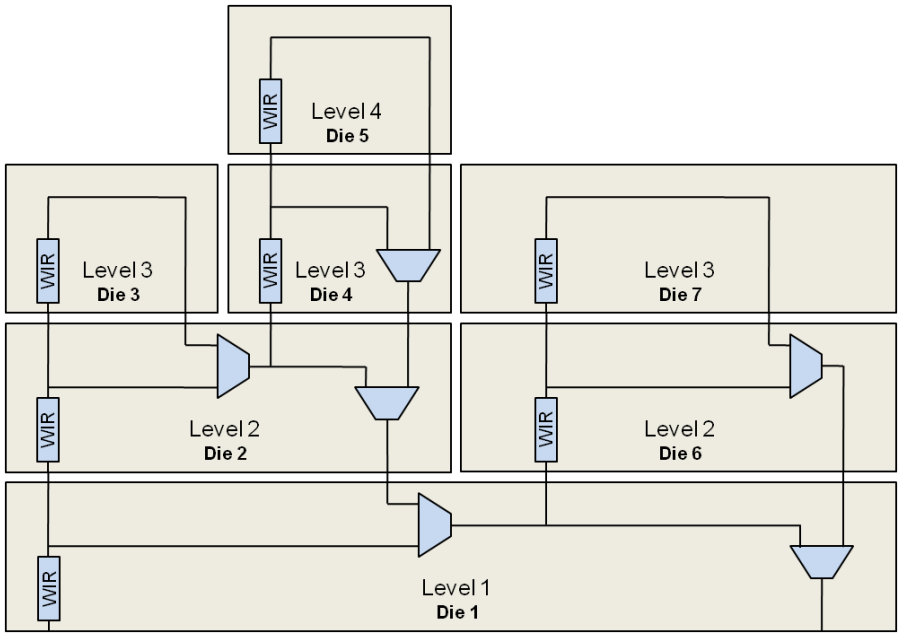


Figure 4.10: Per-level multi-tower test path configuration example.

4.11, which only depicts the WIR path of the serial TAM. Second, it gates-off the UpdateWR signal of an excluded tower, so that its WIR state will not be corrupted while the other WIRs are programmed.

The fact that one WIR contains a bit that determines whether another WIR is part of the WIR path or not, raises the need for an incremental WIR programming mechanism. During the first iteration, only the WIR of the bottom die (1st level) is included in the WIR path and can be programmed to include one or more WIRs of dies on the next level. During the second iteration, the WIR of the bottom die (1st level) and any dies on the 2nd level can be programmed, and so on. Programming the WIRs up to level n would thus require n programming iterations. For example, programming the WIR of die 5 in Figure 4.10, would require 4 WIR programming iterations. Clearly, there are several programming sequences that would lead to the desired result. However, some of them are optimal, while others other sub-optimal in terms of test time. For instance, programming dies 5 and 6 can be accomplished through both of the following programming sequences:

die1	die1, die2, die6	die1, die2, die4, die6	die1, die2, die4, die5, die6

die1	die1, die2	die1, die2, die4	die1, die2, die4, die5, die6

The first sequence is sub-optimal, while the second one achieves optimality by delaying the inclusion of the WIR of die 6, which is on a lower level than that of die 5, until necessary. In general, the stack can be seen as a tree, where the bottom die is the root. Programming should then start from the longer branches, and shorter branches should be left for the end. To be precise, programming of a branch should only start once its

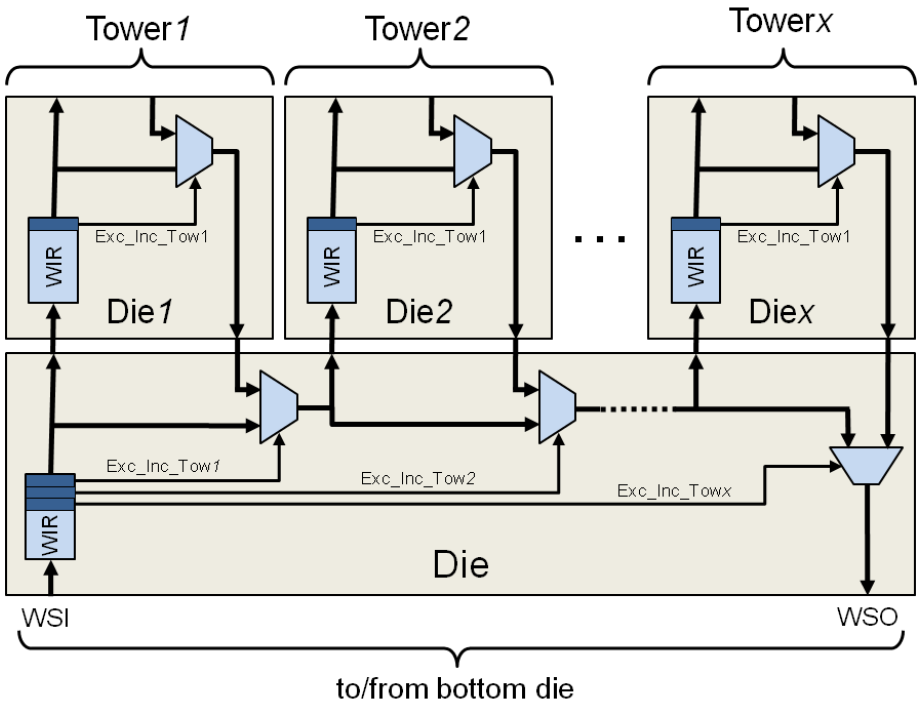


Figure 4.11: Detailed view of the WIR path of a multi-tower stack.

un-programmed depth is greater than or equal to the depth of any other un-programmed branch in the tree. By definition, a shorter, already programmed branch would have to be re-programmed while longer branches are programmed for the first time. As a result, this technique ensures that no WIR will be programmed more than once whenever that can be avoided.

4.5 3D-DfT architecture overview

An overall view of the full-featured 3D-DfT architecture, applied on a generic die is depicted in Figure 4.12. According to the requirements, the serial TAM I/Os are shared with I/Os of the parallel TAM in order to save on area resources. This feature is especially important for pre-bond tests through probe pads that are dedicated for test. If, for instance, the area resources only allow a pads PTAM width of four, the additional probe pads for the STAM I/Os are a significant additional overhead. The requirements on the DfT-Ready die are listed below:

- Its I/Os need to be intercepted by wrapper cells providing observability and controllability of their values.
- It has to contain scan chains that are organized in a serial TAM and optionally also in a parallel TAM. If so, an interface of the respective control signal must be provided.

- The scan chains and the WBR need to be organized in an Extest mode and one or more Intest modes. The configuration of the Intest mode(s) is the die designer's task. An interface of the respective test control signals must be provided.
- If TDC exists, the WBR needs to always be compressed for Intest and uncompressed for Extest.
- If one or more embedded cores exist, an interface of their WSC signals must be provided. Their WIRs have to be multiplexed onto the test instruction path of the die's serial TAM as dictated by IEEE Std 1500. The interconnection (daisy-chain, distribution, multiplexing, broadcast) of their serial and parallel TAM onto the die's serial and parallel TAM respectively is the die designer's task.

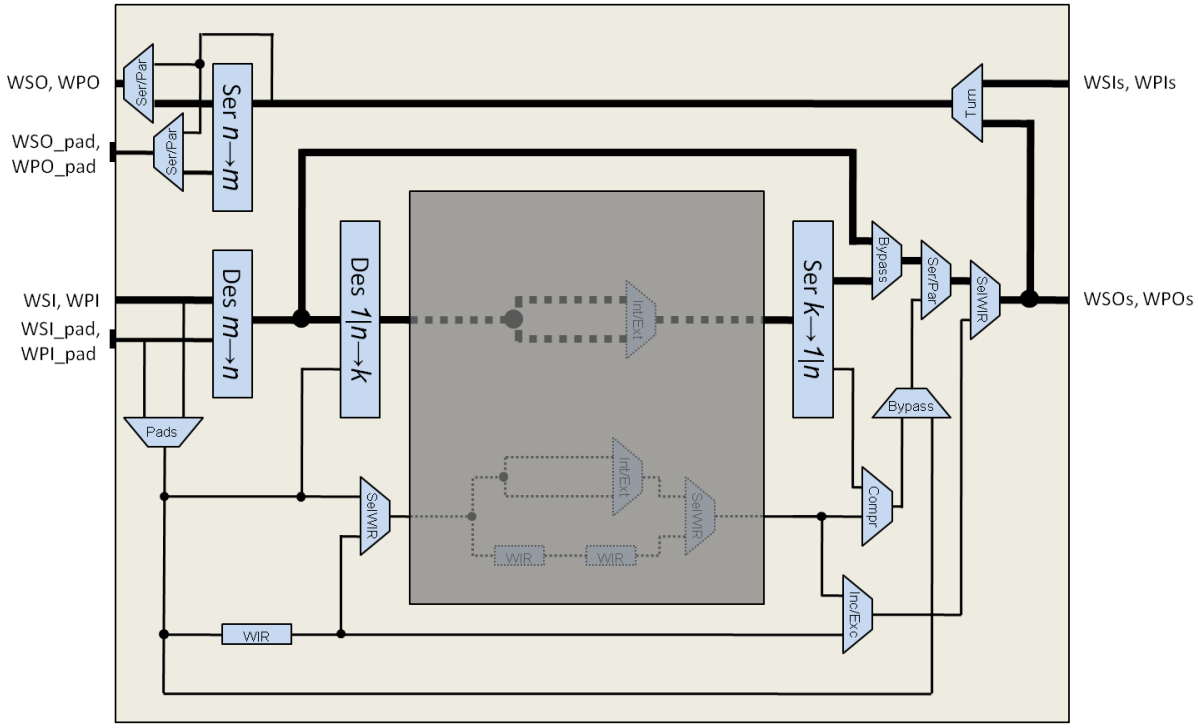


Figure 4.12: 3D-wrapped generic die (shared STAM/PTAM I/Os).

Implementation and Automation

5

The extended 3D-DfT architecture presented in Chapter 4 requires several components in order to be implemented. Some of the key components, such as the multiplexers implementing the various re-configurable test modes (Serial/Parallel, Intest/Extest/Bypass, IncludeTower*/ExcludeTower*, Fullscan/Compression, IncludeCores/ExcludeCores) were already described while presenting the architecture. Some other key components, which need further explanation, are presented in this chapter. These include (1) the *WIR* controlling the test mode of the die, (2) the *width adapters* bridging the different TAMs, and (3) the *I/O sharing* multiplexers between the test and the functional I/Os as well as between the serial and the parallel TAM I/Os. Furthermore, the implementation of the individual components of the extended 3D-DfT architecture needs to be *automated in a flow*. This allows the automated wrapping of designs as 3D-testable dies. The difficulty of automation lies in making the flow generic enough to support a multitude of different design cases. Finally, the 3D-DfT across the stack needs to be verified through simulation. An automatic, configurable flow was setup for that purpose. The functionality of this flow can easily be extended to generate test patterns for the production testing of a 3D-SIC.

5.1 Automation flows

Manually applying the 3D-DfT architecture to every individual design is a time-consuming and error-prone process. So is the generation of the scripts and files that are needed by the ATPG tool in order to generate the test patterns. Electronic design automation (EDA) tools automate the application of pre-defined processing steps to any given design according to a customized configuration. That way, the implementation is sped up significantly and also requires less effort per design. Two discrete automation flows have been designed and implemented in the context of this project. The first flow, the 3D-wrapper insertion flow, applies to individual dies and automates their wrapping according to the extended 3D-DfT architecture. Its purpose is to enable die manufacturers to prepare their dies for testing in a 3D-stack. The second flow, the verification ATPG flow, interconnects the individual 3D-wrapped dies into a stack and facilitates the execution of ATPG on the stack in order to verify the 3D-DfT. This flow can be extended in order to perform generation of test patterns suitable for production testing of 3D-SICs. An additional, third flow that is custom per design and under the die designer's control is needed in order to prepare the die for the 3D-wrapper insertion flow.

5.1.1 Die pre-processing flow

The 3D-wrapper insertion flow starts with the netlist of a DfT-Ready die that has undergone some pre-processing to be prepared for 3D-wrapping. These pre-processing steps push some of the 3D-wrapper functionality inside the hierarchy level of the die and place some requirements on the die. The requirements on the DfT-Ready die are listed below:

- Its I/Os need to be intercepted by wrapper cells providing observability and controllability of their values.
- It has to contain scan chains that are organized in a serial TAM and optionally also in a parallel TAM. If so, an interface of the respective control signal must be provided.
- The scan chains and the WBR need to be organized in an Extest mode and one or more Intest modes. The configuration of the Intest mode(s) is the die designer's task. An interface of the respective test control signals must be provided.
- If TDC exists, the WBR needs to always be compressed for Intest and uncompressed for Extest.
- If one or more embedded cores exist, an interface of their WSC signals must be provided. Their WIRs have to be multiplexed onto the test instruction path of the die's serial TAM as dictated by IEEE Std 1500. The interconnection (daisy-chain, distribution, multiplexing, broadcast) of their serial and parallel TAM onto the die's serial and parallel TAM respectively is the die designer's task.

The above requirements shift some of the weight of DfT onto the die designer's shoulders. In exchange, additional flexibility is gained in the designs that can be handled by the extended 3D-DfT architecture and the flow implementing it. In other words, the increase in the level of abstraction from the details of the die-internal DfT enables the automation of the wrapping of a wider range of design types. The pre-processing steps required in order to comply with the above requirements need to become an integral part of the die designer's flow. A sample pre-processing flow is outlined in Figure 5.1, where the 3D-specific steps are highlighted in red color.

First, the flip-flops of the top-level are organized into scan chains. Then, the WBR is inserted and the various top-level scan elements are organized in an Intest and an Extest mode. These two steps implement 3D-specific functionality and are therefore highlighted in red color in Figure 5.1. The reason for inserting the WBR during pre-processing, instead of with the rest of the 3D wrapper, is the requirement that the WBR needs to be compressed for Intest and uncompressed for Extest, as explained in Section 4.2.4. Therefore, it needs to be inserted before the TDC engine, which is a design-specific DfT component. So, once the Intest/Extest chains including the WBR are ready, TDC can optionally be inserted in the next step. Finally, the serial and the parallel TAMs of any embedded cores that might exist, can be connected onto the serial and the parallel Intest path of the die. Their interconnection topology should be in accordance with Section 4.3. For instance, the TAMs of the embedded cores can be concatenated with the top-level Intest chains during a top-level uncompressed Intest. However, these TAMs

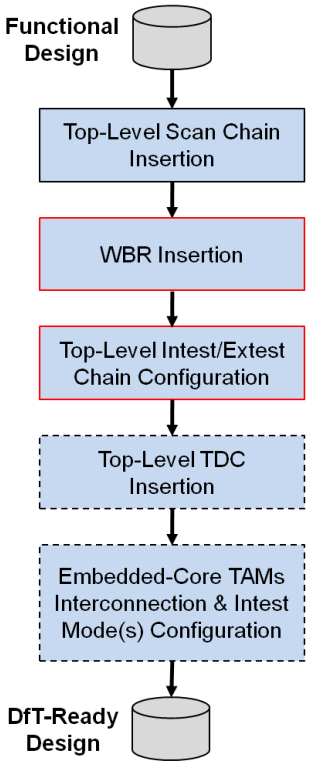


Figure 5.1: Sample die pre-processing flow.

have to be re-configured into a distribution or a broadcast interconnection architecture in order to allow for a top-level compressed Intest mode, where the cores participate in their Extest modes.

5.1.2 3D-wrapper insertion flow

Once the DfT-Ready design is completed, the desired settings of the 3D-wrapper need to be described in a configuration file. This configuration file contains the information required by the 3D-wrapper insertion flow in order to wrap the netlist of the DfT-Ready die. The die configuration file is conceptually organized in five sections. In the following paragraphs, the contents of each section are outlined and a template for each section is also provided. In the first section, general design and library information are given. The names of the top-level modules of the standalone die design and the wrapped design are supplied, as well as the name of the die instance within the wrapper. The paths to the input and output netlist files are listed next. If a custom pad detector implementation is available, its path should be provided here, otherwise a simulation model will be automatically inserted instead. Finally, the Liberty format standard-cell libraries for the design, and their corresponding Verilog models need to be given.

```
#########################################################
# Design module names, file names, paths and lib files
#########################################################
set DIE                  <design top-level module>
set DIE_INSTANCE         <design instance>
set WRAPPER              <wrapped design top-level module>

set DESIGN_FILES         <verilog netlist>
set WRAPPED_DESIGN       <verilog netlist>
set WRAPPED_DESIGN_JTAG  <verilog netlist>
set PAD_DETECTOR         <custom pad detector netlist>

set LIB_LIST             {<Liberty libraries>}
set TECH_LIB_LIST        {<library verilog models>}
```

In the second section, the functional I/Os including the functional clocks of the die are listed. The functional I/Os can be divided into bottom-side I/Os and top-side I/Os. If the die is not a top die, its top-side I/Os can in turn be grouped according to which tower they connect to. Finally, there are as many functional clocks as the die's clock domains.

```
#########################################################
# Die functional I/Os
#########################################################
set PFI                  {<bottom-side functional inputs>}
set PFO                  {<bottom-side functional outputs>}
set PFIO                 {<bottom-side functional bidirs>}

set SFI    {{<tower1 functional inputs>} {<tower2 functional inputs>}}
set SFO    {{<tower1 functional outputs>} {<tower2 functional outputs>}}
set SFIO   {{<tower1 functional bidirs>} {<tower2 functional bidirs>}}

set DIE_CLK              {<bottom-side functional clocks>}
```

The third section describes the interface of the die test I/Os. First, the pin name of the scan enable signal of the die is specified. Next, the test control signals that select the serial or the parallel TAM are given. The die might require two dedicated active-high signals, one for its serial and one for its parallel mode. Another option would be to use only one of these signals to activate a mode while the other mode is active by default. Of course, neither signal would be specified if the optional parallel mode is not implemented. Then, the signals controlling whether the die is in its Intest or Extest mode are specified. The definitions of custom on-die DfT control signals follow. These signals can be either linked to a dedicated WIR bit, or to a dedicated pin, the names of which are also specified. It has to be noted, that the WIR signal name fullscan_compr is reserved for the signal that enables the use of TDC. This section of the configuration file continues with the names of the serial and the parallel TAM I/O pins of the die. Finally, the WSC signal interface of the embedded cores is defined, if any exist.

```
#######################################################
# Die test ports and control signals
#######################################################
set DIE_SE               <die scan-enable signal>
set DIE_SERIAL           <die STAM enable signal>
set DIE_PARALLEL         <die PTAM enable signal>
set DIE_INTEST           <die intest enable signal>
set DIE_EXTEST           <die extest enable signal>

set DIE_TM_SIGNALS(<die test pin>) {<high/low/clk> <WIR/PIN> <pin name>}

set DIE_WSI              <die serial scan-in>
set DIE_WSO              <die serial scan-out>
set DIE_WPI              {<die parallel scan-ins>}
set DIE_WPO              {<die parallel scan-outs>}

set CORES                <YES/NO>
set DIE_WSC(WRSTN)       <die pin name>
set DIE_WSC(WRCK)        <die pin name>
set DIE_WSC(SelectWIR)   <die pin name>
set DIE_WSC(ShiftWR)     <die pin name>
set DIE_WSC(CaptureWR)   <die pin name>
set DIE_WSC(UpdateWR)    <die pin name>
```

The fourth section contains the wrapper test I/Os. The section begins with the pin names of any extra test control signals on the bottom-side of the die. This structure, combined with the corresponding output test signal structure, allows the definition of pass-through test signals as well. Next, the bottom-side I/Os of the WSC and the serial and parallel TAM are given. One variable controls whether a compressed serial mode should be implemented. In that case, the serial TAM and the parallel TAM are bridged through a de-/serializer pair and a multiplexer is inserted to select between the compressed and the uncompressed serial TAM. Subsequently, the names of the probe pad I/Os are listed if the insertion of such pads is desired. Finally, if one or more towers exist, the corresponding top-side wrapper test I/Os for each tower are specified. Each tower's test port is identical to the bottom-side test port, so that a consistent test interface between dies is available.

```
#######################################################
# Wrapper test I/Os
#######################################################
set signals_in           {<extra test signal inputs>}

set WSC(WRSTN)           <pin name>
set WSC(WRCK)            <pin name>
set WSC(SelectWIR)       <pin name>
set WSC(ShiftWR)         <pin name>
set WSC(CaptureWR)       <pin name>
set WSC(UpdateWR)        <pin name>

set WSI                  <wrapper STAM scan-in>
set WSO                  <wrapper STAM scan-out>
```

```
set SERIAL_COMPRESSION   <YES/NO>

set WPI                  {<wrapper PTAM scan-ins>}
set WPO                  {<wrapper PTAM scan-outs>}

# Probe pad I/Os
set PADS                 <YES/NO>

set WSC_PAD(WRSTN)       <pad name>
set WSC_PAD(WRCK)        <pad name>
set WSC_PAD(SelectWIR)   <pad name>
set WSC_PAD(ShiftWR)     <pad name>
set WSC_PAD(CaptureWR)   <pad name>
set WSC_PAD(UpdateWR)    <pad name>

set WSI_PAD              <wrapper STAM scan-in pad>
set WSO_PAD              <wrapper STAM scan-out pad>

set WPI_PAD              {<wrapper PTAM scan-in pads>}
set WPO_PAD              {<wrapper PTAM scan-out pads>}

# Tower I/Os
set IS_LAST_DIE          <YES/NO>

set secondary_ports      {<tower1>                    <tower2>}

set signals_out {{<tower1 test signals>}    {<tower2 test signals>}}

set WSCs(WRSTN)      {<tower1 pin name>          <tower2 pin name>}
set WSCs(WRCK)       {<tower1 pin name>          <tower2 pin name>}
set WSCs(SelectWIR)  {<tower1 pin name>          <tower2 pin name>}
set WSCs(ShiftWR)    {<tower1 pin name>          <tower2 pin name>}
set WSCs(CaptureWR)  {<tower1 pin name>          <tower2 pin name>}
set WSCs(UpdateWR)   {<tower1 pin name>          <tower2 pin name>}

set WSIs           {<tower1 STAM scan-in>       <tower2 STAM scan-in>}
set WSOs           {<tower1 STAM scan-out>      <tower2 STAM scan-out>}

set WPIs          {{<tower1 PTAM scan-ins>}    {<tower2 PTAM scan-ins>}}
set WPOs          {{<tower1 PTAM scan-outs>}   {<tower2 PTAM scan-outs>}}
```

The fifth section contains the JTAG configuration, if applicable. This includes the length of the JTAG instruction register and the opcodes of its instructions. In addition, it includes the pin names of the TAP port. Since only the bottom die can implement IEEE Std 1149.1, pads need to be inserted for its bottom-side I/Os. Therefore, the standard-cell names of the TAP port I/O pads are specified.

```
#######################################################
# JTAG configuration
#######################################################
set JTAG                    <YES/NO>
set JTAG_IR_LENGTH          <JTAG IR length>

set JTAG_EXTEST_OPCODE      <binary opcode>
set JTAG_CLAMP_OPCODE       <binary opcode>
```

```
set JTAG_HIGHZ_OPCODE        <binary opcode>
set JTAG_SAMPLE_OPCODE       <binary opcode>
set JTAG_PRELOAD_OPCODE      <binary opcode>
set JTAG_BYPASS_OPCODE       <binary opcode>
set JTAG_PROGRAM_WIR_OPCODE <binary opcode>
set JTAG_SCAN_OPCODE         <binary opcode>
set JTAG_SCAN_LOS_OPCODE     <binary opcode>

set TDI                      <JTAG scan-in pin name>
set TDO                      <JTAG scan-out pin name>
set TMS                      <JTAG test-mode select pin name>
set TCK                      <JTAG clock pin name>
set TRSTN                    <JTAG reset pin name>

set inpad                    <input pad library cell name>
set outpad                   <bidir pad library cell name>
```

Once the die configuration file and the netlist of the DfT-Ready design are available, the 3D-wrapper insertion flow can be executed to generate the 3D-wrapped design that complies with the extended 3D-DfT architecture. The main steps of the flow are shown in Figure 5.2. First, the I/Os of the wrapper module are created and the sharing logic between the functional and the test I/Os, as well as between the serial TAM and the parallel TAM I/Os is inserted. I/O sharing is explained in greater detail in Section 5.4. The next step involves the generation and insertion of the WIR and is further elaborated in Section 5.2. Subsequently, the width adapters are generated and inserted. This step is further elaborated in Section 5.3. Then, various multiplexers responsible for the test path re-configuration are inserted and connected. These multiplexers multiplex the test data and test instruction paths of the serial TAM, implement the serial/parallel bypass mode of the die and control the inclusion/exclusion of each tower in/from the test path. Finally, the die test interface is connected to the corresponding infrastructure of the 3D-wrapper. For instance, the parallel TAM of the die is connected to the respective width adapters. The resulting netlist enables testing the die in a 3D-setting.

5.1.3 Verification ATPG flow

The verification ATPG flow automates the generation of test patterns for the verification of the 3D-DfT. In general, the verification of the DfT of a module, may that be a standalone die or a stack of dies, is performed in two steps. First, test patterns for a specific test mode of the module are generated. Subsequently, the application of these test patterns is simulated and the actual responses are compared to the expected responses. Successfully passing verification, ensures that a module contains a continuous and properly functioning test path in the verified test mode. The verification ATPG flow enables verifying the 3D-DfT of standalone dies, as well as of partial and complete die stacks. Stack-level DfT verification is performed in addition to die-level DfT verification, because the DfT of several dies in the stack needs to cooperate to enable a stack test. For instance, one die might need to include another die in the test path, and elevate test stimuli to and test responses from that other die. Therefore, ATPG and simulation is also performed on partial and complete stacks in order to verify that the 3D-DfT cooperates properly across the stack.

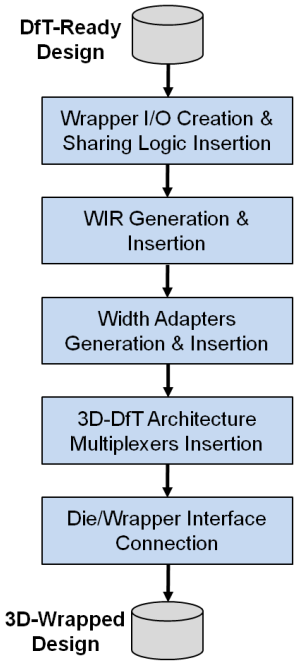


Figure 5.2: 3D-wrapper insertion flow.

First, the information regarding the die stack(s) and the test mode(s) need to be specified in a configuration file. The configuration file can be conceptually divided in three sections. In the following paragraphs, the contents of each section are outlined and a template for each section is also provided. The first section starts by specifying the various dies and the paths to their configuration files. The die configuration files are useful in order to avoid repeating information about each die, such as I/O names and WIR signals. Then, one or more instances of each die are defined. Finally, the die instance or die instances directly above a certain die instance are specified, essentially defining the stack structure. In the case of multiple towers, the order of the die instances should match the order that the towers are accessed in the test path. This order coincides with the order of the secondary ports in the die configuration file.

```
#########################################################
# Stack structure definition
#########################################################
set setup(<die name>)                    <die configuration file>

set inst_type(<die instance name>)       <die name>

set tower_dies(<die instance name>)      {<die instance name(s)>}
```

In the second section of the stack configuration file, one or more stacks can be defined. These can be complete, partial, or even single-die stacks. Only stacks, on which a test mode will be defined, need to be declared. So, a pre-bond test requires defining a single-

die stack, a mid-bond test requires declaring a partial stack, and a post-bond test requires defining the complete stack. The only information necessary to define a stack, are the die instances it contains. The information about its structure can be extracted from the structure of the complete stack that was specified in the first section of the stack configuration file. It has to be noted, that the first die instance in the list of instances of a stack, has to be the bottom die of that stack.

```
#######################################################
# Partial/Complete stack definition(s)
#######################################################
set stack_dies(<stack name>)            {<die instance name(s)>}
set stack_dies(<stack name>)            {<die instance name(s)>}
```

The third section of the stack configuration file contains definitions of test modes on the stacks defined in its second section. Multiple test modes can be defined per stack. First, a unique name has to be assigned to the test mode, and the stack, on which the test mode applies, has to be specified. Then, the test mode mnemonics (e.g. *parallel_intest* or *serial_extest*) of the dies that participate in the test have to be given. The dies that need to be in bypass mode do not need to be explicitly specified. No information about the inclusion/exclusion of towers in/from the test path needs to be included in the test mode mnemonics either. The test path required to enable a certain test mode of a die is calculated automatically. So are the necessary action sequences in order to activate that test path. If, for example, an Intest of a non-bottom die needs to be executed, all dies beneath it are automatically programmed to include the necessary towers in order to reach that die, and to be in bypass mode during test. Finally, the functionality of custom test signals is specified. Such signals can, for example, be programmed to behave as active high/low scan enable signals, or scan clocks. The entry points for these signals always have to on the bottom die of the stack, even though they might be propagated up the stack.

```
#######################################################
# Stack test mode(s) definition
#######################################################
namespace eval TM_<test mode name> {
    set stack                           <stack name>

    set test_mode(<die instance name>)  <test mode mnemonic>
    set test_mode(<die instance name>)  <test mode mnemonic>

    set signals_in(<pin name>)          <test function>
}
```

Once the netlists and the configuration files of the 3D-wrapped dies, and the stack configuration file are available, the verification ATPG flow can be executed. The main steps of the verification ATPG flow are shown in Figure 5.3. As part of the flow, the

netlists of the declared stacks are generated. These contain the wrapped dies as black-box modules in order to keep the stack netlist small in size and independent from any changes in the die netlists. Meanwhile, the files necessary to perform ATPG using Cadence Encounter Test are generated. First, the signal sequence for the initialization of the WIRs is generated. This step is further elaborated in Section 5.5. Next, the pinassign file that describes the functionality of the test pins (scan enable, scan-in, scan-out, scan clock, etc) of the bottom die during testing is generated. Finally, a script for Encounter Test is generated. This script instructs the ATPG tool to first apply the WIR initialization sequence and then assign the functionality specified in the pinassign file to the test pins. After that, Encounter Test is instructed to verify the DfT structures in the supplied design, analyze its potential faults, perform ATPG and write the test patterns to a file. The flow also generates a script for the application of the generated test patterns in NCVerilog. A successful test pattern generation and simulation ensures that the 3D-DfT works as expected in the simulated test mode.

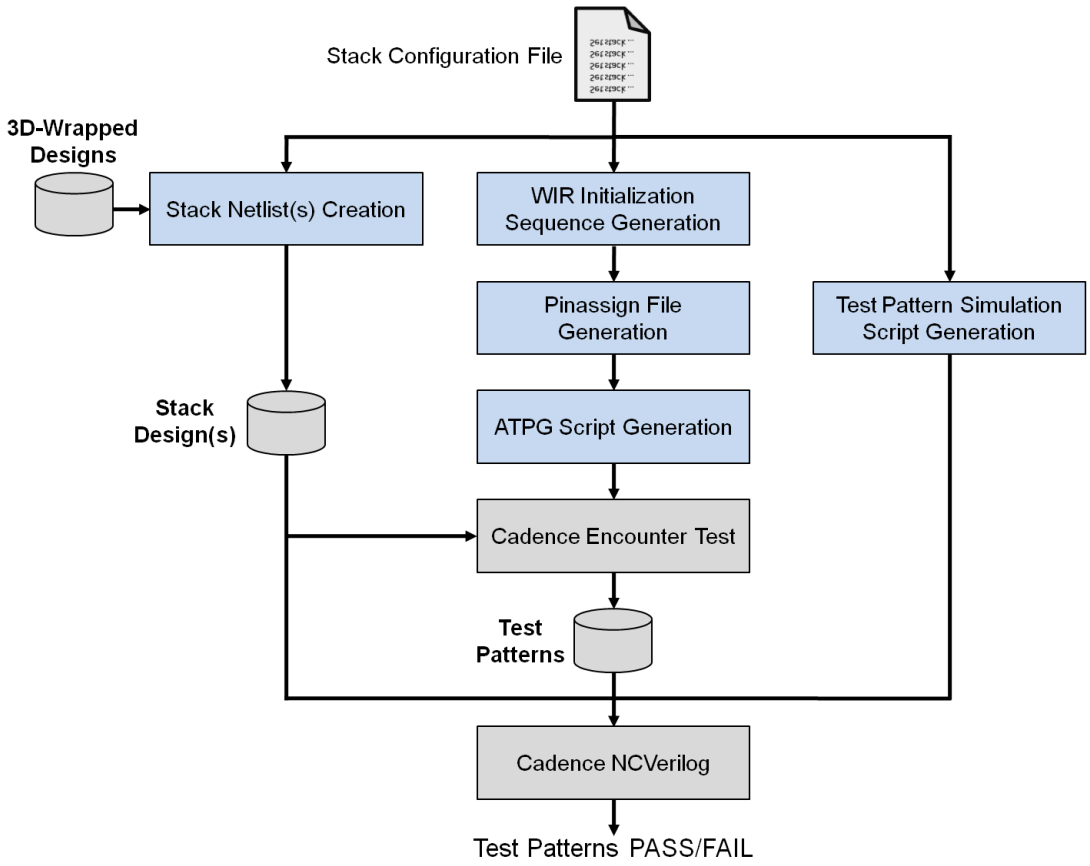


Figure 5.3: Verification ATPG flow.

The usefulness of the verification ATPG flow is currently limited to the verification of the 3D-DfT. However, in industrial flows, ATPG produces patterns for production tests. A fundamental difference in such a flow is that the ATPG should be performed by the die manufacturer on a per-die basis. There are two reasons for this. First, modular ATPG

enables die manufacturers to hide the implementation details of protected IP blocks and only provide the stack manufacturer with a model of their die and the necessary test patterns. The test patterns concern Intest, while the model is limited to the functionality necessary in order to perform interconnect ATPG once the die becomes part of a stack. Second, achieving sufficient fault coverage is the die manufacturer's task. Therefore, he should perform ATPG on his die, and potentially modify the design to optimize the tests in terms of test time and fault coverage. The verification ATPG flow can be used as-is to assist die manufacturers in performing ATPG for their dies. However, some modifications to the flow are required in order to enable it to operate on pre-generated die-level patterns, so that it is of use to stack manufacturers. The modified ATPG flow for stack manufacturers is shown in Figure 5.4. As seen, the pre-generated die Intest patterns are migrated from the die boundary to the stack boundary. The only additional test patterns that are generated at the stack-level, are those for inter-die interconnect Extest. Pattern migration consists in the manipulation of existing patterns of a module so that they can be applied on the module when there is no direct access to its pins. An example of the migration of two test patterns of a module is shown in Figure 5.5. In the example, the pipeline flip-flops in front of and after the module-under-test are assumed to be always shifting. Essentially, the SI stream has been shifted one clock cycle back in time, and the SO stream one clock cycle forward in time, in order to compensate for the pipeline flip-flops.

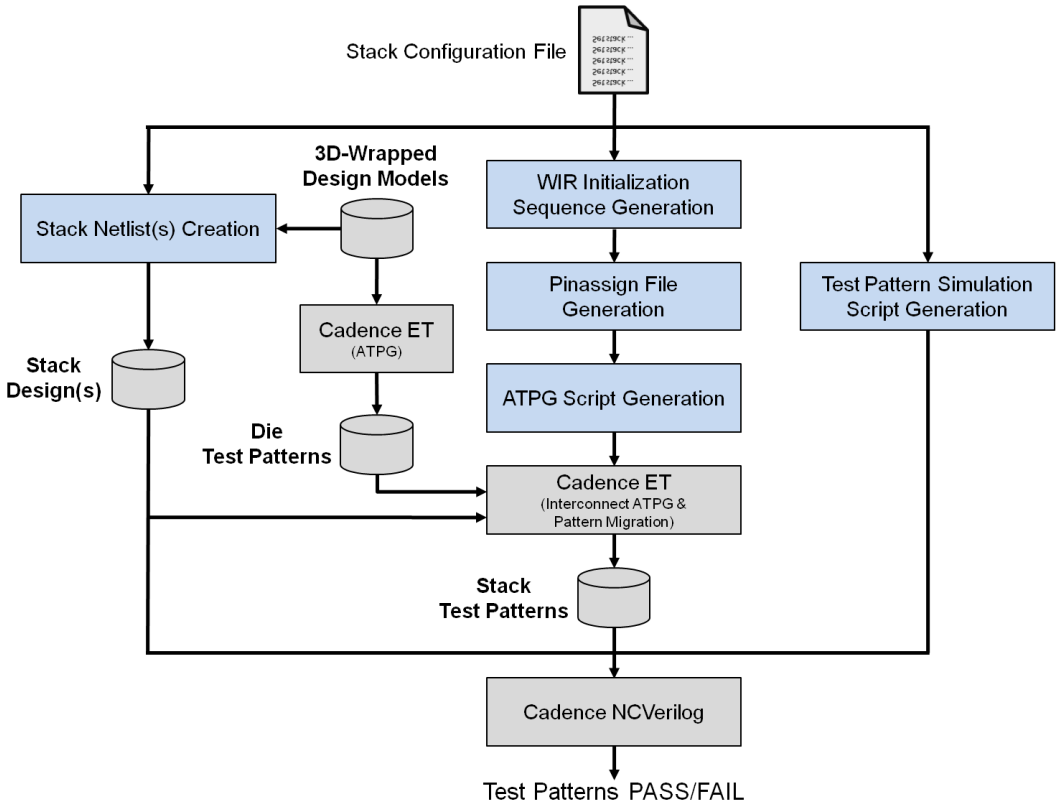


Figure 5.4: Modified ATPG flow for production testing.

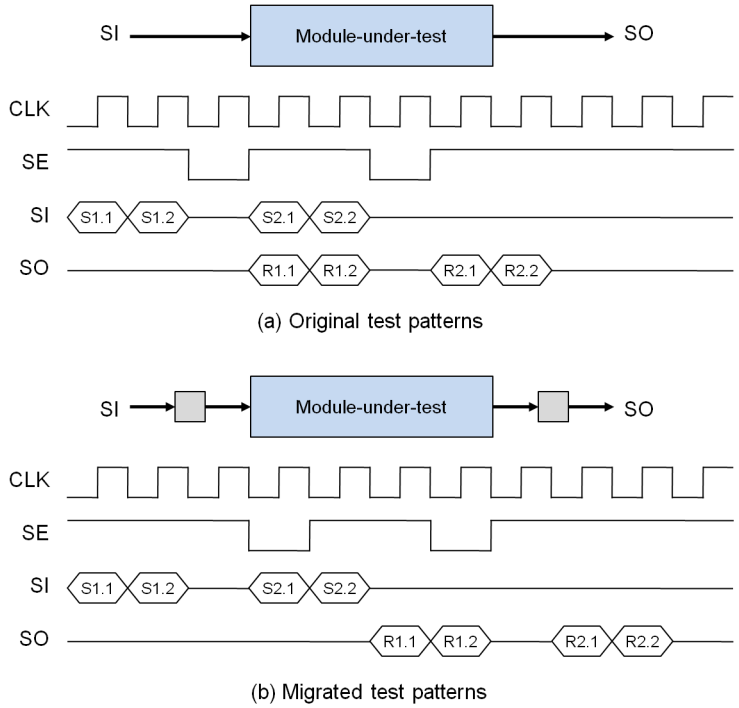


Figure 5.5: Pattern migration example.

5.2 WIR generation

The number of DfT structures on a die is constantly increasing. Furthermore, embedded cores and multiple towers are also introduced. All of the above need test control signals, most of which should come from the WIR in order to limit the number of dedicated test pins needed in a 3D-SIC. The increasing number of test control signals, makes it unrealistic to expect the user to create a custom WIR for each individual design. Instead, automatically generating the WIR based on a high-level description seems a logical choice.

The WIR has the structure shown in Figure 5.6. It is comprised of a shift register, some optional decoding logic and an update register. Programming the WIR involves two steps. First, a binary opcode is scanned into the shift register through the SI while keeping the ShiftWR and the SelectWIR signals asserted and issuing WRCK clock pulses. Then, the ShiftWR signal is de-asserted, the UpdateWR signal is asserted, and a WRCK clock pulse is issued, which causes the update register to capture the signals on the outputs of the decoding logic. These signals are then held constant, defining the test mode the wrapper will remain into, until a reset is issued or a new update operation is performed.

Two methods to automatically generate a WIR based on a high-level description were developed and implemented. The first method is for the generation of a *custom WIR* with arbitrary, user-defined binary opcodes. This method exploits the power of

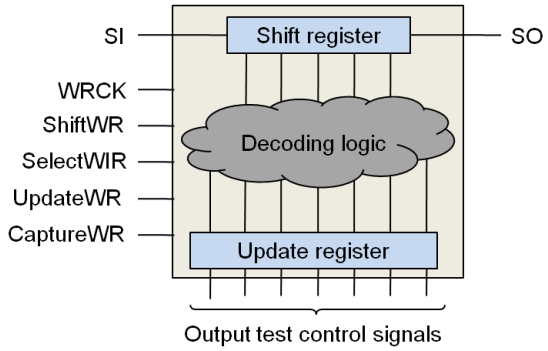


Figure 5.6: WIR structure.

the synthesis tools to generate a WIR with obscured relations between the WIR signals and the binary opcodes that control them. The user has to specify the WIR signals and assign a default/safe value to each one of them. Furthermore, he has to define mnemonic keywords that can be concatenated together to form test mode mnemonics indicative of whether the value of a signal should be 0 or 1 in that mode. Multi-bit signals can also be defined. In addition, the valid test modes to be implemented by the WIR have to be defined as well, and an arbitrary binary opcode has to be assigned to each one of them. The number of bits in an opcode equals the length of the shift register and can be less, equal, or greater than the number of WIR signals, which equals the length of the update register. An example custom WIR configuration is provided below.

```
array set wir_signal {
SERIAL_PARALLEL {{serial parallel}       0}
INT_EXT_WBY     {{intest extest bypass} 10}
ELEVATOR_TURN   {{elevator turn}         1}
INC_CORES       {{incCores}              0}
FULLSCAN_COMPR  {{fullscan compr}        0}
}

array set test_mode {
serial_bypass_elevator                  000
serial_extest_elevator                  001
serial_intest_incCores_fullscan_turn    010
parallel_bypass_elevator                011
parallel_extest_elevator                100
parallel_intest_incCores_fullscan_turn  101
parallel_intest_incCores_compr_turn     110
}
```

The WIR generation script parses each test mode and according to the mnemonic keywords that comprise it, the respective WIR signals are asserted or de-asserted. For example, for the *parallel_intest_incCores_fullscan_turn* mode, so when the opcode 101 is scanned-in, the SERIAL_PARALLEL signal is set to 1, the INT_EXT_WBY signal to 00, the ELEVATOR_TURN signal to 1, the INC_CORES signal to 1 and the

FULLSCAN_COMPR signal to 0. When no mnemonic keyword is specified for a signal in the test mode mnemonic, then the default/safe value is assigned to that signal in that test mode. As a result, when for instance the opcode 000 that corresponds to the *serial_bypass_elevator* mode is scanned-in, the SERIAL_PARALLEL signal is set to 0, the INT_EXT_WBY signal to 10, the ELEVATOR_TURN signal to 0, the INC_CORES signal to 0 and the FULLSCAN_COMPR signal to 0.

The above automatic WIR generation method allows obscuring the relation between loaded opcodes and WIR signals, thus enhancing the security characteristics of the wrapper. Moreover, programming of invalid test modes either intentionally or by mistake is prevented, as only the defined test modes are allowed and any other opcode activates the default/safe test mode. Another advantage of this technique is that the scan length of the WIR is proportional to the number of valid test modes rather than the number of WIR signals. For example, a WIR might have 20 output signals, but only 32 valid test modes. In that case, scanning-in $log_2(32) = 5$ bits is sufficient to program a specific test mode into the WIR. However, as the number of WIR signals increases, the number of test modes also rises, leading to two problems. First, the process of listing the test modes and assigning opcodes to them is becoming tedious and user-unfriendly. And second, the decoding logic starts becoming increasingly more complex, which takes its toll on area and synthesis time.

The second method generates a WIR, the shift and update registers of which are equally long, and the decoding logic of which is only comprised of wires. In such a WIR, there is 1-to-1 correspondence between binary opcodes and WIR signal values. This means that each opcode bit controls one WIR signal and thus the shift register is essentially copied into the update register upon update. This property enables a so-called *sliced WIR* design. One WIR slice containing one bit of the shift register and one bit of the update register is added for each WIR signal. The test modes defined by the user can be automatically translated to opcodes through binary enumeration, once the order of the various bits in the WIR is known. The configuration below describes a *1-to-1 WIR*.

```
set wir_signals {serial_parallel bypass_test extest_intest turn_elevator
excCores_incCores fullscan_compr}
```

Essentially, this structure is never defined by the user, as it can be deduced by the other configuration options. The existence of a parallel mode adds a serial/parallel signal. The bypass/test and Extest/Intest signals are mandatory. One include/exclude signal is added for each tower. Also, an include/exclude signal is included if cores exist. The remaining WIR signals are those included in the structure specifying custom on-die DfT control signals, and linked to a WIR bit. In order to show the correspondence to the first automatic WIR generation method, the same test modes used to demonstrate that method, are also presented as examples of this method. The *parallel_test_intest_incCores_fullscan_turn* mode corresponds to the opcode 111010, while the *serial_bypass_extest_elevator_excCores_fullscan* mode to the opcode 000100.

The automatic 1-to-1 WIR generation method is user-friendly, as only the bare min-

imum information regarding the WIR signals needs to be provided for its configuration. Furthermore, the absence of decoding logic maintains a low area and synthesis time overhead. On the other hand, the relation between WIR signals and opcodes can be worked-out relatively easily and there is no protection against programming an invalid test mode. Another disadvantage is that the WIR scan length might grow unnecessarily large even though the number of valid test modes might actually be a lot smaller.

Experiments were conducted in order to quantify the overhead of the custom WIR compared to the 1-to-1 WIR in terms of area and synthesis time. For that purpose, WIRs with 5, 6, 7, 8, 9, and 10 signals were generated using both methods. In the custom WIR method, different valid test mode densities were also taken into account for the evaluation. The valid test mode density is defined as the ratio of valid test modes over the number of potential test modes by exhaustive binary enumeration of the WIR signal values. Figure 5.7 plots the area of the WIR for an increasing number of WIR signals and for various valid mode densities. Figure 5.8 plots the synthesis time of the WIR for an increasing number of WIR signals and for various valid mode densities. It is clear from both plots, that while the 1-to-1 WIR scales linearly, the custom WIR scales exponentially. As a result, it might only approach the area and synthesis time of the 1-to-1 WIR for very low valid test mode densities. At the same time, the savings it might provide in WIR programming time are fairly small compared to the total test time for a realistically sized design. Consequently, despite the greater flexibility and the sophisticated features of the custom WIR, the 1-to-1 WIR is preferred.

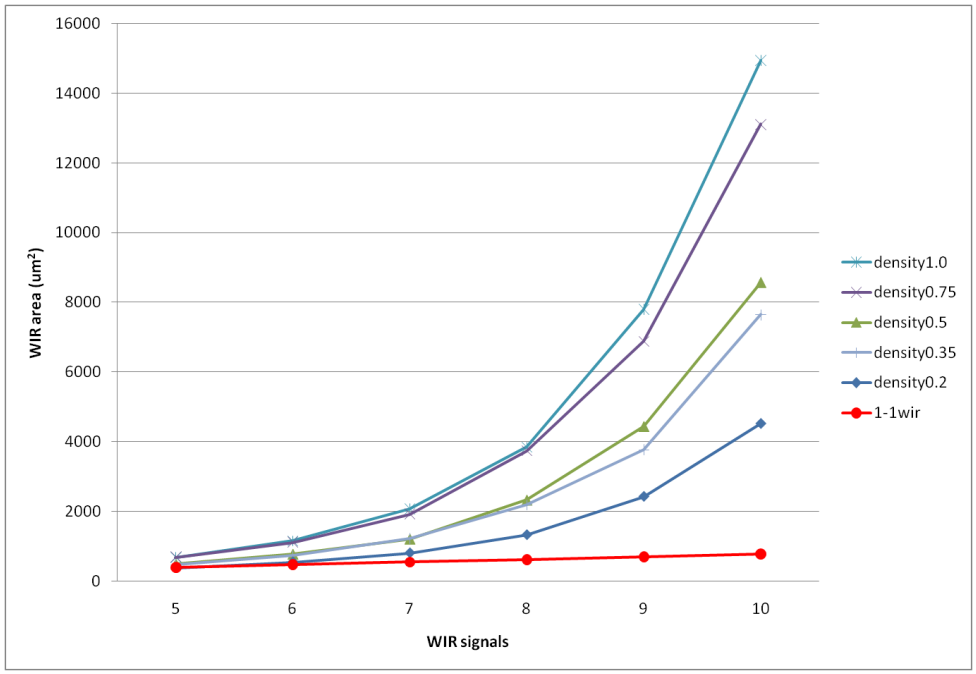


Figure 5.7: WIR area for 1-to-1 WIR and custom WIR with varying valid mode densities.

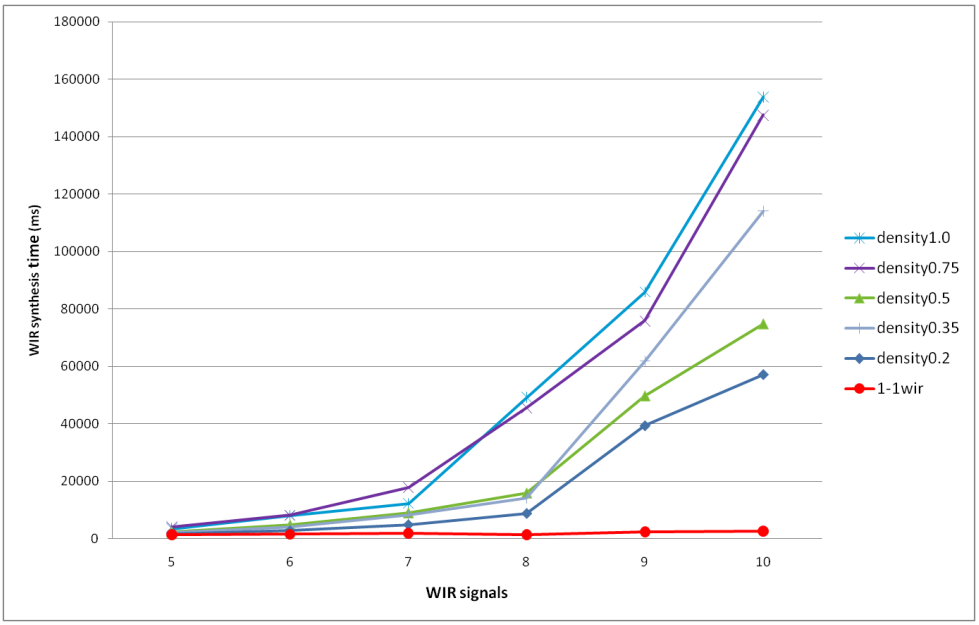


Figure 5.8: Synthesis time for 1-to-1 WIR and custom WIR with varying valid mode densities.

5.3 Width adapters

This section presents the implementation of the width adapters that are part of the extended 3D-DfT architecture and were introduced in Chapter 4. As described before, there are two width adapter pairs, both of which are fully parameterized and optimized in terms of area and included functionality. The first width adapter pair adapts between the stack PTAM width and the pads PTAM width. In order to keep the implementation relatively simple and not waste any test bandwidth on scanning-in dummy bits, a constraint put on the two widths is that the pads width should be chosen so that the ratio of the parallel width to the pads width is an integer number.

In total four cases exist for the input adapter, called pads_input_adapter, depending on the various TAM widths:

pads width = 0, parallel width = 0:
No input adapter is needed as no parallel TAM exists.

pads width = 0, parallel width $\neq$ 0:
The input adapter is comprised of wires connecting its parallel input to its parallel output, as no probe pads exist for the parallel TAM.

parallel width = pads width $\neq$ 0:
The input adapter is comprised of a multiplexer selecting between the two equally wide inputs (WPI/WPI_pad).

pads width $\neq$ 0, parallel width $\neq$ pads width:
The input adapter is a full-fledged deserializer, an example implementation of

which for pads width = 2 and parallel width = 4 is shown in Figure 5.9a. The parallel inputs are multiplexed to the deserialized pads inputs, while an update stage clocked on the negative clock edge increases the clock skew tolerance margin. The required update enable signal is generated by a clock control unit that will be elaborated later.

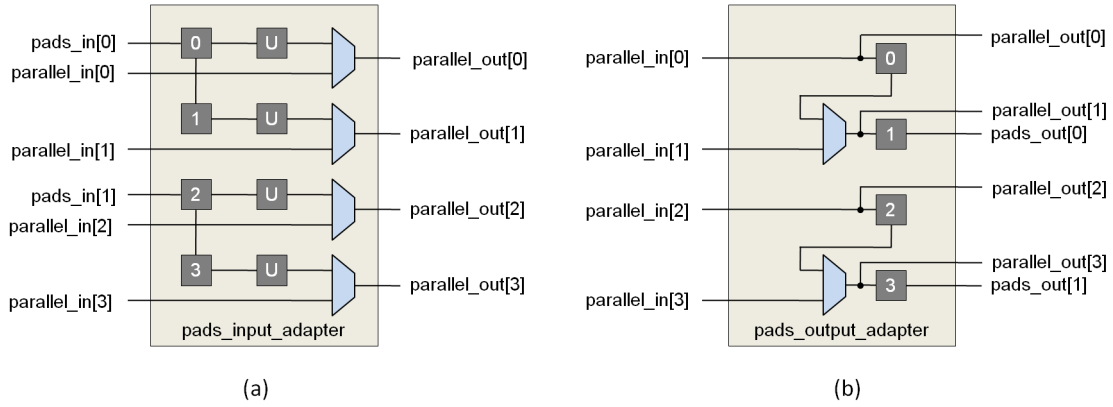


Figure 5.9: Pads width adapters between 2 and 4 (a) input adapter, (b) output adapter.

The corresponding output adapter is called pads_output_adapter and four cases exist for it depending on the various TAM widths:

pads width = 0, parallel width = 0:
No output adapter is needed as no parallel TAM exists.

pads width = 0, parallel width ≠ 0:
The input adapter is comprised of wires connecting its parallel input to its parallel output, as no probe pads exist for the parallel TAM.

parallel width = pads width ≠ 0:
The output adapter is comprised of wires connecting its parallel input to both of its equally wide outputs (WPO/WPO_pad).

pads width ≠ 0, parallel width ≠ pads width:
The output adapter contains a full-fledged serializer, an example implementation of which for pads width = 2 and parallel width = 4 is shown in Figure 5.9b. The parallel inputs are connected to the equally wide parallel outputs, while the pad outputs are fed by the outputs of the serializer outputs. A multiplexer selects between the parallel input that should be captured when loading the serializer, and the output of the previous flip-flop of the serializer that should be captured when the serializer shifts in order to serialize its contents. The required serializer load signal that determines when the serializer is loaded and when it shifts, is generated by a clock control unit that will be elaborated right next.

The clock control unit generates the update enable, the serializer load, as well as the modified clock that drives the internal scan chains. In order to determine when each

event should occur, the clock control unit contains a counter that keeps track of the elapsed clock cycles. The counter needs to be able to count from zero up to the ratio of the parallel width over the pads width, and only while the test data path is being scanned. The WRSTN signal is used to initialize and/or reset its state. The wrapper clock is modified in order to generate the scan clock of the internal scan chains connected between the deserializer and the serializer, as these need to only receive one clock pulse every (parallel width/pads width) clock pulses when the parallel mode is active and probe pads are used for test access. At all other times, the wrapper clock is passed through untouched. Figure 5.10(a) shows a typical waveform showing these functions for the width adapter pair of Figure 5.9.

The other width adapter pair adapts between the stack PTAM width and/or the STAM width, and the internal PTAM width. In order to keep the implementation relatively simple and not waste any test bandwidth on scanning-in dummy bits, a constraint put on the two widths is that the internal width should be chosen so that the ratio of the internal width to the external width is an integer number.

In total six cases exist for the input adapter, called internal_input_adapter, depending on the various TAM widths and the presence or not of a serial compression mode, which requires bridging the STAM and the internal PTAM:

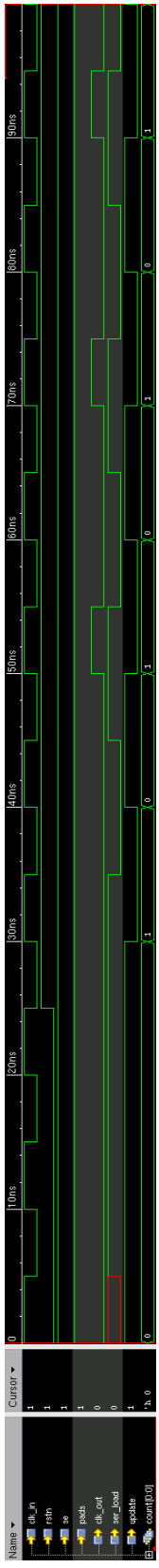
external width = internal width $\neq$ 0, no serial compression:
The input adapter is comprised of wires connecting its parallel input to its equally wide parallel output.

external width = internal width $\neq$ 0, serial compression:
The input adapter contains a deserializer that only deserializes from the serial TAM to the die-internal parallel TAM. Multiplexers selecting between connecting the deserializer outputs or the parallel inputs to the parallel outputs also exist, as the parallel inputs do not need deserialization.

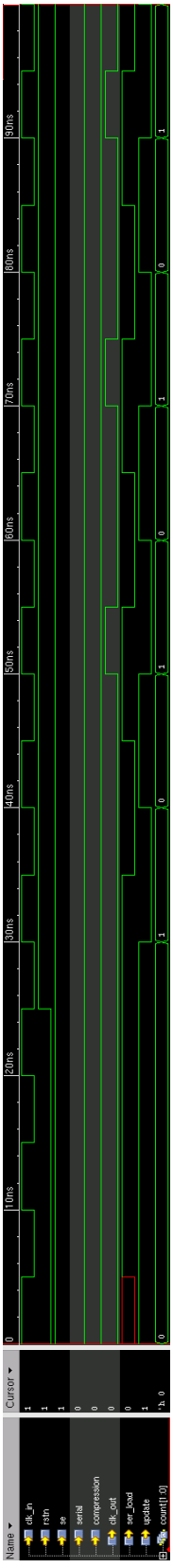
external width = 0, internal width $\neq$ 0, serial compression:
The input adapter contains a deserializer that only deserializes from the STAM to the internal PTAM.

external width $\neq$ internal width $\neq$ 0, no serial compression:
The input adapter contains a deserializer that only deserializes from the stack PTAM to the internal PTAM.

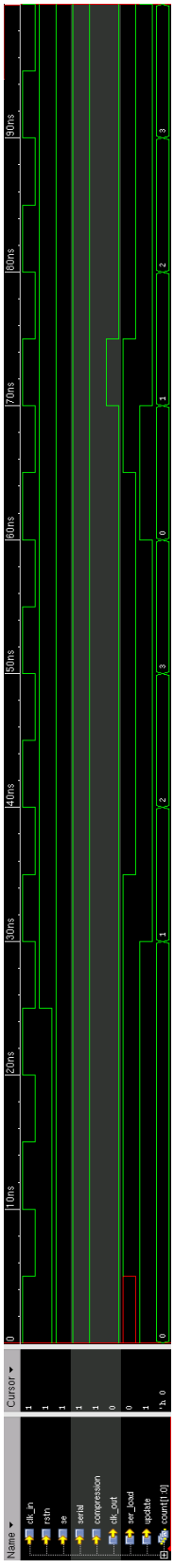
external width $\neq$ internal width $\neq$ 0, serial compression:
The input adapter contains a deserializer that deserializes both from the stack PTAM and the STAM to the internal PTAM. An example implementation for external width = 4, internal width = 8 and serial compression is shown in Figure 5.11a. Multiplexers are used to select between the parallel input and the output of the previous flip-flop of the deserializer. The parallel inputs are selected to implement a deserializer from the stack PTAM width to the internal PTAM width. Selecting their other input implements a deserializer from the STAM width to the internal PTAM width. An update stage clocked on the negative clock edge increases



(a) For the pads width adapters of Figure 5.9.



(b) For the internal width adapters of Figure 5.6 (parallel mode).



(c) IFor the internal width adapters of Figure 5.6 (serial compression mode).

Figure 5.10: Clock control unit waveforms.

the clock skew tolerance margin. The required update enable signal is generated by a clock control unit that will be elaborated later.

All other cases:
No input adapter is needed as either no internal PTAM exists, or no stack PTAM and no serial compression exist.

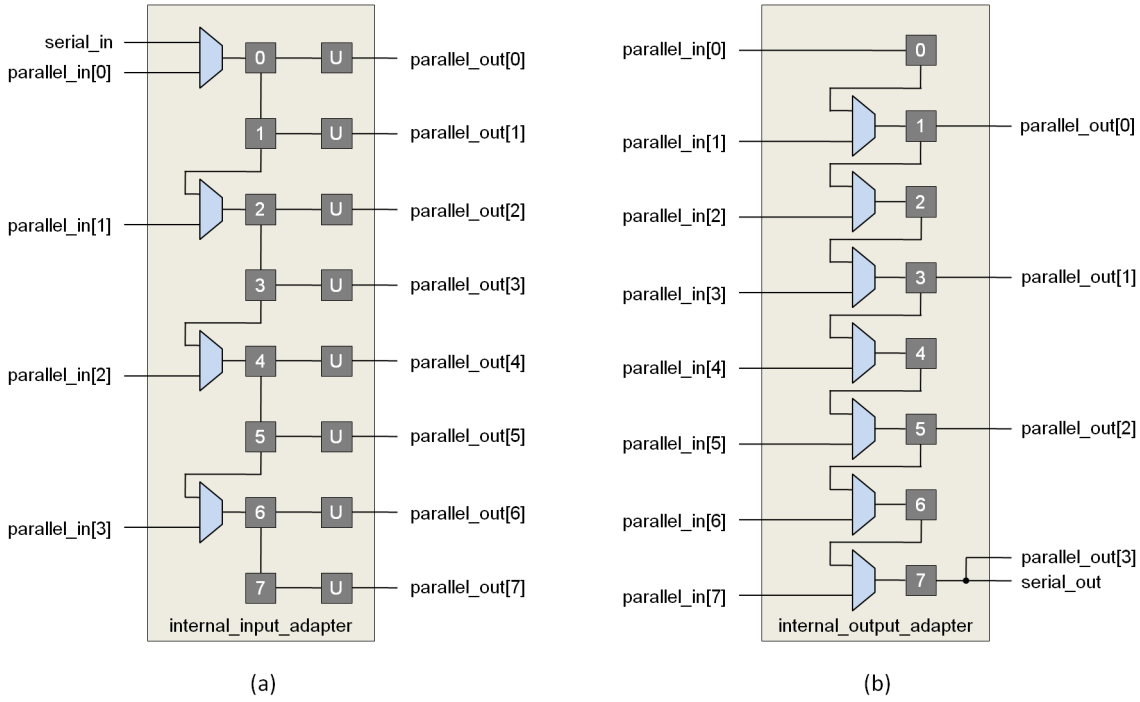


Figure 5.11: Internal width adapters between 1, 4 and 8 (a) input adapter, (b) output adapter.

The corresponding output adapter is called internal_output_adapter and six cases exist for it depending on the various TAM widths and the presence or not of a serial compression mode:

external width = internal width $\neq$ 0, no serial compression:
The output adapter is comprised of wires connecting its parallel inputs to its equally wide parallel outputs.

external width = internal width $\neq$ 0, serial compression:
The output adapter contains a serializer that only serializes from the internal PTAM to the STAM. Wires connecting its parallel inputs to its parallel outputs also exist, as the parallel inputs do not need serialization.

external width = 0, internal width $\neq$ 0, serial compression:
The output adapter contains a serializer that only serializes from the internal PTAM to the STAM.

external width ≠ internal width ≠ 0, no serial compression:
The output adapter contains a serializer that only serializes from the internal PTAM to the stack PTAM.

external width ≠ internal width ≠ 0, serial compression:
The output adapter contains a serializer that serializes from the internal PTAM to both the stack PTAM and the STAM. An example implementation for external width = 4, internal width = 8 and serial compression is shown in Figure 5.11b. Multiplexers select between the parallel input that should be captured when loading the serializer and the output of the previous flip-flop of the serializer that should be captured when the serializer shifts in order to serialize its contents. The required serializer load signal that determines when the serializer is loaded and when it shifts is generated by a clock control unit that will be elaborated right next.

All other cases:
No output adapter is needed as either no internal PTAM exists, or no stack PTAM and no serial compression exist.

The clock control unit generates the update enable, the serializer load, as well as the modified clock that drives the internal scan chains. In order to determine when each event should happen, the clock control unit contains a counter that keeps track of the elapsed clock cycles. The counter needs to count only while the test data path is being scanned. If no serial compression mode is implemented, the counter needs to be able to count from zero up to the ratio of the internal width over the external width, otherwise counting from zero up to the internal width is required. The WRSTN signal is used to initialize and/or reset the counter. The wrapper clock is modified in order to generate the scan clock of the internal scan chains connected between the deserializer and the serializer. These need to only receive one clock pulse every (internal width/external width) clock pulses when not in serial compression mode, and one clock pulse every internal width clock pulses when in parallel mode. At all other times, the wrapper clock is passed-through untouched. Figure 5.10(b) shows a typical waveform showing these functions for the width adapter pair of Figure 5.11 when in parallel mode. Figure 5.10(c) shows the same when in serial compression mode.

Finally, Figure 5.12 shows how these width adapters are cascaded to adapt between all different TAM widths present on a die. Red wires represent the clock and show how the various scan regions are assigned to different clock domains depending on whether they are between a SerDes pair or not. Blue wires carry the update enable and serializer load control signals from an adapter's clock control unit to the input and output adapter respectively. It has to be noted that the WRCK of the die directly above essentially is the clock output of the pads clock control unit, since the stack is traversed using the stack PTAM width rather than the pads PTAM width and thus modification of the timing is required.

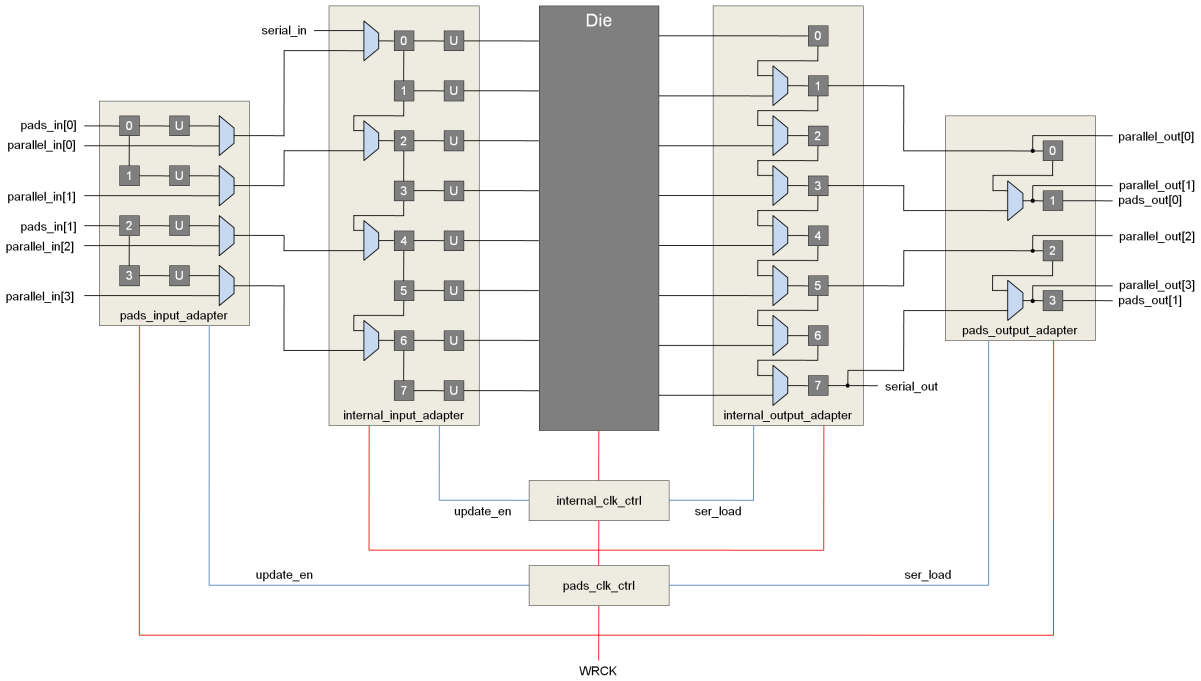


Figure 5.12: Width adapters and interconnection.

5.4 I/O sharing

I/Os, especially probe pads but also TSVs/micro-bumps to a smaller degree, are a relatively expensive area resource. The 3D-DfT architecture presented in Chapter 4 mandates a serial TAM and allows for a parallel TAM, both of which require some interface to the outside world. In order to minimize the number of extra pins that need to be added, sharing of the I/O resources is used. Sharing is achieved by simply reusing the same pin name during the configuration and is flexible in the sense that sharing is optional and can be used, or not used, at will. There are two types of sharing, sharing of functional and test I/Os, and sharing of serial and parallel TAM I/Os. In the case of inputs, the sharing is implemented by a simple fanout as shown in Figures 5.13a and 5.14a, while in the case of outputs, the two signals are multiplexed together as shown in Figures 5.13b and 5.14b. In the case of functional and test I/O sharing the multiplexer is controlled by the scan enable signal. Accordingly, in the case of serial and parallel TAM I/O sharing the multiplexer is controlled by the logical OR of the Serial and the SelectWIR signals to ensure that the serial TAM is selected for both serial test data and test instructions.

All of the above I/O sharing methods are optional and can be used or not used by simply providing dedicated or shared I/O pin names. Combinations are also allowed, for instance one single pin might be shared between a functional signal, a serial TAM signal and a parallel TAM signal by creating a one to three fanout in the case of an input and by cascading two multiplexers in the case of an output. This is particularly useful in cases where the I/O resources are very limited. For example, in a scenario where the probe pad parallel TAM width is just two bits wide, even one extra probe pad for the

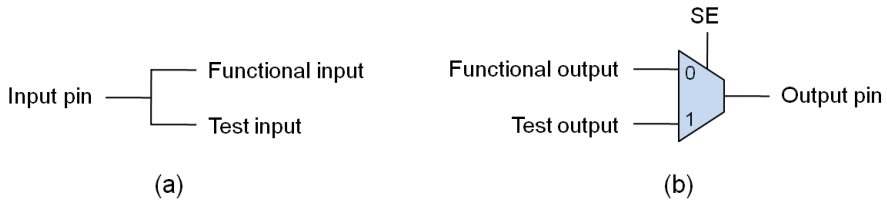


Figure 5.13: Functional and test I/O sharing (a) input, (b) output.

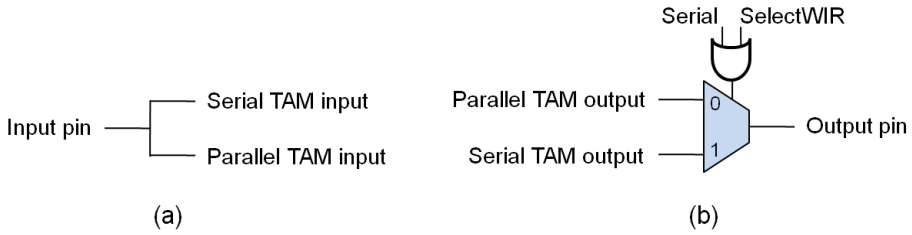


Figure 5.14: Serial and parallel TAM I/O sharing (a) input, (b) output.

serial TAM is a significant area overhead.

5.5 Incremental WIR configuration

In order to prevent uncontrolled growth of the WIR path's scan length, a mechanism for including only the necessary WIRs into the WIR path is required. To accomplish that, one additional WIR bit is introduced for each tower a die has. That WIR bit serves two purposes. First, it controls the elevator/turn multiplexers that include/exclude a tower in/from the test path. Second, it gates-off the UpdateWR signal of an excluded tower, so that its WIR state will not be corrupted while the other WIRs are programmed. The fact that one WIR contains a bit that determines whether another WIR is part of the WIR path or not, raises the need for an incremental WIR configuration mechanism. In order to always have a consistent WIR path, the WIR path has to be gradually opened, starting from a path containing just the bottom die's WIR. At each WIR programming step, one or more WIRs of one extra level in the stack can be programmed, until all WIRs that need to be programmed are reached.

In order to test one or more dies in the stack, other dies beneath them need to act as elevators in their bypass modes. The flow automatically determines which towers of each die need to be included in the test path and which dies need to be bypassed in order to complete the test path down to the bottom die. This leaves the user with the sole task of defining the test modes of the dies of interest. For example, if an Intest needs to be performed on die 4 of the stack depicted in Figure 5.15, dies 1 and 2 need to cooperate. To elaborate, die 1 needs to be bypassed and include die 2 in the test path. Additionally, die 2 needs to be bypassed as well and include die 4 in the test path. Generally, the DfT components of the various dies have to work together and according to each die's test mode to bring the whole stack in a consistent test mode. To accomplish

that, the information about the stack structure and connectivity, the die WIRs, and the test modes to be programmed needs to be combined. Starting from that information, a bit-stream that will bring the stack in the desired test mode has to be generated. This requires an algorithm involving a number of steps shown in Figure 5.17.

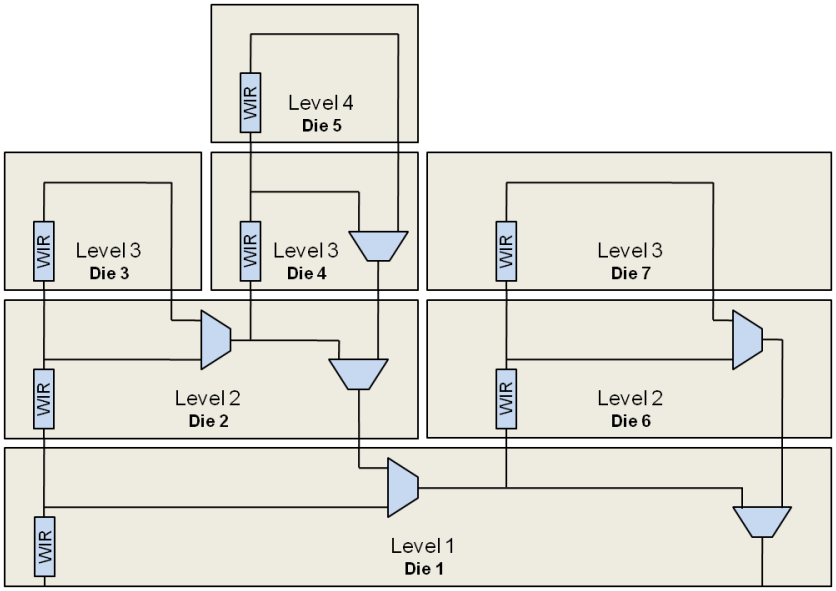


Figure 5.15: Multi-tower stack example.

First, the order of the WIRs of the various dies in the WIR path is determined. In the example of Figure 5.15, the numbering of the dies has been chosen such as to coincide with the WIR order (die 1, 2, 3, 4, 5, 6, 7). In general, the stack can be represented as a tree, where the bottom die is the root. Then, the visiting order of the dies during a depth-first search gives the WIR order. The next step of the algorithm determines which dies need to be included into the test and their test modes, unless already specified. If an Intest mode is specified for die 4, the trace to the root of the tree reveals that dies 1 and 2 are needed in order to complete the test path. Furthermore, since their modes are not specified, they are put in bypass mode to merely transfer test data and instructions up and down the stack. During the third step of the algorithm, the test paths are analyzed and the required number of incremental programming steps is calculated. In the example followed so far, the only test path is the one through dies 1, 2, and 4, which equals to a depth of three. Three programming steps are therefore required to reach die 4. The last two steps of the algorithm are executed iteratively for the calculated number of programming steps.

The first iterative step determines the WIRs that are included in the WIR path in the current programming step by a "longest-remaining-path-first" search. In the example stack of Figure 5.15, if the WIRs of dies 5 and 6 need to be programmed, two paths exist. One is going through dies 1, 2, 4, and 5, and the other one is going through dies 1 and 6. The sub-tree that contains the relevant dies is shown in step 3 of Figure 5.16. A "longest-remaining-path-first" search starts the path traversal from the longest path,

and iteratively increases the search depth by one level in each programming step. In each iteration, any already reached dies do not contribute to the depth of any path(s) involving them. This process is shown in steps 4 through 8 of Figure 5.16. A die of a path is only programmed, once the remaining path depth of the path it belongs to is greater than or equal to the remaining depth of any other path.

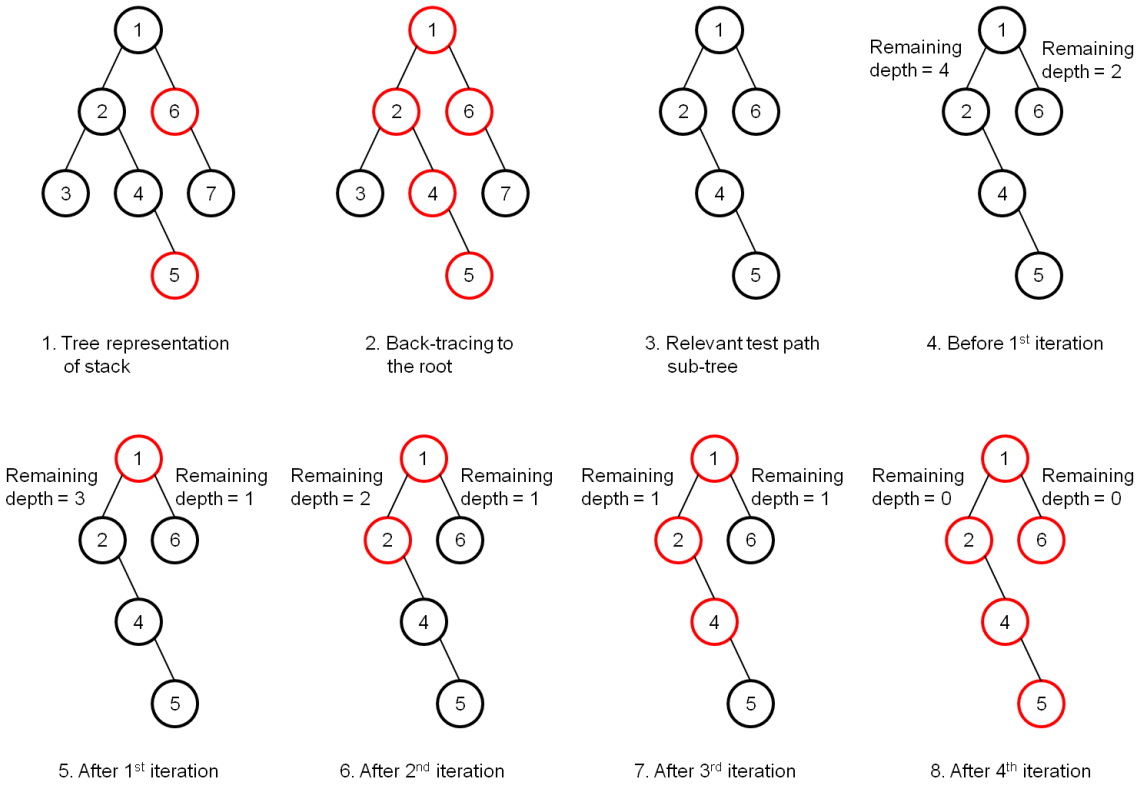


Figure 5.16: Traversal of stack-equivalent tree for incremental WIR configuration.

In the example given before, where an Intest on die 4 needs to be performed, the only test path is the one through dies 1, 2, and 4. So, only the WIR of die 1 is included during the first programming step, the WIR of dies 1 and 2 during the second one, and the WIRs of dies 1, 2, and 4 during the third and last step. The order of those WIRs is the same as their order in the full WIR path determined in the first step of the algorithm.

The second iterative step generates the bit-stream that needs to be scanned-in to appropriately configure the WIRs. The test modes determined in step two are first combined with the information about the WIRs to be included. This information is sufficient to set the bits of each die according to its internal test mode and to the towers that it should include in the test path. The corresponding opcode of each WIR is generated based on the information about the order of its WIR signals. The 1-to-1 correspondence between WIR opcode bits and WIR signals render this information sufficient. Finally, the opcodes of the various WIRs are concatenated according to the WIR order. An UpdateWR signal has to be issued in-between programming steps so that the WIR path is extended with the added WIRs.

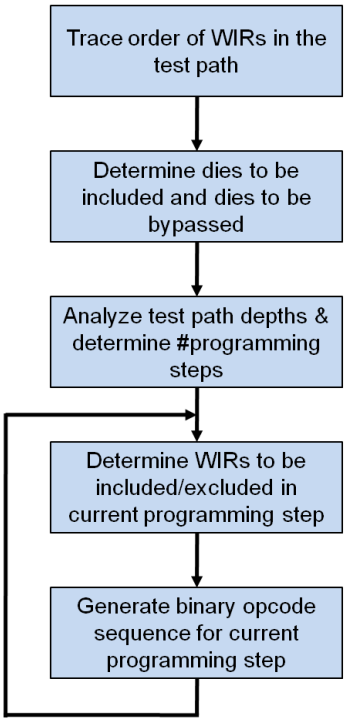


Figure 5.17: Incremental WIR configuration algorithm.

Figure 5.18 illustrates the incremental WIR configuration sequence needed to configure the stack into the test mode used before as an example. First, the WIR of die 1 needs to be programmed to include die 2 into the test path. Then, the WIR of die 1 needs to be identically re-programmed, while the WIR of die 2 is programmed to include die 4. Finally, the WIRs of dies 1 and 2 are identically re-programmed, while the WIR of die 4 is programmed with the desired test mode. The ShiftWR signal is de-asserted and the UpdateWR signal is asserted in-between, so that the WIRs can be updated and the WIR path can be extended accordingly. Once the full WIR configuration sequence has been scanned-in, the stack is in a certain test mode and the actual testing can start.

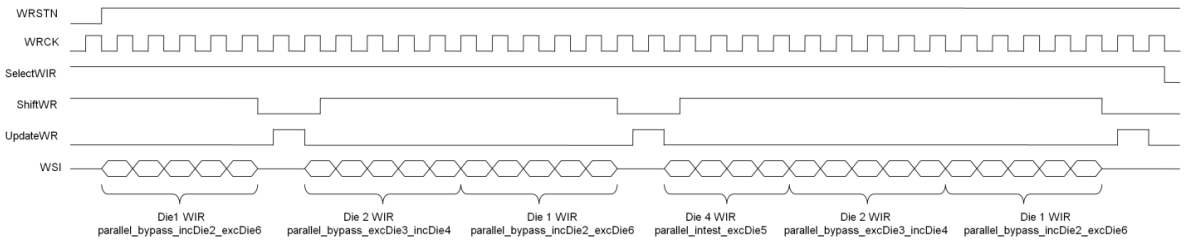


Figure 5.18: Example of incremental WIR configuration sequence.

5.6 Verification test case

The implemented components and automation flows were validated through application to a test case. The test case is comprised of three dies, organized in a stack with two towers. The designs that were used are presented in Table 5.1, and the stack they form is depicted in Figure 5.19.

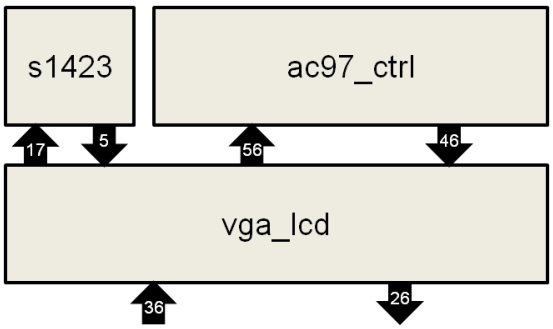


Figure 5.19: Test case stack-view.

Design	Inputs	Outputs	Std cells	Flip-flops
s1423	17	5	341	74
ac97_ctrl	56	46	19191	2302
vga_lcd	87	99	163051	17265

Table 5.1: Characteristics of test case designs.

s1423 is a small ISCAS'89 benchmark circuit [6] that was used as a toy-example. vga_lcd, which is the largest circuit the implemented flow was tested on, is a VGA LCD screen controller. Finally, ac97_ctrl, the size of which is smaller compared to vga_lcd, but still significant, is an AC'97 audio controller. It has to be noted, that only the test mode, and not the actual functionality of the stack, is of interest. Therefore, the dies were only matched in terms of I/O count, not of functionality. First, each die was pre-processed according to the sample flow provided in Figure 5.1. This inserted the on-die DfT, which includes scan chains for all three dies and TDC for the two larger designs. In addition, the WBR was inserted and the following modes were created for each die:

- Serial Intest Fullscan: Internal registers and the WBR in a serial test interface.
- Serial Extest Fullscan: The WBR in a serial test interface.
- Parallel Intest Fullscan: Internal registers and the WBR in a parallel test interface.
- Parallel Extest Fullscan: The WBR in a parallel test interface.

The two larger designs which contain TDC, also implement the following two modes:

- Parallel Intest Compression: Compressed internal registers and the WBR in a parallel test interface. The decompressor is a fanout network and the compressor is an XOR network.

- Parallel Intest Compression Decompression: Compressed internal registers and the WBR in a parallel test interface. Both the decompressor and the compressor are XOR networks.

A 3D-wrapper was then inserted around each DfT-Ready die by executing the flow of Figure 5.2. In the test case stack, the stack PTAM width was chosen to be equal to four. Table 5.2 lists the area overhead added by the 2D-DfT and the 3D-DfT to each die. The 2D-DfT area overhead is measured with the functional design as base, while the 3D-DfT area overhead is measured with the DfT-Ready design as base. The reason for that is that having 3D-DfT without 2D-DfT in a design does not make sense. From the table it can be concluded that the area overhead of the 3D-DfT is negligible and significantly smaller than that of the 2D-DfT for realistically-sized designs. The area consumed by the 3D-DfT is influenced by five components:

- A component that grows linearly with the WIR signal count. This is because the WIR shift register and the WIR update register need an additional flip-flop each, for each additional WIR signal.

- A component that grows linearly with the I/O count of the die. This is due to the fact that each I/O requires a wrapper cell with a fixed area cost, unless sharing of resources with a functional flip-flop is possible.

- A component that is proportional to the parallel TAM widths. The greater the TAM widths that need to be bridged by the adapters, the more area these modules consume.

- A component that grows linearly with the number of towers and the stack PTAM width. This is because each tower requires a dedicated secondary test port and a multiplexer that controls the inclusion/exclusion of the STAM and the stack PTAM of each tower in/from the test path.

- A component that is dependent on whether a serial compressed mode or embedded cores exist on the die, and which is attributed to the STAM multiplexers. These multiplexers multiplex the serial compressed and the serial uncompressed TAM, the serial test data and the test instruction path, the embedded core WIRs on the WIR path, etc.

Design	2D-DfT area overhead	3D-DfT area overhead
s1423	25.32%	27.68%
ac97_ctrl	10.87%	3.23%
vga_lcd	9.03%	0.69%

Table 5.2: Test case designs' DfT area overhead.

Finally, the inserted 3D-DfT was verified through the verification ATPG flow of Figure 5.3. All six possible stacks were created, one for each standalone die, one for each partial stack with either the s1423 or the ac97_ctrl missing, and one for the complete

stack. In order to perform exhaustive verification of the 3D-DfT, ATPG was performed on all possible test modes for each stack. Finally, the application of the generated test patterns was simulated in order to verify that the 3D-DfT works as expected both on a per-die basis and across the stack.

6 Conclusion

This thesis presented three aspects of an extended 3D-DfT architecture: (1) the definition and design of architecture extensions, (2) their implementation, and (3) the automation of the implementation in industrial EDA tools. The original IMEC 3D-DfT architecture was modified and extended to add compatibility with:

Test data compression (TDC): Adapters employing *deserializer/serializer pairs* (SerDes) were introduced to bridge the different test access mechanisms (TAMs). The organization of the adapters was such, that *mid-bond testing of a stack missing the bottom die* was made possible. Also, the DfT, including the wrapper boundary register (WBR), was re-organized in the hierarchy levels of the die and the wrapper, so that *abstraction from the on-die DfT* is accomplished.

Embedded-core wrappers: An interface compatible with the IEEE Std 1500 standardized test interface was added. Also, a mechanism for the *flexible inclusion/exclusion of core wrapper instruction registers* (WIRs) in/from the test path was introduced to prevent uncontrolled scan length growth. The support for embedded cores was also *closely coordinated with the support for TDC*, as many cores implement TDC themselves.

Multi-tower 3D-SICs: The 3D-DfT architecture was extended to support a *scalable number of towers* directly above a die. Therefore it was augmented with multiple secondary test ports and the problems of *test path reconfiguration*, and *test control and instructions* were solved for all towers.

A *3D-DfT insertion flow* was designed and implemented to enable the automated application of the extended 3D-DfT architecture to any given design using Cadence' EDA tools. The developed scripts allow (1) the *automated generation of SerDes* for width adaptation in the presence of TDC, and (2) the *automated generation of the WIR* for test control, based on a high-level description. Moreover, (3) *I/O sharing* between test and functional I/Os as well as between serial and parallel TAM I/Os is supported to save on area for I/Os.

A *verification ATPG flow* was also designed and implemented to facilitate the verification of the 3D-DfT per-die as well as across the stack. The flow automates the execution of ATPG at both the die-level and the stack-level, and the simulation of the application of the generated test patterns. In order to configure the test mode of the stack, an *optimized algorithm for the incremental configuration of the die WIRs in a stack* was designed and implemented. The flow can be extended with migration of die-level test patterns to the stack boundary to become suitable for production-testing ATPG.

The above architecture, hardware components, and automation flows were *verified through a test case* with three dies organized in a two-tower stack. First, the 3D-DfT

insertion flow was applied on each die. Then, ATPG and test pattern simulation was performed with the help of the verification ATPG flow on all possible test modes for the stack. This exhaustive verification proved the correct operation of the 3D-DfT on a per-die basis as well as across the stack.

Future work on the testing of 3D-SICs may address the following:

On-Product Clock Generation (OPCG): Test clocks can be generated by an OPCG module instead of being supplied by the ATE. This especially makes sense for at-speed test clocks, as it is exceedingly difficult and expensive to supply them through pins. OPCG can exploit the clocks generated by a phase-locked loop (PLL) module to generate launch-capture clock cycles on-chip, based on a trigger signal. Programming the OPCG module of a die requires providing test access to it through the stack.

Pattern Migration: As explained in Section 5.1.3, test patterns should be generated at the die-level in a production-testing ATPG flow for 3D-SICs. So, pattern migration from the die-level to the stack-level is required in order to enable the application of these test patterns to the die, once it is incorporated into a stack.

Bibliography

[1] *IEEE standard test access port and boundary-scan architecture, IEEE Std 1149.1-2001*, 2001.

[2] *IEEE standard testability method for embedded core-based integrated circuits, IEEE Std 1500-2005*, 2005.

[3] Rahul Agarwal, Wenqi Zhang, Paresh Limaye, Riet Labie, Biljana Dimcic, Alain Phommahaxay, and Philippe Soussan, *Cu/Sn microbumps interconnect for 3D TSV chip stacking*, Electronic Components and Technology Conference (ECTC), 2010 Proceedings 60th, 2010, pp. 858 –863.

[4] JEDEC Solid State Technology Association, *Wide I/O single data rate (JEDEC standard JESD229)*, 2011.

[5] Kaustav Banerjee, Shukri J. Souri, Pawan Kapur, and Krishna C. Saraswat, *3-D ICs: A novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration*, Proceedings of the IEEE, 2001, pp. 602–633.

[6] F. Brglez, D. Bryan, and K. Kozminski, *Combinational profiles of sequential benchmark circuits*, IEEE International Symposium on Circuits and Systems, 1989, pp. 1929–1934 vol.3.

[7] Michael L. Bushnell and Vishwani D. Agrawal, *Essentials of electronic testing for digital, memory & mixed-signal VLSI circuits*, Springer, 2000.

[8] M.H. Wu C.L. Hsu K.L. Luo L.C. Cheng C.A. Chen, Y.W. Chen and W.C. Wu, *A low-cost DFT architecture for 3D-SIC testing applications*, ETS'12, 2012.

[9] Shang-Yi Chiang, *Challenges of future silicon IC technology*, International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA), 2011, p. 1.

[10] Sergej Deutsch, Vivek Chickermane, Brion L. Keller, Subhasish Mukherjee, Mario H. Konijnenburg, Erik Jan Marinissen, and Sandeep Kumar Goel, *Automation of 3D-DfT insertion*, Asian Test Symposium'11, 2011, pp. 395–400.

[11] Brion L. Keller Vivek Chickermane Subhasish Mukherjee Erik Jan Marinissen, Sergej Deutsch and Navdeep Sood, *Interconnect test for Wide-IO memory-on-logic stacks*, Future Fab Intl. Issue 42, 2012.

[12] David Buck Etienne Racine and Steve Pateras, *DfT and test flows for stacked die*, Informal Proc. International Test Conference 3D-Test Workshop, 2011.

[13] S.-W. Li H.-C. Lin M.-J. Wang H. Chen, J.-Y. Shih and C.-N. Peng, *Electrical tests for three-dimensional ICs with TSVs*, Informal Proc. International Test Conference 3D-Test Workshop, 2010.

[14] Frank F. Hsu, Kenneth M. Butler, and Janak H. Patel, *A case study on the implementation of the Illinois Scan architecture*, ITC'01, 2001, pp. 538–547.

[15] Li Jiang, Qiang Xu, Krishnendu Chakrabarty, and T. M. Mak, *Layout-driven test-architecture design and optimization for 3D SoCs under pre-bond test-pin-count constraint*, ICCAD'09, 2009, pp. 191–196.

[16] Dean L. Lewis and Hsien-Hsin S. Lee, *A scanisland based design enabling prebond testability in die-stacked microprocessors*, ITC'07, 2007, pp. 1–8.

[17] Erik Jan Marinissen, Chun-Chuan Chi, Mario H. Konijnenburg, and Jouke Verbree, *A DfT architecture for 3D-SICs based on a standardizable die wrapper*, J. Electronic Testing (2012), 73–92.

[18] Erik Jan Marinissen, Chun-Chuan Chi, Jouke Verbree, and Mario H. Konijnenburg, *3D DfT architecture for pre-bond and post-bond testing*, 3DIC'10, 2010, pp. 1–8.

[19] Erik Jan Marinissen, Jouke Verbree, and Mario H. Konijnenburg, *A structured and scalable test access architecture for TSV-based 3D stacked ICs*, VTS'10, 2010, pp. 269–274.

[20] Erik Jan Marinissen and Yervant Zorian, *Testing 3D chips containing through-silicon vias*, ITC'09, 2009, pp. 1–11.

[21] Gordon E. Moore, *Cramming more components onto integrated circuits*, McGraw-Hill, 1965.

[22] Ken Smith, Peter Hanaway, Mike Jolley, Reed Gleason, Eric Strid, Tom Daenen, Luc Dupas, Bruno Knuts, Erik Jan Marinissen, and Marc Van Dievel, *Evaluation of TSV and micro-bump probing for wide I/O testing*, ITC'11, 2011, pp. 1–10.

[23] T. Waayers, R. Morren, and R. Grandi, *Definition of a robust modular SOC test architecture; resurrection of the single TAM daisy-chain*, ITC'05, 2005, pp. 10–619.

[24] Xiaoxia Wu, Yibo Chen, Krishnendu Chakrabarty, and Yuan Xie, *Test-access mechanism optimization for core-based three-dimensional SOCs*, ICCD'08, 2008, pp. 212–218.