Automated DfT Insertion and Test Generation for 3D-SICs with Embedded Cores and Multiple Towers

Christos Papameletis[1,3,4] christos@cadence.com
Brion Keller[2] kellerbl@cadence.com
Vivek Chickermane[2] vivekc@cadence.com
Erik Jan Marinissen[3*] erik.jan.marinissen@imec.be
Said Hamdioui[4] s.hamdioui@tudelft.nl

Cadence Design Systems
[1]Feldkirchen, Germany
[2]Endicott, NY, USA

IMEC
[3]Leuven, Belgium

TU Delft
[4]Delft, The Netherlands

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 micro-electronic structures, these 3D chips and their interconnects need to be tested for manufacturing defects. Previously, we defined, implemented, and automated a 3D-DfT (Design-for-Test) architecture that provides modular test access for 3D-SICs containing monolithic logic dies in a single-tower stack. However, the logic dies comprising a 3D-SIC typically are complex System-on-Chip (SoC) designs that include embedded intellectual property (IP) cores, wrapped for modular test. Also, multi-tower 3D-SICs have started to emerge. In this paper, our existing 3D-DfT architecture is extended with support for *wrapped embedded IP cores* and *multi-tower stacks* and its implementation is automated with industrial electronic design automation (EDA) tools.

1 Introduction

Three-dimensional IC stacking using TSVs and micro-bumps comes as a solution to the growing demand for higher performance and lower energy consumption in micro-electronic products. Stacking ICs on top of each other reduces the interconnect length, while the small dimensions of TSV-based interconnects offer high density, low latency, and low power dissipation compared to conventional inter-chip interconnects. The above, combined with the smaller footprint of 3D-SICs, enable the design of novel micro-electronic products.

As all micro-electronic products, 3D chips are susceptible to manufacturing defects and therefore need to be tested. Specifically in 3D integration, it is essential to enable multiple testing opportunities in order to increase the compound stack yield [1]. It should be possible to apply tests to individual dies (pre-bond), partial die stacks (mid-bond), as well as complete die stacks (post-bond).

These tests require DfT infrastructure that provides modular test access to all components by elevating test control, instructions, and data up and down through the stack. This paper extends an existing 3D-DfT architecture and automates its implementation using Cadence EDA tools. The extensions introduce support for wrapped embedded cores, as their use is common practice in modern designs. In addition, stacks branching-off into multiple towers are also supported, as such stack configurations start emerging.

The remainder of this paper is organized as follows. Section 2 briefly reviews prior work on 3D-DfT. Sections 3 and 4 describe the extended 3D-DfT architecture and its automated implementation respectively. A verification case study is presented in Section 5. Finally, Section 6 concludes this paper.

2 Prior Work on 3D-DfT

Several papers exist that deal with 3D-DfT and its optimization [2–8]. This paper is based on the 3D-DfT architecture originally proposed for logic-on-logic stacks by Marinissen et al. in [9–12]. In [13], this architecture was extended for memory-on-logic stacks, especially those containing JEDEC Wide-I/O Mobile DRAM [14]. The main component of the architecture is a die-level DfT wrapper, that can be based on either IEEE Std 1149.1 [15] or on IEEE Std 1500 [16]. Figure 1 shows a schematic view of the 3D-DfT architecture based on IEEE Std 1500.

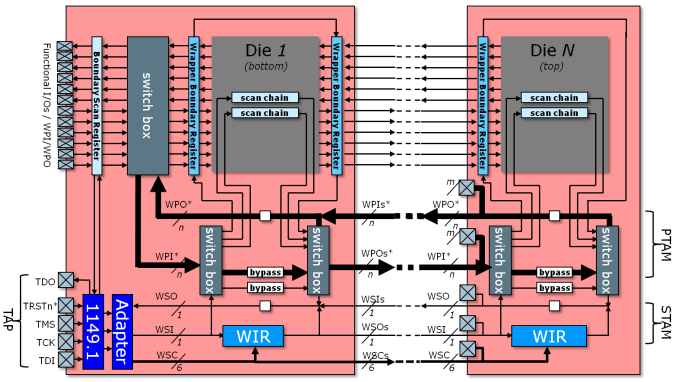


Figure 1: Schematic view of the original 3D-DfT architecture.

The architecture supports pre-bond testing through either probing the fine-pitch micro-bumps [17] or optional additional regular-pitch probe pads at each die. The architecture supports mid-bond, post-bond, and final package testing by allowing elevation of test control and data up and down through the stack from the stack's external I/Os, which are typically located in the bottom die of the stack. The architecture supports a modular test approach, in which dies and their inter-die interconnects can be tested separately. This allows for targeted test pattern generation and reuse, first-order fault diagnosis, and yield attribution. The architecture provides maximum freedom with respect to inclusion or exclusion of certain tests at a particular stage of the test flow and allows for flexible (re-)scheduling of those tests, in order to optimize the test flow

* Part of the work of Erik Jan Marinissen has been performed in the project ESiP, which is co-funded by the ENIAC Joint Undertaking (http://www.eniac.eu/).

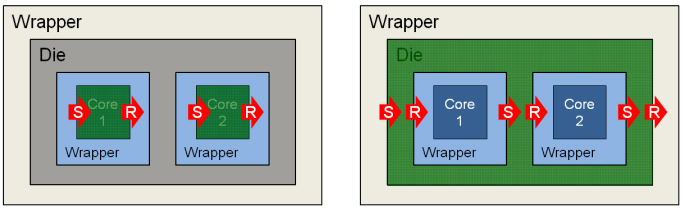
978-1-4673-6377-8/13/$31.00 ©2013 IEEE

and minimize the associated test costs. The implementation of this 3D-DfT architecture has been automated with Cadence RTL Compiler [12] and has been tried out on a TSMC test chip, showing a negligible implementation cost below 0.03% [13].

3 Extended 3D-DfT Architecture

3.1 Extensions for Embedded Cores

Embedded cores of a SoC are similar to dies of a 3D-SIC in the sense that both exploit the concept of modularity to enable the reuse of designs. In order to modularize testing, a 3D-wrapper is inserted around the die, just like an IEEE Std 1500 wrapper is inserted around each embedded core, as shown in Figure 2. Both wrappers support different modes, including Intest, Extest, and Bypass, and might have to work together to enable a certain test, as described in [18, 19] for hierarchical SoCs. Therefore, simultaneous access to the various hierarchical components in the die is a requirement on the test access mechanism (TAM) architecture. A test mode where this is evident is the die Intest, where the embedded core wrappers need to participate in their Extest modes (Figure 2(b)). The reason is that the cores need to provide stimuli (S) to the top-level and capture responses (R) from the top-level of the die.



(a) Cores Intest. (b) Die Intest requires cores' Extest.

Figure 2: Hierarchical die organization and test modes.

For the serial TAM, which combines the test data and test instructions paths, concatenation using a daisy-chain architecture is the only interconnection option, as it enables simultaneous test access, provides scalability, and is compatible with a single-bit interface. For the test instructions path, a mechanism that allows programming the core wrapper instructions registers (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/exclusion of core WIRs* in/from the test path as needed. Perhaps the easiest way to accomplish this is through the addition of two multiplexers, shown in brown color in Figure 3. One multiplexes the test data and test instructions 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.

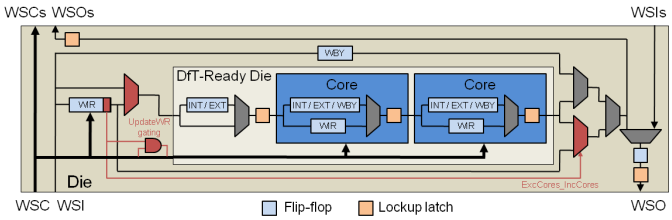
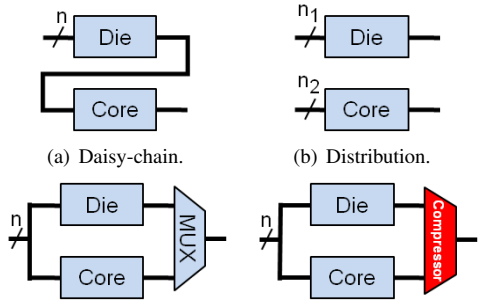


Figure 3: Embedded core serial TAM and test control.

Regarding test control, the wrapper serial control (WSC) is broadcast to the different cores. An additional provision that has to be taken is to 'freeze' the test state of the embedded cores when they do not participate in a test. For that purpose, 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 depicted in Figure 3. As a result, the test mode of the cores will only be updated if their WIRs are included in the test path, otherwise the bypass mode activated upon reset (WRSTN) is preserved. Also, this allows the embedded cores to 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. Finally, in order to ensure the integrity of scan data, lockup latches need to be inserted at the boundaries designating clock domain crossings. These operate on the inverse scan clock and therefore offer tolerance to clock skew of up to half a clock cycle.

Also for the parallel TAM, a consistent test data interface between the die top-level and the cores needs to be created, so that the embedded cores can be included into the test path. There are four potential interconnection topologies.



(a) Daisy-chain. (b) Distribution.

(c) Multiplexing. (d) Broadcast.

Figure 4: Interconnection topologies.

The first one (Figure 4(a)) is the daisy-chain architecture, where all components are concatenated next to each other in the test path. The second topology (Figure 4(b)) is the distribution architecture, where the available TAM width is distributed among the die top-level and each one of the cores ($n = n_1 + n_2$). The third interconnection topology (Figure 4(c)) is the multiplexing architecture, where the full TAM width is exclusively provided to one component at a time. Finally, the fourth topology (Figure 4(d)), is the broadcast architecture, where stimuli are broadcast to all components, and responses are compressed to match the available TAM width. From these, the multiplexing architecture is rejected as it does not support simultaneous access to multiple components, which is necessary in this case. The remaining three interconnec-

tion architectures can be used by the die designer, who thus has a large design space available that allows him to flexibly configure one or more die Intest modes. To support all different DfT solutions, abstraction from the internal organization of the die is required, so the starting point of this work is a 'DfT-Ready' die that should be testable in a 2D context (Figure 3). The inclusion of the WBR in the 'DfT-Ready' die, increases the flexibility given to the die designer even further. It enables sharing of the wrapper boundary register (WBR) with functional registers, resulting in area savings. Moreover, it is a useful provision for the future support of test data compression (TDC).

3.2 Extensions for Multiple Towers

As 3D-SICs become larger and more complex, having the stack branch-off into multiple towers seems 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 challenges of *test path configuration, and test control and instructions* for them.

The challenge of test path configuration consists in elevating test data up and down in the various towers of the 3D-SIC. 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, as outlined in Figure 5. So, first of all, each die destined to be bonded with k dies directly above it, has to implement k 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 [20]. 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.

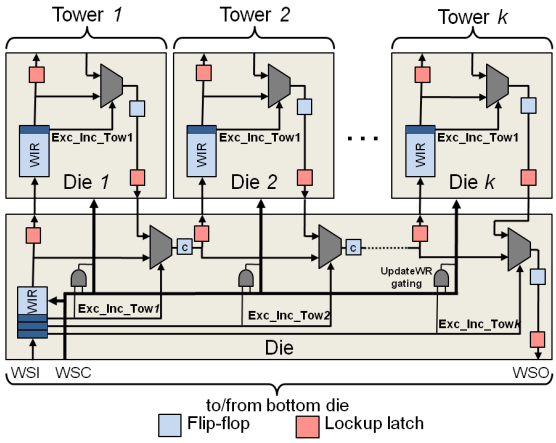


Figure 5: Detailed view of the WIR path of a multi-tower stack.

In addition, as with embedded cores, integrity of scan data needs to be ensured. Even though two dies might operate on the same clock, their inter-die connections and discrete clock trees introduce clock skew, essentially setting them in different clock domains. Therefore, lockup latches on the inverse scan clock are inserted at every inter-die scan output and a flip-flop is required to capture the scan data in the other die, as shown in Figure 5. Normally at least one flip-flop is connected to every scan input, with the exception of tower scan inputs, where a dedicated flip-flop (marked with 'c' in Figure 5) is added to provide a clean timing interface. That way, tolerance to inter-die clock skew of up to half a clock cycle is achieved.

Test control and instructions are an equally important part of multiple towers. In order to ensure the continuity of the WIR path in mid-bond testing and 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 5, 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. In that sense, wrapped embedded cores and multiple towers are handled somewhat similarly. Upon reset, the accessible path only includes the bottom die to ensure its continuity. 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.

4 Automated Implementation

4.1 Flows

Flows were designed and developed in order to automate the implementation of the extended 3D-DfT architecture and the generation of test patterns. The die manufacturer's flow (Figure 6) operates at the die level and specifically on the netlist of a 'DfT-Ready' die, which already contains the necessary die DfT. Such a die should be already possible to test in a 2D context. The execution of the flow is controlled by a die setup file, which contains information regarding the number and naming of I/Os, the number of towers, the TAMs, any user-defined DfT test control signals, and others. This flow first inserts the DfT, producing a 3D-wrapped die netlist and then performs automatic test pattern generation (ATPG) on that to generate die-level test patterns. It also annotates the die setup file with useful information regarding the inserted 3D-DfT, such as the WIR structure.

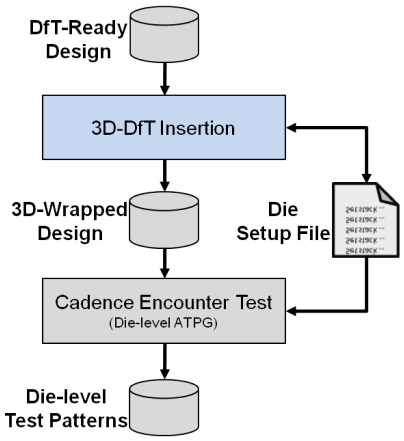


Figure 6: Die-level flow.

The 3D-wrapped die netlists, the die-level test patterns, and the die setup files of the various dies are inputs to the stack manufacturer's flow (Figure 7), along with a stack setup file. The stack setup file describes the structure of the stack, optional partial stack configurations for mid-bond testing, and test modes on each stack. A stack test mode is comprised of the test modes the targetted die or dies should be into for the test. Information about each individual die is retrieved directly from the corresponding die setup file, in order to make the flow more user-friendly and less error-prone. For instance, in order to generate the initialization sequence that configures the stack in a test mode, information about the stack structure and the test mode, included in the stack setup file, is combined with information about the structure of each die's WIR from the various die setup files. This flow first creates the stack netlists and the initialization sequences for the various test modes. Then, depending on each test mode, it migrates the pre-generated die test patterns to the stack boundary for intra-die Intest or performs interconnect ATPG for inter-die Extest. The output of the stack manufacturer's flow are test patterns ready to be applied to a 3D-SIC.

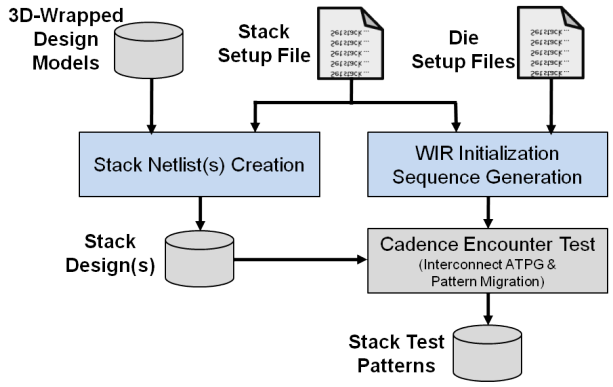


Figure 7: Stack-level flow.

Splitting the ATPG process in two phases, die-level and stack-level, introduces certain benefits. First, performing ATPG at the die-level enables IP protection in the not so unlikely scenario where the die manufacturer and the stack manufacturer are different companies. Second, in that same scenario, the task of achieving sufficient fault coverage lies with the die manufacturer, as it should, given that the stack manufacturer has no control over the DfT of the individual dies. Finally, the typically time-consuming Intest ATPG only has to be performed once for each test mode of each die, rather than for each test mode of the stack.

4.2 3D-DfT Insertion

The 3D-DfT insertion is performed through TCL scripts running on top of Cadence RTL Compiler (RC). These utilize both high-level DfT functions of RC, as well as low-level netlist manipulation functions for fully custom operations. One example of high-level DfT functionality is the multi-mode scan chain configuration mechanism incorporated into RC. This allows for the easy (re-)configuration of registers into different scan chains implementing multiple test modes. Another piece of high-level DfT functionality of RC is the wrapper cell insertion mechanism that intercepts I/Os and provides controlability and observability to them. One final high-level DfT feature is the automated insertion of boundary scan and JTAG. Fully custom operations such as the insertion and connection of a bypass register or of a multiplexor for the inclusion/exclusion of a tower in/from the test path are implemented through fundamental netlist manipulation functions.

An important addition to the automated DfT insertion is the automation of the WIR generation. The reason for that lies in the introduction of embedded cores and multiple towers. Both 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 natural choice.

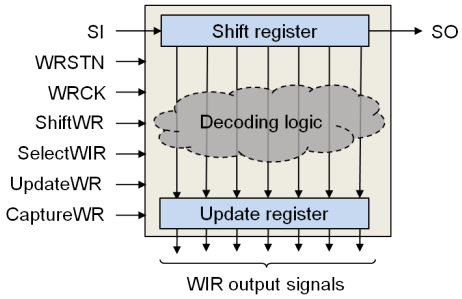


Figure 8: WIR structure.

The WIR is essentially comprised of a shift register, where instructions are scanned-in, an update register, which holds the output signals, and some decoding logic in-between, as shown in Figure 8. The purpose of the decoding logic is to decode instructions scanned into the shift register to output signals stored into the update register. The shift and update registers of the automatically generated WIR are equally long, and its decoding logic 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.

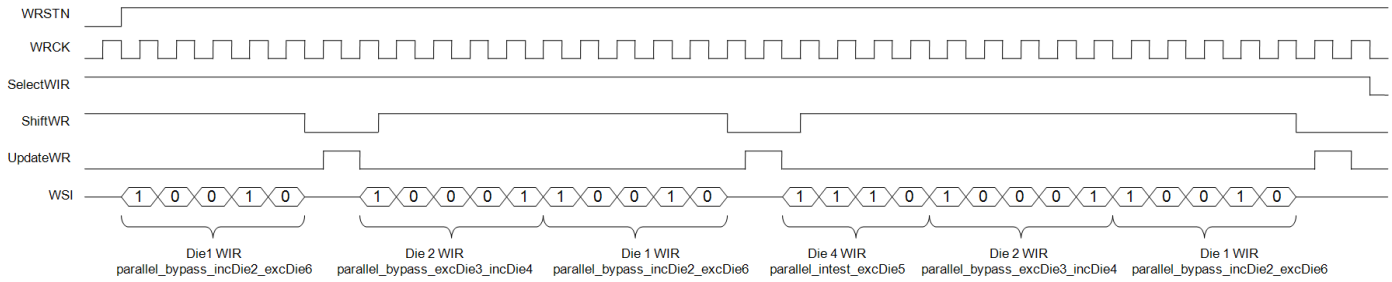


Figure 9: Example of incremental WIR programming sequence.

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.

Essentially, the WIR structure is *never explicitly defined* by the user, as it can be deduced by the other configuration options:

- The bypass/test and Extest/Intest signals are mandatory
- The existence of a parallel mode adds a serial/parallel signal
- One include/exclude signal is added for each defined tower
- An include/exclude signal is added if embedded cores exist

As only the bare minimum information regarding the WIR signals needs to be provided for its configuration, the automatic WIR generation method is user-friendly. Furthermore, the absence of decoding logic maintains a low area and synthesis time overhead.

4.3 Pattern Generation and Manipulation

In order to perform ATPG, the DfT of a die or stack first needs to be configured in the appropriate test mode. For that purpose, a WIR initialization sequence, that can be seen as a test preamble, is an integral part of both the die and the stack manufacturer's flow. As explained before, the WIR programming has to occur incrementally in order to *limit the path length* and *enable mid-bond testing*. 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. The flow automatically determines which towers of each die need to be included in the test path and which dies need to be bypassed or excluded 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 Dies 5 and 6 of the stack depicted in Figure 10, Dies 1, 2, and 4 need to cooperate. In general, 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. Based on those information, a bit-stream that will bring the stack in the desired test mode has to be generated.

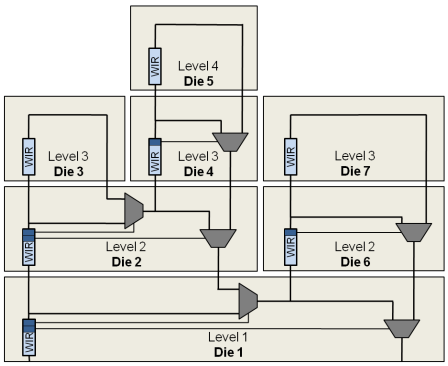


Figure 10: Multi-tower stack example.

An algorithm, optimized in terms of time, was developed and implemented to automate the generation of the WIR programming sequence. The function of the algorithm is illustrated below through an example. If an Intest on Dies 5 and 6 needs to be performed, the WIR of Die 1 is programmed during the first step, the WIRs of Dies 1 and 2 during the second step, the WIRs of Dies 1, 2, and 4 during the third programming step, and the WIRs of Dies 1, 2, 4, 5, and 6 during the last step. An UpdateWR signal has to be issued in-between programming steps so that the WIR path is extended with the added WIRs. An example of an incremental WIR programming sequence is given in Figure 9.

Once the stack has been configured in a consistent test mode, test patterns can be generated. Interconnect ATPG is used to generate stuck-driver and shorted-net test patterns for inter-die interconnects during Extest. Respectively, pattern migration is performed to translate intra-die Intest patterns from the die to the stack boundary for Intest. For that purpose, the stimuli and responses streams are 'padded' with additional bits to account for the pipeline delays introduced by the bypass registers.

5 Verification Test Case

The extended 3D-DfT architecture and the related automation flows were validated through application to a test case. The test case is comprised of three dies, organized in a stack with two tow-

Design	Inputs	Outputs	Std cells	Flip-flops	2D-DfT area	3D-DfT area
s400	3	6	62	21	+89.13%	+49.95%
s1196 (core)	14	14	271	18		
ac97_ctrl	56	46	19191	2302	+10.87%	+3.23%
vga_lcd	87	99	163051	17265	+9.03%	+0.69%

Table 1: Characteristics of test case designs.

ers. The designs that were used are presented in Table 1, and the stack they form is depicted in Figure 11.

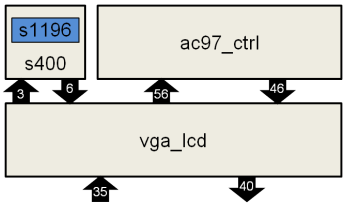


Figure 11: Test case stack-view.

`s400` and `s1196` are ISCAS'89 benchmark circuits [21] that were used to form a testbench for the verification of the embedded core support. For that purpose, `s1196` was wrapped according to IEEE Std 1500, and embedded into `s400` as a core. `vga_lcd` is a VGA LCD screen controller and `ac97_ctrl` is an AC'97 audio controller, both from the IWLS'05 benchmark set [22]. Note, 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.

A 3D-wrapper with a parallel TAM width equal to eight was automatically inserted around each die. Table 1 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 design containing the 2D-DfT as base. The reason for that is that 3D-DfT cannot exist in a design without 2D-DfT. 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.

Finally, the inserted 3D-DfT was verified by performing ATPG and simulating the application of the generated test patterns. All six possible stacks were created, one for each stand-alone die, one for each partial stack with either the `s400`/`s1196` or the `ac97_ctrl` missing, and one for the complete stack. Finally, all possible test modes for each stack were simulated to achieve exhaustive verification of the 3D-DfT.

6 Conclusion

This paper presented (1) the definition and design of 3D-DfT architecture extensions, and (2) their automated implementation in Cadence EDA tools. The 3D-DfT architecture that served as basis was modified and extended to add compatibility with wrapped embedded cores and multiple tower stacks. A mechanism was established for the flexible inclusion/exclusion of core/tower WIRs in/from the test path to prevent uncontrolled scan length growth, and the challenges of test path reconfiguration, and test control and instructions were resolved. A user-friendly technique for the automated generation of a WIR based on a high level description was implemented. Also, an optimized algorithm for the just-in-time programming of WIRs in a 3D-SIC based on the test modes of target dies was designed and implemented.

References

[1] Mottaqiallah Taouil et al. Test Cost Analysis for 3D Die-to-Wafer Stacking. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 435–441, December 2010.

[2] Dean L. Lewis and Hsien-Hsin S. Lee. A Scan-Island Based Design Enabling Prebond Testability in Die-Stacked Microprocessors. In *Proceedings IEEE International Test Conference (ITC)*, October 2007. Paper 21.2.

[3] Xiaoxia Wu et al. Test-Access Mechanism Optimization for Core-Based Three-Dimensional SOCs. In *Proceedings International Conference on Computer Design (ICCD)*, pages 212–218, October 2008.

[4] Li Jiang et al. Layout-Driven Test-Architecture Design and Optimization for 3D SoCs under Pre-Bond Test-Pin-Count Constraint. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 191–196, November 2009.

[5] Brandon Noia et al. Test-architecture optimization for TSV-based 3D stacked ICs. In *Proceedings IEEE European Test Symposium (ETS)*, pages 24–29, May 2010.

[6] Dean L. Lewis et al. Designing 3D test wrappers for pre-bond and post-bond test of 3D embedded cores. In *Proceedings International Conference on Computer Design (ICCD)*, pages 90–95, October 2011.

[7] Brandon Noia, Krishnendu Chakrabarty, and Erik Jan Marinissen. Optimization methods for post-bond testing of 3D stacked ICs. *Journal of Electronic Testing: Theory and Applications*, 28(1):103–120, February 2012.

[8] Amit Kumar et al. TSV and DFT cost aware circuit partitioning for 3D-SOCs. In *Proceedings International Symposium on Quality of Electronic Design (ISQED)*, pages 21–26, March 2012.

[9] Erik Jan Marinissen, Jouke Verbree, and Mario Konijnenburg. A Structured and Scalable Test Access Architecture for TSV-Based 3D Stacked ICs. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 269–274, April 2010.

[10] Erik Jan Marinissen et al. 3D DfT Architecture for Pre-Bond and Post-Bond Testing. In *Proceedings IEEE International Conference on 3D System Integration (3DIC)*, November 2010. Paper 3B.1.

[11] Erik Jan Marinissen et al. A DfT Architecture for 3D-SICs Based on a Standardizable Die Wrapper. *Journal of Electronic Testing: Theory and Applications*, 28(1), February 2012.

[12] Sergej Deutsch et al. Automation of 3D-DfT Insertion. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 395–400, November 2011.

[13] Sergej Deutsch et al. DfT Architecture and ATPG for Interconnect Tests of JEDEC Wide-I/O Memory-on-Logic Die Stacks. In *Proceedings IEEE International Test Conference (ITC)*, November 2012. Paper 12.4.

[14] JEDEC Solid State Technology Association. Wide I/O Single Data Rate, JEDEC Standard JESD229, December 2011.

[15] IEEE Computer Society. IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std 1149.1-2001, June 2001.

[16] IEEE Computer Society. IEEE Standard Testability Method for Embedded Core-based Integrated Circuits, IEEE Std 1500-2005, August 2005.

[17] Ken Smith et al. Evaluation of TSV and Micro-Bump Probing for Wide I/O Testing. In *Proceedings IEEE International Test Conference (ITC)*, September 2011. Paper 17.2.

[18] Anuja Sehgal et al. IEEE P1500-Compliant Test Wrapper Design for Hierarchical Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 1203–1212, Charlotte, NC, USA, October 2004.

[19] Sandeep Kumar Goel et al. Testing of SOCs with Hierarchical Cores: Common Fallacies, Test-Access Optimization, and Test Scheduling. *IEEE Transactions on Computers*, 58(3):409–423, March 2009.

[20] Chun-Chuan Chi et al. DfT Architecture for 3D-SICs with Multiple Towers. In *Proceedings IEEE European Test Symposium (ETS)*, pages 51–56, May 2011.

[21] Franc Brglez et al. Combinational profiles of sequential benchmark circuits. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*, volume 3, pages 1929–1934, May 1989.

[22] Christoph Albrecht. IWLS 2005 Benchmarks. In *Proceedings International Workshop on Logic Synthesis*, June 2005.