Implementing Design-for-Test within a Tile-Based Design Methodology - Challenges and Solutions

Venkat Yellapragada, Suresh Raman, Banadappa Shivaray
Xilinx
India

Ashok Anbalan
Mentor, A Siemens Business
India

Luc Romain, Benoit Nadeau-Dostie, JF Cote, Albert Au
Mentor, A Siemens Business
Canada

Giri Podichetty, Martin Keim
Mentor, A Siemens Business
USA

***Abstract*—A tile based design methodology consists of developing design blocks that are inserted in design layouts by placing blocks next to each other, making a tile-to-tile connection by abutting corresponding physical signal lines at the border of the tile. Very large systems can be easily and rapidly developed by seamlessly integrating tile elements in the layout. Further, the ease of top-level integration underlines the advantages over a bottom-up approach. However, this tile-based approach is incompatible with traditional DFT tools, which were created to work in accordance with the bottom-up design methodology. This paper outlines some of the obstacles to overcome, to support a truly tile-based DFT methodology. We describe here a working solution for a large production design, underlining a successful implementation of a tile-based Memory Test methodology.**

Keywords—tile-based design, bottom-up design, Memory BIST, Memory Repair, IJTAG, IEEE 1687

I. Introduction

A tile-based design methodology is not a new invention. It has been used in industry for many years, see e.g. [1][2]. However, there's no clear definition or description that is universally accepted. For example, in [1] a tile refers to an IEEE 1500 wrapped core. The authors of [1][3] show that such a core-based methodology has its advantages in particular with respect to logic test pattern generation. Similarly, we refer here to a tile as a design object, containing large quantities of logic, as well as embedded memories, both of which must be tested. However, the key difference that will become important here, is that the tiles we use have pass-through signals, allowing (jog-free) abutting of tiles in the layout, eliminating the need for any routing in the parent hierarchy level. Note that our tile-based notation here is driven from layout concepts; the physical location of a signal line (polygon) at the border of one tile matches up to the physical location of the very same signal line in the neighboring tile, and through this make a connection in the layout, without the need of a logical (and physical) port at the IOs of either tile. The latter usually implies some level of routing in the common parent instance, something that is not needed in our methodology.

Further differences are that each of our tiles do not have anchor logic near the ports, like an embedded TAP controller, IEEE 1500 WSP, or a pre-existing IEEE 1687 (IJTAG) host scan interface. Instead we expect the memory test insertion tool to add IEEE 1687 ports as needed, which we may also be used for other IJTAG compliant instruments in the tile. In addition, for the tile-based design methodology to work, all the used DFT tools must insert pass-through signal lines as needed.

The initial problem with the existing memory test tool is that it does not know about the tile-based design methodology. Instead, it follows a hierarchical, core-based design methodology. In this methodology, the DFT interface is inserted in each block, and expected to be connected only to a parent instance. An example is shown in Figure 1. This bottom-up methodology does not work in our design, as it leaves tiles unconnected, breaking the DFT signal connectivity. At the core of this paper, we describe how the used memory test tool can be taught to operate correctly for a tile-based design methodology, including the automation of insertion and connectivity of pass-through signals. The crux of the issue is automation, without which our DFT engineers would have to fix each and every tile connection manually, which is error-prone and time consuming.

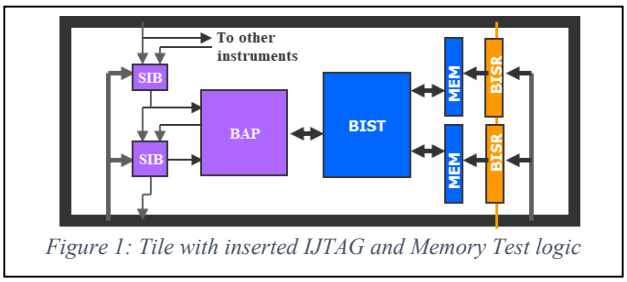

Figure 1: Tile with inserted IJTAG and Memory Test logic

In the next sections, we first review the typical hierarchical, core-based design methodology, and contrast it to the tile-based methodology we use for the production design, shortly outlined thereafter. However, let us first introduce the IJTAG components which form the backbone of the network connecting the memory test hardware components.

978-1-5386-5180-3/18/$31.00 ©2018 IEEE
DOI 10.1109/ITC-Asia.2018.00018

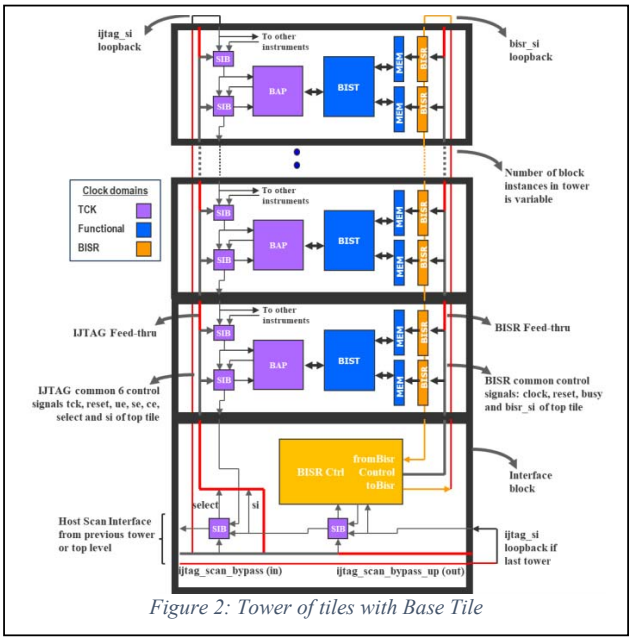

Figure 2: Tower of tiles with Base Tile

II. Design Methodology

A. IEEE 1687 – IJTAG

IEEE 1687-2014, aka IJTAG or internal JTAG, is a design and pattern reuse methodology. It expands the usability of IEEE 1149.1-2001 by adding features, that result in plug-and-play of instruments (IJTAG compliant IP), reduction of shift cycles by dynamically adjusting the JTAG scan network, and more importantly, the flexibility needed by todays designs. An overview and complete description of the standard can be found in [4] and [5] respectively.

In our DFT implementation, IJTAG is used as the backbone to connect all DFT hardware. Figure 1 shows the generic IJTAG and memory test infrastructure implemented in each tile. In this figure, as well as in the following figures, hardware within the test clock (TCK) domain is shown in purple, hardware in functional clock domains is shown in blue, and orange denotes the clock domain used for the implementation of memory repair (BISR and BISR Controller). The SIB, or Segment Insertion Bit, is a suggestion from the IJTAG standard. It implements a scan network switch, where multiple (TDI) inputs are multiplexed to one (TDO) output of the SIB, depending on the programming of a register within the SIB. The BAP, or BIST Access Port, is a component of the memory test implementation. It is a master controller for a multitude of memory BIST controller engines, connecting them to the IJTAG network. Each of the BIST controllers, in turn can test one or several memories, which may or may not have repair capabilities. If so, a BISR register needs to be inserted and connected. We make use of the internal BIRA (Built-In Redundancy-Analysis) engine that enables a fully integrated, and automated computation and application of the repair data.

At time of the insertion of the DFT hardware, both the memory test hardware, as well as the IJTAG infrastructure are added to our design. At the IOs of the tile, the insertion process adds the following IJTAG signals: *clock, scan_in, scan_out, capture_enable, shift_enable, update_enable* and *reset*. This group of signals is called the client scan-interface of the block module.

B. Bottom-up DFT Insertion Methodology

In a design that uses the bottom-up methodology, the parent design usually contains the TAP, memory repair controller and includes the IJTAG network host controllers that interact with the rest of the design including the child blocks. The parent design also includes a number of child block design instances, for which the layout has been signed-off or that are under development. Once all the design components are ready, the top-level design can then undergo back-end design steps, such as synthesis, place & route, timing analysis, etc. Although most of the complexity with respect to DFT is encapsulated inside the block instances, there is still some logic and interconnect in the parent level design that may cause problems when completing the final design phases. One of the major drawbacks of this design methodology is that, any modification in the top-level design requires that the back-end design steps and validation, be re-executed on the parent design. This may have adverse effects on the development schedule and risk to tape-out.

One aspect that makes the bottom-up design methodology different from the tiling design methodology is that the IJTAG and memory repair connectivity is handled exclusively inside the parent design. Once all child block instances are assembled in the parent design, each of them is connected to a dedicated host IJTAG scan-interface in the parent. Dedicated host scan-interfaces and IJTAG network connectivity must be planned in the parent design for the entire system. Consequently, the number of child blocks must be determined before the parent design can be implemented in order to prepare the IJTAG host scan-interfaces for each anticipated child block instance.

The placement of the different child blocks in the layout causes some IJTAG network paths to be longer than others. The propagation delay and skew between the IJTAG signals reaching the child blocks is dependent on the distance these signals have to travel. The child block with the largest path delay and skew between the IJTAG signals defines the maximum IJTAG frequency. However, this may be difficult to estimate, and is usually known only after the final layout has been completed.

C. Tile-Based DFT Methodology

Block tiling is a design methodology that consists of placing several layout blocks next to each other in a very specific way. The layout of these blocks ensures that key ports are strategically placed on the layout boundary such that they align when blocks are placed next to each other. These ports provides power, IJTAG, memory repair and functional connectivity from one block to the next. It eliminates the need to a top-level routing, connecting all tiles through the parent module.

A typical architecture for our design contains one base interface block and many tile blocks as illustrated in Figure 2. The interface block, shown at the bottom of the figure, contains common logic and is also responsible for hosting the IJTAG

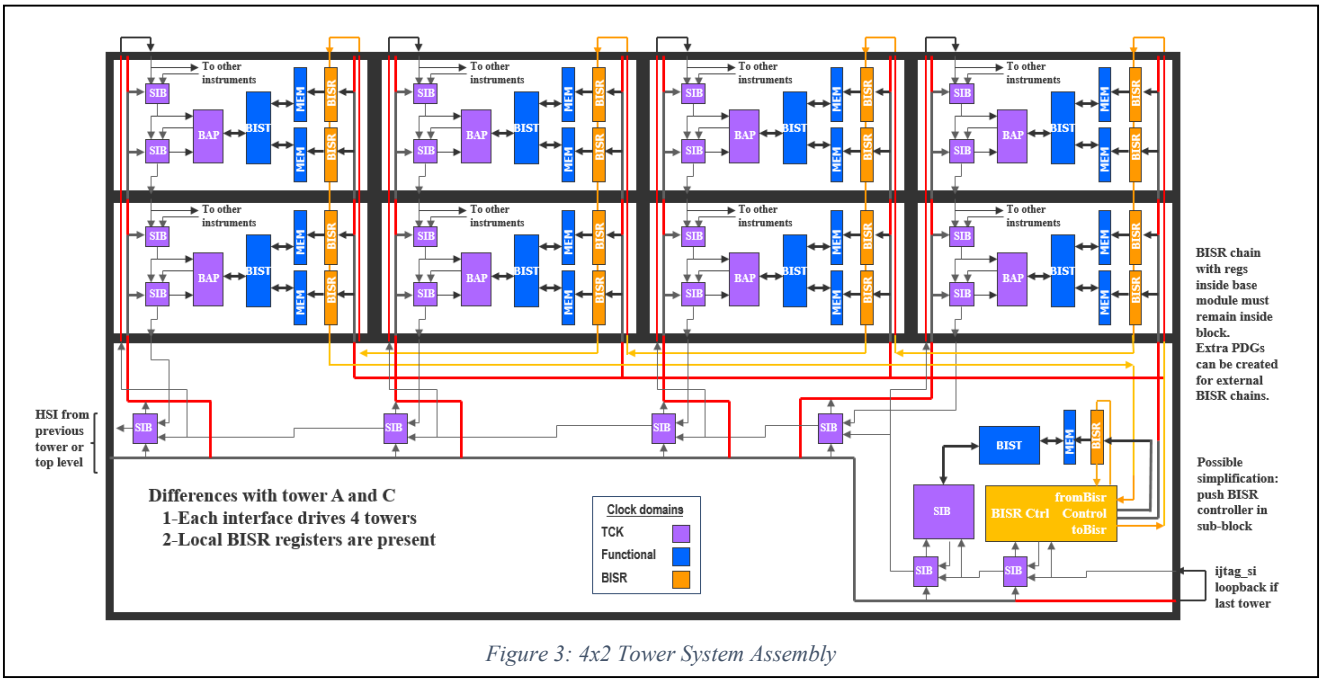

Figure 3: 4x2 Tower System Assembly

networks and the memory repair controller logic. The tile blocks contain functional logic which may be repeated a number of times depending on the specific application. The tile blocks are aligned with the interface block and stacked vertically and horizontally. An assembly composed of a interface block with tile blocks stacked vertically is referred to as a tower assembly. The interface block also has horizontal IJTAG pass-thru signals that allows stacking many tower assemblies horizontally, which is shown in Figure 3.

More tile blocks can be added vertically, simply by aligning the blocks on top of the previous ones. Note that the IOs of the base tile block does not change as the number of tiles in each tower increases. The last tile blocks on top of each tower must have loopback connections on the IJTAG and memory repair chains in order to close the chains.

The advantages of this method is that all blocks used in the tiling system can be developed, verified and signed off concurrently. The interface block can be developed regardless of the final height of the towers. The same block can be re-used in different designs with different tower heights without having to re-spin the interface block or the tile blocks.

Furthermore, different types of blocks can be added to the stack as long as layout is compatible with the tiling configuration. This implies that their corresponding power, IJTAG, memory repair and functional ports align with the other blocks in the tiling scheme. This allows modularity and flexibility when creating new designs.

D. Timing considerations

One of the challenges when developing large systems is to ensure proper timing between the regions of a chip. In a tile based design, the propagation of the DFT signals between the tiles must be designed so as to ensure proper timing is met across the stack. This is necessary in order to allow stacking of an arbitrary number of tile blocks.

Several clocking methodologies were considered for the IJTAG network [6]. However, they either involve rigid layout constraints or the addition of pipeline stages which is not compatible with IEEE Std 1149.1. The proposed solution is to instead propagate the DFT signals, including the clock, from the interface block all the way up to the top of the tower using source synchronous timing [7]. The propagation of the DFT signals is illustrated by red and orange lines in Figure 3. Ideally, the propagation delay of the signals should be balanced because the total skew between the clock and the other IJTAG signals determines the maximum stacking height for a given clock period. The maximum stacking height is determine by:

$$TilingHeight < \frac{TCK\_period}{2\ MaxSkew_{tile}} \qquad (1)$$

Where:
$MaxSkew_{tile}$ is the maximum skew between the IJTAG signals across the pass-thru path in one tile block.

Equation (1) shows that reducing the skew between IJTAG signals increases the number of tiles that can be stacked vertically. Increasing the IJTAG clock period can also be considered in order to increase the maximum stacking height. All the signals are routed through bypass paths simultaneously. This makes it easier to balance the propagation delays and achieve minimal skew between the signals.

The timing of the IJTAG network on the return path towards the interface block is only limited by the clock skew from one block to the next and the scan_out propagation delay. The maximum frequency of operation is determined by:

$$\frac{TCK\_period}{2} > T\_tck\_loop + T\_scan\_out\_prop \qquad (2)$$

Where:
T_tck_loop = IJTAG TCK clock loop timing between level i and i+1
T_scan_out_prop = IJTAG scan_out propagation delay from level i+1 to i

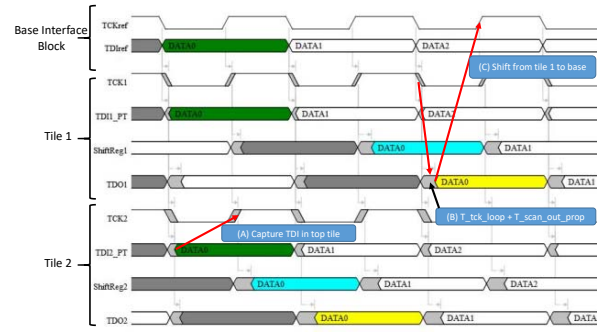

Figure 4 : IJTAG signals across tiles

The waveforms in Figure 4 illustrate IJTAG signals of a tower system that comprises an interface block and a tiling of height 2. For simplicity, only the IJTAG clock (*TCK*), scan_in pass-thru signals (*TDI#_PT*), IJTAG shift registers (*ShiftReg#*) and scan_out (*TDO#*) signals are illustrated, where # corresponds to the tile level 1 or 2. The IJTAG signals from the host interface in the base tile are propagated to the top of the tower via pass-thru signals *TDI#_TP* across the tiles. The *TDI2_PT* pass-thru signal is connected to the *TDI2* input at the top of the tower via a loopback connection; *TDI2* is captured by the *ShiftReg2*. The gray region on the edges of the *TCK#* and *TDI#_PT* signals illustrate the propagation delay and skew on the signals as they propagate thru the tiles. Limiting the skew on the signals during layout is important in order to maintain setup margin (A) when data from the port (*TDI2_PT*) at the top of the tower is captured in *ShiftReg2*. When shifting from *ShiftReg1* in tile 1 to the base tile (C), the delay (B) on the *TDO1* output (caused by the skew on *TCK1* plus the propagation delay from the base tile to the first tile) must be less than ½ TCK period in order to meet setup requirements as specified in equation (2). Since the interface block and the first tile are near, this requirement should be easy to meet.

The clock loop timing is constant between any two tiles and is not affected by the number of tiles. We can see that limiting the skew by balancing the propagation delays across the tiles and optimizing the clock loop delay has two benefits: it increases the maximum height of tiles that can be stacked and also increases the maximum clock speed that can be used on the IJTAG network without having to add pipelining on the control or serial data signals. Such pipelining would complicate the implementation unnecessarily.

III. Memory Test Insertion for Tiles

A. The Production Design

Figure 3 illustrates the principles of the production design for which we implemented the memory test solution. The tiling system shown here contains an interface block, which is host to a 4x2 tiling system.

The interface block is illustrated at the bottom of the figure and contains the host IJTAG and memory repair logic. Four towers of two tile blocks each are attached to the interface block. The red connections are new IJTAG and BISR connections required for tile-based designs. Thick black lines represent existing busses of control lines for IJTAG and BISR. Thin black lines represent the IJTAG serial path. Thin orange lines represent the BISR serial path. The IJTAG signals and memory repair signals are routed from the interface block to the top of the towers via the pass-thru paths. These signals enter each tile block and are propagated towards the top tile. The pass-thru signals for the IJTAG network extend the signals from the SIB modules in the interface block. The pass-thru signals for the memory repair chains include the following signals: *bisr_si, bisr_clock, bisr_scan_enable, bisr_reset, bisr_memory_disable, bisr_select and bisr_clear*.

As discussed above, the pass-thru connections provide a scalable method of connecting the IJTAG and memory repair network connections across tiles while solving timing issues.

Loopback connections are added at the top of the towers from the pass-thru output ports back into the scan_in ports as shown at the top of Figure 2 and Figure 3. The scan_in port, located at the top of the layout, is connected to the first DFT instruments on the IJTAG network inside the tile blocks. The scan_out port, located at the bottom of the tile, is driven by the scan-out port of the last instrument in the tile. The IJTAG network connects all instruments from the top tile and includes all other instruments on the way down towards the interface block.

B. Requirements

As indicated above, Figure 1 shows the memory test infrastructure initially implemented in each tile by the memory test tool. While the memory test hardware and its operation are correct, the needed pass-through signals are missing, rendering the solution unusable for our tile-based methodology. In other words, the blocks do have scan chains for their instruments, but they do not feed the IJTAG and memory repair control signals to output ports for the next tile above, which is required if this block needs to be tiled. Timing issues would also arise due to the long shift paths from the IJTAG host interface in the interface block and the top tile because of the skew and propagation delay between the clock and the other shift signals.

There are two requirements for blocks to be ready for tiling. First, all output signals from the host IJTAG interface and memory repair chains must be routed to output ports via pass-thru connections inside the tile blocks. This is illustrated by the thick red extension line near the top of Figure 5. Secondly, parallel paths feeding the scan_out signal from the lower tiles to

the upper tiles must be created. This is shown by the thin red vertical lines traversing the entire module in Figure 5.

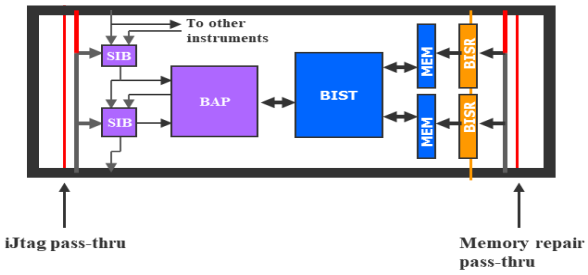

Figure 5: Tile block with pass-through connections

Ideally, the skew between the scan pass-thru path and the clock pass-thru should be minimized such as to allow the maximum IJTAG clock period across the stack and to maximize the total number of tiles that can be stacked in a system for a given clock period.

C. Solution Outline for the Tower Tiles

The DFT insertion in tile based designs can be performed using Tessent Shell [8]. Tessent Shell is a design-for-test environment within which you can perform all the tasks required to insert DFT hardware, generate manufacturing test patterns, and post-silicon tasks such as diagnosis and yield analysis. It is based on the Tcl language that allows the automation of tasks using scripting commands. All necessary netlist editing commands for implementing the pass through signals are natively available. But these basic netlist editing commands are insufficient for the automated solution we require. For this,

(a) the netlist editing script must become fully self-contained and automated, not requiring any user interaction or configuration; and

(b) must be executed as part of the normal DFT insertion process, without the user noticing it.

Only then, we would have an automated memory test insertion method that we could call compatible with our tile-based design methodology. To resolve (a) and (b), the connections of the IJTAG and memory repair pass-thru signals shown in Figure 5 must be automated. This can be accomplished through two parts.

For (a), Tessent Shell also provides a rich set of so-called introspection commands. Using these commands, the internal database containing all design, DFT, and setup data can be

```
# Post-insertion proc
# [start] : Post-insertion proc {{{
proc process_dft_specification.post_insertion {root_wrapper args} {
# [start] : Create pass-thru on bisr chain ports {{{
…
# [end]   : Create pass-thru on bisr chain ports }}}
# [start] : Intercept BISR controller segments {{{
…
# [end]   : Intercept BISR controller segments }}}
# [start] : Perform iJtag pass-thrus {{{
…
# [end]   : Perform iJtag pass-thrus }}}
}
# [end]   : Post-insertion proc }}}
…
# [start] : extract_icl.pre_write {{{
…
# [end]   : extract_icl.pre_write }}}
```

Create BISR pass-thru connections

Modify connections to/from BISR controller

Create IJTAG pass-thru connection

Extract ICL

Figure 6: Outline of a call-back procedure

questioned. For example, the name of the ports and signals of the inserted host scan interface can be requested. Using this requested data, additional netlist editing commands can be executed, for example generating the pass-through ports at the illustrated top of the tile and connecting these up to the already inserted IJTAG signals of the memory test solution.

For (b), we make use of a so-called call-back method. In essence, we automatically load the procedure script of (a) at the tools' start-up time, and instruct the tool to automatically execute the script every time a memory test insertion was performed. Through this, our procedure is called immediately after DFT IP insertion and performs the extra design edits before the design is saved. The execution of this netlist editing script is transparent to the user. Figure 6 shows the principles of the call-back procedure.

The step of ICL (Instrument Connectivity Language) extraction shown at the end of the call-back procedure, computes the IJTAG description of the tile, containing not only the just inserted memory test solution, IJTAG network, and pass-through connections, but also the IJTAG description of any instrument we may have in our design.

To further customize and enable the IJTAG-based DFT insertion flow, we declare an attribute of the local tile IJTAG port names as shown in Figure 7. These attributes are used during ICL extraction to describe the pass-thru paths across the tiles.

```
Attribute ijtag_logical_connection =
   "{ijtag_tck ijtag_tck_up}",
   "{ijtag_scan_bypass ijtag_scan_bypass_up}",
   …
   "{ijtag_ue ijtag_ue_up}",
   "{ijtag_sel ijtag_sel_up}";
```

Figure 7: IJTAG logical connection ICL attribute

D. Solution Outline for the Base Tile

The preparation of the interface block for the tiling flow is similar to the tower tile modules with only a few exceptions. When performing a standard DFT insertion on an interface block, SIBs must be added to the IJTAG network and memory repair chains must be prepared to accommodate a fixed number of towers. The total number of towers must be determined in advance but the total height of the towers does not need to be known. Furthermore, the memory repair solution anchored in the base tile can be implemented in a generic form, not tied to a particular tower height stacking. This is critically important, since this is one of the key advantages of the tile-based design flow.

The interface block has four SIB modules that will be used as the IJTAG hosts for each tower. These SIB modules are not yet associated with any client instruments and their host interface ports are left unconnected. Extra memory repair chains are reserved on the memory repair controller and will be connected to memory repair chains on the towers.

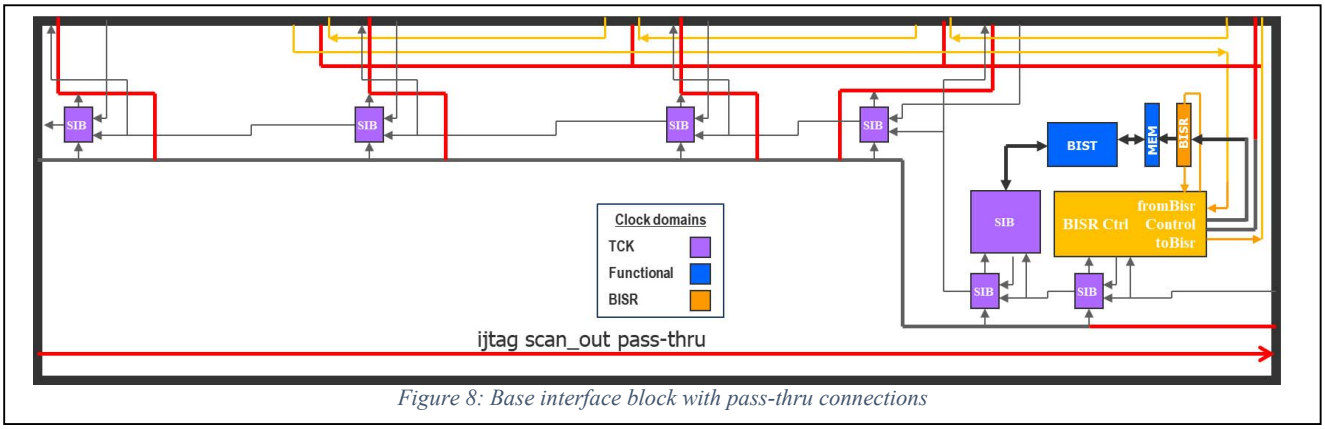

Figure 8: Base interface block with pass-thru connections

As for the tiles blocks, post DFT insertion steps must be executed in order to forward the IJTAG and memory repair signals to top-level ports. These ports will need to be positioned correctly on the layout in order to align them with the corresponding ports on the block layout. The main IJTAG network which comprises all SIBs and instruments in the interface block can be tiled horizontally. In order to do this, all IJTAG control ports must also be forwarded to output ports and a scan_out pass-thru connections must be added as shown in Figure 8.

Once the base and tile blocks completed, they can be tiled in a tower assembly. The IJTAG network and memory repair ports must be closed using loopback connections at the top of each tower. A final assembly example is shown in Figure 3.

Final verification of the assembly module involves performing an ICL extraction. Once the ICL is extracted for the assembly, patterns required to test all instruments within the assembly are generated and simulated.

A significant advantage of the DFT tiling approach is that the insertion of the tile blocks can be fully automated with scripting. All the design complexity is concentrated within the tile blocks while requiring only interconnections at the assembly level. This allows the DFT insertion to scale easily as new devices with different tile stacking requirements are needed.

IV. Summary

In the beginning of this paper we summarized the advantages of our tile-based design methodology. One of these advantages is the independent design of the different types of tower tiles, as well as base tiles, which serves as the anchor points for the towers. A DFT methodology applied to tiles has to solve the same requirements as for cores in a bottom-up design, namely a high quality test solution for all the logic and all the memories in the respective tile module. However, the integration of instances of these modules to form a design is quite different for tiles as for cores, and has in turn implications for the DFT solution for the tile: The DFT solution for a tile must understand the design feed through needed for other tiles it connects with, even though the DFT solution of the tile in question does not require any of these signals or ports. This requires a DFT solution which can understand a meta description of the DFT requirements of the tile.

In this paper we have shown how a classical DFT tool, which was designed for a core-based, bottom-up approach, can be extended to also operate correctly for our tile-based design methodology. The solution is centered on a fully self-contained call-back procedure, which uses design introspection and design editing commands. Using introspection, this call-back procedure was implemented fully generic so it is applicable to all our designs. It is automatically executed at the end of the memory test insertion step. From the user's perspective it appears as if the tool now natively supports our tile-based design methodology. A robust and simple clocking methodology used to operate the IJTAG network based on source synchronous timing was also introduced. Formulae allowing to estimate the maximum frequency of operation and the maximum number of tiles were derived.

References

[1] Yan Dong, Grady Giles, GuoLiang Li, Jeff Rearick, John Schulze, James Wingfield, Tim Wood, "Maximizing Scan Pin and Bandwidth Utilization with a Scan Routing Fabric," International Test Conference, Paper 3.2, 2017

[2] Anuja Sehgal, Jeff Fitzgerald, Jeff Rearick, "Test Cost Reduction for the AMDTM Athlon Processor using Test Partitioning," International Test Conference, Paper 1.3, 2007

[3] Yervant Zorian, Erik Jan Marinissen, Sujit Dey, "Testing Embedded Core-Based System Chips", IEEE Computer, 1999, Volume 32, Issue 6, pp. 52–60.

[4] IEEE Design & Test, 2013, Volume: 30, Issue: 5. Special issue on IJTAG

[5] IEEE Std 1687-2014, IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device

[6] E. G. Friedman, "Clock distribution networks in synchronous digital integrated circuits", *Proceedings of the IEEE,* vol. 89, no. 5, pp.665–692, 1999

[7] B. Anunay, A. Agarwal and P. Khandelwal (2016), Souce synchronous interface timing closure, *EDN Network. [Online].* Available https://www.edn.com/design/integrated-circuit-design/4442391/Source-synchronous-interface-timing-closure

[8] Tessent Shell Reference Manual, Available: https://documentation.mentor.com/en/docs/201711035/tshell_ref/html/manualtitle, [Accessed 16-Feb-2018]