Test-Model based Hierarchical DFT Synthesis

Sanjay Ramnath, Frederic Neuveux, Mokhtar Hirech and FelixNg
Synopsys Inc., Mountain View, CA 94043
{sramnath, fredn, hirech, felixng}@synopsys.com

Abstract

With increasing design sizes and adoption of System on a Chip (SoC) methodology, design synthesis and test automation tools are hitting ***capacity and performance*** *bottlenecks. Currently, hierarchical synthesis flows for large designs lack complete design-for-test (DFT) support. With this paper, we address a solution, involving the introduction of* ***test models*** *in a traditional DFT synthesis flow, that we term* ***Hierarchical DFT Synthesis (HDS)****. We discuss the use of* ***Core Test Language (CTL)*** *based test models combined with physical and timing models to provide a complete flow for* ***chip-level DFT****. In doing so we address some challenges the new flow presents such as Design Rule Checking (DRC), DFT architecting and optimization. We describe methods to overcome these challenges thereby presenting a new methodology to handle* ***complex next generation designs****.*

1.0 Introduction

Recent advances in manufacturing and methodology allow for larger and more complex IC designs. In today's environment, circuit sizes typically exceed million gates introducing further complexity to the design flow in terms of timing, placement and routing. This, coupled with the increasing relevance of the design-reuse paradigm suggests that capacity and performance will soon become major concerns with most DFT tools. In the past, DFT flows advocated a top-down approach to synthesize, optimize and insert test logic on flattened designs. Today, Design re-use and System on a Chip (SoC) methodologies [1] are driving the shift towards hierarchical flows, where pre-assembled blocks are integrated with control logic to form a complete system. These flows recommend a bottom-up approach to perform DFT synthesis on large hierarchical designs. Designers develop blocks concurrently, synthesizing them and implementing DFT at the front end of the design process. This enables predictability and facilitates optimization to minimize the impact of test logic on the design. Once all the blocks are complete and DFT ready, final assembly integrates them and addresses DFT at the chip-level. If each of these blocks is over a million gates large, then reading in the entire design and performing DFT insertion at the chip-level becomes impractical. In addition we also have to account for glue logic between these blocks which might be significant.

The concept of test modeling presents a solution to this problem. Test modeling refers to the abstraction of DFT structures embedded in a design, in the form of a test model. In other words, a test model encapsulates all DFT information needed by a system integrator. The proposed HDS flow uses a test model instead of a netlist representation of the sub-modules during chip-level integration. Thus, we realize a significant improvement in terms of both capacity and performance since the size of the abstract model is typically only a small fraction of the original netlist.

The use of test models instead of complete netlists presents us with a number of challenges, in terms of reusing existing proven technology. In this context we address Design Rule Checking (DRC), DFT architecting and optimization.

DRC can be applied stand-alone, to validate test design rules or as a pre/post-processor to DFT synthesis, to extract information for the purpose of DFT modeling such as sequential cells that violate test design rules and scan chain information. Our implementation of DRC in a traditional DFT synthesis flow is simulation based and therefore relies on the availability of a gate-level netlist. To leverage this technology we present a technique by which we extract representative netlists from test models.

DFT Closure, that is to rapidly and predictably meet all DFT requirements from RTL to GDSII[2], is a mandate for most designs. This needs to be achieved at every stage in the design process, particularly at the chip-level. Therefore HDS must avoid the following:

- Timing violations of design rules and constraints due to test logic.
- Placement violation and routing congestion created by scan path buffers and scan nets.

Although CTL models provide sufficient details for inserting DFT logic and connecting scan structures, they lack information that is required to optimize designs. Design optimization integrated with DFT insertion is already implemented in one-pass DFT synthesis[3]. The challenge lies in enabling this technology in the presence of test models.

This paper describes the combined use of test, timing and physical models to develop a new complete chip-level DFT synthesis flow to handle complex multi-million gate hierarchical designs.

We use Core Test Language (CTL) to describe test models. CTL is the modeling language portion of the proposed P1500 standard for Embedded Core Test[4]. Although the standard is targeted towards SoC methodologies, this paper illustrates a powerful application of CTL to enhance traditional DFT flows.

This paper is organized as follows. In section 2 we present an overview of the classical bottom-up approach to one-pass DFT synthesis. In section 3 we introduce HDS. In this context we briefly introduce some key concepts of CTL and present the new flow and its advantages. In section 4 we discuss the challenges we faced and the techniques we used to migrate existing technology to the new flow. In section 5 we present experimental results and conclude in section 6.

0-7803-7607-2/02/$17.00 ©2002 IEEE

2.0 Classical DFT Synthesis

Figure 1 illustrates a traditional one-pass bottom-up DFT synthesis flow. In this flow, insertion of DFT logic (e.g. test-points) and scan assembly first takes place at a module level. The module designer then hands out a DFT-ready block to the system integrator. This testable block is integrated 'as-is' at higher levels of abstraction. This means during integration, no further changes are allowed to the DFT structures within the block.

As sub-design sizes increase, this flow will soon hit capacity and performance limitations. However it presents several advantages with respect to predictability and optimization during insertion of DFT logic. A key point to be noted here is the fact that once DFT is inserted in the sub-modules, we do not require any information about them other than the DFT structures. Therefore it is possible to model the sub-modules in a more compact fashion instead of retaining the entire netlist during chip-level integration.

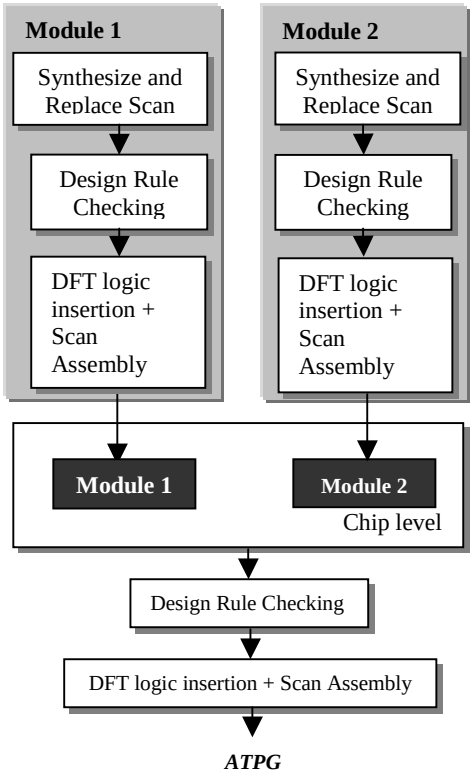

Figure 1. Bottom-up DFT synthesis

3.0 Test Model-based Hierarchical DFT synthesis

3.1 Core Test Language (CTL)

CTL (P1450.6) is being developed as part of the IEEE P1500 standard for Embedded Core Test. The goal is to define a language to describe all the necessary information for test pattern reuse and the needs of test during system integration. Test aspects of a core can be described via CTL so that the core can be integrated as a black box into a SoC design. While [4][5] give more details on the language and syntax, we briefly describe some key concepts that are required for this discussion.

3.1.1 CTL Structure and Syntax

The information contained in the CTL model for a module is classified according to configurations (*modes)* of the module. Figure 2 illustrates this architecture. Every mode has an associated initialization sequence. Some modes contain test pattern information while others contain structural information about the DFT logic included in the module. For the purpose of this discussion we focus on the *InternalTest* mode of operation of the module. This mode allows for the testing of the internal logic of the module through the DFT structures. The CTL description for this mode typically contains the following details:

1. Signals and Signal Groups – Defines the I/O boundary
2. Macros – A template that applies data defined by a pattern in a certain sequence. The initialization sequence for the mode is defined here.
3. Procedures – Define the scan test sequence.
4. Scan Structures – Describes the scan chains.
5. Data Types for various control signals, such as clock, test Mode, and scan enable.

The CTL syntax can be illustrated with a simple example of a DFT ready design shown in Figure 3 and its associated partial CTL model described in Figure 4. The design comprises one scan chain built with 2 multiplexed scan flip-flops and a synchronization latch. Since CTL is under development, the syntax is subject to modification.

The following sections describe the application of test models to HDS.

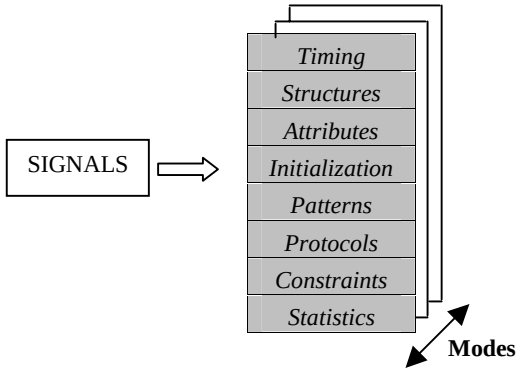

Figure 2. CTL Structure

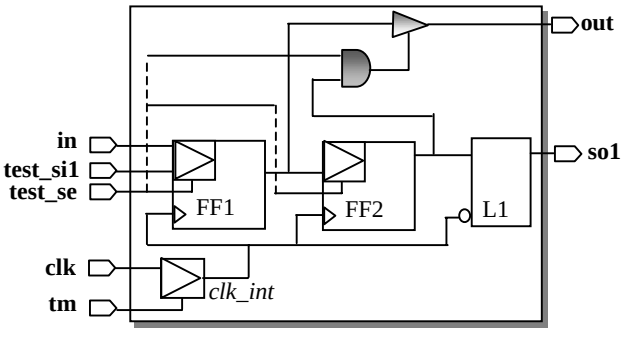

Figure 3. DFT ready design

3.2 Hierarchical DFT Synthesis (HDS)

Figure 5 illustrates the HDS flow. During chip-level integration, we use the test model representation of the sub-modules instead of their netlist representations. Since the sub-modules are DFT-ready, we do not allow any changes to their netlists. Test models are useful here because during integration, we are only concerned with portions of sub-modules that are important for inserting DFT at the higher level of abstraction. CTL enables us to either manually create the test models or integrate the process with automated tools. Thus, the new flow helps accommodate multi-million gate hierarchical designs.

4.0 Challenges

The following sub-sections discuss the challenges imposed by HDS in the context of DRC, DFT architecting and optimization. We present the requirements of each task and describe the techniques devised to meet these requirements by leveraging as much of the existing technology as possible. For the sake of simplicity, all examples assume Multiplexed flip-flop scan style [6].

4.1 Design Rule Checking

4.1.1 Requirements

Our implementation of DRC in a classical DFT synthesis flow relies on symbolic simulation of a test protocol. A test protocol is a formal description of the sequence of operations performed while testing a design. A test protocol for a serial scan design comprises the serial scan-in, parallel measure and capture, and serial scan-out operations [7][8]. The key idea is to logic simulate the process of testing a design and through this simulation verify compliance with scan test design rules. The symbolic simulator is based on a system of classical three-valued logic {*1,0,x*} in addition to values that enable simulation of test protocols. Simulation values are propagated as tokens to establish states in all sequential cells within the design that are then checked for scan compliance. For example, simulation of a scan-in operation should establish an arbitrary known state in all sequential cells within the design. A cell whose state is not controllable represents a design rule violation. This approach generalizes the concept of scan design to any sequential cell that can be controlled and observed through the application of a test protocol. When DRC is invoked on a DFT-ready design, the test protocol is updated with details regarding the new scan structures, if any. Figure 6 illustrates the typical DRC flow.

Symbolic simulation requires a gate-level netlist representation. Modules without such netlists are considered black boxes since we cannot propagate simulation tokens through them. This includes DFT-ready sub-modules with only test model representations.

Therefore, for the purpose of DRC, we need to replace each DFT-ready sub-module with a CTL model by a significantly smaller equivalent netlist that only represents DFT information described in the corresponding test models a process we term *DRC modeling*. This netlist should accurately represent the DFT logic in the sub-module while at the same time preserving the capacity benefit that we realize by using test models.

4.1.2 DRC for HDS

The following sub-sections detail the DRC modeling mechanism. We begin by introducing some definitions that characterize the DFT information that is extracted from a test model representation of a sub-module for the purpose of DRC modeling. We then use these definitions to describe the mechanism in detail.

```
Signals {
    ...
 clk_int Pseudo;
  ...
}

ScanStructures Internal_scan {
   ScanChain "1" {
      ScanLength 2;
      ScanCells "FF1" “FF2”;
      ScanIn "test_si";
      ScanOut "out1";
      ScanMasterClock "clk";
   }
}
Environment “TOP” {
CTL Internal_scan {
   Internal {
      "clk" {
         DataType MasterClock ScanMasterClock;
      }
      "clk" {
         IsConnected Out
         { Signal  clk_int; }
      }
      "test_si" {
         CaptureClock "clk_int" ;
         DataType ScanDataIn ;
         { ScanDataType Internal; }
      }
      "test_se" {
         DataType ScanEnable
         { ActiveState ForceUp; }
      }
      "so1" {
         LaunchClock "clk_int"
         { LeadingEdge;  }
         OutputProperty SynchLatch;
         DataType ScanDataOut
         { ScanDataType Internal;  }
      }
      “out1” {
         IsDisableBy Out Logic !a0 {
            a0 {
               Type signal;
               Name test_se;
            }
         }
      }
      ”tm” {
            DataType TestMode;
            { ActiveState ForceDown;}
       }
    }
}
```

Figure 4. Partial CTL model

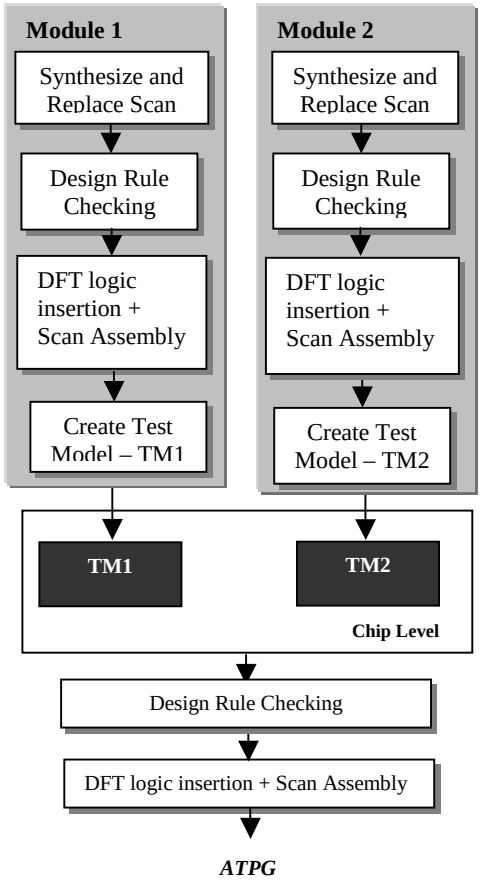

Figure 5. High-capacity DFT flow

4.1.2.1 Definitions

a. Scan Segment

This concept of scan segment has been introduced in the context of hierarchical scan synthesis [9].

A scan segment is a chain containing one or more completely connected scan cells. A scan chain specified as a scan segment is complete and atomic in the sense that it cannot be reconfigured. A scan segment has a scan-in pin *si* and a scan-out pin *so*. A scan segment can be made a part of another scan chain. Therefore in a bottom-up flow, at the module level each scan chain is a scan segment. At the chip-level integration, each scan segment in the sub-module can be a part of a scan chain. Our discussion on DFT architecting will further elaborate on scan segments and how to structure them. Here we will concentrate on the application to DRC.

b. Clock domain

An active edge (leading edge or trailing edge) of each clock is considered to be in a separate clock domain. Both edges of a clock and clocks with different timing characteristics may be used to control edge-triggered scan flip-flops of a scan chain. In order to construct functional scan chains, two adjacent scan flip-flops A and B (A serially driving B) must adhere to the rule that B must be clocked at the same time or before A.

c. Clock ordering

The precedence relationships between scan flip-flops imposed by clock domain timing characteristics are defined at the scan segment level. Capture and launch times for a scan segment are deduced from the capture time of its first scan cell (driven by its scan input) and the launch time of its last scan cell (driving its scan output). This this precedence relationship between scan segments can be respected during chip-level DFT architecting.

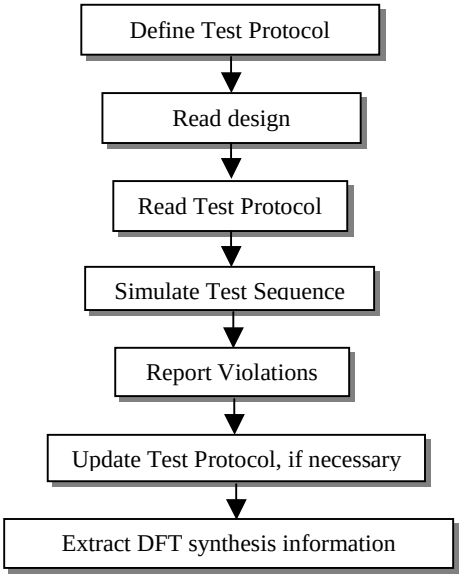

Figure 6. Design Rule Checking

4.1.2.2 DRC Modeling

Figure 7 transcribes the algorithm and Figure 8 illustrates the scheme followed for DRC modeling. The algorithm makes use of the following terminology:

$\boldsymbol{S} = \{s_1, s_2 \ldots s_n\}$ is the set of all scan segments in a DFT-ready sub-module.

$\boldsymbol{N_i}$ is defined as the number of scan cells in scan segment s_i.

$\boldsymbol{C_M} = \{c_{m1}, c_{m2} \ldots c_{mn}\}$ ***is*** the set of all scan master clocks for a scan segment.

$\boldsymbol{C_S} = \{c_{s1}, c_{s2} \ldots c_{sn}\}$ is the set of all scan slave clocks for a scan segment.

Each clock has an associated rise time *r*, and fall time *f*.

$\boldsymbol{A} = \{a_1, a_2 \ldots a_n\}$ is the set of asynchronous sets/resets for a scan segment

$\boldsymbol{E} = \{e_1, e_2 \ldots e_n\}$ is the set of scan enable signals for a scan segment

Figure 8a shows the netlist of a sub-module (M) from which a test model is generated. L1 and L2 denote combinational logic and ff1, ff2 and ff3 are multiplexed D flip-flops. Figure 8b describes the integration of sub-module M at a higher level of abstraction (design Top). L' and L" could be combinational or sequential user-defined logic (UDL). At this level, sub-module M appears as a pure black box. Only the interface and the test model are visible to the DFT

architect. Figure 8c shows the equivalent netlist resulting from DRC modeling.

Therefore we replace the black box by a netlist (*CTL-to-gates)* that we quickly construct from the test model description, the basic idea being to minimize the size of the scan segments . DRC then operates on the modeled netlist.

For DRC, we need to ensure that all control and access pins for scan segments are completely represented in the model. This is required for accurate validation. As long as this property is preserved during the modeling process, there will be no loss of critical information. For example, a scan segment of any sequential length, with one asynchronous input, and one clock domain, can be represented by a single scan cell.

Therefore, we obtain a significant reduction in segment length for most scenarios. In the worst case when $\boldsymbol{A = N}$ or $\boldsymbol{C_M = N}$, there will be no reduction in the segment length. A similar technique can be applied to other scan styles as well by using more complex technology-independent, and pre-defined set of cells.

The following considerations figure in this context:

- We do not perform capture checks since they require functional information that is not available in the test model. Capture checks are more pertinent to automatic test pattern generation.
- In the case of 3-state pins, the model is enhanced with additional structures to represent disabling logic.
- We assume that segments in the test-modeled sub-designs are correct by construction.
- The DRC module is a transient entity and is removed at the end of rules checking. Therefore violations detected on cells of the model must be reported at the boundary of the sub-module and not on the virtual cells that constitute the model.

This technique has the following impact:

- The test protocol needs to be updated before simulation to reduce the number of simulation cycles depending on the length of the DRC modeled segment. The original value is restored at the end of simulation.
- We need to maintain correspondence between the original scan segment and the scan segment in the DRC model. This is critical to reporting violations on the correct entity.
- One of the functions that DRC performs, when run as a post-processor to DFT insertion, is to extract scan structures for reporting purposes. Here, care should be taken here to extract the original scan segment out of the DRC model.

This solution therefore allows us to leverage the current DRC technology, without modifying the DRC engine.

∀ test-modeled sub-module M in Top

∀ scan-segment s_i in *S:*

- ∀ clock $c_{mj\ (j\ =\ 1\ldots n)}$
  Perform clock ordering on C_M based on *r* and *f*
- Create a new netlist SN_i with N_i new identical serially connected scan cells
- **$n = MAX(C_M, A)$**
- Connect clocks in C_M to n scan cells in SN_i (one clock per cell) in order
- If ($n < N_i$) connect c_{mn} to the remaining (N_i -n) scan cells
- Connect signals in *A* to SN_i (similar to clocks)
- Connect signals in *E* to SN_i
- Connect scan segment access pins
- Replace s_i with SN_i

Figure 7. DRC modeling algorithm

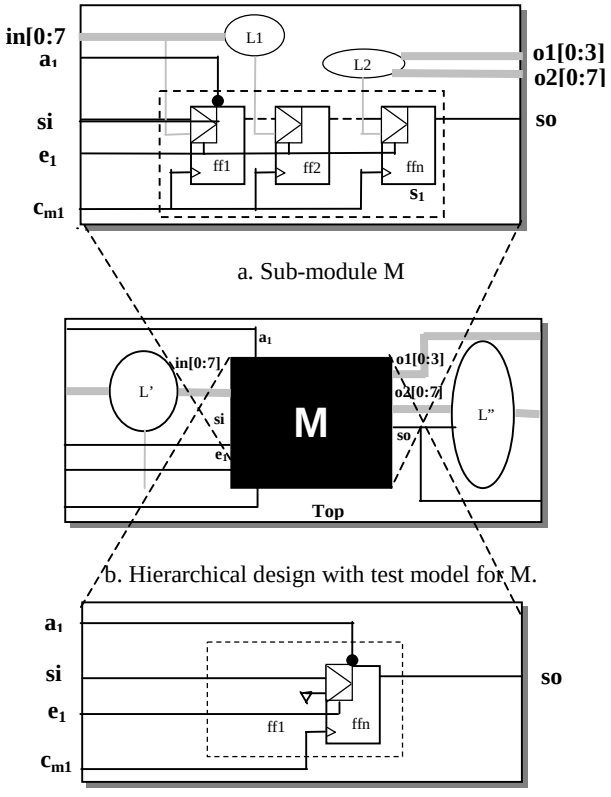

Figure 8. DRC modeling

4.2 DFT Architecting

4.2.1 Requirements

Given a design encapsulating a set of module instances, each pointing to a CTL model, the task of DFT architecting, involves generating a complete DFT plan for the design, integrating DFT structures from the modeled instances. The DFT structures created at the design level must preserve the testability achieved at the module level and provide access to the instances from the design ports.

DFT insertion using CTL models must obey the same rules as the classical flow that works with leaf level cells. Controllability of clock and asynchronous signals, clock sensitivity, balancing of scan chains, avoidance of float and contention conditions during scan shifting are typical constraints that drive DFT architecting. These constraints require that the CTL model contain information such as:

1. Scan structures
2. Enabling/disabling conditions on tri-state ports
3. Test Control ports and purpose
4. Clock dependency
5. Asynchronous control signals

4.2.2 DFT Architecting for HDS

Migrating from a full gate level netlist representation of a sub-module to a CTL model requires changes to the way we evaluate the design for DFT architecting. To facilitate this we utilize the concept of a scan segment as described earlier.

A scan segment is characterized by the following:

1. Scan length
2. Serial scan access ports
3. Scan control signals.
4. Clock synchronization
5. Test control signals

In addition to this we might have information about enabling/disabling conditions for tri-state signals, attached to the scan segment.

Recollect that during chip-level integration, each scan segment in the sub-module can be made a part of a scan chain.

The description of the scan structures in a CTL model identifies the scan serial inputs, serial outputs and global inputs ports of the block. This indicates how the segment should be connected. Information from the CTL model like **CaptureClock, LaunchClocks, DataType, Scan Length** (Figure 4) is modeled as a scan segment object. Therefore we encapsulate all the information about scan structures available in a CTL model in the form of scan segments. At the end of this process, each scan segment becomes analogous to a leaf level scan cell. Once all the scan segments are extracted from the CTL description of a sub-module, chip-level DFT architecture and insertion can happen transparently. By making the scan segment the unit of representation, we blur the difference between cell instances that are leaf level cells and instances that are characterized by CTL models.

Proper attention must be paid to disabling logic on the tri-state signals as this may cause contention or float conditions during scan shifting if not properly accounted for. In the CTL model, tri-state logic driving outputs of the sub-module is represented by the **IsDisabledBy** property and a boolean equation combining inputs of the sub-module. This information is translated and stored as boolean equations in our internal model. During integration, combining these equations will help identify the enabling and disabling conditions for drivers connected to the same bus.

4.3 DFT Insertion and Optimization

4.3.1 Requirements

Since DFT insertion calls for design modification, it is important that the design be optimized in order to meet logical and physical synthesis constraints. CTL models lack timing and cell information that is required during optimization. In the example of Figure 3, the functional output of a scan register or a block is used as a scan output; thereby increasing the load on the register "FF_2" or the module output pin. This timing consideration is illustrated in Figure 9.

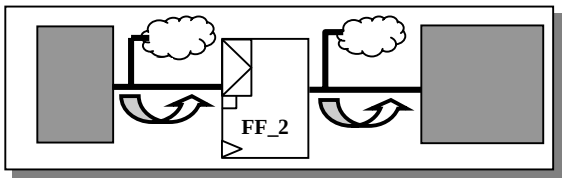

Figure 9. Chip-level timing path

Design optimization during DFT insertion has been implemented in what is known today as one-pass DFT synthesis[3]. This flow is part of our DFT solution and has been proven on a variety of industrial designs. In order to harness this technology for HDS we require information that is not available in the CTL model. This includes loads on inputs, drive strength of outputs, timing path to the closest register etc. Therefore we rely on additional models for logical and physical optimization.

4.3.2 DFT Insertion and Optimization for HDS

4.3.2.1 Logic Optimization

Recently, logic synthesis tools have started using timing models to cope with increased design sizes. The model used, called Interface Logic Model (ILM) [10], is well suited for synthesis and timing analysis. The concept is illustrated in Figure 10 where we show a module and its ILM representation. The ILM shown here encapsulates the following information:

1. The cloud of combinational logic going from input ports to the sequential cells in the fan-out cone of these ports.

2. The cloud of combinational logic between output ports and sequential cells in the fan-in cone of these ports.

3. The cloud of combinational logic between ports.

This partial representation of the design enables us to extract accurate timing information and design port characteristics.

If the full gate level netlist is available, one-pass DFT synthesis integrates DFT insertion and design optimization steps to concurrently fix synthesis design rule and timing constraint violations. For hierarchical designs, availability of ILM and CTL models for sub-modules extends the one-pass DFT synthesis approach to be applicable during chip-level assembly. During chip-level DFT synthesis, unified models that have both CTL and ILM information are used to perform DFT synthesis to meet both the timing and test constraints.

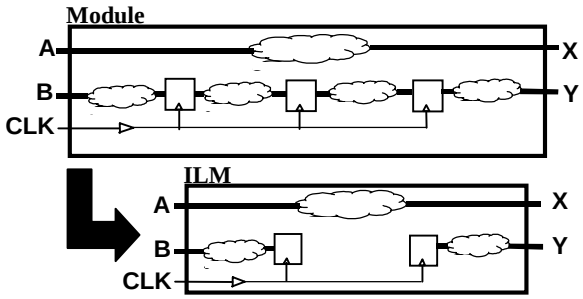

Figure 10. ILM model creation

4.3.2.2 Physical optimization

Timing and placement driven optimization is at the core of today's physical synthesis tools. In addition to logic optimization based on library cell data, physical optimization aims at incorporating parameters such as wire geometry, cell placement to achieve sharper design analysis and better quality of results.

During chip-level integration, routing scan nets is a significant portion of the overall routing process. Scan chain ordering[11] is widely used at the sub-module level as well at the chip-level to reduce overall routing congestion and reduce connection lengths; thereby minimizing the impact of scan nets on timing. Hence, incorporation of physical data during DFT insertion is critical to timing closure. This is illustrated in Figure 11.

By enhancing the unified ILM & CTL models with physical data, we allow better optimization of block level scan structures during chip-level assembly. With a structure similar to the Layout Exchange Format (LEF) representation, the outside geometry of a DFT inserted block and the position of its scan ports is captured as a physical model. The precise position of scan ports on the block provides more accurate information than just the simple block location. As shown in Figure 11, those models drive DFT synthesis to perform optimal partitioning and ordering of block scan structures. This reduces routing congestion due to scan nets. Long scan nets are handled by addition of buffers during timing driven optimization.

Therefore by unifying CTL, ILM and physical models we avoid iterations between DFT synthesis, logic optimization and placement thereby guaranteeing a fully integrated flow.

5.0 Experimental Results

The techniques described above have been implemented as part of our commercial DFT Synthesis tool. Two experiments were run to compare the memory usage and CPU run time between the conventional DFT synthesis flow and the new flow. The comparisons are based on applying the same tool to two different flows, the classical DFT synthesis flow versus the HDS flow using test-models. Table 1 describes the statistics of the design used for the experiments.

In Experiment 1 we inserted DFT on the design containing 3 hierarchical sub-modules. We performed DFT synthesis on each block and saved it as both full gate database and test model representations. During the top level DFT synthesis we compared the memory consumption and CPU run time between using full gate database and using test models. The comparison was done at different stages in the flow. Table 2 describes the results. We observe 2X improvement in memory and 4X improvement in run-time.

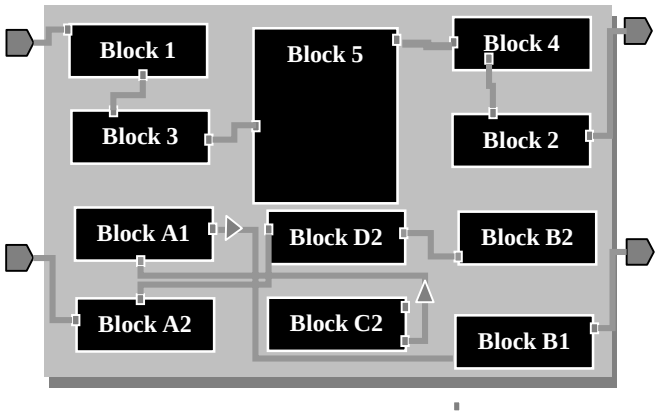

Figure 11. Physical DFT architecture and optimization

In the Experiment 2 we instantiated the design used in the first experiment eight times and performed top-level DFT synthesis. The total number of transistors was around 4 million. Again we compared the memory consumption and CPU run time between using full gate and test model representations. Table 3 describes the results. We observe 7X improvement in memory and 41X improvement in run-time.

The results show that we can obtain significant capacity and performance benefit by replacing netlists with test-model representations, especially for large hierarchical designs.

6.0 Conclusion

Capacity and performance bottlenecks are growing concerns for most commercial EDA tools. In this paper, we present a new hierarchical DFT synthesis flow based on test, timing and physical models. This work was motivated by a comprehensive effort to address the capacity issue with current DFT synthesis flows, and illustrates a powerful application of the IEEE P1450.6 Core Test language (CTL) for test modeling.

The combined use of DFT, timing and physical abstractions has enabled us to solve the capacity bottleneck without compromising on the quality of results. Core Test Language is a technology enabler for modeling DFT information. Using ILM abstractions, timing closure is achieved during top-level DFT synthesis. Physical abstraction enables better ordering of DFT structures that reduces top-level routing congestion.

The techniques described here have been implemented as part of our commercial DFT synthesis solution. Several enhancements to the current implementation are being planned. Some of these being support for test-models with pass-through control signals (such as clocks, resets), and support for models with more complex sequential initialization sequences (as opposed to combinational initialization sequences that are currently supported).

7.0 References

1. Gupta. R.K, Zorian. Y, *"Introducing Core-Based System Design"*, IEEE Design and Test of Computers, December 1997. Pages: 14(4): 15-25
2. Hayat. F, Williams. T.W, Kapur. R, Hsu. D, *"DFT Closure"*, Proceedings of Asian Test Symposium, 2000. Pages: 8-9.

3. Hirech. M and Ramnath. S, "*Moving from one-pass scan synthesis to one-pass DFT synthesis*", European Test Workshop, 2001. 3B.1
4. Kapur. R, Keller. B, Koenemann. B, Lousberg. M, Reuter. P, Taylor. T and Varma. P, ***"P1500-CTL: Towards a Standard Core Test Language",*** Proceedings of 17th IEEE VLSI Test Symposium, 1999. Pages: 489-490.
5. Kapur. R, Lousberg. M, Taylor. T, Keller. B, Reuter. P and Kay. D, ***"CTL the language for describing core-based test",*** Proceedings of International Test Conf, 2001. Pages: 131–139.
6. Scan Synthesis Reference manual, release 2001.08, Synopsys Inc., Mountain View, CA, 2001.
7. Varma. P, *"TDRC – A Symbolic Simulation Based Design for Testability Rules Checker",* Proceedings of International Test Conference, 1990. Pages: 1055-1063
8. Pitty. E.B, Martin. D and Ma. H.-K.T, ***"A simulation-based protocol-driven scan test design rule checker",*** Proceedings of International Test Conference, 1994. Pages: 999-1006.
9. Beausang. J, Ellingham. C and Robinson. M, ***"Integrating scan into hierarchical synthesis methodologies",*** Proceedings of International Test Conference, 1996. Pages: 751–756.
10. Daga. A, Ananthanarayanan. S and Neuveux. F, *"Interface Logic Models in a Hierarchical SoC Design Flow",* Submitted to CICC 2002.
11. Hirech. M, Beausang. J and Gu. X., ***"A new approach to scan chain reordering using physical design information",*** Proceedings of International Test Conference, 1998. Pages: 348-355.

	Area (transistors)
Sub-module #1	1,123
Sub-module #2	54,709
Sub-module #3	455,604
Top level glue logic	1,375
Total	**512,811**

Table 1. Design statistics

	Using Full gates		**Using Test models**		**Improvement (X)**	
Stage	**Mem (kb)**	**Cpu time (s)**	**Mem (kb)**	**Cpu time (s)**	**Mem**	**Cpu time**
Read in sub-modules and top level netlist	44752	5	1680	1	26.6	5
After pre-DFT DRC	175168	193	70656	88	2.48	2.19
After DFT insertion	200784	366	89392	107	2.24	3.42
After post-DFT DRC	208960	507	92176	117	2.26	4.33

Table 2. Experiment 1

	Using Full gates		**Using Test models**		**Improvement (X)**	
Stage	**Mem (kb)**	**Cpu time (s)**	**Mem (kb)**	**Cpu time (s)**	**Mem**	**Cpu time**
Read in sub-modules and top level netlist	387936	41	4960	2	78.2	20.5
After pre-DFT DRC	1082720	4481	128784	145	8.41	30.9
After DFT insertion	1207680	6041	167752	216	7.20	28.0
After post-DFT DRC	1285288	11113	183304	271	7.01	41.0

Table 3. Experiment 2