A High-level EDA Environment for the Automatic Insertion of HD-BIST Structures

A High-level EDA Environment for the Automatic Insertion of HD-BIST Structures

Authors: Benso A., Cataldo S., Chiusano S., Prinetto P., Zorian Y.

Author's version of the manuscript published in the Journal of Electronic Testing, Vol. 16, No. 3, 2000 pp. 179-184.

DOI: 10.1023/A:1008326928340

A High-level EDA Environment for the Automatic Insertion of HD-BIST Structures

Alfredo BENSO, Silvia CATALDO, Silvia CHIUSANO, Paolo PRINETTO

Politecnico di Torino
Dip. Automatica e Informatica
Corso Duca degli Abruzzi 24
I-10129 Torino TO, Italy
{benso, cataldo, chiusano, prinetto}@polito.it
http://www.testgroup.polito.it

Yervant ZORIAN

Logic Vision
San Jose CA, USA
zorian@lvision.com

Keywords: *built-in self-test, embedded cores, EDA tools*

Abstract

This paper presents a High-Level EDA environment based on the Hierarchical Distributed BIST (HD-BIST), a flexible and reusable approach to solve BIST scheduling issues in System-on-Chip applications. HD-BIST allows activating and controlling different BISTed blocks at different levels of hierarchy, with a minimum overhead in terms of area and test time. Besides the hardware layer, the authors presented the HD-BIST application layer, where a simple modeling language, and a prototypical EDA tool demonstrate the effectiveness of the automation of the HD-BIST insertion in the test strategy definition of a complex System-on-Chip.

1. Introduction

Today's complexity of digital systems is significantly increasing. The availability of a big variety of *cores* provides designers the possibility of integrating a large number of different functional blocks in a single IC. Due to their own complexity and the high integration density, embedded cores are mostly tested resorting to BIST capabilities only.

In such complex Systems-on-Chip, the definition of a global and efficient test strategy faces two main issues. From one hand, the designer has to organize the activation of the BIST session of a very large number of different blocks taking into account attributes as *at-speed* testing capabilities, *power* consumption, *area* and *routing* minimization. On the other hand, very tight cost and time-to-market constraints ask for the maximization of the *reuse* and the *automation* of the BISTing process.

The key of the problem is to define a global test strategy approach with the following characteristics:

- Flexibility, in terms of:
  - *Test structure*: the hardware structure inserted to activate the different BISTed blocks has to be optimized and customizable, allowing a trade-off among routing, area, and test time overhead.
  - *Scheduling*: the scheduling definition must allow the designer to activate the BISTed blocks in any order via complex scheduling instructions as *wait* and *conditional* operations.
  - *BIST protocol*: the approach must define a unified approach to access BISTed blocks with different BIST access protocols.
- Reusability: the test structure must be reusable during different phases of the product test cycle (horizontal reuse), and at different levels of integration (vertical reuse) (e.g., System-On-Chip level, board level, etc…).
- High-level description: besides the detailed definition of the actual hardware implementation (hardware layer), the designer should be provided with an application layer, where a high-level scheduling definition language and an ad-hoc compiler allow him to easily define complex scheduling algorithms and automatically derive the required hardware logic.

Most of the approaches proposed so far rely on a single controller to perform device level BIST scheduling. [1] and [2] present a centralized controller, implementing the scheduling of the BIST sessions through the activation of one session a time. In order to reduce the routing cost, [3] proposed a distributed architecture, where intermediate blocks link the units under test to a centralized controller, and are used to control and activate the test of a sub-set of BISTed blocks. Although the novelty of

the approach, both the modularity and the flexibility of the architecture are still limited.

In this paper the authors present a High-Level EDA environment based on the Hierarchical Distributed BIST (HD-BIST), a flexible and reusable approach to solve BIST scheduling issues in System-on-Chip applications. HD-BIST allows activating and controlling different BISTed blocks at different levels of hierarchy, with a minimum overhead in terms of area and test time. The hardware layer of the proposed architecture, already introduced in [4] has been reanalyzed and significantly improved, and is briefly summarized in Section 2.

Besides the hardware layer, the authors present the *HD-BIST application layer*, where a simple *modeling language* and a prototypical *EDA tool* demonstrate the effectiveness of the automation of the HD-BIST insertion in the BIST planning of SoCs. In particular, the high-level language allows describing the system topology, the BISTed blocks access protocol, and the scheduling algorithm, whereas the compiler derives the corresponding synthesizable VHDL RT-level code of the required hardware layer. Whenever a change is made to the project, the designer must simply modify and recompile the code always obtaining a highly-integrated and optimized solution. Using the proposed EDA environment the HD-BIST architecture can be therefore fully customized according to each specific application, trading-off between test scheduling flexibility and overhead in terms of both area and routing.

The proposed architecture is compatible with the on going activities of the Task Force on Scaleable Architecture of the IEEE TTTC P1500 Working Group [5].

The paper is organized as follows: Section 2 summarizes the HD-BIST hardware layer, whereas Section 3 introduces the main features of the prototypical environment implementing the application layer. Eventually, Section 4 draws some conclusions.

2. HD-BIST architecture: basic principle

HD-BIST is a *Hierarchical* and *Distributed BIST* control approach, aiming at managing in an extremely *flexible* way the BIST scheduling of complex SoCs. The units composing the system are assumed to be already BISTed, and the HD-BIST structure provides the additional logic needed to manage the test.

In particular, the test of the different blocks is controlled using the following three types of modules (Figure 1):

- Test Blocks (TBlock): wrappers placed around the BISTed blocks, used to control their BIST procedures through a standard interface. They guarantee the maximum reusability at different levels of integration and the maximum flexibility in terms of access protocol implemented in the target blocks.
- the Test Processor (TProcessor): a centralized controller in charge of:
  - Activating the BIST procedures of each TBlock.
  - Checking the status (finished/not finished) and the result (good/failed) of the test.
  - Locating a failed TBlock and detecting a structural fault of the HD-BIST structure itself.
- the Test Chain Bus (TBus): a bus connecting the TProcessor and the TBlocks in a ring connection. It is actually implemented as a bi-directional link where the communications e carried on through a token-based self-checking protocol. The TProcessor sends the TBlocks different types of tokens via the TBus to implement both the required scheduling and the diagnosis sessions.

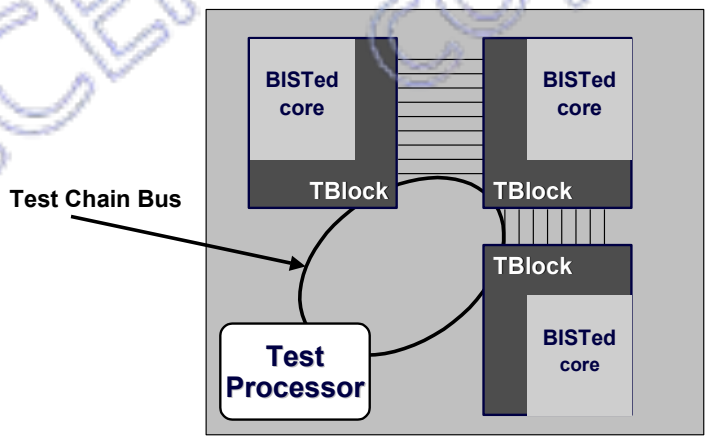


Figure 1: *HD-BIST basic architecture*

The basic architecture, composed of a TProcessor, a number of TBlocks, and a TBus, can be replicated to implement a hierarchy of test chains, each controlled by a dedicated TProcessor. In this case, besides managing the BIST scheduling of the TBlocks included in the lower chain, the TProcessor is also in charge of correctly re-

sponding to the commands received through the upper chain. From the upper chain point of view, the TProcessor is a TBlock, responding to the standard scheduling commands sent on the TBus. In this way, an entire test chain can be managed as a single test entity. This extension of the basic HD-BIST structure highlights its applicability to complex SoCs as well as to complex digital systems based on different hierarchical levels (Figure 2).

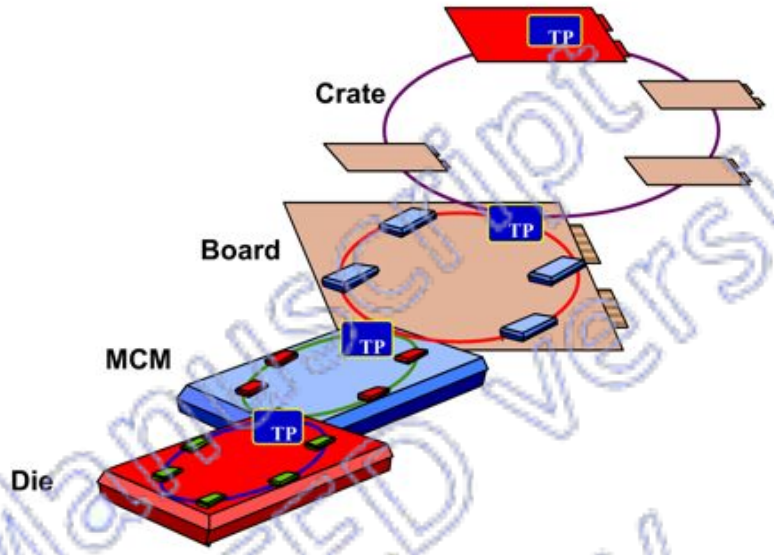


Figure 2: *HD-BIST implementation at different hierarchical levels*

The internal functionality of a TBlock is based on a *Test Status Register* (TSR) and a *Test Control Register* (TCR). All the command tokens sent to a TBlock require writing a single bit in the TCR or reading a single bit from the TSR. This solution allows us to implement a more efficient protocol at both the TBlock and TBus level.

Moreover, to provide the maximum flexibility in test scheduling, each token contains the address of the target TBlock. In particular, each TBlock responds to two addresses: a *single-cast address* that uniquely identifies the block in the chain, and a *broadcast address*, used to address concurrently all the TBlocks of the chain.

To activate the BIST of a given TBlock, the TProcessor sends on the TBus a proper token containing the address of target TBlock. Upon receiving the token, the TBlock activates the BIST procedures of its target block. In the meanwhile, the TProcessor starts polling the target TBlock, waiting for the end of the test. The same procedure is executed (not necessarily in a serial approach) for all the other TBlocks of the chain. In multi-chain architectures, when a TProcessor receives a BIST activation

command from the upper chain, it starts executing the scheduling algorithm of its sub-chain. When all the blocks finished their test, the TProcessor starts the diagnosis phase and collects the addresses of the faulty TBlocks (if any) of the structure. The detailed algorithm implemented to perform the scheduling and the diagnosis procedures, as well as the complete description of the HD-BIST hardware layer, can be found in [4].

3. The High-level EDA environment

The development of the high-level EDA environment aims at providing the designer with a very easy-to-use tool, to fully automate the insertion of the HD-BIST architecture. The tool allows strictly distinguishing between the *application* and the *hardware* layer, covering, at the same time, all the steps of the design flow.

The *HD-BIST Modeling Language* (HDML), an ad-hoc defined high-level description language, allows the designer to describe the planned test strategy (application layer), without taking into account any implementation details. An ad-hoc compiler translates the HDMI description of the HD-BIST architecture and scheduling, and generates the corresponding RT-level VHDL code (hardware layer) of the required hardware blocks, which can be finally synthesized using a standard commercial synthesis tool (Figure 3).

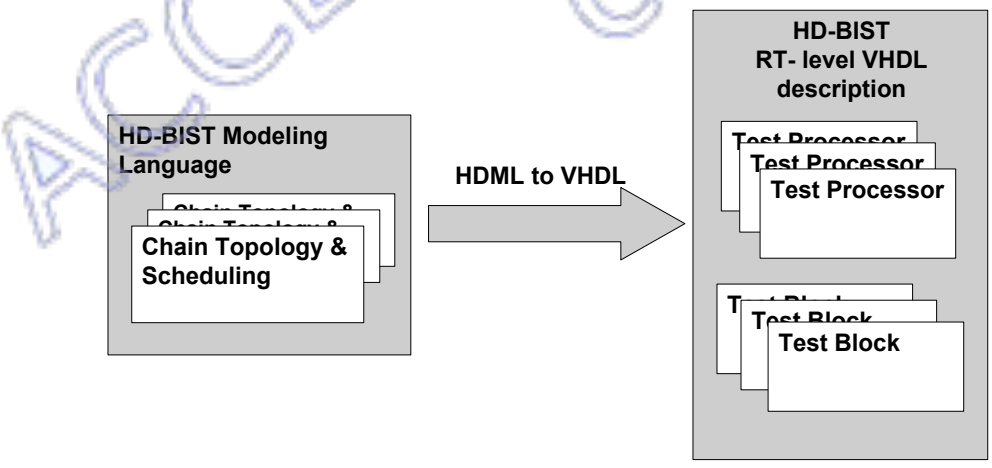


Figure 3: *HD-BIST design flow using the EDA tool*

The language allows describing the system (i.e., the target BISTed blocks, the structure of the planned HD-BIST architecture, and the required BIST scheduling) as

a sequence of *chain* declarations, each one containing its characterization in terms of both *topology* and *scheduling*. Defining topology and scheduling, each TBus belonging to the HD-BIST architecture can be fully customized, in terms of routing overhead, number of blocks and scheduling. In the *topology* section, the designer lists the BISTed blocks belonging to the chain. For each block he specifies the BIST access protocol, consisting in the signals type and protocol required activating and collecting the BIST results. Eventually, the designer specifies some structural parameter as the *width* of the TBus itself.

The *scheduling* section contains the scheduling algorithm specified in terms of *scheduling instruction*, each defining the *scheduling primitive* that the TProcessor must send to each TBlock to manage its BIST procedures. The syntax of each scheduling instruction is:

`<scheduling command> <block>...<block>,`

where `<block>...<block>` is the list of the target TBlocks addresses, whereas the `<scheduling command>` can be one of the following scheduling primitives:

- `Test`: this primitive is used to activate the BIST of a set of TBlocks. If the TBlock is also a TProcessor, this primitive starts the execution of the correspondent sub-chain.
- `Wait`: this primitive forces the TProcessor to wait until the BIST procedure of a set of TBlocks is finished. After the end of the BIST procedures, the TProcessor checks their results and continues with the next scheduling instruction.
- `Stop`: as the wait primitive, the stop primitive waits for the end of the BIST session of the target TBlocks, and then collects their results. At that point, differently from the wait primitive, if one of the targets TBlock failed its test, the scheduling algorithm is aborted.
- `Diagnose`: this primitive activates the diagnosis phase; the whole HD-BIST structure is examined in order to collect the addresses of all the possibly faulty TBlocks, at any level of hierarchy. Given the actual definition of the HD-BIST approach, this primitive can be executed only from the top-level TProcessor, and is therefore not allowed in the scheduling definition of lower level chains.

As shown later in the example, these four simple scheduling primitives allow defining complex scheduling algorithms.

The compiler has been implemented in C++ and required about 4000 lines of code. Ad-hoc classes allow mapping high-level specifications of each chain into the VHDL code of the TBlocks and TProcessors required to implement it. In particular, for each block specified in the *topology* section, the compiler provides the VHDL description of the TBlock needed to interface the BISTed block with the TBus. The TBlock is therefore fully customizable according to the characteristic of the chain (e.g., width of the test chain bus), so as the characteristics of the block itself (e.g., communication protocol for interfacing the BIST logic of the block). Moreover, each *scheduling* section is compiled into a TProcessor described as a Finite State Machine. Each *test instruction* corresponds to a set of tokens sent by the TProcessor over the TBus. The capability of the TProcessor of addressing a single or a group of blocks in the chain is added according to the information about the topology of the chain.

As an example, Figure 4 reports a high-level description of a simple HD-BIST architecture. The system consists of two nested chains. The former, controlled by the top TProcessor (`top`), contains a BISTed RAM core (`BISTedRAM`), a legacy core (`BISTedCore1`), and a TProcessor (`TestProcessor1`) that controls the second chain. The latter links a BISTed ROM (`BISTedROM`) and a second legacy core (`BISTedCore2`). For the sake of simplicity, in the example we omitted the specification of the BIST access protocol of each block.

The scheduling algorithm of the first chain is implemented using the following scheduling instructions:

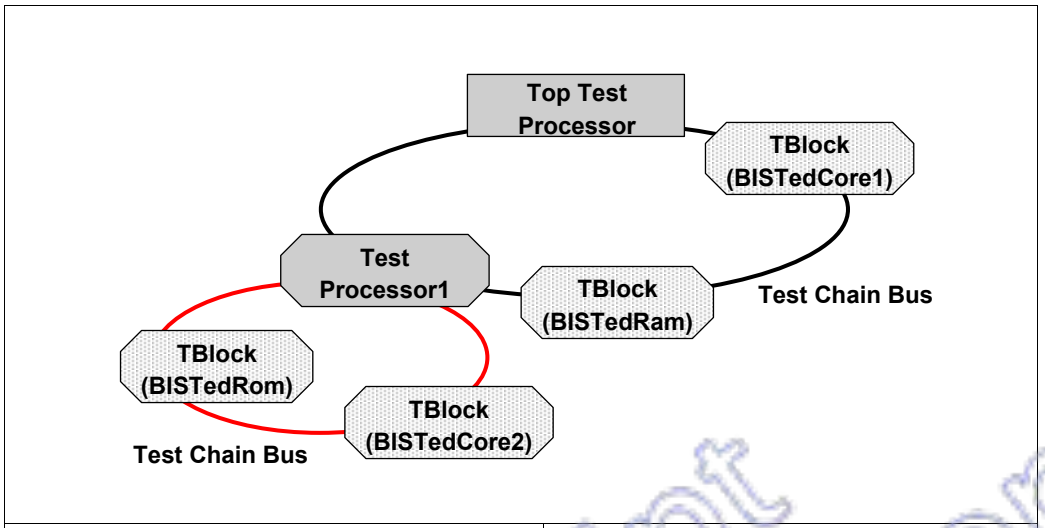
- `test BISTedRAM BISTedCore1`: the `top` TProcessor activates the BIST of both the RAM and the first legacy core.
- `wait BISTedCore1`: the `top` TProcessor waits until the specified block concludes its test session. At the end the TProcessor collects the result of the BIST procedure.
- `stop BISTedRAM`: the `top` TProcessor waits for the end of the BISTed Ram test session and then collects its test results. In case the test fails, the execution of the scheduling algorithm is aborted.

- `test TestProcessor1`: the `top` TProcessor activates the BIST of the `TestProcessor1`. This operation results in the activation of the BIST scheduling of the second chain.
- `stop TestProcessor1`: the `top` TProcessor waits until the second chain concludes its scheduling and then checks the result. Since the `top` TProcessor considers the `TestProcessor1` TProcessor as a TBlock, if some BIST procedure of the second chain failed, the check operation will result in a 'faulty' `TestProcessor1`.
- `diagnose`: at this point, since all the BIST sessions are completed, the `top` TProcessor starts the diagnosis phase and collects the addresses of the possibly faulty blocks. Although the top-level TProcessor is not able to directly address all the blocks of the hierarchy, the diagnosis procedure is designed so that to collect the address of the faulty blocks in any level of the hierarchy.

Similarly, the scheduling algorithm of the second chain is defined by the following two instructions:

- `test all`: the `TestProcessor1` activates the BIST of all the blocks of the chains, the ROM and the legacy core.
- `wait all`: the `TestProcessor1` waits until all the blocks conclude their test session and then checks for their results.

Since `TestProcessor1` is located on a secondary chain, it is not allowed to activate a diagnosis phase, which can only be started by the top-level TProcessor.




```
Chain top
{
  BusWidth 1
  Topology
  {
    TestProcessor1
    BISTedRAM
    BISTedCore1
  }
  Scheduling
  {
    test BISTe  AM BISTedCore1
    wait BISTedCore1
    stop BISTedRAM
    test TestProcessor1
    stop TestProcessor1
    diagnose
  }
}
```

```
chain TestProcessor1
{
   BusWidth 1
   Topology
   {
     BISTedRom
     BISTedCore2
   }
   Scheduling
   {
     test all
     wait all
    }
}
```

Figure 4*: High-Level HD-BIST architecture*

4. Conclusions

In this paper the authors presented HD-BIST, a flexible and reusable approach to solve BIST scheduling issues in System-on-Chip applications. Besides the hardware layer briefly summarized, the authors presented the *HD-BIST application layer*, where the HDML, a simple *modeling language*, and a prototypical *EDA tool* demonstrate the effectiveness of the automation of the HD-BIST insertion in the test strategy definition of a complex System-on-Chip.

5. References

[1] F. Beenker, R. Dekker, and, R. Stans, “Implementing MACRO Test in Silicon Compiler Design”, IEEE Design & Test of Computers, pp. 41-51, April 1990

[2] O.F. Haberl, and T. Kropf, “HIST, A Methodology for the Automatic Insertion of a Hierarchical Self test”, Proc. IEEE Int. Test Conference, pp. 732-741, 1992

[3] Y. Zorian, “A distributed BIST Control Scheme for complex VLSI devices”, Proc. IEEE VLSI Test Symposium, pp. 4-9, April 1993

[4] A. Benso, S. Chiusano, P. Prinetto, and Y. Zorian, “HD-BIST: a Hierarchical Framework for BIST Scheduling and Diagnosis in SoCs”, to apper on Proc. IEEE Int. Test Conference, September 1999

[5] http://grouper.ieee.org/groups/1500/#TaskForces