Automatic Insertion of BIST Hardware Using VHDL

Kwanghyun Kim Joseph G. Tront Dong S. Ha

Department of Electrical Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

Abstract

We present a system which automatically inserts BIST hardware to a circuit described in VHDL. An appropriate VHDL modeling style for automatic insertion of BIST hardware is investigated. Use of BILBO is primarily pursued in the system. Algorithmic and rule-based approaches are used in the insertion of BILBO. Test scheduling and control signal distribution are also performed by the system.

1. Introduction

Due to the increasing complexity of VLSI circuits, the accessibility of the circuit is decreased. This decreasing accessibility makes traditional test generation and application methods ineffective and costly. In order to solve this testing problem, several Built-In Self-Testing (BIST) techniques have been proposed [13]. The basic principle of BIST is to insert extra circuitry so that the generation of test patterns and verification of the test occur within the logic itself. However, along with the benefits, there are some penalties in employing BIST. These include hardware overhead, increased pin-count, possible performance degradation and additional design cost. In order to reduce these penalties, insertion of BIST hardware should be tightly coupled with the design process, and the insertion process needs to be automated for full mechanization of the overall design process.

VHSIC Hardware Description Language (VHDL) was developed to serve as a standard hardware description language with which design data can be communicated between engineers. Another objective of the development of VHDL is the integration of design automation tools through the use of a common language [17]. In this paper, we present a system which inserts BIST Hardware to a circuit described in VHDL. Among the various well-known Design For Testability (DFT) techniques, use of the Built-In Logic Block Observer (BILBO) [1] is primarily pursued due to its practicality and wide applicability. In the following sections, we will first review two existing systems for automatic DFT. We then describe the tasks of Automatic Insertion of BIST Hardware (AIBH) considered in our system. Next, an appropriate VHDL modeling style for AIBH is described. Finally, a detailed procedure for performing each task is given.

2. Previous Work

Several systems for automatic DFT have been reported in the literature [2][3][6][8-10]. Of these, the two most relevant systems are briefly reviewed here. The first system is the Testable Design Expert System [2][12]. The basic strategy of the system is to find a suitable DFT scheme for each part of a circuit. These DFT schemes are called Testable Design Methodologies (TDMs). Examples of TDMs are Scan-Path, LSSD, BILBO, and several methods for testable design of PLAs. This system first partitions the circuit into several parts called "kernels". Then, an applicable TDM is sought for each kernel using a matching process. In the matching process, the I-path concept is used. An I-path is a path through which data can be transferred without any change in data. This I-path is used for identifying possible registers and paths for test pattern generation and response evaluation. If more than one TDM is applicable to a kernel, TDM measures are used to choose between them. TDM measures are estimates of the costs associated with using a TDM. Frames are used for representing knowledge about TDMs. The TDES system has been implemented in LISP.

Another system proposed by Jones and Baker [3] uses AI techniques, including rule-based system and planning, for high-level BIST design. This system relies heavily on the BILBO-based style of testing. Insertion of BILBO is guided by constraints on hardware overhead and testing time which are specified by the designer. The basic strategy employed by the system is to insert an appropriate number of BIBLOs so that the given constraints on hardware overhead and testing time are satisfied. The system presents a range of test plans which meet the constraints imposed by the designer and achieve a balance between area penalty and test time. Each plan satisfies the constraints in different ways. From the set of possible test plans, the designer can select a suitable one that best matches the original design considerations. This system has been developed using the LOOPS multiparadigm programming environment which supports object-oriented, rule-based, procedural and access-oriented programming.

Different AI techniques are used in the implementation of these two systems. However, there are some similarities between the two systems. First, both systems consider the insertion of BIST hardware at the register transfer level. Second, random pattern testing based on the BILBO architecture is used for the testing of Combinational Logic Blocks (CLBs). Third, BILBO insertion is performed locally rather than globally in both systems. Neither of the two systems uses VHDL in the input description of the hardware.

3. Tasks for Automatic Insertion of BIST Hardware

Many of the ideas used in the two systems described above such as the use of BILBO and the I-path concept are used in our system. However, the major difference is the use of VHDL. Given a VHDL description of a design the structure of the design is extracted first. Then the system allocates the testing resources such as Pseudo Random Pattern Generators (PRPGs) and Multiple Input Signature Registers (MISRs) to test CLBs in the design. Resource allocation is followed by the scan-path organization. A scan-path is used to collect signatures and to test registers in the design. Next, depending upon the resource allocation, a test session for each CLB is scheduled. Finally, control signals for distinguishing between several functional modes are distributed to the multi-functional registers in the design. In the following paragraphs, each task performed by our system will be described in detail.

BIST is a form of structural testing. To insert BIST circuitry into an existing design, the structure of the design should be extracted from the description of the hardware and represented internally in a useable database. In our system, Prolog is used for the internal representation of a design since we need a representation which can be processed by a rule-based system. Therefore, the first task performed is the translation of a VHDL description into a Prolog description.

In order to test a CLB using BILBOs, a structural configuration, PRPG --> CLB --> MISR, should be constructed. The symbol --> implies an I-path between two entities. Since PRPG and MISR can be configured from existing registers, any available resource from the orginal design should be used. However, there may not be registers available which can form the configuration: Register -> CLB -> Register. In this case, it may be necessary to add extra registers to the original design. Therefore, the second task of AIBH is the allocation of existing registers to act as PRPGs and MISRs for testing CLBs. Subsequently, it will be indicated where hardware must be added to perform PRPG and MISR tasks.

Another DFT technique used in our system is register scanning. When this technique is employed, testing of a sequential circuit is reduced to the testing of CLBs and it is performed by using PRPGs and MISRs. The third task of AIBH is scan-path organization.

The next task is test scheduling. Test scheduling is an arrangement of testing sessions so that the total testing time can be minimized given the fixed allocation of PRPG and MISR. The final task of AIBH is the distribution of register control signals. Since registers are used in several functional modes, control signals are needed for the proper configuration of registers in each test session.

In summary, the tasks of AIBH considered in our system are as follows:

1. Translation of VHDL description into Prolog description.
2. Allocation of PRPG and MISR for each CLB in a design.
3. Scan-path Organization.
4. Test Scheduling.
5. Distribution of register control signals.

In order to accomplish these tasks, the input description of a design must provide all of the necessary information. In the next section we will discuss an appropriate VHDL modeling style for implementing AIBH.

4. VHDL Modeling Style for AIBH

The level of possible descriptions in VHDL ranges from system level to gate level. It also allows for modeling a design in many different styles, e.g., behavioral modeling and structural modeling. Therefore, in order to use VHDL for AIBH, it is necessary to define a modeling style and level of abstraction appropriate for accomplishing the tasks of AIBH. From the previously listed tasks for AIBH, we can reach the following conclusions about an appropriate VHDL modeling style:

1. The basic elements in BIST, i.e., the PRPG and the MISR, can be configured from the existing system registers. If there is a path from a register to a CLB, that register can be used as a PRPG. Likewise, if there is a path from a CLB to a register, that register can be used as an MISR. This means that we need information about the interconnection between registers and CLBs. Therefore, the most appropriate modeling style is a structural model which is composed of the interconnections between components at the register transfer level. At this level, designs are described in terms of register, bus, MUX, ALU, PLA etc.
2. When random testing is used on a CLB, knowledge of the detailed structure of a CLB is not necessary in order to add BIST circuitry. Therefore, a behavioral model can be used for describing each component. It does not mean that other modeling styles, e.g., dataflow modeling or structural modeling, cannot be used. However, insertion of BIST hardware should be considered before the low level design description is frozen. Thus, a behavioral model is the most appropriate modeling style for describing each component.
3. In order to test a CLB, a path for the configuration, PRPG --> CLB --> MISR, must be set up. Therefore, it is necessary to have information about signals controlling access the data path such as select signals for MUXes and control signals for bus access. All control signals for accessing data paths should be explicitly specified so that control signals for testing each CLB can be derived.
4. A classification or type for each system component must be recognizable from the VHDL description.

These four points describe a modeling style which must be followed to use the AIBH system.

Using this modeling style, an example circuit is described. The data path of TMS 32010 digital signal processor [16] shown in Figure 1 is used as the example circuit. In order to save the space, only a portion of the VHDL description of the circuit is shown in Figure 2. Some control signals which are not shown in Figure 1 are added to the description. As shown, the description is a typical structural decomposition, i.e., component declaration followed by component instantiation. There are two important conventions followed in the description. First, in order to recognize the types of components in the system, prefixes are attached to the name of each component. For example, the prefix "REG" is used for naming all the register components. Second, the types of signals are divided into four classes: data, control, bus and clock signal for identifying control signals. Signals are also classified using prefixes. The prefixes "CTL", "CLK", "BUS" are used for control signals, clock signals and bus signals, respectively. All other signals are assumed to be data signals. In the next section, a detailed procedure for performing each AIBH task will be described.

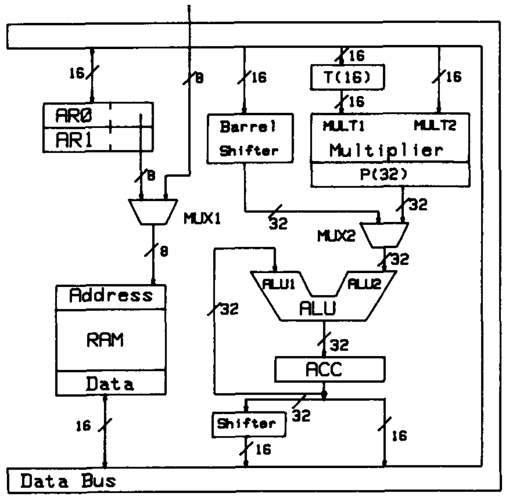

Figure 1. An example circuit.
(Data path of TMS 32010 digital signal processor)

```
---- Interface of TMS32010 Data Path
entitiy TMS_DATAPATH is
    port (ADDRESS_IN    : in  BIT_VECTOR(0 to 7);
          CTL_ALU       : in  BIT_VECTOR(0 TO 3);
           . . .
          CLK           : in BIT);
end TMS_DATAPATH;

---- Structural Decomposition of TMS Data Path
architecture STRUCTURAL of TMS_DATAPATH is

    --- component declaration
    component CLB_MULT
              port (mult1:  in  BIT_VECTOR;
                    mult2:  In  tristate;
                    output: out BIT_VECTOR);
    . . . . . . . . .
    . . . . . . . . .
    component REG_32
              port (data:   in  BIT_VECTOR;
                    clk:    in  BIT;
                    qout:   out BIT_VECTOR);

    ---- local signal declaration
    signal BUS_DATA:     tristate;
    signal TREG_OUT:     BIT_VECTOR(0 to 15);
    . . . . . . . . . . .
    . . . . . . . . . . . .
    signal MULT_OUT:     BIT_VECTOR(0 to 31);

    ---- component instantiation
    begin
       MULT:   CLB_MULT
                 portmap(TREG_OUT,BUS_DATA,MULT_OUT);
       ALU:    CLB_ALU
                 portmap(ACC_OUT,MUX2_OUT,CTL_ALU);
       ACC:    REG_32
                 portmap(MULT_OUT,CLK,PREG_OUT);
       . . . . . . . . . . .
       . . . . . . . . . . .
       MUX2:   MUX_32
                 portmap(BASH_OUT,PREG_OUT,
                         CTL_MUX2,MUX2_OUT);
    end STRUCTURAL;
```

Figure 2. A structural model of the example circuit.

5. Procedure

5.1 Translation of a VHDL Description into a Prolog Description

For AIBH, the structure of a design should be extracted from the VHDL description and stored in the system database. For this purpose, we use a Prolog description of the hardware since we need a representation which can be processed by a rule-based system. There are several methods for describing hardware using Prolog [11]. From these, we decided to use the extensional method [6]. This method is chosen since it closely resembles the VHDL description. As in VHDL, the extensional method description expresses the connectivity between components from a component viewpoint. The Prolog description of a design consists of clauses, each describing the individual modules in a design. The following example shows the clause for describing the P register in the example circuit.

```
preg(reg,[[d,mult,output,data,32],[cp,clk,clk,clk,1]],
        [[q,mux32,input1,data,32]])
```

As shown in the above clause, the predicate "preg" represents the name of the module. A clause has three arguments: the type of module, a list of input ports and a list of output ports. One element of the list of input and output ports is also a list which itself consists of four elements. For example, the list [d,mult,output,data,32] represents the first input port "d" of the module "preg" with the list elements having the following meanings:

- d : the name of the first input port of the module "preg".
- mult : the name of the block connected to the input port "d".
- output : the name of the port of the module "mult" connected to the port "d".
- data : the type of signal at port "d".
- 32 : the size or data path width of port "d".

From this example, it can be seen that a clause for a module has all the connectivity, size and functional information about the module. In VHDL, the primary inputs and outputs are defined in the interface description. Those signals are described as modules in the Prolog description. For example, a primary input port "ADDRESS_IN" in the example circuit is described as

```
address_in(pi,[],[[address_in,mux1,rgt_in,data,7]]).
```

Since there is no input to an input port, the list for the input port is null. The name of the output port is the same as the name of the module since there is only one port. A bus is also defined as a module since it can be considered as a module having inputs and outputs.

These examples give an idea of the contents and format of the Prolog description that can be extracted from the VHDL description. Since the Prolog description has all the information about the structure and is directly processable, the AI type tasks of AIBH can be more readily performed using the Prolog description.

5.2 Allocation of the PRPG and the MISR

During allocation of the PRPGs and MISRs, we use a global embedding technique rather than local embedding in order to maximize resource sharing. In other words, allocation of PRPGs and MISRs for all CLBs is considered simultaneously. The first step is to search all paths from registers to CLBs and CLBs to registers as in [3]. The next step is to allocate PRPGs and MISR to each CLB using the existing registers in the design. Allocation depends on the existence of paths between the CLB and registers. The example circuit has six resources: AR0, AR1, T, P, ACC, ADDRESS, DATA and four CLBs: Multiplier (MULT), ALU, Barrel Shifter (BS) and Parallel Shifter (PS) to be tested. Each port of these four CLBs needs a PRPG or an MISR. The available resources for the ten ports in the example circuit are shown in Table 1. An X in the table indicates that the register can be used as either a PRPG or an MISR for the indicated CLB. In allocating a PRPG and an MISR for a CLB, a decision making procedure is necessary since there could be more than one existing register which can be used as a PRPG or an MISR. For example, there are four registers which can be used as the MISR for the parallel shifter.

Table 1. Covering table for allocation of PRPGs and MISRs.

	PRPG						MISR			
	BS	MULT 1	MULT 2	ALU 1	ALU 2	PS	BS	MULT	ALU	PS
AR0	X	X	X							X
AR1	X	X	X							X
T		X								X
P					X			X		
ACC				X		X			X	
ADD										
DATA	X	X	X							X

The basic strategy used in resource allocation is the minimization of hardware overhead. The major sources of hardware overhead in BILBO-based BIST come from the hardware necessary for implementing PRPGs and MISRs. Therefore, in order to minimize the hardware overhead, the number of PRPGs and MISRs should be minimized. Thus, the objective is to find a minimal set of registers which covers all ports of all of the CLBs in a design. This is the well-known set covering problem and known to be NP-complete. In our system, Garfinkel's a priori deduction and reduction algorithm [14] and tree search algorithm [14] is used to find a minimal cover. The detailed procedure of the algorithm is omitted in this paper. In the example circuit, the minimal cover was found as follows from the covering table in Table 1:

AR0	$-\{T_{BS}, T_{MULT1}, T_{MULT2}, S_{PS}\}$
P	$-\{T_{ALU2}, S_{MULT}\}$
ACC	$-\{T_{ALU1}, S_{ALU}, T_{PS}\}$
uncovered	$-\{S_{BS}\}$

where T_i denotes the PRPG for an input port i and S_j denotes the MISR for an output port j.

There are two problems in the allocation given above. First, S_{BS}, the MISR for the barrel shifter, can be mapped to no available resource. Second, the register ACC is allocated as a PRPG as well as the MISR for the ALU. Since one register cannot be used as PRPG and MISR for the same CLB, a choice should be made between using the register ACC to cover T_{ALU1} or S_{ALU}. These problems show that the resource allocation task cannot be completed using just this algorithm. Since there are several options for unallocated PRPG and MISR, heuristic reasoning is used for completing the resource allocation. This heuristic reasoning part is implemented using a rule-based system. An example of how heuristic reasoning is employed in the resource allocation for the example circuit in Figure 1 is described in the following paragraphs.

In order to reduce hardware overhead, adding extra registers should be avoided if possible. Therefore, the use of existing registers in the design must be considered first even though there is no existing usable path in the design. For example, since the size of register P is 32-bit, it could be allocated as T_{ALU1} or S_{ALU} even though there is no corresponding path. Consider the possibility of using P for T_{ALU1}. In this case, P is used as PRPG for both inputs of the ALU. However, this option is not desirable since the same test patterns will be applied to the two input ports of the ALU. This is different from using AR0 for applying test patterns to both input ports of the multiplier. Since test patterns are applied to the input port MULT1 through the register T, there is a clock delay between patterns at the two input ports of the multiplier. Thus, these patterns can be used to test MULT. Another option is to use the register P as S_{ALU}. However, the register P is already allocated for T_{ALU2}. Thus, this option is not possible either. As a result, adding an extra register is unavoidable. If an extra register needs to be added, it is desirable to add an MISR rather than adding a PRPG. The reason is that if a PRPG is added, an extra MUX and a select signal for the extra MUX must also be added. Therefore, the MISR called EXTRA is added and allocated as S_{ALU} and ACC is allocated as T_{ALU1} .

To cover the unavailable S_{BS} , there are two options, use the register P or use the register EXTRA. In order to use register P to cover S_{BS}, a MUX must be placed at the input of register P. This increases the delay of the multiplier. If the register EXTRA is used as S_{BS}, an additional MUX is still needed, but there is no performance degradation. Based on the above reasoning, resources are allocated as shown in Table 2.

This resource allocation along with the path and conditions for forming the testing configuration are recorded in the system database. For example, the path and conditions for testing the barrel shifter are as follows:

- Path : AR0 --> DATA_BUS --> BS --> NEW_MUX --> EXTRA
- Conditions :
  - DATA_BUS should be driven by the register AR0.
  - The output of BS should be selected at NEW_MUX.

The paths for testing four CLBs in the example circuit are shown in Figure 3. This information is used to perform scan-path organization, test scheduling and control signal distribution.

Table 2. Allocation of PRPGs and MISRs.

	PRPG	MISR
Multiplier	**AR0 AR0**	**P**
ALU	**ACC P**	**EXTRA**
Barrel Shifter	**AR0**	**EXTRA**
Parallel Shifter	**ACC**	**AR0**

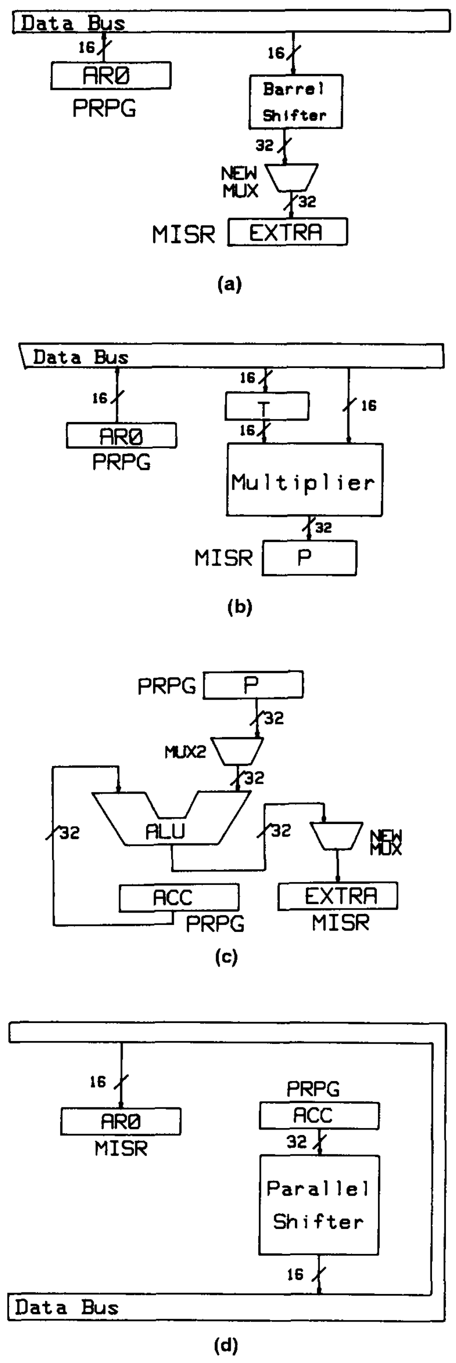

Figure 3. (a) Test of the barrel shifter.
(b) Test of the multiplier.
(c) Test of the ALU.
(d) Test of the parallel shifter.

5.3 Scan-path Organization

In scan-path organization, an attempt is made to set up a scan-path so that all of the registers in a design are configured as a single shift register chain. Under this assumption, the problem of scan-path organization is to find an optimal order of registers so that the wire length of a scan-path can be minimized. This is the well-known shortest path problem on directed graphs. On a graph, a node corresponds to a register and the cost of an edge corresponds to a geometrical adjacency between registers.

We formulated the problem as the all-pairs shortest paths problem [15] and Floyd's algorithm [15] is used for solving the problem. For each pair of registers, the shortest path which goes through all of the registers is found and the length of the path is calculated. Then, the path with the shortest length is chosen. Information on register adjacency is obtained either from user specification or from a back-annotation from a layout.

For the example circuit, the information about the geometrical adjacency between registers is not available. Thus, we arbitrarily assigned the values of geometrical adjacency between registers and the algorithm is applied. The result of the scan-path organization and resource allocation is shown in Figure 4.

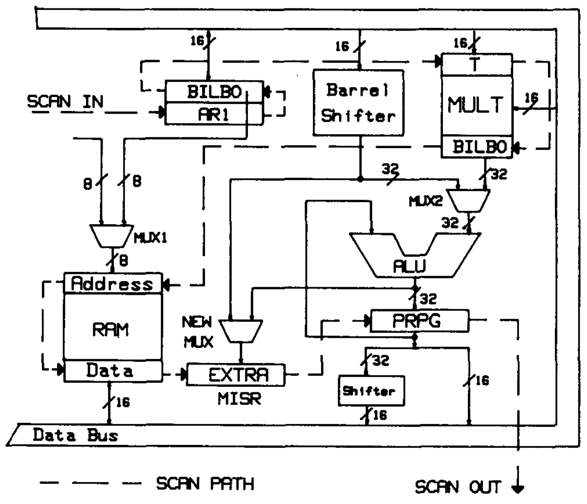

Figure 4. A modified design for BIST of the example circuit.

5.4 Test Scheduling

Concurrent testing of two CLBs is possible if there is no resource sharing conflict in testing two CLBs. Resource conflicts can occur in any type of component such as a register, a bus or a MUX. For example, the parallel shifter and the barrel shifter cannot be tested in parallel because of the sharing of register AR0 and the Data Bus.

In our system, test scheduling is done based on the algorithm proposed by Kime and Saluja [4]. In the algorithm, a Test Compatibility Graph (TCG) is first constructed according to the sharing of resources. In a TCG, a node represents a CLB. An edge between two nodes indicates that testing of two CLBs can be performed simultaneously. One thing that should be mentioned in constructing a TCG is that sharing of PRPG should not be considered as a resource sharing conflict since two CLBs can be tested by the same random test patterns. For example, BS and MULT can be tested simultaneously although the PRPG AR0 and Data Bus are used for testing both CLBs. A TCG for the example circuit corresponding to the resource allocation in Table 2 is shown in Figure 5. Once the TCG is constructed, the test scheduling problem becomes the clique partitioning problem since the minimum number of test sessions corresponds to the minimum number of cliques which can cover all the nodes in the graph. However, the basic clique partitioning algorithm does not produce an optimal solution since the testing time of each CLB is different [4]. Therefore, we used the ordered clique partitioning algorithm [4] for test scheduling. The scheduling result for the example circuit is shown in Table 3. From the table, it can be seen that the testing of the four CLBs is performed in three test sessions.

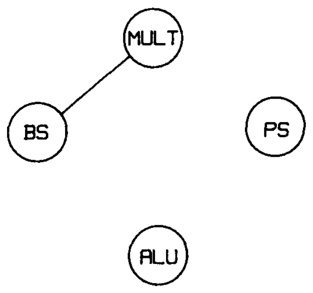

Figure 5. Test compatibility graph for the example circuit.

Table 3. Scheduled test sessions.

	Tested CLB	AR0	P	ACC	EXTRA
Session 1	BS MULT	PRPG	MISR	X	MISR
Session 2	ALU	X	PRPG	PRPG	MISR
Session 3	PS	MISR	X	PRPG	X

(X indicates that the register is not used in that test session)

5.5 Control Signal Distribution

In each test session, a register operates in one of the four functional modes, PRPG, MISR, Scan, and Parallel Load (PL). Since two bits are enough to control these four functional modes, 2n control lines are sufficient to control n registers. The number of control lines can be reduced by finding registers which can be controlled by the same control signals. In our system, this reduction of the number of control lines is performed using the procedure proposed by Kalinowki et al. [5].

As described in [5], n+1 control lines are needed to control n BILBOs assuming that Scan and PL operations are performed simultaneously. One line, say C_1, which is distributed to all BILBOs, is used to distinguish between PL/Scan and PRPG/MISR and n additional lines, say C_2^i, i = 1,2,...,n, are used to distinguish between PL and Scan or PRPG and MISR for each BILBO. Under this assumption, the procedure is concerned with the reduction of the number of C_2 control lines. We will illustrate the procedure using the test session arrangement in Table 3.

The procedure is simply to find groups of BILBOs, called "control partitions", which can be controlled by the same C_2 control lines. In finding a control partition, addition of local control logic to each BILBO is assumed. For example, two BILBOs, AR0 and P, can be controlled by the same C_2 line, called "comparable", although the operational mode in each test session is different. An example of two BILBOs which are not comparable is shown in Table 4. These two BILBOs cannot be controlled by the same signal even with local control logic since the value of C_2 should be the same in session 2 and session 3 but should be different in session 1.

Table 4. Two incomparable BILBOs.

	BILBO 1	BILBO 2
Session 1	PRPG	MISR
Session 2	PRPG	PRPG
Session 3	MISR	MISR

One thing that should be mentioned about control signal distribution is that only BILBOs, i.e., registers which are used as both PRPG and MISR, need to be considered in finding control partitions. There are three types of registers in BIST other than the BILBO. The first type is a register which is used as a PRPG or an MISR, but not both. For example, the register ACC is used only as a PRPG. For this type of register, we do not need to distinguish between PRPG and MISR. Therefore, the C_2 signal of any BILBO can be used to control this type of register. The second type is a register which is used only in the testing mode. An example is the register EXTRA. Since this type of register has just two operational modes, MISR/PRPG and scan, the register can be controlled by C_1 signal. The last type is a register which is not used in testing CLBs such as AR1, T and DATA in the example circuit. Since we are assuming the scanning of all the registers in a design, there are two functional modes, Scan and PL, in this type of register. This type of register can be controlled by the C_2 signal of any BILBO with local control logic.

In the example circuit, there are two BILBOs, AR0 and P, and they are comparable. Therefore, we need only two control signals to control all the registers in the example circuit.

6. Conclusions, Status and Future Work

We have described a system for automatic insertion of BIST hardware. We use VHDL as the modeling language so that the system can be easily integrated with other design automation tools targeted to the use of VHDL. A parser is written for preprocessing VHDL descriptions. The parser is coded in C. Currently, the preprocessor works for restricted VHDL modeling style. Structural information is presently extracted directly from VHDL source code. In order to allow more flexibility in modeling, we are planning to use the IVAN form (VHDL's intermediate form) instead of VHDL source code in preprocessing. Thus, in a future version of the system, the VHDL Analyzer will be used as the first step in preprocessing.

BILBO is the primary DFT technique pursed in the system. We use algorithmic and rule-based approaches for the insertion of BILBO. A rule-based system is presently used for allocation of testing resources which cannot be allocated by the algorithm. Besides the insertion of BILBO, test scheduling and control signal distribution are performed using the algorithms. The algorithms and rule-based system are implemented using Prolog.

The most important feature that the system lacks is flexibility in implementing the BIST architecture. The rule-based system is being modified so that several BIST architectures based on random pattern testing [18] can be considered. An approach being used for this purpose is the constraint satisfaction approach as in [3].

Another issue is the testing of special structures such as memory and PLA. The testing of these special structures will also be considered in the rule-based system. In particular, we are considering the use of BIST PLA rather than testable design of PLA.

The last issue is the post processing of a modified design. In order to complete the lower level design process and verification of the modified design, the modified design should be translated back to VHDL. This post processing is also one of our future tasks.

7. References

1. B. Koenemann, J. Muncha and G. Zwiehoff, "Built-In Logic Block Observation Techniques," Proc. IEEE Int'l Test Conf., 1979, pp. 37-41.

2. M.S. Abadir and M.A. Breuer, "A Knowledge-Based System for Designing Testable VLSI Chips," IEEE Design and Test of Computers, Aug. 1985, pp. 56-68.

3. M.A. Jones and K. Baker, "An Intelligent Knowledge-Based System Tool for High-Level BIST Design," Proc. IEEE Int'l Test Conf., 1985, pp. 743-746.

4. C.R. Kime and K.K. Saluja, "Test Scheduling in Testable VLSI Circuits," Proc. IEEE Int'l Test Conf., 1982, pp. 406-412.

5. J. Kalinowki, A. Albicki and J. Beausang, "Test Control Signal Distribution in Self-Testing VLSI Circuits," Proc. ICCAD, 1986, pp. 60-63.

6. P.W. Horstmann, "A Knowledge-Based System Using Design for Testability Rules," 14th Int'l Symposium on Fault-Tolerant Computing, 1984, pp. 278-284.

7. *VHDL Language Reference Manual Version 7.2*, IR-MD-045-2, Intermetrics, 1985.

8. T.E. Mangir, "EXACT: An Expert System for Testable Design of VLSI," *Int'l Symposium on VLSI Technology*, Systems and Applications, May 1985.

9. A. Kransniewski and A. Albicki, "Automatic Design of Exhaustively Self-Testing Chips with BILBO Modules," Proc. IEEE Int'l Test Conf., 1985, pp. 362-371.

10. H.S. Fung and S. Hirschron, "An Automatic DFT System for the SILC Silicon Compiler," IEEE Design and Test of Computers, Feb. 1986, pp. 45-57.

11. G. Cabodi, P. Camurati and P. Prinetto, "The Use of Prolog Specification and Verification of Easily Testable Designs," Proc. 16th Int'l Symposium on Fault-Tolerant Computing, 1986, pp. 390-395.

12. M.S. Abadir and M.A. Breuer, "Test Schedules for VLSI Circuits Having Built-In Test Hardware," IEEE Trans. Computers, Vol. C-35, No. 4, Apr. 1986, pp. 361-367.

13. F.F. Tsui, *LSI/VLSI Testability Design*, McGraw-Hill, 1987.

14. N. Christofides, *Graph Theory: An Algorithmic Approach*, Academic Press, New York, N.Y. 1975.

15. A.V. Aho, J.E. Hopcroft and J.D. Ullman, *Data Structures and Algorithms*, Reading, MA: Addison-Wesley, 1982.

16. K-S. Lin, G.A. Frantz and R. Simar, Jr., "The TMS320 Family of Digital Signal Processors," Proceedings of the IEEE, Vol. 75, No. 9, Sep. 1987, pp. 1143-1159.

17. M. Shahdad, "An Overview of VHDL Language Technology," Proc. 23rd Design Automation Conf., pp. 320-326.

18. Y.M. El-Zig, "S^3: Self-Test Using Signature Analysis and Scan Path Techniques," Proc. ICCAD , 1983, pp. 73-76.