A Heuristic for Wiring-Aware Built-In Self-Test Synthesis

Abdil Rashid Mohamed, Zebo Peng and Petru Eles
Department of Computer and Information Science,
Linköping University, S-581 83, Sweden.
{abdmo, zebpe, petel}@ida.liu.se

Abstract

This paper addresses the problem of BIST synthesis that takes into account wiring area. A technique for minimizing BIST hardware overhead is presented. The technique uses results of symbolic testability analysis to guarantee testability of all modules in the design. New behavioral-level BIST enhancement metrics are used to guide synthesis in such a way that the number of testability enhancements is minimized. The technique is not only fast but also adds low BIST overhead.

Keywords: BIST insertion, test synthesis, wiring area.

1. Introduction

In deep-submicron VLSI implementation, wiring can take substantial amount of the total chip area. With the development of microelectronics technology, there is a clear trend towards deep submicron implementation, where the interconnecting wires dominate the silicon area cost. It is, therefore, very important to consider the wiring effect in the future deep submicron VLSI implementations.

Since exact wiring information is only available after physical design steps such as floor plan and placement are performed, most of the existing high-level built-in self-test (BIST) and other test synthesis approaches usually do not consider wiring effect. These BIST synthesis approaches, which do not consider the impact of placement of the functional and BIST modules lead to designs which are optimal in terms of the numbers of functional units and BIST resources, but takes more silicon area to implement since the interconnects take a lot of silicon space. Therefore, it is important to take floor planning and wiring cost into account during the BIST synthesis process. To get area efficient designs, the impact of wiring area contribution should be addressed as early as possible so that both functional, BIST and wiring areas can be simultaneously optimized. In this way, resulting designs are likely to be better in terms of total area as compared to the case when wiring is ignored during the synthesis.

Alvandpour and Svensson [1] have developed a heuristic to estimate wiring lengths at register transfer (RT) or higher level of abstraction. Their approach makes use of a few technology dependent parameters, which can be extracted from technology libraries. The approach was later deployed by Hallberg et al. [5] to predict area increase due to wiring in a high-level synthesis system under local timing constraints. Recently, Goel and Marinissen [4] have proposed a model for wiring length computation for core based system-on-chip testing, where they assume the layout of the modules to be known beforehand.

The problem of optimizing BIST insertion at the behavioral and RT levels while taking into account geometrical information of the design has been addressed by us in [6]. There positions of all modules on chip are estimated, and the wire lengths of all interconnections are computed using the technique discussed in [1] and [5]. The area of the design is then computed taking into account the position of the modules, wire lengths and the number of metal layers that are used for wiring with a given VLSI process technology. Simulated Annealing [9] is used to solve the overall BIST synthesis problem. The approach results in very good designs in terms of area since both geometrical information and testability are simultaneously taken into account during BIST synthesis process. On the other hand, since computation intensive Symbolic Testability Analysis (STA) [7], [2], [3] is performed in each optimization loop, the whole approach is very slow.

The aim of this work is to propose and develop a fast and accurate heuristic for addressing the problem of wiring-aware BIST synthesis optimization. The heuristic simultaneously takes into account geometrical information (wiring) and testability during BIST synthesis, hence results in near-optimal designs in terms of *realistic* area cost. It also addresses the drawbacks of the Simulated Annealing based BIST synthesis approach presented in [6].

The exact problem is formulated as follows: given a design represented as a scheduled data flow graph (SDFG) along with allocation/binding information, insert BIST

modules into the design such that all functional modules are self-testable and the total design area is minimized.

The rest of the paper is organized as follows. In Section 2 a short description of our design transformations for BIST is given. In Section 3 some preliminary background information for our approach is provided. In Section 4 our BIST synthesis heuristic is described in detail. Experimental results are presented in Section 5 and some conclusions are drawn in Section 6.

2. BIST Transformations

The main idea of the BIST synthesis approach proposed in this paper is based on a set of BIST design transformations (testability moves). Two classes of transformations have been defined. The first class, known as *conversion for BIST*, provides ways of modifying existing functional registers to give them the capability of test pattern generation or test response analysis. The second class of transformations, known as connection for BIST, provides ways to connect existing BIST registers to uncontrollable or unobservable modules. In doing so, controllability or observability of the given module can be improved.

To illustrate the transformations for BIST we use a very simple RT design, which is obtained from a DFG by allocating a separate module for each DFG operation and a separate register for each DFG variable. This example design is depicted in Figure 1. To make the design self-testable, registers *u*, *x*, c_3, *y* and *dx* are converted to TPGs (Test Pattern Generators) and registers t_4, u_1, y_1 and x_1 are converted to MISRs (Multiple Input Signature Register). These conversion transformations are enough to make the design fully self-testable. However, alternative solutions can be obtained if connection transformations are also applied. For example, by making connections shown with thick dashed lines, it is possible to observe modules **3*, *+1* and *+2* at the MISR M_2 instead of MISRs M_1, M_4 and M_3 respectively. These connection transformations introduce wiring and multiplexer overhead, but in this case MISRs M_1, M_3 and M_4 can be converted back to functional registers. A BIST synthesis optimization algorithm should use a cost function to decide which enhancement transformations to apply.

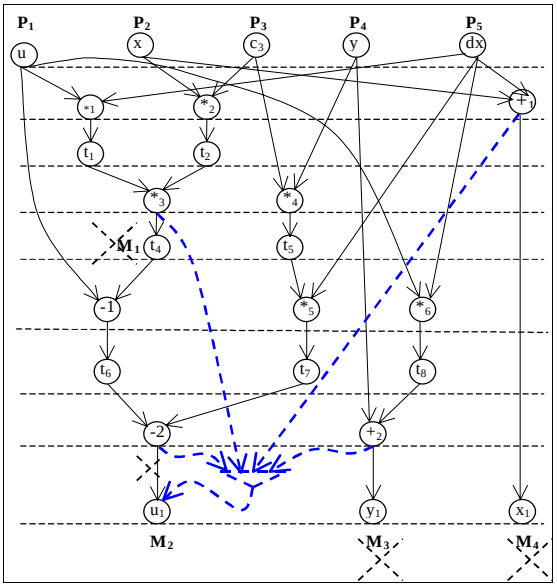

Figure 1: Illustration of BIST transformations

3. Preliminaries

This paper describes a heuristic that performs testability enhancement and guarantees complete testability of each register-transfer (RT) level module while keeping the design area minimum. The heuristic uses behavioral information from a scheduled data-flow graph (SDFG) to do testability analysis of the design and identify operations with testability problems. From now onwards, whenever we use a term DFG we will mean SDFG, unless otherwise explicitly stated. By working at a high-level of abstraction, testability problems can be tackled very early in the design process to avoid expensive and less optimal re-design iterations that can be needed to eliminate testability problems late in the design process. Testability enhancement is performed on the corresponding RT level architectural implementation.

Our definition of testability of DFG operations is based on the use of symbolic testability analysis [2], [3], which asserts an operation to be testable if there is a guaranteed transparent path from on-chip TPGs to the inputs of the operation for supplying pure test patterns, and a transparent path from the output of the operation to an on-chip MISR or BILBO for observing test results. In other words, a DFG operation is testable if its input operands are controllable and its output observable at the *same time*. For the case of the RT designs, a module *m* is considered testable if at least one of the operations mapped to it is testable.

Our approach provides a novel way to quickly explore the design space in search of cheap, yet testable design solutions. Since the optimization problem formulated in Section 1 is NP-hard, a heuristic has been developed which reduces the design space that has to be explored. The proposed heuristic proceeds in the following steps:

A. Controllability enhancement.
B. Observability enhancement.
C. Global testability enhancement.

In each of these steps the following two actions are repeated until complete controllability (step A), observability (step B) and testability (step C) are respectively achieved:

i. Choose a module *m* that is not controllable (or observable, testable respectively).
ii. Visit all possible enhancements for the module *m* and choose the one, which incurs the lowest area overhead.

In order to make the design space exploration efficient, it is important to choose and enhance the modules in such a sequence so as to minimize the overall number of testability enhancements. This is made possible by using our novel *BIST enhancement metrics* (Section 4.1) to help decide in which sequence to enhance untestable modules. Since the heuristic minimizes the overall number of enhancements and the cheapest solution in each enhancement step is chosen, the approach can lead to low cost solutions.

4. Testability Improvement

4.1 BIST Enhancement Metrics

We need to choose uncontrollable (or unobservable, untestable) modules and a sequence in which to enhance them in such a way that the total number of enhancements performed on the design is minimized. We can achieve this objective by ensuring that each time we choose a module to enhance, the enhancement will improve as many other modules as possible.

To solve this problem, we propose an approach, which uses our novel behavioral-level BIST enhancement metrics to guide the testability enhancement process. The *BIST enhancement metrics* are defined below.

Definition 1: *Total Controllability Enhancement Potential (TCEP)* of a given DFG operation or variable node, n_i, is the number of operations and variables whose controllability it can affect.

Controllability of a node n_j can be affected by the controllability of a node n_i if there is a path in the DFG from n_i to n_j and the control step of n_i precedes the control step of n_j. For instance, consider an operation **1* in Figure 2. It can be observed that starting from the operation **1*, it is possible to reach seven nodes namely t_1, **3*, t_4, *-1*, t_6, *-2*, and u_1. Therefore, the value of *TCEP* for the operation **1* is 7.

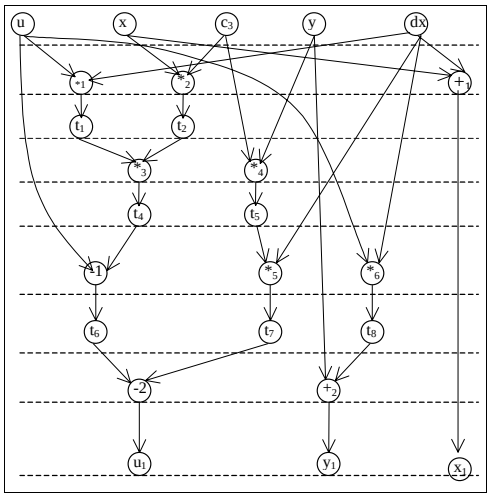

Figure 2: Data Flow Graph Example

Definition 2: *Total Observability Enhancement Potential (TOEP)* of a given DFG operation or variable node, n_i, is the number of operations whose observability it can affect.

Observability of a node n_i can be affected by the observability of a node n_j if there is a path in the DFG from n_i to n_j and the control step of n_i precedes the control step of n_j. For instance, consider an operation *-1* in Figure 2. It can be observed that starting from the operation *-1*, it is possible to traverse the graph upwards and reach 10 nodes namely *u*, t_4, **3*, t_1, **1*, *dx*, t_2, **2*, *x*, and c_3. Therefore, the value of *TOEP* for the operation *-1* is 10.

The BIST enhancement metrics such as *TCEP* and *TOEP* presented so far are computed with reference to the DFG nodes. Controllability, observability and testability enhancements are, however, performed on the RT level architectural representation of the design. Therefore, we need to extend the definitions of the BIST enhancement metrics so that we can apply them to the RT designs.

***Definition 3*:** *RT Total Controllability Enhancement Potential (RTCEP)* of a given RT module, m_i, which implements a set of DFG operations $SM_i = \{op_1, op_2, \ldots,$

Table 1: BIST enhancement metrics: 1-to-1 mapping

Modules	M1	M4	M5	M2	M3	M6	A1	A2	S1	S2
Operation binding	*1	*4	*5	*2	*3	*6	+1	+2	-1	-2
TCEP	7	5	3	7	5	3	1	1	3	1
TOEP	2	2	5	2	8	2	2	5	10	17
RTCEP	7	5	3	7	5	3	1	1	3	1
RTOEP	2	2	5	2	8	2	2	5	10	17

Table 2: BIST enhancement metrics: realistic mapping

Modules	Mult1			Mult2			Add1		Sub1	
Operation binding	*1	*4	*5	*2	*3	*6	+1	+2	-1	-2
TCEP	7	5	3	7	5	3	1	1	3	1
TOEP	2	2	5	2	8	2	2	5	10	17
RTCEP	7			7			1		3	
RTOEP	5			8			5		17	

$op_n\}$ whose respective values of the *TCEP* are given by the set $STCEP_i = \{TCEP_1, TCEP_2, \ldots, TCEP_n\}$ is defined as the maximum *TCEP* value in the set $STCEP_i$, i.e. $RTCEP_i = Max_{j=1}^{n}\{TCEP_j\}$.

Definition 4*: *RT Total Observability Enhancement Potential (RTOEP) of a given RT module, m_i, which implements a set of DFG operations $SM_i = \{op_1, op_2, \ldots, op_n\}$ whose respective values of the *TOEP* are given by the set $STOEP_i = \{TOEP_1, TOEP_2, \ldots, TOEP_n\}$ is defined as the maximum *TOEP* value in the set $STOEP_i$, i.e. $RTOEP_i = Max_{j=1}^{n}\{TOEP_j\}$.

To explain our DFG and RT BIST enhancement metrics, consider an example of a DFG shown in Figure 2. If a *1-to-1* DFG to RT allocation is used, *TCEP* and *RTCEP* metrics are the same. Similarly, *TOEP* and *RTOEP* metrics are the same (see Table 1).

Suppose however, a more realistic allocation as shown in row 1 and row 2 in Table 2 is used. Row 3 shows *TCEP* values and row 4 shows *TOEP* values for the DFG. After applying the definitions above, the values of *RTCEP* and *RTOEP* are shown in rows 5 and 6 respectively in Table 2.

Once testability analysis has identified a set of modules that have to be enhanced, we use the BIST enhancement metrics in order to decide which particular module out of them is to be enhanced first. The actual metric we use is *RTCEP* for the case of controllability and *RTOEP* for the case of observability. For the case of controllability enhancement, our criterion is to prioritize enhancement of the module, which has the greatest value of the *RTCEP* among all uncontrollable modules. Similarly, for the case of observability enhancement, we prioritize enhancement of the module, which has the greatest value of the *RTOEP* among all unobservable modules.

Let us now consider a more general description of using our BIST enhancement metrics. We discuss the enhancement procedure with respect to controllability enhancement. Suppose that $SM_i = \{op_1, op_2, \ldots, op_n\}$ is a set of operations that are implemented by the RT module M_i. Suppose also that respective values of the *TCEP* for the operations in the set SM_i are given by the set $STCEP_i = \{TCEP_1, TCEP_2, \ldots, TCEP_n\}$. Since any RT level functional module M_i implements one or more DFG operations, it follows that $|SM_i| \geq 1$ and $|STCEP_i| \geq 1$. Suppose that after testability analysis is performed on the design, the set of uncontrollable RT modules is found to be $URT = \{M_1, M_2, .., M_m\}$. An uncontrollable RT module $M_x \in URT$ is chosen to be enhanced if there is an operation $p_y \in SM_x$, which it implements such that the operation p_y has the greatest value of *TCEP* among all the operations that are in the union set $STCEP_1 \cup STCEP_2 \cup \ldots \cup STCEP_m$. This means that the module that is enhanced first is the one, whose *RTCEP* value satisfies the criterion, $MaxRTCEP = Max_{i=1}^{m}\{Max_{j=1}^{n}\{TCEP_{i,j}\}\}$.

4.2 BIST Synthesis Heuristic

A general overview of our BIST synthesis heuristic is depicted in Figure 3. In the first step, all modules are made controllable, while in the second step all modules are made observable. After controllability and observability are enhanced, testability of the design is re-checked. If there are still some untestable modules, then their testability has to be enhanced. There should be only a few modules that will remain untestable after the controllability and observability enhancement steps are finished. We remind that a module is considered testable if it is simultaneously controllable and observable. It is, therefore, possible that a controllable and observable module can be untestable. This can happen if a DFG variable is required to be set to different values at the same time, one for enabling controllability and another for enabling observability. Consequently, the associated module becomes untestable since two different values cannot be set to the same variable at the same time [7].

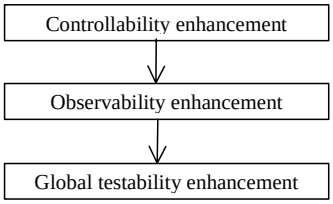

Figure 3: Steps of BIST synthesis heuristic

Our controllability enhancement algorithm, shown in Figure 4, takes as input an SDFG and allocation information and returns a fully controllable RTL design. In a similar way, the algorithm depicted in Figure 5 is used to enhance observability.

Symbols and notations used in the *pseudo-code* of our algorithms are described as follows: *R* is the RTL design representation, *G* is the corresponding SDFG of the design, and *A* is the allocation information depicting the relationship between *G* and *R*. *UCP* and *UOP* are respective sets of all uncontrollable and unobservable operations. They are obtained by performing testability analysis of the SDFG. *UCM* and *UOM* are respective sets of all uncontrollable and unobservable *RTL* modules. They are computed based on the definition of *RTL* module controllability and observability as presented in Section 3.

Procedure *GetTCEP(G, p_i)*, where $p_i \in UCP$**,** computes the *TCEP* value for the operation p_i. *DFGEP* is the set consisting of *TCEP* values of all the uncontrollable or unobservable operations in the DFG. Procedure *GetRTCEP(m_i, DFGEP, A)*, where $m_i \in UCM$**,** computes the *RTCEP* value for the module m_i. *RTEP* is the set consisting of *RTCEP* values of all the uncontrollable modules.

Algorithm: EnhanceControllability

Begin:
Controllable := False;
While *Controllable = False* **do**
DFGEP := ϕ; RTEP := ϕ;
UCP := STA(G);
UCM := UncontrollableModules(UCP, R, A);
If *UCM = ϕ* **then**
Controllable := True;
Else
For *i ← 1, 2, , ..|UCP|* **do**
$t_i := GetTCEP(G, p_i) \mid p_i \in UCP;$
$DFGEP := DFGEP \cup \{ t_i \};$
end for
For *i ← 1, 2, , ..|UCM|* **do**
$t_i := GetRTCEP(m_i, DFGEP, A) \mid m_i \in UCM;$
$RTEP := RTEP \cup \{ t_i \};$
end for
MTE := ModuleToEnhance(UCM, RTEP);
ψ := ControllEnhancements (MTE, R);
C := ϕ;
For *i ← 1, 2, , ..|ψ|* **do**
$C_i := EnhancementCost(E_i) \mid E_i \in \psi;$
$C := C \cup \{ C_i \};$
end for
$SE := E_i \in \psi \mid \cos t(E_i) = Min_{j=1}^{|C|}\{C_j\};$
R := Modify(R, SE);
end if
end while
End.

Figure 4: Controllability enhancement

Algorithm: EnhanceObservability

Begin:
Observable:= False;
While *Observable= False* **do**
DFGEP := ϕ; RTEP := ϕ;
UOP := STA(G);
UOM := UnobservableModules(UOP, R, A);
If *UOM = ϕ* **then**
Observable := True;
Else
For *i ← 1, 2, , ..|UOP|* **do**
$t_i := GetTOEP(G, p_i) \mid p_i \in UOP;$
$DFGEP := DFGEP \cup \{ t_i \};$
end for
For *i ← 1, 2, , ..|UOM|* **do**
$t_i := GetRTOEP(m_i, DFGEP, A) \mid m_i \in UOM;$
$RTEP := RTEP \cup \{ t_i \};$
end for
MTE := ModuleToEnhance (UOM, RTEP);
ψ := ObserveEnhancements(MTE, R);
C := ϕ;
For *i ← 1, 2, , ..|ψ|* **do**
$C_i := EnhancementCost(E_i) \mid E_i \in \psi;$
$C := C \cup \{ C_i \};$
end for
$SE := E_i \in \psi \mid \cos t(E_i) = Min_{j=1}^{|C|}\{C_j\};$
R := Modify(R, SE);
end if
end while
End.

Figure 5: Observability enhancement

The procedure *ModuleToEnhance(UCM, RTEP)* searches for a suitable module to be enhanced, *MTE*. Procedure *ControllEnhancements(MTE, R)* returns *ψ*, which is the set of all possible enhancements for the uncontrollable module to be enhanced (*MTE*). The procedure *EnhancementCost(E_i)* returns the cost of applying the enhancement E_i. *C* is a set, which stores the costs of all the potential enhancements for the module *MTE*. The procedure *Modify(R, SE)* uses the selected enhancement $SE \in \psi$, to modify the RT design.

Many of the notations used in the controllability enhancement algorithm are also used in the observability enhancement algorithm, Figure 5. In addition, this algorithm deploys a procedure *GetTOEP(G, p_i)*, where $p_i \in UOP$, to compute *TOEP* value for the operation p_i and procedure *GetRTOEP(m_i, RTEP, A)*, where $m_i \in UCM$, to compute the *RTOEP* value for the module m_i. In the observability enhancement algorithm, the set *RTEP* consists of *RTOEP* values for all the unobservable modules. *ObserveEnhancements(MTE, R)* is the procedure, which finds all potential observability enhancements *(ψ)* for the unobservable module to be enhanced *MTE.*

4.2.1 Enhancement Selection

We need to get the cheapest solution when a given module to enhance has been decided.

Let $M = \{m_1, m_2, .., m_k\}$ be a set of *k* functional modules that compose an RT level design. Suppose that *PTD* represents a partially testable RT design at a certain moment during our controllability (or observability, testability) enhancement process. Suppose that after testability analysis is performed a module $m \in M$ is selected for enhancement (see enhancement algorithm). Such a module can have multiple controllability (observability, testability) enhancement options that can be used. For example convert its input register to a TPG, connect its input to an existing TPG or built-in logic block observer (BILBO) and so on. Suppose that $E = \{e_1, e_2, \ldots, e_n\}$ is a set consisting of *n* enhancements available for the module *m*. Each of the enhancements $e_i \in E$ is separately applied to the partially testable design *PTD* to get a corresponding enhanced design d_i. Suppose after these enhancements are respectively applied to the partially testable design *PTD* the respective corresponding resulting enhanced designs form a set $D=\{d_1, d_2, \ldots, d_n\}$.

In order to decide which enhancement option *(BIST design transformation)* to use for the module *m* (see Section 2), we evaluate the cost of each improved partially

testable design $d_i \in D$. Out of all the enhancements in the set E, that enhancement $e_i \in E$ which leads to the cheapest improved design $d_i \in D \mid cost(d_i) = \min_{j=1}^{n}\{cost(d_j)\}$ is chosen. The cost that we use is the total design area, which consists of the areas of the functional modules, functional registers, BIST modules, test multiplexers as well as area contribution due to wiring. Wiring area contribution is computed based on the estimation algorithm introduced in [1] and [5].

4.2.2 Global Testability Enhancement

After controllability and observability of all the modules are enhanced, it is still possible for some of them to be untestable. To address this problem, we propose a procedure to fix the remaining testability problems. The algorithm is described in Figure 6.

```
Algorithm: Global Testability Enhancement
Begin:
    // Fix global testability problems
    Ω := φ;
    UTM := UntestableModules(G, R, A);
    While UTM != φ do
        M := FirstUntestable(UTM);
        Enh := Enhance(M, Left, Contr);
        UTM := UntestableModules(G, R, A);
        If UTM != φ then
            DiscardEnhancement(R, Enh);
            Enh := Enhance(M, Right, Contr);
            UTM := UntestableModules(G, R, A);
            If UTM != φ then
                DiscardEnhancement(R, Enh);
                Enh := Enhance(M, Output, Observ);
                UTM := UntestableModules(G, R, A);
                If UTM != φ then
                    Enh := Enhance(M, Left, Contr);
                    Enh1 := Enhance(M, Right, Contr);
                    Ω := Ω ∪{Enh} ∪{Enh1};
                End if
            Else
                Ω := Ω ∪{Enh};
            End if
        Else
            Ω := Ω ∪{Enh};
        End if
        UTM := UntestableModules(G, R, A);
    End for
    // Remove unnecessary BIST overhead
    For i ← 1, 2, , ..|Ω | do
        Ei := GetEnhancement | Ei ∈ Ω;
        DiscardEnhancement(R, Ei);
        UTM := UntestableModules(G, R, A);
        If UTM != φ then
            PutBackEnhancement(R, Ei);
        End if
    end for
End.
```

Figure 6: Global testability enhancement

After all the modules are enhanced and the design becomes testable, it is likely that we have added too much BIST overhead. Therefore, we propose a BIST resources minimization (redundancy removal) phase, whereby we try to remove each enhancement we have added and check if the design remains testable. If the design remains testable after removal, the change is made permanent. Otherwise the enhancement is put back. In this way BIST overhead is reduced while testability is guaranteed. Pseudo-code of our global testability enhancement and redundant BIST hardware removal algorithm is presented in Figure 6.

Symbol Ω represents a set of all testability enhancements that are done on the design. Procedure *UntestableModules(G, R, A)* takes the DFG, the RT design and allocation information, then uses STA to find a list of all the untestable modules, *UTM*. The first untestable module from the list *UTM*, denoted as *M*, is usually the first one to be enhanced.

Procedure *Enhance(M, operand, enhanceType)* adds a BIST enhancement for the module *M*. It is used to enhance output observability or controllability of the left or right input of the module. *DiscardEnhancement(R, Enh)* is used to remove the enhancement, *Enh*, from the design. Procedure *PutBackEnhancement(R, E_i)* puts back the enhancement E_i if its removal renders the design untestable.

5. Experimental Results

Efficiency of BIST insertion approach is evaluated based on the amount of BIST hardware introduced. This is usually computed as the number of TPGs, MISRs, BILBOs and CBILBOs added. Our approach is one of the few which not only shows how many TPGs, MISRs, BILBOs and CBILBOs are added, but also performs quantitative estimation of the wiring cost during the BIST synthesis process. It takes the overall design cost as the optimization objective. Other approaches, since they ignore wiring overhead, do not guarantee efficient designs in terms of total design area.

We have evaluated our approach on several HLS benchmarks. The following technology dependent parameters are used: Wiring pitch (the average width of a 1-bit wire including spacing between the wires) is $0.8\mu m$. The number of metal layers is 2 and wire over routing factor [5] is 0.5.

Sizes of the functional registers and functional modules are extracted from [8]. For the BIST registers, we have assumed a simple relationship between the size of the functional and BIST registers: Register < TPG < MISR < BILBO < CBILBO. The areas of the 16-bit modules used in the experiments are: Multiplier is 250000μm^2, adder is 50000μm^2, subtractor is 50000μm^2, functional register is 15000μm^2, TPG is 20000μm^2, MISR is 30000μm^2,

BILBO is 40000μm^2, CBILBO is 50000μm^2 and multiplexer is 1000 + number_of_inputs * 500.

Characteristics of the designs we used in our experiments are summarized in Table 3. The first set of designs (*Ex2_Simp, Real_Simp, Paulin_Simp, Ovenctrl_Simp* and *Ewf_simp*) has been synthesized using a very simple HLS algorithm such that each DFG operation is implemented using a separate functional module. The second set of designs (*Ex2, Real, Paulin, Ovenctrl and Ewf*) has been synthesized using the algorithm presented in [10].

Our experimental results are summarized in Table 4. Columns *P*, *M* and *B* represent the number of TPGs, MISRs and BILBOs respectively. The column titled "Design Area" represents area of the designs before and after our BIST synthesis heuristic is applied. The column titled "Overhead" shows the hardware overhead of our approach. The last column represents time taken by our heuristic.

The experiments were run on a Sun Solaris workstation with 440MHz CPU and 256MB RAM.

In our experiments, we have taken into account wiring area during the BIST optimization process, as described in this paper. The design cost minimized is the total data path area including area of functional and BIST modules, test multiplexers and *wiring*.

Table 3: Characteristics of the designs

Design name	#Adders	#Subtractors	#Multipliers	#Dividers	#Multiplexers
Ex2_Simp	0	2	5	0	0
Real_Simp	3	2	4	2	0
Paulin_Simp	2	2	6	0	0
Ovenctrl_Simp	5	1	1	1	0
Ewf_Simp	26	0	8	0	0
Ex2	0	1	2	0	7
Real	1	1	1	2	8
Paulin	1	1	2	0	12
Ovenctrl	2	1	1	1	7
Ewf	3	0	1	0	17

Table 4: Experimental Results using our heuristic

Design Name	P	M	B	Design Area (μm^2)		Overhead (%)	CPU time (Sec)
				Before	After		
Ex2_Simp	6	1	1	1735615.58	1810091.63	4.29	84
Real_Simp	6	2	1	2301672.03	2393161.13	3.97	109
Paulin_Simp	5	3	1	2201614.97	2303690.91	4.64	114
Ovenctrl_Simp	6	2	0	1187965.25	1253342.36	5.50	45
Ewf_Simp	10	5	0	4738211.13	4875811.81	2.90	608
Ex2	3	0	1	911001.77	955196.31	4.85	38
Real	2	0	2	1362466.09	1430911.97	5.02	51
Paulin	2	0	2	1114450.72	1183608.25	6.21	42
Ovenctrl	4	0	2	1112465.0	1191276.02	7.08	51
Ewf	3	0	2	1478206.38	1564085.94	5.81	282
Average						5.03	

Table 5: Performance comparison

	Simulated Annealing (Wire considered)		Our Heuristic (Wire considered)		Comparison	
Design Name	Area (μm^2)	Time (Sec)	Area (μm^2)	Time (Sec)	Time reduction (#Times)	Area overhead (%)
Ex2_Simp	1804166	531.99	1810091.63	84	6.33	0.33
Real_Simp	2421214	1085.93	2393161.13	109	9.96	-1.16
Paulin_Simp	2260897	800.66	2303690.91	114	7.02	1.89
Ovenctrl_Simp	1294624	786.07	1253342.36	45	17.47	-3.19
Ewf_Simp	5006007	3118	4875811.81	608	5.13	-2.60
Average					9.18	-0.95

The importance of considering wiring during the BIST synthesis process is already experimentally justified in our previous work presented in [6]. The importance of the work presented in this paper is on getting a faster approach that can be more applicable to realistic large designs instead of the slow Simulated Annealing based approach presented in [6]. We have, therefore, compared our approach with the Simulated Annealing based approach that we presented in [6]. Comparison results are depicted in Table 5. As can be observed from the table, the proposed approach is efficient in terms of run time and, at the same time it also produces good quality results. While run times are on average one order of magnitude faster, the quality of the results produced by the heuristic is on average 1% better than that generated with our implementation based on Simulated Annealing.

6. Conclusions

We have presented a greedy heuristic for wiring-aware BIST synthesis. The approach provides two ways to converge towards testable and cheap solution while keeping computational effort low. It minimizes the overall number of testability enhancements done on the design. This is assisted by our novel BIST enhancement metrics which are used to guide the synthesis process in such a way that each controllability or observability enhancement targets to improve as many modules as possible. This is complemented by a thorough local search of the cheapest solution for each enhancement performed. The cheapest alternative enhancement for a given module is used.

We found out that the heuristic is able to find good solutions at a relatively short computational effort. The heuristic introduces relatively low hardware overhead, 5% on average. It is also one order of magnitude faster compared to a simulated annealing based approach.

7. References

[1] A. Alvandpour and C. Svensson, "A Wire Capacitance Estimation Technique for Power Consuming Interconnections at High Levels of Abstraction", *Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS97)*, Louvain-la-Neuve, Belgium, 1997.

[2] I. Ghosh, N. K. Jha, and S. Bhawmik, "A BIST Scheme for RTL Circuits Based on Symbolic Testability Analysis", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 19, Issue 1*, pp. 111–128, Jan. 2000.

[3] I. Ghosh, N.K. Jha, and S. Bhawmik, "A BIST Scheme for RTL Controller-Data Paths Based on Symbolic Testability Analysis", *Proceedings of the 35th ACM IEEE Design Automation Conference*, pp.554-559, San Francisco, CA; U.S.A., 1998.

[4] S. K. Goel and E. J. Marinissen, "Layout-Driven SoC Test Architecture Design for Test Time and Wire Length Minimization", *Proceedings of Design Automation and Test in Europe*, pp. 738-743, 2003.

[5] J. Hallberg and Z. Peng, "Estimation and Consideration of Interconnection Delays during High-Level Synthesis", *Proceedings of the 24th Euromicro Conference, Vol. 1*, pp. 349-356, Västerås, Sweden, Aug. 1998.

[6] A. R. Mohamed, Z. Peng and P. Eles, "A Wiring-Aware Approach to Minimizing Built-In Self-Test Overhead", *Proceedings of the IEEE International Workshop on Electronic Design, Test and Applications (DELTA 2004)*, pp. 413-415, Perth, Australia, Jan. 28-30, 2004.

[7] A. R. Mohamed, Z. Peng and P. Eles, "BIST Synthesis: An Approach to Resources Optimization under Testing Time Constraints", *Proceedings of the IEEE 5th Design and Diagnostic of Electronic Circuits and Systems Workshop (DDECS2002)*, pp. 346-351, Brno, Czech Republic, April 17-19, 2002.

[8] V.G. Moshnyaga and K. Tumaru, "A Placement Driven Methodology for High-Level Synthesis of Sub-Micron ASICS", *Proceedings of the International Symposium on Circuits and Systems (ISCAS'96)*, pp. 572-575, May 1996.

[9] C. R. Reeves, "Modern Heuristic Techniques for Combinatorial Problems", *Blackwell Scientific Publications*, 1993.

[10] M. Tien-Chien Lee, "High-Level Test Synthesis of Digital VLSI Circuits", *Artech House*, 1997.