A Synthesis-for-Transparency Approach for Hierarchical and System-on-a-Chip Test

Krishnendu Chakrabarty, *Senior Member, IEEE*

Abstract—Test data propagation through modules and test vector translation are two major challenges encountered in hierarchical testing. We propose a new synthesis-for-test approach in which multiplexers are embedded in the behavioral models of the various modules constituting a hierarchical system. This approach can also be applied to system-on-a-chip designs in which synthesizable models are available for the embedded cores. The embedded multiplexers provide complete single-cycle transparency, thereby offering a straightforward yet effective solution to the problems of test data propagation and test vector translation. In order to determine module I/O bitwidths for single-cycle transparency, a global analysis is carried out using a graph-theoretic framework and an optimization method based on integer linear programming. Case studies using high-level synthesis benchmarks and an industrial-strength benchmark show that synthesis for transparency introduces very little area and performance overhead.

Index Terms—Embedded multiplexer, I/O bitwidth, parallel test access, single-cycle transparency, test reuse.

I. INTRODUCTION

HIERARCHICAL test methodologies handle large systems in a divide-and-conquer fashion by relying on precomputed test patterns for each module, which are subsequently translated and applied from the system's primary inputs [16]–[19], [23]. Hierarchical testing techniques offer lower test generation costs and increased test reuse. Tests developed for the individual modules can be reused and there is no need for gate-level test generation for the entire system. However, these techniques must provide mechanisms for justifying the precomputed test vectors for a module from the system inputs to the module inputs. Similarly, the test responses must also be propagated to observable system outputs. This problem is becoming increasingly important with the emergence of core-based system-on-a-chip (SOC) designs [14], [29]. Since the embedded cores in an SOC are not directly accessible from the chip I/Os, the system must devise methods for providing test access.

A number of hierarchical testing techniques have been presented in the literature for justifying test patterns and propagating test responses. Early work in this area was based on the

use of F-paths [12] and I-paths [1], which utilized existing propagation paths between module I/Os and chip I/Os. In [25], a technique based on Boolean difference was presented for test generation and test translation in modular combinational circuits. More recent work has focused on design for testability (DFT) methods. For example, the FScan-BScan method utilizes a combination of full scan and boundary scan [9]. In this method, every core (or module) is made testable by full scan and system-level testability is achieved by isolating all modules using boundary scan. This provides full controllability and observability for every module, and eliminates the need for test translation. However, FScan-BScan introduces high area and delay overheads, and requires enormous test application time due to serial test access. In the related Fscan-Tbus method, full scan is combined with a system-level test bus [9]. However, this method also suffers from area and delay overheads; moreover, the interconnect testing problem cannot be directly addressed using this technique. DFT is also used in [4] to test macro blocks within a circuit using a variety of test techniques. It relies on full, partial, and boundary scan, and uses ad hoc module transparency to reduce test overhead.

To reduce test generation time, a hierarchical test generation strategy was presented in [19]. This approach is based on the notion of *test plans*, which provide control sequences to propagate test data through modules. Recent work on hierarchical testability analysis (HTA) has been directed at the efficient generation of such test plans [16]–[19], [26]. The key idea in these methods is to identify control sequences that allow test data to be *transparently* propagated through modules. DFT enhancements are usually necessary to augment the amount of transparency that can be achieved using HTA. In [14], a DFT approach was presented to make cores in an SOC transparent by extracting their test control/data flow information. While such an approach provides parallel test access and allows any test sequence at the inputs of a module to be propagated to the outputs, it suffers from two main drawbacks 1) high test application time due to the transparency latency (multicycle transparency) required to propagate test data through a module and 2) the test control/data flow must be extracted in order to provide transparency.

Testability-driven behavioral synthesis can produce area-efficient designs with low test-related overhead [23]. However, synthesis for testability approaches have rarely addressed hierarchical or SOC test [5]. These methods have focused primarily on selection of registers for BIST, selection of scan flip-flops, reduction in the number of loops in the datapath, and hierarchical testability enhancement of RTL circuits obtained through behavioral synthesis [10], [13]. In order to provide hierarchical

Manuscript received July 11, 2001; revised February 11, 2002. This work was supported in part by the National Science Foundation under Grant CCR-9875324. A preliminary and abridged version of this paper appeared in *Proc. Int. Conf. VLSI Design*, pp. 431–436, Bangalore, India, Jan. 2001.

The author is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: krish@ee.duke.edu).

Digital Object Identifier 10.1109/TVLSI.2003.810784

Fig. 1. Embedding a multiplexer in the behavioral description of a combinational module *M*. (a) Verilog code. (b) Verilog code with embedded multiplexer.

test capabilities, they typically require behavioral descriptions in the form of control-data flow graphs [5].

Instead of providing complete controllability and observability to core I/Os, more recent work has focused on providing controllability and observability to SOC cores on an “as needed basis” [24]. This is achieved through transparency analysis using nondeterministic finite-state automata, and transparency enhancement through symbolic justification and propagation analysis based on a regular expression framework. However, this approach requires test sequence composition from symbolic tests.

Transparency of modules can be trivially achieved by providing a direct path from each module input to output with the help of multiplexers. However, the area/delay overhead of

Fig. 2. Embedding a multiplexer in the behavioral description of a sequential module *REC*. (a) VHDL code. (b) VHDL code with embedded multiplexer.

such a solution is usually prohibitive. In particular, the interconnect required for routing the additional bypass lines can be excessive. In this paper, we present a synthesis-for-transparency approach that alleviates the above problems. The synthesis method is based on the insertion of multiplexers in the behavioral models of the modules making up the system.

Fig. 3. An example of a system composed hierarchically of synthesizable modules.

These models can describe both combinational and sequential circuits. A behavioral synthesis tool can then be used to derive transparent modules with *embedded multiplexers* that ensure complete reuse of module-level tests and easy system-level test application. The proposed synthesis for testability methodology is also applicable to SOC designs composed of soft cores for which behavioral, synthesizable models are readily available.

A related approach for SOC testing is based on adding a bypass mode using multiplexers and registers [20]. However, this requires packetization of test data, and serial-to-parallel and parallel-to-serial bit-matching circuits—hence, it does not provide single-cycle transparency. In the proposed method for single-cycle transparency, every module in the system is synthesized to operate in two modes—a normal functional mode, and a transparent mode in which all inputs are passed unchanged to the outputs. An external control input is used to select the appropriate mode of operation for a module. In order to test module M , it is set to the functional mode while all other modules are set to the transparent mode. To allow the complete flow of test data through M in its transparent mode, the bitwidths of the I/O ports of M may need to be increased. While full transparency can be ensured by making the input and output bitwidths of M equal, this is often not necessary. The test-related area overhead can be reduced if the bitwidth of M is determined by analyzing the test propagation requirements of the other modules in the system. We formulate this problem using the notion of *test graphs* and determine the bitwidths by solving an integer linear programming model. While the insertion of embedded multiplexers during synthesis is trivial, bitwidth sizing for the various modules in the system is challenging and nontrivial.

The proposed approach offers a number of unique advantages. It provides single-cycle transparency without the addition of registers for bypass or the extraction of control/data flow information. Its conceptual simplicity makes it easy to implement and integrate in the design flow. It does not require any test composition or sophisticated test scheduling algorithms. Precomputed test sets for every module can be readily applied without requiring any transparency latency or test vector translation. These test sets may contain functional vectors, scan vectors, or ordered test sequences for nonscan sequential circuits. The

Fig. 4. (a) System graph for S . (b) Acyclic system graph G for the example of Fig. 5.

test methodology provides parallel test access to the embedded modules, thereby facilitating at-speed test and increasing the

Fig. 5. The example hierarchical system with the feedback loop removed.

coverage of nonmodeled and performance-related defects. The synthesis of explicit transparent paths obviates the need for complex data decoding in test generation for microprocessor circuits. Finally, interconnect testing can be carried out by simply setting all the modules to the transparent mode.

We consider two representative SOC designs as case studies. First, we present experimental results on applying the synthesis-for-test methodology to a nontrivial example system constructed by stitching together several high-level synthesis benchmarks [6], [22]. We show that embedded multiplexers offer a more efficient solution to hierarchical test than the explicit insertion of multiplexers at the gate level. First, we use Synopsys Design Compiler to synthesize each of the benchmarks to be transparent and determine the impact of transparency on area and delay. We then make the overall system transparent by reformulating the transparency requirements for the individual modules. Once again, we determine the effect of the synthesis procedure on the system area and delay. Our results demonstrate that transparency can be achieved with negligible impact on system area and performance. For the overall system, the area increase was 3.2% and the increase in delay (measured by the clock period) was only 1.6%.

We also apply the proposed synthesis approach to two VHDL modules in the LEON core [30]. The LEON core is a hierarchical SPARC-compatible processor developed by the European Space Agency for future space missions. It is available as a highly configurable, synthesizable VHDL model. We show that the two synthesizable modules can be made transparent with no delay overhead and area overhead of no more than 5.2%.

The paper is organized as follows. In Section II, we present the basic concept of transparency for individual modules using embedded multiplexers, and we explain how transparency can be achieved using embedded multiplexers for combinational circuits and for finite-state machines. In Section III, we present our example system composed of several high-level modules and describe the proposed hierarchical test method for this system. We also investigate the problem of determining the bitwidths required for the modules in order to achieve single-cycle transparency. We present experimental results in Section IV.

II. TRANSPARENCY USING EMBEDDED MULTIPLEXERS

Transparency can be achieved by embedding multiplexers in the behavioral models described using a hardware description language such as Verilog or VHDL. For example, consider the Verilog model of a combinational module M shown in Fig. 1(a). M has one 4-bit input port X and one 2-bit output port Z . A transparent behavioral model for M is shown in Fig. 1(b). An additional control input T is used to determine the mode of operation. The control input T is used to switch M from the normal mode ($T = 0$) to a pass-through, transparent mode ($T = 1$). An additional 2-bit output port Y is added to ensure complete transparency of M . Alternatively, the bitwidth of a port can also be increased to provide single-cycle transparency. In general, the bus widths of input ports may also have to be expanded to provide transparent access to other modules in the system. In Section IV, we describe how the overhead of additional ports and associated wiring/interconnect area can be minimized by analyzing the system-level hierarchical testing needs.

Multiplexers can similarly be embedded in the behavioral models of sequential circuits. Fig. 2(a) shows the VHDL model for a finite-state machine (FSM). In order to make it transparent, a multiplexer is embedded in the behavioral model of Fig. 2(b). The FSM operates in the normal functional mode when $T = 0$. However, when $T = 1$, it works in a transparent mode and “out” is connected directly to “in.” Therefore, it now operates as a pass-through combinational circuit. The state transitions of the FSM are not specified for the transparent mode in Fig. 2(b). They can either be interpreted as don’t-cares or the clock to the FSM can be disabled by gating it with T to save power during testing. Since the next-state functions are not affected by the embedded multiplexer, synthesis tools can be expected to provide efficient implementations of transparent FSMs. This approach is similar to the design of testable FSMs based on multiplexer embedding in flip-flops [21].

In high-performance designs, multiplexer embedding should not lead to any performance degradation. For such designs, we first synthesize the high-level modules without embedded any multiplexers. This allows us to determine the critical paths in

Fig. 6. Test graphs and constraints on the edge weights. (a) M_0 (multiplier). (b) M_1 (GCD). (c) M_2 (Barcode). (d) M_3 (Kalman).

the system, and the (critical) primary outputs on which these critical paths terminate. Once this is done, we partition the set of primary outputs of each module into the set of critical outputs C and the set of noncritical outputs NC . Multiplexers are then embedded at the outputs in NC corresponding to the high-level behavioral descriptions of these modules. Unless there is significant retiming during synthesis, multiplexer embedding introduces very little change in the internal latch-to-latch delays within the modules. While this approach does not guarantee that the delay of the synthesized transparent circuit will not exceed that of the original circuit, it increases the likelihood that the performance will not be significantly affected, especially if the synthesis script is run with higher weight assigned to timing optimization. (Most synthesis tools allow the user to assign relative weights to timing and area goals.) A consequence of selective multiplexer embedding is that some amount of test translation is necessary at the outputs of a module that are not fed by an implicit bypass path from the module inputs. The outputs in C cannot be used for transparent test data propagation.

III. SYSTEM-LEVEL TEST STRATEGY

In this section, we describe the hierarchical test methodology using a nontrivial example $\mathcal{S}$ of a system with two 32-bit input ports and two 32-bit output ports and composed hierarchically of several synthesizable modules (Fig. 3). The example was constructed using four high-level benchmark circuits (*GCD*, *Barcode*, *Kalman*, and *am2910*) [22] and a 32-bit combinational multiplier. Each benchmark is a sequential circuit with clock and reset inputs, which are not shown explicitly in Fig. 3. In order to make the example nontrivial and realistic, we introduced a feedback loop and reconvergent fanout, and used bus lines of unequal widths. We also introduced bus truncation and fanout at several places in $\mathcal{S}$.

In order to apply the proposed test methodology to a hierarchical system, we first construct a weighted directed system graph whose vertices are the synthesizable modules in the system and whose edges represent functional interconnections between the modules. The weight of an edge (M_i, M_j) denotes the total width of the buses (including all ports) connecting

M_i to M_j . The system graph for $\mathcal{S}$ is shown in Fig. 4(a). The vertices F_0 , F_1 , and F_2 denote fanout branches in the system. The source and sink vertices in the graph represent system-level primary inputs and primary outputs, respectively. Note that if selective multiplexer embedding is employed, the module outputs that are marked as critical (as described in Section II) do not contribute to the edge weights in the system graph.

Next, we break all cycles in the system graph. This problem is related to the *minimum feedback vertex set* problem, which despite being NP-hard, can be solved efficiently for large problem instances using heuristic methods [2], [8], [11]. In order to reduce the overhead due to system-level I/O pins and interconnect, the feedback loops should be broken in such a way that buses with the least bitwidth are multiplexed to primary I/Os. For many hierarchical systems, this problem is tractable enough to be solved by inspection due to the small problem size. For example, we can see that for $\mathcal{S}$, the cycle can be broken by removing the edge (F_2, M_2) —this implies that the 5-bit input to M_2 must be multiplexed to a primary input. The acyclic system graph G corresponding to Fig. 5 is shown in Fig. 4(b). For notational convenience, the edge weights in G are denoted by $W_1, W_2, \dots, W_{11}$. Ad hoc sharing methods can be used to reduce system-level I/O overhead. For example, X_3 in Fig. 5 can be multiplexed with five lines from the X_2 bus.

The proposed test methodology applies precomputed test sets to the modules in $\mathcal{S}$ in multiple test sessions. Exactly one module is tested in one session—the module under test is set to the functional mode while all other modules are set to the transparent mode. This is achieved using a separate control input for all the modules. This suggests that m control inputs $T_0, T_1, \dots, T_{m-1}$ are necessary for m modules $M_0, M_1, \dots, M_{m-1}$. However, the number of control inputs can be reduced to $\lceil \log_2(m+1) \rceil$ using a decoder since at most one module is tested in any session, which implies that at most one of the T_i 's, $0 \leq i < m$, is 0 in any test session. An additional test session is necessary for interconnect testing—all modules in the system are set to transparent mode for this session.

As discussed in Section II and illustrated in Fig. 1, if a module M 's output bitwidth is less than its input, then additional outputs must be added to M to make it fully transparent. However, this is not always necessary when M is embedded in a larger system. In order to reduce overhead, the increase in the I/O bitwidths of M in a hierarchical system must be carefully minimized by analyzing the propagation requirements of the other modules. The global analysis that we present next leads to a system graph G^* with modified edge weights. In Fig. 4(b), we used $W_1, W_2, \dots, W_{11}$ to denote the edge weights in G . We now use $W_1^*, W_2^*, \dots, W_{11}^*$ for the corresponding edge weights in G^* .

The new edge weights can be determined by generating test graphs for each module in the system. For module M_j , the *test graph* $G_j \subseteq G^*$ contains a vertex M_i if either of the two conditions holds: i) M_i lies on a directed path from the source vertex to M_j , or ii) M_i lies on a directed path from M_j to the sink vertex. In other words, G_j contains M_i if M_i belongs to either the input cone or the output cone of M_j . Similarly, an edge (M_i, M_k) belongs to G_j if it either lies on a path from the source

Fig. 7. (a) Test graph and constraints on the edge weights. M_4 (am2910). (b) The graph G^* with updated edge weights.

vertex to M_j or on a path from M_j to the sink vertex. Figs. 6 and 7(a) show the test graphs for the five modules in $\mathcal{S}$.

For each test graph G_j , we associate a set of constraints on edge weights for the edges in G^* . These constraints, which provide sufficient conditions for transparency, are of two types: i) justification constraints, which ensure that test data from M_j can be transparently propagated from the source vertex to M_j through other modules, and ii) propagation constraints, which ensure that test responses for M_j can be transparently propagated through other modules to the sink vertex. The constraints are obtained as follows.

- 1) *Justification constraints*: If M_i lies on a path from the source vertex to M_j in G_j , then the sum of the weights of the edges directed away from M_i in G_j must not exceed the sum of the weights of the edges incident on M_i . This ensures that the bitwidth at the inputs of M_i is adequate for justifying tests for M_j . If M_i corresponds to a reconvergence point, then it is quite likely that test data for M_j

TABLE I

IMPACT OF EMBEDDED MULTIPLEXERS ON THE AREA AND PERFORMANCE OF HIGH-LEVEL SYNTHESIS BENCHMARKS (DESIGN COMPILER *Isi10k* LIBRARY UNITS):

(a) WIRE LOAD PARAMETER SET TO 0.5 (b) WIRE LOAD PARAMETER SET TO 0.1

Circuit	Original circuit area	Original circuit delay (clock period)	Transparent circuit area	Transparent circuit delay (clock period)	Percentage area increase	Percentage delay increase
<i>barcode</i>	795.76	20.17	822.91	20.85	3.41	3.37
<i>GCD</i>	1327.82	77.40	1375.43	77.29	3.61	-0.14
<i>diffeq</i>	7824.00	81.08	8430	80.85	7.75	-0.28
<i>lru</i>	1913.00	44.78	1919.00	24.94	0.31	-44.30
<i>dhrc</i>	5659.72	76.75	6288.89	77.22	11.09	0.62
<i>kalman</i>	9454.78	72.20	9577.48	72.30	1.08	0.05
<i>am2910</i>	2432.12	30.54	2443.32	40.32	0.45	32.03
Average	—	—	—	—	3.96	-1.24

(a)

Circuit	Original circuit area	Original circuit delay (clock period)	Transparent circuit area	Transparent circuit delay (clock period)	Percentage area increase	Percentage delay increase
<i>barcode</i>	734.00	20.17	753.92	20.85	2.72	3.37
<i>GCD</i>	1228.21	71.45	1243.51	73.41	1.25	-1.32
<i>diffeq</i>	7061.64	81.08	7617.99	80.85	7.88	-0.28
<i>lru</i>	1665.43	25.71	1694.29	25.07	1.73	-2.60
<i>dhrc</i>	5073.4	76.56	5662.23	76.71	11.60	0.18
<i>kalman</i>	8601.63	72.20	8727.71	72.30	1.47	0.05
<i>am2910</i>	2165.04	30.54	2205.00	40.32	0.45	32.03
Average	—	—	—	—	3.87	4.49

(b)

will appear on several of its inputs. Similarly, if M_i corresponds to a fanout vertex and the fanout paths reconverge at M_j , test data for M_j must be justified through multiple output ports of M_i .

- 2) *Propagation constraints*: If M_i lies on a path from M_j to the sink vertex in G_j , then the sum of the weights of the edges incident on M_i must not exceed the sum of the weights of the edges directed away from M_i . This ensures that if a k -bit test response for M_j is incident on one or more inputs of M_i , then it can be propagated through one or more outputs ports of M_i .

Note that the above constraints only reflect sufficient conditions. They can handle any combination of bus truncations and reconvergent fanouts. These conditions are not necessary though since transparency for any module modules can be ensured by providing exactly one justification path from system inputs and exactly one propagation path to system outputs. We only consider sufficient conditions here since we are attempting to minimize the amount of computation necessary to ensure module transparency. Even though some amount of global information is encapsulated in the test graphs, the constraints are local in the sense that we examine only one module at a time in every test graph.

If additional computation for global analysis is permitted, we can determine a unique path to system I/Os from the module under test M_j in each test graph G_j . We can then attempt to determine bitwidth constraints for each edge on these paths. However, path selection is complicated by the presence of fanout vertices, and an appropriate cost function must be used for path selection. For example, the propagation path for M_2 in Fig. 6(c) can either be through M_3 , M_4 , or a combination of M_3 and M_4 . We do not consider an explicit path selection procedure in this paper.

The various constraints on the edge weights are shown beside the test graphs in Figs. 6 and 7(a). The constraints derived from the various test graphs typically overlap, thus these constraints must be combined to obtain the set of global constraints. The total increase in the system-level interconnect for $\mathcal{S}$ is given by $C = \sum_{i=0}^{11} (W_i^* - W_i)$, where the W_i^* s are variables whose values are to be determined and the W_i s are known constants. Our objective is to minimize C subject to the constraints on the edge weights. This can be expressed as the following integer linear programming (ILP) model:

Objective Minimize $C = \sum_{i=0}^{11} (W_i^* - W_i)$ subject to the following.

- 1) $W_1^* \leq W_2^*$,
- 2) $W_2^* \leq W_3^* + W_4^*$,
- 3) $W_4^* \leq W_6^*$,
- 4) $W_6^* \leq W_7^* + W_8^*$,
- 5) $W_8^* \leq W_9^* + W_{10}^*$,
- 6) $W_7^* + W_{10}^* \leq W_{11}^*$,
- 7) $W_1^* \leq W_0^*$,
- 8) $W_4^* \leq W_2^*$,
- 9) $W_2^* \leq W_1^*$,
- 10) $W_6^* \leq W_4^* + W_5^*$,
- 11) $W_8^* \leq W_6^*$,
- 12) $W_{10}^* \leq W_{11}^*$,
- 13) $W_7^* + W_8^* \leq W_6^*$,
- 14) $W_{10}^* \leq W_8^*$.

The above ILP model can be easily solved using a standard public-domain solver (we used *lpsolve* [3]) to obtain G^* as shown in Fig. 7(b). The solver run time was only a few seconds for this example. The edges whose weights have been updated are shown in bold; for the sake of comparison, their original values are also shown. In order to ensure transparency, the

TABLE II
RESULTS ON THE SYNTHESIS OF HIERARCHICALLY-TESTABLE SYSTEM $\mathcal{S}$ COMPOSED USING TRANSPARENT MODULES: (a) WIRE
LOAD PARAMETER SET TO 0.5 (b) WIRE LOAD PARAMETER SET TO 0.1

Circuit	Area	Delay	Percentage increase in area	Percentage increase in delay	Number of I/O ports
Hierarchical system $\mathcal{S}$ (original)	19033.58	73.69	—	—	72
Hierarchical system $\mathcal{S}$ (transparent)	19643.35	74.85	3.20	1.58	85

(a)

Circuit	Area	Delay	Percentage increase in area	Percentage increase in delay	Number of I/O ports
Hierarchical system $\mathcal{S}$ (original)	17180.77	73.69	—	—	72
Hierarchical system $\mathcal{S}$ (transparent)	17730.99	74.85	3.20	1.58	85

(b)

high-level modules must be synthesized with the number of I/Os corresponding to the W_i^* s. The resulting system incorporating these transparent modules can be tested in hierarchical fashion by making complete reuse of the precomputed tests for the individual modules. Note however that these precomputed tests must be generated for the transparent modules since we expect the tests to change (relative to the original circuit) after synthesis with embedded multiplexers. The impact of the resynthesis process on module test sets remains an interesting open problem. An alternative approach is to introduce test points for greater test access and perform test generation for the flattened SOC. However, we expect the number of test points and the test generation time to be high for SOC designs.

IV. EXPERIMENTAL RESULTS

In this section, we present experimental results on the proposed synthesis-for-test method for high-level synthesis benchmarks [6], [22]. In addition to the four benchmarks comprising $\mathcal{S}$, we also use the benchmarks *lru*, *diffeq*, and *dhrc*. All these benchmarks are available as behavioral VHDL models. In order to illustrate the insertion of the embedded multiplexer, we present the *GCD* example in the Appendix. We carried out three sets of experiments using the Synopsys Design Compiler (*lsi10k* library) running on a Sun Ultra 10 workstation with a 333 MHz processor and 256 MB of DRAM. The synthesis time varied from a few seconds for the individual modules to less than 10 minutes for the complete system. The area figures were obtained by initially setting the Design Compiler's wire load model parameter to 0.5 and then changing it to 0.1. This parameter is used by the Synopsys synthesis tool to estimate interconnect area relative to gate area.

First, we synthesized nontransparent and transparent versions of each of the high-level benchmarks to evaluate the impact of embedding multiplexers on their area and performance. We then carried out a case study by synthesizing the example $\mathcal{S}$ formed

from four benchmark circuits and a 32-bit combinational multiplier. We then synthesized an easily-testable version of $\mathcal{S}$ by making each module in it transparent. The optimization model of Section III was used to derive a transparent behavioral model that minimized additional interconnect area. Next, we examined all the benchmarks with explicit multiplexers (inserted after synthesis) to evaluate their impact on area and performance. This allowed us to compare embedded multiplexers with the trivial approach of adding explicit multiplexers. Finally, we applied the synthesis-for-transparency approach to the LEON processor core.

Table I presents experimental results on synthesis using embedded multiplexers for seven high-level benchmarks. Some of these benchmarks are included in System $\mathcal{S}$. The goal of this experiment was to evaluate the effect of synthesis for transparency on the area and delay of individual modules. The average increase in area due to multiplexer embedding is only 3.96%. Interestingly, in many cases, the delay of the circuit decreased (due to efficient resynthesis) despite the multiplexer added to its behavioral description. This is in sharp contrast to external multiplexers inserted at the gate level, which usually increase the delay.

In Table II, we present experimental results on the synthesis of the hierarchical system $\mathcal{S}$ using transparent modules with embedded multiplexers, and using the graph model and optimization framework described in Section III. The results show that complete hierarchical testability of $\mathcal{S}$ can be achieved with an area overhead of 3.2% and performance loss of only 1.6%. As discussed in Section III, the additional system I/O pins for test data can be multiplexed using ad hoc methods based on the functional interconnections between the modules. A systematic strategy for minimizing the number of I/Os needs further investigation.

Finally, in Table III, we present the impact on the area and performance of adding explicit multiplexers. As expected, the trivial method of adding multiplexers at the gate level leads to much higher overhead. The area overhead (9% on average) and

TABLE III
IMPACT OF EXPLICIT MULTIPLEXERS ON THE AREA AND PERFORMANCE OF HIGH-LEVEL SYNTHESIS BENCHMARKS (DESIGN COMPILER *Isi10k* LIBRARY UNITS):
(a) WIRE LOAD PARAMETER SET TO 0.5 (b) WIRE LOAD PARAMETER SET TO 0.1

Circuit	Original circuit area	Original circuit delay (clock period)	Transparent circuit area	Transparent circuit delay (clock period)	Percentage area increase	Percentage delay increase
<i>barcode</i>	795.76	20.17	855.58	22.97	7.52	13.88
<i>GCD</i>	1327.82	77.40	1669.13	81.84	25.07	5.73
<i>diffeq</i>	7824.00	81.08	8532	82.68	9.01	1.97
<i>lru</i>	1913.00	44.78	1949.72	44.78	1.92	≈ 0
<i>dhrc</i>	5659.72	76.75	6550.8	80.62	15.74	5.04
<i>kalman</i>	9454.78	72.20	9606.59	73.85	1.6	2.2
<i>am2910</i>	2432.12	30.54	2544.41	40.70	4.6	32.43
Average					9.35	8.75

(a)

Circuit	Original circuit area	Original circuit delay (clock period)	Transparent circuit area	Transparent circuit delay (clock period)	Percentage area increase	Percentage delay increase
<i>barcode</i>	734.00	20.17	775.9	22.97	5.7	13.88
<i>GCD</i>	1228.21	71.45	1480.24	81.78	20.57	5.69
<i>diffeq</i>	7061.64	81.08	7716.80	82.68	9.28	1.97
<i>lru</i>	1665.43	25.71	1736.43	26.57	4.26	3.35
<i>dhrc</i>	5073.4	76.56	5921.68	80.62	16.72	5.14
<i>kalman</i>	8601.63	72.20	8736.58	73.85	1.57	2.2
<i>am2910</i>	2165.04	30.54	2287.06	40.70	4.6	32.43
Average					8.96	9.23

(b)

Fig. 8. A block diagram of the LEON core.

the penalty on system performance (also 9% on average) are both high.

Finally, we present the results of applying the synthesis approach to the *proc* and *peri* modules in the LEON core. Unfortunately, we were unable to make the synthesis script handle other LEON modules. A block diagram of the LEON core is shown

in Fig. 8, and block diagrams of the *proc* and *peri* modules are shown in Fig. 9. The *proc* module contains the integer unit, clock/reset generation and an optional floating-point unit (not considered in our experiments). The *peri* module is a smaller controller unit used for instantiating all peripherals. Table IV shows the impact of transparency synthesis on these two LEON

Fig. 9. A block diagram of the (a) *proc* and (b) *peri* modules in the LEON core.

modules. The CPU time for synthesis was less than seven minutes in each case. We also conducted an experiment in which we treated *peri* as a hierarchical system composed of synthesizable blocks. The area overhead in this case was only 0.3%.

V. CONCLUSION

We have presented a new synthesis-for-test approach in which multiplexers are embedded in the behavioral models of

the various modules constituting a hierarchical system. This approach can also be used to design a hierarchically-testable system-on-a-chip using synthesizable embedded cores. The embedded multiplexers provide single-cycle transparency, thereby offering a straightforward yet effective solution to the problems of test data propagation and test vector translation. In order to reduce area/performance overhead and maximize test reuse, we have presented a graph-theoretic framework and an optimization method based on integer linear programming. We have presented case studies using high-level synthesis

TABLE IV
SYNTHESIS OF TRANSPARENT LEON MODULES

Circuit	Original circuit area	Circuit delay (clock period)	Transparent circuit area	Transparent ckt delay (clock period)	Area increase (%)	Delay increase (%)
<i>proc</i>	14796.03	68.75	15385.30	68.75	3.9	0
<i>peri</i>	8307.48	25.57	8753.8	24.95	5.2	-2.5

```

-----
-- GCD factorization Benchmark
-----
entity GCD is
  port (CLK          : in bit;
        Reset        : in bit;
        X, Y          : in integer range 0 to 65535;
        gcd_output    : out integer range 0 to 65535);
end GCD;
-----

architecture rtl of GCD is
begin
  BIGLOOP: process
    variable xvar,yvar : integer range 0 to 65535;
  begin
    RESET_LOOP : loop
      wait until CLK = '1';
      xvar := X;
      yvar := Y;
      if ((xvar = 0) or (yvar = 0) or (Reset = '1')) then
        gcd_output <= 0;
        xvar := 0;
        yvar := 0;
      else
        COMP: loop
          wait until CLK = '1';
          exit RESET_LOOP when (Reset = '1');
          if (xvar < yvar) then
            yvar := yvar - xvar;
          else
            if (xvar > yvar) then
              xvar := xvar - yvar;
            end if;
          end if;
          exit COMP when (xvar = yvar);
        end loop COMP;
        gcd_output <= xvar;
      end if;
    end loop RESET_LOOP;
  end process;
end rtl;

```

Fig. 10. Behavioral VHDL code for the *GCD* benchmark.

benchmarks to show that synthesis for transparency introduces very little area and performance overhead.

The results presented in this paper raise a number of interesting open questions. While we can obtain parallel access to module I/Os for at-speed testing, the clock frequency at which patterns are applied is affected by the longest combinational path that spans several modules in transparent mode. This limits the speed at which the test patterns can be applied even though parallel access is available. We are investigating the insertion

of observation points at internal nodes in the system to overcome this problem. We are also developing an algorithm that will facilitate the sharing of system-level I/Os since the entire I/O bandwidth is not necessary for any single test session. In addition, we are studying how BIST can be incorporated in the synthesis framework. SOC pin count limitations can be overcome using a combination of width compression [7], [15] and output space compaction [26]. Even though these approaches will restrict the synthesis approaches to specific test sets, they

```

-----
-- GCD factorization Benchmark (transparent)
-----

entity GCD is
  port (CLK          : in bit;
        Reset        : in bit;
        test         : in bit;
        X, Y          : in integer range 0 to 65535;
        gcd_output,new_output : out integer range 0 to 65535 := 1);
end GCD;

-----

architecture rtl of GCD is
begin
  BIGLOOP: process
    variable xvar,yvar : integer range 0 to 65535;
  begin
    RESET_LOOP : loop
      wait until CLK = '1';
      exit RESET_LOOP when (test = '1');
      xvar := X; yvar := Y;
      if ((xvar = 0) or (yvar = 0) or (Reset = '1')) then
        gcd_output <= 0;
        xvar := 0; yvar := 0;
      else
        COMP: loop
          wait until CLK = '1';
          exit RESET_LOOP when (Reset = '1');
          if (xvar < yvar) then
            yvar := yvar - xvar;
          else
            if (xvar > yvar) then
              xvar := xvar - yvar;
            end if;
          end if;
          exit COMP when (xvar = yvar);
        end loop COMP;
        gcd_output <= xvar;
      end if;
    end loop RESET_LOOP;
    if test = '1' then
      gcd_output <= X; new_output <= Y;
    else
      null;
    end if;
  end process;
end rtl;

```

Fig. 11. Behavioral VHDL code for the *GCD* benchmark with embedded multiplexer.

will provide single-cycle transparency with a small number of SOC pins.

Despite the numerous benefits offered by the proposed synthesis methodology, there are a few limitations that limit its applicability to specific types of hierarchical systems. The embedded multiplexer approach depends on the availability of synthesizable models. However, these models are usually not available for intellectual property (IP) cores. For systems that use a combination of synthesizable and IP modules, we can combine synthesis for transparency with other known DFT methods. For example, IP modules and hard cores, which are usually wrapped for ease of testing, can be treated as pseudo-system I/Os for determining transparency paths. The transparency paths can begin and end at the scan cells within the wrappers.

APPENDIX

We present an example of the *GCD* benchmark to illustrate how transparency is inserted at the behavioral level. Fig. 10

shows the original benchmark's synthesizable behavioral VHDL description and Fig. 11 shows the behavioral model with the embedded multiplexer.

REFERENCES

- [1] M. S. Abadir and M. A. Breuer, "Test schedules for VLSI circuits having built-in self-test hardware," *IEEE Trans. Comput.*, vol. 35, pp. 361–367, Apr. 1985.
- [2] P. Ashar and S. Malik, "Implicit computation of minimum-cost feedback-vertex sets for partial scan and other applications," in *Proc. Design Automation Conf.*, 1994, pp. 77–80.
- [3] M. Berkelaar. Ipsolve 3.0. Eindhoven University of Technology, Eindhoven, The Netherlands. [Online]. Available: ftp://ftp.ics.ele.tue.nl/pub/lp_solve
- [4] F. Bouwman, S. Oostdijk, R. Stans, B. Benetts, and F. Beenker, "Macro testability: The results of production device applications," in *Proc. Int. Test Conf.*, 1992, pp. 232–241.
- [5] S. Bhatia and N. K. Jha, "Integration of hierarchical test generation with behavioral synthesis of controller and data path circuits," *IEEE Trans. VLSI Syst.*, vol. 6, pp. 608–619, Dec. 1998.

- [6] CAD Benchmarking Laboratory. High-level synthesis benchmarks. North Carolina State Univ., Raleigh, NC. [Online]. Available: <http://www.cbl.ncsu.edu>
- [7] K. Chakrabarty and B. T. Murray, "Design of built-in test generator circuits using width compression," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 1044–1051, Oct. 1998.
- [8] S. T. Chakradhar, A. Balakrishnan, and V. D. Agrawal, "An exact algorithm for selecting partial scan flip-flops," in *Proc. Design Automation Conf.*, 1994, pp. 81–86.
- [9] R. Chandramouli and S. Pateras, "Testing systems on a chip," *IEEE Spectrum*, pp. 42–47, Nov. 1996.
- [10] C.-H. Chen, T. Karnik, and D. G. Saab, "Structural and behavioral synthesis for testability techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, pp. 777–785, 1994.
- [11] K.-T. Cheng and V. D. Agrawal, "A partial scan method for sequential circuits with feedback," *IEEE Trans. Comput.*, vol. 39, pp. 544–548, 1990.
- [12] S. Freeman, "Test generation for datapath logic," *IEEE J. Solid-State Circuits*, vol. 23, pp. 421–427, Apr. 1988.
- [13] I. Ghosh, A. Raghunathan, and N. K. Jha, "Design for hierarchical testability of RTL circuits obtained by behavioral synthesis," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 1001–1014, Sept. 1997.
- [14] I. Ghosh, N. K. Jha, and S. Dey, "A low overhead design for testability and test generation technique for core-based systems-on-a-chip," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 1661–1676, Nov. 1999.
- [15] I. Hamzaoglu and J. H. Patel, "Reducing test application time for built-in self-test test pattern generators," in *Proc. VLSI Test Symp.*, 2000, pp. 369–375.
- [16] J. Lee and J. H. Patel, "Architectural level test generation for microprocessors," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1288–1300, Oct. 1994.
- [17] Y. Makris and A. Orailoglu, "RTL test justification and propagation analysis for modular designs," *J. Electron. Testing: Theory Applicat.*, vol. 13, pp. 105–120, Oct. 1998.
- [18] —, "DFT guidance through RTL test justification and propagation analyzes," in *Proc. Int. Test Conf.*, 1998, pp. 668–677.
- [19] B. T. Murray and J. P. Hayes, "Hierarchical test generation using pre-computed tests for modules," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 594–603, June 1990.
- [20] M. Nourani and C. Papachristou, "Parallelism in structural fault testing of embedded cores," in *Proc. IEEE VLSI Test Symp.*, 1998, pp. 15–19.
- [21] S. M. Reddy and R. Dandapani, "Scan design using standard flip-flops," *IEEE Des. Test Comput.*, vol. 4, pp. 52–54, Feb. 1987.
- [22] P. R. Panda and N. D. Dutt, "1995 high-level synthesis design repository," in *Proc. Int. Symp. Syst. Level Synthesis*, 1995, pp. 170–174.
- [23] M. Potkonjak, S. Dey, and R. K. Roy, "Behavioral synthesis of area-efficient testable designs using interaction between hardware sharing and partial scan," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 1141–1153, Sept. 1995.
- [24] S. Ravi, G. Lakshminarayana, and N. K. Jha, "A framework for testing core-based systems-on-a-chip," in *Proc. Int. Conf. Computer-Aided Design*, 1999, pp. 385–390.
- [25] J. Savir, "Testing for single faults in modular combinational networks," *J. Design Automation Fault-Tolerant Computing*, vol. 3, pp. 69–82, 1979.
- [26] M. Seuring and K. Chakrabarty, "Space compaction of test responses for IP cores using orthogonal transmission functions," in *Proc. IEEE VLSI Test Symp.*, 2000, pp. 213–219.
- [27] P. Vishakantaiah, J. A. Abraham, and D. G. Saab, "CHEETA: Composition of hierarchical sequential tests using ATKET," in *Proc. International Test Conference*, 1993, pp. 606–615.
- [28] P. Vishakantaiah, T. Thomas, J. A. Abraham, and M. S. Abadir, "AMBIANT: Automatic generation of behavioral modifications for testability," in *Proc. Int. Conf. Computer Design*, 1993, pp. 63–66.
- [29] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing embedded-core based system chips," in *Proc. Int. Test Conf.*, 1998, pp. 130–143.
- [30] (2000, Feb.) LEON-1 VHDL Model Description (version 2.0). European Space Res. and Tech. Center, Noordwijk, The Netherlands. [Online]. Available: <http://www.estec.esa.nl/wsmwww/leon>.

Krishnendu Chakrabarty (SM'92–M'96–SM'00) received the B.Tech. degree from the Indian Institute of Technology, Kharagpur, in 1990 and the M.S.E. and Ph.D. degrees, all in computer science and engineering, from the University of Michigan, Ann Arbor, in 1992 and 1995, respectively.

Currently, he is an Assistant Professor in the Department of Electrical and Computer Engineering, Duke University, Durham, NC. From 2000 to 2002, he was a Mercator Visiting Professor at the University of Potsdam, Germany. His current research projects (supported by NSF, DARPA, ONR, Army Research Office, and industrial sponsors) include system-on-a-chip test, embedded real-time operating systems, distributed sensor networks, and modeling, simulation and optimization of microelectrofluidic systems. He has published over 120 papers in archival journals and refereed conference proceedings, and he holds a U.S. patent in built-in self-test. He is a coauthor of *Microelectrofluidic Systems: Modeling and Simulation* (Boca Raton, FL: CRC, 2002) and *Test Resource Partitioning for System-on-a-Chip* (Norwell, MA: Kluwer, 2002), and an editor of the book *SOC (System-on-a-Chip) Testing for Plug and Play Test Automation* (Norwell, MA: Kluwer, 2002).

Dr. Chakrabarty is a recipient of the National Science Foundation Early Faculty (CAREER) Award and the Office of Naval Research Young Investigator Award. He was a recipient of a Best Paper Award at the 2001 Design, Automation and Test in Europe (DATE) Conference. He is currently an Associate Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, Editor of *Journal of Electronic Testing: Theory and Applications* (JETTA), and was Guest Editor of a special issue of JETTA on system-on-a-chip testing, published in August 2002. In 2001, he was also a Guest Editor of a special issue of the Journal of the Franklin Institute on distributed sensor networks. He serves as Vice Chair of Technical Activities in IEEE's Test Technology Technical Council, and is a member of the program committees of several IEEE/ACM conferences and workshops. He is a Member of ACM, ACM SIGDA, and of Sigma Xi.