

# Synthesis of Transparent Circuits for Hierarchical and System-on-a-Chip Test<sup>1</sup>

Krishnendu Chakrabarty<sup>†</sup>, Rajatish Mukherjee<sup>‡</sup> and Andrew Exncios<sup>†</sup>

<sup>†</sup>Dept. Electrical & Computer Engineering  
Duke University  
Durham, NC 27708, USA

<sup>‡</sup>Dept. Mathematical & Computer Sciences  
University of Tulsa  
Tulsa, OK 74104

## Abstract

*We propose a 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 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.*

## 1 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 [11–13]. These 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 [10, 18]. Since the embedded cores in an SOC are not directly accessible from the chip I/Os, the system integrator 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 [8] and I-paths [1] which utilized existing propagation paths between module I/Os and chip I/Os. More recent work has focussed on design for testability (DFT) methods. For example, the FScan-BScan method utilizes a combination of full scan and boundary scan [6]. 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 [6]. However, this method also suffers from area and delay overheads; moreover, interconnect faults cannot be tested using this technique.

To reduce test generation time, a hierarchical test generation strategy was presented in [13]. 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 [11, 12]. 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 [10], 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: (i) high test application time due to the transparency latency (multi-cycle transparency) required to propagate test data through a module, (ii) 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 [16]. However, these methods have focussed primarily on selection of registers for BIST, selection of scan flip-flops, reduction in

<sup>1</sup>This research was supported in part by the National Science Foundation under grant number CCR-9875324. Andrew Exncios was supported by the Pratt Engineering Undergraduate Fellows Program, Pratt School of Engineering, Duke University.

the number of loops in the datapath, and hierarchical testability enhancement of RTL circuits obtained through behavioral synthesis [7, 9]. In order to provide hierarchical test capabilities, they typically require behavioral descriptions in the form of control-data flow graphs [4].

Instead of providing complete controllability and observability to core I/Os, more recent work has focussed on providing controllability and observability to SOC cores on an “as needed basis” [17]. This is achieved through transparency analysis using non-deterministic finite-state automata, and transparency enhancement through symbolic justification and propagation analysis based on a regular expression framework. However, this 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 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. 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 [14]. 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  $M$  must be 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.

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 con-

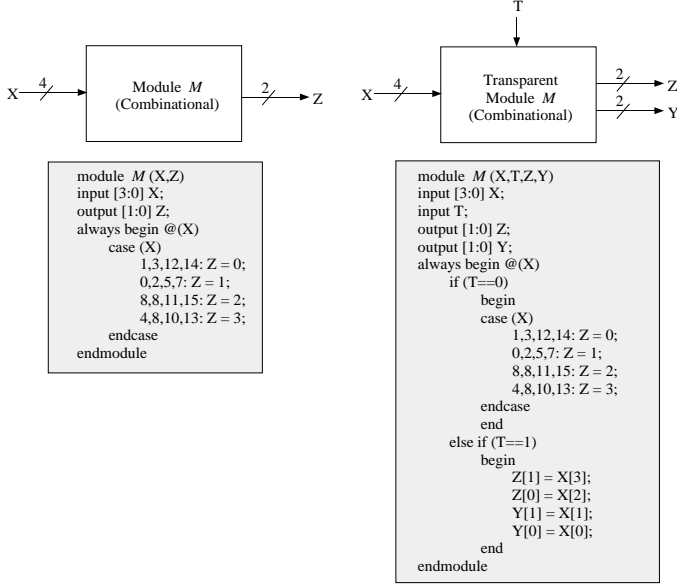
trol/data flow information. Its conceptual simplicity makes it easy to implement and integrate in the design flow. It does not require any test composition or sophisticated test scheduling algorithms. Precomputed test sets for every module can be readily applied without requiring any transparency latency or test vector translation. These test sets may contain functional vectors, scan vectors or ordered test sequences for non-scan sequential circuits. The test methodology provides parallel test access to the embedded modules thereby facilitating at-speed test and increasing the coverage of non-modeled and performance-related defects. 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 non-trivial example system constructed by stitching together several high-level synthesis benchmarks [5, 15]. 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. 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 [19]. The LEON core is a hierarchical SPARC-compatible processor developed by the European Space Agency for future space missions. We show that these two modules can be made transparent with no delay overhead and area overhead of only 5.2%.

## 2 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 Figure 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 Figure 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 3, we describe how the overhead of additional ports and associated wiring/interconnect area can be minimized by analyzing the system-level hierarchi-



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

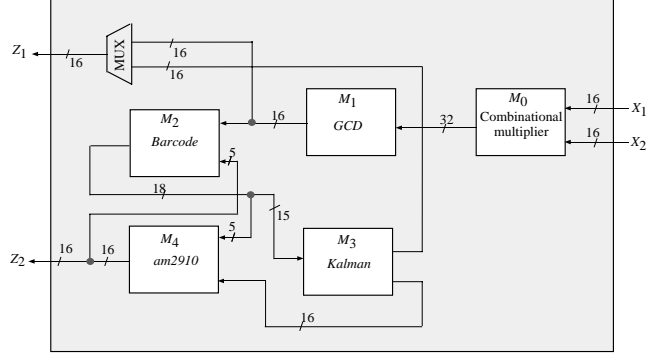
cal testing needs.

Multiplexers can similarly be embedded in the behavioral models of finite-state machines (FSMs). The modified FSM operates in the normal functional mode when the control input  $T = 0$ . However, when  $T = 1$ , it works in a transparent mode and operates as a pass-through combinational circuit. The state transitions of the FSM 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.

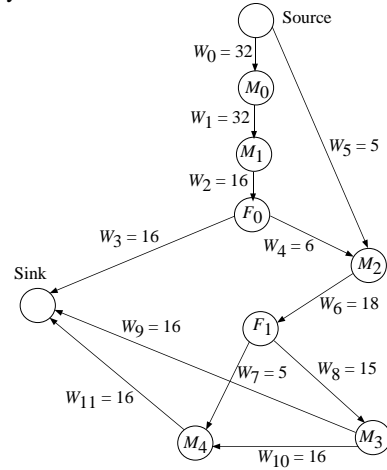
### 3 System-level test strategy

In this section, we describe the hierarchical test methodology using a non-trivial example  $S$  of a system with two 32-bit input ports and two 32-bit output ports and composed hierarchically of several synthesizable modules (Figure 2). The example was constructed using four high-level benchmark circuits (*GCD*, *Barcode*, *Kalman*, and *am2910*) [15] and a 32-bit combinational multiplier. Each benchmark is a sequential circuit with clock and reset inputs, which are not shown explicitly in Figure 2. In order to make the example non-trivial and realistic, we introduced a feedback loop and used bus lines of unequal widths. We also introduced bus truncation and fanout at several places in  $S$ .

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



**Figure 2.** An example of a system composed hierarchically of synthesizable modules.



**Figure 3.** Acyclic system graph  $G$  for the example of Figure 2.

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  $M_j$ .

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 using heuristic methods [2]. 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 however, this problem is tractable enough to be solved by inspection due to the small problem size. The acyclic system graph  $G$  is shown in Figure 3. The vertices  $F_0$  and  $F_1$  denote fanout branches in the system. The source and sink vertices in the graph represent system-level primary inputs and primary outputs, respectively. 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.

The proposed test methodology applies precomputed test sets to the modules in  $S$  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  $\lceil \log_2 m \rceil$  using a decoder since only one module is tested in any session, which implies that only one of the  $T_i$ 's,  $0 \leq i < m$ , is 0 in any test session.

As discussed in Section 2 and illustrated in Figure 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.

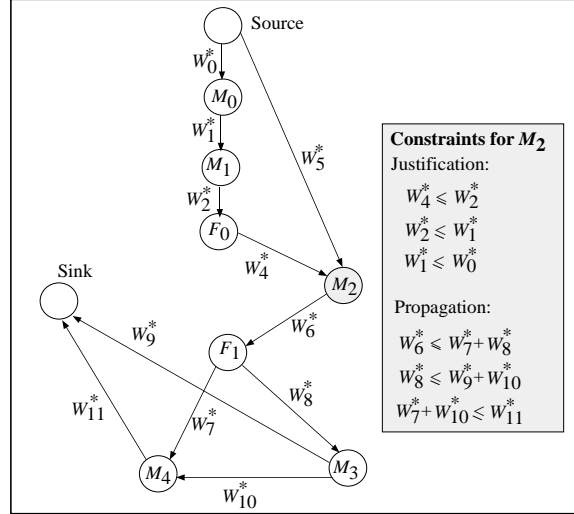
In order to make the system transparent, we now have to transform  $G$  to  $G^*$  by increasing the weights of the edges in  $G$ . For the edges that are incident on  $M_j$ , let us denote the new edge weights by  $w_{i1}^*, w_{i2}^*, \dots, w_{ip_j}^*$ . Similarly, for the edges that are directed away from  $M_j$ , let us denote the new edge weights by  $w_{o1}^*, w_{o2}^*, \dots, w_{oq_j}^*$ .

The edge weights in  $G^*$  can only be determined after a global analysis of  $G$ . In addition, the total increase in the system-level interconnect, given by the sum of bitwidth increases for the edges, should also be minimized. In Figure 3, 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 easily 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. Similarly, an edge  $(M_i, M_k)$  belongs to  $G_j$  if it either lies on a path from the source vertex to  $M_j$  or on a path from  $M_j$  to the sink vertex. Figure 4 shows the test graph for  $M_2$  module 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 (but not necessary) 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 weight of each edge directed away from  $M_i$  must be at most equal to the sum of the weights of the edges incident on  $M_i$ .



**Figure 4.** The test graphs and constraints on the edge weights for  $M_2$  (Barcode).

2. *Propagation constraints:* If  $M_i$  lies on a path from  $M_j$  to the sink vertex in  $G_j$ , then the weight of every edge incident on  $M_i$  must be at most equal to the sum of the weights of the edges directed away from  $M_i$ .

The various constraints on the edge weights are shown beside the test graph for module  $M_2$  in Figure 4. 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:  
 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)  $W_7^* \leq W_6^*$ ; 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 Figure 5. The solver run time was only a few seconds for this example. The edges whose weights have been updated are highlighted; for the sake of comparison, their original values are also shown. In order to ensure transparency, the 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

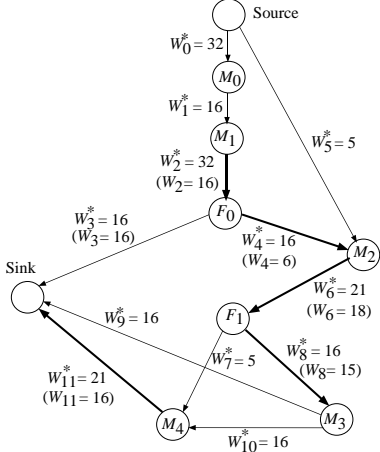


Figure 5. The graph  $G^*$  with updated edge weights.

(relative to the original circuit) after synthesis.

## 4 Experimental results

In this section, we present experimental results on the proposed synthesis-for-test method for high-level synthesis benchmarks [5, 15]. 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 333MHz 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 non-transparent 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 3 was used to derive a transparent behavioral model that minimized additional interconnect area.

Table 1 presents experimental results on synthesis using embedded multiplexers for seven high-level benchmarks. The average increase in area due to multiplexer embedding is only 4%. 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.

Circuit	Original circuit area	Circuit delay (clock period)	Transparent circuit area	Transparent ckt delay (clock period)	Area increase (%)	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	Circuit delay (clock period)	Transparent circuit area	Transparent ckt delay (clock period)	Area increase (%)	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)

Table 1. Impact of embedded multiplexers on the area and performance of high-level synthesis benchmarks (Design Compiler *lsi10k* library units): (a) wire load parameter set to 0.5 (b) wire load parameter set to 0.1.

In Table 2, 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 3. The results show that complete hierarchical testability of  $\mathcal{S}$  can be achieved with an area overhead of only 3.2% and performance loss of only 1.6%. As discussed in Section 3, 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 remains an interesting and important open problem.

We also evaluated the impact of adding explicit multiplexers for transparency to the high-level synthesis benchmarks. As expected, the trivial method of adding multiplexers at the gate level leads to much higher overhead. While the area overhead is very high (9% on average), the penalty on system performance is especially severe (also 9% on average).

Finally, we present the results of applying the synthesis approach to the *proc* and *peri* modules in the LEON core. The *proc* module contains the integer unit, clock/reset generation and the floating-point unit. The *peri* module is a smaller controller unit used for instantiating all peripheral

Circuit	Area	Delay	Area increase (percent)	Delay increase (percent)	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	Area increase (percent)	Delay increase (percent)	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)

**Table 2.** 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	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

**Table 3.** Synthesis of transparent LEON modules.

als. Table 3 shows the impact of transparency synthesis on these two LEON modules. The CPU time for synthesis was less than 7 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%.

## 5 Conclusions

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 a case study using high-level synthesis benchmarks to show that synthesis for transparency introduces very little area

and performance overhead.

## References

- [1] M. S. Abadir and M. A. Breuer, "Test schedules for VLSI circuits having built-in self-test hardware", *IEEE Trans. Computers*, vol. 35, pp. 361–367, April 1985.
- [2] P. Ashar and S. Malik, "Implicit computation of minimum-cost feedback-vertex sets for partial scan and other applications", *Proc. Design Automation Conf.*, pp. 77–80, 1994.
- [3] M. Berkelaar. *Ipsolve*, version 3.0. Eindhoven University of Technology, Design Automation Section, Eindhoven, The Netherlands. [ftp://ftp.ics.ele.nl/pub/lp\\_solve](ftp://ftp.ics.ele.nl/pub/lp_solve).
- [4] S. Bhatia and N. K. Jha, "Integration of hierarchical test generation with behavioral synthesis of controller and data path circuits", *IEEE Trans. VLSI Systems*, vol. 6, pp. 608–619, Dec. 1998.
- [5] High-level synthesis benchmarks, CAD Benchmarking Laboratory, North Carolina State University, <http://www.cbl.ncsu.edu>
- [6] R. Chandramouli and S. Pateras, "Testing systems on a chip", *IEEE Spectrum*, pp. 42–47, November 1996.
- [7] C.-H. Chen, T. Karnik and D. G. Saab, "Structural and behavioral synthesis for testability techniques", *IEEE Trans. CAD*, vol. 13, pp. 777–785, 1994.
- [8] S. Freeman, "Test generation for datapath logic", *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 421–427, 1988.
- [9] I. Ghosh, A. Raghunathan and N. K. Jha, "Design for hierarchical testability of RTL circuits obtained by behavioral synthesis", *IEEE Trans. CAD*, vol. 16, pp. 1001–1014, 1997.
- [10] 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. CAD*, vol. 18, pp. 1661–1676, 1999.
- [11] J. Lee and J. H. Patel "Architectural level test generation for microprocessors", *IEEE Trans. CAD*, vol. 13, pp. 1288–1300, 1994.
- [12] Y. Makris and A. Orailoglu, "RTL test justification and propagation analysis for modular designs", *JETTA*, vol. 13, pp. 105–120, 1998.
- [13] B. T. Murray and J. P. Hayes, "Hierarchical test generation using precomputed tests for modules", *IEEE Trans. CAD*, vol. 9, pp. 594–603, 1990.
- [14] M. Nourani and C. Papachristou. "Parallelism in structural fault testing of embedded cores", *Proc. IEEE VLSI Test Symp.*, pp. 15–19, 1998.
- [15] P. R. Panda and N. D. Dutt, "1995 high-level synthesis design repository", *Proc. Int. Symp. System Level Synthesis*, pp. 170–174, 1995.
- [16] 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. CAD*, vol. 14, pp. 1141–1153, 1995.
- [17] S. Ravi, G. Lakshminarayana and N. K. Jha. "A framework for testing core-based systems-on-a-chip", *Proc. ICCAD*, pp. 385–390, 1999.
- [18] Y. Zorian, E. J. Marinissen and S. Dey, "Testing embedded-core based system chips", *Proc. Int. Test Conf.*, pp. 130–143, 1998.
- [19] "LEON-1 VHDL Model Description (version 2.0)", European Space Res. & Tech. Center, Noordwijk, The Netherlands. <http://www.estec.esa.nl/wsmwww/leon>, Feb 2000.