Synthesizing Fast, Online-Testable Control Units

SYBILLE HELLEBRAND
HANS-JOACHIM WUNDERLICH
ANDRE HERTWIG
University of Stuttgart

The authors present the self-checking bypass pipeline, an online-testable controller structure for data-dominated applications. For most circuits in a standard benchmark set, this structure leads to a performance improvement of more than 30% with an area overhead less than 15% that of conventional online-testable finite-state machines.

MICROELECTRONIC SYSTEMS in safety-critical applications must meet both high-dependability and high-performance requirements. System designers usually attain high performance by implementing pipelined or massively parallel data paths and by decomposing control units into small, communicating finite-state machines.[1-3] To ensure that systems meet quality standards, manufacturers must use offline production and maintenance tests as well as self-checking and fault tolerance techniques. In the past, research efforts focused on developing dependable data path architectures.[4,5] But highly dependable controller/data path systems must also handle control unit errors, which cannot be detected or corrected by the data path.

For such systems, researchers have proposed two main online error detection approaches. One approach is to check the controller outputs using the same techniques as applied to data paths. The other is to apply special techniques for observing the state transitions and the control flow. Approaches in the second category range from state coding based on error-detecting codes to control flow monitoring by on-chip signature analyzers or special monitoring machines.[6-8] Usually, these techniques lead to considerable hardware overhead, performance degradation, or error detection latency. In particular, their impact on circuit speed may not be tolerable for high-performance applications.

Of course, to obtain fast self-checking controllers, we can combine these error detection schemes with the decomposition techniques used for high-performance controllers. In a straightforward solution such as that shown in Figure 1, however, the extra test hardware required for each submachine may lead to an enormous overall cost.

The structure we propose in this article avoids the multiple implementation of online test circuitry for submachines while preserving the advantages of finite-state machine decomposition. It reduces hardware overhead by sharing test circuitry among different units and guarantees an error detection latency of one clock cycle. Furthermore, it is suitable for a high-performance implementation avoiding master-slave flip-flops, and it supports efficient offline built-in self-test (BIST).

Target structure

Figure 2 diagrams the target controller structure. We assume that the controller's states can be partitioned into three groups, and that state transitions occur only between groups, not within groups. The synthesis procedure described later always ensures this property.

A dedicated register represents each group of states. During each clock cycle, one of the three registers—the active register—determines the present state and the output function of the entire controller. Since state transitions cannot map states to states of the same group, direct feedback lines are not necessary. Because the structure thus works as a pipeline allowing each stage to be bypassed, we call it a bypass pipeline.

To verify the correctness of a state transition, a checker that observes only the active register is sufficient. Thus, the three pipeline stages can share the same checker, resulting in considerable hardware savings compared to conventional controller networks with an extra checker for each submachine. Moreover, the checking information can be computed concurrently with the next-state function, keeping the additional delay for checking low. The decomposition of the state transition and output logic into three generally smaller blocks results in a short critical path and hence a high operating frequency. Furthermore, the absence of direct feedback loops reduces the impact of clocking skew, and we can achieve a robust implementation using single-edge-triggered flip-flops only.[9]

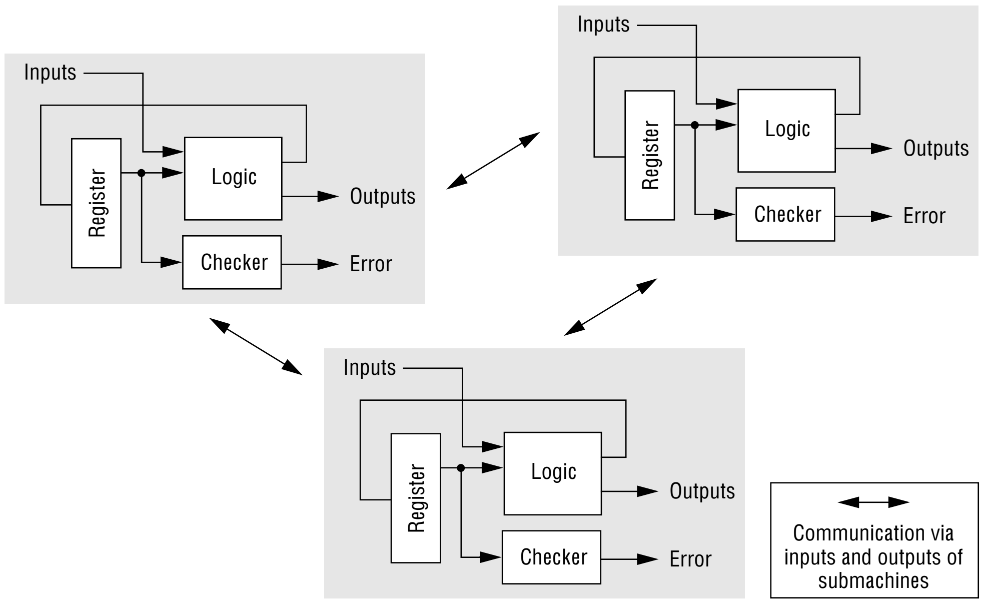

Figure 1. Network of communicating controllers with self-checking submachines.

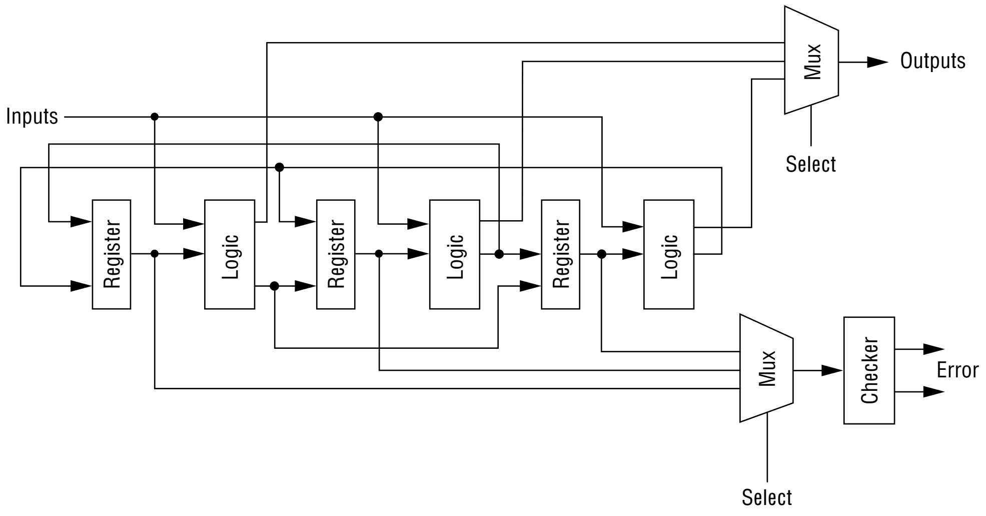

Figure 2. Self-checking bypass pipeline.

For an efficient implementation of the bypass pipeline, a simple mechanism to identify the active register is crucial. The best way to achieve this is to reserve one bit (token bit) of each register for this purpose and to use the remaining bits (state bits) to determine the state. Figure 3 (next page) illustrates the resulting token bit architecture.

Register R_i determines the state of the controller if $t_i = 1$, $i \in \{0, 1, 2\}$. The two-output function τ_i passes the token to one of the registers R_j, $i \neq j$, in the next clock cycle. Hence, exactly one of the three token bits t_0, t_1, and t_2 is always set.

State encoding and checking. In principle, the token bit architecture is compatible with any error-detecting and error-correcting encoding scheme for the state bits. However, to guarantee sufficiently high error detection capabilities for the complete structure, we must ensure that faults in the multiplexer feeding the checker are detected too. A simple strategy is to map the controller's states to code words C, such that set $C^1 := \{(c, 1) \mid c \in C\}$ is an error-detecting, error-correcting code. Figure 4 shows an example for a parity code.

If the controller's states (represented by the state bits) are

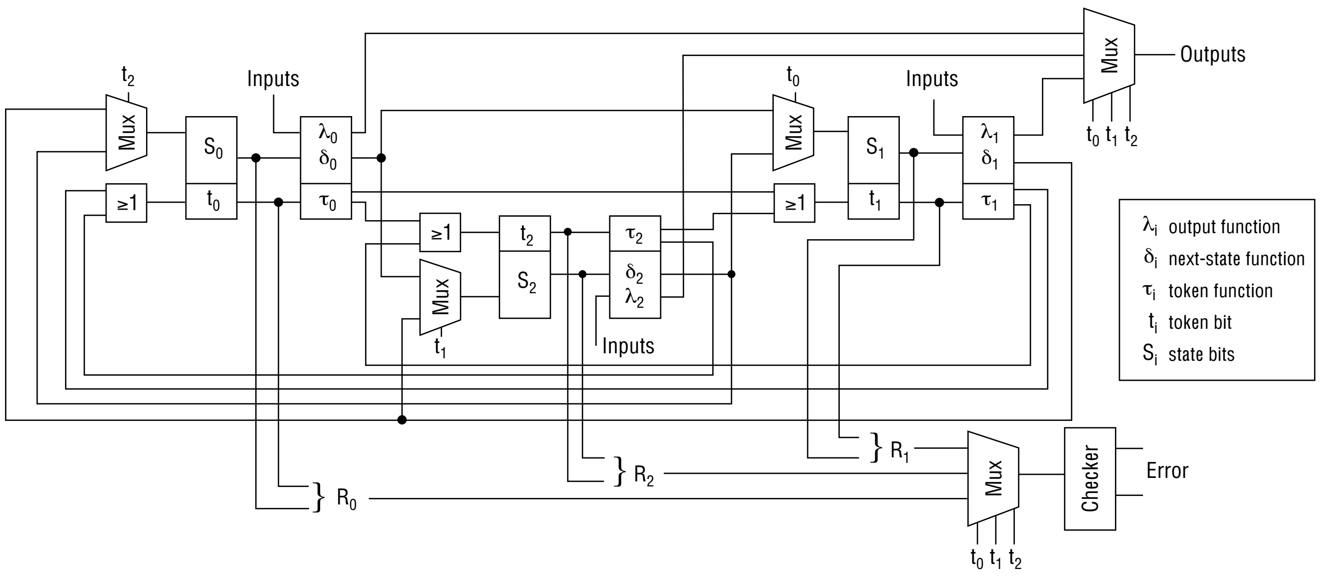

Figure 3. Token bit architecture.

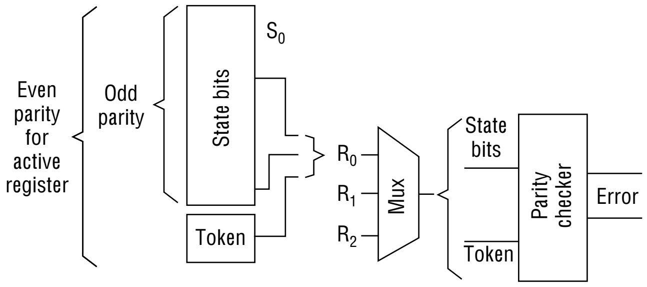

Figure 4. Example for appropriate state encoding.

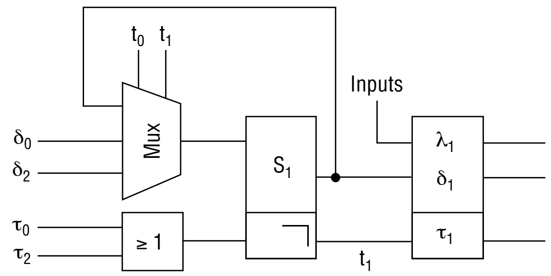

Figure 5. High-performance implementation.

mapped to an odd-parity code, the active register's contents including the token bit always have even parity. The remaining two registers' contents have odd parity. A checker for even parity will automatically detect multiplexer faults that result in passing a wrong input to the outputs. Any other multiplexer fault changing the parity of its output will be detected in the same way.

Similar implementations are possible for general-parity and *m*-out-of-*n* codes. Moreover, the correct configurations of token bits form a one-out-of-three code, and we can easily extend the checking scheme of Figure 3 to check these configurations too.

Performance issues. In contrast to conventional controller networks composed of standard structures, we can implement bypass pipelines using single-edge-triggered flip-flops, with master-slave flip-flops required for the token bits only. Figure 5 shows one pipeline stage for such an implementation. We make the state transitions robust by permanently reloading the contents of a register R_i as long as it is the active register ($t_i = 1$). The contents of R_i are completely irrelevant if it does not hold the token ($t_i = 0$), and a register can never be active during two consecutive clock cycles. Therefore, this method is sufficient to guarantee stable behavior of the controller.

We have analyzed the timing and performance aspects of bypass pipelines in detail elsewhere.[9] In general, the functions τ_i updating the token signals are less complex than the state transition functions, and token passing is not critical. For this reason, the bypass pipeline achieves better performance than a standard decomposition in which the submachines have direct feedback loops and require an implementation based on master-slave flip-flops.

Offline BIST. The proposed bypass pipeline structure not only supports an efficient, online-testable implementation but also allows complete offline BIST at little extra cost. To make the token bit architecture of Figure 3 self-testable, we

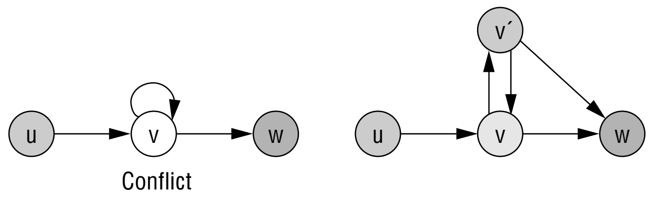

Figure 6. Conflict resolution by state splitting.

implement the registers including the token bit as built-in logic block observer (BILBO) registers. We add a pattern generator to the inputs and use a signature analyzer to monitor the outputs including the checker output. In contrast to conventional controller structures containing direct feedback loops, inserting extra test registers or concurrent BILBOs into the state transition logic is not necessary. We test the output function using all three registers as pattern generators. Three test sessions, each using one register for signature analysis and the remaining two for test pattern generation, cover the state transition logic and the functions generating the token bits. If the multiplexer feeding the checker is appropriately implemented, this procedure also provides a test of the checker.

The structure is also suitable for external testing based on partial scan. Including one of the three registers into a partial scan is sufficient to break all cycles in the structure and ensure an efficient deterministic test.

Synthesis procedure. To synthesize the token bit architectures in Figures 3 and 4, we must solve two problems: First, we must partition the states into groups, with no state transitions within groups. Then, we must select an appropriate scheme for online error detection and generate the structure by assigning suitable state codes and synthesizing the state transition and output logic.

We have shown elsewhere[9] that finding an appropriate state partitioning for a bypass pipeline corresponds to coloring the state transition graph with three colors (three shades of gray in Figure 6). Each color represents a group of states and thus a dedicated register in the final structure. However, the problem of finding the minimum number of colors for a given graph is NP-complete, and, in general, we cannot expect a graph to be three-colorable. But we can overcome this problem by splitting states.

In the example in Figure 6, introducing an additional equivalent state cuts a self-loop at state v. Node v is replaced by equivalent nodes v and v'. That is, both new nodes have the same successor nodes as the original node, and the predecessor nodes of the original node are distributed appropriately among the two new nodes. State-splitting techniques enlarge the state transition diagram, but state transition diagrams for data path/controller systems are usually not as densely meshed as for typical control-dominated designs. Therefore, just by adding a few equivalent nodes, we can transform them to solve the three-color problem.

Table 1. Average results of the proposed synthesis approach.

Circuit	$\|S\|$	$\|S_{total}\|$	$\|S_0\|$	$\|S_1\|$	$\|S_2\|$
planet	48	49	15	18	16
s1488	48	49	14	17	18
s1494	48	49	18	17	14
scf	121	122	36	48	38
s298	218	219	64	83	72
s510	47	71	24	24	23

To minimize the number of extra states required, we use a heuristic[9] that interleaves graph-coloring and state-splitting techniques. The algorithm starts by splitting states to cut all self-loops and then attempts to color a maximum number of nodes using just three colors. Finally, it transforms regions around uncolored nodes into three-colorable regions and recolors them.

As explained earlier, to synthesize the complete online-testable structure, we choose an error-detecting/error-correcting code for state assignment. The code must have desired properties such as sufficient minimum distance, simple checking, and so on. Besides these constraints, area and performance considerations may guide the selection. For practical applications, therefore, we use an additional step for state assignment: We use the Berkeley tool Nova to determine a base code and extend the base code to a parity code with a user-defined number of check bits.[10]

Practical experiments

In contrast to most other performance-driven synthesis methods, our synthesis algorithm achieves its best results for the largest circuits, which cannot be handled efficiently by the other methods. For small controllers, the constant hardware overhead for the token control logic is dominating, and other methods may perform better. As three flip-flops are already needed for token control, we investigated only controllers with more than 32 states in our practical experiments.

In the first step, we applied the state-splitting and coloring techniques to obtain three-colorable state transition diagrams. We used example control units from the finite-state machine benchmark set (version 4) distributed at the 1993 Workshop on Logic Synthesis. Table 1 shows the number of states $|S|$ in the original specification and the number of states $|S_{total}|$ after graph transformation. The three rightmost columns show the number of states in each register.

It is interesting that in most cases only a few transformations of the state transition graphs are sufficient to implement the target structure. The total number of states did not increase significantly after graph transformation except for controller s510. This controller is not part of a data-dominated application, and we wouldn't expect the algorithm to provide the best results here. On average, the increment of the number of states is small, and the bypass pipeline is only moderately larger than the standard structure.

In the second synthesis step, we encoded both the standard structure derived from the original state transition table and the bypass pipeline using Nova. To make the structures testable online, we extended the state codes to parity codes. For the smaller examples, 6 information bits are sufficient for state assignment, and adding 6 additional parity bits corresponds to doubling the information. Therefore, we considered 6 check bits the maximum effort for online checking. If more than 2 parity bits were generated, each state bit was covered by 2 or more check bits for detecting multiple errors. To compare the hardware costs of the different implementations, Table 2 shows representative gate-level equivalents necessary for self-checking standard structures and self-checking bypass pipelines.

Table 2. Area in gate-level equivalents.

	Gate-level equivalents for standard (S) and target (T) structures with *i* parity bits							
	i = 0		*i* = 2		*i* = 4		*i* = 6	
Circuit	S	T	S	T	S	T	S	T
planet	843	950	964	1,084	1,089	1,229	1,154	1,342
s1488	987	1,277	1,118	1,198	1,197	1,371	1,460	1,508
s1494	955	1,190	1,114	1,241	1,269	1,431	1,294	1,542
scf	1,195	1,421	1,485	1,682	1,711	1,829	1,897	1,991
s298	2,828	3,152	3,809	3,923	4,771	4,727	5,041	5,438
s510	381	897	511	1,086	625	1,248	675	1,369

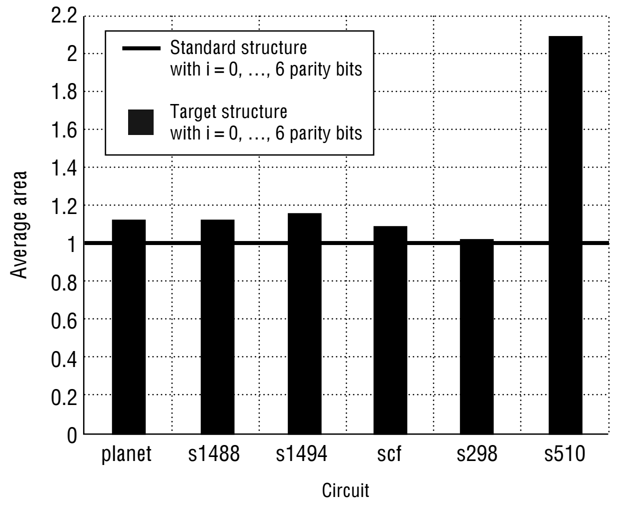

Figure 7. Average area increase.

The columns with zero check bits give the numbers for the basic structures without online checking. The results show the same trends indicated in Table 1. For all examples except s510, the bypass pipeline is only moderately larger than the standard structure. Figure 7 confirms these results, showing the average area costs over all experiments, normalized with respect to the results for the standard structure.

Table 3 indicates that this small increase in area corresponds to a significant performance improvement. It shows the propagation delay for representative configurations. The delay of the slowest stage plus the delay of the additional multiplexers and registers determine the bypass pipeline's speed. The bypass pipeline's total delay is significantly smaller than that of the standard solution. Figure 8 summarizes the average results over all experiments. For some examples, it shows performance improvements greater than 30%.

Table 3. Propagation delay.

	Delay (ns) for standard (S) and target (T) structures with *i* parity bits							
	i = 0		*i* = 2		*i* = 4		*i* = 6	
Circuit	S	T	S	T	S	T	S	T
planet	31.56	22.23	32.50	23.24	33.91	24.20	33.34	24.54
s1488	37.17	25.29	37.57	23.51	38.12	23.88	40.58	26.23
s1494	36.23	28.46	36.59	24.76	37.43	25.32	37.77	25.22
scf	39.34	25.84	42.08	26.64	44.55	26.87	45.11	27.65
s298	73.56	53.84	89.80	68.25	103.50	77.89	104.60	89.00
s510	23.39	21.08	26.01	23.07	26.13	23.21	26.13	23.47

THE BYPASS PIPELINE presented here combines the advantages of high-performance controller structures with online error detection/error correction in highly dependable applications. Our experiments show that compared to standard real-

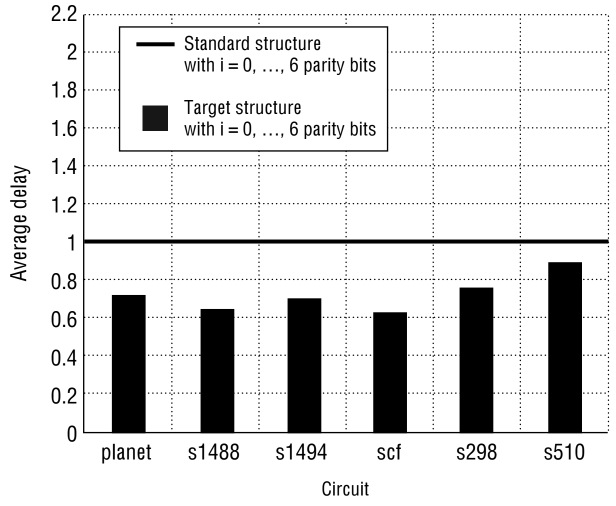

Figure 8. Average performance improvement.

izations, the bypass pipeline can achieve a significant gain in performance at very low cost. The method is unsuitable only for a control-dominated design, such as finite-state machine s510, which has a large number of wait states. The method is compatible with the latest methods of finite-state machine decomposition, state encoding, and logic synthesis. D&T

References

1. P. Ashar, S. Devadas, and A.R. Newton, "Optimum and Heuristic Algorithms for an Approach to Finite-State Machine Decomposition," *IEEE Trans. Computer-Aided Design*, Vol. 10, No. 3, Mar. 1991, pp. 296-310.
2. S. Hellebrand and H.-J. Wunderlich, "An Efficient Procedure for the Synthesis of Fast Self-Testable Controller Structures," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design*, IEEE Computer Society Press, Los Alamitos, Calif., 1994, pp. 110-116.
3. S.C.-Y. Huang and W.H. Wolf, "Performance-Driven Synthesis in Controller-Datapath Systems," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, Vol. 2, No. 1, Mar. 1994, pp. 68-80.
4. D.K. Pradhan, *Fault-Tolerant Computer System Design*, Prentice-Hall, Upper Saddle River, N.J., 1996.
5. T.R.N. Rao, *Error Control Coding for Computer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1989.
6. D.B. Armstrong, "A General Method of Applying Error Correction to Synchronous Digital Systems," *Bell Systems Tech. J.*, Vol. 40, No. 2, Mar. 1961, pp. 577-593.
7. R. Leveugle and G. Saucier, "Optimized Synthesis of Concurrently Checked Controllers," *IEEE Trans. Computers*, Vol. 39, No. 4, Apr. 1990, pp. 419-425.
8. R.A. Parekhji, G. Venkatesh, and S.D. Sherlekar, "Concurrent Error Detection Using Monitoring Machines," *IEEE Design & Test of Computers*, Vol. 12, No. 3, Fall 1995, pp. 24-31.
9. A. Hertwig and H.-J. Wunderlich, "Fast Controllers for Data-Dominated Applications," *Proc. European Design and Test Conference*, IEEE CS Press, 1997, pp. 84-89.
10. T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations," *IEEE Trans. Computer-Aided Design*, Vol. 9, No. 9, Sept. 1990, pp. 905-924.

Sybille Hellebrand is an assistant professor in the Division of Computer Architecture at the University of Stuttgart, Germany. Previously, she worked as an assistant professor at the University of Siegen, Germany. Her main research interests include BIST for high-quality applications and synthesis of testable systems. Hellebrand received her diploma degree in mathematics from the University of Regensburg, Germany. She received the PhD from the Institute of Computer Design and Fault Tolerance, University of Karlsruhe, Germany. She was a postdoctoral fellow at the TIMA/IMAG Computer Architecture Group, Grenoble, France. She is an associate member of the IEEE and a member of the GI (German Society for Computer Science).

Hans-Joachim Wunderlich is a full professor and head of the Division of Computer Architecture at the University of Stuttgart. He previously was an assistant professor at the University of Karlsruhe and a full professor at the University of Siegen, Germany. He is the author or coauthor of four books and more than 80 papers on test, BIST, and fault tolerance. Wunderlich received the diploma degree in mathematics from the University of Freiburg, Germany, and the PhD in computer science from the University of Karlsruhe. He is an associate member of the IEEE and a member of the ACM and the GI.

Andre Hertwig worked as a research and teaching assistant in the Division of Computer Architecture, University of Stuttgart, during the work described in this article. His research interests include synthesis of testable systems and low-power design. Hertwig received his diploma degree in computer science from the University of Siegen.

Send questions or comments about this article to Sybille Hellebrand, University of Stuttgart, Division of Computer Architecture, Breitwiesen Str. 20/22, D-70565 Stuttgart, Germany; sybille@ramona.informatik.uni-stuttgart.de.