Concurrent Error Detection Methods for Asynchronous Burst-Mode Machines

Sobeeh Almukhaizim, *Student Member*, *IEEE*, and Yiorgos Makris, *Member*, *IEEE*

Abstract—Asynchronous controllers exhibit various characteristics that limit the effectiveness and applicability of the Concurrent Error Detection (CED) methods developed for their synchronous counterparts. Asynchronous Burst-Mode Machines (ABMMs), for example, do not have a global clock to synchronize the ABMM with the additional circuitry that is typically used by synchronous CED methods (for example, duplication). Therefore, performing effective CED in ABMMs requires a synchronization method that will appropriately enable the checker (for example, comparator) in order to avoid false alarms. Also, ABMMs contain redundant logic, which guarantees the hazard-free operation required for correct interaction between the circuit and its environment. Redundant logic, however, allows some single event transients to manifest themselves only as hazards but not as logic discrepancies. Therefore, performing effective CED in ABMMs requires the ability to detect hazards with which synchronous CED methods are not concerned. In this work, we first devise hardware solutions for performing checking synchronization and hazard detection. We then demonstrate how these solutions enable the development of three complete CED methods for ABMMs. The first method (Duplication-based CED) is an adaptation of the well-known duplication method within the context of ABMMs. The second method (Transition-Triggered CED) is a variation of duplication wherein the implementation cost is reduced by allowing hazards in the duplicate circuit. In contrast to these two methods, which are nonintrusive, the third method (Berger code-based CED) is intrusive since it requires reencoding of the ABMM with check symbols based on the Berger code. Although this intrusiveness may slightly impact performance, Berger code-based CED incurs the lowest area overhead among the three methods, as indicated through experimental results.

Index Terms—Concurrent error detection, asynchronous burst-mode machines, error-detecting codes, Berger code.

1 Introduction

The numerous advantages promised by asynchronous circuits have recently sparked renewed interest in the development of CAD methods and tools to automate their realization. As these methods mature, the size and complexity of the circuits that they support increase significantly, necessitating similar efforts in order to verify and test their functionality before deployment. At the same time, due to their low power consumption and electromagnetic noise emission, asynchronous circuits are also gaining a strong foothold in mission-critical applications such as avionics and communications. As a result, Concurrent Error Detection (CED) methods, that is, methods that monitor their operation in the field in order to detect and report potential malfunctions occurring due to single event transients, are becoming of increasing importance.

In this paper, we address the problem of CED in Asynchronous Burst-Mode Machines (ABMMs). Although a wide variety of CED methods have been developed for synchronous controllers [1], [2], [3], [4], their asynchronous counterparts are intrinsically different [5], limiting the applicability of these methods. Consider, for example, the traditional duplication method in synchronous circuits, wherein a duplicate of the circuit is added and a comparator continuously checks the two results for consistency. When a similar approach is attempted for CED in ABMMs, its use and effectiveness are jeopardized in two ways. First, the lack of a global clock allows a circuit and its duplicate to produce results autonomously and at their own pace. Consequently, the outputs of these circuits are not continuously equal. Therefore, in order to avoid false alarms occurring from their continuous comparison during CED, a *checking synchronization* method is required. Second, single event transients in redundant logic, which is used to ensure the hazard-free operation of an ABMM, may cause only hazards but no functional discrepancy. Such hazards will go undetected by CED methods that only check the functionality of the circuit. However, ABMM responses have to be not only correct but also hazard-free. Otherwise, a hazard may be misinterpreted by the environment as a logic value change, resulting in erroneous interaction with the circuit and, by extension, erroneous system-level results. Therefore, to ensure that CED methods will detect errors in the interaction between the ABMM and its environment, a *hazard detection* method is also required.

Our aim is to enable the development of CED methods for ABMMs by devising solutions to the two aforementioned problems. In order to address the checking synchronization problem, we propose a method that utilizes control information inherent in the operation of ABMMs. More specifically, our method uses a *Transition Prediction Function* (TPF), which is derived from the functionality of the ABMM, to indicate the completion of computation. In order to address the hazard detection problem, we propose the

- *S. Almukhaizim is with the Department of Electrical Engineering, Yale University, New Haven, CT 06520-8285. E-mail: sobeeh.almukhaizim@yale.edu.*
- *Y. Makris is with the Departments of Electrical Engineering and Computer Science, Yale University, New Haven, CT 06520-8285. E-mail: yiorgos.makris@yale.edu.*

Manuscript received 24 Mar. 2006; revised 24 Sept. 2006; accepted 10 Nov. 2006; published online 13 Feb. 2007.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0115-0306.
Digital Object Identifier no. 10.1109/TC.2007.1025.

addition of specialized circuitry that detects errors causing only hazards but no functional discrepancy at the outputs of the circuit. By using the checking synchronization method and the Hazard Detection Circuit (HDC), we develop three CED methods, all of which guarantee detection of all functional errors and hazards in ABMMs. The first one is the *Duplication-based CED* method, which is an adaptation of the duplication method frequently used in synchronous circuits. The second one is the *Transition-Triggered CED* method, which is a variation of duplication wherein the area overhead is reduced by allowing hazards in the duplicate circuit and eliminating the corresponding redundant logic. The third one is the *Berger code-based CED* method, which encodes the outputs of the controller with a check symbol based on the Berger code. Although encoding the outputs of an ABMM by using the Berger code is straightforward, the key challenge is to ensure that the state encoding adheres to the conditions required to realize an ABMM implementation of the circuit and its Berger code generator. To this end, we develop a state encoding method which guarantees the existence of an inverter-free ABMM implementation of the original circuit,[1] as well as an ABMM implementation of the corresponding Berger code generator. Both Duplication-based CED and Transition-Triggered CED are nonintrusive, that is, they only add hardware in parallel to the original circuit, which is left intact, and, thus, the performance is preserved. In contrast, Berger code-based CED is intrusive since it requires state reencoding. As a result, performance is slightly affected, yet the area overhead is smaller, as corroborated experimentally.

The rest of this paper is organized as follows: In Section 2, we review related work in CED for asynchronous circuits. In Section 3, we briefly describe the class of ABMMs. In Section 4, we discuss the challenges of applying synchronous CED methods to ABMMs and present the proposed solutions. Then, in Sections 5, 6, and 7, we describe the Duplication-based CED, Transition-Triggered CED, and Berger code-based CED methods for ABMMs, respectively. Finally, in Section 8, we assess experimentally the area and performance overhead of the presented methods and discuss their effectiveness.

2 Related Work

Several CED methods for asynchronous circuits have been previously proposed in the literature, none of which, however, is applicable to ABMMs. In [5], the authors investigate the applicability of Duplication-based CED to asynchronous circuits and show that comparison synchronization can be performed using a modified comparator that exploits the local synchronization protocols that exist in many asynchronous circuits. A time window is utilized within which the original circuit and its duplicate should both arrive at the same state. The use of a time window, however, results in the masking of some performance-related faults. Thus, a dedicated hardware monitor is also required in order to detect these malfunctions.

1. Berger code-based CED methods require circuit implementations wherein inverters are only allowed on the primary inputs.

In [6], a CED method for Quasi Delay-Insensitive (QDI) circuits is proposed based on unordered codes. A code is unordered if no code word u is contained in another code word v (that is, $u \subseteq v$), where $u \subseteq v$ if v has a 1 in every bit position where u has a 1. Examples of unordered codes are the one-hot encoding and the dual-rail code, which is typically used to encode the inputs and outputs of QDI circuits. Since the inputs and outputs of a QDI circuit are encoded using unordered codes, which naturally leads to an inverter-free realization of these circuits, any single fault in the circuit will produce a noncode word at the output. Moreover, a spacer $(00\ldots0)$ appears at the output of the QDI circuit between working phases and is used by a Completion-Detection (CD) circuit to indicate the validity of the results. Several CD implementations have been proposed in the literature. In [7], the feasibility of designing VLSI decoders to implement the CD test on asynchronous buses is investigated. In [8], the authors present an activity-monitoring CD method which detects completion if no transitions are observed in the circuit within a period of time and a transition-monitoring CD method which guarantees the detection of all glitches in the completion logic. Finally, an efficient and systematic method to perform CD by using multioutput threshold logic is presented in [9]; an in-depth discussion of various CD strategies can be found in [10].

CED methods have also been proposed for asynchronous circuits through the use of the Differential Cascode Voltage Switch (DCVS) logic [11], [12], which is used to generate handshaking signals between interacting asynchronous modules. In [11], the authors utilize the handshaking signals of the DCVS logic to design a self-checking majority voter. The self-checking majority voter is implemented using a dual voter with self-checking capabilities: The handshaking signals are used to assert the self-checking signals of the voter if the monitored data is not complementary. In [12], a CED method is presented for the class of self-timed VLSI pipelines implemented using DCVS logic. A checker, composed of a tree of dual-rail comparators, compares the signal pairs from the DCVS logic of the pipeline stages to detect errors. In [13], a mixed-signal approach to perform CED for self-timed circuits is proposed. The authors argue that logical faults in these circuits might be undetectable at the primary outputs. Thus, performing CED by using logic (voltage) monitoring would not be effective for all possible faults. A study on the feasibility of performing CED for these circuits by using a Built-In Current Sensor (BICS) is therefore presented.

More recently, a CED method for asynchronous interfaces in globally asynchronous locally synchronous circuits is described in [14]. A checker design is proposed to monitor the handshake control signals at the asynchronous interface of these circuits. If a fault is present, then the system halts and a timeout circuit is used to detect and indicate it. The concurrency of the CED method in [14] was improved in [15]: Whenever the checker detects an error in the handshake control signals exchanged between circuits interacting through their asynchronous interface, D flip-flops are used to latch the error information and report it immediately.

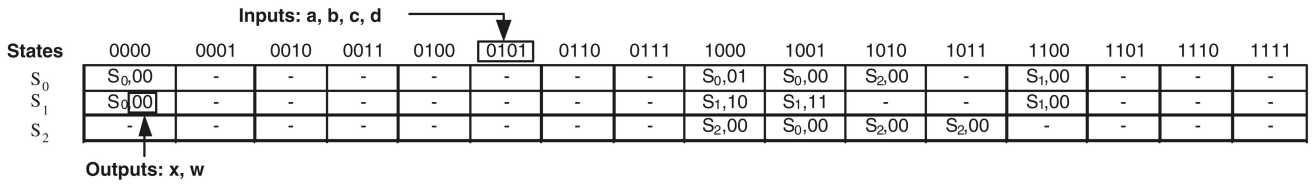

Fig. 1. Example of a symbolic state transition table for defining an ABMM.

Although the above methods sufficiently address the problem of CED in the classes of asynchronous circuits that they were developed for, there are two factors that limit their applicability in ABMMs. First, all of these methods assume the existence of *explicit* completion signals and utilize them in order to synchronize correctness checking. ABMMs, however, operate without completion signals and, therefore, these CED methods cannot be applied. Second, these methods assume that all errors of interest will result in a functional discrepancy. However, ABMMs contain redundant logic wherein transient errors may result only in hazards but no functional discrepancy at the output. Such hazards jeopardize the correct communication of the circuit with its environment and therefore should also be detected. These two limitations motivate the need for development of CED methods specific to ABMMs and pinpoint the contributions of this work.

3 Asynchronous Burst-Mode Machines

ABMMs are widely used for designing asynchronous controllers [16], [17], [18], [19] since they promise improved characteristics as compared to their synchronous counterparts. For example, the performance of a burst-mode implementation of an instruction decoder has been reported as three times better than the performance of a highly tuned synchronous version [20]. Moreover, their low power consumption makes them highly useful in portable applications [21]. In this section, we briefly introduce their fundamentals, we outline the synthesis process for realizing an ABMM implementation from a given Finite-State Machine (FSM) description, and we give an example.

3.1 Fundamentals

ABMMs constitute a class of *Huffman* circuits [22] that is widely used for designing asynchronous controllers [16], [17], [18], [19]. Huffman circuits consist of a set of combinational functions, computing the next state and output of the circuit, and a set of feedback lines, storing the state of the circuit. No clock and no state registers are used in these circuits; however, delay elements are often added to eliminate *essential hazards*[2] [23]. Given the absence of a clock, *communication protocols* are needed to ensure correct interaction between an asynchronous circuit and its environment. These protocols define the properties of the stimuli that the environment is allowed to provide to the circuit, as well as the properties of the responses that the circuit will generate. Based on these protocols, various classes of asynchronous circuits are defined.

The key aspect of the protocol used in ABMMs, as indicated by their name, is that the interaction between the circuit and its environment happens in *bursts*. An *input burst* is defined as a set of bit changes in one or more inputs of the circuit, which are allowed to occur in any order and without any constraint in their relative time of arrival. Once an input burst is complete, and *only* then, the circuit responds to the environment through a hazard-free state and output change. We emphasize the protocol requirement for hazard-free state and output changes. Since no clock is used, synchronization between the circuit and its environment is based on the fact that any change in the state or output of the circuit signifies completion of an evaluation cycle. Therefore, all hazards should be eliminated to ensure correct interaction of an ABMM with its environment. In order to implement a circuit that complies with the aforementioned communication protocol, two features are added during synthesis. First, in order to make the functionality of the circuit critical race-free,[3] *dichotomies* are added to constrain the binary state encoding of the circuit [24]. Consequently, the resulting state codes ensure that a transition between two states never reaches a transient state with a different destination state for the current input. Second, to make the next state/output functions hazard-free, *redundant implicants* are added to their implementation [25].

ABMMs have been extensively studied and are particularly popular in the asynchronous community, partly due to the availability of MINIMALIST [16], [17], [18], [19], which is a comprehensive burst-mode logic synthesis and optimization package. MINIMALIST produces a hazard-free ABMM implementation as long as the following two constraints are satisfied: First, input bursts at any given state should be nonempty and should unambiguously decide the next state; hence, no input burst should be a subset of another input burst. This constraint is called the *maximal set property*. Second, every state should be entered using a unique input burst. This constraint is called the *unique entry point*. If the specifications of a controller satisfy the above constraints, then an ABMM implementation is guaranteed to exist and MINIMALIST produces the minimal-cost hazard-free logic implementation.

3.2 Example

An ABMM is described using a state transition table such as the one shown in Fig. 1. The rows in the table correspond to the current symbolic state, the columns correspond to the

2. Essential hazards arise when a state change completes before the input change is fully processed. To prevent this early state change from propagating through the combinational logic, delay may be added to the feedback.

3. A critical race hazard exists if two state variables change value and the machine's next state depends on the order of arrival of these changes [22], [23].

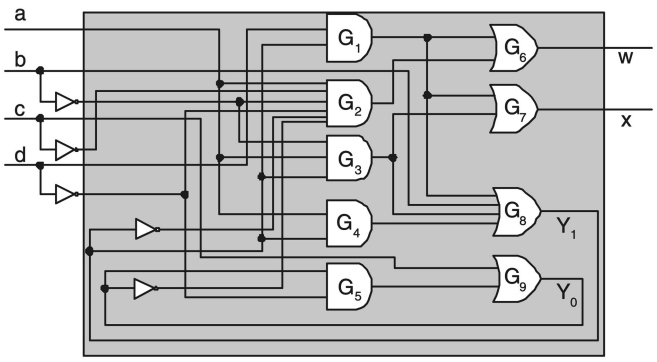

Fig. 2. ABMM implementation of the example.

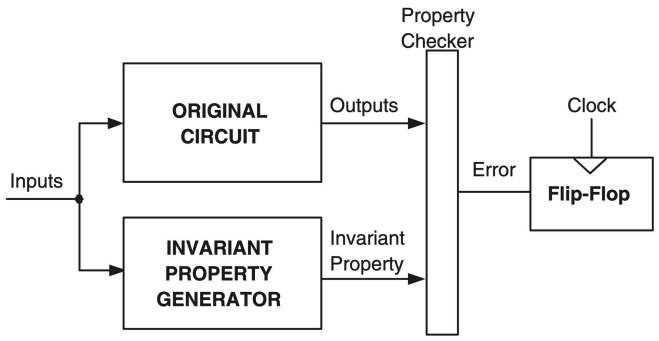

Fig. 3. General structure of synchronous CED methods.

inputs, and the entry indicates the next state and the outputs. For example, suppose that the circuit is in state S_0. Then, an input burst of 1010 will cause a transition to state S_2 and will generate an output of 00. Let us now assume that the next input burst is 1001 (that is, input c is lowered and input d is raised) and that c is lowered first and, then, d is raised (that is, $1010 \rightarrow 1000 \rightarrow 1001$). The circuit responds only after the input burst is complete, so, between the time that c is lowered and the time that d is raised, the next state and output bits do not change. Once the input burst is complete, the circuit makes a transition to state S_0 and computes the output, which, in this case, happens to remain the same, that is, 00.

We note that, depending on the encoding of the states, a critical race may occur during this transition. For example, if the states are encoded as $S_0 = 00$, $S_1 = 01$, and $S_2 = 11$, then the transition from S_2 to S_0 may go through a transient state of 01, which is the state encoding of S_1. In combination with the current input burst of 1001, this will produce a next state of S_1 and an output of 10, both of which are incorrect. Thus, this state encoding would be invalid and is avoided through the use of dichotomies [24].

A dash in a table entry signifies that the corresponding combination of current state and input is not permitted by the communication protocol between the circuit and the environment. For example, if the circuit is in state S_1, then an input burst of 0010 is not allowed to occur. The synthesis process of MINIMALIST starts by performing state minimization on the *symbolic* state transition table constrained such that the reduced state transition table has a hazard-free logic implementation [19]. In the example in Fig. 1, the state transition table is already minimal. Next, dichotomies are added to ensure a critical race-free state encoding. Solving the dichotomies results in the state encoding $S_0 = 00$, $S_1 = 01$, and $S_2 = 10$ for the example circuit and the symbolic states are replaced by their binary values. The last step is to generate a minimal-cost *hazard-free* implementation of the circuit [18]. Fig. 2 shows the resulting ABMM implementation of the example, which includes some logic redundancy to ensure hazard-free operation.

4 CED in ABMMs: Challenges and Solutions

The general structure of a CED method for synchronous circuits is illustrated in Fig. 3. CED is typically based on an *invariant property* of the circuit, which is generated through additional hardware. Examples of such invariant properties include the output of the circuit itself (duplication), the parity of the output [2], [26], the encoding of the output by using Error-Detecting Codes (EDCs) [27], [28], and the value of internal gates [29], [30]. A property checker determines whether the invariant property of the output produced by the monitored circuit matches the one generated by the property generator. An error signal indicates a mismatch between the circuit and the invariant property generator and a flip-flop latches the error signal at the end of the clock cycle. Since the checker is crucial to the correct operation of CED methods, self-checking checkers are commonly used [31], [32], [33].

ABMMs pose two additional challenges over and above their synchronous counterparts for effective CED. The first challenge arises from the fact that the ABMM and the invariant property generator operate asynchronously and there is no clock to indicate when their combined output should be checked. Therefore, in order to avoid false alarms, a *checking synchronization* method is required. The second challenge arises from the fact that, functionally, redundant hardware is included in ABMMs in order to ensure their hazard-free operation. As a result, there exist errors that will only cause hazards but no functional discrepancy at the state or output bits. Therefore, additional provisions for performing *error-induced hazard detection* are required. In this section, we elaborate on these two challenges and propose solutions to enable CED in ABMMs.

4.1 Checking Synchronization

The lack of a synchronizing clock introduces uncertainty as to when to check the responses of the ABMM and the invariant property generator. Therefore, commonly used synchronous CED methods such as Duplication-based CED cannot be directly applied on ABMMs without enforcing that the outputs be checked *only* when both have finished computation; otherwise, false alarms may occur. Process variations, input skew, and the sheer fact that the two circuits are separate entities are a few of the reasons why two identical circuits may operate with different delays. Therefore, in order to avoid false alarms, a *checking synchronization* method is required.

In error-free operation, the output of the invariant property checker constitutes a valid check as long as the outputs of the ABMM and the invariant property generator are not making a transition. Based on the definition of

States	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
S_0	S_0,1	-	-	-	-	-	-	-	S_0,1	S_0,1	S_2,1	-	S_1,1	-	-	-
S_1	S_0,1	-	-	-	-	-	-	-	S_1,1	S_1,1	-	-	S_1,1	-	-	-
S_2	-	-	-	-	-	-	-	-	S_4,0	-	S_2,1	S_3,0	-	-	-	-
S_3	-	-	-	-	-	-	-	-	-	S_0,1	-	S_3,0	-	-	-	-
S_4	-	-	-	-	-	-	-	-	S_4,0	S_0,1	-	-	-	-	-	-

Fig. 4. Symbolic state transition table for the TPF.

ABMMs, these outputs are steady during an input burst and only change after the input burst is complete. Thus, any transition during the input burst can be safely attributed to an error and can be detected by activating the invariant property checker throughout this period. When the input burst is complete, however, the ABMM and the invariant property generator operate asynchronously. Thus, during the period between the end of an input burst and the beginning of the next, the invariant property checker is deactivated. Once the new input burst starts, the checker is reactivated to detect any errors in the previous output burst.

In order to deactivate the checker in between input bursts, we use a *TPF*, which signifies the end of an input burst. Since the TPF controls the checker, its implementation must be hazard-free to ensure that no false positives or false negatives occur. Therefore, the TPF is implemented as an ABMM. The TPF is defined for every specified entry in the state transition table of the ABMM and obtains a logic value of "1" when the current input burst and state combination result in a state transition and/or a change at an output and a logic value of "0" otherwise. For all input bursts that are composed of k bit changes, where $k > 1$, the above definition implies that the TPF is lowered when the first bit changes and then raised again when the input burst is complete. Hence, such an input burst results in multiple output bursts, which violates the maximal set property and the unique entry point constraints of ABMMs. In order to satisfy these constraints, input bursts with k bit changes are decomposed into k input bursts with a single bit change. Each of these k input bursts lowers the TPF and makes a state transition into a dummy state. Then, an input burst with the remaining $k-1$ bit changes is added to every dummy state in order to raise the output and make a transition to the next state to which the original input burst would lead. Therefore, the modified input bursts will unambiguously decide the next state and output and, hence, the maximal set property is satisfied. Moreover, every state in the specification of the TPF can be entered using a unique input burst and, hence, the unique entry-point constraint is also satisfied. Since both constraints are satisfied, a hazard-free ABMM implementation of the TPF is guaranteed to exist.

Example. Consider the controller described in Fig. 1, which has the TPF, shown in Fig. 4. All of the input bursts defined for states S_0 and S_1 have a single bit change and result in a state transition and/or a change in the output. Therefore, all the defined entries in the state transition table of the TPF for S_0 and S_1 have an output of "1." S_2, on the other hand, has an input burst with two bit changes. Hence, this input burst is replaced with two input bursts, each of which has a single bit change. The input bursts that are added to S_2 lower the output and lead to a state transition to two dummy states: S_3 and S_4. In each of these two states, an input burst consisting of the remaining bit changes from the original input burst is added. The input burst in S_3 and the input burst in S_4 will both raise the output and make a transition to state S_0.

4.2 Detection of Error-Induced Hazards

Logic redundancy in ABMMs prevents hazards from occurring during error-free operation, as required by the communication protocol between an ABMM and its environment. As a result, some errors may cause only hazards but no functional discrepancy, so they cannot be detected by checking an invariant property of the output. Therefore, in order to monitor the correct interaction of the circuit and its environment, a hazard detection method is also required.

Example. Assume that the current state in the circuit in Fig. 2 is S_1 and the input changes from 1100 to 1000. Then, the next state should become S_1 and output X should obtain a logic "1" value, as indicated in Fig. 1. However, if an error inducing a logic value of "0" at the output of gate G_4 occurs, then a hazard will appear at output X. This is illustrated in the timing diagram in Fig. 5, wherein dotted lines represent the logic values in the absence of this error. In this example, the change of input b affects the next state function Y_1 before the change in gate G_3 reaches Y_1. During that time, Y_1 is at the logic value of "1" due to gate G_4. Hence, an error in gate G_4 will generate a hazard at Y_1. Subsequently, the hazard at Y_1 will propagate to the output of gate G_3, which, in turn, will result in a hazard at output X of the circuit. Thus, an error at gate G_4 will cause a hazard at Y_1, G_3, and output X but no functional discrepancy at any state or output signals.

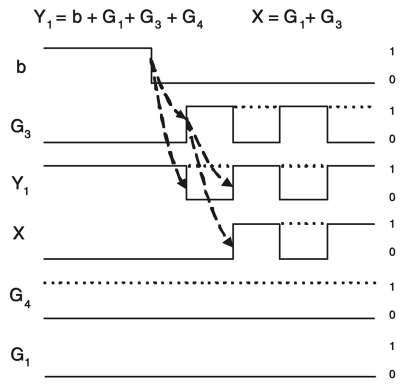

Fig. 5. Timing diagram illustrating a hazard.

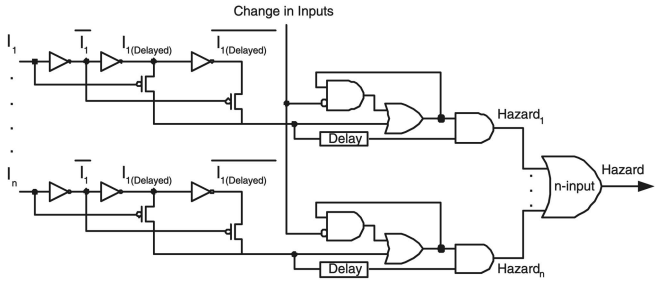

Fig. 6. Hazard detection circuit.

The number of errors that cause only hazards at the output of an ABMM can be quite substantial, exceeding 30 percent in many circuits. In order to detect error-induced hazards, a mechanism to monitor and indicate the occurrence of multiple transitions at any output of the ABMM is required. This can be performed using the HDC illustrated in Fig. 6. This circuit monitors the ABMM by using a separate feedback loop for each output. A feedback loop latches a logic "1" value whenever a transition is detected on the corresponding output. We remind that the outputs of an ABMM are expected to change only after the input burst is complete, at which point they make hazard-free transitions. Thus, if a second transition is detected on the same output before a new input burst occurs, then the hazard signal is asserted. As soon as a new input burst occurs, however, the latched transitions must be reset. For this purpose, the change detection circuit (CDC) in Fig. 7 is used to generate a short reset pulse after each input change, which forces the feedback loops to latch a logic "0" value every time an input changes, thus clearing the latched transitions.

The operation of the HDC is illustrated using the PSpice timing diagram in Fig. 8. We simulate the functionality of the HDC when a hazard appears at one of its inputs, which amounts to observing a hazard at the corresponding monitored output of the ABMM. In this example, the input makes its first transition at time $1ns$ and the feedback loop detects the transition and latches a logic "1" value in the *feedback* signal. Then, a second transition is detected at time $2ns$ and the *hazard* signal is asserted shortly after. Finally, the timing diagram also illustrates the behavior of the circuit when a new input is applied at time $4ns$, which resets the transition latched in the feedback loop.

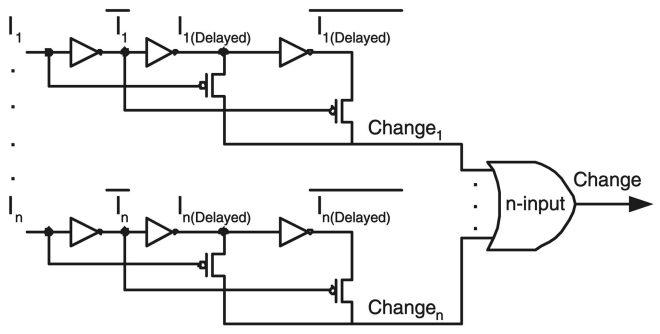

Fig. 7. Change detection circuit.

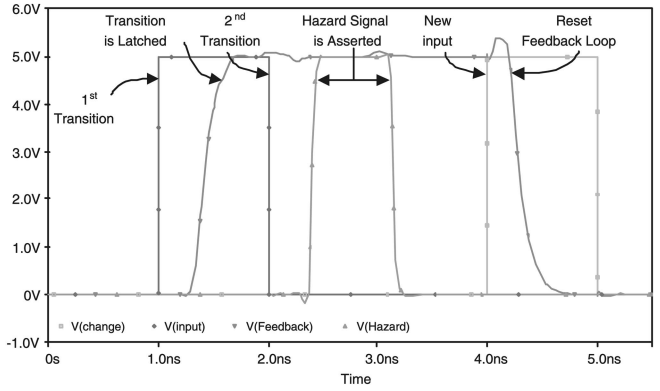

Fig. 8. Timing diagram illustrating the operation of the HDC.

In order to latch a transition in the feedback loop, the width of the hazard must be larger than the delay of the AND and OR gates used in the feedback loop. This width depends on the delays of gates internal to the circuit. In case that it is not adequately large, the design in Fig. 6 can be modified to detect hazards with smaller widths by appropriate transistor sizing of the gates used in the feedback loop. Finally, we note that the HDC does not have to monitor the state lines. A hazard on a state line will *always* result in either a functional error or a hazard at the output lines. Thus, monitoring the output lines suffices for detecting all error-induced hazards in the circuit.

5 Duplication-Based CED

The Duplication-based CED method is illustrated in Fig. 9. In order to perform Duplication-based CED for ABMMs, a replica of the ABMM is used as an invariant property generator and a comparator checks the results of the two circuits to identify any error-induced functional discrepancies. Then, the ABMM implementation of the TPF is added to enable/disable the comparator based on the checking synchronization method described in Section 4.1. Subsequently, the HDC, including the CDC, is added to indicate the occurrence of errors that only cause hazards but no functional discrepancy in the original circuit. Since the duplicate circuit does not interact with the environment, hazard detection in the duplicate circuit is not required.

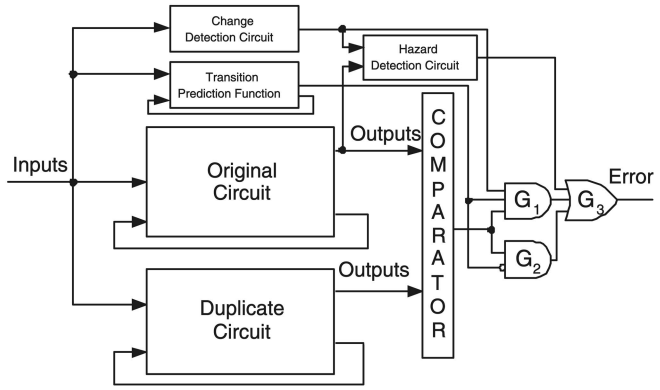

Fig. 9. Duplication-based CED.

States	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
S_0	S_0,00	-	-	-	-	-	-	-	S_0,01	S_0,00	S_2,00	-	S_1,00	-	-	-
S_1	S_0,00	-	-	-	-	-	-	-	S_1,10	S_1,11	-	-	S_1,00	-	-	-
S_2	-	-	-	-	-	-	-	-	-, --	S_0,00	S_2,00	-, --	-	-	-	-

Fig. 10. Symbolic state transition table for the optimized duplicate.

Finally, a few glue-logic gates (G_1-G_3) are used to generate the error output. The error signal is asserted in the following scenarios: 1) The comparator detects a mismatch between the outputs of the original and the duplicate circuit after a new input burst is applied, in which case G_1 indicates an error, 2) the comparator detects a mismatch between the outputs of the original and the duplicate circuit during an input burst, in which case G_2 indicates an error, or 3) a hazard is detected at the outputs of the original circuit by the HDC, in which case G_3 indicates an error. The above error detection scenarios collectively provide complete detection ability for all functional errors and hazards in the circuit.

6 Transition-Triggered CED

In an effort to reduce the area overhead of the duplicate circuit used in Duplication-based CED, we propose in this section a *Transition-Triggered CED* method, which is illustrated in Fig. 11. Our method starts with the observation that hazards in the duplicate circuit do not affect the communication protocol between the original circuit and the environment. Therefore, the redundant logic used to make the original circuit hazard-free is not necessary in the duplicate circuit as long as the CED operation is not compromised. By allowing hazards to occur in the duplicate circuit, its logic implementation can be optimized, which results in reducing its area cost. Hazards on the feedback lines of the optimized duplicate, however, jeopardize the operation of the asynchronous circuit since a hazard may be misinterpreted by the circuit as a change in state. To avoid this problem, multiplexers are utilized on the state lines of the optimized duplicate circuit to select between the current state and the next state. The multiplexers are controlled using the TPF and the CDC to select the next state only when the input burst is complete.

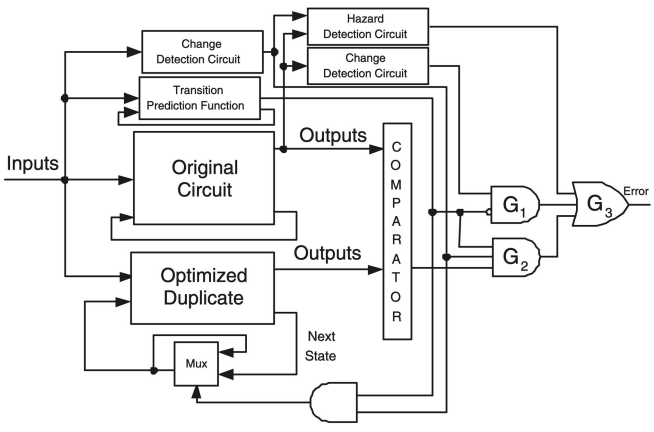

Fig. 11. Transition-triggered CED.

Interestingly, the TPF can be used in conjunction with the CDC to further reduce the cost of CED. The TPF indicates when changes *should occur* at the next state and/or output signals, whereas the CDC in Fig. 7, when added to the output lines, indicates when such changes *actually occur* at the outputs of the circuit. Hence, an error in the original circuit would result in a mismatch between these two signals *during input bursts* and will be detected using G_1. Therefore, the result of the comparator becomes necessary for G_2 only after an input burst is complete and, thus, the functionality of the optimized duplicate can be considered as "don't care" for incomplete input bursts, allowing further logic optimization. The state transition table of the optimized duplicate for the example in Fig. 1 is illustrated in Fig. 10. In contrast to the table in Fig. 1, only entries with a change in the next state and/or output lines are defined for the duplicate circuit; the rest are "don't cares."

Transition-Triggered CED indicates an error in the following scenarios: 1) The output of the original ABMM changes and the TPF indicates no expected change at the output, in which case G_1 indicates an error, 2) the comparator detects a mismatch between the outputs of the original and the duplicate circuit after a new input burst is applied, in which case G_2 indicates an error, or 3) a hazard is detected at the outputs of the original circuit by the HDC. Similarly to Duplication-based CED, the above error detection scenarios, collectively, provide complete detection ability for all functional errors and hazards in the circuit.

The design of the optimized duplicate, the TPF, and the state multiplexers is similar to asynchronous design styles with local clocking [34], [35]. The proposed Transition-Triggered implementation is mostly similar to the work of Nowick and Dill [36], [37] on the design of ABMMs using a local clock. The method proposed in [36] and [37] is for the design of a standalone asynchronous machine that interacts with the environment and, thus, is *hazard-free*. In contrast, the proposed method is used in the CED circuitry and, hence, hazards in the duplicate circuit are *allowed* as long as these hazards do not interfere with the detection of errors.

7 Berger Code-Based CED

In this section, we first briefly review the Berger encoding. Then, we discuss the *two key components* necessary to design a Berger-encoded ABMM: 1) an inverter-free implementation of the original ABMM and 2) an ABMM implementation of the Berger code generator. Finally, we present the complete Berger code-based CED method for ABMMs.

7.1 Berger Encoding

In the Berger code [27], which is an optimal systematic All-Unidirectional Error-Detecting (AUED) code, the r information bits are encoded using k check bits to form an n-bit code word, where $n = r + k$. The k-bit check symbol is the

complement of the binary representation of the number of 1s in the information bits and, hence, $k = \lceil log(r+1) \rceil$. In a Berger-encoded circuit, single errors lead to a noncode word as long as they result in transitions in either the $0 \rightarrow 1$ or the $1 \rightarrow 0$ direction at the output but not both. The authors in [28] show how this constraint can be enforced by redesigning a circuit such that inverters only appear at the inputs. Thus, all single errors will only cause unidirectional effects at the r outputs. A Berger code generator is then added to produce the k-bit check symbol. The r-bit output of the inverter-free circuit, combined with the k-bit check symbol of the Berger code generator, forms a Berger-encoded word, the validity of which is checked using a Berger code checker [31], [38].

In contrast to the circuits targeted in [28], ABMMs impose additional constraints to realize a hazard-free implementation and, therefore, new solutions tailored to their particularities are required. In Sections 7.2 and 7.3, we demonstrate how we can obtain 1) an inverter-free ABMM implementation from a symbolic state transition table and 2) an ABMM implementation of the corresponding Berger code generator.

7.2 Inverter-Free ABMM Implementation

The method in [28] generates an inverter-free implementation of an FSM by pushing the inverters of the combinational logic to the inputs and by utilizing the inverted state bits that are inherently available in the flip-flops of the state register. In ABMMs, however, this is not possible since the state is stored in the combinational feedback and no state register exists. To solve this problem, we propose to reencode the states of the ABMM such that states are uniquely distinguished without the need for inverted state bits. We stress, however, that not all possible state encodings can be used. As mentioned in Section 3.1, certain constraints must be satisfied to guarantee the existence of a hazard-free ABMM implementation.

MINIMALIST uses *dichotomies* [24] to represent the constraints that must be satisfied by the state codes for an ABMM implementation to exist. A dichotomy such as $(Group_1; Group_2)$ specifies two groups of states that must be distinguished from each other and a solution assigns a logic value x to a bit in the state encoding of all the states in one group, say, $Group_1$, while assigning $\overline{x}$ to the same bit in all of the states in the other group, $Group_2$. MINIMALIST encodes the states such that all of the dichotomies are solved and, at the same time, the implementation cost of the resulting ABMM is minimized. In the final encoding, a state is distinguished from incompatible states through either a positive identifier (that is, a bit that is assigned a logic "1" to solve the dichotomy) or a negative identifier (that is, a bit that is assigned a logic "0" to solve the dichotomy). The latter results in inverters on the corresponding state lines. In order to eliminate these inverters, we add to the dichotomies generated by MINIMALIST three additional sets of dichotomies and modify the state encoding procedure such that only positive identifiers are used to solve all of these dichotomies. We next describe the three sets of additional dichotomies by using as an example the symbolic state transition table in Fig. 1. MINIMALIST requires that the state encoding satisfy the dichotomy $(S_0S_2; S_1)$ to ensure that no critical race exists for the state transition from S_2 to S_0. States S_0 and S_1 have multiple single-bit-change input bursts, whereas state S_2 has a single input burst with multiple bit changes. The dichotomy $(S_0S_2; S_1)$ represents the original set of dichotomies, as illustrated in Table 1.

TABLE 1
Dichotomies Used to Generate the State Encoding of an Inverter-Free ABMM Implementation

Dichotomies			
Original	1st Set	2nd Set	3rd Set
$(S_0S_2;S_1)$	$(\mathbf{S_0;S_0})$	$(\mathbf{S_2;S_0})$	$(\mathbf{S_0S_1;S_0})$
	$(\mathbf{S_0;S_1})$	$(\mathbf{S_2;S_1})$	$(\mathbf{S_0S_1;S_1})$
	$(S_0;S_2)$	$(\mathbf{S_2;S_2})$	$(S_0S_1;S_2)$
	$(\mathbf{S_1;S_0})$		$(\mathbf{S_0S_2;S_0})$
	$(\mathbf{S_1;S_1})$		$(S_0S_2;S_1)$
	$(\mathbf{S_1;S_2})$		$(\mathbf{S_0S_2;S_2})$
			$(\mathbf{S_1S_0;S_0})$
			$(\mathbf{S_1S_0;S_1})$
			$(\mathbf{S_1S_0;S_2})$
			$(\mathbf{S_2S_0;S_0})$
			$(\mathbf{S_2S_0;S_1})$
			$(\mathbf{S_2S_0;S_2})$

The first set of additional dichotomies ensures that the output function identifies the current state in states that have multiple input bursts, without the need for detecting inverted state bits. Thus, for each state that has multiple input bursts and every other possible state, the first set of additional dichotomies contains a state that has multiple input bursts on the left side of the dichotomies and one of the remaining states on the right side of the dichotomies. For example, the first set for the example in Fig. 1 is summarized under the second column in Table 1. The states with multiple input bursts S_0 and S_1 appear on the left side of the dichotomies, whereas one of the other states is placed on the right side.

The second set of additional dichotomies ensures the hazard-free operation of the output function during an input burst in states that have an input burst with multiple bit changes, without the need for detecting inverted state bits. Thus, for each state that has an input burst with multiple bit changes and for every other possible state, the second set of additional dichotomies contains a state with multiple bit changes on the left side of the dichotomies and one of the remaining states on the right side of the dichotomies. This is illustrated under the third column in Table 1. For example, since S_2 has a single input burst with multiple input changes, S_2 appears on the left side of the dichotomies, whereas one of the other states is on the right side.

The final set of additional dichotomies ensures that the positive identifiers used to identify a state are not contained in the state encoding of any possible transient state that may appear during state transitions; otherwise, the logic detecting the state might be activated during the state transition and would result in a hazard and/or incorrect results. Thus, for every state transition and state pair, the third set of additional dichotomies contains a state transition on the left side of the dichotomies and a state on the right side of the dichotomies. For example, four state transitions exist in the state transition table in Fig. 1 for the three different states, which results in 12 state and state-transition pairs. For each such pair, a dichotomy is added to the third set of

additional dichotomies, as illustrated under the fourth column in Table 1.

Finally, all invalid and redundant dichotomies are deleted from the set of dichotomies. A dichotomy is invalid if a state appears on both sides of the dichotomy and a dichotomy is redundant if the states on the left and right sides of the dichotomy are a subset of the states on the left and right sides of another larger dichotomy in the three sets. A state encoding that solves the larger dichotomy would also solve the redundant dichotomy. For example, the dichotomy $(S_0; S_1)$ in Table 1 is redundant since any state encoding that satisfies $(S_0S_2; S_1)$ would also solve $(S_0; S_1)$. All of the invalid and redundant dichotomies are shown in boldface in Table 1.

In order to generate a state encoding that satisfies all of the above *initial* dichotomies, we follow the framework of DICHOT [24] to derive an encoding with the minimum number of bits. DICHOT computes the state encoding by finding a set of *prime* encoding-dichotomies[4] with the minimum cardinality that covers all of the initial dichotomies, which is the classical unate covering problem. In order to find a state encoding that provides an inverter-free ABMM implementation, we modify the covering problem such that a prime encoding-dichotomy covers a dichotomy only if the resulting bit assignment solves the dichotomy by using a positive identifier. For example, the state encoding $S_0 = 0011$, $S_1 = 0110$, and $S_2 = 1001$ solves all of the dichotomies in Table 1 by using a positive identifier. The new state encoding is provided to MINIMALIST, which yields the inverter-free ABMM implementation. Finally, the state encoding procedure is repeated for all prime encoding-dichotomy sets with the minimum cardinality that covers the initial dichotomies and the implementation with the lowest area cost is selected. For the example in Fig. 1, the inverter-free ABMM implementation shown in Fig. 12 is generated.

7.3 ABMM Implementation of the Berger Code Generator

Implementing the Berger code generator as an ABMM requires that its symbolic state transition table satisfy the two conditions mentioned in Section 3.1, namely, the maximal set property and the unique entry point. In order to build the symbolic state transition table for the Berger code generator, we start from the symbolic state transition table of the original ABMM. Specifically, for every defined input burst in the state transition table of the ABMM, we substitute the output burst with the corresponding check symbol, that is, the complement of the binary representation of the number of 1s in the output burst. Note that the Berger code generator is defined based on the same input bursts of the original ABMM and, hence, the maximal set property is satisfied. Moreover, transitions between states in the specification of the original ABMM are identical to transitions between states in the specification of the Berger code generator and, hence, the unique entry point requirement is also satisfied. Therefore, there will always exist an ABMM implementation of the Berger code generator which can be synthesized using MINIMALIST. For example, the Berger code generator for the ABMM defined through the symbolic state transition table in Fig. 1 is defined through the symbolic state transition table in Fig. 13, based on which MINIMALIST generates the implementation in Fig. 14.

4. A prime encoding-dichotomy of a given set of dichotomies is one that is incompatible with all encoding dichotomies not covered by it. It essentially provides the value of a single bit in the state encoding of all of the states.

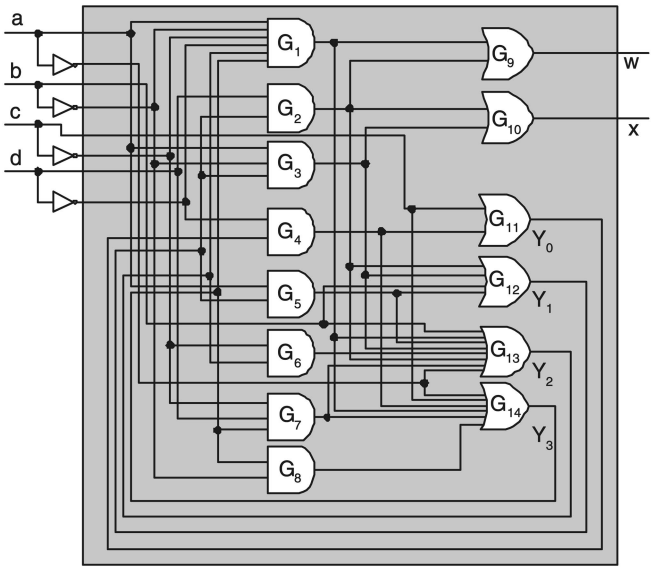

Fig. 12. Inverter-free ABMM implementation.

7.4 Complete Berger Code-Based CED Method

Based on the above discussion, the complete Berger code-based CED method for ABMMs is simple to describe and is illustrated in Fig. 15. First, the inverter-free ABMM and the Berger code generator are added and a Berger code checker [31], [38] is used to indicate whether the output of the ABMM and the output of the Berger code generator form a code word or not. Then, the ABMM implementation of the TPF is added to enable/disable the checker. Finally, the HDC, including the CDC, is added and the same glue-logic gates (G1-G3) used in the Duplication-based and Transition-Triggered CED methods are used here as well in order to generate the error output. Similarly to the Duplication-based CED, the Berger code-based CED indicates an error in the computation in the following scenarios: 1) The output of the original ABMM changes and the TPF indicates no expected change at the output, in which case G_1 indicates an error, 2) the comparator detects a mismatch between the outputs of the original and the duplicate circuit after a new input burst is applied, in which case G_2 indicates an error, or 3) a hazard is detected at the outputs of the original circuit by the HDC. The above error detection scenarios provide complete detection ability for all functional errors and hazards in the circuit.

8 Experimental Results

We applied the proposed CED methods to a suite of 13 benchmark circuits that have been extensively used by the asynchronous design community [16], [18], [37], [39], [40], [41] and include several controllers used in industrial designs. For example, the $rf - control$, $hp - ir$, $p1$, and $p2$

States	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
S_0	S_0,11	-	-	-	-	-	-	-	S_0,10	S_0,11	S_2,11	-	S_1,11	-	-	-
S_1	S_0,11	-	-	-	-	-	-	-	S_1,10	S_1,01	-	-	S_1,11	-	-	-
S_2	-	-	-	-	-	-	-	-	S_2,11	S_0,11	S_2,11	S_2,11	-	-	-	-

Fig. 13. Example of the symbolic state transition table of the Berger code generator.

circuits are part of a low-power infrared controller that was designed at Hewlett-Packard Laboratories as part of the Stetson project [39], the $pe-send-ifc$ circuit was also designed at HP Labs and is part of a high-performance adaptive routing chip used in the Mayfly parallel processing system [40], and the $tangram-mixer$, $concur-mixer$, and $while$ circuits are from the TANGRAM system project at Philips Research Laboratories. The circuits are first synthesized using MINIMALIST to generate an asynchronous implementation in *pla* [42] format. The TPF specification is derived, as discussed in Section 4.1, and is also synthesized as an ABMM using MINIMALIST. The optimized duplicate circuit used in Transition-Triggered CED is produced using *espresso* [42] based on the specification of the original circuit. Finally, the inverter-free implementation of the original ABMM and the Berger code generator are obtained, as described in Section 7.

We first present, in Section 8.1, the percentage of errors that cause only hazards. Then, in Section 8.2, we compare the overhead of the proposed CED methods. Finally, in Section 8.3, we comment on the effectiveness and overhead of the proposed CED methods.

8.1 Error-Induced Hazards

The percentage of errors that cause only hazards in the benchmark circuits is illustrated in Table 2. In small benchmark circuits such as $concur-mixer$ and $opt-token-distributer$, all errors cause an erroneous response at the output since the implementation does not contain any redundant logic and, hence, errors always result in a functional discrepancy. In more complex circuits such as $pe-send-ifc$ and $p1$, more than 30 percent of all possible errors cause only hazards. As the circuit specification becomes more complex, more redundant logic is used to make the circuit hazard-free and, thus, the percentage of errors that cause only hazards also increases.

8.2 Comparison between the CED Methods

The results are analytically presented for the individual components of the three CED methods in Tables 3, 4, and 5. In Table 3, we report the details of the circuits that were used: name, number of inputs (I), number of states (S), number of state bits (Bits), and number of outputs (O). We also report the cost of the original circuit and its duplicate, the TPF, the comparator, the CDC, and the HDC. The last major heading summarizes the total literal and gate count of the Duplication-based CED method. The gate count of the circuits is normalized to the equivalent number of two-input NAND gates. Similar information is reported for the Transition-Triggered CED method in Table 4, with the key differences residing in the cost of the optimized duplicate, the multiplexers, and the slightly more expensive CDC. Finally, in Table 5, we report additional information relevant to the Berger code-based CED method, including the cost of the resynthesized inverter-free implementation of the circuit, the code generator, and the Berger checker.

Several observations can be made based on these results. First, for a few circuits, the cost of the TPF is zero. This is attributed to the simplicity of the specification of these controllers, wherein every input burst is composed of a single input change and, hence, the TPF always indicates a transition. Second, the cost of the optimized duplicate circuit used in Transition-Triggered CED is very close to the cost of the original circuit for small benchmarks. For example, this is the case for circuits $concur-mixer$ and $hp-ir$. In more complex circuits, such as $while\_concur$ and $rf-control$, the cost of the optimized circuit is almost 50 percent of the cost of the original circuit. As the circuit specification becomes more complex, the percentage of redundant logic that can be saved by Transition-Triggered CED also increases. Third, although the number of state bits necessary for inverter-free ABMM is often twice the original number of state bits, which is the worst-case scenario, the inverter-free ABMM implementation increases the area cost

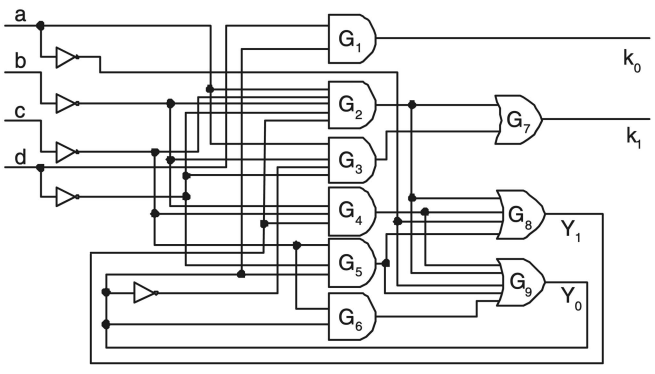

Fig. 14. ABMM implementation of the Berger code generator.

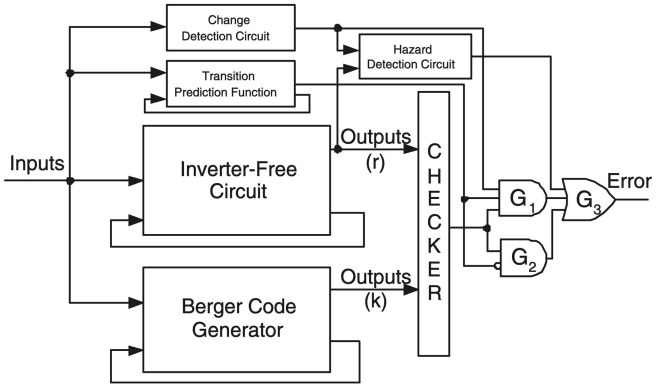

Fig. 15. Berger code-based CED.

TABLE 2
Percentage of Errors That Cause Only Hazards

Circuit Name	% of Errors
concur-mixer	0.0%
martin-q-element	28.0%
opt-token-distributor	0.0%
pe-send-ifc	32.06%
tangram-mixer	15.79%
p1	31.90%
while_concur	23.15%
rf-control	14.76%
hp-ir	5.88%
while	23.61%
p2	26.90%
diffeq	18.45%
barcode	15.22%
Average	18.13%

by an average of only 50 percent over the original ABMM implementation.

The implementation cost of the three CED methods is compared in Table 6. Transition-Triggered CED saves an average of only 3 percent over Duplication-based CED. Sometimes, it is even more expensive. Berger code-based CED saves an average of 15 percent over Duplication-based CED and 11 percent over Transition-Triggered CED. In some cases such as $martin-q-element$, the CED overhead is reduced by 27 percent and 21 percent over Duplication-based CED and Transition-Triggered CED, respectively. With the exception of $pe-send-ifc$, for which the Berger code generator is expensive and Transition-Triggered CED is less expensive, the conjecture of these results is that the Berger code-based method incurs the lowest overhead for performing CED in ABMMs.

Duplication-based CED and Transition-Triggered CED preserve the performance of the ABMM since the original circuit is left intact and the CED circuitry is added and operates in parallel with it. Berger code-based CED, however, requires the use of an inverter-free version of the original ABMM and thus results in a reencoded and resynthesized implementation that may operate at a different speed. For ABMMs, the metric that is typically used to assess performance is the output latency, which can be reasonably approximated by computing the average number of literals per output [41]. The last column in Table 5 compares the performance of the inverter-free ABMM to the performance of the original ABMM. The results indicate that the ABMM in Berger code-based CED is, on average, less than 5 percent slower than the original ABMM.

8.3 Discussion

All three CED methods proposed herein provide complete coverage of all single errors in the monitored circuit, independent of whether the result is a functional discrepancy or only a hazard endangering communication with the environment. With regard to their effectiveness on potential errors in the circuitry added solely for CED purposes, we remind the reader that the objective of CED is to raise the error indication signal whenever the monitored circuit is producing incorrect results. With this in mind and under the single-event assumption, if the error is in the monitored circuit, then the rest of the CED logic will be error free and, therefore, the error will be detected. Otherwise, if the error is anywhere else in the CED logic (including the TPF, output comparator/checker, HDC, and so forth), then the results reported at the outputs of the monitored circuit will be correct. In this case, it is highly likely, yet not guaranteed, that the error will also be

TABLE 3
Experimental Results for Duplication-Based CED

Circuit		Original		Duplicate		TPF		Comparator		CDC		HDC		Total	
Name	I/S(Bits)/O	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates
concur-mixer	3/3(2)/3	26	16	26	16	29	17	28	14	15	8	22	12	154	87
martin-q-element	2/2(1)/2	14	9	14	9	1	0	22	9	10	5	14	8	83	44
opt-token-distributor	4/6(3)/4	74	41	74	41	1	0	38	19	20	11	29	16	244	132
pe-send-ifc	5/5(3)/3	110	58	110	58	134	79	28	14	15	8	22	12	427	233
tangram-mixer	3/2(1)/2	17	10	17	10	1	0	22	9	10	5	14	8	89	46
p1	13/11(4)/14	458	238	458	238	135	76	138	69	70	41	104	59	1371	725
while_concur	4/4(2)/3	41	24	41	24	31	18	28	14	15	8	22	12	186	104
rf-control	6/6(3)/5	75	37	75	37	29	17	48	24	25	14	37	21	297	154
hp-ir	3/2(1)/2	13	8	13	8	33	19	22	9	10	5	14	8	113	61
while	4/3(2)/3	27	16	27	16	1	0	28	14	15	8	22	12	128	70
p2	8/13(4)/16	349	192	349	192	1	0	158	79	80	47	119	67	1066	581
diffeq	14/9(4)/20	345	189	345	189	118	66	198	99	100	59	149	84	1263	690
barcode	13/11(4)/17	327	172	327	172	160	85	168	84	85	50	127	72	1202	639

TABLE 4
Experimental Results for Transition-Triggered CED

Circuit	Original		Opt. Duplicate		Muxes		TPF		Comparator		CDC		HDC		Total	
Name	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates
concur-mixer	26	16	22	14	4	2	29	17	28	14	18	9	22	12	157	88
martin-q-element	14	9	12	7	2	1	1	0	22	9	13	5	14	8	86	43
opt-token-distributor	74	41	46	22	6	3	1	0	38	19	25	14	29	16	227	119
pe-send-ifc	110	58	48	27	6	3	134	79	28	14	28	15	22	12	383	211
tangram-mixer	17	10	13	8	2	1	1	0	22	9	18	8	14	8	95	48
p1	458	238	185	99	8	4	135	76	138	69	80	50	104	59	1116	599
while_concur	41	24	21	12	4	2	31	18	48	24	23	12	22	12	198	108
rf-control	75	37	34	19	6	3	29	17	48	24	35	20	37	21	272	145
hp-ir	13	8	10	6	2	1	33	19	22	9	18	8	14	8	120	63
while	27	16	13	9	4	2	1	0	28	14	23	12	22	12	126	69
p2	349	192	239	126	8	4	1	0	158	79	57	38	119	67	939	510
diffeq	345	189	282	154	8	4	118	66	198	99	91	60	149	84	1199	660
barcode	327	172	269	137	8	4	160	85	168	84	82	53	127	72	1149	611

TABLE 5
Experimental Results for Berger Code-Based CED

Circuit		Inverter-Free		Code Generator			TPF		Checker [31]		CDC		HDC		Total		Performance
Name	I/S(Bits)/O	Lit.	Gates	r	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates	Lit.	Gates	Overhead
concur-mixer	3/3(3)/3	52	29	1	22	13	29	17	4	2	15	8	22	12	152	85	7.69%
martin-q-element	2/2(2)/2	20	11	1	3	1	1	0	3	2	10	5	14	8	59	31	5.56%
opt-token-distributor	4/6(5)/4	106	57	2	32	18	1	0	5	3	20	11	29	16	201	109	7.81%
pe-send-ifc	5/5(5)/3	153	79	2	128	67	134	79	5	3	15	8	22	12	465	252	10.82%
tangram-mixer	3/2(2)/2	24	15	2	17	10	1	0	4	2	10	5	14	8	78	44	5.88%
p1	13/11(8)/14	615	320	3	200	108	135	76	11	5	70	41	104	59	1143	613	1.76%
while_concur	4/4(4)/3	59	33	1	19	11	31	18	4	2	15	8	22	12	158	88	2.08%
rf-control	6/6(6)/5	86	48	1	24	14	29	17	5	3	25	14	37	21	224	121	4.23%
hp-ir	3/2(2)/2	23	14	1	11	5	33	19	3	2	10	5	14	8	102	57	3.85%
while	4/3(3)/3	49	28	1	5	2	1	0	4	2	15	8	22	12	104	56	3.92%
p2	8/13(8)/16	458	238	3	140	74	1	0	12	6	80	47	119	67	818	436	1.84%
diffeq	14/9(8)/20	419	224	3	211	112	118	66	14	7	100	59	149	84	1019	556	0.83%
barcode	13/11(8)/17	449	232	3	181	98	160	85	12	6	85	50	127	72	1022	547	1.74%

TABLE 6
Comparison between the Proposed CED Methods (Percentile Cost Reduction)

Circuit Name	Transition-Triggered vs. Duplication	Berger code-based vs. Duplication	Berger code-based vs. Transition-Triggered
concur-mixer	−1.95%	1.30%	3.19%
martin-q-element	−3.62%	28.92%	31.40%
opt-token-distributor	6.97%	17.62%	11.45%
pe-send-ifc	10.31%	−8.90%	−21.41%
tangram-mixer	−6.74%	12.36%	17.90%
p1	18.60%	16.63%	−2.42%
while_concur	−6.45%	15.05%	20.20%
rf-control	8.42%	27.95%	21.32%
hp-ir	−6.20%	9.74%	15.00%
while	1.56%	18.75%	17.46%
p2	11.91%	23.27%	12.89%
diffeq	5.07%	19.32%	15.01%
barcode	4.41%	14.98%	11.05%
Average Reduction	3.25%	15.15%	11.77%

detected. Should one be interested in guaranteeing its detection, methods akin to the ones employed by the analogous synchronous CED methods (that is, duplication or dual-rail encoding of the cone of logic driving the error signal and addition of a second error indication pin) may be employed. Although some may consider such detection a false positive, it is common practice in CED to err on the side of caution, that is, to report the error even if it is not on the monitored circuit but on the additional circuit used for CED. We stress, however, that the most important property is that the proposed CED methods never produce false negatives, that is, they never fail to report an error on the monitored circuit when one exists.

With regard to the area overhead incurred by the proposed CED methods, at first glance, it appears to be significantly higher than that of synchronous CED methods [1]. However, we would like to be cautious in resorting to such a quantitative comparison since it is rather misleading due to two reasons. First, the overhead is significantly inflated for small circuits due to the cost of the proportionately large HDCs that are attached to the output pins. To demonstrate this point, in Fig. 16, we plot the area overhead of the proposed CED methods versus the original area cost of the benchmark circuits. The curves indicate that the percentile area overhead of the proposed CED methods reduces as the complexity of the benchmark circuits increases. Thus, we anticipate the area overhead will be even lower for larger and more complex ABMMs, eventually becoming comparable to the overhead of CED methods for synchronous circuits. Second, and more importantly, the proposed CED methods offer more robustness than their synchronous counterparts. Specifically, they examine not only the functional correctness of the results but also the correctness of the communication between the circuit and its environment. Synchronization of such communication is distributed, as opposed to a centralized clock used in synchronous circuits, the potential errors on which may not be detected by synchronous CED methods. Finally, although instantiating synchronous CED methods may be easier than their asynchronous counterparts, this is most likely an artifact of their narrower scope, as well as the fact that the former are the outcome of several decades of research, whereas the latter are merely taking their first serious steps.

9 Conclusion

The lack of a global clock and the existence of redundant logic to ensure hazard-free operation pose new challenges in performing CED in ABMMs. As demonstrated in this

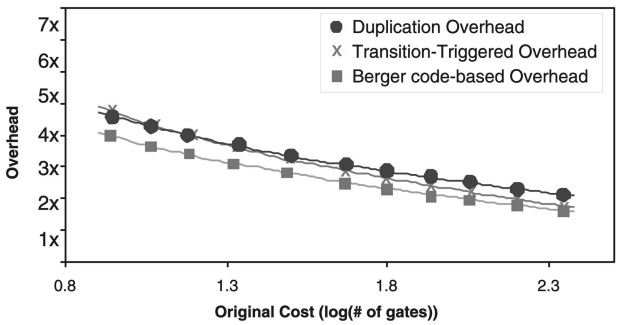

Fig. 16. CED overhead versus area cost of the ABMM.

work, the checking synchronization problem arising from the lack of a global clock can be resolved through additional hardware which exploits information inherent in the operation of an ABMM. Similarly, hazard detection hardware can be used to detect errors in the redundant logic which cause only hazards but no functional discrepancy in the operation of an ABMM. Thus, both the functional correctness of the circuit and its correct interaction with the environment can be monitored. The proposed comparison synchronization and hazard detection methods enable the adaptation of CED techniques from the synchronous domain and ensure their effectiveness in ABMMs. Three such methods, namely, Duplication-based CED, Transition-Triggered CED, and Berger code-based CED, were developed and discussed herein. Among them, at the cost of a minor impact on performance, Berger code-based CED incurs the lowest area overhead, as indicated by experimental results.

ACKNOWLEDGMENTS

The authors would like to thank their colleague Feng Shi from Yale University for helpful discussions on the operation of burst-mode circuits and Professor Steven M. Nowick from Columbia University for providing MINIMALIST, the burst-mode logic synthesis package used in this work.

REFERENCES

[1] S. Mitra and E.J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?" *Proc. Int'l Test Conf.,* pp. 985-994, 2000.

[2] S. Almukhaizim, P. Drineas, and Y. Makris, "Entropy-Driven Parity-Tree Selection for Low-Overhead Concurrent Error Detection in Finite State Machines," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 25, no. 8, pp. 1547-1554, 2006.

[3] M. Goessel and S. Graf, *Error Detection Circuits.* McGraw-Hill, 1993.

[4] S. Almukhaizim and Y. Makris, "Fault-Tolerant Design of Combinational and Sequential Logic Based on a Parity Check Code," *Proc. 18th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '03),* pp. 563-570, 2003.

[5] T. Verdel and Y. Makris, "Duplication-Based Concurrent Error Detection in Asynchronous Circuits: Shortcomings and Remedies," *Proc. 17th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '02),* pp. 345-353, 2002.

[6] S.J. Piestrak and T. Nanya, "Towards Totally Self-Checking Delay-Insensitive Systems," *Proc. 25th Int'l Symp. Fault-Tolerant Computing,* pp. 228-237, 1995.

[7] V. Akella, N.H. Vaidya, and G.R. Redinbo, "Limitations of VLSI Implementation of Delay-Insensitive Codes," *Proc. 26th Int'l Symp. Fault-Tolerant Computing,* pp. 208-217, 1996.

[8] E. Grass, V. Bartlett, and I. Kale, "Completion-Detection Techniques for Asynchronous Circuits," *IEICE Trans. Information and Systems,* vol. E80-D, no. 3, pp. 344-350, 1997.

[9] S.J. Piestrak, "Membership Test Logic for Delay-Insensitive Codes," *Proc. Int'l Symp. Advanced Research in Asynchronous Circuits and Systems,* pp. 194-205, 1998.

[10] V.I. Varshavsky, *Self-Timed Control of Concurrent Processes.* Kluwer Academic, 1990.

[11] N. Kanopoulos, D. Pantzartzis, and F.R. Bartram, "Design of Self-Checking Circuits Using DCVS Logic: A Case Study," *IEEE Trans. Computers,* vol. 41, no. 7, pp. 891-896, July 1992.

[12] D.A. Rennels and H. Kim, "Concurrent Error Detection in Self-Timed VLSI," *Proc. 24th Int'l Symp. Fault-Tolerant Computing,* pp. 96-105, 1994.

[13] B.R. Kishore and T. Nanya, "On Concurrent Error Detection of Asynchronous Circuits Using Mixed-Signal Approach," *IEICE Trans. Information and Systems,* vol. E80-D, no. 3, pp. 351-362, 1997.

[14] D. Shang, A. Bystrov, A. Yakovlev, and D. Koppad, "On-Line Testing of Globally Asynchronous Circuits," *Proc. 11th IEEE Int'l On-Line Testing Symp. (IOLTS '05),* pp. 135-140, 2005.

[15] D. Shang, A. Yakovlev, F. Burns, F. Xia, and A. Bystrov, "Low-Cost Online Testing of Asynchronous Handshakes," *Proc. European Test Symp.,* pp. 225-232, 2006.

[16] R.M. Fuhrer and S.M. Nowick, *Sequential Optimization of Asynchronous and Synchronous Finite-State Machines: Algorithms and Tools.* Kluwer Academic, 2001.

[17] S.M. Nowick, "Automatic Synthesis of Burst-Mode Asynchronous Controllers," PhD dissertation, Stanford Univ., 1993.

[18] S.M. Nowick and D.L. Dill, "Exact Two-Level Minimization of Hazard-Free Logic with Multiple-Input Changes," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 15, no. 8, pp. 986-997, 1995.

[19] R.M. Fuhrer and S.M. Nowick, "OPTIMISTA: State Minimization of Asynchronous FSMs for Optimum Logic," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '99),* pp. 7-13, 1999.

[20] W.-C. Chou, P.A. Beerel, R. Ginosar, R. Kol, C.J. Myers, S. Rotem, K. Stevens, and K.Y. Yun, "Average-Case Optimized Technology Mapping of One-Hot Domino Circuits," *Proc. Int'l Symp. Advanced Research in Asynchronous Circuits and Systems,* pp. 80-91, 1998.

[21] A. Marshall, B. Coates, and P. Siegel, "Designing an Asynchronous Communications Chip," *IEEE Design and Test of Computers,* vol. 11, no. 2, pp. 8-21, 1994.

[22] D.A. Huffman, *The Synthesis of Sequential Switching Networks.* Addison-Wesley, 1964.

[23] S.H. Unger, *Asynchronous Sequential Switching Circuits.* Wiley-Interscience, 1969.

[24] A. Saldanha, T. Villa, R.K. Brayton, and A. Sangiovanni-Vincentelli, "A Framework for Satisfying Input and Output Encoding Constraints," *Proc. 28th Design Automation Conf.,* pp. 170-175, 1991.

[25] E.J. McCluskey, "Minimization of Boolean Functions," *Bell System Technology J.,* vol. 35, pp. 1417-1444, 1956.

[26] K. De, C. Natarajan, D. Nair, and P. Banerjee, "RSYN: A System for Automated Synthesis of Reliable Multilevel Circuits," *IEEE Trans. VLSI,* vol. 2, pp. 186-195, 1994.

[27] J.M. Berger, "A Note on Error Detection Codes for Asymmetric Channels," *Information and Control,* vol. 4, pp. 68-73, 1961.

[28] N.K. Jha and S.-J. Wang, "Design and Synthesis of VLSI Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 12, no. 6, pp. 878-887, 1993.

[29] V.V. Kumar and J. Lach, "Heterogeneous Redundancy for Fault and Defect Tolerance with Complexity Independent Area Overhead," *Proc. 18th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '03),* pp. 571-578, 2003.

[30] C. Bolchini, F. Salice, and D. Sciuto, "A Novel Methodology for Designing TSC Networks Based on the Parity Bit Code," *Proc. European Design and Test Conf.,* pp. 440-444, 1997.

[31] X. Kavousianos and D. Nikolos, "Novel Single and Double Output TSC Berger Code Checkers," *Proc. VLSI Test Symp.,* pp. 348-353, 1998.

[32] S.W. Burns and N.K. Jha, "A Totally Self-Checking Checker for a Parallel Unordered Coding Scheme," *IEEE Trans. Computers,* vol. 43, no. 4, pp. 490-495, Apr. 1994.

[33] E.J. McCluskey, "Design Techniques for Testable Embedded Error Checkers," *Computer,* vol. 23, no. 7, pp. 84-88, July 1990.

[34] O. Yenersoy, "Synthesis of Asynchronous Machines Using Mixed-Operation Mode," *IEEE Trans. Computers,* vol. 26, no. 8, pp. 325-329, Aug. 1979.

[35] J.S. Chiang and D. Radhakrishnan, "Hazard-Free Design of Mixed Operating Mode Asynchronous Sequential Circuit," *Int'l J. Electronics,* vol. 68, no. 1, pp. 23-37, 1990.

[36] S.M. Nowick and D.L. Dill, "Synthesis of Asynchronous State Machines Using a Local Clock," *Proc. IEEE Int'l Conf. Computer Design (ICCD '91),* pp. 192-197, 1991.

[37] S.M. Nowick and D.L. Dill, "Automatic Synthesis of Locally Clocked Asynchronous State Machines," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '95),* pp. 318-321, 1995.

[38] S.J. Piestrak, "Design of Fast Self-Testing Checkers for a Class of Berger Codes," *IEEE Trans. Computers,* vol. 36, pp. 629-634, 1987.

[39] A. Marshall, B. Coates, and P. Siegel, "The Design of an Asynchronous Communications Chip," *IEEE Design and Test of Computers,* pp. 8-21, 1994.

[40] K.S. Stevens, S.V. Robison, and A.L. Davis, "The Post Office Communication Support for Distributed Ensemble Architectures," *Proc. Sixth Int'l Conf. Distributed Computing Systems (ICDCS '86)*, pp. 160-166, 1986.
[41] R.M. Fuhrer, S.M. Nowick, M. Theobald, N.K. Jha, B. Lin, and L. Plana, "MINIMALIST: An Environment for the Synthesis, Verification and Testability of Burst-Mode Asynchronous Machines," Technical Report TR CUCS-020-99, Dept. of Computer Science, Columbia Univ., 1999.
[42] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldahna, H. Savoj, P.R. Stephan, R.K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," ERL memo UCB/ERL M92/41, Dept. of Electrical Eng. and Computer Science, Univ. of California, Berkeley, 1992.

Sobeeh Almukhaizim received the diploma in electrical and computer engineering from Kuwait University, Kuwait, in 1999, the MS degree in computer science and engineering from the University of California, San Diego, in 2001, and the MS and MPhil degrees in electrical engineering from Yale University in 2003. He is currently a PhD candidate in the Department of Electrical Engineering at Yale University. His current research interests include VLSI design and test, fault tolerance, and concurrent error detection. He is a student member of the IEEE and the IEEE Computer Society.

Yiorgos Makris received the diploma in computer engineering and informatics from the University of Patras, Greece, in 1995 and the MS and PhD degrees in computer science and engineering from the University of California, San Diego, in 1997 and 2001, respectively. He is currently an associate professor of electrical engineering and computer science at Yale University, where he leads the Testable and Reliable Architectures (TRELA) Research Group. His current research interests include soft-error mitigation in digital circuits, machine-learning-based testing of analog/radio frequency (RF) circuits, and test and reliability of asynchronous circuits. He is a member of the IEEE and the IEEE Computer Society.