Duplication-Based Concurrent Error Detection in Asynchronous Circuits: Shortcomings and Remedies

Thomas Verdel & Yiorgos Makris
Electrical Engineering Department
Yale University
{thomas.verdel, yiorgos.makris}@yale.edu

Abstract

Concurrent error detection (CED) methods are typically employed to provide an indication of the operational health of synchronous circuits during normal functionality. Existing CED techniques, however, require modification in order to be successfully adapted to asynchronous designs. We discuss the limitations of duplication, the simplest CED method, when applied to asynchronous circuits. We demonstrate that such limitations arise mainly due to comparison synchronization issues and inadequate detection of performance-related errors. We propose a circuit that alleviates the difficulties associated with comparison synchronization and we introduce a methodology that enables detection of errors that do not result in logic discrepancies and, thus, may not be detected through comparison. The proposed techniques are illustrated on example circuits, revealing their ability to render concurrently testable asynchronous designs.

1. Introduction

Advantages such as reduced power dissipation, elimination of clock distribution issues, modularity, and improved performance have enabled asynchronous circuits to carve a widening niche in many systems previously dominated by synchronous circuits [16]. Nevertheless, widespread acceptance of asynchronous designs requires development of elaborate methods and advanced EDA solutions. Unlike their synchronous counterparts, which have enjoyed a high level of design automation since the mid-1970s, similar efforts for asynchronous circuits have not kept up to par. In conjunction with the inherently more difficult asynchronous style [2,8], the lack of CAD support has deterred efforts not only in design but also in all other aspects of asynchronous circuit realization, including test. However, the recent resurgence of asynchronicity as a solution to several limitations encountered in submicron technology [16] has sparked new research efforts in test of asynchronous circuits [3,4,5,13].

Among the several test challenges, in this work we focus on concurrent error detection (CED), a problem that has been extensively studied for synchronous circuits [7,9,11,12]. The objective of CED is to provide a circuit with the ability to monitor itself and report potential deviations from its correct functionality. In systems where data integrity is vital, CED is an absolute necessity. While porting some of the concepts of synchronous CED to asynchronous designs has been successfully attempted in the past [10,14], the applicability of these methods is limited because the underlying design style has proved impractical for large circuits [4]. In contrast, we examine the portability of duplication [15], a simple, yet generally applicable CED method. In duplication, a replica of the circuit is added to the design, possibly diversely implemented to avoid common mode failures [1]. The original and the replica serve as predictors of the functionality of each other and a simple comparator indicates any discrepancy

Proceedings of the 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'02)
1063-6722/02 $17.00 © 2002 **IEEE**

in their outputs, thus detecting potential malfunctions. Although expensive, this technique has proved practical due to its simplicity, effectiveness, and marginal impact on performance.

The key difficulty in applying duplication-based CED to asynchronous circuits is the lack of a global synchronization mechanism, the clock. Without it, it is unclear when the outputs of the original and the replica circuit are expected to match, thus allowing the possibility of false alarms. As we discuss in this paper, this obstacle can be overcome by a modified comparator exploiting local synchronization that exists in asynchronous circuits despite the lack of a global clock. Further analysis, however, shows that synchronized comparison is still unable to detect all fault effects. Due to the asynchronicity in the operation of the original and the replica, a time window within which both arrive at the same state is required. This time window can result in masking of some performance-related fault effects so that the circuit appears to be operating normally when it actually is not. Such faults can still be detected, however, using signals internal to the circuit. Through analysis of the states of the fault-free and the faulty circuits, we determine a set of states that uniquely detect these malfunctions and implement a hardware monitor for them. By combining this information across all faults and by also exploiting states that are not within the set of states for the fault-free circuit, the cost of the monitor function is significantly reduced.

Basics of asynchronous circuits, advantages and disadvantages over their synchronous counterparts, as well as a model for their failure mechanisms are reviewed in Section 2. Section 3 summarizes the typical use of duplication-based CED on synchronous circuits and reveals its limitations on asynchronous circuits. A simple circuit that provides the ability to adequately synchronize comparisons is proposed and illustrated through an example in Section 4. Section 5 presents a methodology for detecting performance-related faults that are not detected through comparison. Example circuits are used to demonstrate the ability of the proposed methods to render a duplication-based, concurrently testable, asynchronous circuit.

2. Review of asynchronous circuits

Often discussed in the literature [2,8,16], advantages of asynchronous circuits include better performance, less area for clock circuitry, reduced power consumption, and better modularity. In synchronous circuits, the clock period is chosen according to a worst-case timing analysis. Regardless of how quickly the computation is actually performed, the circuit is allotted the same worst-case time. Furthermore, since chip performance may vary depending on process variations, voltage, and temperature, additional time, called the clock margin [2], is added to the clock period to compensate for these variables. In contrast, the worst-case scenario does not limit the performance of asynchronous circuits. Each asynchronous element communicates through handshake protocols when it is ready to begin computation and when it is finished computing. Therefore, it takes only the time required to do the particular computation; over several computations it exhibits average case performance [8,16]. In synchronous circuits, clock distribution is also a rapidly growing problem. With increasing chip size, complexity, and speed, keeping clock skew to a minimum and efficiently routing clock lines has proved to be an ongoing engineering challenge that requires a growing proportion of chip-area. This is not a problem in asynchronous circuits since they have no global clock. In terms of power consumption, asynchronous circuits are also beneficial because only the elements necessary to perform the computation are active; idle elements do not consume power. Synchronous circuits, on the other hand, consume power every clock cycle, regardless if useful work is actually being done or not. Finally, the implicit signaling protocols of synchronous circuits can make circuit composition a difficult task. Asynchronous circuits are more modular in that one needs only to match the explicit signaling protocol and the interface to compose a larger circuit from two smaller ones [2].

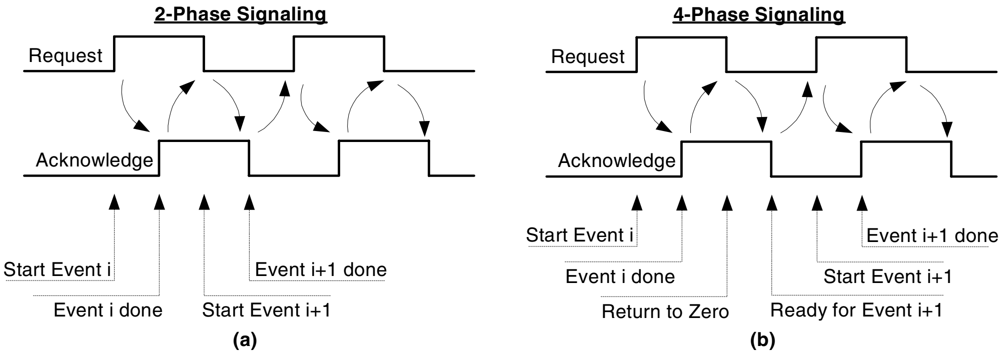


Figure (1): Handshake protocols

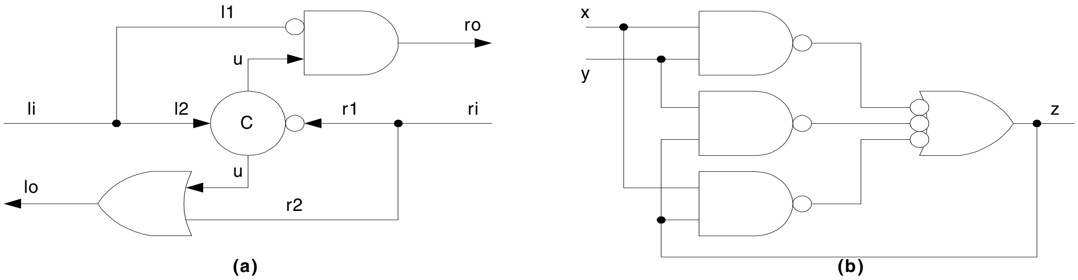


Figure (2): (a) D-Element (b) Possible C-Element implementation

Asynchronous circuits do have drawbacks however. Unlike synchronous circuits that are relatively glitch tolerant, glitches pose a significant problem in asynchronous circuits. Proper, hazard-free operation often necessitates addition of redundant circuitry, which may require more silicon area. Redundant logic poses a test obstacle as well. Lastly, while most designers are familiar with synchronous design techniques, asynchronous circuits can seem relatively cumbersome and difficult to understand. Compounding this difficulty, EDA tools available for asynchronous design are neither as abundant nor as refined as those for synchronous.

From a high-level perspective, each asynchronous module can be thought of as having two parts, a data section and a control section [6]. The data section actually performs the required logical computation. The control section is the synchronization element that enables inter-module communication. Since numerous asynchronous modules may be interacting with each other in a typical circuit, a synchronization standard is required to ensure properly timed communication. This is the purpose of handshake protocols. The two standard signaling protocols used in asynchronous circuits, namely two-phase signaling and four-phase signaling [2], are illustrated in figure (1). The request line, which is controlled by the external environment, is used to inform the circuit that it has a task that it needs the circuit to perform. The acknowledge line is controlled by the circuit itself and indicates its processing state. In the 2-phase scheme, only one transition (either up or down) on the request line and one on the acknowledge line is needed to complete a full handshake. The 4-phase protocol uses only up transitions but requires that both lines return to zero before a new handshake is initiated.

We now include two examples of asynchronous circuits to illustrate how these circuits work and to demonstrate the effects of potential faults. A D-Element [4] is an asynchronous circuit that is used to synchronize two four-phase handshakes. A gate level implementation of this circuit is shown in figure (2)(a). Within the D-Element is another common circuit, the C-Element. An implementation of a C-Element is shown in figure (2)(b). It is a simple state holding device, wherein the output, *z*, follows the inputs, *x* and *y*, when the inputs are the same.

In the standard high-level description notation [3,4], the D-Element is specified internally (i.e. from the circuit's perspective) as:

$$*[[li];u\uparrow;[u];lo\uparrow;[\neg li];ro\uparrow;[ri];u\downarrow;[\neg u];ro\downarrow;[\neg ri];lo\downarrow]$$

In this notation, [α] means wait for an event α (either an assertion, denoted as α, or a deassertion, denoted as $\neg\alpha$). These are events caused by the environment outside of the circuit itself. Transitions initiated by the circuit on a signal β are denoted by either $\beta\uparrow$ (up) or $\beta\downarrow$ (down). A semicolon is used to indicate a sequence of events. For example, [α];$\beta\uparrow$ means wait for signal α to be asserted and then raise signal β. The * character implies that the process is repeated indefinitely and $\|$ denotes that two processes can occur in parallel.

The one-bit, asynchronous full adder found in [13] provides us with another example. Unlike the D-Element, the adder has both data and control portions, and, more importantly, the control section is dependent upon both control and data signals. Figure 3 shows the gate level implementation. A brief discussion of the adder's operation is given here, while a more detailed analysis can be found in [13]. First, the environment determines when the data is ready on inputs *A* and *B*. When *A* and *B* are valid, *nStart*, an active-low signal, is deasserted. If $A = B$, the XOR gate output is zero, and the output of NAND gate G1 is asserted. This then deasserts the output of NAND gate G2, *nCVout*. The timing specification is:

$$[[\neg nStart];\ Start\uparrow;\ nCVout\downarrow;[nStart];\ Start\downarrow;\ nCVout\uparrow]$$

When $A \neq B$, output *nCVout* follows *nCVin*. The timing specification for this case is:

$$[[[[\neg nCVin];\ z\uparrow] \,\|\, [[\neg nStart];\ Start\uparrow]];\ nCVout\downarrow;\ [[[nCVin];\ z\downarrow;\ nCVout\uparrow] \,\|\, [[nStart];\ Start\downarrow;\ nCVout\uparrow]]]$$

The motivation for this design is to speed up carry-propagation in a series of one-bit adders. From a high-level perspective, we see that the adder gets its performance boost from predicting the carry early. When $A=B$, the adder does not need to wait for the carry-in, thus validating its own carry-out signal sooner. This gives the next adder in the series the ability to begin computation earlier and, hence, speed up carry propagation and increase performance.

Failure mechanisms in asynchronous circuits can be classified into three categories, halting faults, premature firings, and faults in redundant logic that cause performance degradation. A halt occurs whenever the circuit can no longer continue through its specification. For example,

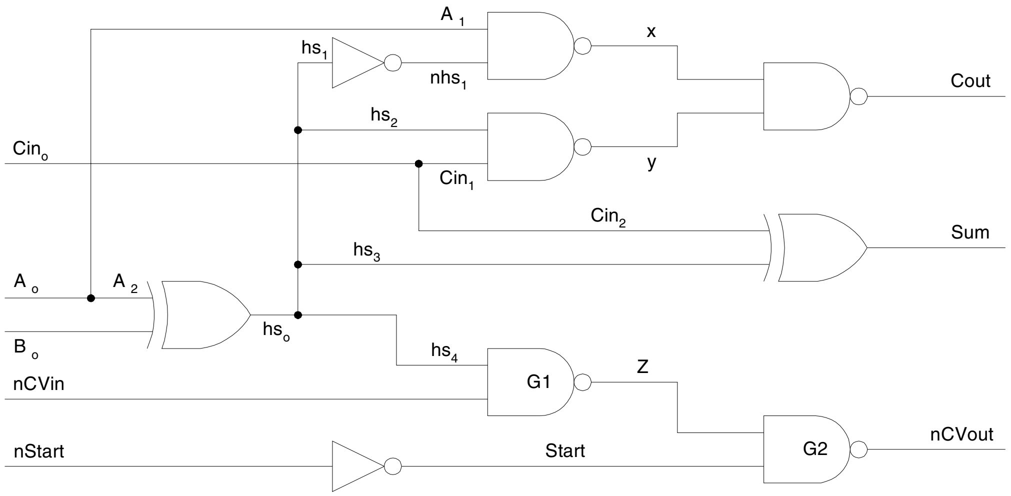


Figure (3): Asynchronous full adder

from the specification for the D-Element, *lo*-s-a-0 would prevent the transition $lo\uparrow$. The environment will wait forever for this transition to occur, halting operation. This cessation of activity makes halts easily detectable [4]. A circuit with a premature firing fault will, under certain conditions, operate faster than the fault-free circuit. The danger of this fault is that inputs or outputs might be falsely indicated as valid. The D-Element has two premature firing faults, *l1*-s-a-0 and *r2*-s-a-0. For *l1*-s-a-0 we see that $[lo\uparrow;[\neg li]]$ and $[ro\uparrow;[ri]]$ may occur simultaneously instead of sequentially as they should. Lastly, faults in redundant logic reduce the speed of the circuit, though the outputs may still be logically correct. For example, when a fault inhibits the carry prediction of the adder, its performance is degraded. One such fault is *hs4*-s-a-0. In this case, if $A=B$, then *nCVout* should transition to zero immediately after *nStart* is deasserted, regardless of the value of *nCVin*. Instead, the circuit is forced to wait for *nCVin* to be deasserted before deasserting *nCVout*. We note, however, that the data output of this circuit will still be correct.

3. Limitations to duplication based concurrent testing

Duplication [15] is one of the simplest non-intrusive CED methods, incurring minimal performance degradation, yet with considerable area overhead. Referring to figure (4)(a), we see that the main idea is to replicate the circuit and connect both sets of outputs to a comparator. An error is indicated if the comparator detects a mismatch at the end of a clock cycle. The clock period guarantees that both the original and the duplicate have ample time to complete the operation required and stabilize to the correct state.

Unlike synchronous circuits, wherein the clock indicates the appropriate comparison time, in asynchronous circuits, we are dependent on local handshakes. Due to process variations, possible diverse implementations to avoid masking in common mode failures [1], and the very fact that the circuit and its duplicate are separate asynchronous entities, we cannot rely on the outputs being exactly the same all of the time. We define the maximum allowable delay between the original and its duplicate due to these factors to be τ_D. In figures (4)(b) and (4)(c) we see two hypothetical timing diagrams for one output of some circuit, O, and its duplicate, D. In the former case we have D lagging O by τ_D while in the latter we have O lagging D by τ_D. The magnitude of τ_D is exaggerated for illustrative purposes, but there are instances where, even though O and D may both be fault-free, the outputs are different.

To overcome the synchronization problem, we need to be able to create a time window of size τ_D so that when a transition occurs on O (D), we indicate an error only when D (O) doesn't make the same transition within the window. Unfortunately, this also enables timing scenarios that prohibit certain types of fault effects from being detected. For any transition, let the instant at which the fault-free circuit makes the transition be t_C. In the case of either a premature firing fault or a fault due to redundant logic, let the instant at which the faulty transition takes place be t_E. It can be shown that if $|t_C\text{-}t_E| < 2\tau_D$ then the comparison process may fail to detect the fault.

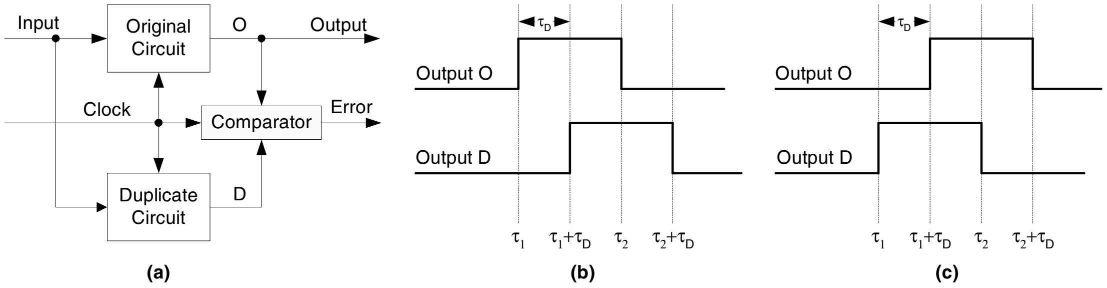
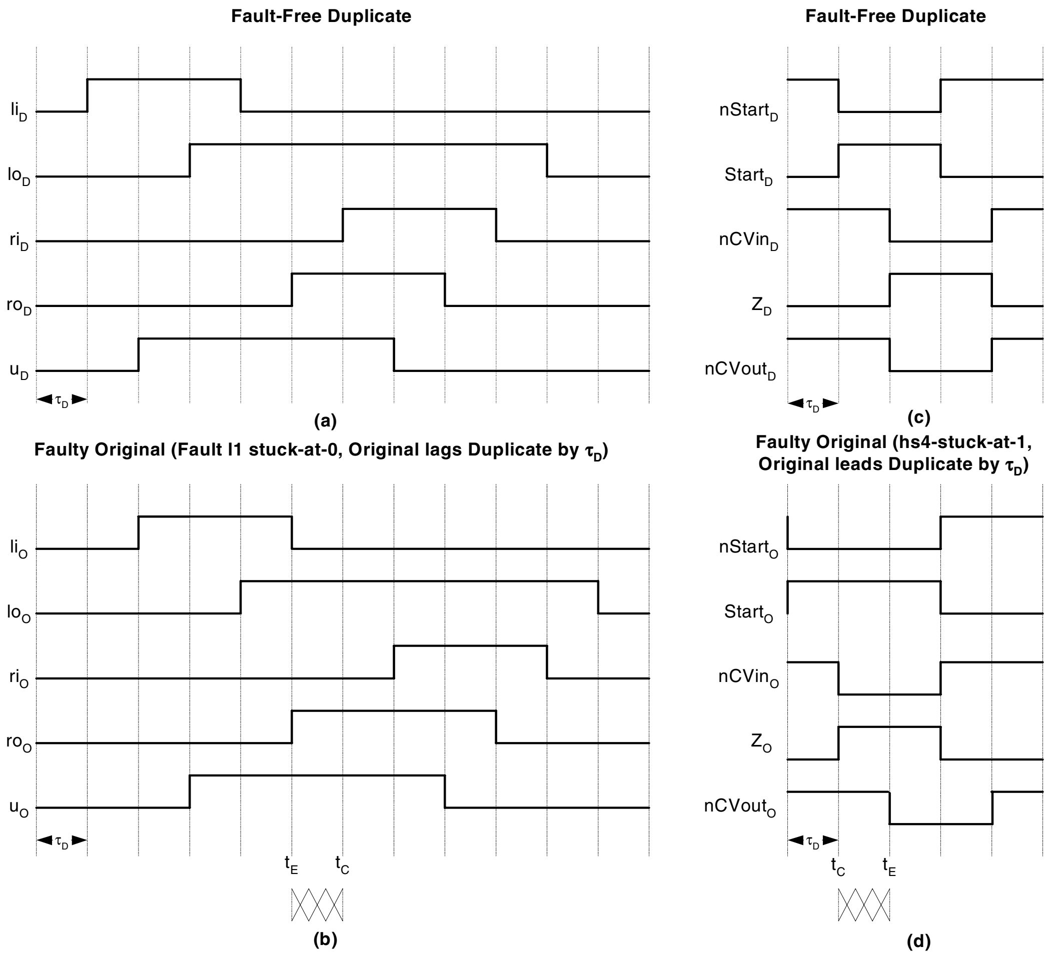


Figure (4): Duplication-based CED and timing diagrams of false negatives



Fault-Free Duplicate

(a)

Faulty Original (Fault l1 stuck-at-0, Original lags Duplicate by τ_D)

(b)

Fault-Free Duplicate

(c)

Faulty Original (hs4-stuck-at-1, Original leads Duplicate by τ_D)

(d)

Figure (5): Timing diagrams of false positive in synchronized comparisons

Figure (5) illustrates the failure of synchronized comparison on the D-Element with a premature firing fault, and on the adder with a fault due to redundant logic. For the D-Element let us assume that the original, O_D, has the fault *l1*-s-a-0, while the duplicate, D_D, is fault-free. In addition, let us assume that O_D lags D_D by τ_D. Since O_D is the controlling circuit, the discrepancy is indicated only by signals ro_D and ro_O. Figure (5)(a) is the timing diagram for D_D and figure (5)(b) is the diagram for O_D. As shown, if O_D is fault-free then $ro_O\uparrow$ will occur at time t_C instead of time t_E. In general, we must allow for the case when D_D lags O_D, so t_E marks the beginning of the comparison window. On circuit D_D, $ro_D\uparrow$ occurs within the window so the synchronized comparator does not detect a discrepancy. The previous formula suggests the same conclusion since $|t_C - t_E| < 2\tau_D$. Thus, the premature firing fault remains undetected.

Repeating the analysis for the adder but assuming that O_A leads D_A, we see that faults due to redundant logic can go undetected. For example, assume that fault *hs4*-s-a-1, which is excited when inputs *A* and *B* are equal, is present in O_A, while D_A is fault-free. Figures (5)(c) and (5)(d) provide the timing diagrams for D_A and O_A, respectively. *nCVout* reflects the fault effect in this situation. In the fault-free scenario, $nCVout\downarrow$ occurs after $nStart\downarrow$. When *hs4*-s-a-1 is present, $nCVout\downarrow$ cannot occur until after both $nStart\downarrow$ and $nCVin\downarrow$. In figure (5)(d), t_C indicates when the $nCVout_O\downarrow$ event should happen and t_E indicates when it actually happens. However, since O_A leads D_A, $nCVout_D\downarrow$ and $nCVout_O\downarrow$ occur within the τ_D window. The resulting performance degradation is masked by the allowed delay.

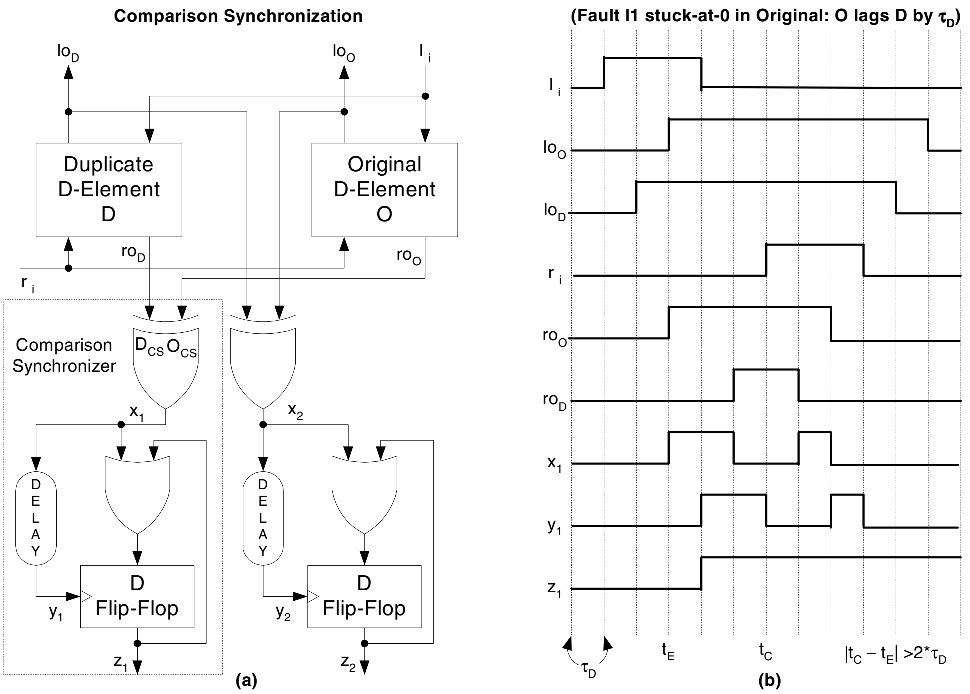


Figure (6): Comparison synchronization circuit and timing diagram

4. Remedial action for synchronization

Limitations of duplication in asynchronous circuits are attributed to the lack of a mechanism to synchronize the comparison. Nevertheless, we can use signals that establish local synchronization within the original and the duplicate to indicate when the comparison results are valid. The comparison synchronizer in figure (6)(a) takes this approach.

If the circuit is duplicated, then all fault types manifest themselves as a discrepancy in the proper timing of the circuit. Therefore, we use a signal from the control portion of each circuit, the original (O_{CS}) and the duplicate (D_{CS}). These signals reflect where each circuit is in the execution of the handshake protocol. As noted earlier, continuous comparison of these signals is insufficient because of the possible delay, τ_D, between them. However, since the magnitude of τ_D can be calculated beforehand, we can use its value to create a time window after a transition on O_{CS} (D_{CS}). If D_{CS} (O_{CS}) also transitions within this window, the circuit is operating correctly. If D_{CS} (O_{CS}) transitions outside of this window, we know that one of the circuits is either operating too fast (a premature firing) or too slow (a fault in redundant logic or a halt).

This is achieved by the comparison synchronizer in figure (6)(a), where the delay attached to the clock-input of the flip-flop is matched to τ_D. Assuming O_{CS} and D_{CS} are the same at the outset, the output of the XOR will be asserted when O_{CS} (D_{CS}) transitions. The output of the XOR is then delayed to the clock of the flip-flop. If the circuit is fault-free, D_{CS} (O_{CS}) will make the same transition, thereby deasserting the output of the XOR before the flip-flop is clocked. If the circuit has a fault that affects the timing of the given signal, the delayed clock pulse will arrive while the output of the XOR is still asserted, setting the flip-flop. Figure (6)(a) shows the two comparison synchronizers necessary to detect all fault effects in the duplex D-Element. The corresponding timing diagram in figure (6)(b) demonstrates detection of the fault *l1*-s-a-0 on the original D-Element. Here the O_{CS} input is ro_O and the D_{CS} input is ro_D. Output z_1 indicates the presence of a fault. While many synchronized comparisons may be necessary, only one output pin driven by the logic OR of their outputs suffices.

l_i	l_o	r_i	r_o	u
0	0	0	0	0
1	0	0	0	0
1	0	0	0	1
1	1	0	0	1
0	1	0	0	1
0	1	0	1	1
0	1	1	1	1
0	1	1	1	0
0	1	1	0	0
0	1	0	0	0

Table 1: Valid set of D-Element

A						**B**						**C**					
l_i	l_o	r_i	r_o	*u*	*X?*	l_i	l_o	r_i	r_o	*u*	*X?*	l_i	l_o	r_i	r_o	*u*	*X?*
1	1	0	0	1	N	1	1	0	0	1	N	1	1	0	0	1	N
0	1	0	0	1	N	1	1	0	1	1	Y	1	1	0	1	1	Y
0	1	0	1	1	N	0	1	0	1	1	N	1	1	1	1	1	Y
0	1	1	1	1	N	0	1	1	1	1	N	0	1	1	1	1	N
D						**E**						**F**					
l_i	l_o	r_i	r_o	*u*	*X?*	l_i	l_o	r_i	r_o	*u*	*X?*	l_i	l_o	r_i	r_o	*u*	*X?*
1	0	0	1	1	Y	1	0	0	1	1	Y	1	0	0	1	1	Y
1	0	1	1	1	N	1	1	0	1	1	Y	1	1	0	1	1	Y
1	1	1	1	1	Y	1	1	1	1	1	Y	0	1	0	1	1	N
0	1	1	1	1	N	0	1	1	1	1	N	0	1	1	1	1	N

Table 2: Invalid set of D-Element with l1-stuck-at-0

***l1* s-a-0**				
l_i	l_o	r_i	r_o	*u*
1	1	0	1	1
1	1	1	1	1
1	0	0	1	1

***r2* s-a-0**				
l_i	l_o	r_i	r_o	*u*
0	0	1	0	0
1	0	1	0	0
0	0	1	1	0
1	0	1	1	0

Table 3: Exclusive sets of faults

5. Remedial action for premature firings and faults in redundant logic

As stated in Section 3, solving the comparison synchronization problem is inadequate to detect all possible faults. The problem arises because many faults do not manifest themselves as logic discrepancies that can be detected through output comparison. However, the presence of a fault signifies some discrepancy in the internal signals of the fault-free and the faulty machine. We propose further remedial action that uses these internal signals to detect faults missed by the comparator, thus guaranteeing data integrity under the single-fault model.

Given an asynchronous circuit and its corresponding timing specification, there are a limited number of states that it can be in. We define the *valid set* of a circuit as the set of all possible states allowed in its normal operation. Next, for each non-halting fault, we determine the *invalid set*, which is defined as the set of possible states that the circuit may reach in the presence of the fault. We further define the *exclusive set* for each non-halting fault as the set of states that belong to the invalid set of the fault but not to the valid set. Since all internal signals are considered, we are guaranteed non-emptiness of exclusive sets. Exclusive sets provide adequate information to detect each fault. Therefore, a function that monitors the circuit to detect any state in the exclusive set of a fault is necessary to detect this fault. Since such functions will be implemented in hardware, however, it is important to minimize their cost. The objective of CED is to identify erroneous behavior of the circuit for any input combination that appears during normal operation; therefore, all states in each exclusive set need to be monitored for concurrent error detection. The union of the exclusive sets is required, but incorporating states in the complement of the valid set can reduce the hardware cost of the monitor function.

We illustrate this method on the example circuits. Table (1) provides us with the valid set for the D-Element, the circuit shown in figure (2)(a). The fault *l1*-s-a-0 is a premature firing fault. It alters the timing specification so that [*lo*↑;[¬*li*]] and [*ro*↑;[*ri*]] may happen in parallel. Outside of these events, the faulty circuit behaves the same as the fault free version; therefore, to observe the fault effect, we focus on the different subsequences that may occur. In the presence of this fault, there are six orderings possible (because [¬*li*] must still follow [*lo*↑] and [*ri*] must still follow [*ro*↑]). Letting 1 = [*lo*↑], 2 = [¬*li*], 3 = [*ro*↑], and 4 = [*ri*], the orderings are **A**=[1;2;3;4], **B**=[1;3;2;4], **C**=[1;3;4;2], **D**=[3;4;1;2], **E**=[3;1;4;2], **F**=[3;1;2;4]. Table (2) shows the states for each sequence. **A** is non-faulty behavior so none of its states are outside of the valid set. The column in table (2) labeled "X?" indicates if the state is in the exclusive set for the fault. The same analysis can be used on the fault *r2*-s-a-0 to produce its exclusive set. The exclusive sets of both faults are shown in table (3). Taking the union of these sets and using states not in the valid set as needed, we find an error function, $E_D = (l_i \wedge r_o) \vee (l_o' \wedge r_i)$.

1063-6722/02 $17.00 © 2002 IEEE

The data-dependent control logic and the more complex timing specification of the adder result in a larger valid set and more invalid sets. For brevity, we assume that inputs A and B are both zero, in which case there is only one non-halting fault, hs_4 s-a-1. It is a fault in redundant logic and reduces the performance of the circuit. In effect, this fault forces the circuit to move through the timing specification used when $A \neq B$. The states in the invalid set are too numerous to list here; however, to simplify matters, we note that the intersection of the valid set and the invalid set for this scenario is empty (because in the valid set $hs_4 = 0$). In addition, every state in the invalid set has hs_4=1 for A=0, B=0. Therefore, a possible monitor function in this case is $E=A' \wedge B' \wedge hs_4$. Performing this analysis for all invalid sets yields the following monitor function, E_A, for detecting all non-halting errors in the adder:

$$E_A = ((A \oplus B) \oplus hs_4) \vee (nStart \wedge Start) \vee ((A \oplus B) \wedge nCVin \wedge z)$$

6. Conclusion

Exploiting the numerous advantages of asynchronous circuits requires that design methods equally efficient to those in the synchronous domain be devised. Toward this end, this paper examines the problems that prevent application of existing duplication-based concurrent error detection techniques to asynchronous circuits and proposes solutions. As demonstrated, synchronization issues can be resolved through the proposed comparator circuit that accounts for acceptable time differences in the operation of the original and the duplicate circuit. Furthermore, fault effects that may not be detected by comparison alone, can still be detected through an additional monitor function that makes use of signals internal to the design. Application of the proposed methods to two example circuits displays the effectiveness of these solutions in yielding concurrently testable asynchronous designs.

References

[1] A. Avizienis, J. P. J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments", *IEEE Trans. Comput.*, vol. 17, no. 8, pp. 67-80, 1984.

[2] A. Davis, S. M. Nowick, "An Introduction to Asynchronous Circuit Design", *Technical Report UUCS-97-013*, Department of Computer Science, University of Utah, 1997.

[3] P. J. Hazewindus, *Testing Delay Insensitive Circuits.* Ph.D. Thesis, California Institute of Technology, 1992.

[4] H. Hulgaard, S. M. Burns, G. Borriello, "Testing Asynchronous Circuits: A Survey", *Technical Report CS-TR-94-03-06*, Department of Computer Science and Engineering, University of Washington, 1994.

[5] M. Kishinevsky, A. Kondraytev, L. Lavagno, A. Saldanha, A. Taubin, "Partial-Scan Delay Fault Testing of Asynchronous Circuits", *IEEE Trans. Comput.*, vol. 17, pp. 1184-1198, 1998.

[6] D. Lu, C. Q. Tong, "High-Level Fault Modeling of Asynchronous Circuits", VTS, pp. 190-195, 1995.

[7] Y. Makris, I. Bayraktaroğlu, A. Orailoğlu, "Invariance-Based On-Line Test for RTL Controller-Datapath Circuits*", VTS*, pp. 459-464, 2000.

[8] C. Myers, *Asynchronous Circuit Design*, Wiley, 2001.

[9] S. Mitra, E. J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?", *ITC*, pp. 985-994, 2000.

[10] Y. Mukai, Y. Tohma, "A Method for the Realization of Fail-Safe Asynchronous Sequential Circuits". *IEEE Trans. Comput.*, vol. 23, no. 7, pp. 736-739, 1974.

[11] M. Nicolaidis, Y. Zorian, "On-Line Testing for VLSI - A Compendium of Approaches", *JETTA*, vol. 12, no. 1-2, pp. 7-20, 1998.

[12] A. Paschalis, D. Gizopoulos, N. Gaitanis, "Concurrent Delay Testing in Totally Self-Checking Systems", *JETTA*, vol. 12, no. 1/2, pp. 55-61, 1998.

[13] O. A. Petlin, C. Farnsworth, S. B. Furber, "Design for Testability of an Asynchronous Adder", *IEE Colloquium on Design and Test of Asynchronous Systems (Ref. No.1996/040),* pp. 5/1-9, 1996.

[14] D. H. Sawin, G. K. Maki, "Asynchronous Sequential Machines Designed for Fault Detection". *IEEE Trans. Comput.*, vol. 23, no. 3, pp. 239-249, 1974.

[15] F. Sellers, M.-Y. Hsiao, L. W. Bearnson, *Error Detection for Digital Computers,* McGraw-Hill, 1968

[16] C. Tristam, "It's Time for Clockless", *Technology Review*, pp. 37-41, Oct. 2001.