A Transistor-Level Test Strategy for C^{2}MOS MOUSETRAP Asynchronous Pipelines

Feng Shi
Electrical Engineering Dept.
Yale University
New Haven, CT 06520, USA

Yiorgos Makris
Electrical Engineering Dept.
Yale University
New Haven, CT 06520, USA

Abstract

We discuss a transistor-level test methodology for C^{2}MOS asynchronous pipelines. Unlike their static CMOS counterparts, wherein testing for stuck-at faults and compliance to a few timing constraints typically suffices, dynamic asynchronous pipelines present new challenges which require more elaborate test solutions. More specifically, many gate-level input/output stuck-at faults of a static pipeline style translate into transistor-level stuck-open/stuck-short faults in the dynamic C^{2}MOS version. Therefore, test methods for transistor-level faults are required for dynamic asynchronous pipelines. To this end, we propose a methodology for testing both gate-level stuck-at faults and transistor-level stuck-open/stuck-short faults in C^{2}MOS pipelines. The proposed method does not employ additional hardware and is capable of detecting both gate-level and transistor-level faults, as we demonstrate on the C^{2}MOS version of MOUSETRAP.

1. Introduction

Starting with the classic delay-insensitive *micro-pipeline* [27], an array of alternative asynchronous pipeline styles have emerged over the last fifteen years. As a first improvement, a four-phase micro-pipeline control approach was proposed in [4] to boost performance by employing faster single-phase transparent latches. From then onwards, many variants of micro-pipelines have been developed using alternative control and latch structures [25, 30, 13, 26, 5, 18]. In order to improve performance, most of these asynchronous pipelines employ fine-grained stages and aggressive timing, i.e. they give up the robustness of delay-insensitivity and require that certain timing constraints are met in order to operate correctly. In addition, even more aggressive structures of fine-grain asynchronous pipelines have also been proposed [25, 29, 23, 24, 3], wherein latches are eliminated altogether. These designs usually employ dynamic logic for the datapaths and exploit the inherent latching properties of dynamic gates. Therefore, by avoiding explicit pipeline latches, the performance is further improved, area is saved, and power consumption is reduced. Delay-insensitive codes such as dual-rail data code have also been adopted [3] to indicate the data state and eliminate the need for a dedicated request signal. Among the various issues related to the development of high-speed asynchronous pipelines, this work focuses on *testability*.

A number of methods have been proposed in the past for testing asynchronous pipelines [14, 9, 15, 10, 17, 16]. Most of these methods focus mainly on testing the classic micro-pipeline [27], which is a *delay-insensitive* architecture, i.e. it operates correctly under arbitrary gate and wire delays, except that the standard bundled data timing convention still needs to be observed when datapaths are included. Since then, however, a new generation of high-speed asynchronous pipelines have been developed, introducing a new set of test challenges which make the above asynchronous test methodologies obsolete. Testing methods for some of these new architectures have been proposed in [22], however, they are only for the static CMOS style and are inadequate to handle dynamic gate-level pipelines such as C^{2}MOS MOUSETRAP.

In this paper, we propose a test method for gate-level C^{2}MOS MOUSETRAP pipelines. Our method proposes an equivalent static model of a C^{2}MOS logic for test generation and exploits a previously developed tool-suite for the static CMOS asynchronous pipelines [22] to generate test patterns for gate-level stuck-at faults in C^{2}MOS asynchronous pipelines. Moreover, we present a test method for transistor-level stuck-on and stuck-open faults in C^{2}MOS logic, which we demonstrate on the C^{2}MOS MOUSETRAP pipeline.

The rest of this paper is organized as follows. In section 2, we briefly review previously proposed test methods for asynchronous pipelines. In section 3, we introduce the basics of gate-level C^{2}MOS MOUSETRAP pipelines and we discuss their test challenges. In section 4, we describe our test method for stuck-at faults in C^{2}MOS asynchronous pipelines. In section 5, we propose test methods for transistor-level stuck-open/stuck-short faults in C^{2}MOS MOUSETRAP pipelines.

2. Previous Work

Several researchers have studied the problem of testing asynchronous pipelines in the past. Pagey *et al.* [14] proposed a test generation method for stuck-at faults in traditional micro-pipelines. However, timing faults resulting in bundling constraint violation are not considered. Khoche *et al.* [9] and Petlin *et al.* [15] developed full-scan approaches for testing micro-pipelines. Their methods are able to test not only for stuck-at faults, but also for delay faults resulting in bundling constraint violation. However, full-scan methods increase the test application time and incur hardware overhead which may be prohibitive for fine-grain asynchronous pipelines. King *et al.* [10] developed a method which generates test sequences for faults inside the C-elements of the micro-pipeline control stages and applies them through the circuit. Roncken *et al.* [17, 16]

Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'06)
0-7695-2498-2/06 $20.00 © 2006 IEEE

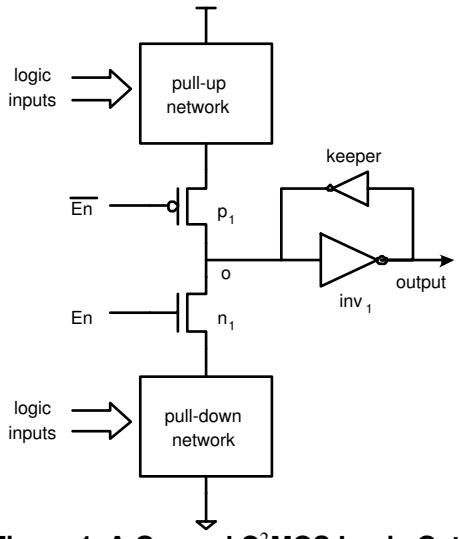

Figure 1. A General C²MOS Logic Gate

proposed a partial scan method to test both datapaths and handshake components in asynchronous pipelines. Test patterns are generated not only for gate I/O stuck-at faults, but also for bridging faults. In addition to voltage testing, I_{DDQ} testing is also performed to improve test quality. Furthermore, a handshaking component, $HOLD$, is introduced to hold the circuit in between a handshaking request and its acknowledgement, in order to regain adequate test control. Shi *et al* [22] developed a general tool-set for testing stuck-at faults in static CMOS asynchronous pipelines, especially ultra-high-speed pipelines which exploit timing constraints to achieve high performance. Moreover, they introduced a fault model and a simple design-for-test (DFT) method to support testing for timing constraint violations, which are critical to the correct operation of ultra-high-speed asynchronous pipelines. However, these methods were developed for static CMOS asynchronous pipelines and do not address the problem of testing transistor-level faults in the dynamic pipelines.

3. Test Requirements of C²MOS MOUSETRAP

Clocked-CMOS (C²MOS) is an attractive approach to implementing gate-level pipelining, which achieves extremely high throughput by partitioning the datapath into the finest-grained stages, each consisting of only a single level of logic with no explicit latches. Figure 1 illustrates the general structure of a C²MOS logic gate. The clock input, En, directly controls the gate through two transistors, one in the pull-up and one in the pull-down network. The gate is enabled and evaluated when En is asserted. When En is de-asserted, it simply holds its value. Moreover, an inverter pair providing weak feedback is typically attached to the gate output to yield a more robust hold operation. C²MOS logic is adapted to gate-level asynchronous pipelines by replacing the clock with handshaking signals to eliminate explicit latches.

A C²MOS implementation of MOUSETRAP [25] (or **M**inimal-**O**verhead **U**ltra-high-**S**p**E**ed **TR**ansition-signaling **A**synchronous **P**ipeline) is illustrated in Figure 2. The structure of a MOUSETRAP asynchronous pipeline includes both control and processing logic. C²MOS identity gates, such as the one shown in Figure 3, serve as latches in the control logic, while C²MOS logic gates in the processing logic provide both logic and latching functionality to eliminate explicit latches. The schematic of the XNOR gates in the control logic is also shown in Figure 4. We note that the above is only one of the possible implementations. A dual-rail implementation was also proposed in [25].

MOUSETRAP pipeline works under a hybrid protocol [25] – *transition signaling* for the handshaking signals, and *level signaling* for the latch enable signal. In its initial state, the pipeline is empty with all the *done*, *req*, and *ack* signals at a low level, and all its latches in transparent mode. The pipeline communicates with the environment through transition signaling, that is, each transition, whether up or down, is treated as a distinct event. When the first request (i.e. a rising transition on req_{N-1}) flows through the pipeline stage $N-1$, $done_{N-1}$ changes to one and En_{N-1} changes to $zero$, which closes the latches in stage $N-1$ in order to lock the stage and block any new request. The same process is applied to the data, which is "bundled", i.e. it arrives before the request. Then req_N rises after a certain delay which matches the delay of the processing logic in this stage, and both $done_N$ and ack_N change to one. As a result, En_N changes to $zero$ to close the latches and lock the results, and En_{N-1} changes to one which opens the latches in stage $N-1$ to accept a new request. After a successive request (i.e. a falling transition on req_{N-1}) flows through stage $N-1$, $done_{N-1}$ changes to $zero$ and En_{N-1} changes to $zero$ to close the latches, and the same process is repeated. Note that for each data item there are two transitions on each En signal, one to capture it and one to release it. Therefore, the XNOR gate serves as a *phase converter* which converts the $done$ and ack signals of transition signaling into level control for the transparent latches. MOUSETRAP is a fine-grain pipeline in the sense that the depth of the data processing logic can be very shallow, comprising in the extreme case just a single level of gates. MOUSETRAP achieves high performance by using simple and fast transparent latches, while avoiding the extra switching activity of a 4-phase communication [4] to simplify the control circuitry.

MOUSETRAP pipelines sacrifice the robustness of a delay-insensitive design style and employ aggressive handshaking protocols in order to achieve high performance. This high performance, however, is obtained at the cost of certain timing constraints that need to be observed for the circuit to function correctly. As a result, traditional testing becomes a more complicated task because not all control stuck-at faults result in pipeline stalling, as in the traditional micro-pipeline architecture [14, 1, 12]. Moreover, delay faults that result in violation of the aforementioned timing constraints also need to be considered. Unlike in the delay-insensitive micro-pipeline, where such faults only result in performance degradation, functional correctness is jeopardized if timing constraints are violated in MOUSETRAP. An intrusive DFT method for testing whether

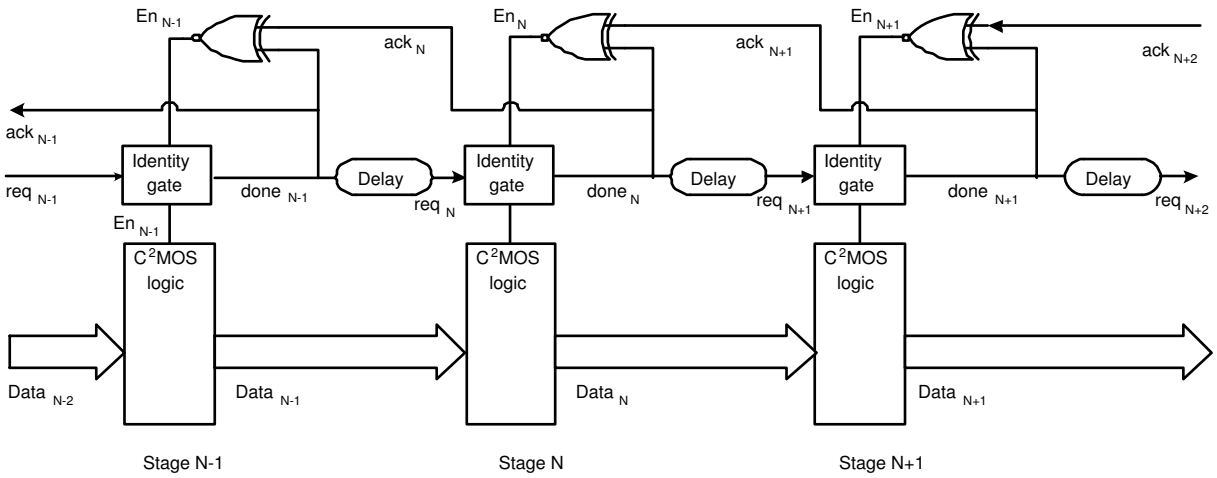

Figure 2. A MOUSETRAP Using C^{2}MOS Logic

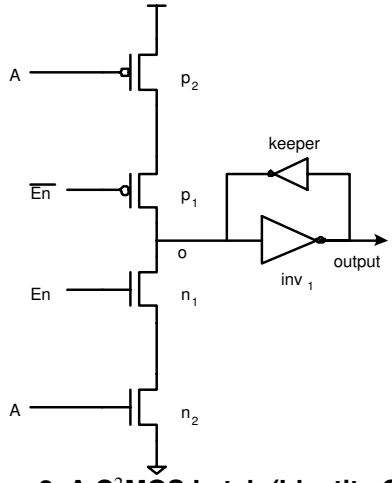

Figure 3. A C^{2}MOS Latch (Identity Gate)

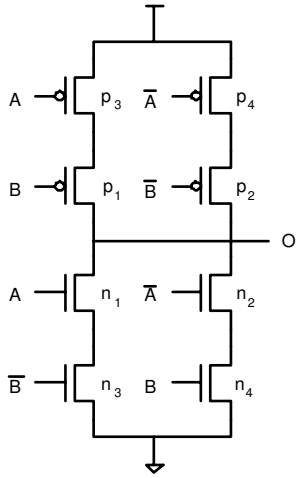

Figure 4. Schematic of An XNOR Gate

these timing constraints are violated in the static MOUSETRAP was proposed in [22]. However, intrusive DFT may be unacceptably expensive in terms of both area and performance overhead, since high-speed pipelines, such as MOUSETRAP, achieve high performance partially due to their very fine-grain pipeline stages. To alleviate this problem, a non-intrusive method for testing timing constraint violations is proposed in [8]. Since timing constraints are identical in the static and dynamic MOUSETRAP, these methods are readily available for the C^{2}MOS version and we will not address them any further in this paper.

The main focus of this paper is on addressing gate-level and transistor-level faults in both the handshaking and the processing logic of the C^{2}MOS version of MOUSETRAP. The method in [22] uses a customized ATPG tool to generate test vectors for stuck-at faults. However, the C^{2}MOS version of MOUSETRAP presents new test challenges over and above its static CMOS counterpart. More specifically, traditional test methods for stuck-at faults are inadequate for testing the dynamic behavior of C^{2}MOS logic. For instance, a stuck-at-0 fault on input En of the C^{2}MOS gate shown in Figure 1 is, in fact, a transistor stuck-open fault, which cannot be handled through conventional methods for stuck-at faults. Therefore, testing for transistor-level stuck-open and stuck-short faults is necessary in C^{2}MOS pipelines.

4. Testing Stuck-At Faults

We start by discussing a strategy for testing stuck-at faults, which remain fundamentally important in the dynamic C^{2}MOS style. Specifically, we describe a simple conversion of the dynamic C^{2}MOS circuit into an equivalent static CMOS circuit for the purpose of test generation. This equivalent model allows the reuse of a previously developed test generation method for the static CMOS style [22], in order to generate test vectors for the dynamic C^{2}MOS style.

4.1. Static Model for Test Generation

Consider the input and output stuck-at faults in a general C^{2}MOS gate, such as the one illustrated in Figure 1. As may be observed, the stuck-at 0/1 fault at input En ($\bar{En}$) is equivalent to the transistor-level fault n_1 (p_1) stuck-open/short (stuck-short/open). Therefore, we defer the description of the test generation method for these faults to section 5. In order to generate test vectors for all other stuck-at faults, we transform the C^{2}MOS logic gate into an equivalent form composed with static

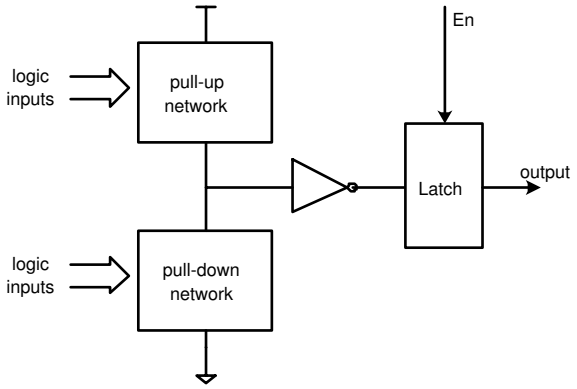

Figure 5. The Equivalent Static Form of a C^{2}MOS Gate for Test Generation

CMOS gates and a latch, as illustrated in Figure 5. In this equivalent static form, the two transistors which latch the gate output are removed and the pull-up/pull-down networks are connected directly to form a static CMOS logic gate. Additionally, an explicit latch is appended to the output to hold the output value when the enable signal is low. Since the circuit of Figure 5 exhibits the same functionality as the circuit in Figure 1, we replace any C^{2}MOS gate with its equivalent static form to generate test patterns for stuck-at faults on logic inputs and outputs. Test generation on the equivalent static form is performed using the method proposed for static CMOS asynchronous pipelines in [22], which we briefly describe in the following subsection for the purpose of completeness.

4.2. Test Generation for Static CMOS Style

The test generation method of [22], which is demonstrated on the static CMOS version of the MOUSETRAP pipeline, divides the circuit in two parts, the handshaking logic (control) and the processing logic (datapath). Each part is then treated individually to reduce the complexity of automatic test pattern generation. In order to test the handshaking logic, an extended version of a tool-suite that was recently developed for testing speed-independent circuits [19, 20, 21] is employed. In order to test the processing logic, simple combinational ATPG suffices by exploiting the transparent latches that are used in MOUSETRAP. Results are reported for the traditional micro-pipeline and for both the linear and non-linear MOUSETRAP pipelines. We refer the reader to [22] for the details of these methods, the key points of which are summarized below.

4.2.1. Testing the Handshaking Logic

Unlike in the traditional robust micro-pipeline, a stuck-at fault in the handshaking logic of MOUSETRAP does not necessarily halt the pipeline. This happens because the correct operation of MOUSETRAP relies upon certain timing constraints, hence it is not delay-insensitive. The test generation method that we propose in [22] is based on a tool-suite that we recently developed for testing speed-independent circuits. This tool-suite performs simulation (SPIN-SIM [19]), ATPG (SPIN-TEST [20]), and test compaction (SPIN-PAC [21]). Its heart is SPIN-SIM, a logic and fault simulator that efficiently detects hazards by extending Eichelberger's classical method [6] to overcome its limitations. In order to improve simulation accuracy, SPIN-SIM adopts a 13-valued algebra [2, 11], maintains the relative order of causal signal transitions, and unfolds time frames judiciously. In addition, complex gates are handled through replacement by pseudo-gates that are equivalent with respect to functionality, timing, and faulty behavior. SPIN-TEST uses SPIN-SIM and an A* search algorithm in order to perform fault-simulation-based ATPG. Finally, SPIN-PAC employs heuristics to combine multiple single-input-change (SIC) vectors into a single multiple-input-change (MIC) vector, as well as an Integer-Linear-Programming (ILP) formulation to compact test sequences via pruning or elimination.

Although we initially developed this tool-suite for speed-independent circuits, we later extended it to handle other classes of asynchronous circuits, such as Delay-Insensitive and Quasi-Delay-Insensitive, by replacing them with their equivalent speed-independent forms. We also extended SPIN-SIM to simulate asynchronous pipelines with particular timing constraints. SPIN-SIM has an inherent feature that comes in handy for this purpose. More specifically, it uses time-stamps to keep track of the relative signal order while simulating speed-independent circuits. This relative signal order is taken into account when evaluating a gate during simulation and increases dramatically the simulation accuracy by eliminating false hazard identification. Based on this capability, timing constraints were easily incorporated as additional relative orders of signals. As a result of these extensions, the SPIN-TEST simulation-based ATPG and the SPIN-PAC test compaction methods are also available for a broader class of asynchronous circuits, including CMOS static asynchronous pipelines.

4.2.2. Testing the Processing Logic

The proposed method for testing stuck-at faults in the processing logic is similar to that proposed by Pagey *et al.* for testing micro-pipelines [14]. When a MOUSETRAP pipeline is empty, all request and acknowledge signals are at a low level and all the latches are transparent. Therefore, all the processing logic blocks are connected in a cascade and can be treated as a single logic block for the purpose of testing. Suppose that C represents the whole block of logic obtained by interconnecting all logic blocks in the pipeline stages in a cascade. Then, any combinational ATPG tool can be used to generate test patterns for stuck-at faults in C. The test patterns are subsequently applied to the input of the pipeline when the pipeline is empty and without sending any request signals. The responses are observed at the output of the pipeline and compared to the expected fault-free responses.

5. Testing Transistor-Level Faults

In this section, we propose a test method for transistor-level faults in the dynamic C^{2}MOS style. While the gate-level stuck-at fault model is used as the predominant fault model for static logic, it is well known that it does not exactly represent the input/output behavior of a faulty MOS logic circuit [28, 7]. This

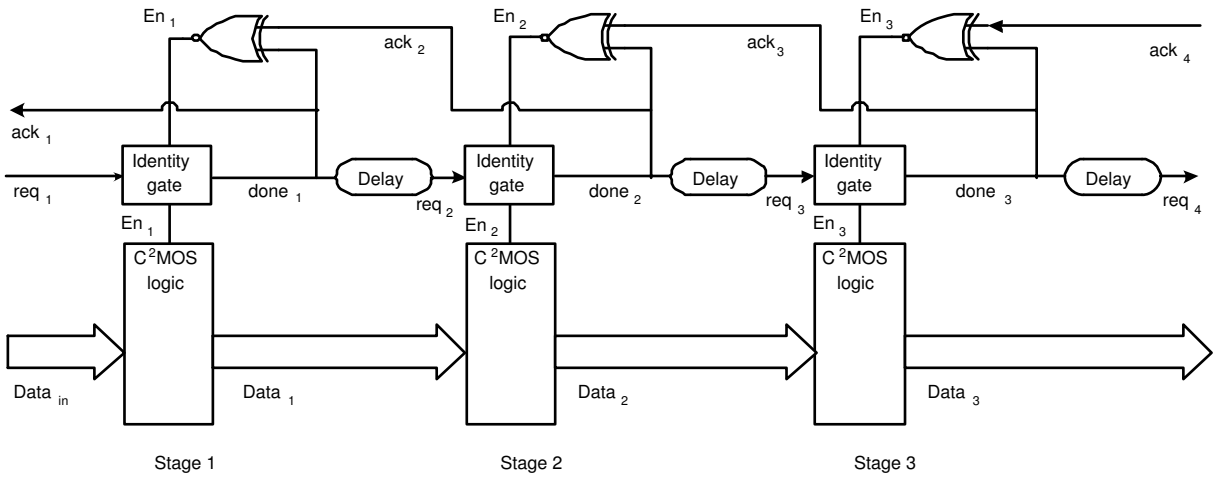

Figure 6. A Three-Stage C²MOS MOUSETRAP

problem is amplified in dynamic logic, wherein some stuck-at faults are in fact transistor-level faults and cannot be handled using traditional test methods. This is the case, for example, for the stuck-at faults on inputs En or $\bar{En}$ of a C²MOS gate. For such circuits, transistor stuck-open and stuck-short faults constitute a more realistic model to investigate.

5.1. Notation

The following notation is introduced to assist in presenting our methods for testing stuck-open and stuck-short faults:

We denote as *PSO*$_1$ (*PSS*$_1$) a stuck-open (stuck-short) fault on the PMOS transistor p_1 in a C²MOS gate such as the one shown in Figure 1. We also denote as *NSO*$_1$ (*NSS*$_1$) a stuck-open (short-short) fault on the NMOS transistor n_1. We denote as a *PSO*$_2$ (*PSS*$_2$) a stuck-open (stuck-short) fault on the PMOS transistor in inverter inv_1 in a C²MOS gate. Finally, we denote as *NSO*$_2$ (*NSS*$_2$) a stuck-open (stuck-short) fault on the NMOS transistor in inverter inv_1.

In order to describe the test patterns for testing stuck-open and stuck-short faults, we also need to define the following test vectors and sequences:

Test vector V_r changes the current value on the request input on the left side of the pipeline (req_1 in the pipeline shown in Figure 6), but keeps the value on the acknowledge input on the right side of the pipeline (ack_4 in Figure 6) unchanged. Test vector V_a keeps the current value on the request input on the left side of the pipeline (req_1 in Figure 6) unchanged, but changes the value on the acknowledge input on the right side of the pipeline (ack_4 in Figure 6). For example, suppose that the current values of the inputs are 01, where the first bit represents the value of the request input on the left side and the second bit represents the value of the acknowledge input on the right side. Then, applying a vector V_r would set the inputs to 11, while applying a vector V_a would set the inputs to 00. Similarly, if the current input values are 11 and a V_a is applied followed by a V_r, then the vector V_a would be 10 and the vector V_r would be 00.

We define test sequence $S_A(n)$ as a sequence of n vectors, which is formed by alternating V_r and V_a vectors starting with a V_r. For example, $S_A(3) = \{V_r, V_a, V_r\}$. We also define test sequence $S_B(n)$ as a sequence of n vectors, which is formed by concatenating n V_r vectors. For example, $S_B(4) = \{V_r, V_r, V_r, V_r\}$. Similarly, we define sequence $S_C(n)$ as a sequence of n vectors, which is formed by concatenating n V_a vectors. For example, $S_C(3) = \{V_a, V_a, V_a\}$.

5.2. Testing Stuck-Open Faults

We present the proposed method for testing stuck-open faults in C²MOS MOUSETRAP in the following way. First, we briefly discuss how to test a general C²MOS logic gate. Based on this, we propose test sequences for the faults in the handshaking logic and a method for testing faults in the processing logic.

5.2.1. C²MOS Logic Gate

As illustrated in Figure 1, the difference between a C²MOS gate and a static CMOS gate is in the two transistors which switch on/off the pull-up/pull-down networks, respectively, and in the inverter pair which is attached to the gate output. Therefore, we first discuss testing of single stuck-open faults in these particular locations.

Consider the PSO$_1$ fault in the C²MOS gate shown in Figure 1. In order to activate this fault, node o needs to be first set to logic $zero$ and then both the pull-up network and transistor p_1 need to be turned on. If the gate is fault-free, node o changes to logic one, otherwise, if p_1 is stuck-open, node o remains at $zero$, hence the fault is activated. Propagation of the fault effect is not an issue, since it can be observed at the gate output. It is obvious that the second step is equivalent to testing a stuck-at-1 fault on the output of the gate. Therefore, the test patterns for such stuck-open faults are generated in two phases. In the first step, a test sequence which sets the gate output to one is found. In the second step, a test sequence which tests for a stuck-at-1 fault at the gate output is generated. The traditional test generation methods for stuck-at faults may be used during the second phase.

For example, in order to test for the PSO$_1$ fault in the identity gate shown in Figure 3, we first set A to one and $En(\bar{En})$

to one ($zero$). This causes the output to become one and internal node o to become $zero$. Then, we apply test vector $A = 0, En(\bar{En}) = 1(0)$, which is able to detect a stuck-at-1 fault on the output. In the fault-free gate, p_1 and p_2 are switched on and node o and the output change to one and $zero$ respectively, but in the faulty gate p_1 is open and node o and the output remain at $zero$ and one respectively. Therefore, the fault is detected.

Consider now the NSO_2 fault in the C^{2}MOS gate shown in Figure 1. In order to test this fault, node o is first set to $zero$, hence the output changes to one. Then, node o is set to one, which causes the output to fall in the fault-free circuit. However, if the stuck-open fault exists, the output remains high and, therefore, the fault is detected. Based on the above discussion, it may be easily observed that any test sequence which detects an NSO_2 fault also detects a PSO_1 fault and vice versa. Therefore, the NSO_2 fault and PSO_1 fault in the same gate are equivalent.

In summary, the following theorem holds for testing the PSO_1/NSO_2 faults in a C^{2}MOS gate:

Theorem 1. *The PSO_1 fault is equivalent to the NSO_2 fault in a C^{2}MOS gate. Both are detected by first applying a test sequence which sets the gate output to one and then a test sequence which tests for a stuck-at-1 fault on the gate output.*

An NSO_1 fault can be tested similarly. In the first step, a test sequence which sets the gate output to $zero$ is found. Then, in the second step, a test sequence which tests for a stuck-at-0 fault at the gate output is generated. Moreover, it can be similarly proved that this fault is equivalent to the PSO_2 fault in the same gate. Therefore, a test sequence generated for NSO_1 faults can also be used for PSO_2 faults and vice versa.

Again, the following theorem holds for testing the NSO_1/PSO_2 faults in a C^{2}MOS gate:

Theorem 2. *The NSO_1 fault is equivalent to the PSO_2 fault in a C^{2}MOS gate. Both are detected by first applying a test sequence which sets the gate output to zero and then a test sequence which tests for a stuck-at-0 fault on the gate output.*

The stuck-open faults on the transistors in the pull-up or pull-down networks in the C^{2}MOS logic shown in Figure 1 are tested through the following method. For the purpose of test generation, a C^{2}MOS logic gate can be transformed into the equivalent static form comprising a static CMOS gate and a latch, as illustrated in Figure 5. Although the two circuit forms have the same functionality, they are not exactly equivalent in terms of test generation for transistor faults in the pull-up and pull-down networks, because of the different location of the latch. However, the following theorem can be used to convert a test sequence for a fault in the equivalent static form to a test sequence for the fault in the C^{2}MOS form.

Theorem 3. *If sequence S detects a stuck-open fault in the pull-up/pull-down networks of the equivalent static form of a C^{2}MOS gate, then S' detects the corresponding fault in the C^{2}MOS gate, where S' is derived from S by inserting patterns when necessary to make sure that the latch is open every time the gate is evaluated.*

Therefore, transistor-level faults in the pull-up/pull-down network of a C^{2}MOS gate can be tested through the test patterns derived from those generated for static CMOS logic.

As a final note, we point out that the stuck-open faults on the transistors in the "keeper" inverter are relatively difficult to test for. Since this inverter only provides weak feedback to keep the output voltage when the enable signal is low and both the pull-up and pull-down networks are disconnected, a stuck-open fault may interrupt the feedback but does not cause any immediate logic error. Therefore, normal voltage test methods are not capable of detecting such faults, calling for more sophisticated test methods or Design-for-Testability (DFT) techniques, which are beyond the scope of this paper.

5.2.2. Handshaking Logic

We now turn our attention to test generation for stuck-open faults in the handshaking logic, such as those in the identity gates or the XNOR gates, which cannot be handled using the traditional methods for synchronous circuits because the circuit has feedback paths. Consider the stuck-open fault on transistors p_2/n_2 in the identity gate shown in Figure 3. It can be proved that the stuck-open fault on p_2/n_2 is equivalent to the stuck-open fault on p_1/n_1. In order to activate the stuck-open fault on p_2, signals En and A are first set to one and transistor n_1 and n_2 are switched on, which sets node o to $zero$. Then, both A and $\bar{En}$ are set to $zero$ to pull node o to one. In the faulty circuit, however, transistor p_2 is stuck-open and node o remains at $zero$, thus the fault is activated on the gate output. Since these are the exact same test requirements as for the stuck-open fault on p_1, the two stuck-open faults are equivalent. Similarly, the stuck-open fault on n_2 is equivalent to that on n_1. Therefore, the test sequence for testing PSO_1 or NSO_1 faults can be used to test for these faults, as summarized in the following theorem:

Theorem 4. *A stuck-open fault on transistor p_2 of the identity gate of the MOUSETRAP handshaking logic, which is equivalent to the PSO_1/NSO_2 faults, can be tested by sequence $S_A(3)$. Similarly, a stuck-open fault on transistor n_2, which is equivalent to NSO_1/PSO_2, can be tested by vector V_r.*

Proof. For a NSO_1/PSO_2 fault, or the stuck-open fault on n_2, the output of the identity gate is set to $zero$ in the initial state. Then V_r activates the fault by setting the req signal to one and the fault effect propagates to the right side of the pipeline, hence the fault is detected. For a PSO_1/NSO_2 fault, or the stuck-open fault on p_2, the output of the identity gate is set to one after test vector V_r is applied. Then V_a acknowledges the pipeline and opens the latch in the last stage. Then, V_r activates the fault by setting the req signal to $zero$ and the fault effect is observed on the right side of the pipeline. □

For example, in order to test the PSO_1 fault in the identity gate of the second stage of the control circuit illustrated in Figure 6, we apply the test sequence $S_A(3) = \{V_r, V_a, V_r\}$. Through the sequence, we first raise req_1 to set node $done_2$ to one, which is initially at $zero$. Then, primary output req_4 changes to one accordingly. After that, ack_4 is set to one to acknowledge the request on req_4. Then, req_1 is set to $zero$. If the stuck-open fault exists, $done_2$ remains at one and so does

primary output req_4; otherwise, both $done_2$ and req_4 change to $zero$. Therefore, the stuck-open fault is detected.

Stuck-open faults in XNOR gates in the control circuit are more difficult to test for. Figure 4 illustrates the transistor level implementation of an XNOR gate. Suppose that there is a stuck-open fault on transistor p_1. In order to detect this fault, node O is first pulled down to zero by setting signal A and B to 01 or 10. In the second step, signal A and B are set to 00, which pulls node O to one in the good circuit. However, if p_1 is stuck-open, node O will remain at $zero$, hence the fault is activated. Note that, in the second phase of test generation, the fault cannot be treated as a stuck-at-0 fault on the gate output O. This is because test vector $AB = 11$ can activate a stuck-at-0 fault on O, but cannot activate the stuck-open fault on p_1. Now consider how to propagate these test patterns to the XNOR gate through the handshaking logic. The following is such an example.

Assume that the pipeline in Figure 6 has a stuck-open fault on transistor p_1 in the XNOR gate of the first stage. Suppose that in the initial state all the latch enable signals are set to one and all other signals are set to $zero$. First, input signal req_1 is set to one, which causes signal $done_1$, req_2, $done_2$, req_3, $done_3$, req_4 and ack_1 to rise, and signal En_3 to fall. After that, V_a is applied to acknowledge the pipeline. Then, req_1 is set to $zero$, which propagates through the stages. When the low level propagates through the first stage but before it goes through the second stage, signal $done_1$ is $zero$ and signal ack_2 is still one, which sets the output of the XNOR gate En_1 to $zero$. This is the first phase for activating the fault. After the low level propagates through the second stage, both $done_1$ and ack_2 are $zero$, which activates the fault in the XNOR gate. In the good circuit, En_1 is high and the first stage is transparent, while in the faulty circuit, En_1 remains low and the first stage is closed. Then, V_a is applied and req_1 is set to one again. Output req_4 will rise if the circuit is fault-free, otherwise it will remain at $zero$. Hence, the fault is detected.

The stuck-open fault on any PMOS transistor other than p_1 in an XNOR gate can be tested in a similar way. It is also easy to prove that the stuck-open fault on p_1 is equivalent to that on p_3, and that the stuck-open fault on p_2 is equivalent to that on p_4, as summarized in the following theorem.

Theorem 5. *A stuck-open fault on any PMOS transistor of an XNOR gate in the control circuit of a C²MOS MOUSETRAP pipeline can be tested by test sequence $S_A(5)$.*

Proof. As illustrated before, if there is a stuck-open fault on transistor p_1 or p_3 and the fault is not activated in the initial state, the fault is activated at the output of the XNOR gate after applying $S_A(4)$ and the pipeline halts. This can be easily checked by applying another V_r and observing the output. If the fault is on transistor p_2 or p_4, then it is activated after applying $S_A(2)$ and the pipeline halts. Therefore, the second V_r is able to indicate whether the fault exists. □

A stuck-open fault on an NMOS transistor in the XNOR gate, however, does not halt the pipeline. For instance, suppose that there is a stuck-open fault on transistor n_1 as illustrated in Figure 4. In order to activate the fault, node O is first set to one. Then, test pattern $AB = 10$ is applied, hence, node O will be set to $zero$ in the good circuit, while it will remain at one in the faulty circuit. Suppose that the faulty gate is in the second stage, input A is connected to $done_3$, and input B is connected to $done_2$. The first step is easy since when the data item passes through the next stage or when the circuit is initialized, En_2 is high and the latches are transparent. Then, In order to set $done_3$ to one, req_1 is set to one. Signal ack_4 is kept at $zero$ to lock the node $done_3$ at one. Note that at the end of this step, En_2 is one since both $done_2$ and $done_3$ are one. After that, req_1 changes to $zero$, which causes $done_2$ to change to $zero$. Now the fault is activated since in the fault-free circuit En_2 is $zero$, while in faulty circuit En_2 is one, which prevents the pipeline from halting. The fault effect is observed after setting req_1 to one and then to $zero$ in the following steps. Signal ack_1 is one in the fault-free circuit since the latches in the first stage are closed, while it is $zero$ in the faulty circuit. Therefore, the fault is detected. In general, the following theorem holds for detecting these faults.

Theorem 6. *A stuck-open fault on any NMOS transistor of an XNOR gate in the control circuit of a C²MOS MOUSETRAP pipeline prevents the pipeline from being filled, hence, it can be detected by applying test sequence $\{S_B(N+1), V_a, V_r\}$, where N is the number of stages in the pipeline.*

Proof. The stuck-open faults on n_1 and n_3 and on n_2 and n_4 in the XNOR gate shown in Figure 4 are equivalent. Any such fault can be activated by first setting the output of the XNOR gate to one, then setting the input to either $AB = 01$ or $AB = 10$. After test sequence $S_B(N)$ is applied to a fault-free pipeline, as illustrated in Figure 7 which is a three stage example, the input state of each XNOR gate is set to one of the above test patterns, and right before that, the gate output is set to one. Therefore, if there is any such fault, it may be activated during this procedure. Thus, the En signal in the faulty stage will be stuck at one and the latches will not be able to close, therefore, the faulty pipeline will never be filled. In this case, another V_r is able to check whether the pipeline is full and, thus, detect the fault. If the fault is not activated in the above procedure, we apply vector V_a. As illustrated in Figure 8 for an example of a three-stage pipeline, the input state of each XNOR gate in the fault-free pipeline switches to the other test pattern after V_a is applied, and right before that, the gate output is set to one. Therefore, the fault must be activated during this procedure. Another V_r detects the fault by checking whether the pipeline is full. □

5.2.3. Processing Logic

If we test the faults in the datapath logic under the initialization state of the pipeline, that is, when all the enable signals are high and all latches are transparent, the datapath logic may be regarded as a whole static CMOS logic block obtained by cascading the equivalent static logic in each stage. As a result, for the transistor faults in the pull-up/pull-down networks of the C²MOS processing logic, traditional test generation methods for combinational static CMOS logic can be directly employed to generate test sequences according to Theorem 3. Since the latches are always open, no transformation is needed for the generated test sequences. For the remaining faults, the test generation process takes place in two steps according to Theorem 1

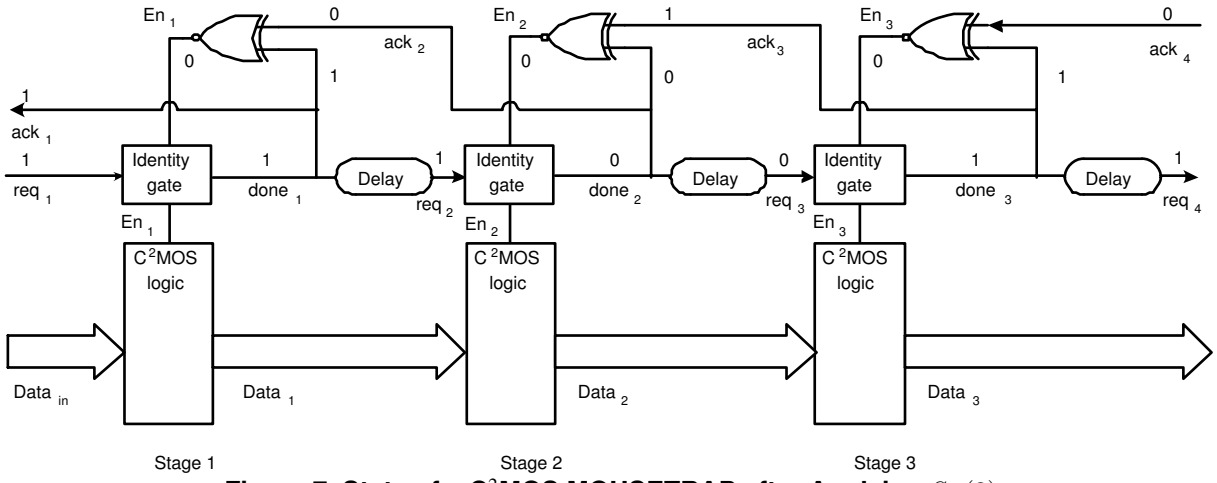

Figure 7. State of a C^{2}MOS MOUSETRAP after Applying $S_B(3)$

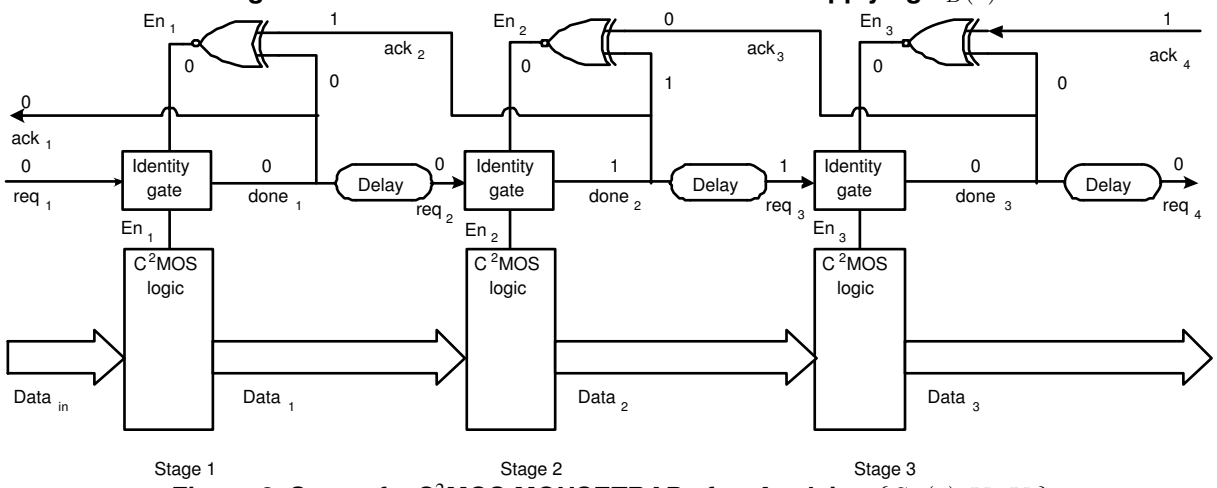

Figure 8. State of a C^{2}MOS MOUSETRAP after Applying $\{S_B(3), V_r, V_a\}$

or 2, both of which may employ the available ATPG tools for combinational static CMOS logic.

5.3. Testing Stuck-Short Faults

In this section, we first discuss briefly how to test a general C^{2}MOS logic gate for stuck-short faults. Based on this, we propose test sequences for the faults in the handshaking logic and a method for testing faults in the processing logic.

We note that, in the following, we use IDDQ tests to detect some of the stuck-short faults in C^{2}MOS circuits. However, IDDQ tests may be obsolete for CMOS processes in 90nm and beyond due to the excessive leakage currents. In that case, we can still use voltage-based tests to detect these faults. Yet, such tests may only *potentially detect* these faults, i.e. some faults may be detected, while others may not, depending on the driving strengths of the pull-up/down networks. Undetected faults, however, do not necessarily reduce the quality of test, since they may not impact the correct functionality of the circuit.

5.3.1. C^{2}MOS Logic Gate

Single stuck-short faults in CMOS static logic are usually detected through IDDQ tests, since when the fault is activated, both the pull-up and pull-down networks are switched on in the faulty circuit, while only one of them is on in the fault-free circuit. However, a stuck-short fault on p_1 or n_1 of the C^{2}MOS gate in Figure 1, or a PSS$_1$/NSS$_1$ fault, may not be detected by IDDQ tests. This is because when the fault on p_1/n_1 is activated, transistor n_1/p_1 is always off in both the fault-free and the faulty circuit. But this does not imply that these faults are undetectable. In fact, they can actually be detected through voltage tests, as explained below:

Theorem 7. *The following procedure detects a PSS$_1$ (NSS$_1$) fault in a C^{2}MOS logic gate. First, a test sequence is applied to set the gate output to $zero$ (one). Then, the latch is closed by setting $\bar{En}(En)$ to one ($zero$). Last, a vector is applied to the logic input which would set the gate output to one ($zero$) if the latch had been open.*

Proof. In the first step the gate output is set to $zero$ (one). Then in the second step both p_1 and n_1 are switched off in the fault-free circuit, while p_1 (n_1) is still on in the faulty circuit. Finally, in the third step the pull-up (pull-down) network is switched on, and the fault is activated and observed. The output remains $zero$ (one) in the fault-free circuit, while it changes to one ($zero$) in the faulty circuit. □

On the other hand, a stuck-short fault on the PMOS transistor in inverter inv_1, or a PSS$_2$ fault, and a stuck-short fault on the NMOS transistor in inverter inv_1, or a NSS$_2$ fault, are easy to detect through IDDQ tests. For fault PSS$_2$, a test pattern

which sets node o to one, or the output to $zero$ detects the fault. For fault NSS_2, a test pattern which sets node o to $zero$, or the output to one detects the fault. Similarly, the stuck-short fault on the PMOS transistor in the feedback inverter can be detected by a test pattern which sets node o to $zero$, or the output to one. Also, the stuck-short fault on the NMOS transistor in the feedback inverter can be detected by a test pattern which sets node o to one, or the output to $zero$.

Theorem 8. *The PSS_2 (NSS_2) fault, or the stuck-short fault on the NMOS (PMOS) transistor in the feedback inverter, in a C²MOS gate can be detected through IDDQ tests by setting the gate output to $zero$ (one).*

The test pattern for any stuck-short fault in the pull-up/pull-down network in a C²MOS gate can be generated through the equivalent static form. First, we generate the test vector v which detects the corresponding stuck-short fault in the equivalent static form through IDDQ tests. Then, we apply v to the logic input of the C²MOS gate, and open the latch by setting En and $\bar{En}$ to one and $zero$ respectively. Thus, the fault in the C²MOS gate is detected.

5.3.2. Handshaking Logic

As in the case of stuck-open faults, stuck-short faults in the handshaking logic need special attention since, generally, they cannot be handled by the tools for synchronous circuits. Based on Theorem 7, the following theorem holds for stuck-short faults:

Theorem 9. *The PSS_1 or NSS_1 faults in any identity gate in the handshaking logic can be detected by sequence $\{S_B(N+1), V_a, V_r\}$, where N is the length of the pipeline.*

Proof. If the fault to test for is PSS_1 and there are r stages behind the faulty stage, where r is odd, or the fault is NSS_1 and there are r stages behind the faulty stage, where r is even, then test sequence $S_B(r+1)$ sets the output of the identity gate under test to $zero$ in case of a PSS_1 fault or one in case of a NSS_1 fault, and then closes the latch in this gate. The following V_r activates the fault by changing the value on the gate output and makes the latch in the faulty stage open in the faulty circuit. Therefore, the faulty pipeline cannot be filled after $S_B(N+1)$ is applied, hence the fault is detected. Otherwise, the fault is activated by applying V_a after $S_B(N+1)$, since after that the value on the output of each identity gate changes. Then, another V_r is applied to detect the fault, since the fault-free pipeline is full while the faulty pipeline is not. Therefore, sequence $\{S_B(N+1), V_a, V_r\}$ detects the PSS_1 and NSS_1 fault in any identity gate. □

Then, according to Theorem 8, any stuck-short fault in the output inverter and the keeper inverter in the identity gates in the pipeline control circuit can be detected either in the initial state or by applying V_r.

The stuck-short faults in the XNOR gates in the pipeline control circuit can be tested in the following way. If the fault is on a PMOS transistor, such as transistor p_1 illustrated in Figure 4, the test pattern for IDDQ tests is $AB = 01$. Assume that this XNOR gate is in the second stage in Figure 6, and input A is connected to $done_3$, and input B is connected to $done_2$. Therefore, A is already set to $zero$ in the initial state. Then we set ack_4 to one to lock the value on $done_3$, and we set req_1 to one, which cause $done_2$ to change to one. Therefore, the fault is activated and detected by IDDQ tests. In general, the following theorem holds:

Theorem 10. *A stuck-short fault on any PMOS transistor of an XNOR gate in the control circuit of a MOUSETRAP can be tested through IDDQ tests by test sequence $\{S_B(N), V_a, V_r\}$, where N is the number of stages in the pipeline.*

Proof. A stuck-short fault in any PMOS transistor in an XNOR gate must be detected by either vector $AB = 01$ or $AB = 10$. Suppose the fault is in the i-th stage of the pipeline. Test sequence $S_B(N-i+1)$ sets the input state of the XNOR gate under test to one of the above test patterns. If this input state activates the fault, the fault is detected. Otherwise, the input state of the XNOR gate may switch to the other vector after V_a is applied following $S_B(N)$ if it is not in the first stage, and the fault is detected. The fault may not be activated in this case for the following reason. If the output value of the XNOR gate is one when the fault is activated, the latch cannot close appropriately and the requests stored in the earlier stages may pass through the latch and cancel each other. If all the requests in the earlier stages disappear, the fault will not be activated. However, another V_r will activate the fault in this case. Note that this vector also activates the fault if it is in the first stage. Therefore, any stuck-short fault in this XNOR gate will be tested by the sequence $\{S_B(N), V_a, V_r\}$. □

If the stuck-short fault is on the NMOS transistor, for example on n_1/n_4, the test pattern for IDDQ tests is $AB = 00$. No matter how the XNOR gate is connected to the pipeline, the fault can be detected in the initial state. If the fault is on n_2/n_3, the test pattern is $AB = 11$ and the fault can be detected after test sequence $S_A(2)$ is applied. The following theorem holds:

Theorem 11. *A stuck-short fault on any NMOS transistor of an XNOR gate in the control circuit of a MOUSETRAP can be detected by IDDQ tests in the initial state, or by applying test sequence $S_A(2)$.*

The stuck-short fault on n_2/p_2 of an identity gate in the pipeline control circuit might be difficult to detect by IDDQ tests. This is the case, for example, if the fault is on p_2 in an identity gate which is in the second stage of the pipeline in Figure 6. In the initial stage, $A = 0$ and $En = 1$. Then, req_1 is set to one, which cause A to rise. Although the fault is activated, the output of the identity gate might be in a metastable state since both the pull-up and the pull-down networks are on. If $done_2$ is resolved to either one or $zero$, the fault is detected. Otherwise, the value of En_2 is unknown and might finally become $zero$, which switches off both the pull-up and pull-down networks, thus making the fault undetectable. If it is assumed that the output of the faulty identity gate must not be in a metastable state, the fault can be detected either in the initial state, or by applying test sequence $S_A(2)$.

5.3.3. Processing Logic

The stuck-short faults in the pull-up/pull-down networks of C²MOS logic in the datapath can be tested using methods for traditional static CMOS logic. In the initial state, the pipeline latches are all transparent, and the whole data path is equivalent to a combinational static CMOS logic block. Therefore, the test generation methods for stuck-short faults in combinational static CMOS logic can be used for these faults.

For testing any stuck-short fault PSS_1 (NSS_1) in a C^{2}MOS logic gate of the processing logic, the following procedure may be applied. First a test vector v_1 which tests a stuck-at-1 (stuck-at-0) fault on the gate output in the whole equivalent combinational logic is applied. Vector v_1 sets the output of the gate under test to $zero$ (one). Then, the latch in this gate is closed by applying the test sequence $S_B(r+1)$, where r is the number of stages behind the faulty stage. After that, a test vector v_2 is applied to the logic input which sets the gate output to one ($zero$) in the whole equivalent combinational logic. The gate output remains at $zero$ (one) in the fault-free circuit, while it changes to one ($zero$) in the faulty circuit. Therefore, the fault is activated. Finally, the fault effect is propagated and observed at the right side of the pipeline by applying test sequence $S_C(r)$. The fault effect propagates to the data output since v_1 is able to test the stuck-at-1(stuck-at-0) fault in the whole equivalent static combinational logic.

For any other stuck-short faults in the processing logic, such as those on the attached inverter pair in any C^{2}MOS logic gate in the processing logic, any test vector which sets the output of the gate under test to one or $zero$, according to Theorem 8, suffices to detect the fault through IDDQ tests when the pipeline is in the initial state.

6. Conclusion

Dynamic C^{2}MOS logic, which is often employed to implement ultra-high-speed gate-level asynchronous pipelines, imposes new test requirements and test challenges over and above those of the traditional static CMOS styles. Using the C^{2}MOS MOUSETRAP pipeline as an example, we demonstrated how to address these test challenges. Initially, we showed the derivation of an equivalent static model, which enables test vector generation for stuck-at faults in C^{2}MOS pipelines through existing tools for static CMOS pipelines. Since stuck-at faults are insufficient to adequately characterize the functionality of dynamic logic, we then proposed methods for testing transistor stuck-open/stuck-short faults in all parts of the C^{2}MOS MOUSETRAP pipelines. Our methods demonstrate that transistor-level testing of dynamic C^{2}MOS pipelines is indeed feasible without the need for additional hardware or highly specialized ATPG tools.

References

[1] P. Beerel and T. Meng. Semi-modularity and self-diagnostic asynchronous control circuits. In C. H. Séquin, editor, *Advanced Research in VLSI*, pages 103–117. MIT Press, 1991.

[2] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell. Delay fault models and test generation for random logic sequential circuits. In *Proc. of the 29th Design Automation Conference*, pages 165–172, 1992.

[3] U. Cummings. Pivotpoint: clockless crossbar switch for high-performance embedded systems. In *Proc. IEEE Micro.*, pages 48–59. IEEE Computer Society Press, 2004.

[4] P. Day and J. V. Woods. Investigation into micropipeline latch design styles. *IEEE Transactions on VLSI Systems*, 3(2):264–272, 1995.

[5] J. Ebergen. Squaring the FIFO in GasP. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 194–205. IEEE Computer Society Press, 2001.

[6] E. B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM Journal of Research and Development*, 9(2):90–99, 1965.

[7] J. Galiay, Y. Crouzet, and M. Vergniault. Physical versus logical fault models MOS LSI circuits: Impact on their testability. *IEEE Trans. Compu.*, C-29:527–531, June 1980.

[8] G. Gill, A. Agiwal, M. Singh, F. Shi, and Y. Makris. Low overhead testing of delay faults in high-speed asynchronous pipelines. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 2006.

[9] A. Khoche and E. Brunvand. Testing micropipelines. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 239–246, 1994.

[10] M. King and K. Saluja. Testing micropipelined asynchronous circuits. In *Proc. International Test Conference*, pages 329–338, 2004.

[11] D. S. Kung. Hazard-non-increasing gate-level optimization algorithms. In *International Conference on Computer Aided Design*, pages 631–634, 1992.

[12] A. J. Martin and P. J. Hazewindus. Testing delay-insensitive circuits. In C. H. Séquin, editor, *Advanced Research in VLSI*, pages 118–132. MIT Press, 1991.

[13] C. E. Molnar, I. W. Jones, W. S. Coates, J. K. Lexau, S. M. Fairbanks, and I. E. Sutherland. Two FIFO ring performance experiments. *Proc. of the IEEE*, 87(2):297–307, 1999.

[14] S. Pagey, G. Venkatesh, and S. Sherlekar. Issues in fault modeling and testing of micropipelines. In *Proc. of the Asian Test Symposium*, pages 107–111, 1992.

[15] O. A. Petlin and S. B. Furber. Scan testing of micropipelines. In *Proc. IEEE VLSI Test Symposium*, pages 296–301, 1995.

[16] M. Roncken, E. Aarts, and W. Verhaegh. Optimal scan for pipelined testing: An asynchronous foundation. In *Proc. International Test Conference*, pages 215–224, 1996.

[17] M. Roncken and E. Bruls. Test quality of asynchronous circuits: A defect-oriented evaluation. In *Proc. International Test Conference*, pages 205–214, 1996.

[18] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. Immediato, and K. Jenkins. Asynchronous interlocked pipelined CMOS circuits operating at 3.3–4.5 ghz. In *Proc. ISSCC*, pages 194–205. IEEE Computer Society Press, 2000.

[19] F. Shi and Y. Makris. SPIN-SIM: Logic and fault simulation for speed-independent circuits. In *Proc. of International Test Conference*, pages 597–606, 2004.

[20] F. Shi and Y. Makris. SPIN-TEST: Automatic test pattern generation for speed-independent circuits. In *Proc. of International Conference on Computer Aided Design*, pages 903–908, 2004.

[21] F. Shi and Y. Makris. SPIN-PAC: Test compaction for speed-independent circuits. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 71–74. IEEE Computer Society Press, 2005.

[22] F. Shi, Y. Makris, S. Nowick, and M. Singh. Test generation for ultra-high-speed asynchronous pipelines. In *Proc. of International Test Conference*, pages 39.1–39.10, Nov 2005.

[23] M. Singh and S. M. Nowick. Fine-grain pipelined asynchronous adders for high-speed DSP applications. In *Proc. of the IEEE Computer Society Workshop on VLSI*, pages 111–118. IEEE Computer Society Press, 2000.

[24] M. Singh and S. M. Nowick. High-throughput asynchronous pipelines for fine-grain dynamic datapaths. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 198–209. IEEE Computer Society Press, 2000.

[25] M. Singh and S. M. Nowick. MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines. In *Proc. International Conf. Computer Design (ICCD)*, pages 9–17, 2001.

[26] I. Sutherland and S. Fairbanks. GasP: A minimal FIFO control. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 46–53. IEEE Computer Society Press, 2001.

[27] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, 1989.

[28] R. L. Wadsack. Fault modeling and logic simulation of CMOS and MOS intergrated circuits. *The Bell System Techinical Journal*, 57(5):1449–1474, May–June 1978.

[29] T. E. Williams. *Self-Timed Rings and their Application to Division*. PhD thesis, Stanford University, 1991.

[30] K. Y. Yun, P. A. Beerel, and J. Arceo. High-performance asynchronous pipeline circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, 1996.