--

--

Identifying Sequential Redundancies Without Search

Mahesh A. Iyer†, **David E. Long***, and **Miron Abramovici***

† *Synopsys, Inc., Mountain View, CA - 94043.*

* *Lucent Technologies - Bell Labs., Murray Hill, NJ 07974.*

Abstract

Previous solutions to the difficult problem of identifying sequential redundancy are either based on incorrect theoretical results, or rely on unrealistic simplifying assumptions, or are applicable only to small circuits. In this paper, we show the limitations of the existing definitions of sequential redundancy and introduce a new concept of c-cycle redundancy as a generalization of the conventional notion of sequential redundancy. We present an efficient algorithm, FIRES, to identify c-cycle redundancies without search. FIRES does not assume the existence of a global reset nor does it require any state transition information. FIRES has provably polynomial-time complexity and is practical for large circuits. Experimental results on benchmark circuits indicate that FIRES identifies a large number of redundancies. We show that, in general, the redundant faults identified by FIRES are not easy targets for state-of-the-art sequential test generators.

1. Introduction

The design of a logic circuit is done manually or by using synthesis tools. In either case, some unintentional logical redundancies may be introduced in the circuit. Identifying redundancies may provide useful feedback for designers, helping them in locating design errors or finding ways to simplify the circuit. Redundancy identification (RID) is very useful in synthesis, since every redundant fault defines a region that can be eliminated from the circuit. In a combinational circuit, an untestable stuck-at fault is always caused by a redundancy. In a sequential circuit, however, untestability and redundancy are not equivalent concepts[1]. Any practical state-of-the-art sequential test generator cannot distinguish between untestable and redundant faults.

The presence of untestable and redundant faults complicates automatic test generation (ATG). In addition to complicating ATG, redundancies have many other detrimental effects. The presence of a redundant fault may preclude the detection of other faults in the circuit[2]. Redundancies increase the chip area, the power consumption, and often the propagation delays in the circuit. Redundancies may unnecessarily reduce the yield of the IC manufacturing process[3]. Many sequentially redundant faults become detectable with full-scan testing, and many sequentially and combinationally redundant faults become detectable with I_{DDQ} testing. Although the circuit remains fully operational in the presence of a redundant fault, full-scan testing or I_{DDQ} testing will reject that faulty circuit and result in a yield loss.

Most of the previous work on the identification of sequentially untestable and redundant faults[4] [5] [6] [7] [8] [9] [10] [11] [12] [13] is based on ATG. Cheng proposed a procedure for identifying and removing a subset of sequential redundancies (called feedback-free sequential redundancies)[4] [5] [6]. The method relies on converting a sequential circuit into a feedback-free model and uses a modified sequential test generator to identify sequentially untestable and redundant faults[4] [6]. The procedure used to verify whether an untestable fault is indeed redundant was highly complex. Later, special classes of untestable faults that are guaranteed to be redundant were identified[5] [6]. It was shown that in a balanced pipeline circuit, every untestable fault is redundant, and that in a general sequential circuit, unactivatable and unpropagatable faults are redundant[5] [6]. Special procedures were devised to identify these subsets of redundant faults. Recently, it has been shown that the particular solutions proposed in[5] [6] are incorrect[14] [15].

Sequential RID based on implicit state enumeration was proposed in[7]. This method uses ATG and relies on the simplifying assumption that the circuit has a fault-free global reset. The approach uses binary decision diagrams (BDDs) as a platform to deal with the reachability information of sequential circuits. Because of the limitations of BDDs, it cannot handle some large circuits. In addition, the method is not applicable to circuits without a global reset mechanism[14] [15].

The method of Agrawal and Chakradhar[8] [9] uses combinational ATG to target certain faults in an iterative array model of finite length derived from a sequential circuit. Their important result is that a combinationally untestable target fault in this array corresponds to a sequentially untestable fault in the original circuit, thus allowing a subset of the sequentially untestable faults to be found by less complex combinational techniques. Their method, however, still requires exhaustive search to identify untestability, and it is not applicable to sequential RID.

Pomeranz and Reddy proposed a method to identify sequential redundancy, based on combinational ATG for multiple faults, which is not practical for large circuits[10]. Their method is an extension of the multi-fault theorem for sequential untestability that was presented in[8] [9]. More recently, they proposed a method to determine whether a sequentially untestable fault prevents the initialization of the circuit[11]. They argue that faults which do not exhibit this behavior can be considered as redundant. The method relies on verifying that the faulty circuit has an initialization sequence and requires knowledge of state transition information; thus this method does not seem practical for large circuits. More importantly, their method assumes that the initialization sequence of the circuit can be altered. From a designer's viewpoint this assumption may be unacceptable, since the reset sequence is considered part of the behavior of the circuit.

The sequential optimization method of Entrena and Cheng [12] [13] identifies a redundant fault when the mandatory assignments required to detect it cannot be consistently justified. Their analysis is performed only in the fault-free circuit and they claim a fault as redundant if the mandatory assignments to detect it end up with conflicts or if the state justification procedure starts looping over previously unjustified states. Both these claims were shown to be incorrect[14] [15] [11].

Formal verification techniques were used for sequential RID by Moondanos and Abraham[16]. Their method uses a restrictive definition of redundancy and relies on verifying that the good and the faulty circuits (for a redundant fault) have equivalent state tables. Moreover, their method is not practical for large circuits, because it relies on building the state transition tables for the good and faulty circuits for each modeled fault.

In earlier work[17] [18], we proposed a Fault-Independent algorithm for REdundancy identification (FIRE) in combinational circuits. FIRE is radically different from any ATG-based approach for combinational RID, which finds a fault to be redundant when all possible ways to detect that fault end up with conflicts. In contrast, FIRE starts with a possible conflict and finds faults for which that conflict is necessary for detection. In addition to being untestable, these faults are also redundant. The key advantage is that FIRE accomplishes RID without any search.

More recently, we proposed an algorithm to Find UNTESTable faults (FUNTEST) in sequential circuits[19]. FUNTEST extends the concept of[17] [18] to sequential circuits using the single-fault theorem from[8] [9] and identifies sequentially untestable faults without search. However, the use of the single-fault theorem to identify sequentially untestable faults precludes identifying redundancy[10]. The FUNI algorithm[20] Finds UNtestable faults without search using Illegal states. FUNI cannot distinguish between untestable and redundant faults.

Main Contributions

In this paper, we first discuss the existing definitions of sequential redundancy and show their limitations. Next, we introduce the concept of a *c*-cycle redundant fault, as a generalization of the conventional notion of sequential redundancy. We present the sequential FIRE algorithm (FIRES) that identifies *c*-cycle redundancies without search. Our important result is that a fault which requires a conflict as a necessary condition for its detection is *c*-cycle redundant.

We assume that all flip-flops (FFs) are controlled by a single implicit clock. Unlike[7], FIRES can be used for circuits without a global reset. FIRES is a direct method to identify redundancies and can be used efficiently in synthesis to remove sequential redundancies. (None of the currently available synthesis tools have such a feature). FIRES runs in polynomial time and is practical for large circuits. It can be used as a preprocessor to any ATG program, which can avoid targeting the faults identified as redundant and thus save the large computational effort associated with them.

The notion of a *c*-cycle delayed replacement of a circuit[21] is an equivalent concept, in the sense that a *c*-cycle redundant circuit is a *c*-cycle delayed replacement of the original circuit. A resynthesis procedure for *c*-cycle delay replaceability was also proposed in[21]. Their procedure is based on BDDs and allows more flexibility during sequential optimization.

The rest of this paper is organized as follows. Section 2 reviews our earlier work on RID for combinational circuits. Section 3 discusses the relation between untestability and redundancy in sequential circuits. Section 4 establishes the theoretical framework for the FIRES algorithm and Section 5 presents the algorithm. Section 6 presents our experimental results and Section 7 concludes the paper.

2. The Combinational FIRE Algorithm

This section briefly reviews the FIRE algorithm for combinational RID[17] [18] [14]. For every stem s, FIRE determines the sets of faults S_0 and S_1, which require, respectively, $s = 0$ and $s = 1$ as necessary conditions for their detection. This is accomplished using an improved version of the uncontrollability and unobservability analysis proposed in[22]. In this analysis, $\bar{0}$ ($\bar{1}$) denotes the status of a line that is uncontrollable for value 0 (1). For every stem s, FIRE propagates $s = \bar{0}$ ($\bar{1}$) to find the set of faults S_0 (S_1) that become untestable if s cannot have value 0 (1). Then the set of faults S that require both $s = 0$ and $s = 1$ (i.e. a conflict) for detection is given by $S = S_0 \cap S_1$. FIRE marks the faults in S as combinationally redundant.

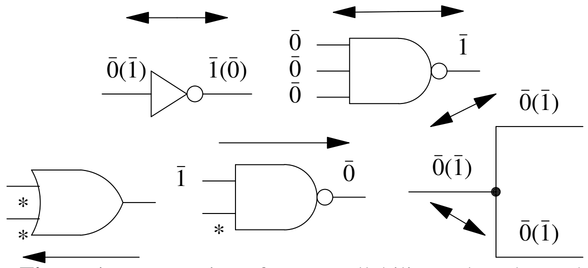

Figure 1. Propagation of uncontrollability and unobservability

Figure 1 illustrates the propagation of uncontrollability indicators. Uncontrollability can propagate forward and backward, as shown by the arrows. For example, the output of a NAND gate is $\bar{0}$ if at least one input is $\bar{1}$, and is $\bar{1}$ if and only if all its inputs are $\bar{0}$. Similar rules apply to other gate types.

In a combinational circuit, if a gate input cannot be set to the noncontrolling value of the gate, all the other inputs become unobservable (unobservable lines are marked with a "*" in Figure 1). The unobservability status propagates from a gate output backward to all its inputs. When all fanout branches (FOBs) of a stem s are marked unobservable, FIRE uses special rules to decide whether s should also be marked as unobservable (to avoid marking faults that can be detected using multiple-path sensitization)[17] [18] [14].

During its analysis, FIRE identifies as untestable, faults that cannot be activated (*s-a*-0 on $\bar{1}$ lines and *s-a*-1 on $\bar{0}$ lines) and faults that cannot be propagated (both faults on unobservable lines).

3. Sequential Untestability and Redundancy

Redundancy is a property of a circuit. A fault is redundant, if the behavior of the faulty circuit cannot be distinguished from that of the fault-free circuit. (Relying on three-valued simulation results to verify this fact may lead to erroneous results.) Pomeranz and Reddy have provided the following definitions[23].

Definition 1: A fault f is said to be *detectable* if there exists an input sequence I such that for every pair of initial states S and S^f of the fault-free and faulty circuit, respectively, the response $Z(I, S)$ of the fault-free circuit to the input sequence I is different from the response $Z^f(I, S^f)$ of the faulty circuit (at some time on some output).

Definition 2: A fault is *untestable* if it is not detectable.

Definition 3: A fault is *partially testable* if there exists an initial state S^f of the faulty circuit and an input sequence I such that for every fault-free initial state S, the response of the fault-free circuit to I starting from S, $Z(I, S)$, is different from the response of the faulty circuit starting from S^f, $Z^f(I, S^f)$. (This definition differs from that in[23], in that it includes testable faults. A fault is testable if it is partially testable for all initial states of the faulty circuit.)

Definition 4: A fault is *redundant* if it is not partially testable.

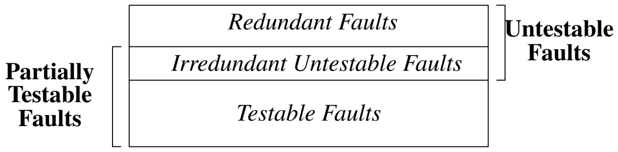

Figure 2. The structure of the fault universe

Figure 2 illustrates the relations between testable, untestable, and redundant faults.

The following example illustrates a shortcoming of Definition 4. It is provided to motivate an alternative definition of redundancy that is introduced in the next section.

Example 1: Consider the circuit in Figure 3 and the fault $c1$ *s-a*-1. This fault is untestable because the requirements to detect it ($c = 0$ and $b = 1$) lead to conflicting assignments on stem a. Independent of its initial state upon power-up, the fault-free circuit can never produce the response $\{d, c2\} = \{1\ 0\}$. However, if the faulty circuit powers up in state $\{b, c\} = \{1\ 0\}$, this output response will be observed. Hence fault $c1$ *s-a*-1 is partially testable and irredundant under Definition 4. However, the circuit in Figure 3 is intuitively redundant, since the same signal a is fed twice into the same gate d through different FFs. □

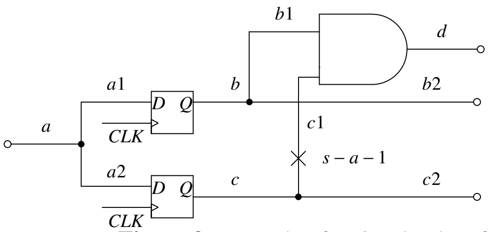

Figure 3. Example of an irredundant fault

Another restricted definition of redundancy that uses specific initialization sequences (without imposing any restriction on time of observation) has also been proposed in[23]. It can be shown that the fault in this example remains irredundant under this restricted definition as well[15].

4. c-Cycle Redundancy

In this section, we introduce the concept of a c-cycle redundancy, which is a generalization of the conventional notion of sequential redundancy. Consider any arbitrary input sequence I. If a fault is redundant according to Definition 4, then for *every* state S^f of the faulty circuit, there must exist some state S of the fault-free circuit such that the response of the two circuits (starting from S^f and S) to I is identical. Now suppose that we power up the faulty circuit and clock it some number of times, applying arbitrary inputs. After this procedure, the set of possible states that the faulty circuit could be in is usually a proper subset of the set of all states, because values in some of the flip-flops will become correlated. Suppose that we only require that every S^f in this smaller set have a corresponding state S in the fault-free circuit for which the response to I is identical. Under this condition, we can still use the faulty circuit as a replacement for the fault-free circuit provided that we clock the faulty circuit a few times before applying I.

Definition 5: Consider the set of states $\{S_c\}$ reachable after powering up the faulty circuit and applying any sequence of c input vectors. (Note that $\{S_c\}$ shrinks as c increases.) Then a fault is c-cycle redundant if it is not partially testable under the assumption that the initial states of the faulty circuit are restricted to $\{S_c\}$.

Note that a 0-cycle redundant fault would be a redundant fault under Definition 4. Also, a c-cycle redundant fault is a c'-cycle redundant fault for any $c' > c$. A circuit with a c-cycle redundant fault can be simplified by removing the redundant region associated with that fault. A c-cycle redundant fault is not the same as an untestable fault that does not prevent initialization[11]. Removing a c-cycle redundant fault only requires the application of c *arbitrary* input vectors before the existing initialization sequence.

We feel that it is reasonable to remove c-cycle redundant faults since it is generally easy to have the circuit clocked a few times before the initialization sequence is applied. In fact, other sequential optimizations may require the same sort of delay before initialization. For example, in recent work, Singhal *et al.*[24] have shown that forward retiming [25] has this same property. It can also be shown that c-cycle redundancy also has the nice property of being *compositional*[14] [15]. (A property of a subcircuit is compositional if that property still holds when the subcircuit is embedded in any larger circuit.) In contrast, with methods such as[11], it is not safe to remove a fault in a subcircuit without first analyzing the full circuit to see whether the whole system will still have a reset sequence. This is because during initialization some inputs to the subcircuit will come from the larger circuit and some outputs from the subcircuit will be observed by the larger circuit. Thus, checking that the subcircuit still has an initialization sequence is not sufficient.

Example 2: Consider again the circuit in Figure 3 and the fault $c1$ *s-a*-1. Example 1 showed that the fault is untestable and irredundant under Definition 4. Now consider clocking the faulty circuit once (with $a = 0$ or $a = 1$) after powering it up. The application of this arbitrary input vector ensures that the FFs b and c have the same value. The initial state of the faulty circuit ($\{b, c\} = \{1, 0\}$) is no longer available and the faulty circuit cannot be distinguished from the fault-free circuit. Hence $c1$ *s-a*-1 is a 1-cycle redundant fault. □

In the previous example, note that $c1$ *s-a*-1 is not redundant (based on Definition 4) and the circuit cannot be simplified (by removing line c1) based on that definition. But considering it as a 1-cycle redundant fault allows circuit simplification. Hence identification of c-cycle redundancy for redundancy removal seems more practical.

5. Identifying c-Cycle Redundancies

5.1 Sequential Uncontrollability and Unobservability Analysis

Extension of the combinational FIRE algorithm to sequential circuits requires the propagation of uncontrollability and unobservability indicators through FFs. Figure 4 illustrates the propagation rules for FFs. Uncontrollability propagates both ways through a FF. Propagation through a FF means going into an adjacent time frame.

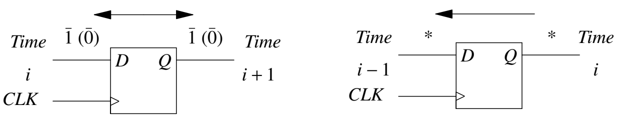

Figure 4. Sequential propagation of uncontrollability & unobservability

Though uncontrollability can propagate forward and backward, unobservability propagates only backward. When a FF output is unobservable, its input is also unobservable. While in a combinational circuit, unobservability propagates unconditionally from the output of a gate to all its inputs, in sequential circuits this is not always the case. Before marking a gate input as unobservable, we ensure that multiple fault-effects from that input in different time frames cannot combine to reach a primary output (PO). This is done to avoid marking faults that would be detected by (sequential) multiple path sensitization. Sequential propagation of unobservability is allowed under the following generalization of the combinational conditions of FIRE.

Unobservability propagates onto l^i (the copy of line l at time i) if

1. The fanouts of l^i are marked as unobservable at time i.
2. For every fanout f^i of l^i, there exists at least one set of lines $\{ p^j \}$, such that
   - f^i is unobservable because of uncontrollability indicators on every line in $\{ p^j \}$; and
   - there is no sequential path from l^k, $i \le k \le j$ to any line in $\{ p^j \}$.

5.2 Rules for Marking Faults

The iterative array model of a sequential circuit consists of repeated time frames of its combinational logic. For a faulty circuit, the fault is present in every time frame. The uncontrollability and unobservability propagations described in the previous section are valid only for the fault-free circuit. In general, they may not be valid in a faulty circuit. To ensure that these propagations will also be valid in a faulty circuit we perform a validation step as follows.

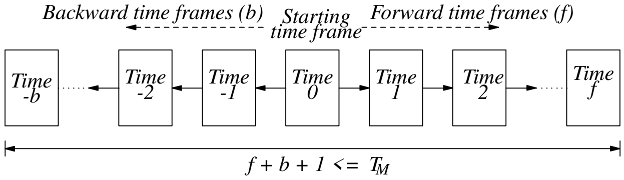

Figure 5. Model for sequential implications

Let time frame 0 represent the time at which the uncontrollability started propagating from some stem s (see Figure 5). The time frames through which uncontrollability and unobservability propagate have positive or negative numbers depending on whether they are forward or backward time frames with respect to the starting time frame. Let f and b be the number of forward and backward time frames respectively. This propagation is allowed for a maximum of T_M time frames ($f + b + 1 \le T_M$). T_M is needed to prevent an infinite propagation of uncontrollability and unobservability (due to feedback loops). However, many propagations terminate before the total number of frames reaches T_M.

Definition 6: A value $\bar{v}$ on a line l^i, $-b \le i \le f$, is *valid* in the presence of a fault *m s-a-u* iff $\bar{v}$ propagates onto l^i in the presence of the faults *m s-a-u* in all time frames $j < i$.

For example, if it was necessary to have $m = \bar{u}$ in a previous time frame to get $l^i = \bar{v}$, then $l^i = \bar{v}$ is invalid in the presence of *m s-a-u*. Note that the validation of an uncontrollability indicator is with respect to a fault; i.e. an uncontrollability may be valid in the presence of one fault and invalid in the presence of another.

For the stem s and the processes $s = \bar{v}$, $v \in \{0, 1\}$ and for each time frame i ($-b \le i \le f$), we form a set of faults S_v^i, as follows

- *Uncontrollable faults:* If a line m has a *valid* $\bar{0}$ ($\bar{1}$) value, at time i, we include *m s-a-*1 (*m s-a-*0) in S_v^i.
- *Unobservable faults:* Let a line m have an unobservability indicator at time i caused by a set of uncontrollability indicators on lines $\{p^j\}$ ($j \ge i$). We include *m s-a-u* ($u \in \{0, 1\}$) in S_v^i, iff the uncontrollability indicator on every line in $\{p^j\}$ is valid in the presence of the fault *m s-a-u*.

We refer to the process of propagating uncontrollability and unobservability to determine the uncontrollable and unobservable faults as *sequential implication*.

5.3 The FIRES Algorithm

FIRES(T_M) **{**

/* $T_M =$ Maximum number of time frames */

For every stem s

Sequentially Imply $s = \bar{0}$ over a maximum of T_M time frames. Let S_0^i represent the sets of the corresponding uncontrollable or unobservable faults.

Sequentially Imply $s = \bar{1}$ over a maximum of T_M time frames. Let S_1^i represent the sets of the corresponding uncontrollable or unobservable faults.

A fault f in any set $S^i = S_0^i \cap S_1^i$ is c_f-cycle redundant.

}

}

Figure 6. Outline of the FIRES algorithm

Figure 6 summarizes the FIRES algorithm. Consider a fault f identified by FIRES in some time frame i. Let l be the leftmost time frame where uncontrollability must propagate for FIRES to identify f as redundant. We use the following rule to associate a c_f with every fault f identified by FIRES.

- $c_f = 0$, if $l \ge i$
- $c_f = i - l$, if $l < i$

It can be shown that a fault f identified by FIRES is c_f-cycle redundant[14]. Clocking the faulty circuit c_f times ensures that the time frames required for the conflict to occur really exist. These rules may overestimate the value of c_f and a more global analysis may be required to determine the minimum c_f.

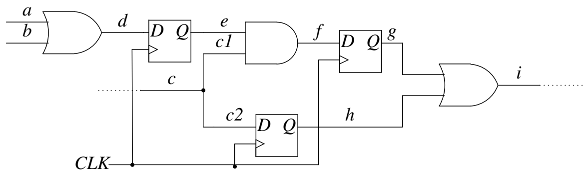

Figure 7. Circuit for Example 3

Table 1. Sequential Implications for Example 3.

Process	Time	Uncont.	Unobs.	Identified Faults
$c = \bar{0}$	0	$c = c1 = c2 = \bar{0}$	$f, e, c1$	$S_0^0 = \{c_1, c1_1, c2_1, f_0, f_1, e_1\}$
	1	$h = i = \bar{0}$	g	$S_0^1 = \{i_1, g_0\}$
	-1	–	d, a, b	$S_0^{-1} = \{d_0, d_1, a_0, b_0\}$
$c = \bar{1}$	0	$c = c1 = c2 = f = \bar{1}$	e	$S_1^0 = \{c_0, c2_0, f_0, e_1\}$
	1	$h = g = i = \bar{1}$	–	$S_1^1 = \{h_0, g_0, i_0\}$
	-1	–	d, a, b	$S_1^{-1} = \{d_0, d_1, a_0, b_0\}$
c-**cycle**	0	$S_0^0 \cap S_1^0 = \{f_0, e_1\}$		
Redundant	1	$S_0^1 \cap S_1^1 = \{g_0\}$		
Faults	-1	$S_0^{-1} \cap S_1^{-1} = \{d_0, d_1, a_0, b_0\}$		

Example 3: Consider the circuit in Figure 7. $c = \bar{0}$ at time 0 implies $c1 = c2 = \bar{0}$ at time 0. This implies $h = i = \bar{0}$ at time 1. This makes line g at time 1 unobservable, which makes lines f, e and $c1$ at time 0 and lines d, a and b at time -1 unobservable as well. A similar analysis with $c = \bar{1}$ shows that the fault effect on e at time 0 and fault effects on d, a and b at time -1 are unobservable. Table 1 summarizes the sequential implications.

When computing S_0 and S_1, all uncontrollability implications remain valid in the presence of the faults found in the corresponding time frames. By the rules of computing c_f, all the faults except g_0 are 0-cycle redundant. Fault g_0 is 1-cycle redundant. □

Among the seven c-cycle redundant faults identified by FIRES in the above example, FUNTEST[19] will report only one (g_0) as untestable, and FUNI[20] will identify two (g_0 and f_0) as untestable by processing the illegal state $\{g, h\} = \{1, 0\}$. The example also shows that FIRES can find faults that are beyond the scope of the combinational ATG theorems[8] [9] (f_0, e_1, d_0, d_1, a_0, and b_0).

6. Results

We used a prototype implementation of FIRES in C++ to identify redundancies in the ISCAS89 sequential benchmark circuits[26]. Our program performs the validation step outlined in Section 5 as an option. It can be shown that all the faults identified by FIRES without the validation step are untestable[14], and those found with the validation step are c-cycle redundant. In general, without validation, FIRES runs faster and finds more faults.

Table 2. Results for benchmark circuits

Circuit	# Fr.	FIRES Without Validation		FIRES With Validation			
		# Unt.	CPU secs.	# Red.	CPU secs.	0-cycle	Max. c
S208	13	60	1.3	57	2.0	0	4
S349	4	2	0.3	2	0.3	2	0
S386	4	27	0.5	27	0.6	0	2
S400	12	1	0.9	1	1.2	0	2
S420	15	217	9.8	208	14.1	0	7
S444	11	11	1.4	11	1.5	11	0
S713	15	32	0.7	32	0.8	32	0
S838	15	529	42.1	508	71.9	0	11
S1238	3	6	2.7	6	2.8	6	0
S1423	10	5	1.4	5	1.5	5	0
S1494	3	1	1.5	1	1.7	1	0
S5378	15	367	52.0	366	69.3	48	11
S9234	15	284	113.2	270	142.8	165	6
S13207	10	893	97.8	893	119.5	67	9
S15850	10	332	312.1	328	434.2	236	9
S35932	5	3984	556.8	3984	684.4	3984	0
S38417	5	147	317.3	147	386.2	115	1
S38584	5	1437	272.0	1437	307.7	1052	3
S499*	15	5	12.2	5	18.6	0	2
S1269*	12	1	0.9	1	1.1	0	2
PROLOG*	5	253	5.9	253	8.7	82	2
S3330*	5	162	6.9	162	8.7	12	1

Table 2 shows the results for the benchmark circuits. *# Fr.* represents the number of time frames which FIRES processed. (The maximum number of time frames is decided depending upon the circuit size, such that $\#Fr. \leq 15$.) *# Unt.* represents the number of untestable faults identified by FIRES without performing the validation step. *# Red.* represents the number of redundant faults identified by FIRES (after validation). *0-cycle* represents the number of 0-cycle redundancies identified (these are combinational, and conventional sequential, redundancies). *Max. c* represents the maximum of the c_f values for the identified faults. All CPU times reported in this section are in seconds for a SUN sparc10. We show only the circuits where redundancies were identified by FIRES. The circuits shown with a "*" were obtained from the ADDENDUM93 directory of the ISCAS89 benchmark set. These circuits, except PROLOG, have no single combinational redundancies.

For example in S420, without validation, FIRES found 217 sequentially untestable faults in 9.8 seconds. 208 out of these 217 faults passed the validation and hence are c-cycle redundant. None of them were 0-cycle and the maximum c was found to be 7. The run with the validation step took about 14 seconds.

The distribution of the number of c-cycle redundancies found for different values of c varies widely from circuit to circuit and can be found in[14].

Unfortunately, we cannot make a fair comparison of our results with any of the previously published ones. The results of[4] [5] [6] [7] [11] are for redundancy removal, those of[5] [6] [12] [13] are based on incorrect theoretical results, those of[7] are not relevant for the original benchmark circuits without a global reset, those of[8] [9] are only for untestability, those of[16] deal with a restricted definition of redundancy, those of[21] are for sequential optimization and include only the smaller benchmark circuits, and those of[10] and[11] do not deal with benchmark circuits.

A deterministic sequential test generator performing exhaustive search can identify all faults found by FIRES as untestable, if given enough time. However, the test generator cannot identify these faults as redundant. We now show that state-of-the-art sequential test generators can have difficulty even in proving that the faults found by FIRES are untestable. We used two sequential test generators, GENTEST[27] and HITEC[28]. In this experiment, the faults found by FIRES (without validation) were passed as the only targets to the test generators. For S5378, GENTEST, allowed to spend up to 100 seconds per fault, used 31 times more CPU secs. than FIRES and aborted 95% of the untestable faults found by FIRES (Table 3).

Table 3. Comparison of FIRES with GENTEST for S5378

Circuit Name	FIRES		GENTEST			Speed-up Ratio
	# Unt.	CPU secs.	# Unt.	# Abo.	CPU secs.	
S5378	367	52.0	19	348	1599.7	31

Similarly, HITEC, allowed to spend up to 20 seconds per fault, could prove untestability only for 52% of the faults found by FIRES in S838 and used 162 times more CPU secs. (Table 4). These examples show that the exhaustive search done by the sequential test generators could not identify as untestable faults found by FIRES without any search.

Table 4. Comparison of FIRES with HITEC for S838

Circuit Name	FIRES		HITEC			Speed-up Ratio
	# Unt.	CPU secs.	# Unt.	# Abo.	CPU secs.	
S838	529	42.1	276	253	6136.8	162

7. Conclusions

This paper has presented FIRES, a new fault-independent redundancy identification algorithm for sequential circuits that is based on conflict analysis. We have introduced the concept of a c-cycle redundant fault, where the behavior of the fault-free circuit is indistinguishable from that of the faulty circuit which has been clocked for at least c times after power-up. The conventional notion of redundancy is a special case of a c-cycle redundancy with $c = 0$. We have shown that a fault whose detection requires a conflict as a necessary condition is c-cycle redundant. FIRES does not assume a global reset mechanism and does not require state transition graph information.

-- --

Any practical state-of-the-art sequential test generator cannot distinguish between untestable and redundant faults. All existing solutions for sequential RID are either based on incorrect theoretical results, or make simplifying assumptions or are limited to small circuits. Hence FIRES provides the *first general solution to the sequential RID problem, practical for large circuits*. FIRES, however, finds only a subset of the redundant faults.

FIRES can be easily integrated with any synthesis system to remove redundancies. Apart from being able to identify sequential redundancy in a practical manner, FIRES has significant advantages in redundancy removal applications because it is fault-independent and the average CPU time it takes to process the entire circuit is negligible. After a redundancy is removed, new redundancies may be created in the circuit. Even combinational redundancy removal using ATG is expensive because many faults may have to be retargeted over multiple passes of test generation to identify newly created redundancies. However, FIRES may at most have to reanalyze previously analyzed stems in such an iterative procedure.

A deterministic sequential test generator performing exhaustive search can only prove untestability for the faults found by FIRES (without search). By processing a single conflict, FIRES may simultaneously identify several redundant faults that would require separate targeting by an ATG-based approach. In general, the faults found by FIRES are not easy targets for state-of-the-art practical sequential test generators and the FIRES CPU times are negligible compared to the complete sequential test generation runs for all faults in the circuit. Thus, FIRES can be used as a preprocessor to any state-of-the-art sequential test generator to obtain significant savings in computation time and to increase the detectable fault coverage.

Acknowledgement

Our special thanks to Prof. I. Pomeranz and Prof. S. M. Reddy for pointing out an error in our initial theory and for many useful discussions and comments. We thank AT&T Bell Laboratories and the ECE Department of the Illinois Institute of Technology for supporting this research. The support provided by S. Davidson, A. Dunlop and S. Wu made this work possible and is gratefully acknowledged.

REFERENCES

[1] M. Abramovici and M. A. Breuer, “On Redundancy and Fault Detection in Sequential Circuits,” *IEEE Trans. on Computers*, vol. C-28, pp. 864-865, Nov. 1979.

[2] A. D. Friedman, “Fault Detection in Redundant Circuits,” *IEEE Trans. on Electronic Computers*, vol. EC-16, pp. 99-100, Feb. 1967.

[3] S. Davidson, “Is I_{DDQ} Yield Loss Inevitable ?” *Proc. Intn’l. Test Conf.*, pp. 572-579, Oct. 1994.

[4] K. T. Cheng, “On Removing Redundancy in Sequential Circuits,” *Proc. 28th. Design Automation Conf.*, pp. 164-169, June 1991.

[5] K. T. Cheng, “An ATPG-Based Approach to Sequential Logic Optimization,” *Proc. Intn’l. Conf. on CAD*, pp. 372-375, 1991.

[6] K. T. Cheng, “Redundancy Removal for Sequential Circuits Without Reset States,” *IEEE Trans. on CAD*, vol. 12, no. 1, pp. 13-24, Jan. 1993.

[7] H. Cho, G. D. Hachtel and F. Somenzi, “Redundancy Identification/Removal and Test Generation for Sequential Circuits Using Implicit State Enumeration,” *IEEE Trans. on CAD*, vol. 12, no. 7, pp. 935-945, July 1993.

[8] V. D. Agrawal and S. T. Chakradhar, “Combinational ATPG Theorems for Identifying Untestable Faults in Sequential Circuits,” *Proc. European Test Conf.*, pp. 249-253, April 1993.

[9] V. D. Agrawal and S. T. Chakradhar, “Combinational ATPG Theorems for Identifying Untestable Faults in Sequential Circuits,” *IEEE Trans. on CAD*, vol. 14, no. 9, pp. 1155-1160, Sept. 1995.

[10] I. Pomeranz and S. M. Reddy, “On Identifying Undetectable and Redundant Faults in Synchronous Sequential Circuits,” *12th. IEEE VLSI Test Symp.*, pp. 8-14, April 1994.

[11] I. Pomeranz and S. M. Reddy, “On Achieving Complete Testability of Synchronous Sequential Circuits with Synchronizing Sequences,” *Proc. Intn’l. Test Conf.*, pp. 1007-1016, Oct. 1994.

[12] L. A. Entrena and K. T. Cheng, “Sequential Logic Optimization By Redundancy Addition and Removal,” *Proc. Intn’l. Conf. on CAD*, pp. 310-315, Nov. 1993.

[13] L. A. Entrena and K. T. Cheng, “Combinational and Sequential Logic Optimization By Redundancy Addition and Removal,” *IEEE Trans. on CAD*, vol. 14, no. 7, pp. 909-916, July 1995.

[14] M. A. Iyer, “On Redundancy and Untestability in Sequential Circuits,” *Ph.D. Thesis, ECE Department, Illinois Institute of Technology*, Chicago, IL - 60616, July 1995.

[15] M. A. Iyer, D. E. Long, and M. Abramovici, “Surprises in Sequential Redundancy Identification,” *Proc. European Design and Test Conference*, March 1996.

[16] J. Moondanos and J. A. Abraham, “Sequential Redundancy Identification using Verification Techniques,” *Proc. Intn’l. Test Conf.*, pp. 197-205, Sept. 1992.

[17] M. A. Iyer and M. Abramovici, “Low-Cost Redundancy Identification for Combinational Circuits,” *Proc. 7th. Intn’l. Conf. on VLSI Design, India*, pp. 315-318, Jan. 1994.

[18] M. A. Iyer and M. Abramovici, “FIRE: A Fault-Independent Combinational Redundancy Identification Algorithm,” *IEEE Trans. on VLSI Systems*, June 1996 (to appear).

[19] M. A. Iyer and M. Abramovici, “Sequentially Untestable Faults Identified Without Search (Simple Implications Beat Exhaustive Search!),” *Proc. Intn’l. Test Conf.*, pp. 259-266, Oct. 1994.

[20] D. E. Long, M. A. Iyer and M. Abramovici, “Identifying Sequentially Untestable Faults Using Illegal States,” *13th. IEEE VLSI Test Symp.*, pp. 4-11, May 1995.

[21] V. Singhal, C. Pixley, A. Aziz and R. K. Brayton, “Exploiting Power-up Delay for Sequential Optimization,” *Proc. European Design Automation Conf.*, pp. 54-59, Sept. 1995.

[22] M. Abramovici, J.J. Kulikowski, and R.K. Roy, “The Best Flip-Flops to Scan,” *Proc. Intn’l. Test Conf.*, pp. 166-173, Oct. 1991.

[23] I. Pomeranz and S. M. Reddy, “Classification of Faults in Synchronous Sequential Circuits,” *IEEE Trans. on Computers*, vol. 42, pp. 1066-1077, Sept. 1993.

[24] V. Singhal, C. Pixley, R. L. Rudell and R. K. Brayton, “The Validity of Retiming Sequential Circuits,” *Proc. Design Automation Conf.*, pp. 316-321, June 1995.

[25] C. E. Leiserson and J. B. Saxe, “Retiming Synchronous Circuitry,” *Algorithmica*, vol. 6, pp. 5-35, 1991.

[26] F. Brglez, D. Bryan, and K. Kozminski, “Combinational Profiles of Sequential Benchmark Circuits,” *Proc. 1989 Intn’l Symp. on Circuits and Systems,* pp. 1929-1934, May 1989.

[27] T. J. Chakraborty, S. Davidson, and B. Bencivenga, “GENTEST: The Architecture of Sequential Circuit Test Generator,” *Proc. Custom Integrated Circuits Conf.,* May 1991.

[28] T. Niermann and J. H. Patel, “HITEC: A Test Generation Package for Sequential Circuits,” *Proc. European Conf. on Design Automation,* pp. 214-218, 1991.