Coverage-based Trace Signal Selection for Fault Localisation in Post-Silicon Validation

Charlie Shucheng Zhu[1], Georg Weissenbacher[2], and Sharad Malik[1]

[1] Princeton University
[2] Vienna University of Technology, Austria

Abstract. Post-silicon validation is the time-consuming process of detecting and diagnosing defects in prototype silicon. It targets electrical and functional defects that escaped detection during pre-silicon verification. While the at-speed execution of test scenarios facilitates a higher test coverage than pre-silicon simulation, this comes at the cost of limited observability of signals in the integrated circuit. This limitation complicates the localisation of the cause underlying a defect. Trace buffers, designed to store a limited execution history, partially alleviate but do not entirely remedy the problem. Since trace buffers typically record only a small fraction of the system state over at most a few thousand cycles, their utility is contingent on the cautious selection of traced signals.
This paper presents a technique for the automated selection of trace signals. While the aim of existing selection strategies is typically to enable the (early) detection of defects or to maximise the recoverable state information, our objective is to facilitate the subsequent automated localisation of faults using consistency-based diagnosis. To this end, we use integer linear programming and automated test pattern generation to identify a subset of state signals through which potential failures are likely to propagate. We demonstrate that our technique complements our previous work on SAT-based fault localisation using backbones. In that context, we evaluate the utility of our results on two OpenCores designs. We show that for this purpose, our technique generates a better selection of trace signals than a related approach recently presented by Yang and Touba.

1 Introduction

Post-Silicon validation deals with debugging early silicon prototypes with the goal of detecting and diagnosing design faults. These faults may be functional, i.e. logical bugs, or electrical, i.e. faults in the circuit design. Electrical faults tend to be trickier to detect since these may be triggered only under very specific conditions and thus this behaviour may not be easily repeatable. In comparison to pre-silicon validation using simulation and formal verification, post-silicon validation is no longer limited by slow software models, but rather can run substantially large traces at speed. However, unlike these software models, there is very limited signal observability. The observability at the chip outputs is typically enhanced by adding additional state to the chip, referred to as trace buffers,

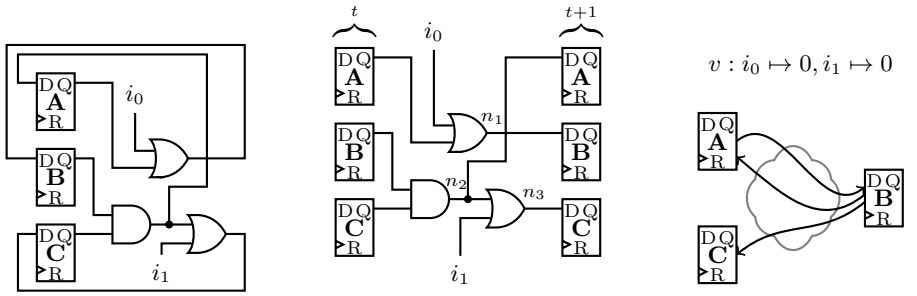
which buffer the values of a small set of carefully selected signals, referred to as trace signals, typically for a few thousand cycles. These buffered values are then used to both detect and diagnose/localise faults. Since, the number of trace signals needs to be small to keep the trace buffer overhead low, these need to be carefully selected. This is generally done manually using key designer insight. While this may be justified for high volume parts such as processors, automation of this step is highly desirable for application to a broad range of designs. This paper addresses the problem of automatically selecting the set of trace signals for their application in aiding fault diagnosis in post-silicon validation. We present a coverage-based algorithm for this. The algorithm takes as input, the design, the set of possible faults, the set of candidate trace signals and a set of test vectors. In this paper we limit the candidate set of trace signals to be the existing state bits in the design, though that is not a requirement of our approach. The algorithm first determines, for each fault and the set of test vectors, the set of candidate signals that the fault-effect, i.e. error, propagates to. It then selects a subset of candidates that maximally covers, i.e. detects, the fault set. This coverage problem is naturally framed as an integer linear programming (ILP) problem. This formulation is related to recent work done in trace signal selection by Yang and Touba [19]. However, the formulation in that paper is geared towards error detection and not fault diagnosis. For a given erroneous trace, the problem of fault diagnosis or localisation deals with identifying which gate had the fault and which cycle the fault was activated. In our recent work [21] we presented an algorithm for fault diagnosis that uses trace buffers. However, the focus of that paper was not on the trace signals, and thus these were arbitrarily selected in that work. The work in this paper is complementary in that it provides a systematic way to select trace signals. In our experimental evaluation in this paper we show that this coverage based trace selection compares favorably with the arbitrary selection in fault diagnosis. We also show that it compares favourably with the error detection based selection [19] by Yang and Touba applied to fault diagnosis. This evaluation is done using two microcontroller designs from OpenCores.

This paper is organized as follows. § 2 covers the background and related work. The technical contributions are presented in § 3 and the experimental evaluation in § 4. Finally § 5 provides some concluding remarks.

2 Background and Related Work

2.1 Automatic test pattern generation

Automatic test pattern generation (ATPG) is concerned with the construction of test scenarios that make manufacturing faults surface if present (for a tutorial, see [5]). The approach is typically based on simple gate-level fault models, the most popular of which is the *single stuck-at fault* model, in which the output of a single gate is permanently stuck at a fixed logic value (0 or 1). For such a fault to become *observable*, its effect (i.e., the incorrect output signal of the gate) has to *propagate* through the circuit (along a *sensitised* path) to one of the primary



(a) A sequential circuit (b) Execution of one cycle (c) Error propagation

Fig. 1: Error propagation in a simple sequential circuit

outputs or to an observable latch. This may not happen with each trace, since the erroneous signal might be masked by other signals.

Example 1. Consider the propagation of errors in the sequential circuit in Figure 1a. The simple circuit comprises three latches (labelled **A**, **B**, and **C**) and a combinational part with two input signals i_0 and i_1. For the sake of simplicity, we omit the primary output signals and assume that latch **C** is observable. Assume that the output of the AND gate in Figure 1a is permanently stuck-at 1, leading to an erroneous result in case the values stored in the latches **B** and **C** are 0 and 1, respectively. For this error to propagate to the latch **C** in the current execution cycle, the input signal i_1 needs to be 0.

The aim of ATPG is to automatically generate input patterns that result in the activation and propagation of faults. In order to trigger the stuck-at 1 fault in Example 1, one of the latches **B** or **C** must hold the value 0. For the fault to propagate to latch **C** through the subsequent OR gate, it is necessary that i_1 is 0. Accordingly, we require input signals that result in different logic values for at least one of the observable signals of the original and the faulty circuit.

Test pattern generation for digital circuits can be formulated as a Boolean satisfiability problem (c.f. [5, §22.2.3]), which can be solved using efficient satisfiability checkers (e.g., [14, 9]). For combinational circuits, this approach is illustrated in Figure 2. By using an XOR gate (or *miter*) to combine the outputs of the original circuit and a duplicate circuit into which a stuck-at fault has been injected, we obtain a new circuit whose output is one if and only if the values of the latches and input signals are chosen such that the fault is activated and propagates to an observable output. Using a satisfiability checker and a propositional encoding of the resulting circuit, we can derive appropriate logic values.

A similar technique can be applied for sequential circuits. In this setting, it is sufficient if the error propagates to an observable output or latch after several execution cycles. While the error in Example 1 may not propagate to

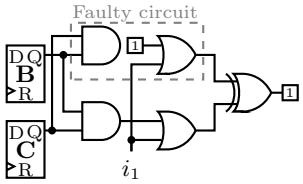


Fig. 2: ATPG as Boolean satisfiability problem

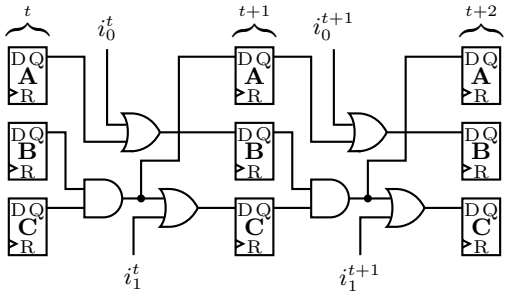


Fig. 3: Execution of two cycles

latch **C** immediately, it does propagate to latch **A**, from where (in a favourable test scenario) it may propagate to latch **B** and eventually latch **C** in subsequent execution cycles. In Boolean satisfiability-based ATPG, this is taken into account by *unwinding* the circuit into an *iterative logic array* (ILA) [1], as shown in Figure 3 for two execution cycles. By encoding a sequence of execution cycles into a propositional formula, it is possible to obtain a multi-cycle test scenario in which the fault is activated and propagates. We refer to this approach as *sequential* ATPG.

Related Work. Mutation testing is a technique related to ATPG that is applied in software testing. The test-case generation technique for Simulink programs presented in [2], for instance, resembles the ATPG approach described above in that it uses fault models (such as stuck-at-faults). The aim of mutation testing, however, is typically to evaluate or increase the coverage of a test suite. Moreover, unlike ATPG, mutation testing for software programs is typically based on syntactic modifications (*mutations*, respectively) of the source code rather than on fault models.

2.2 Trace Signal Selection Using Integer Linear Programming

In an integrated circuit of realistic dimensions only a fraction the system state (stored the in latches) can be recorded in a trace buffer. In Example 1, for instance, we assume that the trace buffer maintains a (limited) history of the

	Fault-free		Fault in **A**		Fault in **B**		Fault in **C**	
	t	$t+1$	t	$t+1$	t	$t+1$	t	$t+1$
A	0	0	1	0	0	**1**	0	0
B	0	0	0	**1**	1	0	0	0
C	1	0	1	0	1	**1**	0	0
Functional vector $v : i_0 \mapsto 0, i_1 \mapsto 0$								

Table 1: Fault-free and erroneous executions of the circuit in Figure 1

logic values of latch **C**. The remaining latches are effectively *unobservable*. Consequently, only errors that eventually propagate to an observable output (or latch) can be detected. Accordingly, whether (and when) an error is caught is contingent on the selection of the trace signals (as well as on the test scenario).

Yang and Touba [19] propose a technique to automatically select trace signals based on the propagation of errors between latches. The approach is based on the following insight: an error that propagates from the faulty gate to a latch may keep propagating over multiple cycles (depending on the test pattern that is applied) until it eventually corrupts an observable signal.

The authors of [19] construct an *error transmission matrix* which holds, for a fixed set of single-cycle test patterns, the information between which latches errors may propagate. The matrix is then transformed into an *integer linear programming* (ILP) problem whose optimal solution identifies a set of latches which capture as many errors propagated from latches in as many test scenarios as possible.

Example 2. We continue working in the setting of Example 1. Figure 1b illustrates one execution cycle in form of an ILA. The latches on the left side represent the state of the circuit in time-frame t, the latches to the right represent the subsequent time-frame $t + 1$.

Table 1 shows four single-cycle executions of the circuit in Figure 1a, starting from a state in which the latches **A** and **B** hold the value 0, and latch **C** holds the value 1. The input test vector is the same in all executions ($i_0 \mapsto 0, i_1 \mapsto 0$). The first execution is fault-free, whereas we introduced transient errors by flipping the values of the latches **A**, **B**, and **C**, respectively, in the remaining three executions. Each of these errors represents a gate-level fault that propagated to the respective latch, whose bits are highlighted in bold in the table. Table 1 illustrates that in the given scenario, an error in **A** propagates to **B**, and an error in **B** propagates to **A** as well as to **C**. The error introduced in **C** does not propagate, since it is masked by the value of **B**. The error propagation between the three latches in this situation is indicated in Figure 1c.

Following the methodology presented by Yang and Touba [19], we obtain the error transmission matrix in Figure 4a. Additional test patterns can be encoded in the transmission matrix by adding more rows. For clarity, we omit the optimisation step described in [19] which reduces the size of the matrix by grouping together *independent* latches whose information can be compressed.

$$\begin{array}{c} \\ (\mathbf{A}, v) \\ (\mathbf{B}, v) \\ (\mathbf{C}, v) \end{array} \begin{array}{c} \mathbf{A}\ \mathbf{B}\ \mathbf{C} \\ \left[\begin{array}{ccc} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{array}\right] \end{array}$$

(a) Transmission matrix

$$\begin{aligned} \text{max: } & \textstyle\sum_{i=0}^{2} R_i \\ S_{\mathbf{B}} &\geq R_0 \\ S_{\mathbf{A}} + S_{\mathbf{C}} &\geq R_1 \\ 0 &\geq R_2 \\ R_0, R_1, R_2 &\in \{0,1\} \\ S_{\mathbf{A}}, S_{\mathbf{B}}, S_{\mathbf{C}} &\in \{0,1\} \\ S_{\mathbf{A}} + S_{\mathbf{B}} + S_{\mathbf{C}} &= 1 \end{aligned}$$

(b) ILP problem

Fig. 4: Selecting trace signals using integer linear programming

The ILP problem obtained from the transmission matrix in Figure 4a in order to select *one* signal to trace is shown in Figure 4b. Each R_i ($i \in \{0, 1, 2\}$) represents a row, and a value of 1 indicates that the corresponding error propagates to a selected latch in the respective test scenario. Consequently, the objective is to maximise the sum $R_0 + R_1 + R_2$. Whether an error is captured in a latch depends on the latches that are traced. In our example, we restrict the trace buffer to only one latch; accordingly, $S_{\mathbf{A}} + S_{\mathbf{B}} + S_{\mathbf{C}} = 1$. Finally, each row in the transmission matrix determines which errors can be captured. The line $S_{\mathbf{A}} + S_{\mathbf{C}} \geq R_1$, for instance, encodes that an error in latch **B** can be captured by either latch **A** or latch **C** in the test scenario v.

Note that this ILP problem does not have a unique optimal solution: assigning 1 to either $S_{\mathbf{A}}$, $S_{\mathbf{B}}$, or $S_{\mathbf{C}}$ maximises the objective. A solution $S_{\mathbf{A}} = 1$, $S_{\mathbf{B}} = 0$, and $S_{\mathbf{C}} = 0$ returned by the ILP solver indicates that we should trace the value in latch **A**. No matter which latch we choose, according to Table 1 we will only be able to track either the error in **A**, or the error in **B** (since faults of **C** do not propagate in this setting).

Related Work. Hung and Wilton [11] base their signal selection algorithm on the expected number of reachable system states that can be "ruled out" by observing these signals. This approach relies on a computationally expensive approximation of the reachable state space.

Yang et al. [20] propose to use unsatisfiable cores obtained from a test scenario that results in a failure and a propositional encoding of the circuit to identify signals that are relevant to the analysis of the failure. Moreover, they propose a SAT-based technique to select trace signals from which relevant signals that cannot be observed can be reconstructed.

Prabhakar and Hsiao [15] use a multiplexed trace signal scheme which enables them to effectively double the number of signals that can be traced. Moreover, the paper proposes a technique to identify signals that can be inferred from traced signals using logical implication and therefore need not be recorded.

Paula et al. [7] proposes to compute *signatures* of states to narrow down the set of predecessor states of the crash state, effectively enabling backwards state stepping. This allows them to identify the error in an earlier cycle in a subsequent test run. A follow-up paper [8] describes how repeated test runs can be used to arbitrarily increase the number of execution cycles which can be recorded by a trace buffer. Both techniques require that the erroneous behaviour can be reproduced repeatedly.

A detailed discussion of techniques that use compression techniques to increase observability is provided by [19].

2.3 SAT-based Fault Localisation

The objective of the trace signal selection algorithm in [19] is to detect as many errors as early as possible. Detecting an error, however, is often the easy part of the post-silicon validation phase. Due to the observability limitations in integrated circuits, *locating* the cause of the error can be a formidable challenge.

Consistency-based diagnosis [16] is a technique that aims at locating the cause of an observed error by identifying fault candidates based on the *golden model* of a system and observations of its actual implementation. It relies on automated reasoning to identify the smallest set of components that explains the inconsistency between the hardware design and the behaviour of the manufactured prototype. The technique has seen a recent spike in popularity (e.g., [17, 18, 4, 3, 21]) due to the improved scalability of satisfiability solvers. The following example illustrates the idea underlying consistency-based diagnosis.

Example 3. Recall the setting from Example 1, in which we postulated a stuck-at 1 fault for the AND-gate in the circuit in Figure 1a. The ILA in Figure 1b which represents one execution cycle of this circuit can be encoded as a propositional formula in which $\mathbf{A}^t$, $\mathbf{B}^t$, $\mathbf{C}^t$ and $\mathbf{A}^{t+1}$, $\mathbf{B}^{t+1}$, $\mathbf{C}^{t+1}$ refer to the values held by the latches in time-frames t and $t+1$, respectively:

$$(\mathbf{A}^{t+1} = \mathbf{B}^t \cdot \mathbf{C}^t) \quad \cdot \quad (\mathbf{B}^{t+1} = \mathbf{A}^t + i_0) \quad \cdot \quad (\mathbf{C}^{t+1} = \mathbf{A}^{t+1} + i_1) \tag{1}$$

As a result of the faulty AND gate, the logic values in $\mathbf{A}^{t+1}$ and $\mathbf{C}^{t+1}$ are corrupted during the execution of the manufactured chip. This fact as well as the initial state and the input values are encoded in the following propositional formula:

$$\begin{pmatrix} (\mathbf{A}^t = 0) \ \cdot \ (\mathbf{B}^t = 0) \ \cdot \ (\mathbf{C}^t = 1) \cdot \\ (\mathbf{A}^{t+1} = 1) \cdot (\mathbf{B}^{t+1} = 0) \cdot (\mathbf{C}^{t+1} = 1) \end{pmatrix} \cdot (i_0 = 0) \cdot (i_1 = 0) \tag{2}$$

Due to the discrepancy between the *golden model* in Figure 1a and the behaviour of the manufactured prototype the conjunction of the formulae 1 and 2 is unsatisfiable. In order to determine the cause of the discrepancy, we can use a *partial maximum-satisfiability* (Max-Sat) solver (see [10], for instance) to identify a minimal set of conjuncts of Formula 1 that are responsible for the inconsistency of Formula 1 and Formula 2. In our example, dropping the constraint $(\mathbf{A}^{t+1} = \mathbf{B}^t \cdot \mathbf{C}^t)$ makes the formula satisfiable, which indicates that a faulty AND gate in Figure 1b is a possible explanation for the inconsistency.

In Example 3 we assume that all latches are observable. While this is a valid assumption in the context of *pre-silicon* debugging, where all signal values can be determined by means of simulation, this information is typically not available in the post-silicon setting. In this setting, the approach described in Example 3 may fail: eliminating the information about $\mathbf{B}^t$, $\mathbf{C}^t$, $\mathbf{B}^{t+1}$, and $\mathbf{C}^{t+1}$ from Formula 2 makes the conjunction of the formulae 1 and 2 satisfiable.

This problem can be addressed by means of unwinding the sequential circuit sufficiently often and constraining the resulting ILA with the information collected in the trace buffer.

Example 4. The two-cycle ILA in Figure 3 can be translated into the following propositional formula:

$$\begin{array}{c}(\mathbf{A}^{t+1} = \mathbf{B}^t \cdot \mathbf{C}^t) \quad \cdot \quad (\mathbf{B}^{t+1} = \mathbf{A}^t + i_0^t) \quad \cdot \quad (\mathbf{C}^{t+1} = \mathbf{A}^{t+1} + i_1^t) \cdot \\ (\mathbf{A}^{t+2} = \mathbf{B}^{t+1} \cdot \mathbf{C}^{t+1}) \cdot (\mathbf{B}^{t+2} = \mathbf{A}^{t+1} + i_0^{t+1}) \cdot (\mathbf{C}^{t+2} = \mathbf{A}^{t+2} + i_1^{t+1})\end{array} \tag{3}$$

Assume that the trace buffer recorded the information $(\mathbf{A}^t = 0)$, $(\mathbf{A}^{t+1} = 1)$, and $(\mathbf{A}^{t+2} = 1)$. Constraining Formula 3 with the information obtained from the trace buffer and the test pattern $i_0^t = 0$, $i_1^t = 0$, $i_0^{t+1} = 0$, and $i_1^{t+1} = 0$ results in an unsatisfiable SAT instance. A subsequent analysis yields that dropping either $(\mathbf{B}^{t+1} = \mathbf{A}^t + i_0^t)$ or $(\mathbf{A}^{t+2} = \mathbf{B}^{t+1} \cdot \mathbf{C}^{t+1})$ makes the instance satisfiable and identifies either the OR gate in time-frame t or the AND gate in time-frame $t+1$ as potential culprits.

Example 4 shows that a consistency-based diagnosis may report more than one fault candidate. In general, this problem cannot be avoided (even if all latches are observable), since both gates are valid fault candidates. Note, however, that the approach identified an exact time-frame in which the respective components may have failed, making it suitable for the analysis of intermittent or transient faults.

For large circuits, the number of execution cycles that can be analysed is limited by the scalability of the underlying logic solver. While in theory it is always sufficient to analyse the *entire* execution, in practice the size of the resulting propositional formula would likely be prohibitive. This problem is addressed in [12] and [21] by *sliding* a *window* of fixed size (backwards) along the execution trace, thus partitioning the execution trace into ILAs of fixed size. The technique presented in [12] targets design debugging and requires full observability to compute Craig interpolants [6], which are used to propagate information across windows. Zhu et al. [21] is aimed at post-silicon validation and relies on *backbones* (see, e.g., [13]) to propagate state information across windows. The backbone of a satisfiable propositional formula comprises all literals which take the *same* value in *all* satisfying assignments of the formula.

Example 5. We continue working in the setting of Example 4. Assume that the scalability of the solver limits the consistency-based analysis technique to windows of size one. As previously established, the information $(\mathbf{A}^{t+1} = 1)$, $(\mathbf{A}^{t+2} = 1)$ and $i_0^{t+1} = 0$, $i_1^{t+1} = 0$ is insufficient to yield an inconsistency in

	Fault-free			Fault in **A**		
	t	$t+1$	$t+2$	t	$t+1$	$t+2$
A	0	0	0	1	0	**1**
B	0	0	0	0	**1**	0
C	1	1	0	1	1	**1**
$i_0^t = 0$, $i_1^t = 1$, $i_0^{t+1} = 0$, $i_1^{t+1} = 0$						

Table 2: Fault-free and erroneous 2-cycle executions of the circuit in Figure 1

time-frame $t+1$ of Figure 3. However, from ($\mathbf{A}^{t+2} = 1$) and ($\mathbf{A}^{t+2} = \mathbf{B}^{t+1} \cdot \mathbf{C}^{t+1}$) (c.f. Formula 3) we can derive the *backbone* $\mathbf{B}^{t+1} = 1$ and $\mathbf{C}^{t+1} = 1$, which is inconsistent with ($\mathbf{A}^t = 0$), $i_0^t = 0$, and time-frame t in Figure 3, resulting in the fault candidate ($\mathbf{B}^{t+1} = \mathbf{A}^t + i_0^t$).

Similarly, from ($\mathbf{A}^t = 0$), $i_0^t = 0$, and time-frame t in Figure 3 we can derive the backbone $\mathbf{B}^{t+1} = 0$, which is inconsistent with $\mathbf{A}^{t+2} = 1$ and time-frame $t + 1$. Accordingly, the analysis yields the AND gate corresponding to $\mathbf{A}^{t+2} = \mathbf{B}^{t+1} \cdot \mathbf{C}^{t+1}$ as a fault candidate.

Related Work. As previously mentioned, there are a number of papers that apply consistency-based diagnosis to address *pre-silicon* debugging (with full observability) by constraining a *faulty* RTL model with *correct* input/output pairs (given as a specification) [17, 18, 4, 3].

3 Improving Coverage-based Trace Signal Selection

In this section, we propose two improvements over the ILP-based signal selection approach of Yang and Touba [19]. Our modifications to the algorithm address the following limitations:

- The approach outlined in §2.2 does not directly take advantage of the *transitivity* of error propagation. As pointed out at the end of §2.1, an error may propagate through non-observable latches for several execution cycles until it corrupts a latch monitored by the trace buffer.
- The fault model of [19] is applied exclusively to latches. Depending on the structure of the circuit, however, some latches may have a higher probability of being corrupted/propagated by gate-level faults than others and thus also may be more useful for fault localisation.

3.1 Multi-cycle Coverage

The following example illustrates the limitation of a trace signal selection algorithm that is based on a propagation depth of one.

Example 6. Table 2 shows a correct and an erroneous execution of the sequential circuit in Figure 1a. As in Example 2, we introduce a transient error by flipping

the value of one latch in the execution. Unlike in Example 2, however, the executions in Table 2 have two cycles. As mentioned previously, an error in latch **A** propagates to latch **B** within one cycle. After an additional execution cycle, however, the error corrupts both latch **A** and latch **C**. If we add this information to the error transmission matrix and the ILP encoding in Figure 4, we are able to derive that observing latch **A** or latch **C**, but not latch **B**, is the optimal solution.

3.2 Injecting Faults in Combinational Logic

The starting point of the analysis in [19] is that a gate-level fault has already propagated to a latch. Depending on the structure of the circuit, however, certain latches might be more susceptible to capturing an erroneous signal originating in the combinational logic and thus also have greater value in fault localisation.

Example 7. In the sequential circuit in Figure 1a, a fault in the AND gate may propagate to latches **A** and **C**. A fault in the OR gates may propagate to latch **B** and latch **C**, respectively. Accordingly, if all gates are equally likely to fail, then latch **C** has a higher probability of being corrupted.

Motivated by the concerns discussed in § 3.1 and § 3.2, the following section describes our modifications to the approach of [19].

3.3 Integer Linear Programming Encoding

To take the structure of the circuit into account (see §3.2), we inject faults in the gates of the combinational part of the circuit *as well as* in the latches. A set of fault simulations are applied on each injected fault repeatedly. Unlike the method proposed in [19], the test patterns used for fault simulation are multi-cycle and generated by sequential ATPG (as described in §2.1). ATPG helps to provide tests which result in the activation and the propagation of the fault to a latch (or primary output). From the fault simulations, we determine the set of latches to which the fault can propagate and add a row to the error transmission matrix accordingly. The corresponding ILP problem can then be built as described in §2.2.

Example 8. We continue working on the setting from Example 7. Suppose we have obtained a 2-cycle test pattern from ATPG. For simplicity, the same test pattern as in Example 6 are adopted. Transient faults are injected in both latches and gates. Table 3 shows the correct and erroneous execution of all injected faults. With the same approach as §2.2, we can obtain the equivalent ILP problem in Figure 5. For simplicity, we assume that trace signals can be observed with equal costs on all latches in this paper. However, in reality, some traced signals may be observed less expensively than others. To put a bias on those latches, their corresponding rows in the transmission matrix can be duplicated intentionally. Duplicating rows in a transmission matrix is acceptable, because ILP is not the bottleneck of scalability in our work. More details will be given at the end of §4.2.

	Fault-free			Fault in **A**			Fault in **B**			Fault in **C**		
	t	$t+1$	$t+2$	t	$t+1$	$t+2$	t	$t+1$	$t+2$	t	$t+1$	$t+2$
A	0	0	0	1	0	**1**	0	**1**	0	0	0	0
B	0	0	0	0	**1**	0	1	0	**1**	0	0	0
C	1	1	0	1	1	**1**	1	1	0	0	1	0
$i_0^t = 0$, $i_1^t = 1$, $i_0^{t+1} = 0$, $i_1^{t+1} = 0$												
	Fault-free			Fault in n_1			Fault in n_2			Fault in n_3		
	t	$t+1$	$t+2$	t	$t+1$	$t+2$	t	$t+1$	$t+2$	t	$t+1$	$t+2$
A	0	0	0	0	0	**1**	0	**1**	0	0	0	0
B	0	0	0	0	**1**	0	0	0	**1**	0	0	0
C	1	1	0	1	1	**1**	1	**1**	0	1	**0**	0
$i_0^t = 0$, $i_1^t = 1$, $i_0^{t+1} = 0$, $i_1^{t+1} = 0$												

Table 3: 2-cycle execution of faults in both latches and internal gates

$$
\begin{array}{c}
\\
(\mathbf{A}, v) \\
(\mathbf{B}, v) \\
(\mathbf{C}, v) \\
(\mathbf{n_1}, v) \\
(\mathbf{n_2}, v) \\
(\mathbf{n_3}, v)
\end{array}
\begin{array}{c}
\mathbf{A}\ \mathbf{B}\ \mathbf{C} \\
\left[\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 0 \\
0 & 0 & 0 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
0 & 0 & 1
\end{array}\right]
\end{array}
$$

(a) Transmission matrix

$$
\begin{aligned}
&\text{max: } \textstyle\sum_{i=0}^{5} R_i \\
S_{\mathbf{A}} + S_{\mathbf{B}} + S_{\mathbf{C}} &\geq R_0 \\
S_{\mathbf{A}} + S_{\mathbf{B}} &\geq R_1 \\
0 &\geq R_2 \\
S_{\mathbf{A}} + S_{\mathbf{B}} + S_{\mathbf{C}} &\geq R_3 \\
S_{\mathbf{A}} + S_{\mathbf{B}} + S_{\mathbf{C}} &\geq R_4 \\
S_{\mathbf{C}} &\geq R_5 \\
R_0, R_1, R_2, R_3, R_4, R_5 &\in \{0, 1\} \\
S_{\mathbf{A}}, S_{\mathbf{B}}, S_{\mathbf{C}} &\in \{0, 1\} \\
S_{\mathbf{A}} + S_{\mathbf{B}} + S_{\mathbf{C}} &= 1
\end{aligned}
$$

(b) ILP problem

Fig. 5: ILP problem with coverage on faults of internal gates

The fact that we use *sequential* ATPG to determine a set of latches to which a fault may propagate addresses the concerns described in §3.1. By injecting faults not only in latches but also in internal gates, we effectively obtain a larger set of latches that are potentially corrupted, which increases the intersection of latches that capture several faults. We use a fixed number of cycles for the generation of the test scenarios; details are provided in § 4.

4 Experimental Evaluation

4.1 Trace Signal Selection

In our experiments, we evaluated our methodology using the single stuck-at-fault model on two benchmarks from Opencores.org: the 68HC05 (127 latches)

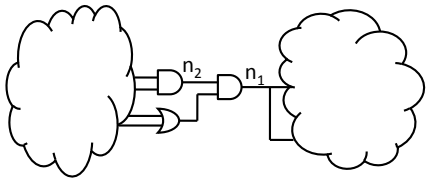


Fig. 6: Nodes with different fanout degrees

and 8051 (2794 latches) microcontrollers[3]. This fault model is chosen for its convenience as it is well understood. This is not a limitation of our approach. Faults are injected on both latches and internal gates and fault simulation used to build an error transmission matrix. Two issues need careful consideration in constructing the error transmission matrix.

First, each row in the error transmission matrix represents the detection of a fault in the circuit. Including all possible faults, whose number is proportional to the size of the circuit, can result in very large matrices. To reduce the number of faults that need to be considered, we limit the fault sites to the outputs of fanout-free regions and take advantage of well-known results on fault equivalence. As shown in Figure 6, node n_1 has a fanout-degree of two, while the fanout-degrees for node n_2 is one. A stuck-at-1 fault occurs at node n_2. If this fault is activated by a vector and propagates to node n_1, it is equivalent to a stuck-at-1 fault at n_1. If the fault on node n_2 is not propagated to node n_1 for a vector, it is masked. Thus, the two rows corresponding to these two stuck-at-1 faults at n_1 and n_2 in the error transmission matrix are exactly the same. We refer to those nodes with fanout-degree larger than 1 as fanout-points. As a result, it is sufficient to consider faults on fanout-points without losing any error transmission information.

Second, to build the error transmission matrix, we need to know where each fault can propagate to. This is achieved by using fault simulation. Unlike the proposed method from [19], we use multi-cycle test patterns obtained from sequential ATPG for fault simulation. The test patterns was limited to 6-cycle tests to manage test generation time. A test generated for a fault is used in fault simulation for all the faults.

In our experiments, we limited the number of trace signals to be 5% of the total number of latches which is the candidate set of trace signals. We used the CPLEX ILP solver and AMPL modeling language[4]. For the 68HC05 benchmark, the ILP solver returns 6 optimized trace signals which can capture 63.75% of stuck-at faults. For the 8051 benchmark, 140 trace signals are identified that capture 31.70% of stuck-at faults. This coverage is strongly related to the length

[3] OpenCores projects available online at http://opencores.org/project
[4] CPLEX for AMPL available online at http://www.ampl.com/CPLEX

of the test vectors since some of the faults may not be covered at all using the 6-cycle tests.

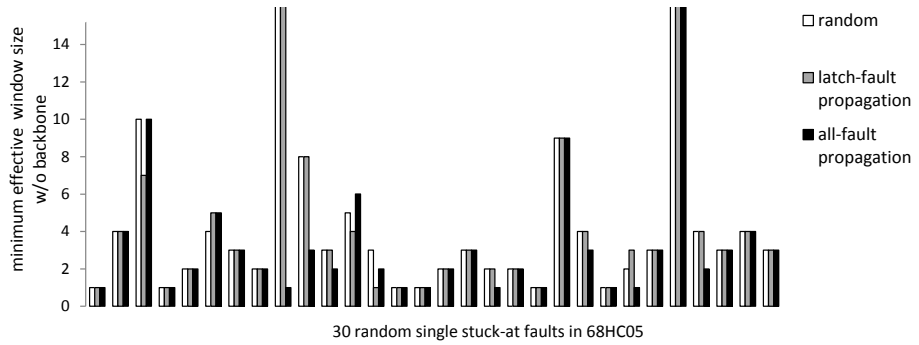
4.2 Evaluation Method

To evaluate the trace signals selected by our methodology, 30 single stuck-at faults are injected to both the 68HC05 and 8051 benchmarks to generate 60 faulty circuits. To mimic the post-silicon debug process, instead of running a real chip prototype, each faulty circuit is simulated by a tailored test vector. The aim of this tailored test vector is to cause the specific fault to be activated and observed at some latch or circuit output, just as an erroneous trace would. The rationale for this is as follows. As our experimentation is based on simulation rather than at-speed post-silicon validation, it is much slower and thus is limited to a length of a few thousand cycles. The likelihood of a bug being detected by a random trace of this length is quite low. As a result, we tailored one test vector for each faulty circuit. Each of these vectors are 3000 cycles long and the bug in the circuit is guaranteed to be activated roughly every 100 cycles. However, there is no guarantee the error will be observed at the outputs or in the trace buffers, and it is exactly this aspect of the fault propagation that we wish to observe. For this purpose, during simulation, the execution trace is recorded on the selected trace buffers and output pins. This is then used as constraints for the offline SAT-based analysis described in §2.3. By using the sliding window analysis along this execution trace, we can determine whether the bug can be localised. Our metric for the quality of the trace signals is the size of the window required to localise the fault. A smaller window indicates a higher quality of selection as it results in a more scalable localisation algorithm. This is because a smaller window means that a smaller number of circuit unfoldings need to be considered in the analysis. Thus with limited capacity of analysis engines such as SAT solvers, the size of the circuit that can be accommodated is much larger.

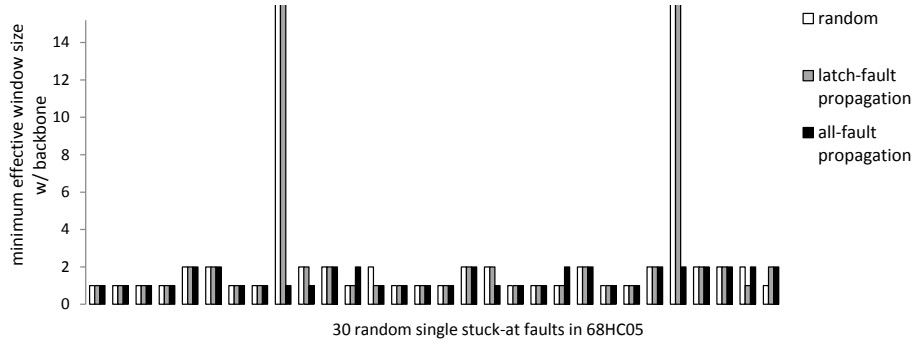
Compared to trace signal selection, the evaluation phase is the more time-consuming. Selecting trace signals only involves one call to the ILP solver, which is able to solve a circuit 10 times larger than 80hc51 in a few minutes. However, in the evaluation phase, the SAT solver is called repeatedly for each window and its running time grows exponentially to circuit size. Evaluating one set of selected trace signals on 30 faults in 80hc05 usually takes more than 10 hours. Therefore, to find a better set of trace signals which allows bugs to be detected in smaller windows is crucial in our work.

4.3 Experimental Results

Three different sets of trace signals are compared in this evaluation process. The first set of trace signals are randomly selected. The second set of trace signals are selected using the approach of Yang and Touba [19] (latch-fault propagation), i.e., the error transmission matrix is built based on bugs injected only on latches and single cycle fault propagation. The last set of trace signals are selected based on our approach (all-fault propagation) described above. Further, as in



(a) Sliding window without backbone

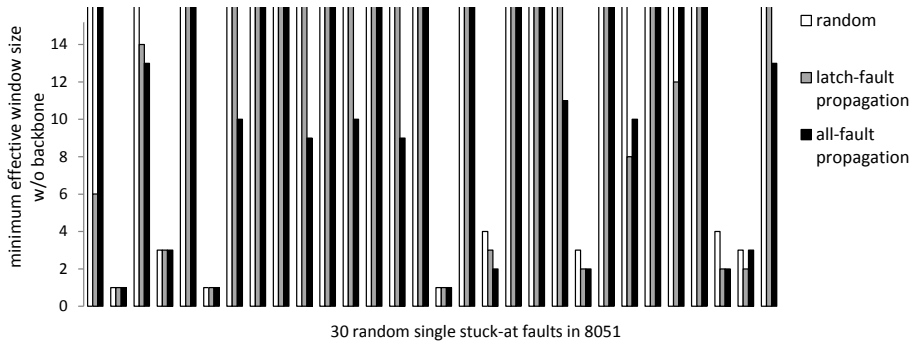


(b) Sliding window with backbone

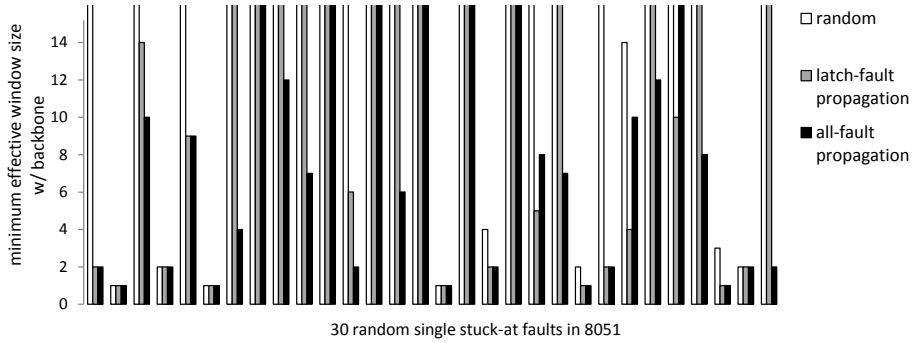
Fig. 7: Minimum window sizes to detect bugs randomly injected in 68HC05 with three different sets of selected trace signals

our previous work, each set of trace signals is evaluated by two different sliding windows, i.e. with using backbones and without using backbones.

In Figure 7 and Figure 8, each graph represents one of the two sliding window analyses on different benchmarks. On the x axis, there are 30 randomly injected single faults for both benchmarks. The y axis represents the minimum window size required to detect the corresponding fault by the SAT-based fault localisation approach described in §2.3. We used a limit of 15 time-frames for the size of the sliding window to manage experimental run times. There are $30 \times 4 = 120$ set of comparisons between the three trace buffer selection approaches. The first random method rarely performs better than the other two. Compared to the second latch-fault propagation method, our all-fault propagation approach gives better performance (detects a bug with a smaller window size) in 29 cases and worse performance in 13 cases. Further, more importantly, among the 29 cases



(a) Sliding window without backbone



(b) Sliding window with backbone

Fig. 8: Minimum window sizes to detect bugs randomly injected in 8051 with three different sets of selected trace signals

where the all-fault propagation method is better, 17 of them cannot be detected by the second approach with window size up to 15. It is very possible that the bug cannot be detected by those trace signals independent of window size. Conversely, among the 13 cases where the latch-fault propagation is better, only 3 of them cannot be detected by our approach with window size up to 15. Thus, our approach can detect $17 - 3 = 14$ more faults among the total of 120 cases than the latch-fault propagation approach.

5 Conclusions

This paper considers the problem of selecting trace signals in post-silicon validation for use in fault-localisation. It uses a coverage based problem formulation that maximizes the number of faults that can be detected at the trace signals

using a limited number of trace signals. In contrast to a the coverage based formulation of Yang and Touba, our formulation considers faults at all circuit sites, and not just latches. This increases the likelihood of fault localisation being able to isolate faults to these sites. Further, in our formulation we consider multi-cycle fault propagation, which more accurately captures real fault propagation compared to the single cycle propagation of Yang and Touba's method. The value of these differences is reflected in the experimental results where we compare various trace signal selection algorithms in terms of their ability to reduce the window size needed in sliding window consistency based fault-localisation. This metric is a proxy for scalability, as a smaller window size indicates that fewer time frames are needed and thus a larger circuit can be accommodated in each time frame. Our method performs much better than both random selection, as well as the Yang and Touba method. Specifically it can detect and localize 14 more faults of a total of 60 faults than the Yang and Touba method.

References

1. M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital systems testing and testable design*. Computer Science Press, 1990.
2. A. Brillout, N. He, M. Mazzucchi, D. Kroening, M. Purandare, P. Rmmer, and G. Weissenbacher. Mutation-based test case generation for Simulink models. In *Formal Methods for Components and Objects*, volume 6286 of *LNCS*, pages 208–227. Springer, 2010.
3. Y. Chen, S. Safarpour, J. Marques-Silva, and A. Veneris. Automated design debugging with maximum satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 29:1804–1817, 2010.
4. Y. Chen, S. Safarpour, A. Veneris, and J. Marques-Silva. Spatial and temporal design debug using partial MaxSAT. In *Great Lakes Symposium on VLSI*, pages 345–350. ACM, 2009.
5. K.-T. Cheng and L.-C. Wang. Automatic test pattern generation. In *EDA for IC System Design, Verification, and Testing*. CRC Press, 2006.
6. W. Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. 22(3):250–268, 1957.
7. F. M. De Paula, M. Gort, A. J. Hu, S. J. E. Wilton, and J. Yang. Backspace: formal analysis for post-silicon debug. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 5:1–5:10. IEEE, 2008.
8. F. M. de Paula, A. Nahir, Z. Nevo, A. Orni, and A. J. Hu. TAB-Backspace: unlimited-length trace buffers with zero additional on-chip overhead. In *Proceedings of the 48th Design Automation Conference*, Design Automation Conference (DAC), pages 411–416. ACM, 2011.
9. N. Eén and N. Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.
10. Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 4121 of *LNCS*, pages 252–265. Springer, 2006.
11. E. Hung and S. Wilton. On evaluating signal selection algorithms for post-silicon debug. In *Quality Electronic Design (ISQED)*, March 2011.

12. B. Keng, S. Safarpour, and A. G. Veneris. Bounded model debugging. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 29(11):1790–1803, 2010.
13. J. Marques-Silva, M. Janota, and I. Lynce. On computing backbones of propositional theories. In *European Conference on Artificial Intelligence (ECAI)*, pages 15–20. IOS Press, 2010.
14. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *Design Automation Conference (DAC)*, pages 530–535. ACM, 2001.
15. S. Prabhakar and M. Hsiao. Multiplexed trace signal selection using non-trivial implication-based correlation. In *Quality Electronic Design (ISQED)*, pages 697–704, 2010.
16. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, April 1987.
17. S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 13–19. IEEE, 2007.
18. A. Sülflow, G. Fey, R. Bloem, and R. Drechsler. Using unsatisfiable cores to debug multiple design errors. In *Great Lakes Symposium on VLSI*, pages 77–82. ACM, 2008.
19. J.-S. Yang and N. A. Touba. Efficient trace signal selection for silicon debug by error transmission analysis. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 31(3):442–446, 2012.
20. Y.-S. Yang, B. Keng, N. Nicolici, A. G. Veneris, and S. Safarpour. Automated silicon debug data analysis techniques for a hardware data acquisition environment. In *International Symposium on Quality of Electronic Design*. IEEE, 2010.
21. C. S. Zhu, G. Weissenbacher, and S. Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In *Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2011.