Fault-Diagnosis-Based Technique for Establishing RTL and Gate-Level Correspondences

Srivaths Ravi, Indradeep Ghosh, *Member, IEEE*, Vamsi Boppana, and Niraj K. Jha, *Fellow, IEEE*

***Abstract*—In this paper, we address an important problem associated with hierarchical design flows (termed the mapping problem): identifying correspondences between a signal in a high-level specification and a net in its lower level implementation. Conventional techniques use shared names to associate a signal with a net whenever possible. However, given that a synthesis flow may not preserve names, such a solution is not universally applicable. This work provides a robust framework for establishing register-transfer level (RTL) signal to gate-level net correspondences for a given design. Our technique exploits the observation that circuit diagnosis provides a convenient means for locating faults in a gate-level network. Since our problem requires locating gate-level nets corresponding to RTL signals, we formulate the mapping problem as a query whose solution is provided by a circuit diagnosis engine. Our experimental work with industrial designs for many mapping cases shows that our solution to the mapping problem is 1) fast and 2) precise in identifying the gate-level equivalents (the number of nets returned by our mapping engine for a query is typically one or two even for designs with tens of thousands of VHDL lines).**

***Index Terms*—Fault diagnosis, incremental synthesis, signal correspondences, verification.**

I. Introduction

CIRCUIT designs are typically represented at several levels of abstraction such as a geometric description of the layout, a gate-level netlist, or a higher level specification in a hardware description language like VHDL. Synthesis tools yield a hardware design by gradually refining a high-level specification of the design [register-transfer level (RTL)] to a lower level implementation (gate level or layout) using a series of mapping and optimization phases.

Several applications in the top-down design approach require designers to identify a portion of the gate-level implementation that corresponds to a section of its RTL specification. For example, "design bugs" and "late-arriving functionality" force incremental changes to a specification and the gate-level implementation synthesized from it. Since the designer usually has some investment in the original implementation [1], it is desirable to reuse as much of that implementation as possible. This has inspired the Synopsys engineering change order (ECO) compiler [2], which uses the gate-level incremental synthesis techniques (specifications and implementations are gate-level networks) [1], [3], [4] to carry out engineering changes (ECs) to RTL specifications. The methodology takes the logical descriptions synthesized from the original and modified RTL specifications and identifies equivalent logic regions for reuse through intensive formal verification techniques. In this way, the new implementation is gradually modified to inherit as much of the original netlist structure as possible.

A clear disadvantage of the above approach is that a significant portion of the time is spent on identifying the large portions of the design that are unlikely to change (formal verification of industrial designs typically takes several hours to days for completion). An alternative way to make incremental synthesis work would be to provide a viable means for identifying the *appropriate* portion that changes in the gate-level implementation due to a modification of the RTL specification. We can clearly do this if we can establish a correspondence between a signal of interest in the RTL specification and a net or set of nets in its gate-level implementation. This is the *mapping problem* tackled in this paper.

Another application that critically requires an efficient solution to the mapping problem is RTL to gate-level verification. This is required to improve the performance of the core verification engine that compares a "mapped" (low-synthesis effort) gate-level netlist obtained from the RTL circuit with the implementation netlist. Such verification techniques have been known to scale well to large designs only when a large number of internal correspondences can be established between the two netlists [5]. Therefore, a technique that efficiently maps RTL signals to gate-level nets in the implementation netlist would significantly improve their overall performance.

From the above discussion, we see that a solution to the mapping problem has the potential to affect applications as diverse as incremental synthesis and design verification. The main bottleneck that has confronted researchers so far is the absence of a clear way for identifying the gate-level net corresponding to an RTL signal. Traditional approaches have relied on using shared names to relate signals and nets, whenever possible. However, this approach has a major disadvantage since names are usually not preserved across the multiple phases that make up a synthesis framework. This is true not only when multiple vendors cater to a synthesis framework, but also when a single vendor is responsible. Even otherwise, finding an exact solution to the mapping problem may be practically impossible. For example, a signal in the VHDL specification may not appear as a signal in the gate-level network due to optimizations such as constant propagation, technology mapping, etc. Given that an application such as EC restructures the logic around an RTL signal of

Manuscript received August 7, 2000; revised June 1, 2001. This paper was recommended by Associate Editor S. Reddy.

S. Ravi is with the NEC Computer and Communications Research Laboratories, Princeton, NJ 08540 USA.

I. Ghosh and V. Boppana are with the Fujitsu Laboratories of America, Sunnyvale, CA 94086 USA.

N. K. Jha is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA.

Publisher Item Identifier S 0278-0070(01)10637-8.

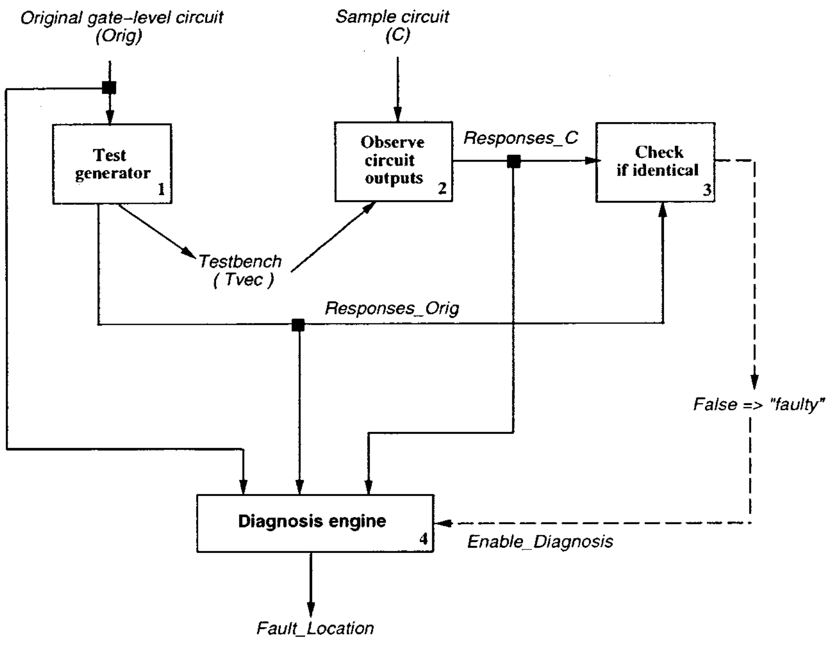

Fig. 1. Flow chart for diagnosis.

interest, even a solution that can identify the gate-level equivalents of RTL signals in topological proximity to the EC is desirable.

Our solution to the mapping problem is motivated by observations from a different domain, namely, *circuit testing and diagnosis* [6]. Testing is performed to determine if a circuit is faulty. Once a circuit is found to be faulty, fault diagnosis is performed to locate the fault. Typical chip fault diagnosis proceeds as shown in Fig. 1. A testbench T targeting a set of modeled faults for the original gate-level implementation Orig is generated by using a test generator, e.g., HITEC [7] (Step 1). The vector sequence Tvec is then applied to the given circuit C (Step 2). If the circuit responses so obtained do not match the expected responses, C is declared faulty (Step 3). A diagnosis engine looks at the faulty responses and Orig to pinpoint the location in Orig, which is faulty in C (Step 4).

A natural solution to the mapping problem follows from the above statements. First, we introduce a suitable *fault* at the signal of interest in the RTL description. Note that this modification is introduced only for analysis and does not denote a permanent addition. Then, we simulate this *faulty* RTL description to generate the responses for a selected testbench (if the testbench is generated by a gate-level sequential test generator such as HITEC, then the injected fault is a single stuck-at fault. This is further detailed in Section III). Lastly, a fault diagnosis engine looks at these *faulty* responses and the original gate-level implementation for identifying the desired gate-level net or set of nets. Our experiments indicate that this natural solution is a practical plugin to any synthesis system. We also use this solution to propose an incremental synthesis method that effectively tackles the problem of making ECs to RTL specifications.

The rest of this paper is organized as follows. Section II introduces the mapping problem and Section III presents the diagnosis-based solution methodology. Section IV applies this solution to the problem of making ECs to RTL specifications. Section V presents experimental results and Section VI concludes the paper.

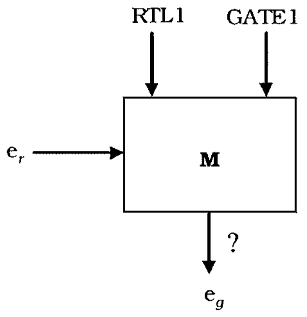

Fig. 2. Inputs and output for the mapping problem.

II. Mapping Problem

In this section, we formally define the mapping problem. Then, we use a simple example to demonstrate the shortcomings of a conventional solution. Lastly, we illustrate why fault diagnosis offers a natural solution to the mapping problem.

Consider the RTL description of a circuit $\mathrm{RTL1}$ with a set of signals $\mathrm{E_r}$ and a set of processing elements $\mathrm{V_r}$ such that $\mathrm{RTL1} = G(\mathrm{V_r}, \mathrm{E_r})$ ($\mathrm{RTL1}$ is a graph with vertex set $\mathrm{V_r}$ and edge set $\mathrm{E_r}$). Let $\mathrm{GATE1}$ be the gate-level implementation synthesized from $\mathrm{RTL1}$ with a set of gates $\mathrm{V_g}$ and a set of interconnections $\mathrm{E_g}$ such that $\mathrm{GATE1} = G(\mathrm{V_g}, \mathrm{E_g})$. Then, the mapping problem M is defined as follows.

Definition 1: Let $\mathrm{e_r} \in \mathrm{E_r}$. Does there exist an $\mathrm{e_g} \in \mathrm{E_g}$ such that $\mathrm{e_r}$ and $\mathrm{e_g}$ are functionally equivalent?

The inputs and output of the mapping problem are shown in Fig. 2. To understand the problem better, let us consider an instance of the mapping problem and a conventional solution for it.

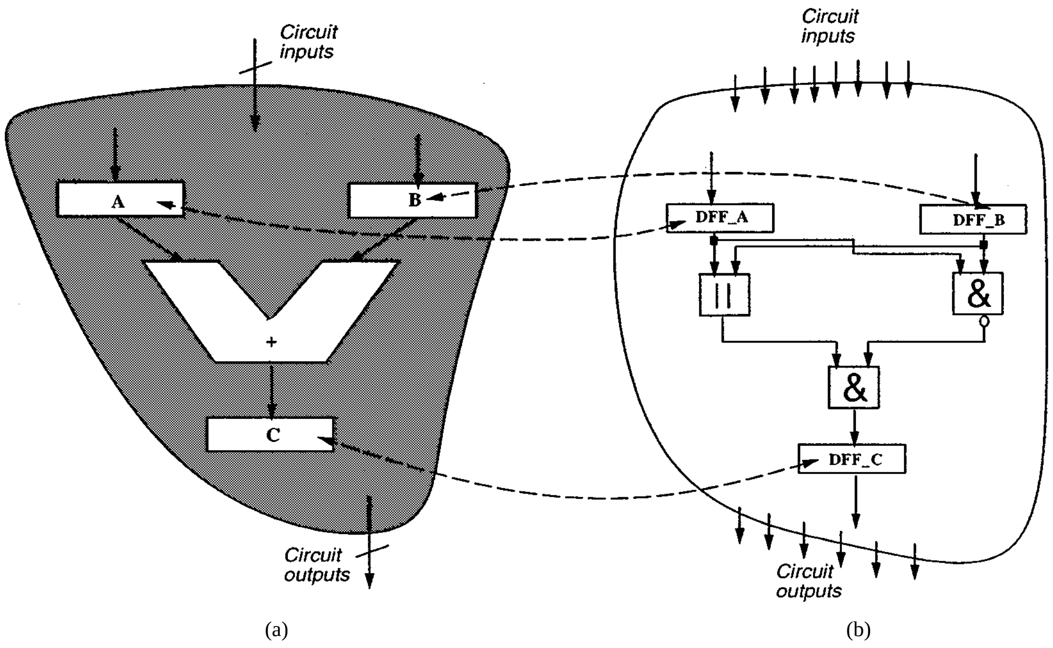

Fig. 3. Parts of (a) an RTL circuit and (b) its gate-level netlist with name correspondences preserved.

Example 1: Fig. 3(a) shows the structural view of part of an RTL circuit, where the sum of signals A and B is assigned to signal C. Assume that these latched signals are single-bit values and that a gate-level netlist synthesized from this circuit description is also available [Fig. 3(b) represents a part of the gate-level netlist]. Consider the mapping problem of identifying the net in Fig. 3(b) corresponding to signal B in Fig. 3(a).

A conventional solution to this problem is shown in Fig. 3(b). This solution assumes that the names of the nets are preserved across the different synthesis phases. Consequently, the latch corresponding to signal A is named $\mathrm{DFF\_}A$, and so on. With this information, we identify the output of $\mathrm{DFF\_}B$ as the relevant net of interest. This solution is limited in its applicability for the following reasons.

1) An existing system must maintain correspondence between the names present in input and output descriptions of the individual synthesis phases. Maintaining such databases places a considerable burden on the end-user system.
2) The stringent naming requirements must not only be met by the existing synthesis phases, but also by any new optimization plugins added to the synthesis path. If these plugins are developed by other vendors, the proposed solution is likely to break down. ■

In the above example, we saw a conventional solution based on naming correspondences and its limitations. In the next example, we outline a robust scheme for overcoming these limitations. Details of the solution methodology follow in the subsequent sections.

Example 2: Consider now the RTL circuit shown in Fig. 4(a), where signal B is set to logic value 0. If we simulate this circuit with a testbench, the set of responses output by the circuit is likely to be different from the set of responses output by the circuit in its normal functioning. We can restate the previous statement in a stronger manner as follows. The set of responses output by the modified RTL circuit is identical to the set of responses output by a gate-level netlist [see Fig. 4(b)] with a stuck-at 0 (SA0) fault on $\_N16$, the gate-level equivalent of signal B. In other words, a diagnosis engine examining the *faulty* RTL responses and the original gate-level netlist would output $\_N16$ with an SA0 fault as the solution. In this way, we locate the gate-level equivalent ($\_N16$) of signal B. ■

III. Fault-Diagnosis-Based Solution Framework

In this section, we present an overview of the proposed solution methodology followed by details of the constituent steps.

A. Solution Paradigm

In Section I, we saw that fault diagnosis provides a direct means for locating faults in a circuit. Before proceeding further, let us first describe the fault diagnosis problem formally. Suppose that we have a fault-free gate-level circuit $\mathrm{GATE1} = G(\mathrm{V_g}, \mathrm{E_g})$ with vertex set $\mathrm{V_g}$ and edge set $\mathrm{E_g}$ and a set of test vectors Tvec targeting single stuck-at faults in GATE1. Then, the diagnosis problem D (see Fig. 5) can be defined as follows.

Definition 2: If Resp is the sequence of responses that a faulty implementation of GATE1 yields on receiving Tvec, which net $\mathrm{e_g}$ (or nets) in GATE1 could have a single stuck-at fault to cause such a sequence of responses?

Since the mapping problem M (Definition 1) requires locating the gate-level net $\mathrm{e_g}$ corresponding to an RTL signal $\mathrm{e_r}$, we can formulate the mapping problem as a question whose answer is provided by diagnosis (Definition 2). In other words, we can reduce M to D using the transformation T described below.

1) Introduce a single stuck-at fault at signal $\mathrm{e_r}$ in $\mathrm{RTL1}$. That is, set $\mathrm{e_r}$ permanently to logic value 1 [stuck-at 1 (SA1) fault] or to logic value 0 (SA0 fault). This constitutes the *stuck-at transformation* F shown in Fig. 6.

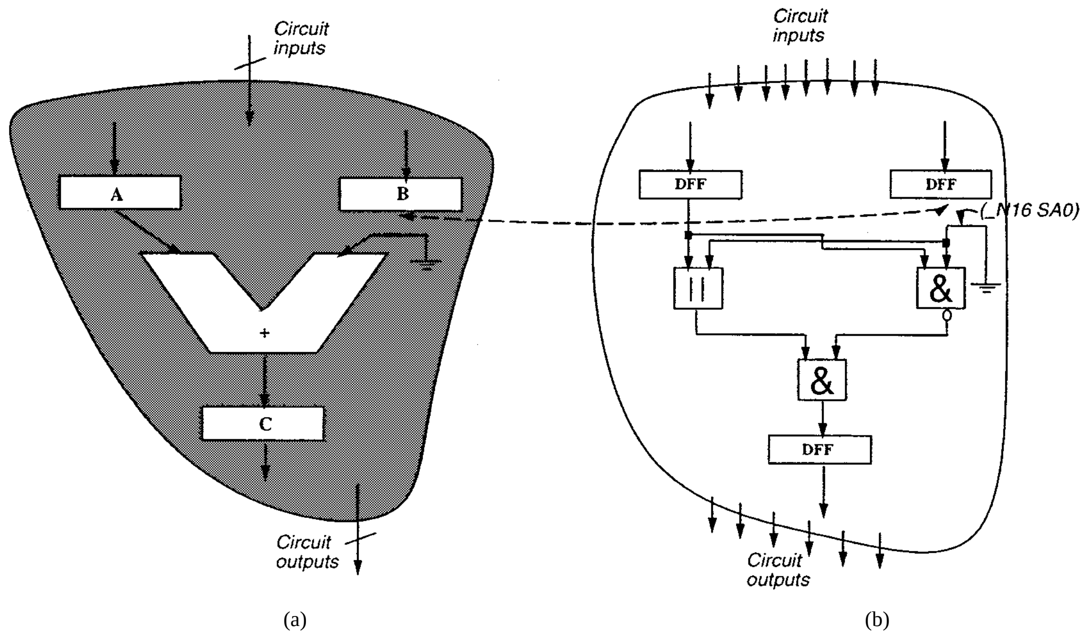

Fig. 4. Parts of (a) an RTL circuit and (b) its gate-level netlist with the mapping solution determining the correspondence.

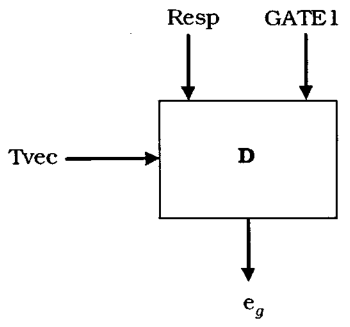

Fig. 5. Diagnosis problem D.

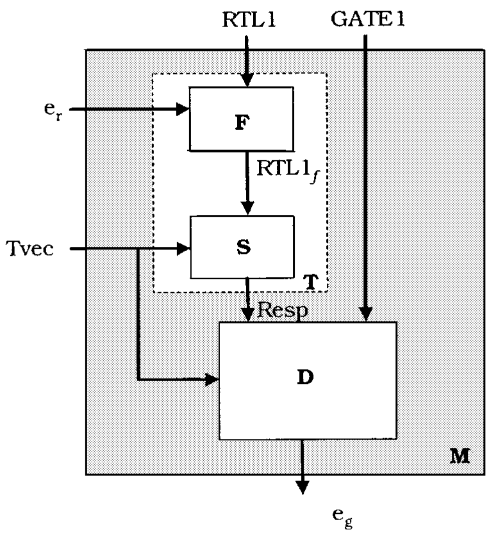

Fig. 6. Reduction T to solve the mapping problem M.

2) F creates a circuit (at the RTL) with a single stuck-at fault. We now simulate this *faulty* RTL circuit $\mathrm{RTL1_f}$ with the testbench Tvec for the gate-level circuit $\mathrm{GATE1}$ to obtain a set of faulty responses Resp (phase S in Fig. 6).

3) The composition FS constitutes the desired reduction T.

Fig. 6 now completes the solution methodology. When a gate-level diagnosis engine $\mathrm{D_e}$ examines the faulty responses along with the description of $\mathrm{GATE1}$, it solves the diagnosis problem D by determining the net or a set of nets in $\mathrm{GATE1}$, which, when faulty, will produce Resp. This net corresponds to the gate-level equivalent of $\mathrm{e_r}$ since the fault was introduced at $\mathrm{e_r}$ (note that a single stuck-at transformation can identify nets to within the equivalence class of faults; this is further discussed in the subsequent sections). In this way, reduction T gives us the opportunity to use an existing solution $\mathrm{D_e}$ to solve M. The strengths and limitations of $\mathrm{D_e}$ (along with the circuit under consideration) clearly contribute to the extent to which we can solve M successfully.

With the help of the following example, we illustrate the RTL transformation technique and show how our methodology can be systematically applied.

Example 3: Consider process $\mathrm{Proc1}$ shown in Fig. 7, which forms a fragment of a VHDL description. We are going to use VHDL throughout this paper for consistency. However, similar transformations can be done on Verilog descriptions as well. $\mathrm{Proc1}$ assigns different values to the parameter offset based on the values taken by $\mathrm{control}$. Let us consider the mapping problem, which examines the following query: *which gate-level net(s) corresponds to* $\mathrm{offset(0)}$ *(bit 0 of offset)?*

We can systematically apply the aforementioned solution paradigm as follows (with $\mathrm{RTL1}$, $\mathrm{GATE1}$, and $\mathrm{RTL1_f}$ assuming the usual meaning).

1) In order to establish a correspondence between $\mathrm{offset(0)}$ with an unknown net or set of nets, say U, in the gate-level network $\mathrm{GATE1}$, we first introduce a stuck-at fault at line $\mathrm{offset(0)}$. This is shown in Fig. 8, where assignments to offset in lines 3–5 are modified to incorporate an SA1 fault at $\mathrm{offset(0)}$ (using the VHDL concatenation operator &). This transformation yields the new RTL description $\mathrm{RTL1_f}$.

```
Procl: process(control)
begin
   case control is
      when 3 => offset <= “0011”;
      when 6 => offset <= “0100”;
      when others => offset <= “0000”;
   end case;
end process Procl;
```

Fig. 7. Fragment of VHDL code called $Ex1$.

```
Procl: process(control)
begin
   case control is
      when 3 => offset <= “001” & ‘1’;
      when 6 => offset <= “010” & ‘1’;
      when (others) => offset <= “000” & ‘1’;
   end case;
end process Procl;
```

Fig. 8. Example of the stuck-at transformation: $\mathrm{offset}(0)$ in $Ex1$ set to 1.

2) Next, given the gate-level network GATE1, say with six inputs and eight outputs, an automatic test pattern generation tool is run to obtain a testbench $\mathrm{T_{GATE1}}$ that targets all the single stuck-at faults in GATE1. Suppose that $\mathrm{T_{GATE1}}$ is the vector sequence given below

$$\mathrm{T_{GATE1}} = \{000011,\ 010011,\ 100000\}. \tag{1}$$

We now simulate $\mathrm{RTL1_f}$ with $\mathrm{T_{GATE1}}$ to get the output responses Resp. Suppose that Resp is as follows:

$$\mathrm{Resp} = \{10111111,\ 11011011,\ 00000011\}. \tag{2}$$

3) A diagnosis engine now accepts GATE1 and Resp as its inputs. A simple way of looking at the diagnosis procedure is to associate a lookup structure that successively prunes the list of candidate faulty nets as it sequentially examines both the expected and faulty responses. This is illustrated in Fig. 9. Suppose that we have an initial list of four candidate faults $\mathrm{L_f}$ at the end of the first step as follows:

$$\mathrm{L_f} = [G7\,\mathrm{SA0},\ G39\,\mathrm{SA1},\ \_N38\,\mathrm{SA0},\ \_N40\,\mathrm{SA1}]. \tag{3}$$

The next comparison of responses prunes $\mathrm{L_f}$ to $[G7\,\mathrm{SA0},\ \_N40\,\mathrm{SA1}]$. Diagnosis finally outputs $\mathrm{L_f}$ with a unique member $[\_N40\,\mathrm{SA1}]$. This means that $\_N40$ in GATE1 is the faulty gate-level net that could have produced the computed responses Resp. Or, $\_N40$ is the desired gate-level equivalent of RTL signal $\mathrm{offset}(0)$. ■

Note that fault diagnosis (Step 3 in the above example) can be done in many ways. Static techniques [8] use precomputed information in the form of fault dictionaries to match the faulty responses produced by the defective circuits. Dynamic techniques [9] diagnose the fault behavior of the circuit while the test set is applied. More recently, integrated techniques use small amounts of precomputed information in tandem with dynamic algorithms to perform efficient fault location [10].

B. Reduction—A Closer Look

In this section, we examine the individual steps of the reduction in greater detail. First, we examine stuck-at transformations and then discuss the test generation and diagnosis phases of the solution.

1) Stuck-At Transformations: The stuck-at transformation that introduces an SA0 or an SA1 fault at an RTL signal for subsequent processing can be formally defined as follows.

Definition 3: The *stuck-at transformation* for an RTL signal S modifies all assignments to S so that S takes a constant value, i.e., either logic 0 (for an SA0 transformation) or logic 1 (for an SA1 transformation).

In many cases, we can apply either the SA0 transformation or the SA1 transformation (Example 3) to find a solution to the mapping problem. However, in some cases, we may have to use both the transformations. This is because the precision with which diagnosis can identify a gate-level net is limited by the faults to which a fault on that gate-level net is equivalent. For example, consider the NAND gate network for an XOR function of two variables shown in Fig. 10 [11]. The classical example for equivalence is that line c SA0 and line g SA1 are indistinguishable because they produce the same faulty responses for all input vectors. In fact, the equivalence class containing $c\,\mathrm{SA0}$ is $\mathrm{E_0} = \{c\,\mathrm{SA0},\ g\,\mathrm{SA1},\ d\,\mathrm{SA0},\ b\,\mathrm{SA1}\}$. Likewise, the equivalence class containing $c\,\mathrm{SA1}$ is $\mathrm{E_1} = \{c\,\mathrm{SA1},\ f\,\mathrm{SA1}\}$.

Extending the above example to the mapping problem, we can see that if line c is the desired mapping solution and the RTL stuck-at transformation is actually targeting $c\,\mathrm{SA0}$, then our solution will be accurate to the resolution level of $\mathrm{E_0}$ (elements of $\mathrm{E_0}$ are indistinguishable by definition). Similarly, if we use only the RTL SA1 transformation that is actually targeting $c\,\mathrm{SA1}$, then our solution will be accurate to the resolution level of $\mathrm{E_1}$. However, if we use both the transformations and compare the results, we find that the only net identified by both the transformations is net c (in other words, the intersection of $\mathrm{E_0}$ and $\mathrm{E_1}$ finds the net faulty in both $\mathrm{E_0}$ and $\mathrm{E_1}$, which is c). In this way, application of both the SA0 and SA1 transformations at the RTL [so as to target SA0 and SA1 faults on the corresponding gate-level net(s)] improves the accuracy of the mapping solution.

The stuck-at transformation is a simple and powerful tool that is applicable to *any* signal in the VHDL description of a circuit irrespective of the language constructs used. It can be applied to any RTL signal that is expressible as a bit or bit-vector data type. The next example illustrates a typical instance of how this transformation can be incorrectly applied and the subsequent side-effects.

Example 4 (Latch Inference Problem): Consider, for example, the VHDL conditional construct shown in Fig. 11(a) and the problem of applying an SA0 transformation to $B(15)$. Direct modification of the construct to incorporate a permanent logic 0 at $B(15)$ preserves the $B(15)$ assignment in the **if** portion while modifying the assignment in the **elseif** portion. This is shown in Fig. 11(b). However, the indicated modifications do not completely achieve the objective of introducing logic-0 settings on all assignments to the $B(15)$ signal. This is because of potential implicit latch inferences

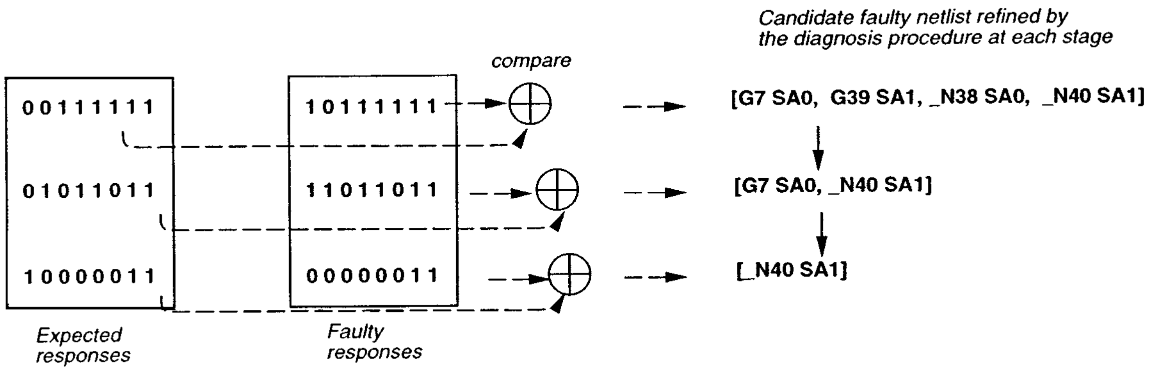

Fig. 9. Snapshot of faulty net identification using fault diagnosis.

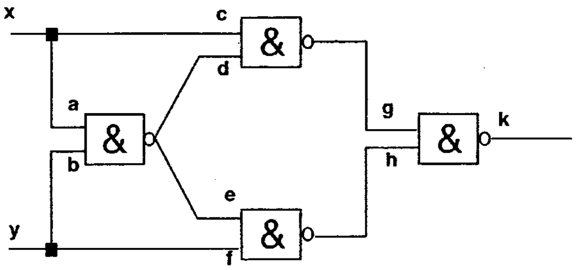

Fig. 10. NAND gate network for the XOR function.

```
if (a=1) then
  B(15 downto 0) <= 0;
elseif (a=2) then
  B(15 downto 0) <= C(15 downto 0) + '1';
endif;
```

(a)

```
if (a=1) then
B(15 downto 0) <= 0;
elseif (a=2) then
  B(14 downto 0) <= C(14 downto 0) + '1';
  B(15) <= '0';
endif;
```

(b)

Fig. 11. VHDL code fragment (a) in its original form and (b) with an incorrect application of the stuck-at transformation.

```
if (a=1) then
B(15 downto 0) <= 0;
elseif (a=2) then
  B(14 downto 0) <= C(14 downto 0) + '1'
  B(15) <= '0';
else
  B(15) <= '0';
endif;
```

Fig. 12. VHDL code fragment with the correct usage of the stuck-at transformation.

in a synthesis system [12]. In other words, Fig. 11(b) ignores the presence of an implicit **else** construct that executes the "$B(15)$ <= $B(15)$" statement in Fig. 11(a) [note that the "$B(14$ downto $0)$ <= $B(14$ downto $0)$" statement is still inferred]. Therefore, the correct transformation should explicitly include an **else** statement as shown in Fig. 12.

Let us now analyze how differences between the code in Figs. 11(b) and 12 can affect the overall solution. Suppose that signal a takes the following sequence of values $\{3, 3, 3, 1, 2\}$. Then, $B(15)$ in Fig. 11(b) evaluates to $\{X, X, X, 0, 0\}$. However, $B(15)$ in Fig. 12 evaluates correctly to $\{0, 0, 0, 0, 0\}$ since $B(15)$ has an SA0 fault. Since a diagnostic engine compares the expected and faulty responses to obtain the fault location, the faults identified in the two cases are likely to be different. ■

2) Test Generation and Diagnosis: The RTL description generated after applying the stuck-at transformation is characterized by an SA0 or SA1 fault on an RTL signal. Since our approach requires this fault to be detected and located in the original gate-level description, we will now examine how this affects the success of our method.

Observation 1: The probability of locating the gate-level equivalent of an RTL signal is limited by the testability of the faults induced in the gate-level implementation by the two stuck-at transformations applied to the RTL signal.

Let us analyze the above observation in greater detail. If the faults introduced on the RTL signal alter the output responses for the given testbench, then they are detectable. Otherwise, if neither of the two faults gets detected, we cannot proceed further. Extending these statements further, we can say that if a circuit is highly testable, then it is very likely that we can find a mapping solution. Available information about the economics of circuit design and manufacturing [13], [14] indicate that at least 98% fault coverage is necessary to reduce field failures and cut down costs. This statistic as well as empirical evidence suggests that our solution methodology will fare well for circuits designed in the industry. Note that even if the SA0 (SA1) fault on a line is untestable, but the corresponding SA1 (SA0) fault is testable, our method is still applicable.

While test vectors that guarantee high fault coverage are crucial for fault detection, their diagnostic quality in terms of their ability to distinguish fault pairs is also important. For example, suppose that testbench T1 (T2) detects fault f1 because the circuit with fault f1 generates a response R1 (R1′) that differs from the normal response R_{T1} (R_{T2}). Likewise, suppose that testbench T1 (T2) detects fault f2 because the circuit with fault f2 generates a response R1 (R2) that differs from the normal response R_{T1} (R_{T2}). This means that faults f1 and f2 are indistinguishable when testbench T1 is applied while distinguishable if testbench T2 is applied. In other words, while either testbench

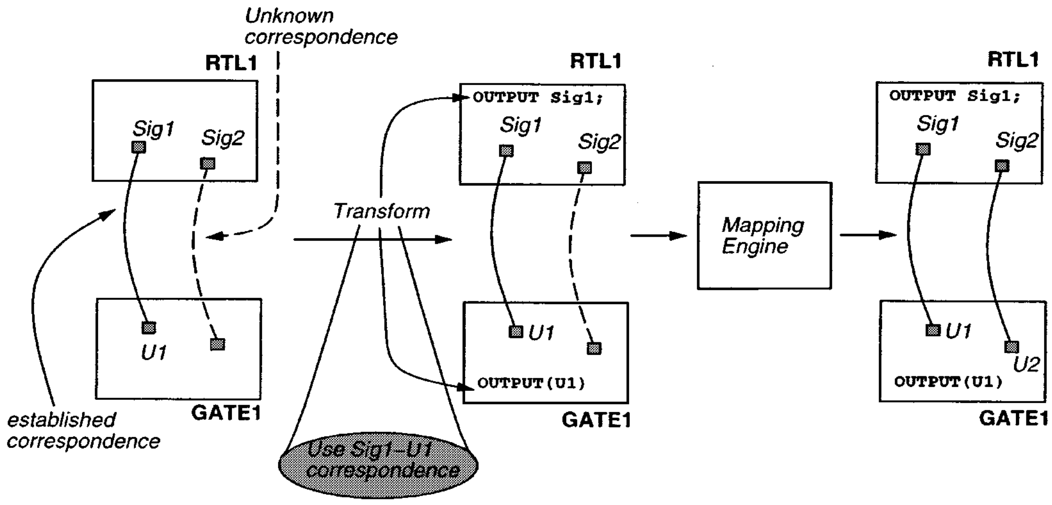

Fig. 13. Iterative strategy for enhancing the effectiveness of the mapping solution.

$\mathrm{T1}$ or $\mathrm{T2}$ is suitable for fault detection, only testbench $\mathrm{T2}$ facilitates diagnosis.

Diagnostic test generation [15] provides a means for deriving a test set that can facilitate both fault detection as well as subsequent diagnosis. Test vectors generated using diagnostic test generators are the ideal candidates for our methodology because they can potentially resolve the mapping problem with one stuck-at transformation. However, empirical evidence indicates that a gate-level sequential test generator like HITEC [7] that does not explicitly target diagnosis is able to achieve comparable precision.

For many designs (with high or low fault coverage and/or diagnosability), we can iteratively improve the performance of the diagnosis-based mapping solution as illustrated in the following example.

Example 5: Consider the objective of determining the mapping solutions for signals $\mathrm{Sig1}$ and $\mathrm{Sig2}$ in the RTL circuit description $\mathrm{RTL1}$ (see Fig. 13). Assume that our diagnosis-based mapping engine first established correspondence of $Sig1$ with net U1 in the gate-level implementation $\mathrm{GATE1}$, and then failed to determine the gate-level equivalent net of $\mathrm{Sig2}$.

One way to get around the above bottleneck is to reuse in a meaningful manner the *established* correspondence, namely, that of $\mathrm{Sig1}$ and $\mathrm{U1}$. A simple step would be to make $\mathrm{Sig1}$ and $\mathrm{U1}$ primary outputs in $\mathrm{RTL1}$ and $\mathrm{GATE1}$, respectively (again, note that this transformation is for analysis performed by the mapping engine only). This improves the testability and diagnosability of the given circuit *without* violating existing correspondences in any way. The transformed $\mathrm{RTL1}$ and $\mathrm{GATE1}$ descriptions are now fed to the mapping engine to determine the gate-level equivalent net of $\mathrm{Sig2}$ (net $\mathrm{U2}$ in Fig. 13). ■

Generalizing the above strategy for the case when mapping solutions to multiple signals must be computed, a design can be modified at any iteration by the introduction of primary inputs and/or primary outputs at RTL signals and gate-level nets whose correspondence has already been established. For a given gate-level netlist, this transformation yields a new netlist that maintains a one-to-one correspondence with it, but with enhanced testability and diagnosability characteristics. Hence, the mapping engine now computes a testbench that can detect and identify a larger set of faults than before. Therefore, more RTL to gate-level correspondences can be established when the modified RTL and gate-level descriptions are used in the next iteration.

IV. Application to Incremental Synthesis

In this section, we detail how the mapping solution is applied to a practical problem like incremental synthesis. First, we outline our framework for incremental synthesis and then illustrate its working with the help of an example.

A. Overview

In Section I, we suggested a viable alternative for tackling ECs of RTL specifications that relies on identifying the *appropriate* portion that changes in the gate-level implementation due to a modification to the RTL specification (see Fig. 14). We can clearly do this if we can establish a correspondence between a signal of interest in the RTL specification ($\mathrm{RTL1}$) and a net in its gate-level implementation ($\mathrm{GATE1}$). This is the mapping problem we have illustrated in Section II.

With the mapping solution in the incremental synthesis framework, no synthesis effort is necessary for obtaining the new implementation $\mathrm{GATE2}$. We can now view incremental synthesis as a "*find-and-replace*" mechanism wherein we first *find* the region of interest $\mathrm{G(S1)}$ in $\mathrm{GATE1}$ using the mapping solution and then *replace* it with a new gate-level section $\mathrm{G(S2)}$. This "find-and-replace" mechanism can be implemented systematically as shown in Fig. 15 to realize $\mathrm{GATE2}$ as follows.

1) First, identify region $\mathrm{G(S1)}$ in $\mathrm{GATE1}$ corresponding to the EC made to $\mathrm{RTL1}$ using our solution to the mapping problem (Steps 1–3). For this purpose, we first extract using RTL constant propagation and redundancy analysis an envelope encompassing the EC. This simply can be the process boundary or the smallest boundary consisting of latched inputs and outputs (latch boundaries are

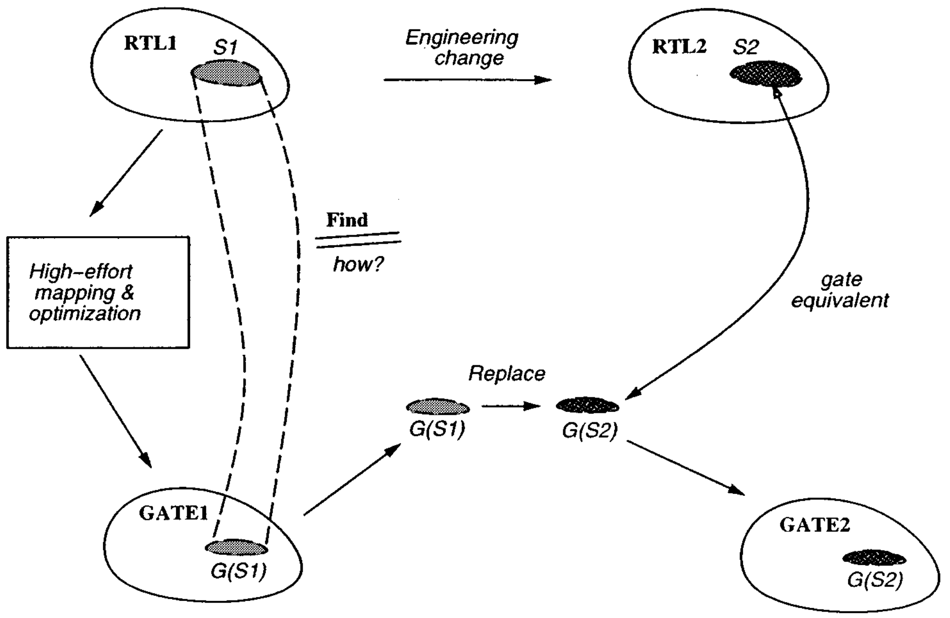

Fig. 14. Incremental synthesis paradigm.

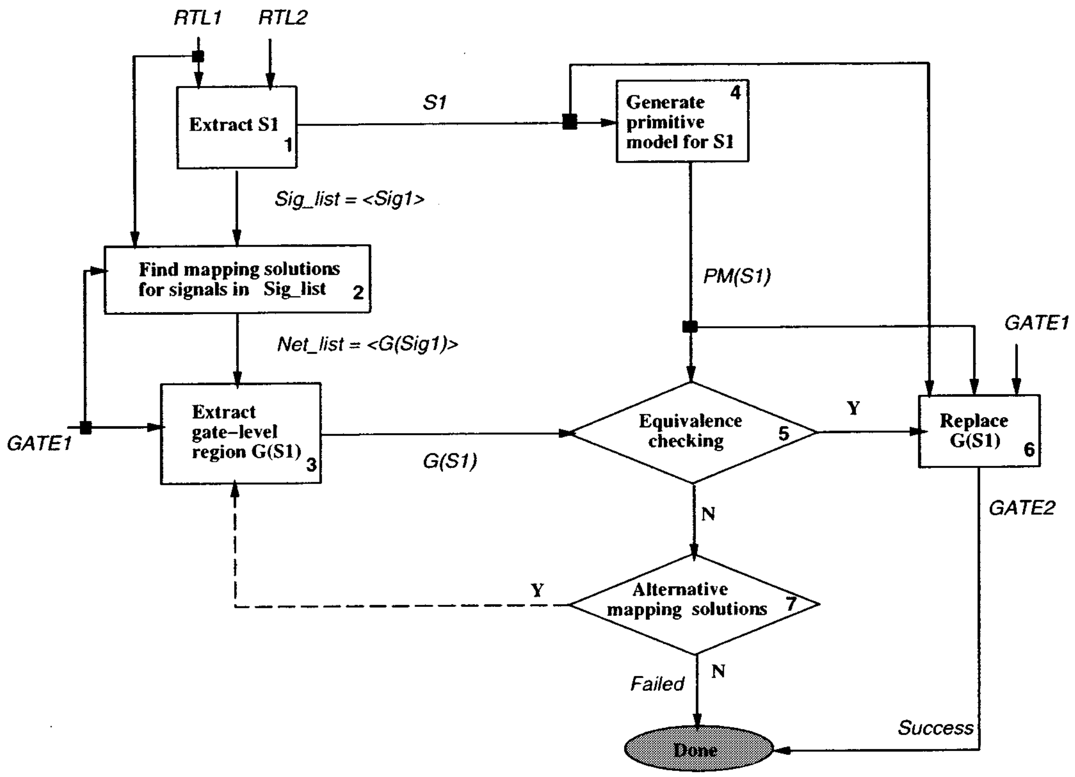
RTL1
RTL2
Extract S1 1
S1
Generate primitive model for S1 4
Sig_list = <Sig1>
Find mapping solutions for signals in Sig_list 2
PM(S1)
GATE1
Net_list = <G(Sig1)>
Extract gate-level region G(S1) 3
Equivalence checking 5
Y
Replace G(S1) 6
GATE1
G(S1)
GATE2
N
Y
Alternative mapping solutions 7
Failed
N
Success
Done

Fig. 15. Incremental synthesis. Detailed flow.

often used by formal verification techniques [16], [17] to reduce sequential logic verification to a combinational equivalence check). We call this list of inputs and outputs Sig_list and generate the mapping solutions for the different signals using the framework in Section II. This gives us G(S1) in GATE1.

2) Next, generate a *primitive model* PM(S1) for S1 (Step 4) and verify that the identified region G(S1) in GATE1 is equivalent to it (Step 5). PM(S1) is a module-level description of S1 and can assume any equivalent form. Its purpose is to have a structural or Boolean description that our equivalence checker uses to compare with the identified G(S1) for equivalence.

3) Having established the above correspondence, replace G(S1) with the gate-level implementation of PM(S1) in GATE1 to synthesize GATE2.

```
0 Proc1: process(control)
1 begin
2    case control is
3       when 3 => offset <= "0010";
4       when 6 => offset <= "0100";
5       when others => offset <= "0000";
6    end case;
7 end process Proc1;
```

Fig. 16. $Ex1$ with the desired EC.

If the equivalence checker fails (Step 5), we examine if an alternative mapping solution is possible and reapply the previous steps.

B. Incremental Synthesis Example

In this section, we demonstrate how a find-and-replace-based incremental synthesis framework can be used effectively to carry out ECs of RTL specifications.

Example 6: Consider again the VHDL code fragment $\mathrm{Ex1}$ shown in Fig. 7. Since the rest of the code is not shown, assume that $\mathrm{Proc1}$ contains the only assignments to offset in the entire specification and also that $\mathrm{control}$ is a three-bit signal. Consider the design error where offset in line 3 is erroneously assigned a "0011" value instead of a "0010" value. Then, the EC in question is the transformation of line 3 in Fig. 7 to that shown in Fig. 16. ■

Clearly, the extent of change affected by the EC shown in Fig. 16 is likely to be small. If $\mathrm{Proc1}$ forms a portion of a medium-sized design with around 10 000 lines of VHDL code, then the EC shown affects less than 0.01% of the RTL description. In the following example, we will show how the different steps in Fig. 15 can be systematically applied to carry out the EC introduced in Example 6.

Example 7: Consider again the EC discussed in Example 6 assuming that $\mathrm{Proc1}$ is the extracted $\mathrm{S1}$ in Fig. 15. Therefore, $\mathrm{control}[2:0]$ and $\mathrm{offset}[3:0]$ are the RTL signals for which we apply our solution to the mapping problem (Section II). The mapping solution yields the region $\mathrm{G(Proc1)}$ shown in Fig. 17, where $\_N21$ is the gate-level net corresponding to $\mathrm{control}[2]$, and so on.

We are now ready to verify that the marked region in the gate-level netlist indeed corresponds to $\mathrm{Proc1}$ in the RTL description. For this purpose, we generate the primitive model for $\mathrm{Proc1}$, $\mathrm{PM(Proc1)}$, as shown in Fig. 18. This primitive model and the extracted "sea_of_gates" from Fig. 17 form the inputs to our equivalence checking engine which results in "affirmative" verification.

We then apply the EC to the primitive model in Fig. 18. In other words, we modify constant "0011" to "0010" in Fig. 18. A gate-level description is then synthesized from the modified primitive model, which then replaces $\mathrm{G(Proc1)}$ shown in Fig. 17. This completes incremental synthesis. ■

V. Experimental Results

The techniques described in this paper were evaluated within the framework of an inhouse synthesis flow with the help of five example designs. The examples were chosen from a suite of Fujitsu designs used in telecommunication and networking applications. A designer-specified set of mapping instances in these examples were then selected for analysis. We applied the diagnosis-based solution methodology (Section III) to find the corresponding gate-level nets. Once the mapping solutions were determined, we validated them using the equivalence checker [18].

Typical synthesis of a circuit in our set-up proceeds as follows. At the top-level, the Synopsys Design Compiler [12] takes in the RTL specification of the circuit as its input along with the specified constraints. High-effort synthesis is then used to optimize the design for the specified constraints. The output of Design Compiler is then customized and technology-mapped using an industrial cell library to a gate-level netlist. Then, iterative improvement procedures are used at the logic level to enhance the quality of the final gate-level netlist for area and/or delay constraints. Most nontrivial name correspondences between the RTL signals and the gate-level nets except those of the design's inputs and outputs are lost at this stage.

Table I describes the characteristics of the different benchmarks. Circuit GPIO is a general-purpose input–output controller that is used as an interface circuitry in system chips. $\mathrm{ALM\_sw1}$ is an asynchronous transfer mode switch part while $\mathrm{EXE\_MEM}$ is a memory controller. $\mathrm{B\_PACK}$ and $\mathrm{ATH\_NET}$ form portions of popularly used chip sets. Columns 2 and 3 indicate the size of the specifications and the number of signals at the RTL. Columns 4 and 5 give postsynthesis gate-level statistics in terms of the number of flip-flops and number of nets, respectively. Columns 6, 7, and 8 report the results of test generation (fault coverage, number of test vectors, and test generation time) obtained from running the gate-level sequential test generator HITEC [7] on the different circuits. Note that HITEC generates test vectors only for fault detection and does not explicitly target fault diagnosis. The central processing unit (CPU) time was measured on a 360-MHz UltraSparc 60 workstation with 512-MB dynamic random access memory.

Our experiments to study the effectiveness of the diagnosis-based solution framework consisted of first selecting a designer-specified set of mapping instances for analysis and then applying our diagnosis-based mapping solution. The results are summarized in Table II. Columns 2 and 3 describe the mapping instances and their classification into the functionality (*Data* bit or *Control* bit) that they describe. For example, mapping instance $B5$ is a parity control bit set in a nested conditional construct. Mapping instance $A2$ is a data bit assigned values in a case construct. This example was specifically selected since a design error in one of the assignments to $A2$ made it a suitable candidate for the incremental synthesis procedure described in Fig. 15. For any mapping instance, both SA0 and SA1 transformations were applied to the RTL signal for generating the faulty responses using HITEC-generated test vectors. Column 4 reports the number of VHDL lines affected by carrying out an SA0 or SA1 transformation. A state-of-the-art fault diagnosis engine [19] was then used to determine the gate-level net (or nets) corresponding to the RTL signal. This engine exploits three-valued fault simulation to perform efficient fault diagnosis. This is important because faulty responses can contain some X values and

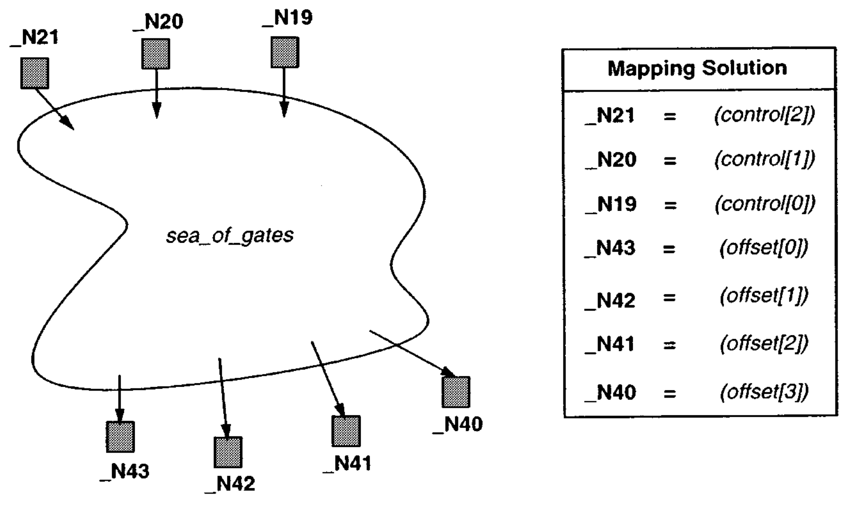

Fig. 17. $G(\mathrm{Proc1})$ extracted from the original gate-level netlist.

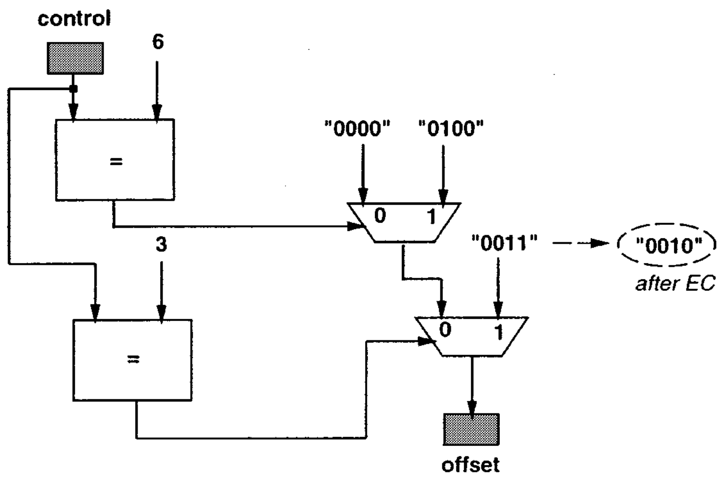

Fig. 18. Primitive model of Proc1.

TABLE I
CIRCUIT CHARACTERISTICS

Circuit	# VHDL lines	# Sig	# FFs	# Nets	FCov. (%)	# TVecs	TGen (s)
GPIO	808	97	148	1937	99.39	1396	56
ALM_sw1	3511	316	1490	17216	98.38	1446	492
B_PACK	6469	906	424	6229	98.65	752	188
EXE_MEM	4157	623	939	14424	96.64	979	338
ATH_NET	26566	1452	869	13906	99.42	4277	287

these must be considered by the diagnosis engine for accurate fault location. Column 5 indicates the total CPU time taken by the diagnosis scheme for determining the mapping solution. In Column 6, we indicate the size of the mapping solution in terms of the number of faulty gate-level nets common to the equivalence classes returned by the SA0 and SA1 transformations. None of the mapping solutions had shared names relating the RTL signals and the gate-level nets.

The average number of nets returned by our mapping engine for the different mapping instances is 1.8. Expressed as a percentage of the total number of nets in each case, this number amounts to only 0.028%. We performed an additional experiment with example GPIO. We carried out the mapping process for all the RTL signals and found that gate-level mappings for 95 of these signals (98%) were returned by our engine.

With the average diagnosis time being 71.1 s, these results demonstrate that our solution is fast and highly accurate for the example designs. The technique proved to be highly effective in identifying correspondences in complex control logic even for large designs. For example, the gate-level equivalent for $D2$, which is a start-of-data-transfer flag in the EXE_MEM memory controller example required only 39.7 s for the identification of a net among 14 424 existing nets (see Table I). Formal verification techniques, in contrast, cannot handle the size of the cir-

TABLE II
MAPPING SOLUTION STATISTICS

Circuit	Mapping Inst.	Description	# Transformed VHDL lines	# DiagTime. (s)	SolnSize # nets	Shared Names
GPIO	A1	Control	2	6.0	2	None
	A2	Data	22	16.7	1	
	A3	Data	2	10.1	1	
	A4	Data	2	9.7	1	
	A5	Control	1	4.0	1	
ALM_sw1	B1	Control	1	71.2	2	None
	B2	Control	1	66.8	2	
	B3	Data	5	65.3	1	
	B4	Data	5	68.4	1	
	B5	Control	6	65.5	2	
B_PACK	C1	Control	3	12.8	3	None
	C2	Control	2	12.2	3	
	C3	Control	3	13.4	3	
	C4	Data	2	12.5	2	
	C5	Data	2	13.8	2	
EXE_MEM	D1	Control	2	75.6	1	None
	D2	Control	2	39.7	3	
	D3	Data	4	34.5	1	
	D4	Data	3	34.3	2	
	D5	Data	3	31.8	2	
ATH_NET	E1	Control	1	158.5	1	None
	E2	Control	1	231.0	4	
	E3	Control	1	262.6	2	
	E4	Control	1	209.5	1	
	E5	Control	1	232.1	1	

cuits considered due to the well-known state-space explosion problem.

Also noteworthy is the extremely small number of lines in the RTL specification that needed to be changed (temporarily) in order to perform our transformations (the least and most number of lines transformed were 1 and 22, respectively). The above observations clearly demonstrate the effectiveness and ease-of-use of our mapping technique on typical design styles and sizes. Note that the approach is precise by construction using three-valued RTL simulation as well as fault diagnosis. Its performance is, therefore, constrained only by the testability of the gate-level net under SA0 and SA1 faults. In that sense, the overall testability of the original implementation greatly influences the success of this methodology.

VI. Conclusion

This paper addresses an important problem in hierarchical design flows, namely, the mapping problem of establishing correspondences between RTL signals and gate-level nets. We contribute an efficient solution that reduces the mapping problem to a diagnosis query, thereby enabling the application of a rich set of techniques developed in the area of circuit diagnosis to this problem. Our experiments with industrial circuits clearly demonstrate the efficiency and resolution with which the mapping problem can be solved in practice and applied to an application like incremental synthesis. We believe that techniques such as ours that transform one level of the hierarchy and analyze potential transformations that result in similar changes at a different level of hierarchy will not only increase in popularity, but will be key in establishing signal correspondences in future design flows.

References

[1] D. Brand, A. Drumm, S. Kundu, and P. Narain, "Incremental synthesis," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 14–18.

[2] "ECO Compiler: Technology Backgrounder," Synopsys, Inc., Mountain View, CA, Tech. Rep. UCB/ERL M96/22, Apr. 1997.

[3] C. Lin, K. Chen, S. Chang, M. Marek-Sadowska, and K. T. Cheng, "Logic synthesis for engineering change," in *Proc. Design Automation Conf.*, June 1995, pp. 647–652.

[4] G. Swamy, S. Rajamani, C. Lennard, and R. K. Brayton, "Minimal logic re-synthesis," Univ. California, Berkeley, CA, 1996.

[5] "Formality: Methodology Backgrounder," Synopsys, Mountain View, CA, Feb. 1998.

[6] M. Abramovici, M. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. Piscataway, NJ: IEEE Press, 1990.

[7] T. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *Proc. Eur. Design Automation Conf.*, Feb. 1991, pp. 214–218.

[8] I. Pomeranz and S. M. Reddy, "On the generation of small dictionaries for fault location," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 272–279.

[9] E. M. Rudnick, W. K. Fuchs, and J. H. Patel, "Diagnostic fault simulation of sequential circuits," in *Proc. Int. Test Conf.*, Oct. 1992, pp. 178–186.

[10] V. Boppana and W. K. Fuchs, "Integrated fault diagnosis targeting reduced simulation," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1996, pp. 267–271.

[11] E. J. McCluskey and E. Clegg, "Fault equivalence in combinational logic networks," *IEEE Trans. Comput.*, vol. 20, pp. 1286–1293, Nov. 1971.

[12] "Design Compiler: Technology Backgrounder," Synopsys, Mountain View, CA, June 1997.

[13] J. D. Giacomo, *VLSI Handbook*. New York: McGraw-Hill, 1988.

[14] E. J. McCluskey and E. Buelow, "IC quality and test transparency," in *Proc. Int. Test Conf.*, Oct. 1998, pp. 295–301.

[15] I. Hartanto, V. Boppana, J. H. Patel, and W. K. Fuchs, "Diagnostic test pattern generation for sequential circuits," in *Proc. VLSI Test Symp.*, Apr. 1997, pp. 196–202.

[16] J. Mohnke, "A signature-based approach to formal logic verification," Ph.D. dissertation, Univ. Halle, Halle, Germany, 1999.

[17] J. R. Burch and V. Singhal, "Robust latch mapping for combinational equivalence checking," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1998, pp. 563–569.

[18] *ASSURE User Manual*, Fujitsu Labs of America, Sunnyvale, CA, Dec. 1998.

[19] V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and P. Bollineni, "Multiple error diagnosis based on X-lists," in *Proc. Design Automation Conf.*, June 1999, pp. 660–665.

Srivaths Ravi received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, India, in 1996 and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, in 1998 and 2001, respectively.

He is currently a Research Staff Member with NEC Computer and Communications Laboratories, Princeton, NJ. His current research interests include testing and synthesis of digital circuits.

Dr. Ravi received the Siemens Medal from the Indian Institute of Technology in addition to various awards at the 1998 and 2000 International Conference on VLSI Design.

Indradeep Ghosh (S'94–M'98) received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, India, in 1993 and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, in 1995 and 1998, respectively.

He is currently a Member of Research Staff with the Advanced Computer-Aided Design Research Group, Fujitsu Laboratories of America, Sunnyvale, CA. He has authored or coauthored more than 20 technical papers in journals and conferences. His current research interests include design for testability, high-level test generation, built-in self-test, high-level synthesis, and high-level design verification and diagnosis.

Dr. Ghosh received the Honorable Mention Award at the International Conference on VLSI Design in 1998. He is the Audio–Visual Chair of the Tutorials Group of the IEEE Computer Society Test Technology Technical Committee.

Vamsi Boppana received the B.Tech. (hons) degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, India, in 1993 and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Illinois at Urbana-Champaign in 1995 and 1997, respectively.

He is a Cofounder and Principal Engineer of Zenasis Technologies, a VLSI design company creating leading-edge automated transistor-level optimization technologies. He has authored or coauthored over 35 technical papers and has five patent-pending inventions. He has served on the program committee of the Asia South Pacific Design Automation Conference and as a Session Chair at several computer-aided design and test conferences, including the International Conference on Computer-Aided Design. His current research interests include all aspects of VLSI design, test, and verification.

Dr. Boppana received the Indian Institute of Technology TCS Best Project Award in 1993, the University of Illinois Van Valkenburg Fellowship in 1995, the Best Paper Award at the VLSI Test Symposium in 1997, and a Fujitsu Laboratories of America Intellectual Property Contribution Award in 1999.

Niraj K. Jha (S'85–M'85–SM'93–F'98) received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1981, the M.S. degree in electrical engineering from the State University of New York, Stony Brook, in 1982, and the Ph.D. degree in electrical engineering from the University of Illinois, Urbana, in 1985.

He is currently a Professor of Electrical Engineering with Princeton University, Princeton, NJ. He is also the Director of the Center for Embedded System-on-a-Chip Design, funded by the New Jersey Commission in Science and Technology. He was the Program Chair of the 1992 Workshop on Fault-Tolerant Parallel and Distributed Systems. He is currently the Editor of the *Journal of Electronic Testing: Theory and Applications* (JETTA) and has served as a Guest Editor for a JETTA Special Issue on High-Level Synthesis. He has authored or coauthored two books entitled *Testing and Reliable Design of CMOS Circuits* (Norwell, MA: Kluwer, 1990) and *High-Level Power Analysis and Optimization* (Norwell, MA: Kluwer, 1998) and more than 180 technical papers. His current research interests include computer-aided design of integrated circuits, low-power design, digital system testing, and distributed computing.

Dr. Jha received the AT&T Foundation Award, the NEC Preceptorship Award for research excellence, and the Best Paper Award at the International Conference on Computer Design in 1993, the Fault-Tolerant Computing Symposium in 1997, the International Conference on VLSI Design in 1998, and the Design Automation Conference in 1999. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: ANALOG AND DIGITAL SIGNAL PROCESSING and IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS.