Hierarchical Error Diagnosis Targeting RTL Circuits

Vamsi Boppana, Indradeep Ghosh, Rajarshi Mukherjee, Jawahar Jain and Masahiro Fujita

Fujitsu Laboratories of America
Sunnyvale, CA 94086
{vboppana,ighosh,rmukherj,jawahar,fujita}@fla.fujitsu.com

Abstract

Diagnosis algorithms targeting design errors in RTL circuit descriptions are presented in this paper. The algorithms presented exploit the hierarchy available in RTL designs to locate design errors. Xlists [1] are shown to be useful to capture the effects of design errors within components of RTL designs. Information from the simulation of Xlists is used to systematically diagnose components in error. Experiments are performed on RTL benchmark circuits using a prototype that we have developed to demonstrate the rapid and accurate location of errors. They also show that diagnosis at the RTL offers a significantly superior alternative to diagnosis at the gate-level both in terms of diagnostic accuracy and computational efficiency.

1. Introduction

Debugging/Diagnosis represents a dominant cost in design development. Diagnosis is typically performed after a failed verification step. It may involve formal or simulation methods or a combination of both techniques. Rapid and accurate identification of bugs helps reduce design turn-around time significantly.

Much of the previous work on error diagnosis has primarily been targeted at the gate and lower levels of design [1–4]. However, most design activity presently takes place at the RTL and it is difficult to relate errors at the RTL to errors at a lower level. In fact, a relatively simple error at the RTL may translate into extremely complex errors (most often into multiple errors) at a lower level. Hence, it is critical to address the diagnosis problem at the RTL.

Tamura [5] presented a method for using HDL information in a single error diagnosis algorithm. Diagnosis based on CDFGs has also been discussed in recent work [6, 7]. Practical experience with the use of currently available diagnosis tools on state-of-the-art processors has been discussed [8, 9]. In addition, methods on improving the debuggability of the design have also been proposed [6]. Also useful to note is the relatively more comprehensive treatment of the problem of debugging in the software compiler domain that has resulted in powerful and widely used tools such as Dbx [10]. However, significant problems remain with the state-of-the-art diagnosis solutions applicable to large RTL designs.

The problems occur both because of the difficulty involved in modeling RTL errors (that impedes the progress of cause-effect diagnosis [11] techniques) and because of the difficulty involved in the development of model-independent, effect-cause, techniques (like path-tracing) to RTL designs. Simulation-based techniques similar to gate-level techniques that enumerate possible erroneous values at internal nodes are not easy to apply to the RTL because of the large multitude of binary logic value combinations that each RTL object (for example, a word) can assume in the event of an error. The problem is even more serious because multiple RTL objects are often influenced by typical RTL bugs.

One other application of diagnosis for design errors can be in the domain of designs with soft intellectual property (IP) cores or legacy cores which are mostly available at the RTL. These designs are becoming popular to cut design turn around times. The cores can be designed by a third party as in the case of soft IP or in-house as in case of legacy cores. Very often these designs require certain fine tuning to customize them to fit a new environment. However, in a new design, people who are customizing the core might have limited knowledge about the complete functioning of the core. This might lead to inadvertent introduction of bugs at different points in the design during customization. With a proper diagnosis framework as presented in this paper it will be easy to pinpoint the component that has a design error without completely understanding the design.

In this paper, we present a solution to the problems discussed above by exploiting the modeling power of the recently proposed Xlist model [1]. In that work, it was shown that Xlists can capture the effects of all possible errors at any set of nodes in the design and gate-level diagnosis algorithms were designed around that observation. In this work, we observe that Xlists can be used not only to capture the effects of all possible errors at a given set of nodes, but also to capture the effects of all possible errors *within* any component of the design. This observation, coupled with the fact that RTL designs are typically hierarchical in nature (and are hence amenable to modular diagnosis) gives rise to a powerful diagnosis framework. Our main contributions are:

- Generation and use of Xlists to capture the effects of all possible errors within RTL components.
- Systematic, hierarchical diagnosis algorithm based on analyzing the responses of Xlists at the RTL.

2. Perspective

Figure 1 illustrates the problem being solved. We are given a buggy implementation and a specification of a circuit along with a failing test set. The implementation is a hierarchical RTL circuit with design errors (say in leaf component *A*. By processing the obtained responses and the expected responses for the test set we want to identify the leaf

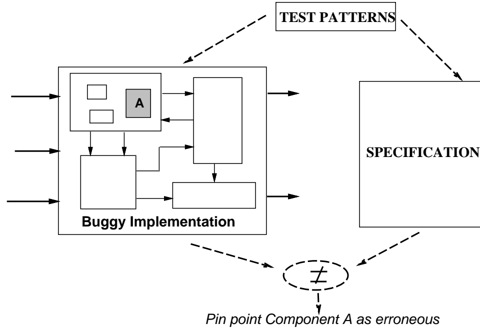

Figure 1: Diagram depicting diagnosis problem

component *A* as the erroneous module.

Before going into the details of Xlists which we have used to solve the problem, we state the assumptions being made. In addition to the buggy RTL that is given to us we assume that the RTL description is cycle accurate. We assume that the test set available to us excites the bugs in the circuit and propagates the effect to primary outputs. Further, we assume that we are given the expected responses at the primary outputs for each time frame. This can be obtained either from a specification or by simulating a predesigned RTL that is assumed to be correct. The buggy RTL may have resulted from the correct RTL during fine tuning or optimization. We assume that only primary inputs are controllable and primary outputs are observable. Though a lot of intermediate outputs may be added at this level of abstraction, it is not always easy to construct the expected responses at these points.

3. Background

Xlists have been used to provide improved fault diagnosis when compared with conventional fault models (like stuck-at/bridging) because of their ability to capture the effects of arbitrary defects at circuit nodes [1].

Definition:*A set of nodes whose actual values would be replaced during simulation by the value X (don't care), to capture the effects of any error completely contained in any* **dominance** *region of this set of nodes, is called an* **Xlist** *[1].*

Example: The following example illustrates the use of Xlists to capture arbitrary errors. Let there exist some design error(s) within nets *a*, *b* and *c* of a circuit. The errors may be as simple as a polarity inversion error (for example, *a* should actually be *not(a)*) or could be an arbitrarily complex function spanning all three nodes. The definition of an Xlist {*a, b, c*} can be used to capture all such errors. Introduction of the Xlist {*a, b, c*} implies three-valued simulation of the failing vector set to obtain primary output responses, with the values of *a*, *b* and *c* replaced (during simulation) by the values *X*, *X* and *X*. This replacement guarantees the capture of all possible errors at these nodes by virtue of covering the effects of all possible erroneous functions ((0,0,0), (0,0,1), ..., (1,1,1)) at these nodes.

A diagnosis procedure using a typical matching-based scoring scheme was designed based on the use of Xlists. The procedure takes the buggy implementation, a set of Xlists, a set of failing vectors (vectors which when simulated on the circuit produces erroneous results at the primary outputs), and a set of expected responses at the primary outputs as its input. It then ranks Xlists based on their ability

```
// Each L is an Xlist of nodes
// MATCHCOUNT, PARTIALMATCHCOUNT,
// MISMATCHPENALTY and DROPTHRESHOLD
// are input parameters
// Match means a (0, 0) or (1, 1) combination
// Mismatch means a (0, 1) or (1, 0) combination
// Partialmatch means a (0, X) (1, X) combination
Procedure XlistDiagnose(Impl, Xlists, Vectorset, Spec)
{ for each Vector {
// Let CurrVec be the current vector
 for each Xlist L {
  if (Score[L] > DROPTHRESHOLD) {
    for each primary output {
    // Let CurrPo be the current primary output
      if (Match(Impl(L, CurrVec, CurrPo),
         Spec(CurrVec, CurrPo))) {
        Score[L] += MATCHCOUNT;
      elseif (PartialMatch (Impl(L, CurrVec,
        CurrPo), Spec(CurrVec, CurrPo))) {
        Score[L] += PARTIALMATCHCOUNT;
      else {// MISMATCH
        Score[L] -= MISMATCHPENALTY;
        }
      } } } } }
}
```

Figure 2: Procedure *XlistDiagnose*: ranking set of Xlists

to *explain* the observed errors. Explaining errors refers to the fact that simulation with X values at the nodes in an Xlist produces primary outputs that cover/contain (X contains both 0 and 1) the observed errors. The intuition behind the design of the procedure was that if even the introduction of X values at specific nodes of the circuit cannot explain an error, then those nodes cannot completely contain the error. A variation of that procedure is presented in Figure 2 here for completeness. The procedure also takes as input three parameters MATCHCOUNT, PARTIALMATCHCOUNT and MISMATCHPENALTY which may be used to run the procedure with varying degrees of confidence in the error model.

In the procedure, a three-valued simulation is run for the entire failing vector set for each Xlist. During the simulation any logic value appearing on a net in the Xlist is replaced with an X. After the simulation at each primary output for each time frame whenever there is a match in the logic value between expected and simulated, the score for the Xlist is incremented by MATCHCOUNT. Whenever there is an X in the simulated response, the the score for the Xlist is incremented by PARTIALMATCHCOUNT. Whenever there is a mismatch between the simulated and the expected response, the score is decremented by the MISMATCHPENALTY. This is because even Xs at the Xlist nets cannot explain the error and hence the Xlist is not a good candidate to search for the error.

A high confidence in the error model would imply that the diagnosis algorithm could be run in an *"exact"* mode with parameter values of 0, 0 and ∞, respectively. This is because if there is a mismatch between expected and simulated response even with Xs on the nets of a Xlist, then that Xlist may be discarded as a candidate to explain the error. On the other hand, only a moderate confidence in the error

Figure 3: ALU circuit illustrating component diagnosis

model would imply a run of the procedure in a *"heuristic"* mode with values of 10, 5 and 20, respectively. In this case we are trying to find out the Xlist that best explains the error as no single Xlist may be able to completely explain the error. Note that since this is a simulation based diagnosis technique it can also handle high impedance states or Z values in the circuit. In that case the RTL simulation needs to be four valued instead of three. We now show how we obtain a candidate set of nets to include in a Xlist and how we obtain different Xlists to do an effective hierarchical error diagnosis.

Capturing component error effects with Xlists: We first notice that the effect of *any* error on nodes internal to a component can be captured/covered by the introduction of X values at the outputs of that component. Hence, an Xlist generation scheme that covers the outputs of RTL components would be able to capture the effects of any component errors in the design. *i.e.* each Xlist will consist of all the outputs of a single component and there will be as many Xlists as there are components in the design.

Further, unlike a specific, logic value-based model (like the stuck-at or multiple stuck-at model), if an Xlist is generated containing the outputs of a component, then its effects at any primary output of the design will be guaranteed to contain/cover the effects of any component that is contained in the first component.

We illustrate the diagnostic reasoning with the help of Figure 3. The circuit consists of four components (comparator, adder/subtracter, multiplier and a multiplexer). Let us assume that there exists a design error in the multiplier component and that there exists a vector that detects the error in the multiplier. This implies that the select signal *CONTROL* of the multiplexer is set to 1.

Now, we ask the question: *Can the error be within the ADD/SUB component?* This question is answered by designing an Xlist consisting of all the output nets of the ADD/SUB component and simulating the design under the influence of that Xlist. We immediately notice that the error can still be observed at the output (because the select signal of the multiplexer is still set to the value 1). *Thus, the error cannot be within the ADD/SUB component*. This demonstrates that even without explicitly considering any error within the design, we were able to deduce a powerful conclusion about the location of errors in the design.

In obtaining the prior diagnostic inference, we have taken advantage of the fact that an Xlist at a component's outputs completely covers all possible errors within that component. Further diagnosis to locate the actual erroneous subcomponent within the multiplier can now take place without considering sub-components within the ADD/SUB component. We now develop the algorithms required to perform such a hierarchical diagnosis.

Figure 4: Hierarchical diagnosis example

4. Hierarchical Diagnosis using Xlists

In this section we illustrate the hierarchical error diagnosis algorithm using an example RTL circuit. Then we formalize the algorithm.

Single error diagnosis: Most RTL circuits of today are written in some high-level description language (HDL) like VHDL or Verilog. The design style is usually hierarchical with the top level entity consisting of an interconnection of various components. Each of these components can in turn be an entity consisting of an interconnection of lower level components. Thus the design is built up in a hierarchical manner from leaf nodes consisting of simple RTL functions.

Consider the example RTL circuit shown in Figure 4. The circuit has four primary inputs and three primary outputs some of which are buses. It consist of four RTL components - *COMP1, COMP2, COMP3, and COMP4*. The diagnosis algorithm proceeds in a hierarchical fashion. Initially, the Xlists are formulated as the intermediate outputs of the top level components. Thus we have four Xlists at the start of the diagnosis. In Figure 4, the first Xlist corresponding to the component *COMP1* consists of the nodes [*h(0), h(1), k, f(0), f(1)*], the second Xlist corresponding to component *COMP2* consists of nodes [*a(0), a(1), a(2), b(0), b(1), c, d*], and so on. After this, three-valued RTL simulation is run on the complete circuit and the matching and scoring for the error values is done for each Xlist as explained in the previous section. The *exact* option is used while doing the diagnosis. The simulation vector set used currently gives high stuck-at fault coverage at the logic level. Methods for generating high quality test sets from golden high-level specification descriptions have been described in previous work [12, 13]. Such an approach was chosen because generation of diagnostic test vectors was not considered in this work.

At this point, if the error is confined to a particular component, its Xlist will probably show up as being the only one

```
Procedure RTLDiag(Circuit R, Test Set T)
{
  C_list = R;
  while (C_list contains non-leaf nodes) {
    j = 0;
    for each component C in C_list {
      for each subcomponent C' within C {
        Xlist(j) = Outputs(C');
        j = j + 1; }
      }
    scorearray = XlistDiagnose(Xlist, R, T, exact);
    if !check(scorearray) {
      RTL_multCompDiag(R, T, C_list);
      exit;
      }
    C_list = NULL;
    for each Xlist(l) with scorearray(l) = 0 {
      C1 = subcomponent corresponding to Xlist(l);
      Add_to_C_list(C1);
      }
  }
}
```

Figure 5: RTL error diagnosis

having a zero score out of all the Xlists. This is because all mismatches with the expected values during the simulation can be explained by placing X values at the output of that particular component. Thus the algorithm has succeeded in isolating the error source to a top level component. Say this component is the shaded component *COMP2*.

Now the algorithm can be used to go deeper into the hierarchy to isolate the error site further. In the circuit of Figure 4, component *COMP2* in turn consists of three different components - *COMP2_1, COMP2_2, and COMP2_3*. The Xlists corresponding to the components are [*b(0), b(1), r1*], [*a(0), a(1), a(2), d, g1(0), g1(1)*], and [*c*], respectively. The diagnostic simulation is rerun with these three Xlists and the scores are examined to find the Xlist with zero score. Say that this is the second Xlist. Then the algorithm has further isolated the error to the shaded component *COMP2_2*. If this is a leaf cell as in this case, then the RTL code can be examined for errors. Otherwise the diagnosis can proceed by creating Xlists of components within *COMP2_2*.

Multiple error diagnosis: Note that there is no guarantee that after the diagnostic simulation only one Xlist will have a zero value. This can arise if (i) there are errors in multiple components, (ii) if an error spans over two or more components like an interconnection error, (iii) if the error effects of one component pass exclusively through another component before reaching the primary outputs, (iv) if an error in a different component can cause exactly the same behavior as the current error. In the first two cases no Xlist will have a zero value and the lower level diagnosis can proceed by considering the components whose output Xlists have higher scores, thus making them the most likely candidates to have errors. In the third and fourth case, X values at the output of either component can explain the error effects at the primary outputs. Hence, multiple Xlists can have zero values. In such cases diagnostic resolution can be enhanced by adding probe points at the outputs of the various components whose Xlists have zero value, as pseudo-primary outputs. This added observability is usually enough

```
Procedure RTLmultCompDiag(Circuit R, Test Set T,
                                            C_list)
{
  while (C_list contains non-leaf nodes) {
    j = 0;
    for each component C in C_list {
      for each subcomponent C' within C {
        Xlist(j) = Outputs(C');
        j = j + 1; }
      }
    scorearray = XlistDiagnose(Xlist, R, T, heuristic);
    S_list = Select_list(scorearray);
    C_list = NULL;
    for each Xlist(l) in S_list {
      C1 = subcomponent corresponding to Xlist(l);
      Add_to_C_list(C1);
      }
  }
}
```

Figure 6: Multi-component RTL error diagnosis

to pin down the error to a single component. These pseudo-outputs can be removed after error correction before logic synthesis. Note that adding pseudo primary outputs is only possible when the expected responses at those points can be obtained.

Hierarchical diagnosis algorithm: We are now in a position to formally describe the algorithm. Figure 5 gives the top level pseudocode of the algorithm. The algorithm takes as input an RTL circuit R and a test set T. Initially it puts the top level circuit into a component list *C_list*. The Xlists are then formulated as the output lines of each subcomponent of the components in the C_list. With these Xlists, the *exact* version of the XlistDiagnose algorithm is called. In this algorithm, three-valued simulation is done on R with T. There is a penalty of 100 for any mismatch between the true and the expected value. Thus the Xlist which is able to explain all possible errors has a score of 0 and this is reflected in the *scorearray* that is returned. The procedure *check* returns a true value if there are one or more 0 scores in the scorearray. If there are no 0 scores in the array then no definite diagnostic resolution has been obtained and a heuristic procedure called *RTLmultCompDiag* is used to identify the most likely places of error. This procedure is explained later. However, if there are 0 scores in the array then the algorithm goes deeper into the hierarchy and formulates a new *C_list* with the subcomponents whose Xlist had a score of 0. This is done is procedure *Add_to_C_list*. The hierarchy level is lowered in this manner in an iterative manner until there are only leaf components in the *C_list* and the error cause has been narrowed down to a leaf component.

If during the process of diagnosis, a scorearray is obtained with no 0 values, then from that point the algorithm *RTLmultCompDiag* takes over. This is shown in Figure 6. In this case the procedure *XlistDiagnose* is run with a heuristic where there is a score increment for an Xlist if there is a match, a score decrement if there is a mismatch and a lower score increment if there is a partial match between the expected simulation result in a cycle and a simulation with X values on the nodes of the Xlist. A few Xlists with the highest scores in the scorearray are then chosen for further diagnosis. The components corresponding to these

Table 1: Circuit details

Circuit	Type	Bit Width	HDL lines	Gates	FFs	Components		Vecs
						Level1	Level2	
Paulin	Data Flow	4	624	826	32	5	24	1132
GCD	Ctl. Flow	16	396	888	50	5	4	587
ALU	Combinat.	32	262	19532	-	3	1000	34

Xlists are the most likely candidates for having errors. This selection is done in the procedure *Select_list* and the number of Xlist to be chosen can be a user defined parameter. Thus in this case we end up with a number of leaf nodes which have the highest likelihood of having the error.

5. Experimental Results

In this section, we will discuss experiments performed on RTL benchmark circuits to demonstrate the performance of the hierarchical diagnosis algorithm and to compare it with gate-level diagnosis algorithm. All our experiments were performed on a SPARCstation20 with 64MB of memory.

Three circuits were used in our diagnosis experiments. Details about these circuits are shown in Table 1 and are also available amongst other work from previous work targeting high-level test generation [12]. *Paulin* is a data-flow intensive circuit popularly used in the literature, *GCD* is a control-flow intensive circuit that computes the GCD of two numbers by iterative subtraction and *ALU* is a combinational circuit. Each of these circuits have two levels of hierarchical components. Table 1 lists details about the circuits. Listed in the order of columns are the name of the circuit, type of the circuit, bit width used in the design, number of lines of HDL code used for synthesis, number of gates in the resulting gate-level design, number of FFs in the design, number of components at each level of hierarchy in the design and the number of vectors used for diagnosis. Since test generation was not targeted in this work, test vectors generated from high-level test generation [12, 13] were used.

Four kinds of typical RTL coding errors were introduced into our RTL descriptions and diagnosed. Wrong/missing/additional logic (single/multiple changes) in a randomly selected component, logic changes in the controller state machine/control (single/multiple changes), multiple component errors and interconnection (single/multiple changes) errors in a component were studied. These are referred to as *Component, Control, Mul. component, Interconnect* errors, respectively.

Results of the hierarchical diagnosis algorithm for the diagnosis of each kind of error are presented in Table 2. The table lists in the order of the columns, the name of the circuit, type of the error being diagnosed, ranks of the component containing the error in level1 (lev1) and level2 (lev2) and the time taken for diagnosis at each level. A '-' entry indicates that diagnosis was only relevant for only one level of hierarchy (for example, the controller, consisting of only random logic, did not have any sub-components).

We have obtained exact diagnosis at level1 in each of the diagnosis instances. Further, in all cases except that of multiple component errors, perfect diagnosis was achieved, i.e., the erroneous component was identified at each level. Even for the case of multiple component errors, at least one erroneous component was identified as a top candidate in two instances and it was identified as the second best candidate in the other instance. Note that the maximum time taken for any of our diagnosis algorithms was only 48.7 seconds. This time was spent on the the largest circuit *ALU* containing 19,532 gates.

For several errors, at level1, even though the error was not in the controller, the controller component also assumes the highest score during the *exact* mode $(0, 0, \infty)$ of the algorithm. This is because if the output of the control part is X, then large portions of the designs assume unknown values and hence many primary outputs assume the X value and explain the error. The problem, however, was easily overcome in each of the cases by simply running the algorithm in the *heuristic* mode that distinguishes partial matches from exact matches using parameters (10, 5, 1000). The intuition here is that an Xlist consisting of the outputs of the controller would produce far more partial matches than the true error component and hence does not explain the error as well as the true error component.

Hierarchical vs. gate-level diagnosis: We now present results from an experiment comparing our hierarchical diagnosis algorithms with gate-level diagnosis. For this purpose, a state-of-the-art, gate-level, diagnosis algorithm [14] capable of locating local, multiple errors was used. This algorithm is capable of rapidly and accurately locating multiple errors if they lie within a fixed structural distance of any node in the circuit. For example, an error of multitude 10 was accurately diagnosed in approximately 23 seconds in circuit c7552 from the ISCAS 85 benchmark circuits containing approximately 3800 nodes. Note however, that this algorithm does not use any information about the hierarchy of the design; it operates on a flat netlist.

Errors were introduced into the RTL code in a manner so that synthesis creates a multiple error situation consisting of nodes within a fixed structural distance of a specific node in the gate-level netlist in order to be able to apply both RTL and gate-level diagnosis algorithms. The RTL diagnosis algorithm operates as described previously and its results are examined to note the rank of the actual error component amongst all components. Also noted is the size of the code that is identified to be in error. Results of the gate-level algorithm are examined to note the rank of the node within a fixed distance of which the actual error is located.

Table 3 presents results of the comparison. The columns in the table represent the name of the circuit, type of the error, rank of the actual error component in the list of ranked components (size of the erroneous HDL code/total code size), rank of the targeted error node, time taken for hierarchical diagnosis (sum of both level1 and level2 times) and the time taken for gate-level diagnosis.

The data clearly indicates that performing diagnosis at the RTL is beneficial both in terms of the quality of diagnosis and in terms of the time taken for diagnosis. We first note that in spite of making an effort to introduce the error

Table 3: Hierarchical versus gate-level

Circuit	Error Type	Diagnosis Quality		Diagnosis Time (sec)	
		Hierarchical	Gate-level	Hierarchical	Gate level
		Erroneous Xlist Rank (Err. HDL lines/Tot. lines)	Error Xlist Rank (Total xlists)		
Paulin	Component	1 (9/624)	8 (800)	8.4	974.7
Paulin	Interconnect	1 (27/624)	84 (800)	13.4	602.2
GCD	Component	1 (33/396)	9 (888)	3.2	368.1
GCD	Interconnect	1 (9/396)	23 (888)	3.3	368.3
ALU	Component	1 (9/262)	1† (19532)	48.8	402.4
ALU	Interconnect	1 (9/262)	2‡ (19532)	45.9	2809.1

† 538 ties which could not be resolved even in a second run
‡ 5 ties

Table 2: Results of hierarchical diagnosis

Ckt.	Error Type	Erroneous Xlist Rank(s)		Diagnosis Time (sec)	
		lev1	lev2	lev1	lev2
Paulin	Component	1†	1	7.1	1.2
Paulin	Control	1	-	3.0	-
Paulin	Mul. component (2 errors)	1†	2, 9	8.0	9.6
Paulin	Interconnect	1†	1†	8.1	4.8
GCD	Component	1†	-	3.2	-
GCD	Control	1	-	1.5	-
GCD	Mul. component (2 errors)	1†	1, 2	2.0	1.5
GCD	Interconnect	1†	-	3.3	-
ALU	Component	1	1	7.4	41.4
ALU	Control	1	-	8.1	-
ALU	Mul. component (3 errors)	1	1, 56, 188	8.0	48.7
ALU	Interconnect	1	1 (14 ties)	7.3	38.6

† indicates an initial tie with the controller component that was resolved in a second run of the diagnosis routine. Run times shown include the both steps

to be amenable to gate-level diagnosis, the accuracy of the gate-level diagnosis algorithm maybe inadequate. In contrast, since the error actually occurred at the RTL, *diagnosis results are exact for the RTL algorithm* for everyone of our errors. Portions of code identified to be erroneous do not exceed 33 lines in any of our diagnosis experiments. It is also worth noting the significantly smaller runtimes required by the hierarchical diagnosis algorithm. *The runtimes are at least an order of magnitude smaller in every case*. It is perhaps only intuitive to expect such a dramatic reduction in the runtimes because, unlike the gate-level algorithm, diagnosis at each level of hierarchy eliminates the need to examine large portions of the circuit at each step.

6. Conclusions

We presented effective algorithms for hierarchically diagnosing RTL circuits. The effectiveness of our algorithms was derived from the modeling power of Xlists that enabled the capture of all possible component errors during simulation and the use of hierarchical information available at the RTL. Exact or near exact diagnosis was achieved in all of our diagnosis experiments. Our experiments also demonstrated that the time required for diagnosis can be improved by at least an order of magnitude over flat, gate-level diagnosis for typical RTL errors. Further, valuable intuition on the behavior of errors and their migration across multiple stages of design was obtained during this work. Extensive experimentation on industrial RTL designs to study errors and their effects is part of future research plans. Efforts are also on to automatically generate good diagnostic test sets from RTL circuits.

References

[1] V. Boppana and M. Fujita, “Modeling the unknown! towards model-independent fault and error diagnosis”, in *Proc. Intl. Test Conf.*, Oct. 1998, pp. 1094–1101.

[2] M. Tomita, T. Yamamoto, Sumikawa F, and K. Hirano, “Rectification of multiple logic design errors in multiple output circuits”, in *Proc. Design Automation Conf.*, June 1994, pp. 212–217.

[3] S.-Y. Kuo, “Locating logic design errors via test generation and don’t-care propagation”, in *Proc. European Design Automation Conf.*, 1992, pp. 466–471.

[4] H-T. Liaw, J-H. Tsaih, and C-S. Lin, “Efficient automatic diagnosis of digital circuits”, in *Proc. Intl. Conf. Computer-Aided Design*, Nov. 1990, pp. 464–467.

[5] K. A. Tamura, “Locating functional errors in logic circuits”, in *Proc. Design Automation Conf.*, June 1989.

[6] M. Potkonjak, S. Dey, and K. Wakabayashi, “Design-for-debugging of application specific designs”, in *Proc. Intl. Conf. Computer-Aided Design*, Nov. 1995, pp. 295–301.

[7] M. Khalil, Y. L. Traon, and C. Robach, “Towards an automatic diagnosis for high-level design validation”, in *Proc. Intl. Test Conf.*, Oct. 1998, pp. 1010–1018.

[8] A. Kinra, A. Mehta, N. Smith, J. Mitchell, and F. Valente, “Diagnostic techniques for the ultrasparc microprocessors”, in *Proc. Intl. Test Conf.*, Oct. 1998, pp. 480–486.

[9] M. E. Levitt et. al., “Testability, debuggability and manufacturability features of the ultrasparc-i microprocessor”, in *Proc. Intl. Test Conf.*, Oct. 1995, pp. 157–166.

[10] M. A. Linton, “The evolution of dbx”, in *Proc. USENIX Conference*, 1990, pp. 211–220.

[11] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital System Testing and Testable Design*, New York, NY: Computer Science Press, 1990.

[12] I. Ghosh, A. Raghunathan, and N. K. Jha, “A design for testability technique of RTL circuits using control/data flow extraction”, in *Proc. Intl. Conf. Computer-Aided Design*, Nov. 1996, pp. 329–336.

[13] P. Vishakantaiah, J. A. Abraham, and D.G. Saab, “CHEETA: Composition of hierarchical sequential tests using ATKET”, in *Proc. Intl. Test Conf.*, Oct. 1993, pp. 606–615.

[14] V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and P. Bollineni, “Multiple Error Diagnosis Based on Xlists”, in *Proc. Design Automation Conf.*, June 1999, pp. 660–665.