Correcting Design Errors in Components and Connections

Johan de Kleer , Alexander Feldman , Ion Matei
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA
email: {dekleer, feldman, matei}@parc.com

Abstract

We present an approach for applying model-based diagnosis techniques for redesigning an almost correct design. Model-Based Diagnosis reasons backwards from undesired observations to identify which component(s) contribute to the symptom. Conversely, Model-Based Redesign reasons backwards from desired input-output behavior to identify which component(s) need to be changed to achieve that intent. They are facets of the same underlying task: identifying the causes for deviations from desired behavior. Debugging almost correct designs is a significant problem in product design. The approach makes Design Space Exploration easier because the search needs only to find a "close enough" design. It is also applicable to situations where the requirements to an existing system are slightly changed. The approach has been applied a large benchmark of digital circuits including higher cardinalities (24,619 single faults, 100,000s of multiple faults). We demonstrate our approach by correcting design errors arising from incorrect component choices as well as wiring errors. It generalizes to any discipline to which Model-Based Diagnosis has been applied (including digital and analog circuits, qualitative models, computer programs).

1 Introduction

On the face of it, design and diagnosis seem unrelated. Design is an act of creation to achieve some purpose. Diagnosis is an investigation to determine what went wrong. Design requires wide-ranging open world thinking while diagnosis requires meticulous examination of evidence preferably knowing the design. One usually thinks of knowledge of design helping diagnosis, not the converse. In this paper we show that many aspects of design and diagnosis are actually quite similar and that diagnosis can aid design as much as design can aid diagnosis.

Consider diagnosis (Figure 1). Some artifact is not working as intended. Usually we know a great deal about its design. When the artifact was originally constructed it functioned as intended. Over time some degradation occurred so

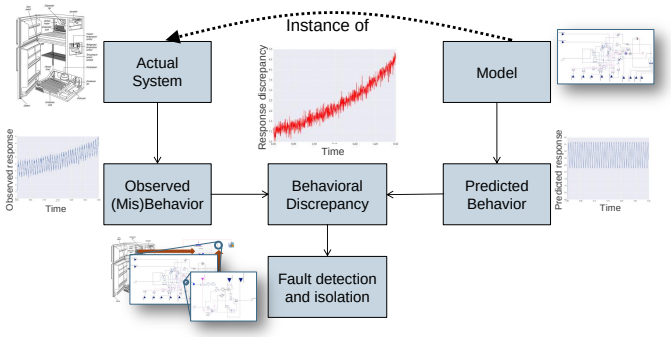

Figure 1: Usual Model-Based Diagnosis Architecture

that the artifact no longer functions as intended because it no longer accurately mirrors its design. For some reason, be it wear, malicious intent, faults, etc., the artifact has become different than its design. The task of a diagnostician is to identify what that difference is. The diagnostician hypothesizes differences which would produce the observed symptoms. Such an hypothesized difference is a strong candidate for what parts of the artifact need to be replaced to restore the original intent of the design.

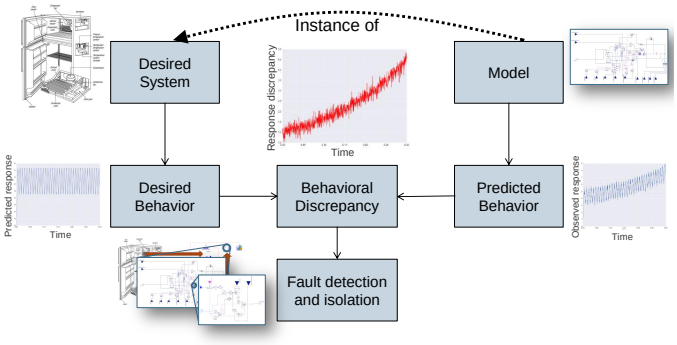

Figure 2: Model-Based Diagnosis Architecture for Design

Consider design. A designer develops a plan to construct an artifact which when completed meets a desired behavior. The key insight is that both the designer and the diagnoser are working with a model (the design) and observations. The

task of the diagnostician is to construct a model to explain the symptomatic observations. This is expressed as a deviation (additions, removals, or modifications) of the intended design. The task of a designer is to construct a model which manifests the desired intent (Figure 2). The final output of a designer is a model of an artifact which satisfies the desired behavior. The final output of a diagnostician is a model which explains the observations. Both construct models which align with observations. In one case the observations are symptoms and in the other observations are desired behaviors.

Some might argue that design involves leaps of creativity that are not inherent to diagnosis. In the extreme the duality breaks down: design is just diagnosis where the initial design is just a blank slate and diagnosis will construct a design which satisfies the users. This is clearly absurd and we do not want to argue for that. We observe from long experience with designers that the vast majority of design effort is spent modifying or augmenting an existing design that has not quite met desired specifications. For the vast majority of design time, designers and diagnosticians are engaged in exactly the same type of activity.

2 Full-Adder example

We illustrate these ideas with a very simple example before considering systems of $1000's$ of components. Consider the correctly designed Full Adder of Figure 3. It adds 3 bits, a and b with a carry in ci and computes the two bit output. Given inputs $ci = 0, a = 1, b = 1$ and observation $co = 0$, we know something is wrong with this circuit. One should expect the binary result $0 + 1 + 1 = 10$ and instead we get 00. Model-based diagnosis [de Kleer and Williams, 1987] localizes the fault to either $O1$ or $A1$, no other component fault can influence the observed symptom.

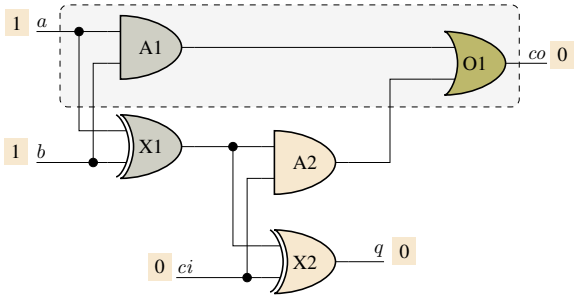

Figure 3: Correctly designed full adder and a non-nominal observation lead to two diagnoses: (1) O1 is faulted stuck-at-zero and (2) A1 stuck-at-zero.

Consider how exactly the same mechanism can aid design. Suppose we instead start with an incorrectly designed Full Adder. The designer has $O1$ mistakenly being an AND-gate (see Figure 4). co should be 1. The fault could be A3 stuck-at-0 or that A3 should be replaced by an OR-gate.

3 GDE (General Diagnosis Engine) Paradigm

We adopt the general framework of [de Kleer and Williams, 1987] which is more formally described in [de Kleer *et al.*, 1992]. We briefly summarize the key features here:

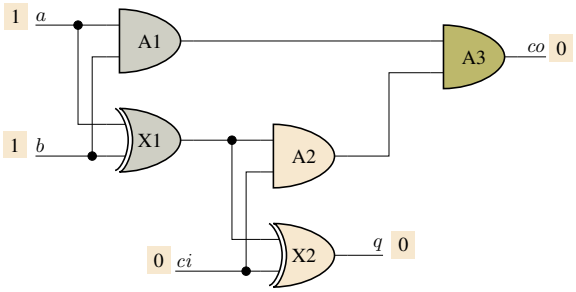

Figure 4: Badly designed full adder: A3 should really be an OR gate

Definition 1. *A system is a triple (SD,COMPS,OBS) where: SD, the system description, is a set of first-order sentences; COMPS, the system components, is a finite set of constants; OBS, a set of observations, is a set of first-order sentences.*

Definition 2. *Given two sets of components Cp and Cn define $\mathcal{D}(Cp, Cn)$ to be the conjunction:*

$$\Big[\bigwedge_{c \in Cp} AB(c)\Big] \wedge \Big[\bigwedge_{c \in Cn} \neg AB(c)\Big],$$

where $AB(x)$ represents that the component x is ABnormal (faulted). A diagnosis is a sentence describing one possible state of the system, where this state is an assignment of the status normal or abnormal to each system component.

Definition 3. *Let $\Delta \subseteq COMPS$. A diagnosis for (SD,COMPS,OBS) is $\mathcal{D}(\Delta, COMPS - \Delta)$ such that the following is satisfiable:*

$$SD \cup OBS \cup \mathcal{D}(\Delta, COMPS - \Delta)$$

Definition 4. *A diagnosis $\mathcal{D}(\Delta, COMPS - \Delta)$ is a minimal diagnosis iff for no proper subset Δ' of Δ is $\mathcal{D}(\Delta', COMPS - \Delta')$ a diagnosis.*

Definition 5. *An AB-clause is a disjunction of AB-literals containing no complementary pair of AB-literals.*

Definition 6. *A conflict of (SD,COMPS,OBS) is an AB-clause entailed by SD $\cup$ OBS.*

Definition 7. *A minimal conflict of (SD,COMPS,OBS) is a conflict no proper sub-clause of which is a conflict of (SD,COMPS,OBS).*

Theorem 1. *Suppose that Π is the set of minimal conflicts of (SD,COMPS,OBS), and that Δ is a minimal set such that,*

$$\Pi \cup \{\bigwedge_{c \in COMPS - \Delta} \neg AB(c)\}$$

is satisfiable. Then $\mathcal{D}(\Delta, COMPS - \Delta)$ is a minimal diagnosis.

We also assume the usual axioms for equality and arithmetic are included in SD.

The framework has been extended to fault modes. For example if an inverter has two fault modes stuck-at-0 (SA0) and stuck-at-1 (SA1) these can be represented as:

$$INVERTER(x) \wedge AB(x) \rightarrow SA0(x) \vee SA1(x)$$

$$SA0(x) \rightarrow output(x) = 0$$
$$SA1(x) \rightarrow output(x) = 1$$

The notion of mode diagnosis can be defined analogously. A mode diagnosis M for (SD,COMPS,OBS) is a conjunction of mode literals or good literals ($\neg AB(x)$) for each element of COMPS such that $SD \cup OBS \cup \{M\}$ is satisfiable. For convenience we notate every mode diagnosis by its AB modes. For example, the Full Adder in Figure 3 has the following two mode single fault diagnoses: $[SA0(O1)], [SA0(A1)]$.

4 Extension to Redesign

Instead of fault modes such as stuck-at-1 and stuck-at-0, the fault modes represent the replacements which might be possible. For example, an AND gate maybe should have been a NAND, NOR or OR:

$$AND(x) \wedge AB(x) \rightarrow NANDm(x) \vee NORm(x) \vee ORm(x).$$

Where NANDm, NORm and ORm all represent the respective models for NAND, NOR and OR:

$$AND(x) \wedge \neg AB(x) \rightarrow ANDm(x).$$

A redesign is a mode diagnosis where the modes are alternative behaviors, except that OBS is what the designer intended. Nothing needs to be physically observed. As a consequence, all the usual algorithms for model-based diagnosis apply. GDE reports the following single fault redesigns for the Full Adder of Figure 4 with the indicated design intent: [NOR(A2)] [NAND(A2)] [NAND(A3)] [OR(A3)].

GDE can treat errors in wiring as well as components. For this paper we will consider the possibilties that a designer has wired an input to a gate incorrectly. Any gate that is wired to its own fanout would be rejected by any design tool, so we only consider the remaining possible miswirings. Table 1 lists the 50 possible miswirings for the full-adder under this criteria. To represent wiring errors we model each input wire as another component that can be faulted. The rules for faulty components are constructed automatically by GDE. Each rule says an input is either wired as indicated, or is in a fault mode where it is connected to some other output gate. For example, the fault model for the input wire to exclusive-or gate X1 is:

$$WIRE(C(in1, X1)) \wedge AB(WIRE(C(in1, X1))) \rightarrow$$
$$in1(X1) = a \vee in1(X1) = ci \vee in1(X1) = out(A1)$$

These classes of design errors are motivated by what is seen [Al-asaad and Hayes, 2000] in the analysis of digital design errors: Gate Substitution Errors (GSE), Gate Count Errors (GCE), Input Count Errors (ICE), and Wrong Input Errors (WIE).

5 Redesign

In Model-Based Diagnosis GDE is given a correctly designed system, a faulty instance of it, and determines possible faults in the system to explain the physical observations (Figure 5). In Model-Based Redesign GDE is given a correct designer intent, a faulty instance of it, and determines possible redesigns of the system to ensure it performs as intended.

Gate	Terminal	Possible Rewirings
A1	in1	ci, a, X2, X1, A2
	in2	ci, b, X2, X1, A2
A2	in1	ci, a, b, X2, A1
	in2	a, b, X2, X1, A1
O1	in1	ci, a, b, X2, X1, A2
	in2	ci, a, b, X2, X1, A1
X1	in1	ci, a, A1
	in2	ci, b, A1
X2	in1	ci, a, b, O1, A2, A1
	in2	a, b, O1, X1, A2, A1

Table 1: All possible miswirings of the full-adder with specified connection model.

We assume that the design is close enough to being correct and we are initially provided with an intended observation which violates a simulated observation. GDE then attempts to identify a design change which guarantees that the intended and simulated observation match. As there may be multiple distinct observations, every design change will take into account all past observations. Figure 6 illustrates the process whose ultimate objective is that the design becomes as close as possible to the intent. We are provided a sequence of inputs to the system being designed. The design simulation will output a corresponding sequence of outputs according the current design. If the input-output pair fails to match the intent, GDE redesign is invoked to update the design, to avoid all past failed input-output pairs and support all past successful input-output pairs. This is performed by an unmodified GDE using fault modes for possible design faults. For example, the minimum cardinality diagnoses are the minimum cardinality redesigns. On the other hand, if the design succeeds, it is still not guaranteed to meet the intent for all future inputs.

To improve the quality of the redesign we must construct novel inputs which satisfy all past observations yet reduce as yet unencountered failing observations. In combinational circuits ATPG (Automatic Test Pattern Generation) tools come closest to what is needed. ATPG tools are typically used to generate inputs to maximize manufacturing fault detection. In our case we need detect misdesigns, not manufacturing faults. (Although it has been observed that inputs generated in this way can have significant misdesign coverage [Al-asaad and Hayes, 2000].) In addition, we need to generate inputs which best differentiate among remaining unencountered misdesigns. Our approach is again based on GDE. First, GDE is used to identify redesigns which both satisfy past input-output pairs and produce different output(s) for some as-yet-unobserved input(s). Second, GDE construct inputs which best differentiate among unencountered misdesigns. It is critical that this search be both efficient and generate as few novel observations as possible — otherwise this approach is of little use. Our current approach is based on generating random inputs focussed only on the set of unencountered misdesigns. In the benchmarks that follow we will always limit the possible misdesigns to a cardinality or type as the number of possible redesigns can be unbounded.

For the benchmarks which follow we assume that the circuit does not change between observations (although we do not yet take complete advantage of non-intermittency assumptions as described in [Raiman *et al.*, 1991].)

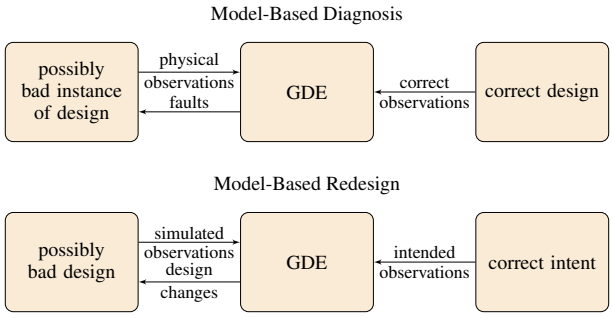

Figure 5: Analogy between design and diagnosis

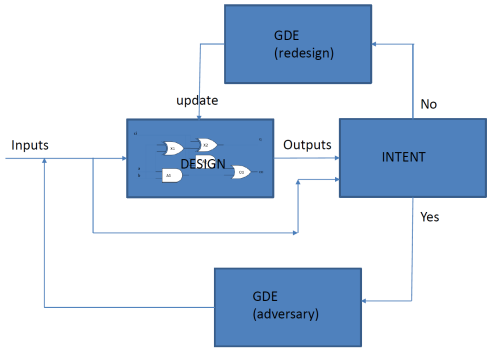

Figure 6: GDE's role in design.

6 Ambiguity Groups

In diagnosis an ambiguity group is a set of diagnoses which are indistinguishable given a set observations. The same phenomena arises in design. An ambiguity group is a set of redesigns all of which restore the designer's intent. Suppose the designer's goal is a circuit which inverts twice, but it has been misdesigned as in Figure 7. There are two indistinguishable redesigns which achieve the designers' intent: BUFFER(f1) and INVERTER(f2). Namely, the inverter f1 can be replaced by a buffer, or the buffer f2 can be replaced by an inverter. From an input/output perspective the redesigns are indistinguishable.

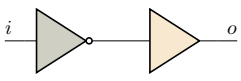

Figure 7: Badly designed two inverters

Model-based diagnosis approaches can diagnose multiple simultaneous faults. Analogously our redesigner can repair such design errors. The redesigner may make surprising changes. Suppose the designer intended to create the circuit of Figure 3, but mistakenly created the circuit of Figure 8 in which the ANDs were replaced with NANDs. GDE, which finds simplest diagnoses first, also finds simplest redesigns first, will leave the two NANDs unchanged and replace the OR gate with a NAND. (In this case, simplicity is determined by cardinality.) This circuit is a correctly designed full adder, but is an alternate design from what the designer likely intended.

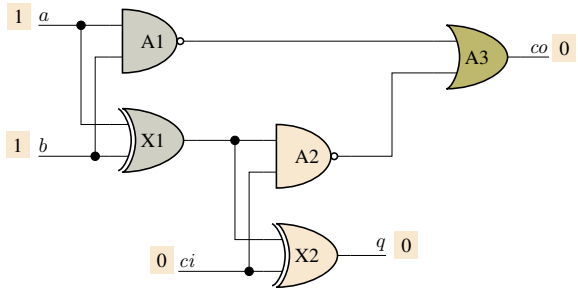

Figure 8: A misdesigned full-adder with two ANDs replaced by NANDs.

When considering wiring errors, possible redesigns introduce ambiguity groups as well. Consider the bad design where the top input to the Full-Adder's A1 is instead connected to input b. There are 2 redesigns which can restore the bad design to correct functioning: (1) move A1's top input back to circuit input a, (2) move A1's bottom input to circuit input a. If all the gate inputs are order insensitive (as is the case with the Full Adder), then there will be an ambiguity group associated with each gate. Thus, the Full Adder benchmark has 10 mutant wirings for which there are ambiguity groups.

7 Benchmarking

To evaluate our approach we have produced a benchmark from the DXC [Kurtoglu *et al.*, 2009] competition. Consider gate errors first. For each benchmark circuit we create a mutant for each possible gate misdesign of Table 2. This produces a large set of redesign cases. Consider the single fault mutants of the full adder. The 2 XOR gates have 4 misdesign patterns, and 2 AND gates have 3 misdesign patterns and 1 OR gate has 3 misdesign patterns. Therefore there are $4+4+3+3+3=17$ misdesigns in the single fault benchmark for the full adder. One of the possible misdesigns is the circuit of Figure 4.

A benchmarking construction challenge arises because a mutation does not neccessarily introduce a fault. Whether a fault is introduced depends on the mutated gates' function in the overall circuit. Even considering all possible input vectors, a particular 2-input gate may only see two inputs 00 or 11 - with 10 and 10 never occuring *for any input.* In this case it does not matter whether the particular gate is an OR or an AND as they both produce identical outputs for 00 and 11. As a consequence mutating that particular gate from OR to AND or AND to OR does not introduce a design error. Hence, they are not included in the benchmark.

Gate	Mutant(s) Considered
AND	NAND, NOR, OR
NAND	AND, NOR, OR
OR	AND, NAND, NOR
NOR	AND, NAND, OR
XOR	AND, NAND, OR, NOR
INVERTER	BUFFER
BUFFER	INVERTER

Table 2: Mutant patterns used to construct benchmark. One of many possible.

Circuit	Type	G	$m*$	m	a	$\bar{i}$
adder	from this paper	5	17	17	0	1.6
c17	simple logic	7	18	18	0	1.6
74182	carry generator	19	55	55	0	1.7
74283	4-bit adder	36	100	100	0	2
74181	4-bit ALU	65	179	179	2	2.4
c432	priority decoder	160	418	397	80	6.7
c499	ECAT	202	630	627	15	6.4
c880	ALU and control	383	971	971	184	3.4
c1355	ECAT	546	1494	1485	217	9.0
c1908	ECAT	880	1762	1746	582	7.2
c2670	ALU and control	1193	2545	2148	845	6.5
c3540	ALU and control	1669	3581	3507	617	9.7
c5315	ALU and selector	2307	5133	5132	792	9.2
c7552	ALU and control	3512	7716	7353	363	9.3

Table 3: Effort needed to redesign single fault component mutants. G is the number of gates in the circuit. $m*$ is the number of mutants possible, m excludes those mutants which produced a correct result or we could not find a test vector for and $\bar{i}$ is the average number of replacements needed to correct the m mutants. a is the number of benchmarks which had multiple correct redesigns.

For the smaller benchmarks, GDE's test-vectors are guaranteed to isolate a correct redesign. For larger benchmark circuits (> 18 inputs) the redesign is not absolutely guaranteed to be correct for unobserved inputs. There is a chance of a redesign error - no different than real practice. Table 3 compiles our redesigner analysis of most of the ISCAS benchmarks. The cost $\bar{i}$ is the average number of replaced components needed to restore functioning. There are numerous sources of variability which contribute to its being an estimate: (1) at each redesign step, there are usually multiple redesigns which meet all known observations and we pick a random one, (2) at each redesign step there are almost always multiple intent violations and we just pick a random one, (3) for larger circuits of more than 18 inputs we may not try all of them which can lead to underestimates.

Figure 9 plots the average number of observations needed to identify the simplest redesign of a circuit vs. the size of the possible redesign space. Each point in the figure represents one of the benchmark circuits. In analogy to diagnosis tasks, the number of observations needed is roughly logarithmic in the size of the possibility space. GDE has been designed to detect faults in large systems. The average CPU time required to analyze each of the $24,619$ (sum of entries in colum $m*$) sample points is less than 1s; most of the CPU time spent in GDE discovering new input vectors.

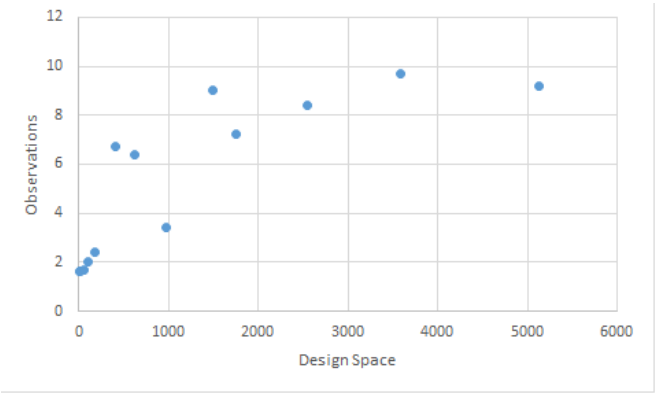

Figure 9: Number of observations to achieve a correct design vs. size of design space (for single component mutants).

MBD can diagnose any number of simultaneous faults. Analogously, our redesigner can resolve any number of simultaneous design errors as well. The number of possible component design faults scales dramatically with circuit size. For example, the benchmark 74283 4 bit adder has $156,736$ cardinality ≤ 3 design errors in our benchmark. Figure 10 plots the number of observations needed to repair higher cardinality design errors. The plots are similar for other circuits. Higher-cardinality design errors are not particularly difficult for our approach.

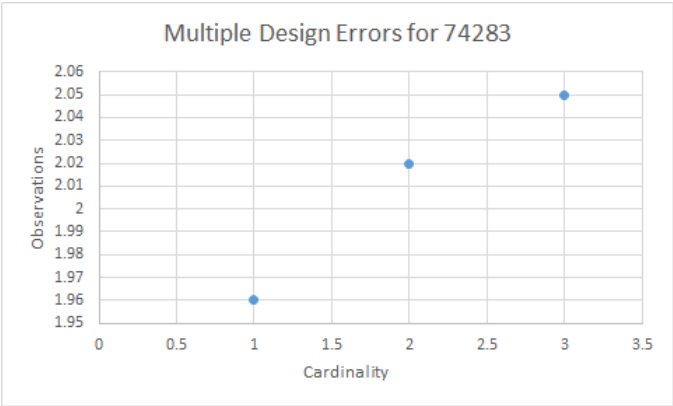

Figure 10: Number of observations to resolve higher cardinality component design errors in the 74283.

The low number of observations needed to do redesign of high cardinality design errors may seem initially surprising. However, there are a number of reasons for this. First, there is no efficient approach to generate *all* symptoms for higher cardinality design errors in large circuits. Also, the number of benchmark examples grows exponentially with circuit size.

Circuit	G	$m*$	m	a	$\bar{i}$
adder	5	67	67	10	2.6
c17	7	110	110	12	2.6
74182	19	1467	1444	500	9.7
74283	36	3385	3284	1333	7.2

Table 4: Effort needed to redesign all single fault mutants including both component and wiring errors. G is the number of gates in the circuit. $m*$ is the number of mutants possible, m excludes those mutants which produced a correct result or we could not find a test vector for and $\bar{i}$ is the average number of observations needed to correct the m mutants. a is the number of benchmarks which had multiple valid redesigns.

So we do not have ground truth for larger circuits (only for the first 6 rows of Table 3. Formally verifying [Drechsler, 2013] even modest designs can be arbitrarily difficult. Second, many of the design errors will interact and affect different outputs independently. Third, many higher cardinality design errors, can be resolved by lower cardinality changes.

Table 4 provides the data for the redesign of mutants containing both component and wiring errors. There are signficantly more possible wiring errors than gate substitutions even for single faults. The cost of isolating a redesign is always much higher with the presence of wiring errors. As expected, many wiring mutants have ambiguity groups as even for single faults, there can be large number of ways to rewire a mutant to correct functioning.

Wiring errors introduce complexities not present in simple gate errors and requires a small analysis step outside of the GDE framework. We remove from the benchmark any wiring errors that introduce loops — we assume that any design tool would have detected these in the first place and warned the designer. When constructing possible redesigns for mutants we also remove all repairs that would result in the introduction of a loop or leave an input disconnected.

8 Generalization

Although this approach can be used to debug digital designs, it can be used with any technology that can be modeled using model-based diagnosis. To debug a new type of system all model-based diagnosis needs is a new component model library. Model-based diagnosis has been applied to a vast array of system types: qualitative [de Koning and Bredeweg, 1998], data-bases, excel spreadsheets, Modelica models, analog circuits, quantitative models, programs [Abreu *et al.*, 2006], etc. We have used digital examples in this paper primarily because of the availability of large benchmarks to demonstrate and test the approach.

9 Related Work

The idea of using model-based diagnosis for redesign was recognized early by the DX community [Alberts *et al.*, 1993; van Eldonk *et al.*, 1996; Bakker *et al.*, 1994] Much of that work was aimed at redesigning to meet a slightly changed purpose. None recognized that GDE/Sherlock could directly do redesign without modification. None can diagnose wiring errors. One of the contributions of this paper is to show that can be achieved at scale for multiple simultanenous design faults in component choice and wiring errors (not enough inputs, too many inputs). None describe how to isolate such design faults. [Friedrich *et al.*, 1999] addresses model-based diagnosis of hardware designs expressed in VHDL by building abstractions of the circuit to be modified.

[Könighofer and Bloem, 2011] applies ideas from model-based diagnosis to the repair of simple imperative programs. It uses similar intuitions behind the approach of this paper. However, its approach is much more complicated requiring a combination of program analysis, error localization, determining diagnoses and using templates to construct repairs. Our approach is achieved with one simple mechanism: GDE. It is very fast and is successfull on exhaustive benchmarks of 100,000s of test cases. Spectrum-Based Fault Localization [Abreu *et al.*, 2006] for simple programs uses very crude models to suggest suspicious areas of very large programs. It cannot suggest redesigns or propose new program inputs to isolate design errors.

There is significant research in designing systems, including digital circuits, via Genetic Algorithms [Frenz *et al.*, 2009]. Genetic Algorithms explore a space of possible designs by introducing random mutations until the design best matches the desired intent. Our redesigner uses MBD techniques to directly determine which components of the system need to be changed to achieve the designer's intent.

[Pill and Quaritsch, 2013] describes an approach for debugging product specifications (which are a kind of design) based on LTL. Their strong fault models precisely define the search space - as with the approach in this paper.

The General Redesign Engine [Feldman *et al.*, 2009] also uses MBD to redesign a circuit. However, its goal is to redesign a working circuit into another working circuit which is more tolerant to faults. The resulting circuit has a higher probability of meeting the designer's intent when components fail. Our redesigner is not concerned with component failure.

The task of synthesizing a digital circuit from an abstract specification (e.g., VHDL [Ashenden, 2008]) or from an input-output specification is the basis of a large industry. VHDL tools can be given a specification of a designer intent and automatically generate and layout a more detailed implementation. Karnough Map [Mano, 2001] techniques are given input-output vector specifications and synthesize a combinatorial logic circuit which manifest the desired function. Both guarantee "correct-by-construction". Yet, far too many modern circuits contain faults. Books are written about why that is. Most designs require multiple iterations to be shipped and even then design errors remain. Part of the reason is that few modern ICs can be designed fully automatically (e.g., approaches like Karnough Maps do not scale to modern designs) and need significant human designer input. Debugging IC designs is a significant source of delay in delivering new products.

References

[Abreu *et al.*, 2006] R. Abreu, P. Zoeteweij, and A.J.C. van Gemund. An evaluation of similarity coefficients for software fault localization. In *Proceedings of the 12th IEEE Pacific Rim Symposium on dependable Computing (PRDC'06)*, Riverside, CA, USA, December 2006.

[Al-asaad and Hayes, 2000] Hussain Al-asaad and John P. Hayes. Logic design validation via simulation and automatic test pattern generation. *Journal of Electronic Testing*, 16(6):575–589, Dec 2000.

[Alberts *et al.*, 1993] L.K. Alberts, R.R. Bakker, D. Beekman, and P.M. Wognum. *Model-based redesign of technical systems*, pages 201–211. 1 1993.

[Ashenden, 2008] Peter J. Ashenden. *The Designer's Guide to VHDL, Volume 3, Third Edition (Systems on Silicon) (Systems on Silicon)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3 edition, 2008.

[Bakker *et al.*, 1994] R.R. Bakker, S.J.M. van Eldonk, P.M. Wognum, and Nicolaas Mars. *The use of model-based diagnosis in redesign*, pages 552–647. 2 1994.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, April 1987. Also in: *Readings in NonMonotonic Reasoning*, edited by Matthew L. Ginsberg, (Morgan Kaufmann, 1987), 280–297.

[de Kleer *et al.*, 1992] J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[de Koning and Bredeweg, 1998] Kees de Koning and Bert Bredeweg. Using GDE in educational systems. In *ECAI*, pages 279–283, 1998.

[Drechsler, 2013] Rolf Drechsler. *Formal verification of circuits*. Springer Science & Business Media, 2013.

[Feldman *et al.*, 2009] Alexander Feldman, Gregory M. Provan, Johan de Kleer, Lukas D. Kuhn, and Arjan J. C. van Gemund. Automated redesign with the general redesign engine. In *Eighth Symposium on Abstraction, Reformulation, and Approximation, SARA 2009, Lake Arrowhead, California, USA, 8-10 August 2009*, 2009.

[Frenz *et al.*, 2009] Christopher M. Frenz, Steve Peters, and Wilson Julien. Evolution of digital logic functionality via a genetic algorithm. *CoRR*, abs/0907.4426, 2009.

[Friedrich *et al.*, 1999] Gerhard Friedrich, Markus Stumptner, and Franz Wotawa. Model-based diagnosis of hardware designs. *Artificial Intelligence*, 111(1):3 – 39, 1999.

[Könighofer and Bloem, 2011] Robert Könighofer and Roderick Bloem. Automated error localization and correction for imperative programs. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, FMCAD '11, pages 91–100, Austin, TX, 2011. FMCAD Inc.

[Kurtoglu *et al.*, 2009] Tolga Kurtoglu, Sriram Narasimhan, Scott Poll, David Garcia, Lukas Kuhn, Johan de Kleer, Arjan van Gemund, Gregory Provan, and Alexander Feldman. First international diagnosis competition - DXC'09. In *Proc. DX'09*, 2009.

[Mano, 2001] M. Morris Mano. *Digital Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 3rd edition, 2001.

[Pill and Quaritsch, 2013] Ingo Pill and Thomas Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1053–1059. IJCAI/AAAI, 2013.

[Raiman *et al.*, 1991] O. Raiman, J. de Kleer, V. Saraswat, and M. H. Shirley. Characterizing non-intermittent faults. In *Proc. 9th National Conf. on Artificial Intelligence*, pages 849–854, Anaheim, CA, July 1991.

[van Eldonk *et al.*, 1996] S.J.M. van Eldonk, L.K. Alberts, R.R. Bakker, F. Dikker, and P.M. Wognum. Redesign of technical systems. *Knowledge-Based Systems*, 9(2):93 – 104, 1996. Models and techniques for reuse of designs.