A Model-Based Automated Debug Process

Doug Manley, Agilent, Loveland CO, doug_manley@agilent.com  
Bill Eklow, Cisco Systems Inc., San Jose CA, beklow@cisco.com

1 Abstract

As components, boards and systems become more and more complex; the cost of troubleshooting failures is getting greater and greater. Test strategies try to detect and diagnose failures early on in the test process (component and structural tests), but inevitably some defects are not detected until functional testing is done on the board. Finding these defects usually requires an "expert", with engineering level skills in both hardware and software. Depending on the complexity of the product, it could take several months (even years) to develop this level of expertise. During the initial product ramp, this expertise is usually most needed and often unavailable. Debug time is usually very long and scrap rates are generally high.

This paper will present a process/strategy to automate the debug process and allow for information sharing early in the design cycle between the hardware and software designers, and the people that are responsible for building the product. The tool that is presented in the paper will provide a framework not only to document engineering information, but also to refine that information during product manufacturing as more defects and causes are identified.

Case studies involving high complexity networking products will be presented in the paper. The studies will demonstrate the effectiveness of the debug tool and highlight some issues that will also be addressed later in the paper. In addition, the paper will discuss some future uses of the tool to help provide Design for Test and Design for Diagnosis information very early in the design cycle (prior to detailed hardware and software design).

2 Introduction

Generally, technicians and engineers use a “directed” approach to debugging functional failures. In practice, a tree is developed where the nodes of the tree represent symptoms and the branches represent observations. Once a symptom has been confirmed by an observation, another branch is followed to another confirmed node. Observations can consist of making measurements, running additional tests, visual inspection or querying internal registers or memory. Hopefully, once the tree has been traversed to the end, a diagnosis can be made. This is often a lengthy process and requires a tremendous amount of detailed knowledge about the inner workings of the board and the functional tests. This approach becomes even more difficult when there are several complex FPGA’s and/or ASIC’s on the board (any of which may be considered as complex as an entire board).

Since a technician/engineer often focuses on a single path (branch) during a directed debug process, information from other tests (passing or failing) or observations could be overlooked. This information could be very valuable in the diagnosis. For instance, if two components are being considered in the debug of a failing test, and one of the components has been used significantly in several passing tests, the component which is used in the passing tests may be considered a less likely candidate for the failure.

The directed debug approach may also promote behavior that can add more time to the debug process. Technicians/engineers may get hung up on details of a problem when a “logical” cause may already be known. In other words, a technician/engineer may already know to a certain confidence level that replacing a component will fix the problem. Yet, they will continue to troubleshoot until the cause is completely proven. At an 80% confidence level, replacing the part may be more cost effective in a
production environment than getting to a 100% confidence level (a process which could take several hours to several days longer).

On the other extreme, after debugging several boards a technician may fall into the trap of complete ignoring the debug tree and going directly to the repair based on his experience debugging similar failures (experience based debug). This approach may work in some instances but generally fails because the diagnostic resolution is usually poor for a single test (versus several passing and failing tests). Often, too, the repair decision is based on emotion instead of reasoning.

Finally, the directed debug approach does not promote information sharing. Generally, information required to debug specific problems stays with the engineer or technician. Once a difficult problem has been solved, the information is not archived to a "common" repository, dooming another engineer/technician to repeat the same painful debug process.

History-based systems (fault dictionaries and expert systems) have been used to address some of the problems described above. However, it can take a substantial amount of time to acquire the amount of information needed for the system to be effective. In cases where information is not present (very often early in the manufacturing cycle), the technician/engineer is required to go back to the directed approach to diagnose the defect. While this approach may become effective over a long period of time, it offers no benefit until a large number of boards have been effectively debugged.

Many times in complex systems it is hard to provide enough symptom resolution to ensure a unique symptom for every failure. In this case, there is an additional risk of history-based systems in that the sample size may not be large enough to determine if the diagnosis is a true diagnosis for every case with a given symptom, or if some cases require a different diagnosis. By focusing on a specific cause and effect, debug data which is added to the history-based system is doomed to address specific problems and not classes of failures, making the requirement for data gathering even more substantial. Some reasoning should be part of the diagnosis process as well.

This paper will present a model-based system that is used to describe how the hardware is tested by the functional tests. The model-based system uses information from several diagnostics to indict and acquit hardware based on how it is used in passing and failing tests. The model-based system provides a good template for defining useful information from the hardware and software developers. The system also provides the framework to archive information that is learned during debug (using this information to enhance the model itself, leading to more accurate diagnosis of defects).

3 A Model-Based Approach to Debug

A number of Artificial Intelligence (AI) techniques have applied to automating fault diagnosis. These include Rules-Based Reasoning (RBR), Bayesian Networks (BNs), Neural Networks (NNs), Qualitative Reasoning (QR), and Model-Based Reasoning (MBR). An overview of these can be found in [Lin98].

The Fault Detective MBR technology described in this paper is an outgrowth of experience with the Agatha RBR system described in [Allred91]. While Agatha succeeded in automating fault diagnosis for a class of boards, it suffered from the fundamental limitations of RBR, i.e., the correspondence between failure symptoms and faults responsible for them must be known before the rules base can be built. As a result automation is not available during new product ramp when most needed.

In a Model-Based Reasoning system, a model is built that describes the correct operation of the device. A diagnosis accounts for the differences between the model of correct operation and observed faulty operation. MBR systems can be constructed from information available at design time therefore automated diagnosis is available during new product ramp.

Some MBR systems model the structure of the device under test (DUT) as in [Krysander2002]. In [Lin98] an object model of the DUT is used. In [Simpson91] dependency graphs are used.

The Fault Detective MBR uses a qualitative model of the correct operation of the DUT’s suite of functional tests. For each test, the model includes:

1. The components utilized to some degree. This represents the test path, e.g., a processor and memory. Components may be described hierarchically.
2. The specific operations utilized by each test, e.g., a processor may read in one test and write in another.
3. A qualitative (high, medium, low) specification of each specific operation’s fraction of total component functionality, e.g., a processor read or write represents a small amount of its overall functionality.

4. A user-defined qualitative specification of the failure mode, e.g., failed reading bank 1.

5. A qualitative specification of each component’s a priori failure rate, e.g., an ASIC might have a higher failure rate than a resistor.

The above represents a model of how a functional test exercises the DUT both structurally (the components) and functionally (the operations).

The set of qualitative test observations about a particular DUT is called in a syndrome. A syndrome consists of one or more symptoms. A symptom may represent either a passing or failing observation of the DUT. Because Fault Detective uses a qualitative definition of symptom, the reasoning engine is completely independent of the specifics of the physical measurements that constitute the test.

Because the concepts of structure, operation, and syndrome are so general, a Fault Detective model can be used for automated diagnosis of almost any electrical device.

As an example, a DUT consisting of a processor and three memory banks is illustrated below. All parts share the same physical bus.

![Figure 1](image_url)

The test suite consists of three tests. The first test reads and writes Memory 1; the second reads and writes Memory 2; the third reads and writes Memory 3. While one memory is operating the others must disable themselves from the bus. Components are listed in bold, operations in italics. The test model is as follows:

1. A successful Test1 requires the Processor to read and write data and Memory 1 to enable, read and write data. Memory 2 and Memory 3 must disable themselves.
2. A successful Test2 requires the Processor to read and write data and Memory 2 to enable, read and write data. Memory 1 and Memory 3 must disable themselves.
3. A successful Test3 requires the Processor to read and write data and Memory 3 to enable, read and write data. Memory 1 and Memory 2 must disable themselves.
4. A successful Disable/enable is considered to use a low amount of the memories’ total functionality.
5. A successful memory read and write is considered to use a high amount of total functionality because every cell of the memory is read and written.
6. All components are considered to have an equal a priori failure rate.

Let’s consider several syndromes \((S_1, S_2, S_3)\) and their associated diagnoses \((D_1, D_2, D_3)\). \(F\) denotes a failing symptom, \(P\) a passing one. A diagnosis consists of a list of candidate diagnoses each a plausible explanation for the observed syndrome. Each candidate has an associated weight. A higher weight candidate is a more likely explanation, a lower weight candidate a less likely one. Weights are calculated using an approximation of Bayes’ Rule, i.e. the weight is proportional to the probability that a given candidate is faulty given the syndrome.

Each failing test can be seen as a conflict set consisting of the components and operations used and therefore may account for the failure. The syndrome results in collection \(C\) of such sets. Each candidate diagnosis represents a minimal hitting set. A set \(S\) that has a non-empty intersection with every set in collection of sets \(C\) is called a hitting set. If no element can be removed from \(S\) without violating the hitting property, the set is considered minimal [Reiter87]. A minimal hitting set can be composed of one or more components. For convenience the weights
are normalized to a 100 point scale.

1. \( S_1 = \{\text{Test1:F, Test2:F, Test3:F}\} \)
\( D_1 = \{\text{Processor:25, Memory1:25, Memory2:25, Memory3:25}\} \). Because all tests fail, each component has an equal weight. The Processor may not work or one of the parts may hopelessly hang-up the bus. Three tests fail so there are three conflict sets and each is \{Processor, Memory1, Memory2, Memory3\}. Each element of \( D \) is a minimal hitting set that by itself can account for the failure. Non-minimal hitting sets such as \{Processor, Memory1\} are also plausible explanations and are in practice seen as sequential repair of minimal hitting sets, e.g., first fixing the Processor, then Memory1, etc.

2. \( S_2 = \{\text{Test1:P, Test2:F, Test3:F}\} \)
\( D_2 = \{\text{Memory1:35, Memory2:31, Memory1:31, Processor:3}\} \). Even though Test1 passed, Memory1 is the most likely candidate because Memory1 never demonstrated the capability to disable itself because this operation is only demonstrated in Test2 and Test3. This fault explains how Memory1 could pass its read/write test but still cause the other two tests to fail.

3. \( S_3 = \{\text{Test1:F, Test2:P, Test3:P}\} \)
\( D_3 = \{\text{Memory1:97, Processor:1, Memory2:1, Memory3:1}\} \). While Memory1 exhibited the ability to disable itself in Test2 and Test3, it never demonstrated any ability to read/write or enable. Only unlikely intermittent operation of the other parts could account for Test1’s failure.

Low weight candidates exist in \( D_2, D_3 \) because Fault Detective recognizes that some operations may work intermittently. For example, a timing fault may exist inside the Processor.

This example demonstrates how the qualitative MBR can successfully diagnose very complex faults despite having no domain-specific knowledge of bus architecture. In \( D_2 \) we were able to correctly determine that Memory1 is probably interfering with the bus despite the fact that Memory1 successively passes its own memory test.

4 Experimental Results

Cisco and Agilent jointly developed an experiment to test the effectiveness of the Fault Detective tool. The experiment started in October, 2000 and was conducted on a 48 port, Ethernet controller card. The card was a medium complexity card consisting of 8 ASIC’s, 14 large scale devices, 25 memory devices, 12 PHY’s (Physical Interface components) and several miscellaneous small scale devices. This board was responsible for a very high debug inventory at our contract manufacturer, which made it a good candidate for the experiment (there were several test cases available). A block diagram is shown in Figure 2.

The first part of the experiment consisted of building a model, collecting test data from 10 boards, applying the test data manually to the model and comparing the results with empirical data obtained by actually fixing the boards. Not all diagnostics were run during the experiment. The experiment concentrated primarily on traffic tests although some “static” (non-traffic) tests were also executed.

The model was built by the test engineer, with some help from the design engineer and diagnostic engineer. The model consisted of 84 devices and 133 tests. A TCL script was written to run the diagnostics and collect the pass/fail information. When the information was collected, there was correlation on only 2 of 10 boards.

Several attempts were made to adjust the model to improve the correlation, but there was little to no improvement. After significant investigation into the design and diagnostics, it was determined that there was
not enough diagnostic resolution to provide unique pass/fail syndrome information (the status of error registers was not displayed by the diagnostics). Additional information was collected from the diagnostics to try to make up for this lack of resolution. It turns out that a bug in the diagnostics and incorrect modeling with respect to the additional information that was collected compounded the problem. During the investigation, it was determined that the error and status information could be collected by reading ASIC registers manually. This information was included in the model and the erroneous information was discarded. The resulting model was much more effective in diagnosing the failures and resulted in a much higher correlation (80%).

The second phase of the experiment was to automate the data collection and processing in the debug environment (do the debug) on a sample of 97 boards. The model fared significantly better in the second phase. Of the 97 boards, 50 boards were fixed after the first rework. Of the remaining 47 boards, an additional 18 were fixed on the second attempt, 2 were fixed on the third attempt and 1 was fixed on the fourth attempt. In total, 73% of the boards were repaired by the automated process. Of the remaining 26 boards, all but 5 fell into 4 categories of failure syndrome. These boards were subsequently fixed with help from engineering and the information that was obtained from the failing tests was integrated into the model. Subsequent to the experiment, the model was able to diagnose greater than 85% of the failures on the board.

The first experiment provided the following insight:

- The modeling process promoted a better understanding of the board design, ASIC design and diagnostics. Since diagnostics were being developed by another group at Cisco, there was minimal incentive for the test engineer to understand how the tests actually exercised the logic. Manual/directed debug promoted a “learn as you go” understanding of the diagnostics. Since the test engineer was responsible for the model development, much more up front understanding of the tests and their effect on the hardware was required. This early model development served as an effective transfer of information from the design group to the test group.
- Coverage does not equal diagnosis. The process of building the model made it evident where diagnostics could not resolve down to a single component (if 2 or more components were exercised exactly the same way by all tests). The first experiment clearly showed the consequences of lack of diagnostic resolution. It was much easier to see the benefit of features such as error checking on chip boundaries and loopbacks, because of their ability to increase the diagnostic resolution.
- A model-based diagnostic tool could provide reasonable results. At 70% to 85% effectiveness, a model-based tool provides a significant advantage over manual/directed debug, which requires significant time and expertise. The model-based approach can be automated, allowing debug to be done by unskilled operators. Boards that can’t be debugged (15% to 30%) are then moved to the manual process. Repair information can then be used to enhance the model and increase the effectiveness.

A second experiment was performed on a Switch Supervisor card. This card was slightly more complex than the Ethernet controller, though there was a fair amount of “shared” logic between the two cards. There were a similar number of large scale components and memory, but the Supervisor also had a daughter card with 4 more ASIC’s and several memory devices.

Using experience from the first experiment, extensive investigation of the hardware and diagnostic tests was performed before beginning the model development. The investigation was aided by the fact that some of the logic and tests were shared between the two boards. The shared logic allowed us to leverage the reasoning applied to the Ethernet controller and significantly reduced the development time.

The second experiment looked at 25 boards. In this case we achieved a 92% success rate within two reworks. No boards were repaired after a third test cycle. The second experiment provided the following insight:

- Confirmation of the final results from the first experiment. With a good understanding of the process of developing the model, we were able to achieve very good results early on. Again, diagnostic resolution was very important. In this case, we had a good understanding of the diagnostic resolution up front and were able to enhance the resolution by reading error and status registers. It seemed that a lot of information could be gained about the diagnostic
resolution just by looking at the model.

- On this product, a lot of scripting was required to determine the data flow and the effect of errors on the data flow. Error counters, packet counters and status registers were analyzed, not only to determine if the right number of packets flowed through the data paths, but also to determine what happened to the data after an error occurred. It would be advantageous to model the data flow in the Fault Detective environment.

One benefit of a model-based system is that information about diagnostics can be very easily conveyed between the diagnostic developer and the model developer (test engineer). The diagnostic engineer can easily convey how the hardware is tested in the form of the model. The test engineer can also convey diagnostic coverage and resolution issues by relating them to the model as well. We observed this during both experiments. Communication about what the diagnostics were doing and problems that we were experiencing was much easier when done in the context of the model itself.

As was mentioned before, information from failure analysis was used to enhance the model. The enhancements very quickly brought the success rate up to a relatively high level. It was found that the very detailed failure syndrome that was generated by the model-based system provided an excellent classification system. There was little ambiguity between syndromes, meaning that boards exhibiting the same syndrome almost always had the same fault. This was not the case when examining a single failing test. Boards failing the same single test often times had many different faults. This unique syndrome ensured that when failure analysis was done on a board with a specific syndrome, the same repair could confidently be applied to all other boards with that syndrome.

5 Model-based System Enhancements

Our case study confirms the results of [Preist97] where Fault Detective demonstrated large (US$600K to $100000 savings/yr.) economic benefits in Hewlett-Packard’s own manufacture of PC motherboards. However, challenges in this case study suggested new two new enhancements to the MBR. The first challenge was that some DUT’s required extensive traffic testing, i.e., the test strategy consisted of running perhaps ten’s of millions of IP packets through the DUT. Some packets may be dropped, others may be re-routed depending on the state of the device. While the qualitative MBR was capable of both modeling and diagnosing traffic tests, it was cumbersome to do so and required some board-specific quantitative reasoning to complement the qualitative reasoning. Our response to the traffic test challenge was to develop a quantitative MBR for the traffic test domain. Details of the approach are beyond the scope of this paper.

Secondly it was observed that the qualitative model was not only sufficient for automating non-traffic test diagnosis, but could be used to assess qualitative fault coverage, fault isolation, and even to suggest new tests that might improve fault coverage and fault isolation. A model analysis tool, Fault Detective Test Analyzer was created to provide these benefits.

Model analysis consists of

1. Entering the qualitative model
2. Injecting simulated qualitative faults into the DUT and mathematically predicting the resulting syndromes – perhaps hundreds of thousands of syndromes
3. Analyzing the simulation results in order to determine key metrics, e.g.,
   a. the percentage of faults detected by at least one failing test
   b. the average number of repair attempts required to fix the DUT – a measure of fault isolation
   c. suggestions of new tests that might improve fault coverage or fault isolation

Model analysis is significant because it allows the quality of test suites to be estimated before manufacture and provides critical feedback to test writers to improve their test suites while still in development. With the inclusion of model analysis, Fault Detective is not only reactive (diagnosing faults given a test suite) but proactive (helping to improve the test suite thereby improving diagnoses).

6 Conclusion

Trouble shooting complex boards is a time consuming, costly problem. An automated, model-based system provides significant benefit early on and additional benefit as the model is refined due to feedback of failure analysis results. The model-based approach provides a framework for the design engineer, diagnostic developer and test engineer to communicate information. This
process guarantees that the correct information is provided to the test engineer to facilitate the most effective debug strategy. In addition, this model-based strategy also provides a framework to communicate issues back to the design engineer and diagnostic developer. Finally, the information that is required for the model provides significant resolution to help categorize failing boards. Most syndromes are unique and repairs applied to a given failure syndrome will most likely be successful on any other board with the same syndrome.

To be effective, the model developer must fully understand the hardware and the effects of the diagnostic tests on the hardware. A lack of diagnostic resolution can significantly impact the results of the model-based tool. Early analysis of the diagnostic resolution is advantageous and can be performed by analyzing the model itself. An accurate model and diagnostics with significant resolution can provide a very effective debug tool. This, in turn, can provide significant cost savings over a manual/directed debug process. By continuously improving the reasoning behind the model, based on debug results, the model can become more and more effective over time – potentially eliminating the need for manual/directed debug.

Cisco has realized significant cost savings by employing this model-based system. Average debug time has been reduced by several hours per board. Many boards that were deemed un-fixable, were fixed by using the model-based system, rather than the directed debug approach that was used. The model-based process is a proactive process, versus the reactive directed debug process. By promoting better understanding of the hardware and diagnostics, and earlier involvement in the design process, the result is a better hardware and diagnostic design, in addition to a better debug process.

7 References


Chris Price, Computer-Based Diagnostic Systems, Springer 1999