On Automating Failure Mode Analysis and Enforcing its Integrity*

Ann T. Tai Kam S. Tso
IA Tech, Inc.
Los Angeles, CA 90024

Savio N. Chau
Jet Propulsion Laboratory
Pasadena, CA 91109

May 16, 2005

Abstract

This paper reports our experience on the development of a design-for-safety (DFS) workbench called Risk Assessment and Management Environment (RAME) for microelectronic avionics systems. Our objective is to transform DFS practice from an ad-hoc, inefficient, error-prone approach to a stringent engineering process such that DFS can keep up with the rapidly growing complexity of avionics systems. In particular, RAME is built upon an information infrastructure that comprises a fault model, a knowledge base, and a failure reporting/tracking system. This infrastructure permits systematic learning from prior projects and enables the automation of failure modes, effects and criticality analysis (FMECA). Among other unique features, the most important advantage of RAME is its capability of directly accepting design source code in hardware description languages (HDLs) for automated failure mode analysis, which enables RAME to be compatible and to evolve with most electronic-computer-aided-design systems. Through an initial experimental evaluation of the RAME prototype, we show that our approach to FMECA automation improves failure mode analysis turn-around-time, completeness, and accuracy.

Keywords: Design for safety, failure mode analysis, FMECA automation, information infrastructure, design source code

Submission Category: Practical experience report

*The work reported in this paper was supported in part by Small Business Innovation Research (SBIR) Contract NAS3-02096 from the Jet Propulsion Laboratory, National Aeronautics and Space Administration.

CALTECH, contract.

1 Introduction

Risk identification and mitigation are essential in design for safety, especially for microelectronic avionics systems. However, the widely used, traditional practice of failure mode, cause, and effect analysis (FMECA) that relies on manual, textual-document manipulation are often ambiguous, inconsistent, and error-prone. Those shortcomings are primarily due to that the analysis correctness heavily relies on the knowledge and experience of individual system/quality-assurance engineers who perform FMECA worksheet generation. Furthermore, the manual production of FMECA documents can absorb a significant amount of time during the system design stage, and is unable to provide system engineers with timely feedback for design modifications. For example, the FMECA worksheet for the IEEE 1394 interface design at JPL took 6 person-month to complete.

As the increased device complexity and reduced development cycle time collectively have made FMECA automation highly desirable, a number of commercial tools were developed in order to facilitate the time-consuming process of performing FMECA (see [1], for example). Nonetheless, the capabilities of those tools are generally limited to automating the process of organizing data, providing a graphical interface for users to enter their analysis results, and formating the reports according to Military Standards. More specifically, the major inadequacy of those tools is that they lack the capability of automating the most crucial part of FMECA, namely, the time-consuming and error-prone process of identifying failure modes.

With the motivation of transforming DFS practice from a traditional ad-hoc process that relies on error-prone, textual-document manipulation to a stringent engineering process, we develop a DFS workbench called the Risk Assessment and Management Environment (RAME). RAME comprises a static analyzer that accepts and processes design source code in hardware design language, a FMECA automation engine, a fault model, a knowledge base, a test reporting and failure tracking system (TRFTS), and a data mining tool (DMT-WEKA). The last two components together enable the knowledge base and fault model to be built and kept up-to-date along a project's life cycle and across successive projects, constituting a close loop that enables the information infrastructure to evolve.

Failure mode analysis is a widely used means for design validation. By identifying possible failure modes of devices used in a design and tracing their potential effects on system behavior at the application level, design inadequacies can be resolved in an early stage so that risks that have serious impact on mission success can be mitigated in a timely fashion. We view consistency, completeness, and accuracy collectively as failure mode analysis *integrity*. Accordingly, the specific objective of our RAME project is to improve both the integrity and turnaround time of design validation and to reduce the likelihood that design changes initiate at a late stage of a system life cycle due to overlooking a design flaw or a failure mode that may lead to a severe consequence. In addition, RAME intends to relieve system engineers from the tedious and error-prone process of manually generating FMECA worksheet and to enable them to gain time for design inadequacy mitigation.

The remainder of the paper is organized as follows. Section 2 presents the information infrastructure of RAME. Section 3 describes our approach to FMECA automation. Section 4 reports an initial experimental evaluation of the RAME prototype using the actual design source code of JPL's System Input/Output board. Section 5 concludes the paper.

2 Information Infrastructure

2.1 RAME Architecture

As shown in Figure 1, the architecture of RAME comprises six components which together enable high-integrity failure mode analysis. In particular, the HDL static analyzer and FMECA automation engine carry out failure analysis, with the crucial support from the information infrastructure consisting of the fault model, knowledge base, TRFTS, and DMT-WEKA. More specifically, TRFTS allows engineers to report test results and track failures based on a unified taxonomy and terminology, serving as the information source of the knowledge base and fault model, while DMT-WEKA extracts information concerning failure mode, cause, and effect from TRFTS for building and maintaining the knowledge base and fault model.

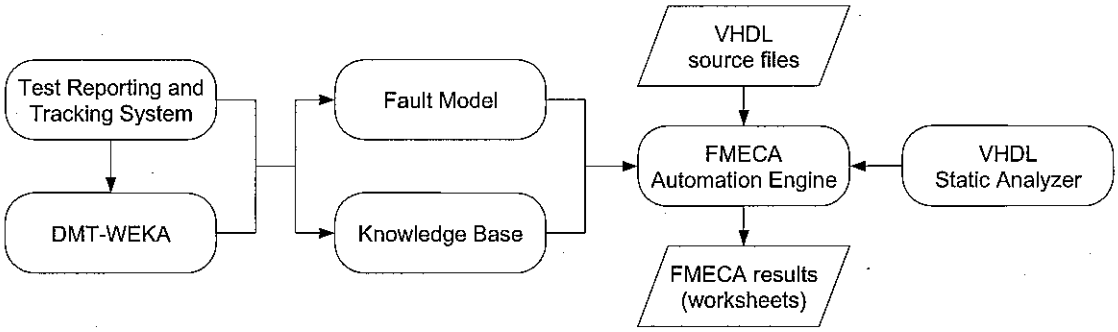

Figure 1: RAME Architecture

The goal of the information infrastructure is to 1) enable high-confidence FMECA automation by avoiding inconsistency, omission, and miscommunication, and 2) facilitate design, test, and reliability engineers to collaborate on risk assessment and management throughout a system's life-cycle. More succinctly, the information infrastructure addresses the following issues:

Presentation integration: Enforcing the use of *unified* taxonomy and terminology, starting from the user-interface level, to let all the components of RAME have a consistent view on the system under analysis and communicate correctly.

Information sharing: Based on the unified taxonomy and terminology, letting all the components of RAME collaborate share information. For example, appropriate entries in the test report database (a subsystem of TRFTS) are extracted for constructing and updating the fault models and knowledge base, which in turn, are employed for FMECA automation.

Assurance of data integrity: DMT-WEKA plays an important role in RAME for determining the relationships among device types, failure modes, and types of cause and effect. Evidence and confidence metrics are computed and used as the integrity indicators of the associations among those different entities.

Within the information infrastructure, the fault model defines the possible failure modes associated with each device type and the probable causes of a failure mode with which a device fails. For example, a device

such as a capacitor can be associated with failure modes open and short. The probable causes of "open" will include "Cracked solder joint" and "Anomalous overstress," whereas the probable causes of "short" will include "Part degradation."

The knowledge base supplies information necessary for interpreting the terms used in the HDL design source code under analysis and thus to enable FMECA automation. Knowledges encompass device types, signal types, and their characteristics that must consider when performing failure mode analysis. For example, by helping identify signal types during analyzing an interface board, the knowledge base will make the FMECA automation engine beware of certain signals are bus signals so that each of them must be analyzed as an array. Knowledges are established and continuously enriched by the off-the-shelf data mining tool WEKA which extracts useful information from TRFTS and translates them into organized knowledges.

2.2 Development of TRFTS

TRFTS is developed for recording, updating, and managing the reports concerning anomalies that are revealed during testing or simulation of avionics systems prior to launch at different levels, namely, chip, board, subsystem, and system. TRFTS is also the front end that enforces the use of a unified taxonomy and terminology for reporting failures. In turn, this ensures correct data exchanges among all the existing components in RAME and permits additional component tools to be developed/integrated into RAME without violating analysis integrity. The development of TRFTS is based on Web technologies, Java and XML, in order to create portable code and portable data that can run on any computing platforms including handhelds.

Specifically, test reports are be stored in XML format to enable knowledge discovery from test or simulation data. TRFTS consists of several server-side components implemented on the Java 2 Enterprise Edition (J2EE) platform. A number of JavaServer Pages (JSPs) are developed for TRFTS. The JSPs enable the user to access TRFTS using any Web browsers, such as the Microsoft Internet Explorer, Netscape Navigator, or open-source Firefox and Mozilla.

As XML schemas are designed to express shared vocabularies and allow machines to carry out rules defined by the designer, they are heavily exploited for the specification of failure report contents, structures, constraints, and semantics. As an example, Figure 2 shows an excerpt of the Failure Report XML Schema that specifies the *Failure Level* field and input-value choices for this field. The JSP translates this schema element into an HTML multiple choice element so that the user can select one of the permissible input values for the field. With this approach, updates of TRFTS, such as adding a field or a permissible choice of input value, can be easily done by modifying the appropriate XML schema. Finally, the test reports are organized as individual XML files and stored in the TRFTS database which supports data mining for knowledge discovery.

2.3 Data Mining: Building and Enriching Fault Model and Knowledge Base

Data mining supports FMECA Automation by building and continuously enriching the knowledge base and fault model. The data mining subsystem in RAME's information infrastructure employs the open-

```
<xsd:element name="Failure_Level">
 <xsd:simpleType>
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="Gate" />
   <xsd:enumeration value="Chip" />
   <xsd:enumeration value="Board" />
   <xsd:enumeration value="Subsystem" />
   <xsd:enumeration value="System" />
   <xsd:enumeration value="Mission" />
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
```

Figure 2: XML Schema of the Field for Failure Level

source-code WEKA[1] to process and visualize data, and to extract knowledge from TRFTS. WEKA contains constituent tools for data pre-processing, summarization, classification, regression, clustering, association, and visualization.

For the purpose of FMECA automation, we let WEKA perform data summarization, which is the abstraction and generalization of data, to aggregate the information in TRFTS concerning failures, causes and effects; and we let WEKA carry out data association that discovers the relationships between data items; for example, to relate failure modes to device types and probable causes to failure modes. Clearly, association analysis is the most essential data mining mechanism for automating FMECA.

Further, we regard the certain data items available in TRFTS (such as device types, failure modes, probable causes) as the universe and let each item type have a Boolean variable indicating its presence or absence in individual reports. Thus Each test report can be marked by a vector of Boolean values that reflects items that frequently appear together. These patterns can then be translated into the form of *association rules*. For example, the pattern in which a *resistor* tends to fail in the "open" failure mode and it is usually caused by "cracked solder joint" is translated into the following rule:

```
device=RESISTOR failure=OPEN ==> cause=CrackedSolderJoint
[support = 20%, confidence = 90%]
```

Levels of *support* and *confidence* are two metrics of rule validity. They respectively reflect the usefulness and certainty of discovered rules. A support of 20% for the above association rule means that 20% of all the test reports under analysis show that CrackedSolderJoint is the cause of OPEN failure for RESISTOR. A confidence of 90% means that 90% of the test reports that reveal RESISTOR has Open failure mode also indicate the failure mode is caused by CrackedSolderJoint. In RAME, association rules are considered valid and entered into the knowledge base or fault model if the relationships they state satisfy both a minimum support threshold and a minimum confidence threshold.

[1]WEKA stands for Waikato Environment for Knowledge Analysis, and is developed at the University of Waikato in New Zealand.

3 FMECA Automation

3.1 General Approach

Over the past few years, hardware description languages (HDLs) have been widely used in electronic design automation (EDA). Two HDLs, VHDL and Verilog have become the dominant *de facto* industry standard HDLs. The VHSIC Hardware Description Language (VHDL) offers a broad set of constructs for describing even the most complicated logic in a compact fashion. It can be used to model hardware systems and components, from gate level to system level. VHDL is designed to meet a number of requirements throughout the design process: 1) to describe how it is decomposed into subsystems and how those subsystems are interconnected, 2) to specify the functions of an system entity, using familiar programming language constructs, and 3) to allow the design of a system be simulated prior to implementation and manufacturing, which means that correctness verification can be performed without costly hardware prototyping.

Verilog also describes hardware systems with structure and behavior models similar to VHDL. Currently, most ECAD tools support both VHDL and Verilog [2]. Although our development is based on VHDL, all the techniques and tools implemented in RAME, including the FMECA automation engine, are compatible to both VHDL and Verilog except that an additional static analyzer will be needed for parsing design source code in Verilog.

Our approach to automating FMECA, as illustrated in Figure 3, is based on 1) the structural and behavioral information embedded in the VHDL source files, and 2) the fault model and knowledge base in the RAME information infrastructure.

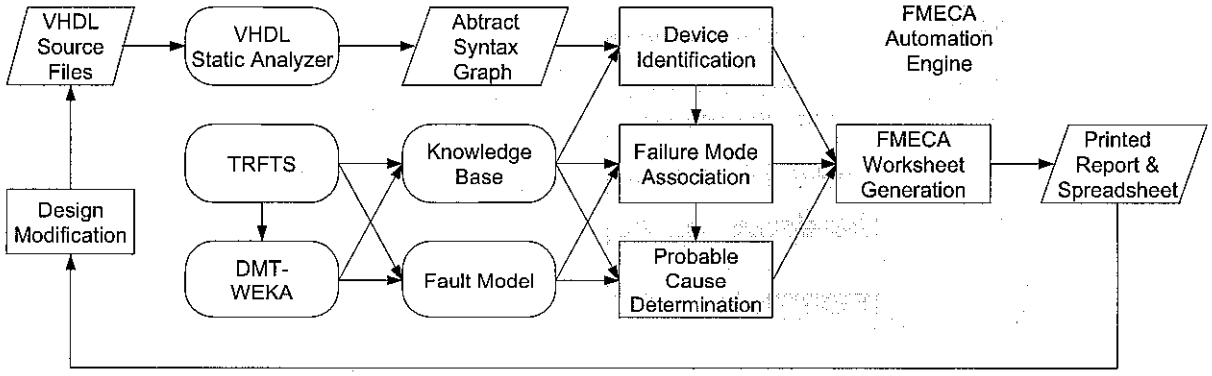

Figure 3: FMECA Automation

The VHDL source files are first analyzed by the VHDL Static Analyzer, which parses the design source code according to the formal grammar of VHDL. The analyzer is able to detect and report any syntax errors, resolves all the names used in the VHDL code to their respective declarations (including overloaded function names), and checks types used by each named elements. If no syntax or type-check error is found, the analyzer generates the *abstract syntax graph* which enables FMECA automation.

From the structural perspective, the FMECA Automation Engine traces the signals from outer layer of

a system under analysis (e.g., a connector of an SIO board under analysis) to the inner layer (e.g., an ASIC inside the SIO board), and from the higher level (e.g., the board level) to the lower level (e.g., the chip level). The engine takes the advantage of a VHDL specification in which signals can be identified from the port list of the system under analysis (e.g., all the signals at the connectors), or from a list specified by the user. In particular, a recursive trace is based on the design's architectural information encoded in the VHDL files. The trace stops when it reaches "terminals" such as power, ground, connector, or an entity that has no further elaboration (e.g., an IP-based ASIC which does not have a VHDL design file for architecture specification).

Through the trace, the FMECA automation engine collects all the devices (e.g. resistors and capacitors) along the paths. It subsequently exploits the information in the knowledge base and fault model to generate the FMECA worksheet.

3.2 VHDL Static Analyzer

RAME's VHDL Static Analyzer (VSA) is able to analyze IEEE VHDL-93 compliant designs, such as JPL's SIO board design (see Section 4). VSA consists of two components, namely, a *lexical scanner* and a *syntax interpreter*. The lexical scanner groups characters in a VHDL source code file into tokens, which are *keywords* defined in VHDL specifications, such as `entity` and `port`, *identifiers* used in the design files such as `DIOA_CLK` and `I2C_0_ID`, *VHDL operator symbols* such as `=>` and `:=`, and VHDL *punctuation symbols* such as parentheses and commas

Hence, the output of the lexical scanner is a stream of tokens which is subsequently passed to the syntax interpretor. According to the VHDL grammar, the syntactic interpretor then organize the tokens to form syntactic structures. For example, the three tokens "`TDI`", "`=>`", and "`MSIO_A_TDI`" are grouped into a syntactic structure called an *expression*. Expressions are in turn combined to form statements, etc. Those syntax structures eventually constitute an *abstract syntax tree* whose leaves are the tokens produced by the lexical scanner. More specifically, the abstract syntax tree is represented by a collection of C++ objects tightly connected by pointers. With the tree structure, the objects can then be accessed by a set of C++ methods for FMECA automation. Figure 4 shows a portion of an abstract syntax tree generated by the syntax interpretor (for the design validation of JPL's SIO board).

3.3 FMECA Automation Engine

The FMECA automation engine (FAE) automates the otherwise labor-intensive process of performing failure modes, effects and criticality analysis of hardware designs, by exploiting the information infrastructure as follows:

- Use the C++ methods to access those design information that is organized into an abstract syntax tree, including the entities that comprise the design, the architectural structure of those entities, and the input/output signals of each entity.
- Use the information about system parts, signal types, and failure modes that are accommodated in the knowledge base. The system parts allow FAE to identify the type of device (e.g., `CDR31_22pf` is a capacitor). The signal types allow FAE to identify power and ground signals (e.g., `I2C_A0_3.3VRTN` is

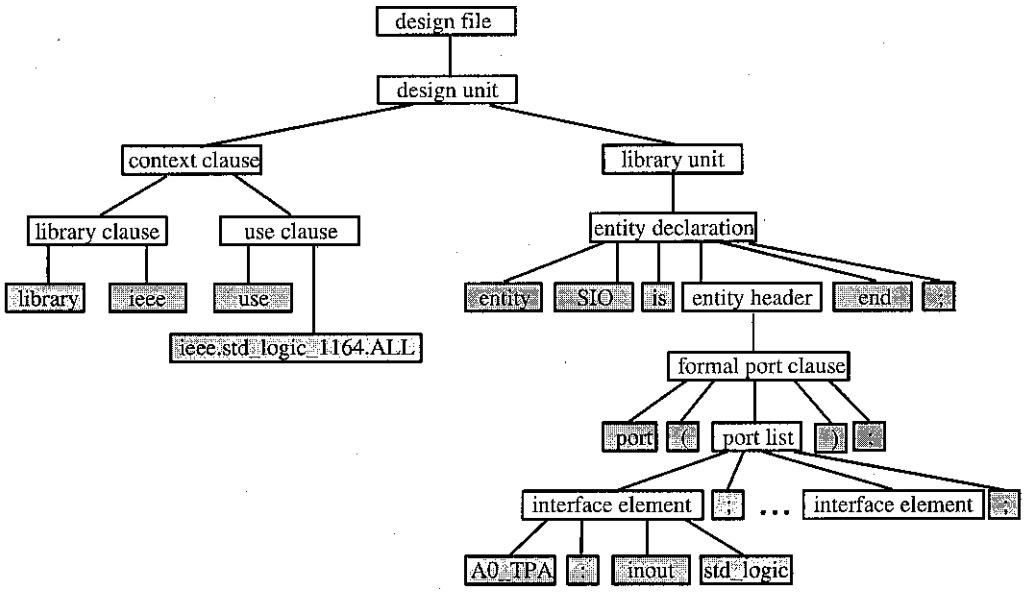

Figure 4: Portion of the Abstract Syntax Tree for the VHDL SIO Design

a ground signal). Knowledge about which of the bus signals requires an array data structure for the automated analysis can also be found in the knowledge base (e.g., treats `AD` as a bus signal `AD(31:0)`). The failure modes allow FAE to associate possible failures with each device type (e.g., the failure mode of a `resistor` is `Open`).

- Use the association rules that are provided by the fault model to determine the relationships among device types, failure modes, and probable causes.

Device Identification

The first step in the automated FMECA process is to identify all the devices (i.e., the structure primitives at a given abstraction level) whose failures may affect system operation. In VHDL, a module (which could be an ASIC, a board, a subsystem, or a system) is described using an *entity declaration* which specifies the interface in terms of *ports*, and the connecting points for input and output signals. FAE traces the signal one by one according to the list of signals (i.e., the *port list*), from the outside of the module to its inside until it cannot trace any further, that is, it reaches the power, ground, connector, or a terminal entity, such as an ASIC that does not have a VHDL design file. But when FAE encounters a device which does have a VHDL design file, it will trace down one level further. During the trace, FAE collects all the devices along the trace path and to constructs a list of devices for that signal.

Since we are interested in the type of a device (e.g. `capacitor`) instead of the specific device (e.g. `CDR31_22uf`), the FAE uses the System Parts information in the knowledge base to look up the type of a device. The System Parts information is specific to the target system and it can be either obtained from the project design engineers or through data mining of the test reports.

Failure Modes Identification

After a list of devices is established, FAE relates particular potential failures to each of the devices, based on the association rules for devices and failures that are discovered through data mining in TRFTS and stored in the fault model.

Another piece of information we use during FMECA is the mode of each signal. The VHDL declaration of each signal includes its mode, such as `in`, `out`, and `inout`. This information allows FAE to identify whether the failure is for the input or output of a particular device, for trace purpose.

Probable Causes Identification

The probable causes of failures are extracted from TRFTS and stored in the knowledge base. From TRFTS, DMT-WEKA also produces the association rules for the fault model which relate the probable causes to a failure mode of a particular device type. Accordingly, based on the fault model, FAE determines the probable causes for the failure modes of every identified device, signal, and failure mode.

Multi-Level Trace and Failure-Mode Effect Identification

An avionics system is usually designed as a hierarchical collection of modules. ECAD tools that have the ability to generate hierarchical models are increasingly used for design specification, which results in a trend that VHDL source files become available not only at the chip or board levels but also at the subsystem and system levels. For example, the ECAD tool I-Logix Statemate Magnum which is able to generate hierarchical system models in VHDL and Verilog is used for avionics architecture design at JPL [3]. Accordingly, it is important that FAE has the ability to trace the devices, their signals and the effects of potential failures from one level of a design to the next.

FAE uses a recursive algorithm to perform top-down trace of failure effects across multiple levels. When a signal is traced to an entity which is a VHDL structure, FAE loads the VHDL file of that entity and traces the signal through that structure. Figure 5 shows an example in which the trace starts from the connector `CONN1394` at the SIO board level and ends at the point when the `MSIO_A` ASIC is reached. Note that when the trace reaches the terminator `MGC1394_A_TERM` (at Segment 2 in the path of the trace), FAE recognizes that it is a structure at one abstraction level below (i.e., the chip level next to the board level) and thus loads the `MGC1394_A_TERM.vhd` file to trace the internal structure of the entity `MGC1394_A_TERM`. FAE then reaches the resistor `RM1005_55ohm` and the capacitor `CDR33_0_0461uF_50V` at the chip level. In Figure 5 we can see the trace of Segment 2 branches into Segments 3 and 4 which are in a chip-level structure in the VHDL hierarchical design model. After the chip-level trace, it comes back to Segment 2 at the board level and continues until it reaches `MSIO_A` ASIC.

It is worth to note that `MSIO_A` must be treated as a termination entity because it is defined as a `black_box` in the VHDL design source code for the SIO board. That means, there is no further architectural information about the entity (i.e., an IP-based ASIC for which no design details are supplied) available in the VHDL design source code.

```
Segment 0: SIO_x_036X2000->A0_TPA
Segment 1: CONN1394->A0_TPA
Segment 2: MGC1394_A_TERM->A0_TPA
Segment 3: RM1005_55ohm->MGC0_TPA
Segment 3: RM1005_55ohm->local_TPBias0
Segment 4: CDR33_0_0461uF_50V->local_TPBias0
Segment 4: CDR33_0_0461uF_50V->local_1394_3_0463VGND
Segment 2: MGC1394_A_TERM->local_1394_3_0463VGND
Segment 5: MSIO_A->A0_TPA
```

Figure 5: Tracing of the A0_TPA Signal

When the VHDL design files at all levels of the design are available, this recursive algorithm will exhaustively collect the information that will enable FAE to analyze the effects of a failure mode of a particular device D (or an entity of another type). More specifically, the top-down trace starts from D at the top level and goes down level by level successively until reaching a termination entity. Along the path of the multi-level trace, all the entities that D has dependency with are entered into a tree-type data structure. Upon the completion of the trace for every top-level device, FAE searches through the information organized in the data structures in a bottom-up manner, summarizes the failure modes (a collection denoted as F) of the top-level entities that are related to a failure mode f of a bottom-level entity e, and concludes that F is the top-level effect of the failure mode f of device e. From a view similar to that suggested by [4], the usage of the above algorithm can be generalized. More succinctly, a failure mode f_n of an entity e at design abstraction level n will be 1) an effect of a failure mode f_k of an entity e' at level k ($k < n$), and 2) a cause of a failure mode f_m of an entity e'' at level m ($m > n$), if the trace results indicate such dependencies.

Finally, FAE is able to access the criticality and system-level effect of the failure mode of a particular device by considering the fault tolerance mechanisms provided by a VHDL design and by taking into account the redundant components identified during the recursive trace procedure. For example, in the trace shown in Figure 5, FAE recognizes that the failure modes of the resistor RM1005_55ohm and capacitor CDR33_0_0461uF_50V can affect the signal A0_TPA of the SIO Board. In terms of the SIO design, that implies a disabled port 0 in the 1394A bus. Nonetheless, this is a failure mode of low criticality because the 1394A bus architecture implements three redundant ports so that a single disabled port will not have a system-level effect.

4 Experimental Evaluation of RAME Prototype

In order to validate the RAME prototype, we carry out an experimental evaluation by applying the prototype to an actual VHDL design, namely, the System Input/Output (SIO) board which implements a fault-tolerant bus interface. Among other objectives, the experimental validation address on the following:

Data integrity: To examine whether RAME can ensure that data are entered into TRFTS and utilized for FMECA automation correctly and consistently. This is indeed to validate the effectiveness of enforc-

ing the use of unified taxonomy and terminology.

Completeness: To examine whether the model- and knowledge-based FMECA automation is able to exhaustively identify the failure modes of a VHDL design.

Accuracy: To examine whether FMECA automation is able to produce correct entries for the failure mode of each device and the probable causes of those failure modes.

Turnaround time: To examine whether FMECA automation can significantly reduce the time required to perform a design validation.

4.1 Overview of SIO Board

The VHDL design of the SIO board selected for RAME evaluation implements the IEEE 1394 and I2C bus interfaces among the flight computers and microcontrollers in the X2000 avionics system, and their internal PCI bus interface [5]. The architectural structure of the board is shown in Figure 6.

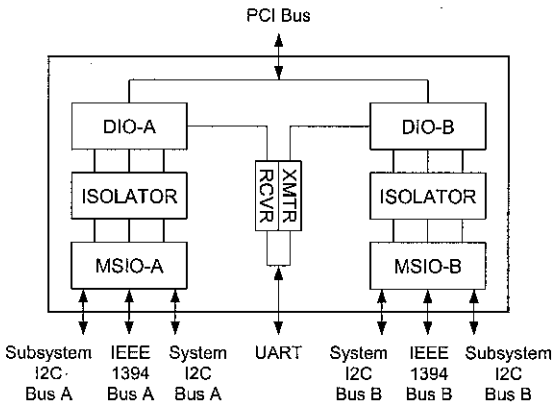

Figure 6: Architectural Structure of the SIO Board

As shown in the figure, the SIO board consists of two redundant bus interface units, each of which has a 3-port IEEE 1394 bus interface and two I2C bus interfaces. The bus interface unit is composed of two ASICs, the Digital I/O (DIO) ASIC and the Mixed Signal I/O (MSIO) ASIC. The DIO ASIC implements the link layer of the IEEE 1394 bus, the two I2C bus controllers, and the logic for fault tolerance mechanisms of the buses; the MSIO ASIC serves as the physical layer in the overall data-communication architecture and implements an analog interface to the IEEE 1394 and I2C bus cables.

4.2 Types of Analysis Output Document

To evaluate RAME's enforcement of data integrity, we manually enter into TRFTS the data from test/simulation results concerning the VHDL SIO board. The reports are then processed for DMT-WEKA to create the knowledge base and fault model which enable FAE to analyze the SIO VHDL files and to generate the FMECA worksheet. The FMECA worksheet so generated are then imported into Microsoft Excel.

In addition, FAE provides three options for the types of output documents The following are the three different types of worksheets that can be generated using the different options, namely, 1) *signal-list worksheet* which is generated based on a list of signals; we use this option in the RAME experimental evaluation because the FMECA worksheets manually produced by JPL engineers was based signal lists, 2) *port-list worksheet* for which FAE analyzes a VHDL modules by tracing the signals based on a list of ports, and 3) *complete-list worksheet* which allows the user to choose not to include the Open failure mode if there is a pullup resistor; that failure mode is traditionally intentionally ignored in JPL's traditional manual FMECA practice. Figure 7 shows the Web interface which allows us to submit a VHDL design source code file and make choices of output document type.

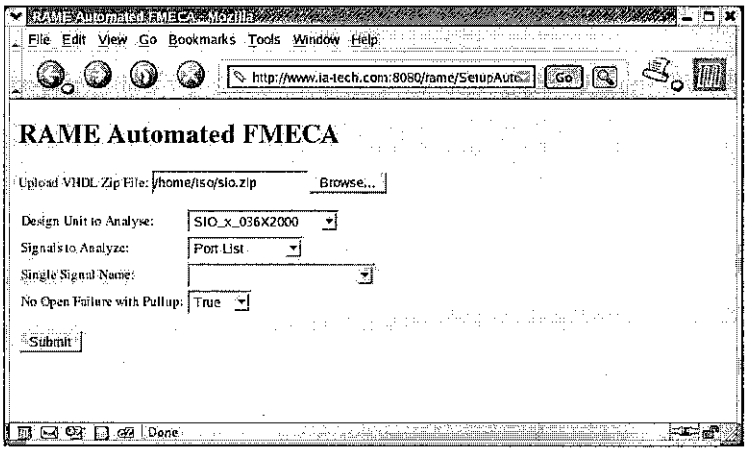

Figure 7: Web Interface to FMECA Automation Engine

4.3 Results Evaluation

We conduct the experimental evaluation a PC/Linux platform in which the PC is a Dell Dimension 8200 desktop computer with a 2.8 MHz Pentium 4 processor and 1 GB of memory running Red Hat 9 Linux. The SIO design source code consists of 14 VHDL files with a total of 17,854 lines of code.

We exercise all the three different options explained earlier for the experimental validation of FMECA automation. While the signal-list and port-list options both yield a FMECA worksheet of 872 lines in 1.27 seconds, the complete-list option results in a worksheet that has 101 more entries (since it includes the Open failure mode of the chips, as discussed above) and takes 1.29 seconds.

In order to evaluate data integrity, completeness, accuracy, and turn-around time of FMECA automation, we compare the worksheet produced by FAE with that generated manually at JPL. The comparison results show that while the JPL worksheet has 689 items the Signal-List Worksheet which is automatically generated based the same VHDL design source code, has 872 items. By carefully inspecting all the entries of the two worksheets, we become able to explain where those differences come from:

1. The manually generated worksheet missed a number of internal signals. For example, in analyzing the `UARTA_SIN_P` signal, the manually generated worksheet only has three items for the Open, ShortVCC, and ShortGND failure modes.

```
26CLV32RH input UARTA_SIN_P Open
26CLV32RH input UARTA_SIN_P Input shorted to P3.3V
26CLV32RH input UARTA_SIN_P Input shorted to P3.3VGND
```

But the UARTA_SIN_P signal connects to the SIN pin of the DIO ASIC chip after it passes through the driver 26CLV32RH. As a result, there should also be Open, ShortVCC, and ShortGND failure modes for the SIN signal at the DIO_ASIC_A, as identified by FAE:

```
26CLV32RH input for UARTA_SIN_P Open
26CLV32RH input for UARTA_SIN_P Input shorted to P3.3V
26CLV32RH input for UARTA_SIN_P Input shorted to P3.3VGND
DIO_ASIC_A input for SIN Open
DIO_ASIC_A input for SIN Input shorted to P3.3V
DIO_ASIC_A input for SIN Input shorted to P3.3VGND
```

Since there are a total of 28 such signals, the automatically generated worksheet has 84 more items.

2. The manually generated worksheet omitted the trace of both paths in the 1394 terminators. For example, in analyzing the A0_TPB signal, that worksheet only considered the path of the 55ohm resistor and 270pF capacitor. But it was neglected that there is another path in which the 55ohm resistor connects to a 5K resistor (part RM1005_5K). If this 5K resistor fails and becomes open, the effectiveness of the terminator will degrade. Therefore, the FAE-generated worksheet has one more item to take into account for the failure of the 5K resistor. Moreover, besides A0_TPB, there are many such signals passing through the terminators, which contributes to the discrepancy between the manually and automatically generated worksheets.

3. When analyzing the power and ground signals, e.g., 1394A_3.3V[2], the manually generated worksheet uses one item ("Any single capacitor between 1394A_3.3V and 1394A_3.3VGND") for all the capacitors connecting the 1394A power and ground, while the automatically generated worksheet enumerates exhaustively all the capacitors affecting the signal.

4. When analyzing the MSIO_A_TMS signal, the automatically generated worksheet identifies a 7K resistor (part RM1005_7K) while the manually generated worksheet considers it as a 10K resistor. By inspecting the VHDL design source code, it is confirmed that SIO_x_036X2000.vhd is a 7K resistor, as identified by FAE.

5. There are two signals (P3.3V and P3.3VGND) analyzed in the manually generated worksheet but they do not appear in the port list of the VHDL file. After examining the schematic diagram, we confirm that those two signals are not designated as the ports of the SIO board. Consistently, those signals are not included in the port list of the VHDL design source code, and consequently, they do not appear in the automatically generated worksheet. It is very likely that the discrepancy in the manually

[2]The VHDL files created by Mentor Graphics use slightly different names, e.g. MGC1394A_3_0463V is used instead of 1394A_3.3V. The FAE worksheets use the names in the VHDL files. But for the discussion in this report, we change them to the ones used in the JPL worksheet.

generated worksheet was caused by neglecting to make all the updates in the worksheet according to the updated version of the SIO board design (where the two signals have been eliminated).

Aside from these differences, the worksheet generated by FAE is the same as the manually generated worksheet. More specifically, as described in the above, automated FMECA is superior to manual FMECA, with respect to completeness and accuracy. Moreover, automated FMECA is able to generate accurate entries in a worksheet so long as the VHDL file is free of editorial errors. Finally, in terms of turnaround time, generating each of the three worksheets (using signal list, port list, and complete list) took less than 1.5 seconds. Clearly, FMECA automation will lead to significant improvement in design and design modification efficiency.

5 Concluding Remarks

As described in the paper, RAME's information infrastructure enables not only FMECA automation but also continuous enhancement of knowledge about failure modes, causes, and effects, which in turn, ensures failure mode analysis efficiency and integrity. In summary, RAME is distinctive from those previously developed failure mode analysis tools in several respects as follows. First, none of those previously developed tools accepts HDL design source code as input file for analysis, which seriously prevents those tools from being compatible with other ECAD (electronic computer-aided design) tools and from becoming a widely accepted tool in industry. Therefore, the ability of RAME to accept HDL design source code as FMECA input is an important advantage.

Second, the extent of automation implemented in existing tools is generally very limited when they are compared with RAME. In particular, most of them are intended to escort a manual FMECA process by supplying online menu, input checking, choice of library functions, and/or facilitating the post-analysis documentation process by providing automatic plotting/graphing capabilities. In contrast, RAME enables the automation of the analysis aspect of a FMECA process and emphasizes to improve the integrity of the FMECA process by using model- and knowledge-based techniques. Further, with respect to the means of automation, most previously developed automation tools for failure mode analysis were implemented to emulate the manual process of failure analysis (see [6], for example). In contrast to those "cookbook-based" approaches, we use a semi-formal approach to the automation by directly processing HDL design source code based on its grammar, enhancing analysis completeness and consistency.

Finally, while none of the existing tools addresses the issue of automatic learning from previous projects, RAME's self-enriching information infrastructure continuously enhances the coverage and integrity of failure mode analysis and gives to RAME the most unique advantage.

References

[1] C. J. Price, "Effortless incremental design FMEA," in *The Proceedings of Annual Reliability and Maintainability Symposium*, (Las Vegas, NV), pp. 43–47, 1996.

[2] V. Berman, "Standard Verilog-VHDL interoperability," in *Proceedings of the VHDL International Users Forum (VIUF)*, (Oakland, CA), pp. 142–149, May 1994.

[3] S. Chau, R. Hall, M. Traylor, and A. Whitfield, "Using COTS tools in the development of a model based avionics architecture tool," in *Proceedings of the International Council on Systems Engineering*, (Melbourne, Australia), July 2001.

[4] J.-C. Laprie, editor, *Dependability: Basic Concepts and Terminology*, vol. 5 of *Dependable Computing and Fault-Tolerant Systems*. Vienna, Austria: Springer-Verlag, 1992.

[5] S. Chau *et al.*, "The implementation of a COTS based fault tolerant avionics bus architecture," in *Proceedings of IEEE Aerospace Conference*, (Big Sky, MT), Mar. 2000.

[6] D. Palumbo, "Automating failure modes and effects analysis," in *The Proceedings of Annual Reliability and Maintainability Symposium*, (Anaheim, CA), pp. 304–309, 1994.