Validating The Safety Of Embedded Real-Time Control Systems Using FMEA

Peter L. Goddard; Hughes Aircraft, Fullerton

Key Words: Software safety, Software FMEA, FMECA, Software failure modes

SUMMARY AND CONCLUSIONS

Validating embedded real time systems for use in safety critical applications is difficult for most applications. When these systems are based on commercially available microprocessors and/or microcontrollers, the validation task can be made significantly more difficult by the lack of basic data integrity protection on board the processor and peripherals. Additionally, basic address boundary protection may not be provided by the real time scheduler being used. Hardware FMEAs need to trace faults through their effect on the software. Additionally, the software design, including the real time scheduler or operating system, needs to be completely analyzed to ensure that hardware data integrity failures and software failures cannot cause the control processing to place the controlled system into an unsafe state.

The techniques needed to perform hardware FMEAs are well known in the reliability engineering discipline. However, techniques which will allow the validation of software are not well known and are difficult to apply. A variety of software safety analysis techniques have been developed, including software fault trees and time Petri nets. These techniques attempt to assess the correctness of the software design when it is operating on unfailed hardware. All software analysis techniques are severely limited when the integrity of the data being processed cannot be guaranteed.

Hughes Aircraft has adapted and extended traditional FMEA techniques to include assessment of software failures. Hughes has been using the resulting technique to assess the safety of embedded real-time control systems designed for use in automotive applications. The use of FMEA techniques in assessing the software safety of these controllers has allowed analysis of the effects of a more comprehensive set of potential failures, including data corruption, than is practical using other software safety analysis techniques. The ability to assess the results of data corruption has proven to be crucial in providing feedback to design teams about the potential safety risks of the designs being analyzed.

1. *INTRODUCTION*

Processing elements which provide safety critical control functions need to be analyzed to determine if their contribution to the probability of system hazard occurrence is within acceptable limits prior to allowing their use in deployed systems. These assessments take the form of analyses and tests designed to assess the safety of the control function when no hardware failures are present, and to determine any fault sequences which can result in the occurrence of defined system hazards. The results of these assessments are then used to determine a numerical probability of occurrence for each system hazard. Alternately, when numeric probability figures cannot be calculated, the number of independent faults which must occur to cause each hazard can be used as a figure of merit for system acceptance. Analysis results are compared to system requirements to determine the acceptability of the control processing design. These requirements can be specific regulatory requirements, customer specification requirements, or may be derived from less specific sources, such as liability considerations. For any safety critical design, the developer has an obligation to ensure that the processing design achieves the needed safety in the expected use environment. System safety must be maintained under normal, unfailed, operation, and with failures present in the processing hardware and software. Validating that small, real-time, embedded control processing systems achieve the needed safety can be difficult due to the close coupling between the processor hardware and software. Failure modes and effects analysis (FMEA) provides a cost effective way of assessing the impact of both hardware and software failures in embedded control systems.

2. *SOFTWARE VALIDATION*

2.1 Verification Approaches

Analytical verification methods for assessing the effect of hardware failures are well known within the reliability discipline. Failure modes and effects analysis [1] and fault tree analysis (FTA) [2] have been used to assess many safety critical hardware systems and are proven methods. Analytical verification methods for software exist, but are not as well known within the reliability discipline. Examples include Software fault tree analysis [3,4], Petri Net analysis [5] , and Time Petri Nets [6]. Each of these analysis methods can provide an assessment of the software design under conditions of healthy processing hardware. However, software coupled failure effects, induced by hardware failures which compromise data integrity, cannot be assessed using these analysis techniques.

Hardware failures which compromise processing system data integrity are not a concern for most large systems, or for small systems with adequate protection built into the hardware and operating systems. When the hardware provides a carefully constructed memory architecture, requiring multiple failures to impact more than one bit of any word, parity schemes can be used to ensure any memory corruption is readily detected. Similarly, parity on busses and registers, arithmetic residue codes, cyclic redundancy codes, and similar techniques can be used to ensure detection of data integrity loss. However, many commercially available microcontrollers do not provide support for data integrity protection. Also, some real-time

0149-144X/93/$3.00 © 1993 IEEE

operating systems and schedulers are relatively unsophisticated and cannot make use of the on-board data integrity features which are available. Using microcontrollers in embedded, real-time control processing applications allows the designer to take advantage of device resident specialized features which can include communications, pulse width modulated signal generation, input buffering, and analog to digital conversion. Using the wide range of device resident functionality available can result in development and production cost savings in addition to requiring less space. These savings can be important in high volume and/or space constrained applications, particularly automotive applications. However, safety critical control systems which use these devices must be carefully designed and their safety thoroughly validated prior to use.

Embedded control systems for safety critical applications require designs which protect against hardware failures, software failures, and failures which cross the hardware/software boundary; the system must never be allowed to enter an unsafe state. Analytical methods applied to validate these system must allow the analyst to assess the safety of normal operations and the system impact of both hardware and software failures. Extensive performance testing can provide partial verification of system and hardware performance under the expected use environment when failures are not present. Petri net analysis can be used to assess the potential for the control design to enter an unsafe state or to deadlock during normal operation. FMEA applied to hardware allows assessment of the effects of single point hardware failures. FMEA can be extended to consider some multiple fault conditions using Event Sequence Analysis [7]. Software and system performance testing, fault trees, and Petri net analysis of the software design allows software operation on unfailed hardware to be assessed. FMEA applied to software allows assessment of the impact of single point software failures and of those failures in hardware whose effects are determined by the software. For systems where undetected data hardware integrity failures are possible, software FMEAs have significant advantages over software fault tree analysis. Software FMEAs are inductive, thus they ensure that the analyst has assessed the impact of all potential failures. They are also smaller than software fault trees, thus more easily managed, for systems where hardware data integrity is not assured. Software FMEAs provide a unique addition to the analytical tools available to the system safety analyst.

2.2 Software FMEA

Software FMEA is performed based on a software hazard analysis which establishes a mapping between the defined system hazards and the software states which can cause the hazards. The analyst then develops failure modes for the input variables and for the software logic for each software routine. An FMEA is performed on each software routine to map the input variable and software logic failure modes to the output variable failure effects for the routine being analyzed and then to the critical software variable states which were established by the software hazard analysis. If the resultant effects of any failure mode maps to a set of critical variable values which are defined as hazardous, then that failure is a potential single point hazard cause.

Software Hazard Analysis. Performing a software FMEA requires that the system hazards be translated into software compatible terms so that the effects of software failures can be evaluated which respect to the system hazards. This is the first step in the software FMEA process. Each defined system hazard which could be caused by the control processing is translated into a set of software variables and associated values which directly equate to the hazard at the system level. A fault tree analysis of the system design is performed to identify the mapping between system hazards and software variable values. The fault trees use the system hazards as the root undesired event. The fault tree logic is developed deductively until the specific hardware control signals and the associated values which relate to each hazard and are identified. The fault trees are then extended into the top level software design to identify the software variables and values which map to the system hazards. The level of design detail at which to end the fault tree mapping of the system hazards to software variables is system dependent. The optimum level of detail is one which provides a mapping to the minimum size set of variables which are independent of each other and which fully describe the hazard occurrence conditions in software variable terms. Experienced fault tree analysts should recognize the appropriate level of detail without difficulty. The results of this fault tree analysis are then documented in tabular form, as shown in Figure 1. This table, mapping between the system hazards and the software design, allows the analyst or analysis team to readily identify which software failure effects, if any, relate to each hazard.

		Critical Software Variables			
		Variable 1	Variable 2	• • •	Variable n
Hazard 1	Cause 1	Value	Value	• • •	Value
	Cause 2	Value	Value	• • •	Value
	⇓	⇓	⇓	⇓	⇓
	Cause n	Value	Value	• • •	Value
Hazard 2	Cause 1	Value	Value	• • •	Value
	Cause 2	Value	Value	• • •	Value
	⇓	⇓	⇓	⇓	⇓
	Cause n	Value	Value	• • •	Value
⇓	⇓	⇓	⇓	⇓	⇓
Hazard n	Cause 1	Value	Value	• • •	Value
	Cause 2	Value	Value	• • •	Value
	⇓	⇓	⇓	⇓	⇓
	Cause n	Value	Value	• • •	Value

Figure 1. SW Hazard Analysis Results

Input Variable Failure Modes. The analyst must develop a set of failure modes for all input variables, by type, and for the software logic of each routine which is to be analyzed. These failure modes must be logically complete, allowing all possible failures to be assessed. Failure modes for software variables, as shown in the partial list of Figure 2, are the set of potentially erroneous values for the variable type. For example, a boolean variable has the failure modes of true when it should be false and false when it should be true. The only other possible values for a boolean variable are true when it should be or false when it should be; these are both correct values. A similar process is followed in developing failure modes for other variable types. An analog variable, integer or real value, is either correct (within tolerance) or has the failure

mode of above correct value or below correct value. The failure mode set, when combined with the success state(s) for the variable, is a logically complete statement of all possible variable states. Failure modes for enumerated variable types are handled in a similar manner, however, the number of possible failure modes can be large if the number of possible enumerated values is large and the values are not part of a rank ordered set. Failure modes may also need to be developed for closely coupled variable sets. For example, an analog variable which contains an measured sensor input is sometimes coupled with a validity flag, requiring the analyst to consider both variables together during the analysis.

Variable Type	Failure Mode/Effect	Code
Boolean	True When It Should Be False	T
	False When It Should Be True	F
Boolean With Validity Flag	True When It Should Be False – Validity Flag Set to Valid	TV
	False When It Should Be True – Validity Flag Set to Valid	FV
	Validity Flag Set to Invalid With Variable Value Correct	I
Analog (Integers, Reals)	Value Error > Tolerance – High	H
	Value Error > Tolerance – Low	L
Analog With Validity Flag	⇩	⇩

Figure 2. SW Variable Failure Modes

Software Logic Failure Modes. The analyst develops software logic failure modes for each routine being analyzed based on the logical functionality of the routine. For example, if a routine calculates a value it will have the failure modes of calculates value high and calculates value low. When combined with the success state of calculates value correctly, a complete logical set of the possible outcomes of the assigned functionality "calculates a value for" is formed by the failure modes. The analyst determines logic failure modes for each function which is assigned to the processing of each routine being analyzed. These failure modes are then used, in conjunction with the failure modes for the input variables, as the software failure modes for the FMEA.

Output Failure Effects. The output failure effects for each module are based on the variable types which are either defined outputs of the routine being analyzed or are global variables which are accessed by the routine. If global variables are accessed by the routine, software errors or hardware failures could potentially change their value. The failure modes for the output variables of a routine are based on their type and are the same as those developed for the failure modes for input variables. Additional failure modes will need to be developed to allow assessment of failures which cause either no effect or which cause a correct effect under conditions of failure. For example, if an analog variable is calculated incorrectly, but the incorrect value is detected by the software and the variable's validity flag is set to false, the software is performing correctly. However, the effect of this condition needs to be traced through the software to ensure that the validity flag is read and appropriate actions taken. The output failure effects for correct actions under conditions of failure will need to be developed for each routine being analyzed and carefully assessed as an input failure mode at higher levels of indenture.

Software FMEA Methodology. Software FMEA is similar in structure and approach to hardware FMEAs. However, as discussed above, the failure modes used for assessing software are different. The failure modes for software are based on logical synthesis of the possible error states instead of on physical failure phenomena. Also, the mapping between software elements is based on the software variables instead of wiring diagrams. Because of the highly modular structure of software, software FMEAs fit naturally into a matrix notation. Matrix FMEA **[8,9,10]** approaches yield a highly compact FMEA notation which is compact, and can be assigned to multiple analysts. Also, the logical structure of software minimizes the need for extensive remarks and explanatory material, a known limitation of matrix FMEA approaches **[10]**.

To perform the FMEA, the analyst structures a matrix with input variable failure modes and software logical failure modes along the vertical axis of the matrix and output variables along the horizontal top row of the matrix. As shown in Figure 3, the matrix is then completed by inserting the codes for the appropriate failure effect at the junction of the input failure mode being analyzed and the output variable which is being effected. Each failure mode is assessed until the matrix for the routine being analyzed is complete. The output variables of the routine are then used as a mapping to the input of other software routines. If the system and software have been developed using either structured systems analysis **[11]** or the real time system design approaches of reference **[12]**, the data flow diagrams, developed during the software design, provide a mapping of the variables between modules. Once the FMEA matrices for all routines have been completed, the output effects of each matrix are mapped to either the input of one or more software routines, or to one or more of the critical software variable values identified by the software hazard analysis. If a failure mode maps to a set of critical values defined by the software hazard analysis, then the failure mode causes the hazard associated with that value set. The performance of software FMEA closely parallels the methods described in reference **[10]**, and can easily be supported by automated aids, such as data base managers.

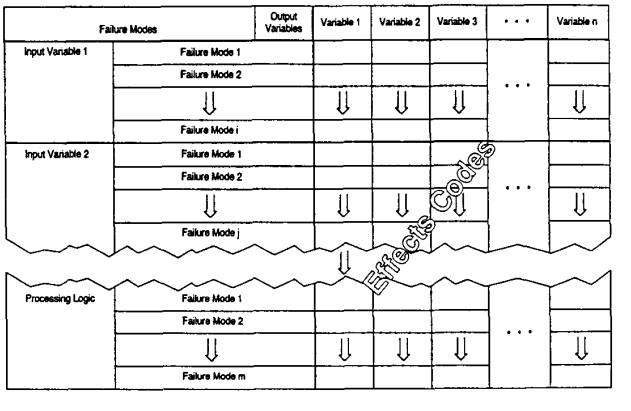

Figure 3. SW FMEA Matrix

Software FMEA can be performed at many levels of design detail. The only requirement is that the software input and output variables along with the functional assignment to each software element be identified. Final code is not required for software FMEA. Performing the software FMEA as early as practical in the design process allows early identification of single point failures which can lead to hazards. The software, and possibly the hardware, design can then be changed to ensure that the system delivered for use cannot be driven to an unsafe state by any single point software failure. If a thorough hardware FMEA has been performed on the system, the design can be tailored to ensure that no single point failure, hardware or software based, can cause system safety to be compromised.

Analysis Limitations

Software FMEA can accurately and effectively identify software design and data integrity based vulnerability to hazards. The analysis technique, when combined with hardware FMEA, provides an invaluable tool for assessing the behavior of an embedded control system under single point failures. However, FMEA must be supplemented by analyses and tests designed to ensure that the control system cannot enter a hazardous state during unfailed operation. Also, simulation of failures which impact control loop constants may be needed. A static analysis, such as software FMEA cannot fully assess the dynamics of control loops. Detailed timing analysis to identify any possible race conditions may also be needed. If the applications software is not running under a known, certified, real-time executive, analysis of the executive software will be required.

3. *CONCLUSIONS*

Failure modes and effects analysis is a crucial part of the validation analyses required for safety critical, embedded, real-time control processors. When hardware FMEAs are combined with software FMEAs, a complete assessment of system response to single point failures results. This assessment can be used to help direct the team assigned to the control processing design toward robust hardware and software designs which can successfully provide safety critical services in an environment where hardware data integrity is not assured. When hardware and software FMEAs are supplemented with analysis techniques which assess operation under normal conditions and simulation of dynamic timing and failure occurrence conditions, a complete verification and validation of the safety characteristics of embedded, real-time, control systems can result.

REFERENCES

1. Military Standard 1629A, *Procedures for Performing a Failure Mode, Effects and Criticality Analysis*, 24 November 1980.

2. Vesely, W. E. *et al*, *Fault Tree Handbook* U. S. Nuclear Regulatory Commission, NUREG-0492, January 1981.

3. Harvey, P. R., *Fault Tree Analysis of Software*, Masters Thesis, University of California, Irvine, 1982.

4. Leveson, N. G., Cha, S. S., and Shimeall, T. J., *Safety Verification of Ada Programs Using Software Fault Trees*, University of California, Irvine, February 1991.

5. Peterson, J. L., *Petri Net Theory and the Modeling of Systems*, Prentice Hall, 1981.

6. Leveson, N. G., and Stolzy, J. L., "Safety Analysis Using Petri Nets", *IEEE Transactions On Software Engineering*, Vol. SE-13, No. 3, March 1987.

7. Yellman, T. W., "Event Sequence Analysis", *Proceedings of the Annual Reliability and Maintainability Symposium*, January 1975.

8. Barbour, G., "Failure Modes and Effects Analysis by Matrix Method", *Proceedings of the Annual Reliability and Maintainability Symposium*, January 1977.

9. Herrin, S., "System Interface FMEA by Matrix Method", *Proceedings of the Annual Reliability and Maintainability Symposium*, January 1982.

10. Goddard, P. L., and Davis, R., *Automated FMEA Techniques*, Final Technical Report, RADC-TR-84-244, AD-154161, 1984.

11. DeMarco, T., *Structured Analysis and System Specification*, Yourdon Press, 1978.

12. Hatley, D. J. and Pirbhai, I. A., *Strategies for Real Time System Specification*, Dorset House Publishing, 1987.

BIOGRAPHY

Peter L. Goddard
Hughes Aircraft Company
P. O. Box 3310
Fullerton, Calif. , 92634, USA

Pete Goddard is currently employed as the head of the Systems Dependability Technology Section of the Systems Effectiveness Department of Hughes Aircraft, Ground Systems Group. He holds a bachelors degree in Mathematics from the University of LaVerne, and a masters degree in Computer Science from West Coast University. Mr. Goddard has previously published technical papers in the proceedings of the Annual International Logistics Symposium (SOLE), the RAMS Symposium, the AIAA Computers in Aerospace Symposium, and the NCOSE Symposium. He was the principle investigator for the 1984 Rome Labs sponsored "Automated FMEA Techniques" research study and was program manager, and part of the research team for the 1991 Rome Labs sponsored "Reliability Techniques For Combined Hardware And Software Systems" research study.