"WHAT DO YOU MEAN - IT'S FAIL-SAFE?"

EVALUATING FAIL-SAFETY IN PROCESSOR-BASED VITAL CONTROL SYSTEMS

PRESENTED AT THE

AMERICAN PUBLIC TRANSIT ASSOCIATION

1990 RAPID TRANSIT CONFERENCE

HYATT REGENCY HOTEL
VANCOUVER, B.C.
JUNE 2 - 7, 1990

By:

David B. Rutherford, Jr.
RAIL TRANSPORTATION SYSTEMS, INC.

"WHAT DO YOU MEAN - IT'S FAIL-SAFE?"

EVALUATING FAIL-SAFETY IN PROCESSOR-BASED VITAL CONTROL SYSTEMS

By:
David B. Rutherford, Jr.
RAIL TRANSPORTATION SYSTEMS, INC.

The advent of processor and microprocessor-based vital rail control systems, both on-board and wayside, has raised some very important and difficult safety issues. The fail-safety of the system is no longer obvious to the end user, as it was in systems constructed of relays. The safety is embodied within proprietary software, in combination with sophisticated digital and discrete hardware, and not necessarily available for inspection or evaluation.

It is becoming increasingly important for end users, both private and public, to require documentation and demonstrate some understanding of the fundamental principles upon which their systems are designed. A thorough, documented analysis by the manufacturer or independent party provides security and protection for the transit authority and their customers as well.

But what standards should be applied in evaluating the level of fail-safety claimed for a processor-based system?

In this paper, the question of standards is broadly investigated. Should there be an industry standard? Do de facto standards already exist in the industry? Should standards of analysis be the focus as opposed to standards of design principles?

The purpose of the paper is to raise the issue of verification of fail-safety in processor-based systems and stimulate thinking on the subject.

CAN PROCESSOR-BASED SYSTEMS BE MADE FAIL-SAFE?

The answer is a subjective **yes**. The comparison of safety levels achieved by a conventional safety relay system and that of a processor-based system performing the same function is ultimately a subjective matter, since rigorous quantification of the level of safety of either system is not possible.

In this context, however, it is generally recognized by the railroad industry that vital processor systems can be made as safe as, or safer than, their relay or discrete-circuit counterparts.

In addition, the benefits of increased functional capability and ease of application are bonuses, offsetting the disadvantages of increased complexity and difficulty in evaluation.

HOW ARE PROCESSOR SYSTEMS MADE FAIL-SAFE?

Fail-safety must be a design criteria inherent in every level of a processor-based system. A representative set of system levels follows:

A. Primordial Level

At the primordial level the logical rules applied to the function to be performed, be it wayside interlocking and signalling or carborne ATP, must assure safe operation.

An example of the primordial level in conventional interlocking systems is a relay interlocking logic design. Primordial safety is the assurance that the interlocking will provide safe operation if the conceptual logic is implemented as intended.

B. Conceptual Level

The conceptual level is taken for granted in traditional relay systems. The logical interconnection of safety relays, devices directly accepted as fail-safe by historical precedent, is known to be conceptually sound, and design standards are well recognized.

The concepts involved in making processor-based systems fail-safe, however, are less well understood by the end user, and design standards vary with each manufacturer.

There are a number of concepts currently in use. Here are a few of them, along with their distinguishing features:

1. Checked Redundancy:

   o identical hardware and software duplicated 2 or more times.

   o agreement on each decision required by 2 or more units for a permissive output

   o vital comparison hardware

   o if N units must agree, N+1 units allow a degree of fault-tolerance

2. N-version Programming:

   o N systems required

   o software independently generated and unique to each system

   o hardware may or may not be identical

   o agreement on each decision required by 2 or more units for a permissive output

   o possibility of N-version programming within one processor

3. Single processor - diverse operations:

   o each permissive decision (and output) requires concurrence of diverse processing within one hardware system

   o each permissive output is checked by performing diverse operations or via diverse channels

4. Single Processor - numerical assurance:

   o each permissive decision requires unique numerical representation, containing current verification of permissive conditions

   o diverse results may be required on alternating time cycles to avoid permanent storage of permissive results

C. Design Level

At the design level, the system's hardware and software components are designed to perform the required function utilizing the prescribed concept. Also at the design level processors are chosen, non-vital digital hardware and discrete vital hardware are designed, languages selected, software algorithms constructed and code written.

D. Implementation Level

The implementation level involves fabrication of the design into a physical system. Printed circuit board layout, construction and assembly, custom device design and fabrication, compilation and insertion of software into physical devices are all parts of the implementation process.

E. APPLICATION LEVEL (PRIMORDIAL)

Application of the system may again require decisions at the primordial level. Those systems which are programmable, which have the ability to conform to a wide variety of applications (wayside interlocking systems for example), require logical constructions defining operations at the primordial level.

F. APPLICATION LEVEL (INTERPRETIVE)

The system may be required to interpret parametrical information on how it is to be applied. It is essential at this level that the application specific information is correctly programmed into the system and is interpreted as intended.

ASSURANCE OF FAIL-SAFETY AT EACH LEVEL

The analysis of system safety as a whole requires analysis of each level. The analysis requirements and the procedures and techniques used to fulfill them may vary with each level.

A. PRIMORDIAL SAFETY

Primordial Safety is analyzed macroscopically for assurance that the logical constructions intended provide safe operation in the specific application.

Analysis of primordial safety continues to be performed predominantly by manual methods; for example, circuit checking relay designs for route locking and signal interlocking.

Application of the closed-circuit principle is another example of manual confirmation, as in the verification that a carborne system produces train lines which are energized to indicate permissive functions (such as withholding braking, etc.).

B. CONCEPTUAL SAFETY

Here the term 'Conceptual Safety' means the ability of the concept to provide the assurance of fail-safety, assuming the design and implementation conform in every way to the concept.

The digital circuitry in nearly all processor-based systems may be assumed to fail in any manner, and quite often in modes which are unpredictable, since the configuration of the integrated circuits themselves are not usually known to the designer. Those systems which try to anticipate all possible failure modes are thus running a high risk of working from an incomplete list. Multi-processor schemes have been developed to allow hardware failures in one unit to be revealed by comparison with others. However, all of these concepts have potential pitfalls. The end user should be aware not only of the pitfalls themselves, but also the methods used by systems' manufacturers to circumvent them.

The following are potential problems within each of the concepts considered. Systems utilizing these concepts would address these and other problems in analysis.

1. Checked Redundancy

   Two or more processors, executing identical software in identical hardware environments, compare results, requiring two or more to be identical before allowing a permissive output to be initiated and/or maintained.

   Possible problems:

   a. Software errors

      Although not failures (software itself does not fail), errors may exist after debugging which may result in an erroneous output state or the negation of a critical test.

   b. Depth of Result Comparison

      At what stage in the decision making process are the results compared? If only the final decisions are compared, intermediate results in each system may differ for a variety of reasons, but have identical outcomes. In this case one or more could eventually produce erroneous outcomes based on incorrect intermediate decisions.

   c. Check of Vital Output Circuits

      In the case where there is ultimately one output circuit for each permissive function, how is it checked? Is it checked by all N processors via independent interface circuitry? Via common interface circuitry? Is it checked by only one processor?

   d. The Vital Comparison Mechanism

      What are the possible failure modes of the vital comparison mechanism (M out of N results required to be identical)?

2. N-Version Programming

Two or more processors each execute unique, independently written software programs in identical or different hardware environments. As with checked redundant systems, each processor system's results are compared with all others and correspondence among M out of N results is assumed to prove it correct. Using this method, it is assumed that software errors will be self-revealing.

Possible Problems:

a. Software Independence

It is very difficult, if not impossible, to construct completely independent versions of software to meet the same set of specifications. This may in part be due to ambiguity in the specification and subsequent differences in interpretation. In order that results be comparable within acceptable time limits, some collusion among software teams may be required.

b. Depth of Result Comparison

As in Checked Redundant systems, decisions must be compared at some stage. Here it is even more difficult to compare intermediate decisions, since each version may perform the intermediate processes in a different order.

c. Numerical Accuracy

In non-Boolean decision making processes where mathematical computation is used, round-off errors among the N versions may produce identical decisions having different numerical values. Some of these may be outside the arbitrary decision boundary, making consensus among the N versions inaccurate or impossible in some circumstances.

d. Check of Vital Output Circuits & Vital Comparison Mechanism

These problems exist for N-Version Programming as well as for Checked Redundancy systems.

3. Single Processor Using Diverse Operations

In this single-processor concept, critical operations are performed in diverse ways, either via diverse software operations or using diverse hardware channels differing in some respect. The purpose of diversity is to make hardware failures self-revealing. Permissive outputs are allowed only if the results of the diverse operations correspond.

Possible Problems:

a. Failure Mode Anticipation

Diverse operations are performed to make failures self-revealing. This implies that all failure modes are known, or guessed, or anticipated in some way. As stated earlier, the complexity of the integrated circuitry comprising processor systems makes anticipating the complete set of failure modes a difficult task.

b. Decision Verification

The decisions made concerning the correspondence of diverse processes must be verified by another agent. It is risky to allow the processor making a decision to also judge that it has made it correctly. Assuring the accuracy and fail-safety of this agent is then another primary critical task.

c. Software Errors

Single processor systems dependent on diversity are susceptible to software errors as are Checked Redundancy systems above.

4. Single Processor Using Numerical Assurance

In this single processor concept, large unique numerical values are constructed by combining numerical values representing each of the critical constituents of a permissive decision. The states of the outputs are verified numerically to correspond with permissive resultant values.

Possible Problems:

a. Decision Verification

The correctness of the numerical verification of the permissive outputs must be assured by some external agent. As with other single processor systems, the analysis of this agent is a primary critical task.

b. Software Errors

Permissive numerical values or short cut computation procedures must not be allowed to exist in the software code.

C. Design Safety

Analysis of fail-safety at the design level begins with the identification of all critical functions required to be performed. After critical functions are identified and enumerated, a detailed analysis should show how each function is vitally implemented. This analysis should relate directly to the concept being employed.

Each system will most likely have portions of the hardware designated as 'vital', with the remainder being non-vital. The vital implementation of each critical function involves the interaction among the non-vital hardware, the executing software, and vital hardware.

The design analysis must show that the hardware-software interactions used to perform those algorithms required to implement critical functions assure safe operation in the event of hardware failure, or failure of software instructions to execute, do to transients or uncorrected software errors.

D. Implementation Safety

The realization of the design into physical hardware is yet another level which requires detailed analysis. Physical separation of components, printed circuit traces and wiring, threshold settings of critical devices, correct component assembly, etc., are among the areas where failure modes could be introduced or at least made more likely.

Analysis at the implementation level concentrates on the failure modes of vitally designated portions of the circuitry and the correct installation of the software version analyzed at the design level.

E. APPLICATION SAFETY

Processor-based systems tend to be flexible by nature. Ease of application to a variety of environments is rightly considered a big advantage over traditional systems. The ability to program parametrical information, however, introduces a need for analysis at the application level. Assurance is required that the engineer applying the device or system set the parameters correctly, and that the system correctly interpreted what the engineer intended.

ANALYTICAL TOOLS

There are a variety of tools available for use in analyzing the fail-safety of a device or system. Some of the most common are listed here.

A. FAULT HAZARD ANALYSIS

Performed on the system as a whole and each sub-system individually to identify, from a system perspective, all functional faults which could produce hazardous situations in which a dangerous event could occur.

B. FAULT TREE ANALYSIS

A method which enables all functions critical to a particular fault to be arranged in hierarchical order. It may be used to identify the set of critical functions associated with each fault.

C. FAILURE MODES AND EFFECTS ANALYSIS

A standard method used to analyze discrete component circuitry, in which the effects of all failure modes and their combinations are identified and analyzed.

D. SOFTWARE ANALYSIS

Most current software analysis methods emphasize verification and validation, as opposed to assurance of fail-safety. 'Soft Tree' techniques have been used with some success.

E. OPERATIONAL TESTING

Structured, comprehensive operational testing may be the ultimate test of primordial safety. This procedure should exercise the vital primordial logic itself, exclusive of any non-vital protection mechanism.

SUMMARY

Processor-based systems and devices can be designed and fabricated to assure fail-safety. The conceptual framework used as a basis for design may have inherent problems, however each of these problems has solutions, and those solutions may vary with manufacturer.

Analysis of the system is necessary to assure the fail-safety of its primordial logic, its conceptual framework, the design techniques used in realizing the concept, its as-built physical configuration and its application specific programming.

There are probably as many techniques for vital processor-based system analysis as there are manufacturers of such systems. The comprehensiveness and effectiveness of these analyses will also vary.

CONCLUSIONS

It is important for the end user of vital processor-based systems to be aware of the manner in which fail-safety is assured in his system. Indeed, it is important that fail-safety IS assured, preferably by a comprehensive, effective, well documented and structured analysis.

However the design of these systems, along with their conceptual framework and the intricacies of their analysis, is a complicated business.

Addressing the question of standards may allow for some well-needed insight into the evaluation of these vital systems. If concepts and design practices and techniques are not to be standardized, allowing for new and better systems to be developed, perhaps standards could be set for methods of analysis.

At the very least, the industry could be asking common questions and receiving some minimum of documented assurance concerning the fail-safety of the increasingly popular vital processor-based system.

"WHAT DO YOU MEAN - IT'S FAIL-SAFE?" PART II

EVOLVING STANDARDS FOR THE EVALUATION OF FAIL-SAFETY IN PROCESSOR-BASED VITAL CONTROL SYSTEMS.

PRESENTED AT THE

AMERICAN PUBLIC TRANSIT ASSOCIATION

1991 RAPID TRANSIT CONFERENCE

PHILADELPHIA, PA
JUNE 9 - 13, 1991

By:

David B. Rutherford, Jr.
RAIL TRANSPORTATION SYSTEMS, INC.

"WHAT DO YOU MEAN - IT'S FAIL-SAFE?" - PART II

EVOLVING STANDARDS FOR THE EVALUATION OF FAIL-SAFETY IN PROCESSOR-BASED VITAL CONTROL SYSTEMS

By:
David B. Rutherford, Jr.
RAIL TRANSPORTATION SYSTEMS, INC.

In Part I of this paper, presented at the 1990 Rapid Transit Conference in Vancouver, B.C., the question of industry standards for processor-based vital control systems was broadly discussed. After describing several concepts currently used for the implementation of vital processor-based designs, the paper raised the following issues:

- o What criteria should be used for evaluating the level of fail-safety claimed for a processor-based system?
- o Should there be industry-standard criteria?
- o Should standards of analysis be the focus, or standards of design principles?

Over the course of the past year, some standards for analysis and evaluation criteria appear to be evolving. This paper describes those standards.

ARE STANDARDS NECESSARY?

In the past five years, a number of public transportation authorities have commissioned independent third-party reviews of vital processor-based carborne and wayside equipment. SEPTA, WMATA and Metro-North are among those authorities. These reviews ranged from complete independent analyses of the system hardware and software to independent reviews of analyses provided by the equipment manufacturers.

The question which should be answered by such a review is, of course, "Is it safe?". At best, however, the reviewer was probably able to answer "Yes - in the opinion of the reviewer, it is safe.".

How safe is it? Presumably, safe enough. As safe or safer than 'conventional' equipment? Most probably, yes.

Quantitative answers to these questions are, by the very nature of the equipment involved, difficult to realize. The methods by which the quantitative answers are derived prove just as difficult to evaluate. Each reviewer evaluates the system under inspection using his own criteria.

AN APPROACH TO STANDARDIZING EVALUATION CRITERIA

The California Public Utilities Commission has led an aggressive effort to define system design requirements and evaluation and analysis criteria for vital processor-based systems for two large rapid transit projects; the fully-automated Metro Green Line in Los Angeles and the BART Extension Program in the Bay Area. Each of these projects has included such requirements in its specifications

The approach taken in these specifications stresses three elements: visibility, design and analysis requirements, and the calculation of a quantitative measure of system safety assurance. It does not constrain the suppliers' choice of safety assurance concept or design method. It does, however, impose requirements regardless of the design approach used.

VISIBILITY

This approach emphasizes making visible the factors upon which the assurance of fail-safety in a specific design depends. This is done in the context of the safety assurance concept employed by the system.

Within each broad system concept, that system's safety assurance relies upon some set of factors which must be demonstrated to be true. It is of primary importance to identify each of those factors, thereby making a complicated system's underlying safety foundations clear and visible.

Four concepts currently used in vital control systems were described in Part I of this paper, and are briefly catalogued here.

Checked Redundancy

Checked redundancy is a safety assurance concept in which two or more processors, executing identical software in identical hardware environments, compare results, requiring two or more to be identical before allowing a permissive action or condition to be initiated or maintained.

Safety assurance in the checked redundancy concept is dependent upon, among other factors, software correctness, the extent to which and the delay with which hardware failures are detected by comparison of two or more systems, the degree to which the hardware systems are independent, and the vitality of the comparison mechanism.

N-Version Programming

N-version programming is a safety assurance concept in which one or more processors execute multiple, unique, independently written, software programs in identical or different hardware environments. Each software system's permissive results are compared with one or more others' and their correspondence is assumed to provide safety assurance.

Safety assurance under the n-version programming concept is dependent upon, among other factors, the independence of the various system's software, the extent to which and the delay with which software errors are detected by comparison of two or more software systems, and the vitality of the comparison mechanism.

Diversity and Self Checking

Diversity and self-checking are safety assurance concepts in which critical operations are performed in diverse ways, using diverse software operations and/or diverse hardware channels, and where critical system hardware is tested with self-checking operations. Permissive outputs are allowed only if the results of the diverse logical operations correspond and the self-checks reveal no failures.

Safety assurance in systems using diversity and self-checking is dependent upon, among other factors, the completeness of the anticipated set of hardware failures that the diverse operations and self-checks are designed to reveal, and the effectiveness of the diverse operations and tests in revealing them. In addition, it is dependent upon software correctness and completeness, and the vitality of the diverse operations' comparison mechanism.

Numerical Assurance

Numerical assurance is a safety assurance concept in which permissive decisions are represented by large unique numerical values, constructed by combining numerical values representing each of the critical constituents of a permissive decision. The presence of restrictive system states and the allowance of permissive actions are verified numerically.

Safety assurance in systems using the numerical assurance concept is dependent upon, among other factors, the accuracy and uniqueness of the numerical data structures and the vitality of the mechanism that performs the numerical verification.

Emphasis on visibility requires that each of the factors upon which safety assurance is dependent be described clearly and completely. For example, two of the concepts described above depend upon software correctness, or 'error-free software'. It is important that a manufacturer whose system design depends upon software correctness make that fact visible. Once this design dependency is made visible, the manufacturer can then be required to provide an analysis demonstrating the extent to which the software in his system is error-free.

A list of some of the factors upon which safety assurance is dependent in various system concepts are:

Software Correctness

Dependence upon the software being error-free.

"Error-free software" is defined here as software being free of any error which could adversely affect the safe implementation of a vital function.

Software errors include errors in the high-level structural logic, errors in the logic of algorithms, and coding errors; in short, an error in any logical decision outside the set of those decisions contained in the primordial system logic.

Since absolute correctness is difficult if not impossible to demonstrate, quantitative measurement of the expected error rate is preferred, as discussed below.

Software Completeness

Dependence upon execution of software routines which are unrelated to the functional system requirements. Hardware self-checking routines are an example; in this case, errors of omission are significant, and software must be demonstrated to be complete.

Software Data Structure Accuracy and Completeness

Dependence on the correctness and/or completeness of software data structures and tabular application data.

Hardware Failure Characteristics

Dependence on the absence of hardware failures in defined modes in circumscribed portions of the system.

Dependence on the self-revealing characteristics of hardware circuits or components under failure.

Independent System Comparison Validity

Dependence upon the comparison of two independent processes to validate a permissive decision, condition, or action and the depth in the decision making process to which the comparison is made.

System Independence

Dependence upon the degree to which two or more hardware and/or software systems or processes are independent.

Failure Anticipation

Dependence upon the accuracy and completeness of the set of anticipated hardware failure modes and the effectiveness of the tests constructed to reveal them.

In addition to a clear description of the system safety assurance concept used and the identification of all factors upon which safety assurance is dependent, visibility includes a thorough description of the techniques used to satisfy those dependencies and an explanation of the manufacturer's hardware and software design and implementation standards.

SYSTEM DESIGN AND ANALYSIS REQUIREMENTS

Fail-safety, in the context of processor-based systems, is defined here as the property of the system that assures the safe implementation of all vital functions under all conditions, including the occurrence of hardware failures and the execution of software errors.

In order to evaluate the assurance of fail-safety in processor-based systems, the systems' hardware and software are classified into generalized categories. Broad design requirements are then imposed on each category. Hardware is designated as CLASS I, II, or III hardware, while software is designated as Vital or Non-Vital software.

HARDWARE CLASSIFICATIONS & REQUIREMENTS

CLASS I Hardware (Vital Hardware)

CLASS I hardware is traditional 'Vital Hardware', whose failure modes and characteristics can be accurately and comprehensively identified, predicted and exhaustively tested. In CLASS I hardware, the occurrence of failure modes that could have unsafe consequences are eliminated, prevented or otherwise accounted for by design; they are not accounted for statistically.

Examples of CLASS I hardware circuits and components:

- Vital discrete rate-decoding filter
- Vital current threshold detection
- Vital signalling relay
- 4-terminal capacitor

The assurance of safe implementation of vital functions by CLASS I hardware is verified by traditional Failure Modes and Effects Analysis (FMEA) methods, conforming to the following requirements:.

CLASS I Hardware FMEA Requirements:

The FMEA shall show that no single failure mode produces an unsafe condition.

The FMEA shall classify all failure modes as self-revealing or non self-revealing.

The FMEA shall show that all combinations of failures produce no unsafe condition, other than combinations of independent, self-revealing failures.

Using the FMEA technique to evaluate the safety assurance of CLASS I hardware results in a Pass/Fail evaluation. Presumably if the result is 'Fail', the circuit can be redesigned to pass, perhaps by using selected components with recognized failure mode characteristics. However, there is no quantitative measure of safety assurance for CLASS I hardware. The circuit either meets the FMEA acceptability criteria or it does not.

CLASS II Hardware (Non-Vital Hardware used to Implement Vital Functions)

CLASS II hardware is hardware, the failure of which, may have an adverse effect on the implementation of vital functions, but whose failure modes are not directly analyzable. CLASS II hardware includes the processor-based system CPU, memory, addressing logic, and port selection circuitry; in short, any integrated digital circuitry used within the processor-based system performing vital functions.

The failure modes within and their effects on this complex circuitry may be not only non-analyzable, but quite possibly innumerable. CLASS II hardware failure modes are revealed or otherwise accounted for by means other than vital hardware design techniques. These means include software-driven self-checking, comparison of independent hardware circuits, and numerical techniques.

CLASS II Hardware Design Requirements

Requirements for CLASS II hardware are rigorous. It must be designed so that for each and every possible failure or combination of failures, it can be demonstrated that either:

- o the failure(s) has no adverse effect on the safe implementation of a vital function, or
- o the failure(s) is vitally detected and subsequent action vitally assures that no unsafe effect is produced, or
- o an upper bound on the probability that the failure(s) has an unsafe effect (PFU) is calculable, as well as a lower bound on the mean time between unsafe failures (MTBUF).

CLASS II Hardware Analysis Requirements

In addition to other requirements, certain concept-specific requirements must be demonstrated by analysis.

Cycle-Checking Designs - in those systems in which failure modes have been identified and the effects of their occurrence anticipated, analysis shall demonstrate:

All failures that could aversely effect the safe implementation of each vital function have been anticipated.

The mechanism or test designed to detect the occurrence of each failure is effective in revealing the failure.

The reaction of the system, once the failure is revealed, shall be to assume and maintain a safe state or states.

The mechanism designed to reveal the failure and the subsequent safe system reaction to the occurrence of the failure cannot itself be compromised by the original failure, any subsequent failure or combinations of failures, or by the execution of errors in the software.

Checked-Redundant Designs - in those systems where hardware failures are revealed by comparison of independent hardware circuits, analysis shall demonstrate:

All failures that could adversely effect the safe implementation of each vital function are revealed by detecting the differences in the points of comparison between the systems.

The comparison mechanism has no unrevealed modes of failure that could compromise its ability to detect and react to differences between the systems.

The reaction of the system, once the failure is revealed, shall be to assume and maintain a safe state or states.

The mechanism designed to reveal the failure and the subsequent safe system reaction to the occurrence of the failure cannot itself be compromised by the original failure, any subsequent failure or combinations of failures, or by the execution of errors in the software.

Numerical Assurance Designs - analysis of systems in which safety is numerically assured by the guarantee of an upper bound on the PFU shall demonstrate:

The method used to calculate the PFU is valid and the calculations are accurate.

All ancillary functions required to be performed for the PFU to be considered valid are performed correctly.

All factors upon which the validity of the PFU depends are complete and correct.

The mechanism and techniques used to verify numerical results are such that failures cannot reduce the value of the PFU of the processor-based system in which it is used.

Implications of CLASS II Hardware Requirements

The implications of these requirements are significant. If it is indeed the case that the failure modes in CLASS II hardware are non-analyzable, and further that it is not practically possible to demonstrate that each and every failure has been anticipated and tested or declared a safe failure, then this approach requires that a system using CLASS II hardware be designed so that a quantitative evaluation be made of the unsafe effects of failures.

The parameters chosen to measure the extent of the adverse effects of CLASS II hardware failures on the fail-safety of the system are:

PFU - given a hardware failure, PFU is defined as the probability of that hardware failure having an unsafe effect, and

MTBUF - defined as the mean time between those hardware failures which have an unsafe effect.

CLASS III Hardware (Hardware used to implement ONLY non-vital functions).

CLASS III hardware is defined as hardware whose operation under normal or failure conditions can be demonstrated to have no effect on the safe implementation of any vital function.

SOFTWARE CLASSIFICATION AND REQUIREMENTS

The software contained in processor-based vital systems is easily the least understood system component and the most difficult to analyze.

The classification of software into Vital and Non-Vital software is usually an obvious one, however it is still necessary to explicitly identify all vital software.

Vital Software

Vital software is defined as that software required for the implementation of a vital function. In addition, vital software is any software whose execution could affect the implementation of a vital function.

Non-Vital software is then defined as software whose normal operation has no effect on the implementation of any vital function.

There is a caveat, however. Since the execution of errors in non-vital software operating in the same processor environment as vital software could affect the implementation of a vital function, non-vital software executing in the same processor as vital software must meet the same requirements as vital software with regard to error execution and detection.

What comprises a software error is discussed above under 'Visibility'. It will be noted here, however, that in this context it is assumed that software does not and cannot fail. Software may contain errors in the logical organization of its architecture, structure and algorithms, in the content of its data base, and in its coding and compilation, and required logical functions may be omitted. But software does not fail in the sense that it worked correctly at some point but is no longer correct due to a change. A failure in the processor, its peripherals, the memory devices in which the software is stored, or any physical component which impairs correct execution of the software, is considered a hardware failure.

Vital Software Design and Analysis Requirements

The requirements for vital software include that it be designed so that it can be demonstrated that either:

- o the software is error-free, or
- o the upper bound on the probability that the execution of a software error will have an unsafe effect (PEU) is calculable, as well as a lower bound on the mean time between the execution of such errors (MTBUE).

PEU and MTBUE are parameters chosen to measure the adverse effect of the software error execution on the implementation of vital functions.

Vital Software Analysis Requirements

No predefined analysis method is specified for the analysis of vital software. The analysis requirements are simply that it demonstrate that the design requirements have been met, i.e., that the logic, structure, algorithms, data base, coding, and subsequent translation into machine language either:

- o contain no errors that, when executed, can adversely effect the safe implementation of any vital function, or
- o have a calculable upper bound on the PEU and a calculable lower bound on the MTBUE.

Further, the calculation of PEU and MTBUE shall be shown, if applicable.

Implications of Vital Software Requirements

As with the requirements for CLASS II hardware, the implications of these requirements on Vital software are significant.

It is generally conceded by the software reliability industry that demonstrating software correctness to the extent that a non-trivial software program can be declared error-free is a practical impossibility. This implies that vital software must be designed so that quantitative measurement of both the likelihood that a given error would be unsafe, and the average time between execution of such errors is possible.

The extent to which these quantitative measurements can be accurately made is a function of the conceptual design and the modeling tools available to measure or predict error-rates in software.

Some designs lend themselves to the calculation of an upper bound on the probability that any error could have an unsafe effect. Others require the assumption be made that all errors would be unsafe and rely upon demonstrating that the mean time between such errors is long enough to be acceptable.

QUANTITATIVE MEASURE OF THE ASSURANCE OF FAIL-SAFETY

The design and analysis requirements for processor-based vital control systems outlined above include quantitative measurements for CLASS II hardware and Vital software system components.

These measurements are meant to predict the level of the assurance of fail-safety of those system components. Of the remaining system components, CLASS I, or Vital hardware is assumed to be qualitatively analyzable by traditional FMEA methods to an acceptable level. CLASS III hardware and non-vital software components are assumed to have no effect on the safe implementation of vital functions.

Thus MTBUF and MTBUE represent the quantitative measure of the assurance of fail-safety of a system. These two parameters are combined and included in an overall system measurement parameter, MTBHE, or Mean Time Between Hazardous Events. The MTBHE may include other quantitative factors, but it is assumed here that MTBUF and MTBUE are the dominant constituents.

A hazardous event is defined as the occurrence of an unsafe condition that exposes passengers, personnel or equipment to injury or damage. MTBHE is defined as the mean time between such events.

Once a parameter such as MTBHE is established as a measure for the assurance of fail-safety, the next question is what is its minimum acceptable value?

The minimum values for MTBHE specified for the California transit projects referenced above appeared in the system design requirements as:

Each individual ATP system shall have an MTBHE of greater than 10E7 years, and the MTBHE of the whole of the ATC system shall be greater than 10E5 years.

(The phrase 'individual ATP system' used here means an individual interlocking controller, vehicle ATP system, or the like.)

The criteria used to select these values will not be discussed here, however they are comparable with the accepted risk figures of other transportation systems, including scheduled passenger airline service.

It is interesting, however, to discuss the implications of MTBHE. A system-wide MTBHE value of 100,000 years (10E5 years) appears to be a rather large number, but it equates to the probability of 1 in 10,000 for the occurrence of a hazardous event in the first 10 years (assuming a constant failure/error-execution rate).

On the other hand, the somewhat lower value for the system-wide MTBHE of 1000 years (10E3 years) equates to the probability of 1 in 100 for the occurrence of a hazardous event in the first 10 years. While the acceptance of required MTBHE values is a subjective decision, the probability of a hazardous event in the first 10 years of 1 in 100 appears quite high.

SUMMARY

An approach to standardizing the evaluation of the assurance of fail-safety in processor-based vital control systems is being introduced in the specifications of two large transit projects in California with the encouragement of the California Public Utilities Commission.

The approach requires:

o The safety assurance design concept and the factors upon which the safety assurance of the system depends be made clearly visible.

o The classification of hardware into:

CLASS I	-	Vital hardware.
CLASS II	-	Non-vital hardware used to implement vital functions.
CLASS III	-	hardware not used in the implementation of vital functions.

o The classification of software into vital and non-vital software.

o Evaluation of the effects of failure in CLASS I hardware by traditional, non-quantitative FMEA techniques.

o Evaluation of the effects of failure in CLASS II hardware by techniques which result in either:

    the identification of all failure modes and subsequent analysis that demonstrates the related effects are not unsafe, or

    the calculation of quantitative values for the probability that any given hardware failure will have an unsafe effect (PFU) and for the mean time between such failures (MTBUF).

o Evaluation of the existence and the effects of error execution in Vital software by techniques which result in either:

a comprehensive and accurate demonstration that the software is error-free, or

the calculation of quantitative values for the probability that the execution of a given software error will have an unsafe result (PEU) and for the mean time between the execution of such errors (MTBUE).

- o Measurement of the assurance of fail-safety of the system as a whole by the quantitative parameter MTBHE (mean time between hazardous events).

- o Selection of an acceptable value for MTBHE as 10E7 years for each individual ATP system , and 10E5 for the ATC system as a whole.

CONCLUSION

It has yet to be seen whether existing processor-based systems or systems currently under development can meet these requirements rigorously. It may well prove to be true that the comprehensive analysis of individual failure modes in CLASS II hardware and the demonstration of error-free Vital software are practical impossibilities.

If it is the case that such non-quantitative methods are ultimately not acceptable in evaluating the level of safety assurance in a processor-based design, then surely it should be the goal of system suppliers to design systems which allow definitive calculation of safety levels.

The evolution of standards for the evaluation of safety assurance in processor-based vital control systems should compliment an industry-wide goal for the development of vital systems which are designed to enable quantitative verification of their level of safety.