FAILURE MODES IN EMBEDDED SYSTEMS AND ITS PREVENTION

Ms. Samitha Khaiyum, MCA, M Phil,(PhD)
Senior Lecturer and Research scholar, MCA (VTU),
DSCE, Bangalore-78

Dr. Y S Kumaraswamy, M Sc, PhD, PDF (IISC)
Sr Prof and Head, MCA (VTU),
DSCE, Bangalore-78

ABSTRACT: *Systems failures do not occur in a vacuum; while a single event may trigger the failure, investigation often reveals that a history of managerial and technical decisions produce conditions turning a single event into a disaster. At the minimum, investigating case studies provides lessons on what to avoid. By systematic studies of failure, it may be possible to draw general conclusions and improve practice as a whole.*

Unfortunately, good systems failure studies are rare. Embedded systems failure is a volatile topic and the field is filled with a vast amount of noise, urban myth, and political agendas.

INTRODUCTION: Systems fail for various reasons, ranging from inevitable hardware breakdowns to avoidable loss of morale. However, the accidents that result in system failures are not unique and have often been encountered before.

Technological failure modes in embedded systems can be divided into two main groups: hardware failure modes and software failure modes; the toughest failures to prevent however are those caused by subtle interactions between hardware and software.

Some examples of software failure modes are:
*Buffer overflow: the computer memory is smaller than the programmer expected, so during operation of the embedded system, one of the programs in the system is accessing wrong parts of the computer's memory.
*Dangling pointers: this error is common in non-safe programming languages in which the human programmer is responsible for making sure that every pointer points to the right memory location at all times.
*Resource leaks in which programming errors lead to the loss of computer control over some of the hardware resources; memory leaks are the simplest form of resource leak.
*Race conditions in which specific relative timing events of different components of the system leads to unexpected behavior. Such race conditions are often hard to detect by testing only.
*Semantic design, for example: the meaning of an arrow between two subsystems in a visual software environment should be the same as the interpretation of it by the hardware.

Some examples of hardware failure modes:
*Electrical failure: short-circuiting, too high voltage/current
*Mechanical failure: jamming of a valve
*Temperature effects: deformation of components
*Material failure: corrosion

Some examples of software failure causes are:
*Deadlock: two or more processes are each waiting for the other to finish, so none of the processes ever finish.
*Resource starvation: a process doesn't get the resources it needs, so it can never finish.
*Too small memory

*Noise
*Shared interfaces with other systems

Some examples of hardware failure causes:
*Hostile environments: any factor which prevents a system from functioning correctly.
*Badly calibrated sensors
*Choosing the wrong dimensions
*Manufacturing/assembly process deficiencies

Some failures are not caused by hardware or software, but are caused on the system level.

An example of a system failure cause: Operational failure: human operators make mistakes too.

At the minimum systems failures should be studied so as to prevent the same failure from occurring twice. However, in addition to uncovering specific engineering failures, it is possible to draw broader conclusions about engineering practice by studying failures. Systems failures happen for a reason, and these reasons do not have to be unique to either the situation or discipline.

Overview:

Due to the increasing capabilities and functionality of embedded systems, it is difficult to prevent or sometimes even detect failure modes. One way to ensure reliability is extensive testing using techniques such as probabilistic reliability modeling. One of the problems with these techniques is that they are only used in the late stage of development. It is better to design quality and reliability in, in the early stages of development.
To detect failures in the design process it is important to perform different tests on the system (especially on the software) at the beginning of the design. But tests are often expensive and they also should provide the correct information: the usability of test results depend on the quality of the test. So it is not always easy to come up with an appropriate test.

Dynamic analysis in the software world is the testing and evaluation of software by executing programs on a processor. An example of a dynamic analysis on hardware could be vibration and stress analysis.

These days engineers have developed a static analysis for software, which is test-free: no specific tests need to be developed and the software can be checked for flaws without having to execute the program.

There are a number of possibilities to reduce the chance of failure occurrences. But some failures need to be treated more urgent than others. At first one should look at the frequency with which a systems fails, this is called the failure rate of a system. It is desired that systems don’t fail, but if a failure is very rare it is often not necessary to take steps.

Another aspect of a failure mode is its severity. An electrical appliance that short-circuits can be life threatening, whereas the jamming of a valve in vending machine is less life threatening.

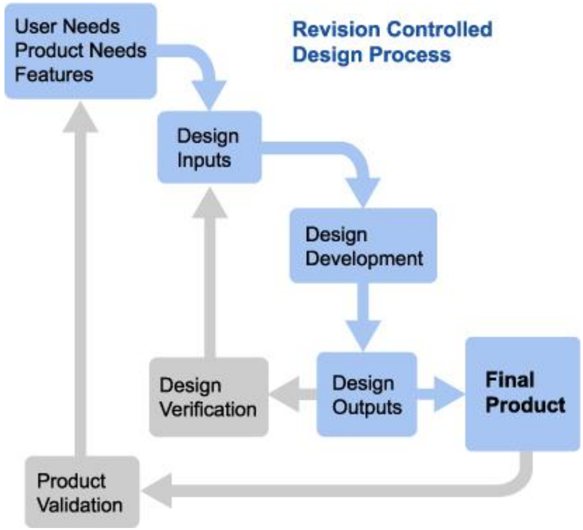
Despite all the effort an engineer can put into designing a system that doesn’t fail, failures will always occur. For example: an average cell phone these days contains as much as 2 million lines of software code. It is very likely that in one of those lines a fault is introduced. Systems are getting even more complex. For instance: that same cell phone is expected to have as much as 10 million lines of code in 10 years. Therefore a design should be more robust. When the system detects something goes wrong it can signal this and go into a safe mode until the user takes appropriate actions. Take for example again the jamming of a valve of the vending machine: the machine can light all its leds to signal something is wrong and cease providing soda until it is repaired.

Failures are also to be expected when separate systems have to work together,.So they have to cooperate in order to play the entire theme correctly.

The cost of designing embedded control systems tends to increase exponentially with increasing reliability: removing X% of the faults in a system will not necessarily improve the reliability by X% (a study at IBM proved that with removing 60% of the errors, only 3% reliability was added). In some cases, it may therefore be more cost effective to not investigate in more trustworthy systems but to pay the failure costs. However one has to keep in mind that this strategy can lead to a bad name of the company for selling systems that cannot be trusted. There is always a trade-off between different design criteria, depending on the system. Off course, critical systems should always be designed as reliable as possible.

This all stresses out how important it is to rule out failures in the design process. Fortunately, engineers have developed some procedures to do this systematically. All the following procedures can be used in what is called safety engineering. The study of failures is an important aspect of designing embedded control systems as it can safe time, money and even lives, and helps with eventual future modification of a system.

Design Process and Control



The process starts by generating the User/Product's set of needs that are documented as the features of the system. This is typically described by documenting the intended use of the device to be designed.

The Design Inputs or Requirements phase is the next critical point. The basic form fit and function are determined and then broken down to a list of requirements. These can then be partitioned into:

- Electronic Requirements
- Software Requirements
- Mechanical Requirements
- Environmental Requirements.

Documenting the Design Input in a Requirements Specification and is adept at converting and documenting informal and even verbal requirements is required. The design input, whether formal or informal, results in a list of "SHALLs". These can be:

- Directly Stated
- Derived by the Design Engineer from Stated Requirements

- Referenced by a Stated Requirement to another document
- Referenced to an Agency specification

It is the design engineer's responsibility to generate a design which complies to this list of SHALLs.

The design process produces many documents, which have various names and means of subdivision and organization, but most are similar to the Design Specification and Test Plan.

These design inputs are used for the product Design and the generation of the Design Specification. The Design Specification converts the raw requirements to detailed and parametric specifications from which a design can be generated.

Also generated is the test plan which provides test procedures and pass/fail criterion linked through the design specification to the requirements document. This linkage is usually called *Requirements Traceability* and the tool often used to demonstrate this linkage, is a *Requirements Verification Matrix.*

These outputs combined with schematics, software code, PCB databases, Mechanical models, etc. are the resultant of the design activity and are the Design outputs. All Design outputs combined together results in the Final product.

Design Reviews are a required part of all Benchmark Design Projects and the review board consists of the design team, independent reviewers and usually the customer. These give the design team benefit of the review board‘s aggregate experience and ensure that the required level of rigor and design discipline has been used in the execution of the design process.

Design Verification / Validation

The Verification and Validation Process is a methodology used to prove conformance to product features and design inputs/requirements.

Design Verification confirms that the results of the design process meet the design requirements. Design Validation ensures that the resultant device or product performs the intended function.

The Test Plan provides the various methods to prove conformance to the requirements, which include:

- Inspection/Audit
- Analysis
- Review
- Test

Inspections or audits can be performed by the design team, an independent team or the customer. These run the gamut from ensuring compliance to process through determining “as designed / as built” criterion.

Analysis is usually performed by the designer and offered in support of a statement of compliance or in defense of the design during a review.

Test is the most visible method used in the verification process. A series of electrical, mechanical and environmental stimulus is applied to the design or its subsystems and the responses are measured and judged for compliance to the specifications and requirements.

References:

1. Embedded Systems Case Studies, Carnegie Mellon University, Spring 1999, Michael Collins

2.Embedded Control Systems Design/Failure modes and prevention

3.www.ece.cmu.edu/.../koopman07_dependability_everyday_**embedded**_abs.pdf