Section 7 Introduction to Designing for Safety

7.1 Introduction

Safety must be the top priority in an Engineer's mind. Making a mechatronic system safe is particularly challenging since it incorporates mechanical, electrical and software components (plus hydraulic and/or pneumatic components in some cases). In this chapter issues related to safety will be discussed and some methods for producing safer systems will be introduced.

7.2 Risk Assessment and Risk Reduction

Risk assessment involves identifying all of the risks faced by anyone who may be harmed by the mechatronic system. This typically means operators (who might be customers) and maintenance people. The process of risk assessment is broken down into several stages:

i) Determination of the limits of the mechatronics system (*e.g.* maximum range of motion, maximum speed, maximum acceleration, maximum voltage, maximum pressure, maximum temperature, etc.).

ii) An analysis to identify potentially hazardous phenomena.

iii) Evaluation of the degree of risk

If no prototypes are available then stage (i) must be conducted using the design values. However, this process must be repeated again with the prototype(s) and with the final product.

The process of risk assessment is always followed by the process of risk reduction. Risk reduction involves changing the design (and the resulting product) until all risks are reduced to an acceptable level. Some methods for risk reduction are discussed in the remainder of this chapter.

7.3 System Faults and Fault Management

The mechatronic system must satisfy its design specifications. These typically include functional requirements (*e.g.* accuracy) and non-functional requirements such as cost and power consumption. Taking priority over this are the safety requirements. The safety requirements should address the problem of risk reduction and reduce all risks to an acceptable level. The ability of a system to meet both its functional and safety requirements is limited by the presence of faults within the system, where "fault" refers to any kind of defect. Faults can be classified as:

i) Random hardware component failures.

ii) Systematic faults due to the design of the system.

iii) Errors in the design specifications.

Random faults may be analyzed statistically and given sufficient data the probability of a hardware component failing over a given period of time may be estimated. Systematic faults are not random and cannot be analyzed statistically. Their effect of the reliability of the system is therefore much more difficult to predict.

The goal of "fault management" techniques is to address these faults in order to produce a safe and dependable system. Fault management may be grouped into:

i) Fault avoidance.
ii) Fault removal.
iii) Fault detection.
iv) Fault tolerance.

Fault avoidance techniques aim to prevent faults due to the design of the system or errors in the design specifications. The use of "factors of safety" when designing the system is one such approach. This should be familiar in relation to the design of electrical and mechanical hardware. When designing a control system the gain and phase margins are also forms of safety factors. Proper signage, instruction manuals and training procedures can also be considered as techniques for avoiding human faults. Fault removal techniques are aimed at finding and correcting faults before the system enters service. This typically involves extensive testing. Fault detection techniques are used to detect faults during service, allowing appropriate actions to be taken. The objective of fault tolerance techniques is to allow the system to operate correctly in the presence of faults.

Fault management techniques that should be implemented at the design stage fall under the categories of:

a) **System architecture**: This is the main approach to fault tolerance. The proper system architecture provides protection against random component failure and some types of systematic faults.

b) **Reliability engineering**: This involves techniques for predicting the probability of the system failing to meet its functional and safety requirements due to random component failure, and altering the design until it meets the required reliability.

c) **Quality management**: Maintaining high quality standards throughout the design, manufacturing and testing processes is crucial to producing a dependable system.

7.4 Some Fault Detection and Fault Tolerance Techniques

The goal of fault tolerance is to design the system such that the presence of faults does not result in system failure. Regarding the design of the pushbutton switches to start and stop a machine it is common practice to use a normally open switch for starting the machine and a normally closed switch for stopping the machine. This will prevent the machine from starting unintentionally when one of the wires connected to the start switch is broken and will stop the machine (or prevent it from starting) if one of the wires connected to the stop switch is broken. However, this will not protect against hardware component failures, for example if the contacts of the stop switch become stuck in the closed position. Another common design procedure is to attach spring-loaded brakes to all motion actuators. The brakes are released by an electromagnet. If power to the actuators is shut off (either intentionally or by accident) the electromagnets will also be shut off, and the brakes will be engaged by the springs. This should then prevent uncontrolled motion from taking place. Of course, if any of the brakes fail then this safety approach will fail.

All more sophisticated methods for fault tolerance are based on some form of hardware and/or software redundancy. An example of hardware redundancy is the triple modular redundancy (TMR) arrangement shown in Figure 7.1

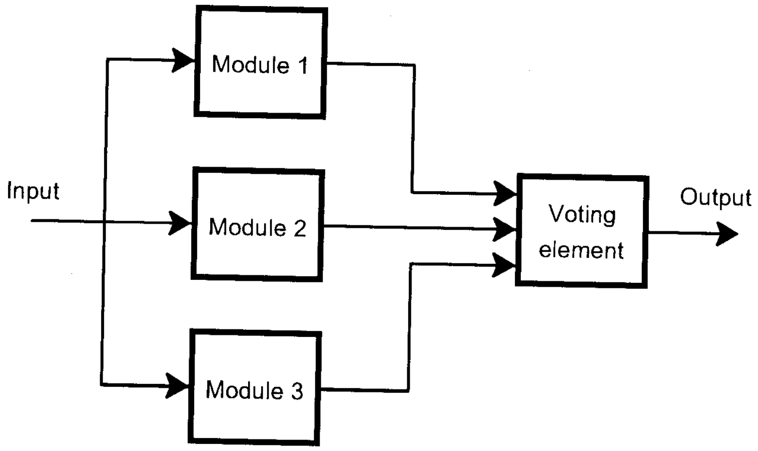

Figure 7.1 A triple modular redundancy arrangement (from Storey, 1999).

With TMR, three modules and a voting element are used in place of a single module. The voting element reads the outputs from the three modules and outputs the majority view. In this way the TMR will tolerate the failure of a single module and will produce the correct output. Clearly, the voting element must be simple and reliable otherwise the TMR arrangement is useless. The TMR is also subject to one of the problems associated with all forms of redundancy. It is based on the assumption that faults will occur with the individual modules independently. If the three modules are identical this arrangement provides some protection against random hardware component failures but does not protect against any systematic faults. A systematic fault (such as a software bug) will affect each module in the same way. So the module outputs may be both identical and incorrect and the redundancy achieves nothing. Failures that result from similar faults in each module are termed “common-mode failures”. Common-mode failures may be reduced using the “design diversity” approach. With design diversity the redundant modules have the same functionality but are designed independently by different teams using different hardware and software. This reduces the likelihood of common-mode failures but does not improve the reliability if there are errors in the design specifications.

Many fault tolerance techniques rely on using some form of fault detection. Several fault detection methods exist. For example if a sensor for measuring a room’s air temperature outputs a value outside the expected range then the sensor probably has a fault (*e.g* if it says the air temp.

is –100 °C). This is an example of "range checking". Similarly, sudden changes in the outputs of a sensor normally indicate a sensor fault.

The use of "watchdog timers" is another common fault detection technique. A watchdog timer is a hardware timer that will turn on an output if it is not reset before its timing period finishes. This may be used to detect both hardware and software faults. For example, let's say a pneumatic cylinder is equipped with a proximity sensor to detect when it is fully extended and we know it should take less than 1 second to fully extend. We should create a watchdog timer with a period of 1 second. Under normal functioning, when the valve is turned on the proximity sensor will be activated by the cylinder in under 1 second and the timer will be reset properly. If we turn on the valve and the watchdog timer does not get reset then we have detected that either the valve, cylinder or the proximity sensor has failed. In another example, if a software program is running properly it will reset a watchdog timer every 0.1 seconds as part of its normal operation. If the program "hangs" it will not be able to reset the timer and the timer output will turn on after slightly more than 0.1 seconds. This signal may then be used to reconfigure the system to recover its proper function. This approach may be further improved if additional hardware is used to ensure that the timer resetting is only occurring periodically (*i.e.* check that the reset signal is not stuck in the ON state). Note that proper functioning may be lost during the time required for fault detection and reconfiguration. This is known as "dynamic redundancy". (The TMR is an example of "static redundancy"). A common approach to dynamic redundancy is the "standby space" arrangement shown in Figure 7.2. If the fault detector indicates that one of the modules is faulty then the switch will select the other module and output its signal. Of course design diversity must be employed to reduce the probability of common-mode failures as before. Dynamic redundancy requires less hardware than static redundancy. For example, the standby spare arrangement uses two modules instead of the three required with TMR and both will tolerate the failure of one module. If three modules are used in a dynamic arrangement then the failure of two of them may be tolerated. The disadvantages of dynamic redundancy are its reliance on reliable fault detection, and the disruption that occurs during detection and reconfiguration.

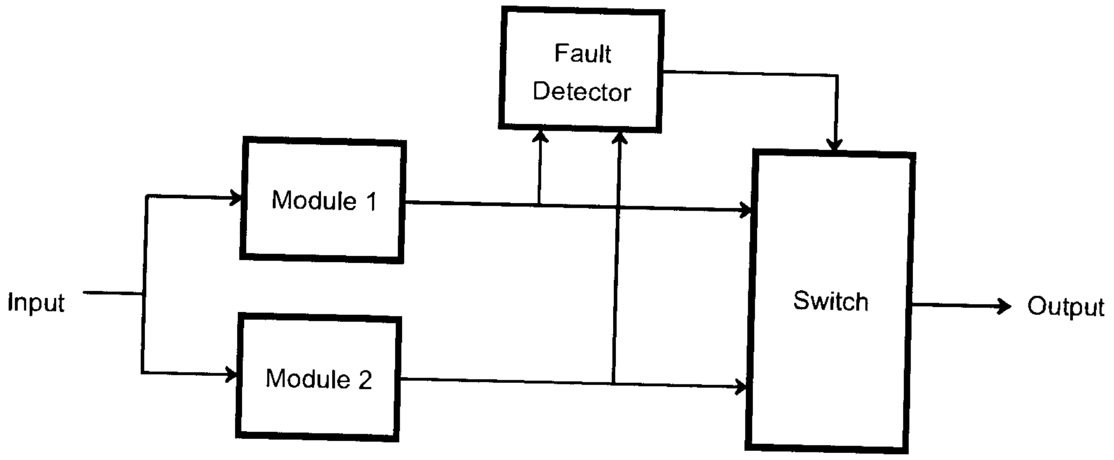

Figure 7.2 A standby spare arrangement (from Storey, 1999).

A "self-checking pair" is another fault detection technique that may be used as part of a dynamic redundancy arrangement. This is illustrated in Figure 7.3. The outputs of the two modules are compared using either hardware or software. If they disagree then a failure has been detected and the module output signal should not be used. Note that in some instances it may be advantageous to use a software simulation of module 1 (based on its dynamic model) for module 2. Module 2 is then known as a "digital twin."

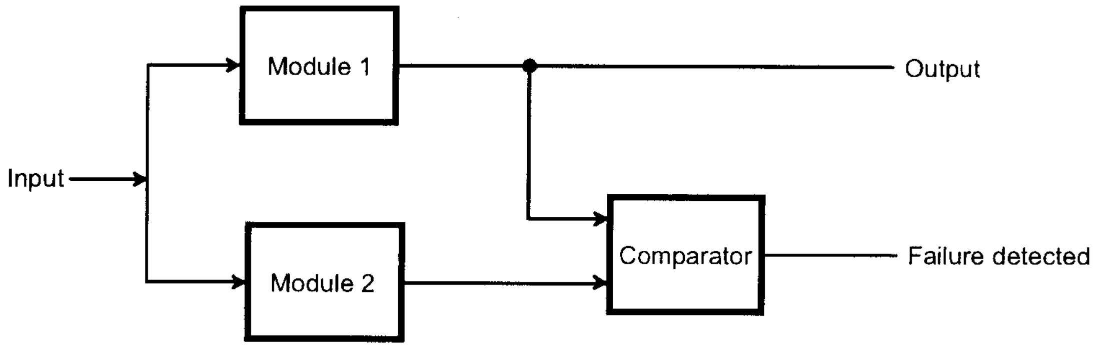

Figure 7.3 A self-checking pair (from Storey, 1999).

7.5 Other System Architecture Issues

It is well known that partitioning a system (both hardware and software) into smaller parts or modules is a desirable design approach. One of the advantages of partitioning is that it aids the comprehension of the system. Large monolithic structures of hardware or software are hard to understand and are therefore more prone to errors. A second advantage is the partitioning provides some isolation between the modules such that the effects of faults are localized as much as possible. A third advantage is the reliability of the modules is easier to test and verify. These modules form the layers of a hierarchy with high level functions at the top and input/output interfacing functions at the bottom. Even with this partitioned approach care must be taken to prevent an unsafe design. For example the structure shown in Figure 7.4 is undesirable. The high level checks the state of safety switch that has been transmitted up to it from the lower levels, makes a decision to turn the actuator either on or off, and then transmits that decision back down to lowest level. The lowest level then controls the actuator. If any of the modules at the different levels operates incorrectly then the actuator may operate in a dangerous fashion. A better approach to partitioning is shown in Figure 7.5. Now the high level sends a command "operate the actuator if it is safe to do so" to the lowest level. A single module in the lowest level checks the state of the safety switch and then controls the actuator appropriately. Clearly this will result in safer actuator operation. Please note that this approach may not be practical for all systems.

7.6 Other Reliability Issues

The more a system can be tested the greater our confidence in its reliability. This has lead to the idea of systems capable of "self-testing". This self-testing is done while the system is in service. For example, an accelerometer used in an air bag controller to detect a crash incorporates a microprocessor and a tiny actuator that may be used to push the sensing element. The microprocessor completes the self-test by verifying that the output of the pushed sensing element is correct.

Another design issue relates to the newness of the hardware and the software development tools. Safety critical elements of a system should use components that have an extensive record of reliable operation rather than new "start of the art" components whose reliability is unproven.

7.7 Human Factors

Mechatronic systems typically have a human operator and are maintained by humans. Human errors are particularly challenging for the designer since they are occur much more frequently than hardware or software faults and they can be hard to anticipate. Proper signage, manuals, and training procedures will help to reduce operator errors. Safety systems such as machine guarding and anti-tie down circuits are also helpful. To reduce the possibility of faults caused by incorrect maintenance the system should be designed for ease of maintenance (also known as "design for maintainability").

7.8 Safety Standards

All mechatronic systems must comply with the applicable safety standards and government legislation. Many organizations produce safety standards. These include: CSA, ISO, EN, IEC, RIA, etc.

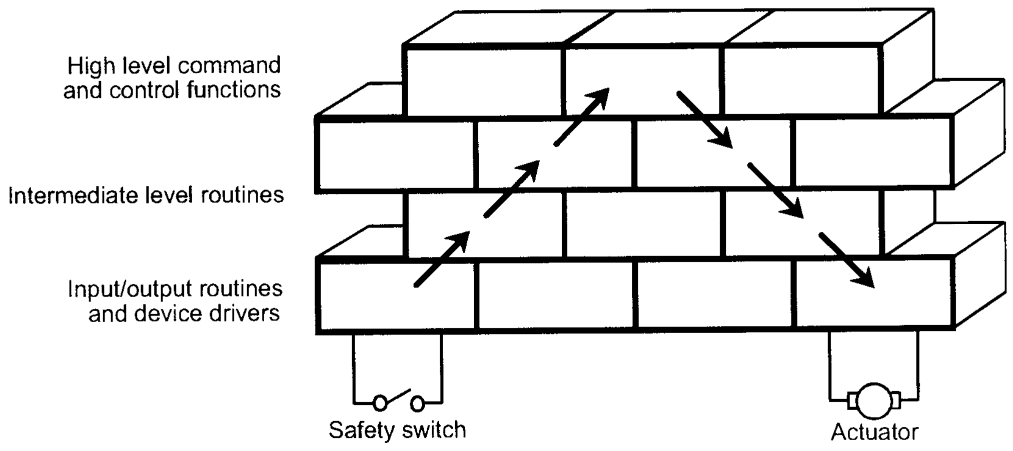
High level command and control functions
Intermediate level routines
Input/output routines and device drivers
Safety switch
Actuator

Figure 7.4 A poorly structured system (from Storey, 1999).

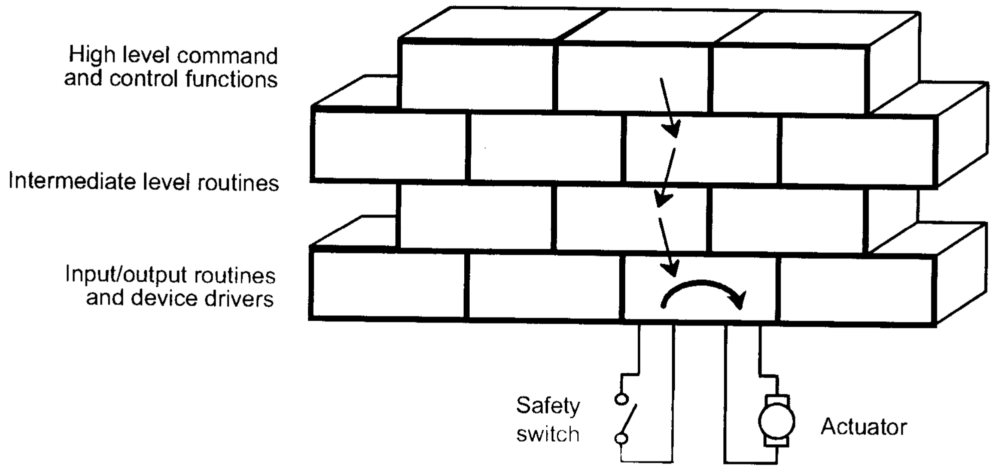
High level command and control functions
Intermediate level routines
Input/output routines and device drivers
Safety switch
Actuator

Figure 7.5 A better approach to system partitioning (from Storey, 1999).

Bibliography

R. Crowder, "An Overview of Risk Assessment and Management as Applied to Drive Systems", University of Southhampton, 1999.

R. Perry, "Programmable Electronic Safety Systems", IEEE, 1993.

N. Storey, "Design for Safety", University of Warwick, 1999.