Abstract
A general approach is presented to designing closed loop systems to be safe in the presence of hardware failures. Specific recommendations are made as to hardware and software features to include in microprocessor-based safety related systems to attain a fail-safe system.

Overview
Safety is a difficult thing to define. It ultimately becomes a value judgment, and it must be recognized that absolute safety is not possible. Moreover, safety in the use of any medical instrument requires that the user be trained in its proper use and operate the instrument in accordance with its directions for use. While this discussion will draw on examples from a closed loop drug delivery system, the concerns and techniques are applicable to any computer based system with safety implications.

Hazards can arise from several sources: from failures or sequences of failures in some component(s) of the system (the system breaks down), from a systematic or inherent fault (a design defect), from operator misuse, or from improper maintenance of the system.

Fundamental to any effort to create a safe system is an understanding of the potential hazards which the use of the system might entail. This understanding must include the environment in which the system will operate, and the people who will use and maintain the system. Only then can the system be organized so as to preclude the occurrence of these hazards. The ideal form of organization is to arrange it so that a specific hazard is physically impossible. Otherwise, specific features must be incorporated into the system to control hazards.

This process is best approached by a formal hazards analysis as part of the system requirements definition. Here, a list of potential hazards is generated along with an estimation of the severity of each hazard and its probability of occurrence[1].

We can address all of these hazard sources in the development of medical devices to varying degrees. Here, we will concern ourselves primarily with techniques by which a system can be made to fail in a safe manner in the event of a break down. It possible to achieve very high levels of safety in instrument design with acceptable impact on system cost.

Definitions
A FAILURE is the inability of a system or system component to perform its required function.

An ACTIVE FAILURE is the combination of one or more failures which are dangerous.

A PASSIVE FAILURE is a failure which is not dangerous in and of itself, but which in combination with other failures is dangerous.

INDEPENDENT FAILURES are failure which have no common cause.

DEPENDENT FAILURES, or those with a common cause (also referred to as COMMON MODE failures), must be treated as a single failure.

The SYSTEM CRITICAL TIME is the period of time during which the system can operate abnormally without causing harm to the patient or to the user.

Active failures must be detected and acted upon within the system critical time.
An OPERATIONAL WINDOW is the period of time over which we judge the probability of two or more independent failures combining to become dangerous to be acceptably low, and at the beginning of which tests are made to determine that specific failures are not present in the system. We strive to minimize the duration of this window so as to minimize the probability of multiple failures within the window combining to become dangerous. The measures that are taken to detect and react to failures typically look for passive failures only at the beginning of an operational window. Note that this low frequency of testing is acceptable only for passive failures: active failures must be detected and acted upon within the system critical time in order for the system to be safe.

Each system must be evaluated on its own merits as to the appropriate operational window for each failure mode under consideration. Any one system may have several different operational windows defined for different potential failure modes. The safe operational window must be determined and enforced independent of what the end user does with the product.

Examples of operational windows include:

- System power cycle. This may lead to an unacceptably long operational window for instruments which may remain powered up for several days.

- The beginning of a measurement cycle.

- The initiation of a treatment regime.

Failure States

If we assume that failures are independent and occur one at a time, the occurrence of a single failure will place the system into one of four possible states:

1. The system detects the failure, alarms, and enters the fail safe state. This is the preferred state for any failure.

2. The system is "dead" - it performs no function. If this is a safe state, it is acceptable. It is highly desirable to transform it to state 1.

3. The system continues to operate but in a hazardous fashion. This is the condition we must eliminate. The system must be modified to attain an acceptable level of safety. This is an active failure.

4. The system continues to operate with (apparent) impaired functionality. In order to judge the hazard associated with this failure, we must postulate all additional failures, one at a time, with this failure in place. This is potentially a passive failure.

Fail Safe System

It is axiomatic that a fail safe system can contain no single point failures which make the system unsafe. Experience has shown, however, that many hazards arise from a sequence of individually innocuous failures. The extent to which is necessary to worry about a sequence of more than two failures combining to become hazardous is dependent on the criticality of the system. Experience has also shown that it is not very much more difficult to treat a sequence of indefinite length than to treat of sequence of length two.

To remove some of the subjectivity as to what is safe, we will define a fail safe system as one in which

No sequence of failures occurring singly within an operational window shall cause injury to the patient or to the user.

Microprocessor System Safety

Self diagnostics provide a cost effective way to enhance the safety of processor based systems. The approach is to use the most reliable elements of the system to test the less reliable ones, and to expand the known good system to include the entire system. Starting with the central processing unit (CPU) (the most difficult to test because one fault may make it impossible to detect others), the known good system is grown to include the random access memory (RAM), read only memory (ROM), and critical input/output functions.

This section is intended as a guide for the hardware and software design required to ensure the safety of microprocessor based systems. The tests detailed here are a minimum set which the author feels are adequate for a closed loop device. These tests are not intended to be restrictive: equivalent tests of equal or greater fault coverage may be used.
CPU

The Central Processing Unit, at a minimum, consists of a set of registers, Arithmetic Logic Unit (ALU), instruction decoder, and system timing and control functions. Each of these functional elements must be verified to be functioning correctly.

CPU Registers

The CPU registers must be verified to contain no "stuck-at" faults. Each bit of each register must be verified to be capable of being both set and reset independently of every other bit, both in the register containing the bit being tested and in every other register.

Each register is written with a pattern and read back to confirm the ability to store the data correctly. For this test, a pattern of walking ones and a pattern of walking zeros is suitable.

To detect the possibility of cross-talk between registers, each register is loaded with a test pattern. The one's complement of that pattern is in turn be loaded into every other register, and the first register read back to confirm that it has not changed. A test pattern of all zeros and of all ones is suitable.

The ability of the status bits to be set and reset appropriately from test conditions must be verified. The independence of the status bits must be verified.

Conditional instructions must be verified as executing correctly from the corresponding status bits, both singly and in combination with multiple status bits.

The ALU must be verified for correct function. While it is generally not practical to test every instruction type with every addressing mode, every instruction type and every addressing mode should be verified. While it is not practical to test every argument value, significantly more than two argument values should be used in the course of which every data bit assumes both zero and one.

Failure of any element of the CPU must be detected within the instrument critical time.

Read Only Memory (ROM)

The read only memory must be verified not to have changed since its initial programming. The method recommended to ensure the ROM's integrity is a 16 bit cyclic redundancy check (CRC) of the ROM data pattern. The remainder of the used ROM data pattern is computed prior to programming the ROM, and is appended to the end of the ROM. This results in a zero remainder when the ROM is checked. This method provides far superior fault coverage compared to a simple checksum, at only a modest increase in complexity.

The choice of the generator polynomial is somewhat arbitrary. Vasa[3] has published an efficient software implementation of:

\( X^{16} + X^{12} + X^5 + 1 \)

The original article was for an F8 microprocessor. We have since ported that implementation to 8085's, 8080's, and V40's(8086's). As devices become more complex, and as greater functionality is added to the software, the sheer size of the ROM may require that the CRC generator be implemented in hardware in order to complete the ROM test within the system critical time.

The choice of the initial remainder (initial value of the CRC register) is again arbitrary. We find it convenient to use hexadecimal FFFF so that the CRC routines can also be used for communications purposes.

The CRC test, whether implemented in hardware or software, must be run in two parts: it must be run on a sub-set of ROM to verify that the CRC sub-system can detect an error, and it must be run on the entire ROM to verify that the entire ROM data pattern is unchanged.

Failure of the ROM must be detected within the instrument critical time. Hardware error detectors must be verified capable of detecting errors at least at the beginning of the operational window.

Random Access Memory (RAM)

Any microprocessor-based instrument contains Random Access Memory (RAM) which is critical to the performance of the instrument. To ensure the safety of these instruments, include tests of the RAM in the form of software self-check routines.

Single channel RAM must be verified to contain no stuck at faults; each bit of each RAM location must be verified to be capable of being both set and reset independently of every other bit, both in the location containing the bit being tested and in every other location.
It is permissible, but denigrated, to adudge certain areas of RAM to be non-critical. These non-critical areas need not be tested for cross talk into themselves; however, any used RAM location must be tested for cross talk into all critical RAM location. The microprocessor stack is critical RAM.

In the stuck-bit test, each location of the RAM is written with a pattern and read back to confirm its ability to store the data correctly. For this test, a pattern of walking ones and a pattern of walking zeros is suitable.

In the cross-talk test, each RAM location must be loaded with a test pattern. The one's complement of that pattern must in turn be loaded into every other RAM location, and the first RAM location read back to confirm that it has not changed. A test pattern of all zeros and of all ones is suitable.

To protect against disturbances which might cause an otherwise undetectable change in critical system parameters, critical parameters must be stored in diverse form. It is not sufficient to merely store a duplicate copy: the redundant information must be stored in a modified form, such as one's complement or two's complement. The equivalency of these two forms must be verified at least every system critical time, or before the use of the critical data.

The time required to perform the cross-talk test for the single channel RAM grows as the square of the RAM size. As the critical time becomes short, or as the RAM size becomes large, the test cannot be completed within the system critical time.

One solution to this problem is to provide a second RAM in the hardware and to compare data output from both RAMs for equality. This substantially reduces the amount of software testing required, but longer the test time back to acceptable levels. There are, however, still requirements for software tests to ensure the correct operation of the system. These tests are different than the ones used in single channel RAM systems.

Duplicate RAM's must be provided in which to store diverse versions of all RAM data. Any write to RAM (except any purpose of comparator testing, see below) must be written to corresponding locations of both RAMs. The data in the check RAM must be stored inverted from that stored in the main RAM. This makes it almost impossible for a common disturbance to effect both RAM's without detection.

An added degree of immunity to external disturbances can be obtained if no two address or data lines are common between the main and check RAMs, and if at least one address line is inverted between the main and check RAM. A comparator must monitor all reads from the two RAMs and must generate an error signal if any read differs.

A means must be provided to load different data into the main RAM and the check RAM in order to verify the comparator. Different data patterns are individually written to the two RAMs. This data is then read back in the normal way and an error is expected. If no error is indicated, then a passive failure of the comparator exists. The test pattern should cause the comparator to see a both a one-zero and a zero-one on every other location. The comparator must be tested at least once at the beginning of an operation window before any reads of critical data from RAM.

The data path test verifies the data bus connections between the processor and the RAM system for opens, shorts, and stuck-at faults which might be common to the main and check RAMs. A dedicated byte in the RAM is written with a pattern and read back to confirm the ability to store the data correctly. For this test a pattern of walking ones and a pattern of walking zeros is suitable.

The address path test verifies the address bus connections between the processor and the RAM system for opens, shorts, and stuck-at faults. A list of addresses is selected such that the address bits form walking ones and walking zeros patterns, plus the lowest and highest addresses, over the whole addressing range of the RAM. A cross-talk test is then performed over these addresses. One address is selected as a check byte and another as a test byte. The check byte is monitored for any changes while the data byte is complemented and restored. This is repeated through all possible combinations of addresses in the list.

The whole RAM exercise tests for the failure mode in which a seldom-used location in the RAM develops a fault which could go undetected for an unacceptably long period of time, thus increasing the possibility of multi-bit faults. Every byte in RAM is treated as a complement written, read again, and restored. Any errors generated in this manner will be detected by the hardware RAM comparator. Failure of the RAM must be detected within the instrument critical time.
**Unused Memory**

Means must be provided such that instruction fetches from unused portions of memory, whether ROM, RAM, or otherwise unused, yield a HALT instruction (with a subsequent watchdog activation) or result in the execution of an error routine, or the assertion of an error signal. This may be accomplished by means of pull up and/or pull down resistors on the data bus, or by explicitly decoding unused memory areas to generate an error signal.

**Critical I/O**

With the capability of a functioning processor, the verification of critical I/O function is relatively straightforward. Failure of critical I/O functions must be detected within the instrument critical time.

**Power Supplies**

Many subtle problems can result from out of tolerance power supplies. Failure of power supplies must be detected within the instrument critical time.

**Watchdog**

External hardware in the form of a watchdog is required to monitor that the processor is in fact running at some acceptable level of functionality. That minimum level of functionality is the successful completion of the processor tests of the CPU, ROM, RAM, and critical Input/Output functions.

The watchdog consists of an external time window comparator which is periodically strobed by the processor; failure of the strobe to arrive within the watchdog window must activate the watchdog.

Activation of the watchdog must override all critical processor controlled functions and must place the instrument into a safe state.

The long edge of the watchdog window must be less than the system critical time; the short edge of the watchdog window should be longer than the time to complete these diagnostics in an otherwise idle processor.

The watchdog time base must be independent of the processor time base. In order to detect crystals running at a harmonic of the intended frequency, the watchdog must be capable of detecting a two to one change in the processor (or watchdog) time base.

To ensure the successful completion of the microprocessor kernel diagnostics, an additional test in the form of a distributed calculation must be interspersed between each of the tests for CPU, ROM, and RAM. The result of this distributed calculation must be verified to be correct to enable generation of the watchdog strobe.

To verify that the watchdog is functional and is capable of overriding critical system functions, it must be tested at the beginning of an operational window by purposely activating it and verifying that it does in fact override processor control of critical functions. Both edges of the watchdog window must be tested. Failure of watchdog must be detected at the beginning of an operational window.

**References**

