MISRA The Motor Industry Software Reliability Association

MISRA c/o Electrical Group, MIRA, Watling Street, Nuneaton, Warwickshire, CV10 0TU, UK.
Telephone: (024) 7635 5290. Fax: (024) 7635 5070. E-mail: misra@mira.co.uk Internet: http://www.misra.org.uk

Report 3

Noise, EMC and Real-Time

February 1995

PDF version 1.0, January 2001

Acknowledgements

The following were contributors to this report:

Neil Andrewartha, Rolls-Royce and Associates Ltd
Terry Beadman, MIRA
Keith Longmore, Lotus Engineering
Tony Moon, AB Automotive Electronics Ltd
David Newman, Ford Motor Company Ltd

Summary

The subject of EMC is far too large and complex to discuss exhaustively in this document. References will therefore be offered which should be consulted by readers wishing to acquire detailed information. EMC standards generally relate level of immunity to criticality of function. Emissions of RF energy, which are also covered in the standards, are not considered relevant to this document.

Whether a system contains software or not, in principle, if electromagnetic interference cannot enter the critical parts of the system, it cannot cause malfunction. This means that hardware measures must be considered to be the prime means of defence: poor hardware design cannot be compensated for by software techniques. Often, hardware designs are compromised on the basis of reduced costs, and compensating in software; this policy is often a direct cause of poor EMC performance. However, nor it should be assumed that hardware alone is sufficient protection; rather, it should be assumed that some disturbing signals or corrupted data will enter the system, which will require a defensive approach to software programming.

This document will generally assume that hardware has been designed to reject EMI, and will consider only what additional steps may be taken in software.

Recommendations

These recommendations are organized as they appear in the MISRA Guidelines.

Software lifecycle

Requirements specification

Noise and electromagnetic compatibility

- Although RF emissions are part of electromagnetic compatibility (EMC), only immunity is considered here. Hardware measures to minimize electromagnetic interference (EMI) are the prime defence; however, this section describes only some additional measures that may be taken in software.

- Information on hardware EMC measures may be obtained from documents such as papers from international conferences and seminars, or the *Guidelines for the Achievement of EMC in Motor Vehicles* in which standards and sources are listed. Test methods and performance levels are given in ISO standards and the draft EC Directive amending 72/245/EEC.

- EMC standards identify test levels and failure modes for hardware functions. These may not be related directly to software integrity levels.

- EMC measures, in common with other aspects of system design, involve a trade-off between hardware and software solutions. Each has its own implications that should be assessed when making design decisions.

- Specific techniques for defence against EMI, whether hardware or software, should not be attributed to an integrity level, since individual techniques may behave differently under diverse circumstances, and may not give consistent results.

- EMI effects can only be minimized, not prevented, either in hardware or software. The most important role that software has in relation to EMC is in providing a means of recovery from the effects of interference.

- Hardware filters may introduce effects in the control system response that may be impossible to compensate for fully in software.

- Software may be used to:

  - digitally filter data
  - compare data with constant values, or values inferred from related signals, to identify errors

  - perform error detection and correction
  - dynamically adjust scaling to optimize signal-to-noise ratio.

- The following considerations apply to processors:

  (a) If the processor suffers EMI, software generally cannot provide any defence, and the system will require reinitialization.

  (b) Single chip controllers may improve immunity to some types of EMI, but may be less immune to others, depending on the design of the controller. Their integrated nature may make diagnosis and correction by software of EMI effects difficult.

  (c) It is essential to have an independent hardware watchdog designed to ensure reset and reinitialization in the event of EMI corruption. Do not rely on software watchdogs alone.

  (d) When choosing the timeout interval for a hardware watchdog, consider the response time of the system in the event of a fault.

  (e) Processors may be less immune to EMI when operating at the limit of their clock rates.

- The following considerations apply to data held in memory:

  (a) Corruption of data in memory can be identified and corrected by software. Routines to check for compatibility with expected results, against other data or by checksum comparisons, may be useful.

  (b) Data in ROM is generally less likely to be corrupted than in other types of memory. Critical data stored in RAM should be periodically recalculated and should recover from corrupted values.

- The following considerations apply to I/O devices:

  (a) The dynamic range of analogue to digital converters should be chosen to avoid "clipping" and maximize signal-to-noise ratio. Software cannot necessarily detect and compensate for signals that are corrupted in this way.

  (b) If multiplexed I/O is employed, it should be reinitialized by the software prior to use.

  (c) Whilst noise outside the frequency response of the system may be minimized by the use of either hardware or software filtering, in-band noise can be hard to detect and correct, especially if it affects a pulse train. Use software to compare frequency-critical data with expected or related data, and use default values, or resample, if an unexpected result is observed.

(d) Registers in programmed I/O devices should be regularly reloaded, or compared with their initialization data in ROM and reinitialized if in error.

(e) For slowly changing signals, compare newly acquired values with previous values, and discard if the variation is excessive.

- Communications data and processes may be corrupted by EMI. Software detection and correction techniques can be very effective, and should be used in addition to hardware measures.

Design

Real-time implications

- The use of CASE tools can assist in the assessment of the timing performance of the initial software design.

- There are many interrelated aspects of the initial design that are best analysed and resolved using a powerful computer-based modelling tool, such as:

  - types of input/output regimes
  - estimates of the number of statements or instructions to be executed
  - processor and memory resource knowledge and information
  - average instruction execution times
  - knowledge of any real-time kernel, executive or operating system
  - software filtering requirements
  - failure management requirements
  - relationship between the high level language statements and the code generated.

- Many of the aspects involved will be poorly defined at the outset and will have a large processing variability associated with them, so a large contingency for unknowns will be needed in the initial estimate of processing power and memory. Such aspects include:

  - searching (e.g. table lookup)
  - sorting
  - iteration (e.g. statistical, curve fitting methods)
  - stack, list, or queue oriented processes.

- Queue-based systems pose particular problems associated with computer resource planning and allocation. An understanding of the consequences of queues and the techniques to handle them is essential.

- Interrupts are useful in achieving good response times in a real-time control system. However, the quality and robustness of the software in this area can have a major impact on the overall system reliability. Interrupt routines require special attention to ensure robustness because:

  (a) Spurious interrupts due to noise can saturate the processor through excessive activity inhibiting other parts of the software from initiating fall-back operations.

  (b) Program structures using interrupts may create an unbounded number of paths through the software, increasing the difficulty of integration and system testing.

  (c) Asynchronous interrupts make it difficult for the software design to cater for all scenarios, which may result in data corruption.

  (d) Several concurrent processes initiated by interrupts that compete for resources may reach deadlock.

  (e) There can be conflicts of priority between one task's real-time response requirements and other tasks' integrity requirements.

- For robust interrupt handling, a selection of these or compatible techniques are recommended:

  (a) Use hardware filtering and buffering as the first line of defence against noise and excessive rates of data.

  (b) Use sufficiently independent and diverse mechanisms that can be provided by hardware and software to back up and protect interrupt handling, such as masking, prioritization and supervisor modes.

  (c) Maintain strong separation and modularity between the interrupt routines and other non-interrupt routines.

  (d) Use an existing kernel where possible.

  (e) Avoid using the program control stack for data. Set stack lengths to the theoretical maximum.

  (f) Follow suppliers' recommendations for interrupt design.

  (g) Employ personnel with experience of designing interrupt handlers for their implementation and modification.

  (h) For the higher integrity levels, consider using an independent monitor processor to initiate a safe state.

- The diverse nature of many of the proposed techniques means that complementary mechanisms can be overlaid to provide increasing robustness with rising integrity levels.

Floating point arithmetic

- Floating point arithmetic has several advantages related to quality and reliability:

  - numerical range is increased
  - the potential for numerical overflow is reduced
  - manual scaling by the programmer is unnecessary, thus one source of software error is eliminated
  - numerical accuracy is increased due to greater word length.

- The use of floating point, however, does not solve all numerical problems and may introduce some new ones. An awareness and understanding of general numerical methods remains essential to resolve issues relating to:

  - conversions to and from real and integer numbers and rounding problems
  - handling of divide-by-zero conditions
  - handling "Not-A-Number" situations
  - propagation of numerical errors through filters and integrators
  - build-up of rounding errors
  - the effect of differences in small numbers within ill-conditioned equations.

- In many real-time applications, the use of hardware to implement floating point may be required to meet response time requirements. This imposes different and special requirements on the system and software development processes:

  (a) Floating point co-processors can be corrupted by EMI and fail independently from the main processor. Therefore, the additional EMC constraints and on-board diagnostics requirements for floating point units are similar to those required for off-chip RAM and other separate devices.

  (b) It may be necessary to save and restore floating point data on receipt of an external interrupt. There can be a considerable processing overhead associated with combining floating point hardware with a requirement to handle a high input data rate.

  (c) Validation of floating point hardware units is a major and time consuming exercise and evidence should be sought from the semiconductor supplier.

- The use of hardware floating point arithmetic is not the ideal solution in every situation. An alternative approach is to use partial floating point implemented in software, only where it is required.

Fault management

- Many microcontrollers have on-chip watchdogs, which can provide powerful protection against some software errors, but will not protect against failure of all hardware components. If this is an unacceptable risk, then an off-chip watchdog or a separate monitor processor with separate timing circuits can be used.

Software quality planning

Quality assurance

Changes during production

- Production runs in the automotive industry are typically much longer than those in the semiconductor industry; consequently service support is a major issue.

- Masked microcontrollers are partially custom devices that are tested generically. It is not unknown for a device to undergo a process change and pass generic testing, but fail in the application domain.

- Users of microcontrollers should select manufacturers who notify of process changes as they are likely to control the change well. They should test advance samples at high and low temperature, under temperature change, high and low supply voltage and for EMC. They should also check restart of oscillation and any affected inputs and outputs for transient immunity.

- Problem containment depends on batch traceability. Semiconductor manufacturers should provide batch traceability of all products. Distributors of semiconductors and suppliers of subassemblies should not impede traceability, as root cause analysis depends on it.

Contents

Page

1. EMC in software-based systems

1.1 Definitions

EMC: The ability of an electrical or electronic system to function in its electromagnetic environment, without being unacceptably affected by, or adding unacceptably to that environment.

"Hung" processor
Processor "crash"
Processor "lock up"
A condition in which a processor incorrectly responds, or fails to respond to its programmed instructions, which may only be recovered from by performing a "hard" reset.

These terms are often used interchangeably, but there is no generic term, and, in principle, they mean different things.

Interference (EMI) Degradation of performance of an equipment, transmission channel, or system caused by an electromagnetic disturbance.

Disturbance An electromagnetic phenomenon which may degrade the performance of a device, equipment, or system.

1.2 Introduction

The subject of EMC is far too large and complex to discuss exhaustively in this document. References will therefore be offered which should be consulted by readers wishing to acquire detailed information. EMC standards generally relate level of immunity to criticality of function. Emissions of RF energy, which are also covered in the standards, are not considered relevant to this document.

Whether a system contains software or not, in principle, if electromagnetic interference cannot enter the critical parts of the system, it cannot cause malfunction. This means that hardware measures must be considered to be the prime means of defence: poor hardware design cannot be compensated for by software techniques. Often, hardware designs are compromised on the basis of reduced costs, and compensating in software; this policy is often a direct cause of poor EMC performance. However, nor it should be assumed that hardware alone is sufficient protection; rather, it should be assumed that some disturbing signals or corrupted data will enter the system, which will require a defensive approach to software programming. (Reference: MISRA Report 4 *Software in Control Systems* — Fault Tolerance, Fault Recovery)

This document will generally assume that hardware has been designed to reject EMI, and will consider only what additional steps may be taken in software.

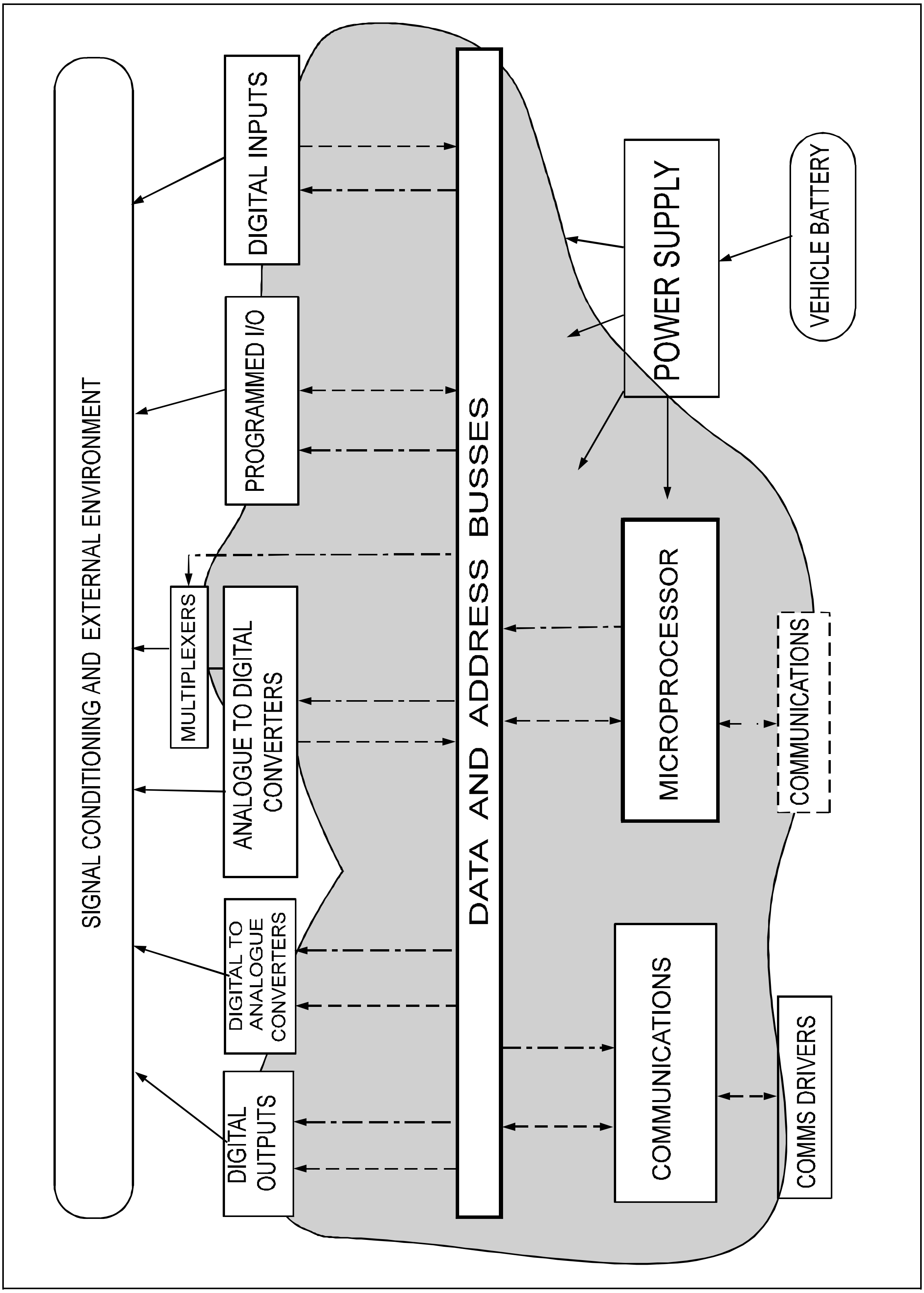


Figure 1 The scope for protection from EMI by software techniques

1.3 The central processor

1.3.1 Possible effects of EMI

If the processor itself becomes affected by EMI, it cannot usually take action to recover its composure; recovery will be dependent on the system’s watchdog to perform a "hard" reset and reinitialisation.

Misinterpretation/disruption of microcode:

- incorrect action resulting from an instruction
- processor "lock-up"
- incorrect branch, compare, or result

usually only recoverable by a "hard" reset, since the processor "gets out of step" with itself.

Missing or corrupted clock pulses:

- incorrect interpretation of an instruction
- loss of synchronism with program instructions

usually only recoverable by a "hard" reset.

Corruption of data on the data or address bus:

- misinterpretation of data
- incorrect branching
- incorrect interpretation of instructions

Recovery is possible from corrupted data; otherwise a "hard" reset is the only possible recovery solution.

Stack-pointer corruption:

- incorrect recovery from sub-routine
- incorrect data recovered from stack
- jump to non-existent memory location

usually recoverable by reinitialisation, providing this clears and resets the stack pointer to zero.

Corruption of compiled internal register contents:

- incorrect program execution
- corrupted result of compare instruction

- incorrect interrupt action

usually requires program reinitialisation.

Premature failure of the chip (extreme):

Large voltage spikes appearing on supply or signal lines to the processor chip can cause damage to the chip, especially if a regular occurrence. A circuit in which this happened would normally be very unstable; however, the EMC condition known as "load dump", if not adequately protected against in the power supply design, could with a single occurrence destroy a processor chip.

An occurrence of a single large spike can, however, do sufficient damage to a processor chip to cause a latent failure state, in which case the chip fails at a later, unpredictable time.

1.3.2 Single chip controllers

In single chip controllers and systems employing co-processors or multiple processors, the problem is more complex.

Single chip controllers usually contain, as well as the central processor, some I/O, program memory, RAM, and on-board clock. The latest types may also contain a numeric co-processor.

Single chip controllers used in automotive applications are often custom versions of standard "off-the-shelf" processors; this often means that rather than a single chip doing all functions for an electronic control module, or alternatively a host of individual chips, there will be perhaps two or three chips of custom design. It is difficult to draw meaningful comparisons between customised systems of this type and commercial "off-the-shelf" single chip designs. The latter are much more likely to be genuinely **single** chip, containing all functions of a system.

A distinction must also be drawn between the behaviour to be expected from commercial products and that to be expected of custom designs: often this shows up in such areas as EMC performance-commercial designs are frequently designed for other than the automotive environment.

Opinion as to whether single chip processors are more or less susceptible to EMI than multi-chip designs is divergent, so the only practical course for this report is to present the pros and cons for each type, and recommend careful consideration in deciding which route to take.

Single chip processors may be totally self-contained, or else allow bus connections to external devices; this distinction will be used to differentiate between genuine single chip, and multi chip types.

Self-contained single-chip processors

Pros

- Lack of external bus causes less emissions of EMI, therefore, potentially, less susceptibility to EMI.
- Only single point filtering for power supply protection.
- Silicon technology used in many single-chip processors may reduce susceptibility to EMI.

Cons

- I/O is very likely to be programmed, therefore may be more subject to corruption (see Programmed I/O)
- Lower visibility of system component parts-may be harder to detect EMI corruption

If EMI does find its way onto the chip, its effects may be much more difficult to recover from, and the outside world may be brought directly (via filtering) on to the chip.

Single-chip processors with external bus

Pros

- Greater flexibility in application
- May allow sensitive or vulnerable parts of I/O to be made remote from central processor, thereby improving processor's ability to cope with I/O corruption, and improve processor isolation from the outside world.
- Permits external access to memory, thus aiding emulation and diagnosis, if suitable peripheral equipment is available.

Cons

- Possibility of EMI entering the bus structure directly.
- Protection measures needed for each circuit element.
- External watchdog may itself be vulnerable to EMI.

Single-chip controllers rely on most of their I/O being defined by control registers, which are set by software. This is a source of potential susceptibility. [Refer to the section on programmed I/O later in this document.]

There is also a greater risk of noise effects if analogue, digital, high current, and the processor circuitry share the same piece of silicon.

1.3.3 Numeric co-processors

These are considered separately, and the relevant section should be consulted.

1.3.4 Protection of the central processor

Any safety-related circuit employing a microprocessor in the automotive environment must employ a watchdog arrangement, preferably which is accessed at time-critical points in the software cycle, rather than just at the extremities of the cycle. In a routine which encompasses a time critical function, it is frequently desirable to employ a local error/fault detection/recovery/default scheme, to ensure that the occurrence of a fault invokes default action, perhaps with a background fault-recovery routine that allows the program cycle time to remain within desired limits. If error/fault recovery fails, the default scheme can, in the limit, invoke a system reset/reinitialisation. In general, the purpose of the systems watchdog is principally to cater for central processor becoming "hung". It should only be invoked by serious error situations, not by any one fault (Reference MISRA Report 4 *Software in Control Systems* — Watchdogs].

If possible, the processor, irrespective of its maximum clock frequency capability, should not be run at its maximum clock speed, since this may make it more susceptible to EMI-induced corruption, both in terms of clock pulse corruption, misinterpretation of opcodes, and corruption of microcode. This is because the technology of the processor is the limiting factor in determining maximum clock frequency; any disturbance when it is operating at the limit of its technology's capability is much more likely to result in an interference effect.
However, because technology is the limiting factor, a fast processor using a high clock frequency need not be more susceptible to EMI than a processor with a slower performance and low clock frequency.

It should not be assumed that the presence of a d.c. regulator chip will prevent processor interference by supply-borne EMI, nor that the regulator itself will be immune from the effects of EMI.

1.4 Memory

Traditionally, vehicle systems have operated with program and fixed data held in mask-programmed ROM, and a small amount of RAM for temporary data storage—"scratch pad" memory.

Masked ROM is normally, of itself, not corruptible by EMI (although the data or instructions read from ROM may be corrupted during the read cycle of the processor); RAM may be more easily corrupted. However, systems are now appearing which are "end-of-line" programmed, into EEROM, or battery supported NOVRAM. Some systems also employ EPROMs for retention of data. These types of memory are somewhat more easily corrupted, and potentially, permanent corruption is possible.

1.4.1 Possible effects of EMI

Corruption during a read cycle:

- data faulty, causing incorrect action
- corrupted instruction causing incorrect action or processor "lock up".

Faulty data may be detected and a recovery process invoked; faulty action usually requires a "hard" reset and reinitialisation.

Corruption during a write cycle:

- faulty data value stored
- faulty address pointer value stored
- data stored at incorrect memory address

Faulty data may be detected, providing:

(a) it is read back and checked at the time of writing; or, if this is impractical,
(b) checked against expected value before use.

The latter assumes that the data would be sufficiently corrupted to be significantly in error. Address corruption, including corruption of the address information for storing the data, is likely to require a "hard" reset and reinitialisation which will be likely to affect another part of the program cycle.

Permanent corruption of data or instructions held in ROM (extreme):

- will cause recurring fault condition, which may vary in seriousness from minor data corruption to address or instruction faults leading to processor "lock up" ("hung" processor).

Not recoverable, except by replacement of the affected ROM, or replacement of its contents with uncorrupted code.

1.4.2 Protection of memory

ROM is normally adequately protected by attention to power supply design and individual decoupling at each chip. Corruption of data during the read cycle may be detected by multiple read-and-compare operations; corruption of instructions however, needs a "voting" system, or a monitor arrangement to ensure that a sequence is correctly carried out. The latter type of arrangement carries a cost penalty, and also both schemes may carry a speed penalty.

The use of EEROM, NOVRAM or EPROM is a more contentious matter. EPROMs are rarely permanently affected by EMI, though it is a possibility which should not be overlooked, but EEROM and NOVRAM need special care to ensure that permanent corruption does not occur.

Particular attention must be given to power supply and decoupling issues; during the start-up sequence, it is advisable to carry out a CRC check of memory contents, comparing against a value held in masked ROM or possibly hardware.

1.5 I/O circuitry: analogue to digital converters

The essential prerequisite for the use of analogue sensors is that the output signal is as large in amplitude as practicable consistent with allowing enough overrange to avoid the possibility of clipping, and any signal conditioning circuitry is band-limited to minimise the effects of high frequency noise. It may also be appropriate to employ balanced-pair wiring with differential amplifiers to minimise common-mode interference at lower frequencies.

It is not uncommon in automotive applications for a single A-D converter to be multiplexed to a number of incoming signals, and it should be borne in mind that if EMI corrupts the A-D converter itself, all the multiplexed signals may be corrupted. The choice of multiplexer channel switching rate may have a significant bearing on the seriousness of the effects of incoming EMI; if the multiplexer is affected by EMI, such that its channel switching rate is corrupted, it may become out of step with the conversion, and cause incomplete or faulty conversion of some of the incoming signals to digital form.

It is advisable to restrict the use of multiplexed analogue inputs to those signals which are of relatively low criticality.

1.5.1 Potential effects of EMI

Multiplexer corruption:

- loss of synchronism with the A-D converter control signals
- inhibition or corruption of data transmission in one or more channels
- "clipping" of data on one or more channels

It is possible to detect faulty data, but if channels are sampled out of synchronism with the program expectations, wrong but plausible data may go undetected. Missing data will be easily detected, however, "clipped" data may be difficult to detect, and if the "clipping" of a dynamic signal causes significant harmonic amplitudes to be present, this may cause aliasing or other problems. Reinitialisation may be needed to restore synchronism.

Noise causing saturation of analogue signal conditioning circuitry:

- can cause bias change in amplifiers
- can result in "clipping" of analogue signals
- can cause frequency-domain distortion of dynamic signals

Often very difficult to detect, unless the result is out-of-range data; however, the situation will recover itself after removal of the interfering signal.

1.5.2 Protection against EMI

It is often assumed that differential input A-D converters are less susceptible to EMI than are single-ended input types. This should be treated with caution: the use of a differential input type may simply allow an additional route for interference or an additional corruption mode to occur. Often, differential input types are chosen where there is a specific low-frequency interference problem, or a specific grounding structure problem. Attention to minimising the source of this type of problem to allow use of a single-ended input A-D may be a better option than employing a palliative approach such as employing a differential input A-D.

Sufficient "headroom" should be allowed in the dynamic range of the A-D to ensure that corrupted input signals do not cause the input to be driven to, or beyond, permissible limits. Some A-Ds give erroneous results when operating close to, or outside, their permitted dynamic range.

Assuming that all relevant measures have been taken in hardware, for a relatively slow-moving signal, a number of conversions of the signal may be taken, and averaged in software.

If signal amplitude is sufficient, performance may be optimised by software-controlled attenuation, in order to maintain, as far as possible, a high signal amplitude at the A-D converter input. The level of attenuation may be predicted from other parameters; signals then falling outside the expected range for the level of attenuation may be considered erroneous, and corrective steps taken. It is important to ensure that dynamic range of any analogue signal is also carefully considered, and designed to maintain as large a signal-to-noise ratio as possible.

Multiplexers must have sufficient signal voltage capability to ensure that disturbance signals superimposed on wanted signals does not result in interference with the multiplexer’s proper operation.

1.6 Digital I/O

Many vehicle functions require a simple "is switch on or off?" monitor. However, such parameters as engine speed or road speed require that the frequency of a pulse train is measured, and converted into the appropriate units. This type of signal may also contain positional information, as when ignition advance is involved. The two types of signal are handled together. (The third type of digital I/O, communications-is discussed separately)

It is relatively straightforward to protect slow state-monitoring circuitry in hardware; if a need exists to add software protection, this need only be, say, three reads, which must give the same result.

1.6.1 EMC effects on fast digital I/O

Apparent additional pulses in a pulsetrain:

- may be read as wrong, but plausible, data.

Change of bias conditions in pulse squaring circuit:

- may cause amplitude change of squared waveform
- may inhibit squaring circuit
- may distort waveform from squaring circuit.

Apparent extra edges on positional signal waveform:

- edge detection circuitry is often used for position detection; edge-detection circuitry is usually more susceptible to EMI corruption than level detection circuits.
- may occur due to corruption of switching point in edge detection circuitry.
- may be inhibition of positional information.

1.6.2 Protection in software

In principle, it is difficult to protect against "in-band" noise. The first requisite is to employ the largest possible amplitude of pulse train; with, for example, a magnetic speed transducer, this may be problematic, given the large speed range of vehicle components such as the engine, or road wheels. Since the voltage output by magnetic devices is dependent on the rate at which the magnetic flux is changed, output voltage increases quite dramatically from a magnetic transducer with increasing speed. There will be lower signal to noise ratio and therefore greater vulnerability at low speeds, as well as lower confidence in the receiving circuit accurately measuring the speed.

There are two main ways in which software can help: by comparing with other related parameters, an expected speed may be calculated against which the measured speed can be compared; or several readings of speed may be taken and averaged, or any wildly different readings discarded. These methods however, only increase confidence in the readings, and cannot, in themselves, guarantee 100% correctness.

Some significant benefit may be obtained by employing transducers with inbuilt signal conditioning, however, if utilised, they must be designed for the automotive environment.

Positional information either takes the form of, for example, locating top dead centre or a related angular position in the engine rotational cycle, or referencing the position of, say, a throttle "jack". The latter is often carried out by energising a stepper motor to one extreme of movement, then counting steps back to the required position, and relating all other adjustments to the set reference position. Interference can cause the stepper to be inaccurately set; however, since it is not using any input circuitry in a feedback configuration, providing the driver circuits are adequately protected in hardware, there is probably a greater likelihood of errors due to mechanical reasons.

The engine rotational position signals are employed by engine management systems for timing injection of fuel, for setting ignition timing, and for timing the reading of for example the knock sensor signal. If noise enters the signal circuitry, injection of fuel may be at the wrong point in the firing cycle, or worse, ignition may be wrongly timed. It is most common for the reference signal to consist of two simultaneous pulses, both of which must be present. Whilst it is theoretically possible for interference to affect both signals, it is rather less likely that two simultaneous signals could be spuriously generated at another point in the engine cycle. This is to a large extent obviated by the configuration of the (usually) magnetic sensors: the second, reference, sensor is maintained in the magnetically saturated state, except at the reference point itself, so is fairly immune to EMI. Interference occurring at the reference point could inhibit the detection of the reference, possibly causing a misfire, but the more likely effect is that other firing signals are corrupted, causing misfiring or perhaps a "knock". Such effects are usually confined to one cycle, since at the next reference signal, correct synchronism with the rotational cycle will be restored. Software may be configured so that the reference pulse edge is only "looked for" within a time window, the latter being adjusted to be appropriate for the engine speed and expected reference position. Once the edge is detected, any further "reads" of that signal should be inhibited until the next engine cycle.

1.6.3 Programmed I/O

Often I/O devices are used which must be set to a required configuration by commands in software; that is to say, control words loaded into registers in the I/O device to configure lines as input or output, and possibly for negative-going or positive-going signals, edges or levels, and in some cases, for frequency counting. This is especially the case with single-chip processors. [See reference earlier in this document]

Both communications drivers (e.g. UARTs) and digital I/O port devices (e.g. 8251, 8255, 6821) require to be configured in software.

1.6.4 Effect of EMI

Interference may cause the contents of a program register to be changed; this will have the result that, for example, an input becomes an output, or perhaps is disabled; a negative-going requirement is changed to a positive-going requirement, and so on. Signals may then not be present when required, or may be wrongly interpreted.

1.6.5 Protection in software

In principle, the status of the device should be checked at regular intervals, and re-written if incorrect. In practice, this is may not be either possible, or may cause additional problems. A typical programmed I/O device has a control register; a status register; and whatever tri-state buffers or additional circuitry is necessary to perform its task. The status register (e.g. in the case of 8255 USART) is provided so that error states may be detected, and flagged to the processor for reset of the device and rewrite of the control register word. Some devices (e.g. 8251) do not have a status register.

It should be recognised that the status register, in the face of EM interference with the chip’s operation, may not reflect the control register state; the chip manufacturer’s literature should be consulted to assess what it interrogates. For example, in the case of the 8255, the status register accesses external pin states; in the 6821, it may access only parts of the internal circuitry which do not necessarily indicate the state of external pins.

Before **any** programmable I/O device is written to for programming the control register, it must be reset. A failure to reset the chip may result in erroneous operation. It is not uncommon to apply a reset signal more than once prior to reprogramming.

It must also be said that not all chips carrying the same number, but different manufacturer’s logos, are internally similar, carry reverse protection diodes, access status in the same way, respond to interference in the same way. It is advisable to maintain supply from the manufacturer whose chips have passed EMC tests in the design stage.

If possible, the simplest action to ensure minimisation of programmed I/O errors is to read back the status register contents at regular intervals, and compare with expected status word in memory and initiate a reset/rewrite if necessary; otherwise, it is possibly worth adopting a regime to reset and rewrite the control register contents at intervals. It must be recognised, however, that I/O data must be disabled during this process, so it should not be carried out on a simple timed basis; possibly an acceptable timing would be "after *x* program cycles, providing engine speed (road speed, etc) is less than *y*". It is not endearing to users of the vehicle to find the engine management system (ABS system, traction control system) goes into a default state during an ill-advised overtaking (panic braking, need-to-get-out-of-the-way-of-that-large-truck) manoeuvre, for instance!

1.7 Communications

Communications serves three purposes in an automotive context:

- downloading of data to, or interrogation by, off-board diagnostic devices
- communication of data to a dashboard display module, or central on-board diagnostic device
- communication between intelligent control devices, either as part of a whole vehicle multiplex wiring control scheme, or between, for example, an engine management system and transmission controller.

With increasing complexity of control schemes, there is widened scope for control functions to be embedded in more than one controller module.

Communications schemes comply with a protocol which is a set of rules which all participants in the communications scheme must obey in order to communicate. There are four schemes currently recognised for vehicle use: SAE J1850; CAN; VAN; and ISO 9141.

For details of these schemes, see MISRA Report 1 *Diagnostics and Integrated Vehicle Systems* and the relevant standards. Most modern schemes require synchronous communication, since synchronous schemes are more efficient in their use of available bandwidth: asynchronous schemes require that every bit is synchronised, whereas synchronous schemes require that every "packet" of data only is synchronised. In general, asynchronous schemes are efficient for small amounts of data, and generally are restricted to 8 bit data words. The application of Hamming or Manchester codes is less efficient with asynchronous schemes, since e.g. Hamming requires 3 bits of an 8 bit word.

In general, there is much more widespread, and cheaply available, hardware for synchronous communications than asynchronous. Often, current practice is to build in synchronous communications onto processor or interface chips "in with the price". Modern protocols employ very sophisticated error correction and detection arrangements; in the case of synchronous systems, the receiving circuit must carry out this function, and if an error is detected, request retransmission if it cannot be corrected. However, it is often more efficient to ignore erroneous data, and wait for the next transmitted "packet". In synchronous systems, there is a large overhead of header information broadcast with each "packet" of data, which makes such systems inefficient for small amounts of data.

1.7.1 EMI effects

Interference may corrupt data communicated to another controller; with most communications protocols this is easily detected. There are well-established error correction schemes in existence, and the use of these is recommended. It must be recognised that error correction cannot cater for all magnitudes of error in a communications scheme, and in cases of serious corruption, it is likely that retransmission will be required.

One problem is that the sending circuit may become temporarily "hung" due to the processor or a line driver device being corrupted, leaving the receiving circuit in a wait state, expecting data that never arrives. In the meantime, control of a critical system may have been lost, with catastrophic results.

In the case of communications to a dashboard display this may have no serious effect, other than annoyance; if it affects off-board diagnostic tools, the operator will perceive the problem and reset the system. The major problem is with a vehicle multiplex system, or inter-controller communications.

1.7.2 Protection schemes

It is assumed that the system designer will have selected an appropriate protocol for the communications scheme, taking into account the environment, criticality, and cost implications. It is not for this document to recommend communications protocols.

It is most important to ensure that the system cannot become "hung". Consider the situation that a receiving device is inhibited from sending an "acknowledge" signal due to the influence of EMI. The sending device is then set into a "loop" waiting for the "acknowledge"; but the receiving device is only programmed to send one "acknowledge", so the system becomes "hung". There must be a local watchdog arrangement which, in the event that the circuit becomes "hung" resets both ends of the communication circuit, and reinitiates transmission of the data.

It is also essential that data is checked for correctness. Most transmission protocols encompass an error checking scheme: addition of parity bits; CRC checks; and multiple transmissions with compare being most common. The most serious problem associated with corrupted data is that of propagation: wrong, but plausible, data communicated to another system may cause a catastrophic problem—especially if it results in erroneous data being passed on to yet other systems in the vehicle. The integrity of transmitted data must be assured before it is allowed to be used by its receptor; any data anomaly or uncertainty MUST result in at least re-transmission, or preferably, default action while the sending system is interrogated or reset.

The boundaries of and paths for propagation of transmitted data must also have high visibility to default and error recovery schemes, in order that, in the event of an error being reported, **all** actions can be taken to correct the fault.

It is often believed that fibre-optics offer vastly better defence against EMI in multiplex systems, compared to wired systems. This is often not true, since the LED transmitters are themselves very prone to direct illumination by RF energy, causing erroneous transmission, or inhibition of transmission for the period when the RF energy is present. **This cannot be protected against by hardware or software**, except, up to a point, by the use of a metal enclosure, with the LED well inside the enclosure, and only sufficient aperture to allow access to the fibre-optic cable. Frequently, automotive systems do not employ metal enclosures, so fibre-optic systems must be treated with caution.

Probably the most effective medium from an EMC point of view is a balanced twisted pair signal bus, possibly as a screened twin.

In general, faster data rates imply greater susceptibility to EMI; they will certainly imply greater propagation of noise into the world at large. Reduction of propagated interference requires that risetimes of edges should be no faster than absolutely necessary; ensuring this is also likely to reduce susceptibility to EMI.

1.8 General software mechanisms for EMI protection

Software may be used to:

- filter or band-limit
- average
- compare
- parity or CRC check
- scale.

Filtering may be used to band limit, or to provide "notch" filtering for specific problem frequencies. There is less applicability for the latter in automotive applications than most, because of the random, wideband nature of automotive interference signals.

Averaging minimises the effect of data which is significantly away from the mean of several readings, as well as smoothing the effect of lower level noise.

Comparing a number of memory reads aids in identifying a situation in which EMI has caused a momentary corruption of a read.

Parity and CRC checking may be used to check for transmitted data integrity, or for the integrity of stored data.

Scaling may be used to optimise signal-to-noise ratio of input data. It may be used to range select A-Ds; or to adjust attenuation of input signal conditioning.

1.9 Single-chip controllers

In principle, single-chip controllers are little different from conventional systems. However, they present greater problems for EMC, because of the high level of integration, and, often, programmability of I/O. It is not untypical for a single chip controller to have on board numerical co-processor, 8 or 16 channels of programmable I/O, serial data link, scratch-pad RAM, EPROM or EEROM, timers, interrupt handling, and clock circuitry.

If EMI has an effect, it is much more likely to be catastrophic than in the case of a multi-chip system, where there may well be EMC protection applied at an individual chip level. However, the cost-effectiveness of the single-chip solution is rather seductive, sufficiently that the risks associated with automotive use may be ignored or at least trivialised. The use of single chip controllers for safety-critical applications must be tempered with significant caution, and even then, not proceeded with unless there is a very good assurance that the controller being considered has been expressly designed for the automotive environment.

1.10 Integrity: relationship with EMC

It is accepted that the norm for integrity classification is converging to five levels. MISRA has accepted the use of this standard. The classification of EMC integrity into five levels, if software protection from EMI is the sole target of the classification, is probably less than sensible. EMC integrity is achieved by a combination of actions and techniques and levels of rigour which are generally uniquely configured for the product under scrutiny, to cater for its particular circumstances. A discussion of how this is achieved is therefore appropriate.

Irrespective of the test type and configuration, EMC performance is specified as a set of levels of effect (incapacity) resulting from EM disturbance, against a set of test levels, forming a matrix of results. [ISO] Five levels of incapacity are specified:

I. No effect
II. Continued operation, possibly with excursion of some parameters outside specified limits; no noticeable effect to a casual observer or driver
III. Continued operation, with excursion of some parameters outside specification limits, with clearly degraded performance; during exposure to RF; automatic recovery on removal of RF
IV. Interruption of correct operation during exposure to RF; unassisted recovery on removal of RF
V. Interruption of correct operation during exposure to RF; driver action needed to restore normal operation.

All of these effects assume that there will be no permanent damage to the victim circuit.

ISO specifies up to four test levels for a range of up to seven test severities, any of which may be specified by a manufacturer or supplier to align with any of the effect levels shown above. This is best interpreted as aligning test level with criticality of a **function**, which effectively mandates testing at the highest severity, and assessing effects based on their criticality, and therefore the minimum test levels at which the effects occur.

This matrix of test and effects levels do not necessarily align with **software** integrity levels, because they are considering the system **functions**, not software design issues.

Test results are also dependent on which test type is used; differing test types produce differing results with the same victim circuit. Different victim circuits exhibit different modes of susceptibility for the same test type. This is due to the fact that coupling mechanisms differ for different circuitry as well as different test types. Unfortunately correlation of different test types is little better than empirical, so, for practical purposes, the number of variables in this matrix cannot be reduced. However, the ACEA (Club of European Car Makers) draft test standard for compliance with EC Directive on EMC only suggests a minimum test level for each test type for homologation purposes, for systems having direct influence on the control of the vehicle. It is widely accepted, despite the ACEA draft, that whole vehicle test is the test type most representative of the "real world", and this can be taken as one basis for quantifying software integrity for EMC. The EC Directive does not consider, for example, conducted transients internal to the vehicle systems, and is thus inappropriate for use as a total design aim.

Integrity based on the above becomes a matter of setting a design target test level for unaffected operation for each function based on its criticality, and testing to ensure that the design targets are met.

The complementary influence is that of technique. It cannot be specified that any particular software technique for resisting the effects of EMI offers a higher integrity than any other software technique, neither does rigour offer a convincing argument. Therefore a measure is needed which reflects the reality of the EMC situation.

In principle, EMC is primarily resisted by hardware means: filtering, choice of circuit configuration, components, etc. To this line-up may be added software filtering, parity checking, CRC checking, multiple processes, etc. As criticality increases, it may not be appropriate to suggest that, for example, no protection should be supplanted by an increasing number of techniques up to a combination of all available techniques at high level criticality: what is needed at high levels of criticality is a better understanding of the limitations of the victim circuit which need to be addressed, and an application of the appropriate techniques to deal with those limitations effectively. The issue to be addressed, therefore, is perhaps one of procedural and intellectual definition.

Another measure important to EMC tolerance is recovery procedures. Most software based systems require some form of watchdog arrangement—even if only from a customer satisfaction point of view. As criticality increases, it becomes essential to provide something more—a scheme to handle the detection and recovery from the fault condition. This is not uniquely an EMC matter, however, and is considered elsewhere in MISRA reports, so will not be dealt with in detail here.

1.11 Bibliography

[1] Data Transmission, Edited by Tugal and Tugal; Published by McGraw-Hill, Second Edition, 1989.

[2] Data Communications Pocket Book, Author: M Tooley, Published by Butterworth Heinemann, Second Edition 1992.

[3] ISO/TC22/SC3/WG1 Document N422: CAN/ High Speed Protocol : Road Vehicles-Serial Data Communication for Automotive Application. October 1990 (Available from Society of Motor Manufacturers and Traders Ltd, Forbes House, Halkin Street, London SW1 as TEC/1157/1990 A.)

[4] ISO/TC22/SC3/WG1 Document N 426 E: Low Speed: Road Vehicles-Serial Data Communication for Automotive Application. September 1990.(Available from Society of Motor Manufacturers and Traders Ltd, Forbes House, Halkin Street, London SW1 as TEC/36/1991.)

[5] ISO/TC22/SC3/WG1 Document N 429 E: Low Speed (SAE J1850): Road Vehicles-Serial Data Communication for Automotive Application.(Available from Society of Motor Manufacturers and Traders Ltd, Forbes House, Halkin Street, London SW1 as TEC/1157/1990 B.)

[6] SMMT Guidelines for Electromagnetic Compatibility in Road Vehicles; August 1992. Available from Society of Motor Manufacturers and Traders Ltd., Forbes House, Halkin Street, London SW1.

2. Timing constraints and considerations

2.1 Introduction

A real-time system can be defined as one that is required to react within a specified time to physical events and changes. The response times involved can have a very broad range, from just microseconds, (for example, 1 degree of engine crank rotation at 7000 RPM is 24 microseconds) up to tens of minutes, for example the engine heat transfer. Most control systems have a mixture of sensors and actuators with response time requirements spread between these extremes. The system and software designs, and choice of hardware has to accommodate this spread.

The initial project planning and design phase to meet the timing requirements is a complex and interactive process involving estimates associated with factor such as:

- The dynamics of the objects to be controlled (see MISRA Report 4 *Software in Control Systems*)
- Assumptions about the control algorithms to be used
- Types of input/output regimes (see Use of Interrupts)
- Relationships between high-level-language statements and code generated
- Estimates of number of statement or instructions to be executed
- CPU and Memory Resource knowledge and information
- Average instruction execution times.
- Knowledge of any Real-time Kernel, Executive or Operating System
- Input/output dependency/reconfiguration requirements
- Direct or mapped input/outputs
- Estimes of Queueing times (see Queueing for Computing Resources)
- Software filtering requirements (see EMC requirements)
- Failure management requirements (see Diagnostics)
- Contingency for growth and unknowns
- Past experience, etc.

2.2 CASE tools

Many Computer Aided Software Engineering (CASE) tools have modelling features to aid and support this initial design phase and can play an important role in the generation of quality estimates and documenting the results. A primary requirement in this early phase of the system and software design is to ensure a high degree of confidence that the requirements will be achieved and that sufficient computer resources have been allocated to meet the processing, control and response time requirements.

Computer Aided Software Engineering (CASE) tools are primarily designed to assist programming personnel to effectively apply software engineering techniques. Increasingly however, these tools support hardware design as well and a better interpretation of the acronym might now be Computer Aided **Systems** Engineering. These tools, which are now entirely workstation or PC based, help to automate the processes within the development life-cycle, from analysing initial requirements to production of executable program code. They provide a range of facilities associated with structured design, modelling and documentation, including:

Requirements:

Requirements Capture
Interface to modelling facilities
Requirements Analysis
Formal mathematical methods analysis
Requirements Simulation

Design:

Modular Design
Logical Design
Data Structure Design
Graphical User Interface
Prototyping
Computer resource modelling
Queue modelling & analysis
Code Implentation
Hardware/physical Design
HDL/VHDL Interface

Integration and Test:

Test data preparation
Module test
Transfer of manufacturing data
Static Analysis
Dynamic Analysis
Documentation composition
Reverse Engineering

Examples of typical commercially available CASE tools are (in alphabetic order):

ASA & Geode
CASE-W
OBJECTMAKER
MENTOR
SOFTWARE-THRO-PICTURES
SYSTEM ARCHITECT
VERILOG

AUTO-G
DEC-DESIGN
MASCOT
SABER
STATEMATE
TEAMWORK
EXCELERATOR

Many of these tools, electronic systems CAD tools and control systems modelling tools (see Feedback) are now being interfaced and integrated into "total development environments".

"Tools architectures" and "CASE Repositories" are being developed in association with standards activities both from a base of the general computer supply industry and from within specific industrial sectors.

Examples of these developments are:

CBASS (SafeIT project)
MSR (German automotive industry)
COHESION (DIGITAL)
SOFTBENCH (HP/IBM)

2.3 Processing variability

The quality and consistency of the estimates associated with input/output handling will be influenced by knowledge of potential variability.

Some processing algorithms can have variable, data-dependent and even unpredictable execution times. Examples of these are:

Searching e.g. table lookup	(data dependent)
Sorting	" "
Iterations e.g. statistical curve fitting methods etc.	" "
Stack, list, queue oriented processes	" "
Recursion	(unpredictable)

These techniques may be imbedded within proprietary and standard real time kernels, and high level languages and hence transparent to the application user.

2.4 Realtime kernels

When planning and designing embedded real time systems the decision of whether to use an existing Kernel or Executive versus designing a dedicated real-time scheduling mechanism is of major significance:

In favour of using an existing kernel	**Some disadvantages of using existing kernel**
Project Quick starts	Timing overheads and variability
Proven	Memory overheads
Lower cost	Confidence in documented execution times
Flexibility	
Reconfigurability	Over generalisation leading to unnecessary complexity for any one application
	Hidden undesirable effects

There is a multitude of accepted techniques available to optimise the allocation of CPU time and resources. (See [1]–[7]) Many have their origins in the larger and more complex systems, with large number of I/O devices. Generally, the smaller the system, the less appropriate many of these techniques are likely to be, and the more appropriate it may be to adopt dedicated and unique designs. However, for the higher levels of controllability, use of existing products has a benefit in demonstrating "best available technology".

2.5 High level languages

Whilst high level languages offer advantages in reducing complexity and adherence to structured techniques, they can introduce not only overheads in run-time processing time, but can also introduce a degree of hidden and hence unknown variability, due to effects such as stack and list processing, parameter passing, etc. (See MISRA Report 6 *Verification and Validation* for other aspects of high level languages)

2.6 Noise and communication errors on inputs and outputs

Noise introduces not only variability in processing and response times, but will generally slow the system down. The design must accommodate the worst case requirement to be able to process maximum identified error rates while maintaining a service (see "Effects of EMC on Real-time Systems").

2.7 Queuing for computing resource

It is normal in multi-tasking and message based systems to use queues as a means of organising and prioritising the use of available processing power. Part of design and resource estimating involves assessing the effects of this on response times and estimating maximum queue lengths to determine storage requirements.

An understanding of, and the application of, Queueing Theory is necessary to undertake this design work [1, 2], although the queuing algorithms involved may be part of CASE modelling tools.

The consequences of queueing for computer resources has much in common with the more familiar types of queues in every-day use, such as the traffic queue at a roundabout or the queue in a bank or supermarket. Namely, at busy periods the nature of random arrival for service can cause:

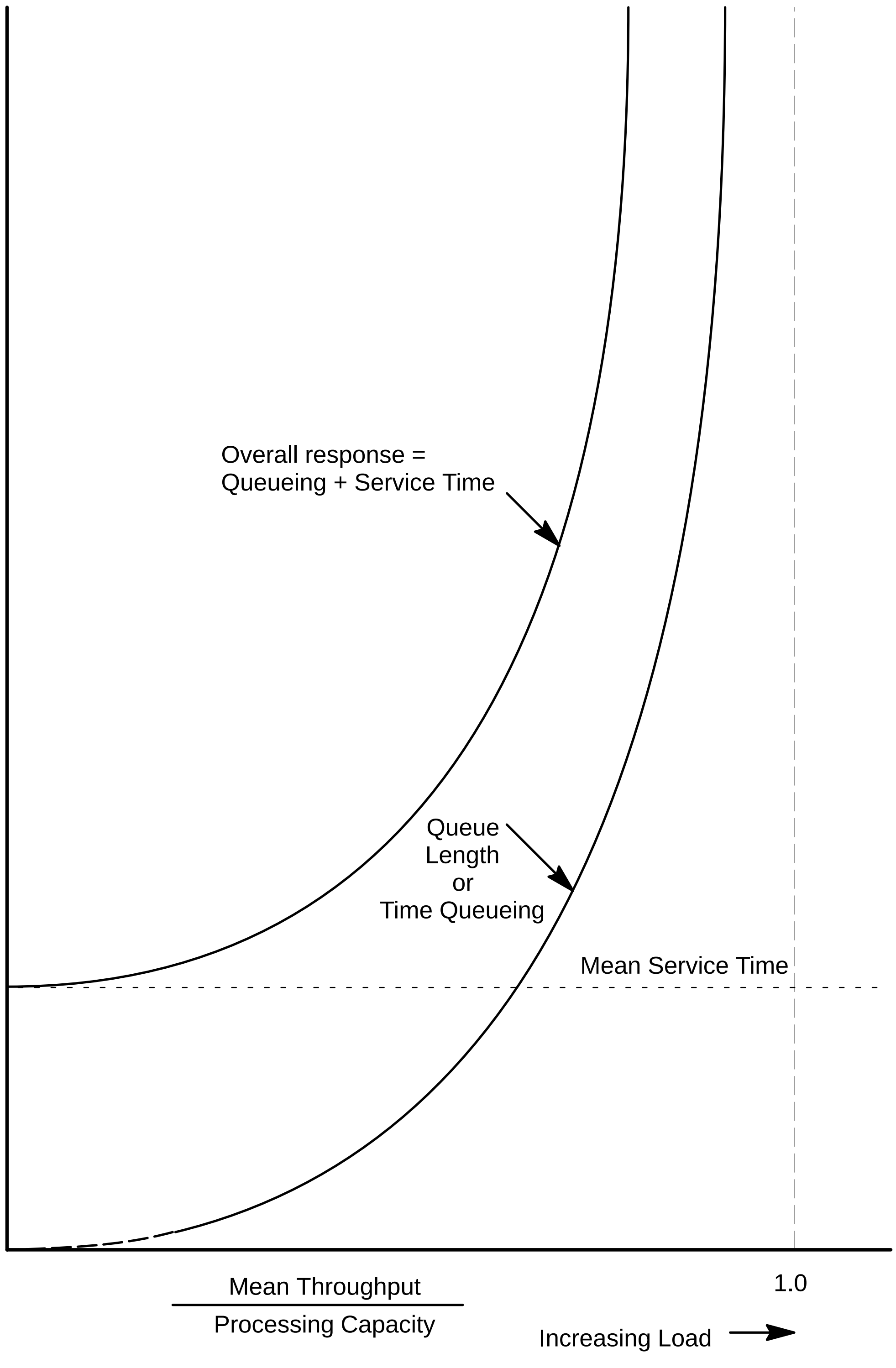
- Queue waiting time to be much longer that the service time.
- Queue lengths to dramatically rise with the risk of overflowing the queueing space allocated.

To ensure that both the waiting time and queue length are always short and containable, the processing resource available needs to be significantly greater that required simply to service the mean throughput of information (see Figure 2).

2.8 References for queueing

[1] G. Newell, *Application of Queuing Theory,* Berkley California University, Chapman-Hall, 1971.

[2] E. Page, *Queueing Theory in Operational Research*, Hull University, Butterworths, 1972.



The Impact of Queueing Time

3. The use of interrupts for critical features

3.1 Introduction

Interrupt mechanisms take many forms, even in the hardware design, and any attempt to summarise and make general recommendations will be incomplete. The most significant sub-division of this subject, is into External Interrupts associated with data input and Internal Interrupts designed to optimise and aid processing. Most of the discussion and recommendations are associated with external interrupts, however, many of the general recommendations are applicable to both.

Consider the following examples of external and internal sources of interrupts:

External

Pulses directly into the processor
Pulses via an external timers and counters
A to D conversion complete signal
Serial and parallel data receive/transmit ready
One-time level change (ie an accelerometer)

Internal

Timers and clocks
Software initiated resource "Exceptions"
Error "Traps"

Clearly, the interrupt handling techniques that are suitable for one type of source, may not be suitable for all, however, some broad principles can be established and a variety of general and specific recommendations given.

3.2 Potential sources of errors

(a) Noise and unplanned increases in input data rate can saturate and lock-up all the CPU time, causing loss or corruption of data, inhibiting other parts of system from executing failure modes, fall-back operation or the normal control operations.

(b) Some types of interrupts create effectively an infinite number of paths through code, reducing confidence that all possibilities have been catered for in the design or that they can be adequately tested in a realistic timescale.

(c) The software design, and choice of priorities, may not cater for all circumstances of asynchronous data access, causing chaos within the processor, leading to loss and corruption of data.

(d) Several processes may compete for common resources, such as dynamic allocation of data space or use of the stack, associated with interrupts, causing corruption of data and leading to "deadlock" or "livelock".

(e) Processor hardware timing is not always consistent or correctly documented, this may have most impact on the very time critical interrupt routines.

(f) There is potential for conflict between priority related to one task's real-time response, or data capture requirements and priority associated with another task's safety criticality. For example, interrupts from a high data rate, but low integrity source, such as communication messages from diagnostic aids, must not inhibit timely processing associated with a low data rate, high integrity requirement, such as vehicle braking, air-bags etc.

Even in a demanding real-time environment where the input data rates are high and the normal operating features can be actioned and demonstrated in a short period of time, the consequences of the types of error and failures identified above may take a very long time to be exposed.

3.3 Benefits of interrupts

Whilst most of the advantages of using interrupts are associated with optimisation of the hardware resources and achieving consistent response times, some benefits in robustness, and reliability, can also be attained by effective use of the internal interrupt mechanisms, which are sometimes called exceptions and traps.

For example, some micro-processor architectures have internal interrupts for arithmetic overflow and divide-by-zero [5]. Not only can these be used during development as debugging aids, but consideration should be given to incorporating them into on-board diagnostics for in-service traps and error logging.

3.4 Examples of techniques to improve robustness of interrupts

The overall objective of good interrupt routine design is to create an predicable and analysable software system. The detailed recommended techniques given below and are grouped in association with: Hardware; General Software Design; specifically to Interrupt Routine Design; and Data, however, the recommendations are all closely related.

Where the recommendations have impact on the five specific error scenarios identified previously in section 3.2, this is shown as:

"Can impact (a,b,c,d,e or f)"

3.4.1 Hardware related

1. Provide electronic filtering (analogue and/or digital) to control "slew-rate" and reduce the bandwidth of potential interrupts. Can impact (a).

2. Use edge triggering rather than level triggering, in conjunction with "slew-rate" control as above.
Can impact (a).

3. Provide intelligent I/O devices to buffer inputs in hardware and reduce the number of interrupting channels and the rates of interrupts. However, such devices can themselves be corrupted by noise and care is required to ensure that the noise problem is not transposed from a software into an electronics form.
Can impact (a,b & f).

4. Limit the use of interrupts. For example, to one level only. Can impact (a-f).

5. Use Test-and-Set instructions or a signalling mechanism, such as Dekker/Dijskra/Lamports Semaphores, to protect and mark as "in-use" any common resources.

Can impact (c&d) [6, 8].

6. Use a memory protection and User/Supervisor mechanism as part of partitioning the software in association with Controllability levels.
Can impact (b&c).

7. Use a Watchdog Timer and /or a Budget Timer to free lock-up situations. Can impact (b-e).

8. Use an independent observation technique such as a separate monitor processor with diverse software to detect and react to a lock-up.
Can impact (b-e).

3.4.2 Relating to general software design

1. Design in strong separation, partitioning and modularity between interrupt routines, application program and data from the outset [2]. Can impact (b-d).

2. Ensure there is a reserve of processing capacity and CPU time under full-load conditions. Use a modelling facility of a CASE tool if appropriate.
Can impact (d,e & f).

3. Control the use of the Enable and Disable interrupt instructions, and use only in matched pairs. Excessive use of these instructions introduces complexity and the potential for data corruption and lock-up; particularly in failure modes and abnormal situations.
Can impact (c&f) [1].

4. Ensure that the watchdog is serviced in a way that will trap the maximum number and type of lock-up situations. Can impact (c-f).

5. Disable interrupts during power-up initialisation. Can impact (a & b).

6. Tie-off and mask-out un-used interrupt vectors when not configured in the system and when Failure Management has determined a fault. Un-used interrupts should point to an error handling routine. Can impact (a).

7. Take account of the system/processor suppliers recommendations for multi-context software design, such as requirements to save and restore co-processor / floating-point working environments, etc. Some multiple level systems disable lower priorities, some do not, etc.
Can impact (a-e).

8. Group the interrupt routines and resource allocation tasks into a self-contained "Kernel". Can impact (a-e)

3.4.3 Relating specifically to interrupt routine design

1. Design the interrupt routine to match the requirements of the source and type of interrupting hardware.
Can impact (a-f).

2. Assign the internal clock or Failure Management routine to the highest priority of interrupt level.
Can impact (a-f).

3. Minimise processing from within interrupt routines.
Can impact (a,c & f).

4. Avoid re-entrancy and recursion to interrupt routines. Use semaphores or flags to indicate if an interrupt occurs while servicing earlier or multiple interrupts. Can impact (b-d & f).

5. Validate all data taken in by interrupt. The extent and sophistication of run-time validation techniques is dependent on the criticality of the application and the type and source of interrupts. Can impact (a&c).

6. Incorporate software error-trap interrupts such as overflow, divide-by-zero, etc. into on-board diagnostics. The sources of error can be in remote tools, such as compilers, linkers, etc. as well as in the application code, so this has impact on the overall software quality.

3.4.4 Relating to data structures and access

1. Ensure consistent data-sets for background processes as part of the modular design [6].

Write to a system variable from a single module or context in a multi-context system.
Can impact (c & d).

2. Where feasible, use independent stacks for program execution control and related data activity.
Can impact (b-d).

3. Ensure the stack length is set to the theoretical maximum design requirement to cater for the absolute worst case combination of execution activity.
(See Timing Constraints and Modelling)
Can impact (a-d).

4. When defining scratch workspace, allocate fixed dedicated storage in preference to dynamic storage for each interrupt routine, or each execution context.
Can impact (c&d).

5. Control dynamic memory "leaks". Check, and re-set if necessary, the stack pointer during run-time where appropriate. However, it should be recognised that reliance of this technique, particularly during development,without an appropriate diagnostic could mask the source of an error and only deal with the detected symptoms. Can impact (b-c).

3.5 Associating the recommendations for interrupt design with potential sources of error

Table 1 that follows summarizes the effectiveness of each of the recommendations against the five sources of error identified in the introduction.

3.6 Associating the recommendations for interrupt design with controllability levels

3.6.1 The rationale

The minimum requirement for a minor commercial, and non-safety related real-time/engineering application in a "friendly environment" has been assumed, to determine the lowest resource combination that could be viable. 7 out of the 27 recommendations have been selected to meet this criteria. Other business considerations may dictate a more robust design.

Conversely, for the highest level of controllability anticipated there would appear to be no grounds for omitting any of the 27 recommendations. This is compatible with a policy of a high level of both redundancy and diversity.

Where the recommendations has been identified as "progressive" (%), the symbols "R+%" and

Table 1 Recommendations against sources of errors

			Potential Sources of Error				
Ref.	Title	(a)	(b)	(c)	(d)	(e)	(f)
		Noise	Infin.	Chaos	Locks	Time	Prior.
1.1	Filter ins	HI%					
1.2	Edge trig	I					
1.3	Buffer ins	HI%	I%				I%
1.4	Limit ints	I	I	I	HI	I	HI
1.5	Test-and-set			HI	HI		
1.6	Mem protect		I	I			
1.7	Watchdog		HI	I	HI		I
1.8	Monitor CPU		HI	HI	HI		HI
2.1	Partitions		HI%	I%	I%		
2.2	Reserves				I%	I%	HI%
2.3	Disables			I%	I%	I%	I%
2.4	W/D service			I%	I%	I%	I%
2.5	Power-up		I	I			
2.6	Masks	HI					I
2.7	Suppliers recs			I	I	HI	
2.8	Kernel	I	I	I			
3.1	Specifics	I				I	I
3.2	Clock level	I	I	HI	I		HI
3.3	Minimise	I%		I%	I%	I%	I%
3.4	Re-entrancy		I	I	I		I
3.5	Validate	I%		I%			
3.6	Traps						
4.1	Data-sets			I	HI		
4.2	Data stack		HI	HI	HI		
4.3	Stack size	HI	HI	HI	HI		
4.4	Scratch mem			HI	HI		
4.5	Stack reset		I	I	I		
Design experience		HI%	HI%	HI%	HI%	HI%	HI%

Key:

I has Impact
HI has High Impact
% Progressively increased application of this technique is possible and has pro-rata benefits.

"HR+%" have been used in the following table of recommendations to indicate increasing application as the controllability level increases.

Table 2 that follows is offered only as an example of relationships. Other factors influencing the design could include: complexity of the application; availability of the electronic mechanisms; equivalent levels of redundancy and diversity planned at each of the controllability levels; business judgement; etc.

3.7 Recommended testing and validation requirements

1. Conduct an independent design review using, for example, the aforementioned techniques and recommendations as a check-list.

2. Stress test by overloading with data, transients and noise into interrupting channels.

3. Repeat the above during a power-up/down.

4. Check recovery mechanisms by, for example, forcing Watchdog timer resets, etc.

5. Stress test during sensor/actuator failure.

6. Repeat stress tests in extremes of environmental and EMC conditions.

3.8 Summary

Interrupt routines can be designed to ensure that inherent and known weaknesses are eliminated or minimised. This can be achieved though the design of an predictable and analyseable software system. Many complimentary and overlapping techniques exist. Using the principles of redundancy and diversity, these techniques can be layered to produce levels of robustness associated with the categories of Controllability.

Even within the discipline of software engineering, the design of interrupt routines and a real-time kernel requires very specialised knowledge. It is recommended that only personnel with previous experience of this activity are employed to implement and modify these routines for controllability levels 3 to 5.

3.9 References for timing constraints and interrupts

[1] J. McDermid, *The Software Engineering Reference Book*, Butterworth Heinemann, 1993.

Table 2 Recommendations against controllability levels

Ref.	Title	Nuis.	Distr.	Debil.	Diff.	Uncont.
1.1	Filter ins	R	R+%	HR+%	HR+%	HR+%
1.2	Edge trig		R	R	R	R
1.3	Buffer ins			R	R+%	HR+%
1.4	Limit ints				R	HR
1.5	Test-and-set				R	R
1.6	Mem protect				R	R
1.7	Watchdog	R	HR	HR	HR	HR
1.8	Monitor CPU			R	HR	
2.1	Partitions			R	R+%	HR+%
2.2	Reserves		R	R+%	HR+%	HR+%
2.3	Disables		R	R+%	HR+%	HR+%
2.4	W/D service	R	R+%	HR+%	HR+%	HR+%
2.5	Power-up	R	HR	HR	HR	HR
2.6	Masks		R+%	R+%	HR+%	HR+%
2.7	Suppliers recs	R	HR	HR	HR	HR
2.8	Kernel			R	R	HR
3.1	Specifics	R	R+%	R+%	R+%	HR+%
3.2	Clock level			R	R	HR
3.3	Minimise			R	R+%	HR+%
3.4	Re-entrancy		R	R	HR	HR
3.5	Validate			R	R+%	HR+%
3.6	Traps				R	R
4.1	Data-sets			R	HR	HR
4.2	Data stack		R	R	HR	HR
4.3	Stack size	R	R	HR	HR	HR
4.4	Scratch mem			R	R+%	HR+%
4.5	Stack reset					R
Design experience		R	R	HR	HR	HR

Key:

R	Recommended
HR	Highly Recommended
+%	Progressively increased application of this technique is possible and is recommended

[2] J. Cooling, *Software Design For Real-Time Systems*, Chapman and Hall, 1991.

[3] B. Wichmann, *Software in Safety Related Systems*, British Computer Society and Wiley, 1992.

[4] E.Klugmain, *Micro-processor System Design*, Prentice Hall, 1977.

[5] D. Hall, *Micro-processors and Interfacing*, McGraw Hill, 1992.

[6] D.Whiddett, *Concurrent Programming For Software Engineers*, Ellis Horwood, 1987.

[7] K. Apt and E. Olderog, *Verification of Sequential and Concurrent Programmes*, Springer-Verlag, New York, 1991.

[8] M. Ben-Ari, *Principles of Concurrent Programming*, Prentice Hall, 1982.

[9] C. Lien and C. Yang, "Specification and Quality Assurance of Timing Constraints in Real-time Systems Developments", *Software Practice and Experience Journal*, Wiley, November 1992.

[10] J. Craine and G. Martin, *Microprocessors in Engineering Science*, Addison Wesley, 1985.

[11] A. Burns and A. Welling, *Real-time Systems and their Programming Languages*, Addison Wesley, 1990.

4. Real-number arithmetic and floating point

4.1 Introduction

In most vehicle control systems applications the fixed point arithmetic capability of a 16-bit processor gives more than adequate accuracy for real number arithmetic and use of either hardware or software Floating Point (FP) is not contemplated. However, as the complexity of the arithmetic computations increases rounding errors rapidly become significant.

In real-time systems where response times are important generalised software FP routines are unlikely to offer a viable solution and the addition of a hardware floating point begins to look attractive. The hardware may be either a "co-processor" or "on-chip" so the pros and cons of each are considered.

In some industrial sectors, notably the military, nuclear and aerospace sectors, failures have been witnessed and concerns has been expressed [1]. At least one major supplier of microcontrollers for embedded applications does not recommend, nor is planning to offer, hardware floating point. Conversely, hardware floating point is used successfully in many non-real-time non-critical applications so an analysis of the root causes is necessary so that guidance can be given if and where appropriate.

4.2 Potential concerns

Three potential concerns have been identified:

1. There are a number of difficulties with verifying that programs using real-number arithmetic are free from overflow, divide-by-zero, or errors accumulated from the effects of rounding. Also, small errors in highly "ill-conditioned" equations can produce large errors in solutions. These difficulties may be reduced by FP, but they cannot solved entirely. Some of the issues affecting the use of floating point are evident from an examination of the IEEE Standard for Binary Floating-Point Numbers [2]. Numbers can be too large or too small to be represented in the usual normalised form and are know as NaNs or "Not-a-Number". NaN errors are usually flagged as errors in on-chip FP implementations. This may be a factor in the choice of a processor.

2. Potential for corruption by electro-magnetic interference is also given as reasons for avoiding FP in some critical applications. This appears to be true even where the same circuit technology is used for both CPU and FP logic. The explanation of this rests in an understanding of the different failure modes of the CPU and FP:

   - If the arithmetic circuits built into the CPU suffer either temporary or permanent corruption, then program control is usually lost within fractions of a seconds, as add and subtract instructions are used in most transfer of control instructions. The effect is that the processor locks-up and trips the watch-dog timer long before any arithmetic error associated with applications reaches an output device.

   - Conversely, if separate circuits for FP arithmetic are used, such as a co-processor, then these may be subjected to either temporary or permanent corruption that may not at the same time corrupt the CPU. These errors may will not be detectable and can be passed on as corrupt outputs.

3. Validation of a new (or modified) hardware floating point processor is no small undertaking. Although it is now accepted that no complex software system cam ever be totally and exhaustively validated, even checking every floating-point combination of: add, subtract, multiply & divide of two numbers over the full range would take over 1000 years on most processors! Validation routines used normally test just samples of boundary and "worst-case" combinations, and even then, take many days to complete. There are documented cases of some of the major processor suppliers having to implement design changes to correct for "pattern-sensitive" errors found well into production [3].

These and other concerns, together with the five levels of FP validation are discussed by Wichmann [4]. The five levels correspond to the generally accepted software integrity levels incorporated into the IEC standard and can be related to MISRA’s controllability levels.

4.3 General recommendations

1. Ensure that the three types of concerns described above are understood.

2. Recommend the use of scaled integers and fixed point arithmetic where possible.

3. Extend the normal fixed-word lengths where necessary by double precision. Round and truncate before output.

4. Use partial software FP for specific variables where (2) and (3) are inadequate.

5. Range clip results from both fixed and FP calculations.

6. Use the overflow information for failure management and diagnostics.

7. Pay special attention to the "complexity factor" associated with very large and very small real-numbers, even if hardware FP is used.

4.4 Recommendations for hardware floating point use

1. Follow Wichmann’s recommendations for FP validation.

2. Seek evidence of validation form suppliers.

3. Recommend that "fit-for-purpose" assurances are sought from the hardware suppliers for specific applications and environments.

4. EMC test at the earliest opportunity.

5. Use on-chip FP implementations rather than co-processors.

6. Ensure that the suppliers recommendations for save and restore of the co-processor environment are followed in the design of multi-tasking/multi-context interrupt routines (see Use of Interrupts).

7. Use all FP unit error flags/messages to invoke failure management procedures.

8. Interweave a diagnostics routine within the application to compare sample fixed and floating point calculations. Invoke failure management when differences are detected.

4.5 References for floating point

[1] Ministry of Defence (UK) Interim Standard 00-55, April 1991.

[2] IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE std 754-1985.

[3] Du Croz (NAG), "Floating Point Validation", Chapter 7 of *Software Quality and Reliability*, edited by D. Ince, Chapman-Hall, 1991.

[4] B.A. Wichmann, "A Note on the Use of Floating Point in Critical Systems", *The Computer Journal*, **35(1)**, 1992.

5. Microcontroller hardware conformance considerations

5.1 Introduction

This section identifies the validation requirements required for software-based systems associated with environmental considerations. It will also consider the issues associated with the design and testing of fail-safe mechanisms such as watchdogs.

5.2 Environmental issues

5.2.1 Environmental stresses and tests

Environmental stresses break down into the following categories:

- Temperature
- Humidity
- Contaminants (salt spray, grease, petrol, dust, etc.)
- Vibration, shock and bump
- Supply voltage
- Transients
- Quiescent current
- EMC immunity and emissions
- Electrostatic discharge (powered and unpowered)
- Endurance (monitored temperature cycling)
- Abuse.

Each vehicle manufacturer has their own variations on the test procedure to test against any of the above hazards, although many are tending to standardise on ISO definitions and procedures. The MIRA Guidelines [5] consider these hazards with particular reference to an after-market product which may be fitted to a variety of vehicles. They should be used to form the basis of a test regime for an after-market product or any generic product that may be fitted to a range of vehicles.

Tests are usually performed on individual components connected to a dummy harness and test box. Testing is also performed on the whole vehicle. In addition, prior to launch a new vehicle is subjected to a tough programme of summer and winter tests. EMC tests have traditionally been performed on a whole vehicle, but component level emc tests are being used increasingly which may be performed in a screened room rather than an anechoic chamber.

5.2.2 Relationship between testing and software

In their ability to withstand most of the above test regimes, microcontroller-based systems are little different from any other electronics. Their main difference is their reliance on running through a sequence of instructions to give correct functionality. A failure in any instruction can disrupt the sequence with consequences that are difficult to predict. However, CMOS digital technology is fairly tolerant of noise in itself, and with an on-board ROM, quite extreme conditions are required to cause the processor to mis-read its instructions. A much more common event (See note 1) is mis-reading information from the RAM, but this can have the same result if the RAM itself is used as a stack to contain return program addresses. Information in RAM is particularly vulnerable because it can be corrupted at any time while it is being stored.

In order to test for tolerance of the environment, it is necessary to test the electronic module both in vehicle and with a dummy harness connected to a test box. Some tests can only be carried out on the sample at component level, and some tests take so long that the component can only be put through an automatic test sequence that repeats itself whilst an environmental stress is applied. The basic problem is that the unit under test (UUT) is likely to be executing a small subset of its program whilst the environmental stress is actually being applied.

5.2.3 Whole vehicle tests versus component level tests

Any weaknesses in a microcontroller program, such as inadequate software filtering of certain inputs, may not come to light unless those inputs are exercised during a test. This is particularly the case with the full-chamber EMC test in which the whole vehicle’s performance is checked whilst it is being illuminated with a powerful electromagnetic field. The nature of the test is such that nobody can be in the vehicle when it is being tested, so the tests are normally carried out with the vehicle in standard set ups without any of the controls being changed during the test. The economics and availability of test facilities for running these whole-vehicle test regimes is such that only for highly safety-critical systems such as ABS is it usual to install the means to provide actuation of controls (in this case operation of the brakes) to exercise the software through its normal operating cycle.

The formal EMC whole-vehicle sign-off test, in particular, has shortcomings with respect to many electronic systems in that their reaction to changing inputs may not be tested adequately. In this respect, component level testing, usually performed by the component supplier rather than car maker, is particularly important. It is necessary to ensure that the microcontroller does indeed read correctly every input over its normal range of states whilst the stress is applied. Similarly every output needs to be tested over its full range with the stress applied. Bulk current injection is a particularly useful technique as it is relatively low cost, and enables component suppliers to perform these tests. Car makers need to specify component level emc tests to be performed on all modules containing software, in which all input states and all output states are tested for correct operation under stress. This is particularly easy for modules with an ISO 9141 interface. How injected currents relate to radiated field strength is a matter for on-going research [3], but in the absence of industry standards, component suppliers and car makers should agree sensible pass levels for these bulk current injection tests, based on the results from whole vehicle radiated tests if available.

5.3 Testing failure management strategies

5.3.1 Failure management strategies

Even an electronic module that has passed all its validation tests may occasionally fail, due to exceptional events such as lightning, welding or arcing nearby or a combination or sequence of unexpected events. Research into the effects of lightning [2] indicate that fields much higher than those used during EMC tests can occur for very short durations. The effect of EMC is most noticeable on analogue circuitry.

The most likely effect on digital circuitry [4] is to cause corruption of the RAM data, and this can lead to corruption of the stack, resulting in a program crash. The most likely course of events is a single bit corruption which results in the program counter pointing to the wrong address when returning from a subroutine or interrupt. Typically the microcontroller will fall out of synchronisation with its code, so that it starts executing the second or third fetches from ROM of an instruction as if it were the first. The number of different routes through the program thereafter are such that it is not possible to predict behaviour. A watchdog needs to be fitted which will reset the microcontroller if it is not behaving properly. Correct behaviour is ensured through writing correctly to a watchdog register within a defined time. Care needs to be taken to ensure reliable operation.

5.3.2 Recovery strategies

Even if the microcontroller does fall out of synchronisation with its code (crash), there are various techniques that may be employed within the program to limit the time before the microcontroller can recover control. Microcontroller manufacturers may be able to provide guidance on their specific products, and where available these should be followed. Examples of techniques that can improve reliability are given below:

1. Fill all unexecuted code space in the ROM with single fetch instructions leading to an instruction that will put the micro into a defined state (such as a software reset) if ever executed. This will cause it to resynchronise and put itself into a defined state as soon as possible.

2. Scatter single fetch instructions throughout the program at regular intervals to re-synchronise the micro if it falls out of synchronisation in the code space.

3. Do not leave executable code in RAM too long without checking its integrity (including return addresses from sub-routines).

4. Around each subroutine, put a software reset instruction.

5. Ensure that the watchdog is kicked from only one point in the program which is part of the function that performs the core functionality. It should not in general be kicked from a subroutine, and should never be kicked from an interrupt routine.

6. The instructions first executed after a power-on-reset are particularly vulnerable as the microcontroller's oscillator may not have settled down, the supply voltage may be rather low and the watchdog will probably not have been enabled. One microcontroller manufacture recommends making the first twenty instructions NOP (no operations). Another defensive technique is to ensure that the reset pulse is long enough.

7. It should normally be impossible to stop the watchdog once it has been enabled (but see the next paragraph).

5.3.3 Permanently powered microcontrollers

Microcontrollers that are permanently powered from the battery (to operate vehicle security, etc.) have a further design complication. They normally require the use of a low power mode (called stop, standby, halt, idle, etc.) when the vehicle is not in use. It is usual to have to stop the on-board watchdog in order to drop into a low power mode. The instructions to do this must be carefully protected in ROM from an errant micro accidentally stopping its watchdog and going to sleep. Such a micro would be particularly difficult to re-start as the battery would normally need disconnection to provide a power-on-reset. A number of strategies are possible to overcome this. Where a watchdog can indeed be stopped from within the program, the program needs to be examined carefully to see if it is possible for the micro to find accidentally any "stop watchdog" instructions when running out of synchronisation with its program. If any such instances are discovered, they should be protected by preceding them with the following:

```
a branch to the stop instruction    BRA STOP_INST
a single byte instructions          NOP
reset.                              RESET
the stop instruction                STOP_INST STOP
```

The action of the watchdog, and particularly any means to stop it, should be examined seriously in the hazard analysis that should normally support the development of software which has to meet high reliability criteria [4].

5.3.4 Watchdog design

Many microcontrollers, especially for automotive use, have a watchdog built into the hardware of the chip which can be enabled under software control.

Ideally the watchdog should have an independent oscillator which must be guaranteed to start and to continue to run at a lower supply voltage than the main micro. This should be verified by test. Watchdogs that share the same oscillator as the micro (such as on-board watchdogs) have an inherent failure mode in that a failure of the master (common) oscillator will not put the microcontroller into a defined state if the master oscillator stops running. This hazard should be considered in the design of the hardware such as when doing a hardware FMEA on the microcontroller.

For systems which require a high level of integrity (see note 2), an independent watchdog may be required that will put the hardware into a safe state if there is a failure of the main micro or its associated support systems (such as power supply, oscillator & reset circuit). Having built an acceptable level of protection into the software, it is essential to test the behaviour of the watchdog circuit.

5.3.5 Testing watchdog circuits

The action of the watchdog should be tested thoroughly to ensure that it does indeed re-initialise the circuit in such a way that full control is resumed quickly enough to cause as little inconvenience as possible and certainly not to put the vehicle at any kind of risk (see note 2). If lengthy operations such as testing and sizing of ram are required after a power-on-reset, these may need to be skipped at a watchdog reset. The essential is to resume control as quickly as possible. Re-initialisation must ensure that all functions operate again correctly. For example, there should be no flag bits left in ram from before the crash that cause certain functions to behave incorrectly, and this needs to be demonstrated by test.

Several methods are suggested to test the watchdog:

Method 1

Patch code with infinite loops. At various points in the program, branches to infinite loops should be inserted. These should be inserted into both the main body of the code and interrupt routines. This method will test the integrity of the watchdog without any change to the hardware, but lacks any element of randomness.

Method 2

Modify the software and hardware to make use of an unused input. Patch the program to branch to an infinite loop whenever this input is activated. This method provides a more realistic test, especially as the stimulus can be applied randomly, but may be difficult when the product is at a late stage of development if there are no unused inputs.

Method 3

(For devices receiving data from a serial data bus): Patch the code so that if a certain message is received, the processor goes into an infinite loop.

Method 4

Apply a severe electromagnetic disturbance to the system (more severe than any of the acceptance tests described in section 1). A note on how this may be done is in note 2.

The acceptance criterion of all these tests must be that full control is resumed within an acceptable time period.

5.4 References

[1] Hazard Analysis: DEF Stan 00-56/1, available from the Directorate of Standardisation, Kentigern House, 65 Brown Street, Glasgow, G2 8EX.

[2] EMC Reference: SMMT Guidelines for Electromagnetic Compatibility, available from SMMT, Forbes House, Halkin Street, London SW1.

[3] R. Ball, P. Jennings, and P. Lever, "EMC Testing at Rover Cars" in *Engineering Science and Education Journal*, December 1992.

[4] Laurent Perier, "Designing with Microcontrollers in Noisy Environments" in *Electronic Product Design,* March 1993.

[5] T.B. Beadman, *MIRA Quality Assurance Guidelines for Electronic Alarms/Immobilisers*, available from MIRA, (01203) 348541.

5.5 Notes

1. Information (unpublished) supplied to the author by Texas Instruments based on work that they had done into failures of microcontroller-based metering systems such as water, gas & electricity metering in which the monitoring device is active for very long periods and corruption of data can result in financial loss.

2. An example of a simple tester is a device that consists of the following:

Metal box with a spark plug screwed in
Standard car induction coil
Make and break circuit (such as a 555 timer driving a MOSFET or relay at 0.5 sec intervals)
12 Volt power supply

Copper cored HT lead between coil and spark plug.

If the copper HT lead is held near enough to a microcontroller, it is more or less guaranteed to make it misbehave. The circuit should be zapped repeatedly and correct behaviour demonstrated between zaps, without any intervention.

6. Concerns with conformity of production

6.1 Semiconductor and motor industry rates of change

Microcontrollers will be purchased in volumes ranging typically from a few thousands to in excess of a million units over a time period of typically five years between major redesigns, but production runs in excess of twenty years are not unknown. Thereafter there will be a call for service replacement parts over the following ten years. This timescale is long in terms of the semiconductor industry, for which the time interval between major changes of process is about three years. In a motor industry production run of over ten years, many of the silicon processes will inevitably change (note 1). As an example, one should consider the changes in personal computers that have taken place during the production of a single popular model. Maintaining the integrity of the product over its effective lifetime (from pre-production to end of support as a service item) is by no means a trivial task. Indeed, because of the rapid growth in car electronics in the previous decade, service support is an issue that is only beginning to emerge.

6.2 Testing masked microcontrollers

Masked microcontrollers are a class of semiconductor that pose particular difficulties to maintaining integrity because of the way in which they are tested. They fall into a class of devices that lies between a custom integrated circuit (ASIC) and an off-the-shelf item in that they are customised to the extent that code is embedded into their ROM and this defines their functionality. They are tested, however, by generic test patterns independent of the actual instructions in the ROM. ASICS, on the other hand, are tested with a fully custom test pattern set that the designer supplies and which reflect the actual use to which the device is put. A process change on a microcontroller may therefore be approved because the device operates satisfactorily with its generic test patterns, but it may fail in application because the actual sequence of instructions used in a particular program gives causes a malfunction. This is not a hypothetical case, and has actually caused a much more serious concern than any errors in software in development.

6.3 Process changes affecting microcontrollers

The builders of electronic modules using microcontrollers are advised to take the following steps with their semiconductor suppliers:

1. Make sure that the microcontroller manufacturer notifies the customer of all process changes affecting their products. If in doubt, ask for more details. Semiconductor suppliers vary in their performance with respect to notifying their customers of changes, and it is probable that the ones who do better at informing their customers have better control over the process change because of its potential visibility to customers.

2. Request samples of any part that is subject to change, and carry out at least the following tests:

    - High temperature
    - Low temperature
    - Ramp between temperature extremes
    - High supply voltage
    - Low supply voltage
    - EMC emissions (comparing new version with old)

If the device stops an oscillator at certain times, check that it always re-starts.

If the changes affect the inputs or outputs, then transient tests should be performed.

If the changes affect packaging then either test data should be requested or full environmental tests should be performed.

When the semiconductor manufacturer sends out a process change notification, a failure to respond implies acceptance.

6.4 Past examples of concerns with microcontrollers

1. A customer’s particular sequence of instructions generated a spike inside the chip which corrupted data being accessed from the ram, causing the processor to make an incorrect jump.

2. Some mask sets of microcontrollers can fail to recover their state correctly when returning from servicing an interrupt in certain conditions. In particular, certain flags are not correctly restored.

6.5 Batch (lot) traceability

If a concern does arise in volume production, it is essential to contain the situation by being able to identify the product at risk efficiently. The electronic module manufacturer should code all product such that the component parts can be tracked back to their respective delivery notes. These should be able to provide sufficient information to the semiconductor manufacturer to be able to trace the parts through its system. Special care should be taken if semiconductors are purchased through a distributor that the integrity of this chain exists back to the manufacturer.

This information is essential for two purposes:

1. Analysis of the root cause of the problem

2. Containment by identifying product at risk

A requirement of ISO 9000 is to be able to trace all materials through the production process. Manufacturers of electronic modules for the motor industry should ensure that their batch traceability meets this standard, and that the suppliers of their integrated circuits equally meet these standards.

6.6 Notes and references

1. Author’s discussions with multinational consumer electronics manufacturers with large factories in S. Wales (Hitachi, Sony, Panasonic).

2. Peter Neuman, "Inside Risks", in *Communications of the ACM,* 1989.