O.R. REBREATHER SAFETY CASE

FMECA Volume 5: Firmware and Software

DOCUMENT NUMBER: FMECA_OR_V5_080812.doc

CONTRIBUTORS: **Dr. Alex Deas, Dr. Sergei Malyutin, Dr. Vladimir Komarov, Dr. Sergei Pyko, Dr. Vladimir Davidov**

DEPARTMENT: **Engineering**

LAST UPDATE: **12th August 2008**

REVISION: **A4**

APPROVALS	
_____AD_____ Project Manager	_____ _____ Date
_____SP_____ Firmware Subproject Manager	_____ _____ Date
_____SM_____ Software Subproject Manager	_____ _____ Date
_____BD_____ Verification Team Leader	_____ _____ Date
_____VK_____ Quality Officer	_____ _____ Date

Controlled Document ☐ Classified Document ☐ **Unclassified if clear.**

Revision History

Revision	Date	Description
A1	2nd June 2008	Former DV and failure documentation compiled and formatted to match CASS Scheme Template structure.
A2,3, 4	12th August 2008	15th June: Check lists from QP24 and NASA filled in and added. Edits, abbreviations section added. 5th Aug 2008: Independent reviews comments from a safety panel member addressed. 12th Aug 2008: Internal review and issue for audit approved

Table of Contents

1 PURPOSE AND SCOPE

This document is a top down Failure Mode Effect and Criticality Analysis of the first Open Revolution rebreather submission developed by Deep Life Ltd in respect of the firmware and software.

All references to "the system" refer to the electronics in the Open Revolution rebreather family developed by Deep Life Ltd.

References to "mandatory checks" refer to the pre-dive checks performed by that specific rebreather controller or rebreather monitor.

The scope of this document is a firmware and software FMECA under Deep Life Quality Procedure QP-20 (Safety Critical System) and QP-24 (Safety Critical Software).

2 TARGET AUDIENCE AND ABBREVIATIONS

The target audience is the Safety Review Committee for the project, and the external safety auditors. Efforts have also been made to open the review up to the widest peer review as a public process, to achieve the highest level of safety in this system.

Given the wider audience than is normal for this type of document, and that some independent members of the safety review panel have expertise in fields other than electronics or software[1], a special effort has been made to make it as accessible as possible despite the highly technical nature of the subject matter.

It is assumed that the reader is an educated person, interested in safety systems. Tools such as Wikipedia and Google allow the reader to explore new concepts off-line in a way previously only dreamed of. Where new concepts are met, this approach is encouraged by the use of hyperlinks in the softcopy of this document.

It is important that the desire for open access does not cloud the issues for the specialist, who will want detailed answers to their searching questions regarding the safety design and lifecyle of the system. This necessitates a degree of precision that demands the use of specific technical terms.

In general, the use of a simple word has been used in preference to a medical term, where that term might not be well known. For example the word *eye* instead of *ocular* in some contexts, or *skin* instead of *membrane* in others, even if the result sounds a little clumsy. Otherwise, the abbreviations used here are those in widespread use in electronic and software engineering, a list of which is below:

ADC – Analog-Digital Converter.

ASIC - Application-Specific Integrated Circuit.

CCR - Closed Circuit Rebreather

CPU – Central Processor Unit.

CRC - Cyclic Redundancy Check is a special function that is used as a checksum to detect alteration or corruption of data during transmission or storage.

DAC – Digital-Analog Converter.

EDA – Electronic Design Automation is the category of tools for designing and producing electronic systems.

[1] Good safety practice is to include independent users, technicians and managers on the safety committee, as well as experts in the field, to ensure the ergonomic aspects of the safety case are properly considered. This has been done for this project, with a roughly equal split between specialists and independent members.

EMC - Electromagnetic Compatibility: how much interference electronics generates.

EMS – Electromagnetic Susceptibility: how sensitive an electronic device is to interference.

Firmware – For a logic array, it is the circuit configuration of that array. It is often used for computer programs despite the word *software* being available which is both better understood and satisfactory. As both logic arrays and microcomputers are used in the application, to avoid confusion the word *firmware* here will refer to the logic array configuration and the word *software* will be used to describe the program that the microcomputer runs.

FMECA - Failure Modes, Effects and Criticality Analysis is a widely used technique to analyse possible failure modes of complex systems and their impact on the safety functionality. This document takes a broad approach in including the methods that give rise to the failures: failure to apply appropriate methods, or misapplication of methods.

FPGA – a Field-Programmable Gate Array is a chip containing programmable logic gates or components which are configured to perform basic logical functions.

Hypoxia[2] – Insufficiency of oxygen in the breathing gas

Hyperoxia – hazardously excessive level of oxygen in the breathing gas

Hypercapnia –hazardously excessive level of carbon dioxide in the breathing gas

MCU - Microcontroller Unit is a microcomputer which controlled by internal firmware.

PCB – Printed Circuit Board.

PFD – Peripheral Field Device, sometimes referred to mistakenly as a Head Up Display, where the PFD refers to a visual warning on the periphery or edge of the visual sphere.

PP02 – Partial Pressure of Oxygen

PPCO2 – Partial Pressure of Carbon Dioxide

RAM – Random Accessed Memory, a dynamic memory storing temporal data.

ROM – Read-Only Memory is a computer memory which stores the firmware.

SIL – Safety Integrity Level, one of 4 levels of safety functions defined in the EN61508, where SIL 4 is the most onerous, and SIL 1 is the least.

SCR – Semi-closed Circuit Rebreather

]TTA - Time Triggered Architecture is a software architecture for safe distributed embedded real-time systems, where a specific time interval is allotted to each function. TTA can be applied at an inter-chip level, or within a single processor that does not interrupts.

UML - Unified Modeling Language, an object modelling and specification language used in software engineering.

3 SOURCE DATA

The failure modes listed in this document are drawn from numerous sources, of which the primary documents are:

1. Green Book documents describing the detailed specification and implementation for:
    a. Rebreather Base Unit,
    b. Umbilical Terminator,
    c. Topside Unit and Top Side Software.

[2] Use of hypercapnia instead of hypercapnea is used, as the international corruption of suchlike words has become the most popular. Similarly, mention is made of diluent instead of Make-up-gas. Otherwise British spelling is used throughout.

   d. Base Unit controller firmware, FPGA section (further referred to as "firmware")
   e. Base Unit controller firmware, MCU section (further referred to as "software")
2. The circuit diagrams and source code for all subsystems.
3. DV_OR_Power_080319.pdf, dealing with the power system reliability and power transition states
4. DV_OR_SaphionCells_080415.pdf, dealing with the suitability of the batteries for this application and providing a detailed characterisation of their charge and discharge characteristics, so the battery life can be predicted with an accuracy of +/-10% when fully charged, with the accuracy increasing to +/-2% tolerance as the batteries discharge.
5. DV_Umbilical_Terminator_PSUs_A3r_080422.pdf, dealing with the fail safety of the umbilical terminator power systems
6. DV_Safe_O2_dosing_080516.pdf, dealing with the fail-safety of the oxygen injectors in the event of a total loss of power or major controller failure
7. FMECA_OR_V6_080526.pdf, listing the top down failure modes against which the significance of different failure modes are assessed
8. FMECA_OR_V3_Elec_FMA_070404.pdf, describing the electronic systems and their provisions for safety critical applications
9. FMECA_OR_V2_Elec_MTBCF_070103.pdf, describing the reliability of the underlying computing platforms for the high SIL subsystems.
10. ORRB_UnifiedDataTransferProtocol_RevS_20080710.pdf describes the data transfers between all main functional units, including use of multiple channels and checksums.
11. TF_Rebreather_Environment_Emulator_080403.pdf describing a test fixture that is able to apply power brown-outs, power noise, power dropouts and emulate oxygen cell failures.
12. The formal rebreather verification tool, Revision 3.1, published on www.deeplife.co.uk/or_models.php. This is believed to be by far the most detailed tool available describing the environment and operation of a rebreather. It also provides a formal model of the primary rebreather fault modes.
13. DV_CO2_Sensing_060104.pdf describing the operation and limits of the PPCO2 sensing system.
14. DV reports on the O2 sensors, including:
    a. DV_O2_sensor_Teledyne_R22D_210910(ia).doc
    b. O2_Cell_Study_Release_070309.doc
    c. Teledyne cell failure report 061212.pdf
15. Log books from all project leaders.
16. UML Description of all software modules.

All DV reports listed above, and all FMECA reports, are available on the Deep Life Web Site: www.deeplife.co.uk/or_dv.php and www.deeplife.co.uk/or_fmeca.php respectively, with a capture date of 2nd June 2008. Circuit diagrams are published with the FMECA Vol 2.

Green books, log books and UML are confidential documents, but relevant key extracts are provided herein.

4 SIL COMPLIANCE OBJECTIVE

The objective of the rebreather part of the system is compliance with SIL 4 in accord with EN61508:2001. This requires a mean time between critical failure better than one billion hours, and a system availability of 100,000 hours subject to routine maintenance and preparation.

The SIL 4 objective has been concluded by applying the processes in EN61508 with the ALARP principle (As Low As Reasonably Practicable risk), in the context of a rebreather which is supplied as an Open Circuit replacement with more than 10,000 units in use.

The objective of the communication and monitoring parts of the system is SIL 2, in accord with EN61508:2004.

This FMECA fault list considers "plausible failures" as any failure with a probability greater than one in a billion hours of diving, multiplied by the number of faults listed, so the aggregate risk is less than 1 in 10^9.

5 SAFETY REQUIREMENTS

The safety critical requirements have been identified as:

1. Maintaining the PPO2 within the limits needed to support life. This comprises managing risks of hypoxia, central nervous system oxygen toxicity and pulmonary oxygen toxicity, regardless of metabolic rate, ascent or descent rate, or the availability or otherwise of supplied gases.
2. Keeping the loop PPCO2 and the diver's retained CO2 below the limits that would become a threat to the safety of the diver.
3. Mitigating and detecting floods that represent risk of drowning.
4. Provision of good communications to the surface and the bell for commercial divers.
5. Monitoring of poisons likely to be present in the loop, and avoidance of poisons from the equipment itself.
6. Mitigation of oxygen fire hazards.
7. Mitigation of thermal hazards.
8. Mitigation of pressure hazards (explosion, implosion, trans-membral injection, eye damage, noise, fire).
9. Mitigation of mechanical hazards, including pinch or failure to actuate hazards.
10. Mitigation of electrical failure hazards.
11. Mitigation of software failure hazards.
12. Mitigation of total electronics failure hazards.
13. Mitigation of lighting and visibility hazards.
14. Mitigation of depth reporting hazards in the presence of thrusters and ROVs.
15. Mitigation of general diving hazards including insufficient breathing gas.
16. Mitigation of equipment maintenance hazards that could affect the safe function of the equipment.
17. Mitigation of caustic cocktail and other toxic chemical hazards.
18. Management of Decompression risks

The causes of these failures are listed in FMECA Volume 6 of this series, and their interrelation is described by the Fault Tree Analysis in FMECA Volume 7.``

6 FIRMWARE FAILURE MODES, EFFECTS AND CRITICALITY ANALYSIS (FFMECA) OF ECCR

The firmware is the Verilog code from which are synthesised logic structures that are implemented in a FPGAs.

A detailed Green Book is available for the entire FPGA firmware. This describes the design and methods. These are to the same standard as Deep Life use for full custom ASICs, which have hundreds of millions of gates but which must work first time, due the multi-million dollar costs of deep submicron phase masks and the long lead times involved with ASIC fabrication and packaging. The extensive tools available for Verilog verification have been applied to this design, as is evident in the Green Books.

Dual redundancy is used with a calculated failure frequency. This is published as FMECA Volumes 2 and 3 of this series. This means that if the FPGA fails, there is no effect on the overall system safety or functionality, other than:

1. One of the two oxygen injectors will not be able to inject gas. This should not be material, but there is an increased risk of a failure of oxygen control due to sudden blockage of the second injector.
2. Half of the scrubber temperature sensors would be lost: this should not have an effect on the scrubber life computation as it uses primarily the amount of CO2 the diver is producing, but would reduce the accuracy of the scrubber health monitor.
3. Other non-critical sensors would be lost, including loop pressure.
4. One communication channel will be lost. Where the system is configured to use FPGA communications only, such as a diver using a single PFD in a SCUBA mode, then the diver will notice a complete loss of status lights. Where the user has configured these to operate in Stealth mode, the user would not be aware of the failure so it is essential in that situation that the hypoxia risk monitor is enabled.
5. The data packets to all other subsystems would show an absence of FPGA data, whereupon the FPGA would be interrogated and the FPGA control functions taken over by the microcontroller safety monitor as well as the state alerted to the diver and supervisor (for SSUBA).
6. For the purposes of this review, the situation is considered where there is a failure of all redundancy. Despite the calculated MTBCF of 2.8 billion hours, this case can occur when the batteries are discharged through a lack of attention on the part of the diver or technician prior to the dive. Where there is a power loss, the system signals the battery health to the diver throughout the discharge cycle, and ultimately fails with the injectors in a fail safe state. The loop shut off valve actuates, shutting the loop: this is a fail safe design at the cost of a substantial holding current. If no loop shut off valve is fitted, then the loop should still remain breathable for 30 minutes, but cannot be guaranteed to do so.
7. All logging is lost in the event of an FPGA failure, but logging up to the failure can be recovered: the FPGAs have extensive log storage associated with them.

7 SOFTWARE FAILURE MODES, EFFECTS AND CRITICALITY ANALYSIS (SFMECA) OF ECCR

The software forms the following roles in the operation of the system:

1. Monitoring the FPGA functions
2. Optimisation of loop parameters, within limits permitted by the FPGA

3. Taking over total loop control in the event of an FPGA failure
4. Signalling safety data to the diver and supervisor by the PFD, Topside Unit and related client software. That is those safety data displays use software.
5. Maintaining the PPO2 within the limits needed to support life in the event of an FPGA failure.

The failures can result from bugs in code, hardware failures in the microcontrollers (hard or soft), failures in clocking systems, failures in reset signals, failures in data transfers to and from memory as a result of jitter or other parameters, failure or corruption of memory. All these failures can be hard or soft, that is permanent or transient. MCUs have a considerably higher failure rate than FPGAs, due to the higher bandwidth and clock rate: the Shannon information bandwidth of the MCU is in fact much higher than for the FPGA once it has been configured.

7.1 Loss of safety data information displays

If a failure of the signalling software occurs, i.e. a part of software that provides monitoring and gives the signals, then there would be a loss of diver safety data and the dive should be aborted: dives should be planned so this can always be carried out safely. That signalling software has been assessed at SIL 0 (no SIL) to SIL 1. A total loss does not occur because critical data is provided also by the FPGA.

7.2 Loss of PFD functions

The PFD performs safety functions, including control of the automatic shut off valve, monitoring the diver's helmet, alert switch, respiration and communications. This is a SIL 2 to SIL 3 application, carried out in software on an ARM 7 processor in the PFD: that processor is completely separate from the processor in the rebreather controller.

Dual redundancy in the PFD is used for commercial diving, but for SCUBA diving where there is no helmet or supervisor, single instance is used as the application has a lower SIL level (SIL 1).

7.3 Loss of rebreather controller monitoring functions

The monitoring software that works within the rebreather in conjunction with the FPGA has been assessed at SIL 2 to SIL 3, forming a SIL 4 system with the FPGA. Loss of the monitoring function would reduce the safety integrity level of the system by one SIL level, i.e. from SIL 4 to SIL 3.

7.4 Failure of PPO2 Optimisation

There is a microcontroller that monitors the primary controller: that controller has the means to take over safety functions. It is also used to optimise safety functions carried out by the primary controller. However, the FPGA does not allow MCU software failures to take the whole unit out of operation.

Rebreather MCU software failure could result in the PPO2 optimisation function requested by the FPGA no longer being serviced. This would result in the PPO2 no longer being optimised: it would first excurse by the FPGA to a level of between 0.3 and 1.6 ATM, then to the set point with a 0.05 ATM limit either side in a bang-bang controller mode.

7.5 Failure of PPCO2 Monitoring

Rebreather MCU software failure could result in a total loss of PPCO2 monitoring. This should not be a life threatening situation but should result in the dive being aborted. Again, the FPGA does not allow MCU software failures to take the whole unit out of operation.

7.6 Failure of Flood Detection

Rebreather MCU software failure could result in a total loss of flood detection, though it is very unlikely: the function is replicated by the FPGA. Again, the FPGA does not allow MCU software failures to take the whole unit out of operation. This should not be a life threatening situation but should result in the dive proceeding with additional checks and care.

7.7 Failure of Respiratory Monitoring

Rebreather MCU software failure could result in a total loss of respiratory monitoring. This should not be a life threatening situation but if the diver is very deep then the diver should return to a shallower depth if on SCUBA (if on SSUBA, the supervisor should take over the monitoring function using the diver's audio channel).

In both cases, the dive should be able to proceed safely with additional checks and care.

However, the FPGA does not allow MCU software failures to take the whole unit out of operation.

7.8 Failure of Communications

The PFD MCU is involved in the provision of good communications to the surface and the bell for commercial divers. Failure of the PFD MCU software in both of the helmet digitising processors at the same time could result in a total loss of voice communications and the diver should return to the bell: it applies only to commercial diving.

The PFD MCU is a primary safety information display, the loss of which should result in an abort of the dive, and if there is no other monitoring available, a bail out.

There is no impact on the maintenance of the safe breathing loop.

7.9 Failure of Poison Monitoring

Rebreather MCU software failure could result in a total loss of poisons monitoring, particularly carbon monoxide and hydrocarbons. The dive could continue with additional care.

7.10 Failure of O2 Fire Mitigation

The software plays only a very marginal role in the O2 fire mitigation, by the provision of requests to open and close valves slowly during predive checks on the SCUBA version of the equipment.

7.11 Failure of loop thermal monitoring

Rebreather MCU software failure could result in a total loss of loop thermal monitoring. This should not affect the dive, which may proceed with special care to check the thermal comfort of the loop, and loop humidity.

7.12 Failure of electronics failure management

Rebreather MCU software failure could result in the inability to manage a subsequent failure of the FPGA and other electronics. This would be a dual failure but could occur through gross mechanical damage, total power loss and by flooding. It is mitigated by both FPGA and MCU detecting a total power loss or other gross failure and putting the oxygen injector into a safe state: enough energy is stored to enable this to occur.

7.13 Failure of software

Rebreather MCU software failure could result in false safety data being transmitted, including false optimisation data for the PPO2 control. The FPGA will limit that data and provide a correct replicate. The FPGA does not allow the PPO2 to move outside the range 0.3 to 1.6 ATM, regardless of the instruction from the MCU, and regards an error of more than 0.1 as indicating a potential failure of the MCU, invoking self checks of the MCU and a handshake protocol for switching off the MCU if it fails to respond as expected in a timely manner.

7.14 Failure of electronics

Failure of the electronics should be detected and managed by the MCU unless it is a gross and total failure that affects the ability of the MCU to operate. All sensors are checked and screened throughout the dive.

The total failure scenario is managed by the FPGA and MCU detecting a total power loss or gross failure and putting the oxygen injector into a safe state before the failure becomes total: this is fast enough to manage a complete and total sudden power loss – this has been tested by removing the batteries and switching off umbilical power.

7.15 Failure of lighting and visibility

Top side software could signal the lights and cameras to be switched off on the commercial SSUBA rebreather. The supervisor can overcome this by invoking another instance of the software and switching the lights and cameras back on.

7.16 Failure of depth reporting

Efforts have been expended to create a pressure sensor to measure depth that is not subject to helium drift. The best available practice has been applied.

If an error in depth reporting exists, there is a risk that the diver's umbilical or the diver could be sucked into a thruster or come into contact with a moving underwater structure or vehicle, with fatal consequences.

Further efforts will be applied in this area using 3-axis gyros or accelerometers, to track the diver's position, but this is a research topic at this time and not available in an integrated product, or one that can be used in a saturation environment.

7.17 Failure of equipment maintenance monitoring

Rebreather MCU software failure could result in errors to check the scrubber life, sensor replacement, sensor operation and other features used to check the proper maintenance of the equipment.

Such a failure should be obvious and gross, such that the unit could not be dived in this situation. The FPGA does the onboard logging and would reject faulty data, as the CRCs would be wrong.

8 SOFTWARE FAILURE MODES, EFFECTS AND CRITICALITY ANALYSIS (SFMECA) OF MONITORS AND DIVE COMPUTERS

For the Open Revolution sports product range, rebreather monitors, tank contents monitors and dive computers provide sensory data to the diver.

A dual independent MCU is used for these products: one to gather the data and provide a buddy display, the second to check the data and advise the diver.

The MCU that is checking the data, is an ARM processor using a MAC protocol such that its failure is detected by the originating MCU, so a buddy alert can be posted. The ergonomics have been designed to make any such failure immediately obvious.

Other than this, the same hazards exist as for the eCCR, except that there is no O2 injector control: this is performed by the diver. The ergonomics of other sensory data has been designed to make the failure apparent.

The sports products use an auto-shut off system on the loop, forcing the diver to bail out when a serious failure occurs.

The sports diver is trained and instructed to carry completely independent bail out, tables and a depth timer.

The use of twin MCUs, one to perform an act and the second to check, is a considerable improvement on the use of single MCUs that are used by all sports dive electronics at present. The MCUs in the O.R. products follow the same safety design processes as for the eCCR, again a large improvement on contemporary units.

9 REDUNDANCY INTERFACE FAILURES, ECCR

The Open Revolution eCCRs have an interface between the FPGA and MCU, and in dual scrubber units, there are two completely independent sets of such controllers: quad redundancy. As these act independently, a behaviour of one controller may not be shared by the behaviour of another, and this can appear confusing to the user.

The dual redundancy can mean that one side of the rebreather controller can “wake up” and control the PPO2 before the other wakes up. To overcome this problem, there is a mutual wake up process.

In quad redundant configurations, such as the dual scrubber SSUBA rebreather, one pair may wake up before the other under some circumstances, including a dive being started without the supervisor being aware, or without pre-dive checks. This situation has been tested in manned underwater trials and validated as not being hazardous.

There are a large number of connectors on the SSUBA configuration. Failure of those connectors can result in the loss of important safety data. The connector operation is managed by checks prior to every dive, and by scheduled replacement.

All connector channels are dual redundant at the pin level, with two independent channels from the rebreather through to the supervisor. Some sections of those communication channels identified as being more prone to failure, such as the main umbilical lines, are quad redundant using two different forms of transmission media.

10 FOUNDATIONS

10.1 Training and Qualifications

As required by CASS and EN61508, the Project Leader, Verification Leader and the Software Team Leader are qualified as FIEE where they qualified in Europe, or the equivalent level if outside the UK. The project leader is an FIEE and FIET, with over 33 years of continuous electronic and software design experience.

All team members are qualified in their field of work, to degree level with at least 5 years of experience.

All staff on the project have been trained in the 61508 requirements and have attended training on the CASS Process, either externally or internally.

A training matrix was used to verify that staff allocated to the project are properly trained.

Training was given to staff, including use of lectures, written material and examinations, in areas where special risks occur. This training was repeated in several cases where weaknesses were detected by unannounced internal audits or by the review panel, and the associated work was reverified.

10.2 Power Systems: Sufficiency

The SSUBA versions of the product have quad redundant power systems:

1. Two independent power sources in the umbilical, which are maintained as independent supplies for LEFT and RIGHT rebreather channels right through to the Base controller.
2. There are two Base controllers, each of which is dual redundant, and have two power supplies from two independent batteries. The batteries are in different one atmosphere compartments, outside the breathing loop.

For the SCUBA version of the product, there are twin redundant power supplies provided by the two independent battery sources, with rigorous supply testing during pre-dive checks.

The MTBF shows that dual redundancy of supplies is sufficient, subject to sufficient pre-dive checks being carried out on the supply. The report DV_OR_SaphionCells_080415.pdf describes the pre-dive checks of battery life: the unit shows no-dive if there is less than 70% battery life on any battery.

The power drain of the product should enable 10 hours of operational life, with hundreds of hours of standby. The unit switches off the batteries completely when the batteries are discharged to the level that further discharge would degrade the battery capacity permanently.

10.3 Power Systems: Brown-out

Most safety critical systems try to establish a stable power supply and have only a reset in the event of brown out. This approach is difficult, because there is such a wide range of possible brown out conditions.

The O.R. products take the converse approach, of assuming the power supply is a fundamentally unstable resource. The equipment is designed to provide the required degree of safety in this environment, with features that provide a fail safe oxygen supply in

the powered down state, and clean transitions between the powered up states. The system defends itself from power down even second, with a partial power down process occurring: this is described in DV_OR_Power_080319.pdf, where brownouts are actively sponsored by the two controllers in each Base controller as a means of checking that both sides are working correctly.

Given this approach, it is no surprise that the equipment withstands all power supply noise conditions tested, from a 1us total brownout, to a 1s brownout, to permanent power loss. The document TF_Rebreather_Environment_Emulator_080403.pdf describes a test fixture offered commercially by Deep Life, to test the power stability of rebreather controllers: this is a development from a device described in a report delivered by Dr. Wray of the UK HSE in April 2008 on a method for testing rebreather controllers for short duration brownouts.

The equipment withstands all brown-outs, from 1us to 10ms, where upon it enters its brown out management code.

10.4 Power Systems: Power interruption

Secondary cells are used, that are soldered in, so there is no possibility of "battery bounce" and suchlike defects.

However, the system has been tested using swept power interrupts from 1us to 1s using the methods described in the previous section.

10.5 Power Systems: Power noise

EMS is the subject of a separate DV report.

The electrical noise environment envisaged by CE and FCC regulators does not reflect the possibility of extremely high noise the equipment may experience near ships radar or during underwater plasma cutting operations. Therefore separate testing has been carried out, described in the report Compliance_EN14143_DLORRevC_070427.pdf, Section 26.3 where the equipment was exposed to current densities of 3,000 Amps per sq. metre underwater, and also to high current arc welding underwater immediately adjacent to the equipment. No malfunction of the equipment was observed in any test.

The equipment is designed using best practice in EMS and EMI performance, including burying all tracks in inner layers of circuit boards with non-signal ground layers on all external surfaces of all boards.

10.6 Power Systems: Power shorts to signals in connectors

The Umbilical Terminator has a wiring board between the terminator electronics and the connectors, which provide opto-isolators for each digital line and transformer isolation for all electrical lines, in addition to use of surge suppressors and over-voltage protection to cover both EMI discharges as well as water penetration into connectors which could connect the 24V umbilical power to any signal pin. All connectors have metal shrouds, which are grounded, such that a ground connection is made before a signal connection: this minimises the stress on the surge and over-voltage protection provisions in an operational environment[3].

[3] Wiring board added following manned underwater trials, where water did penetrate connectors causing shorts from power to signal. Fitted as a recall and rework to all units.

10.7 Watchdog Timers

The Base Controllers are dual redundant. Each controller has an integral watchdog timer, that forces the controller into a recovery mode if it does not carry out the required watchdog functions within each 10 second period (in sleep mode, every second in normal operation). Additionally, each controller tests the other for a prompt and response handshake every 10 seconds. This provides a three layer watchdog that provides comprehensive management of soft failures in the dynamic or static storage of either controller.

Check sums are used on all data transmissions: a unified data protocol is used: this is described in the Unified Data Transfer Protocol listed in Section 2.

10.8 Clocks

Clocks are another Achilles heel of electronics: any clock failure causes a severe malfunction of the electronics. To mitigate these risks to SIL 4 level, dual redundant clocking is used in each of the Base controllers.

10.9 Code Transfers from Memory

The initial prototype used Xilinx FPGAs, which are configured by a download following reset. The transfer of configuration code from external memory to the FPGAs during the start up sequence was identified as an unacceptable safety hazard for SIL 3 and above during earlier safety reviews – any memory or transfer fault would cause an electronics fault: See FMECA_OR_V3_Elec_FMA_070404.pdf. As a result, the entire Base Controller was redesigned using Actel low power FPGAs where the logic configuration is stored internal to the FPGA as a cell configurator. The same Actel FPGA family is used in the active surfaces and flight controllers in civil aircraft, including in Boeing 777 and Airbus 380 aircraft.

Xilinx FPGAs remain in the umbilical terminators: these have a lower SIL rating (SIL 2), and are dual redundant, so the chance of both FPGAs suffering the same problem at the same time is extremely low.

10.10 MCU, ROM and Memory Self Test

The MCU executes a self test sequence whenever it powers on, and whenever it exists from sleep mode. The test checks:

- CPU in the MCU
- RAM self test
- ROM test, using CRC check
- Peripherals and sensors.

10.11 Guards against illegal jumps

A scheduler traps all processes that do not return within the allotted time slot: this is a fundamental feature of the Time Triggered Architecture.

Illegal accesses to memory locations are trapped by Abort handling, and control is returned to the scheduler to abort the task that originated the illegal access.

All errors are logged by the scheduler in a dedicated data area, and may be accessed by an external utility via the USB port.

10.12Guards against variable corruption

The scheduler includes a super-process that checks all system level variables against allowed ranges.

All application tasks check the input and output variables against allowed ranges.

10.13Stack checks

There are no stack checks required because of correct memory mapping: as soon as the stack increases beyond its allowed space, it creates an illegal address access which is trapped as described above using the same mechanism as for illegal jumps.

10.14 Program calculation checks

The FPGA monitors the results continuously and vetoes any attempt to move the operating parameters outside the safe limits.

11 SAFETY ARCHITECTURE

The Umbilical Terminator, communications and Top Side unit, have no micro-controllers: they are configured using Verilog HDL, developed using a proven ASIC design path (with Mentor and Synopsys Toolsets).

The primary rebreather controller, that is, the controller that drives the oxygen injector, is not a micro-controller, but state machines configured using Verilog HDL in a 3 million gate FPGA, using an Actel part that is used for active surface control in modern civil airliners.

A complete Time Triggered Architecture (TTA) is used for the microcontroller that monitors the primary controller: that controller has the means to take over safety functions. It is also used to optimise safety functions carried out by the primary controller. For example, the primary controller ensures the PPO2 is within safe limits, but provides data to the microcontroller which is computed and the result used to fine tune the PPO2 control.

TTA is the highest level of safety architecture for a processor, or set of interconnected processors, used in automatic car braking systems by Mercedes Benz, and in critical weapon and aviation applications. The architecture has been proven with decades of experience. The TTA requires that there be no interrupts, and instead time slots are allocated for each safety task. Monitoring software ensures each task completes within its time slot, and does not permit one task to rob resources from another safety function. The use of TTA greatly simplifies the task of verifying the correctness of each software module, and the software as the integral of all modules. TTA is described very widely in the literature, to the extent that no specific references should be required: a Google search will find very many papers on the topic.

12 TIME TRIGGERED ARCHITECTURE IMPLEMENTATION

The implementation of TTA in the FPGA is the default when there are no external variables that can force a majority of the state machines into a subprocess: the only such external variable is the reset signal.

For the microcontroller that acts as a safety monitor, the implementation of TTA is more challenging. To ensure TTA is applied universally, a TTA supervisor system is used. This is a small code kernel described in the following sections with the help of flow diagrams showing the actions for:

- Build time
- Run time
    - Initialisation
    - Task setup and control
    - Interrupts and watchdogs.

A full UML description of the TTA Supervisor has been inspected and the code itself has been checked using the Decision control point method.

12.1 Definitions and Processes in the TTA Supervisor

Entity	Description
Task	A procedure with a standard pre-defined interface that realises an application function, which is called in the
	time triggered control system in strictly determined time periods, and which has a strictly defined maximum
	execution time
Task Descriptor	A data structure that describes the task, including its unique identifier and its maximum execution duration
Task Plan	A data structure, which determines the sequence of tasks calls, including each task call period and call phase
Task Plan Descriptor	A human readable source file that contains the sequence of task calls with the period and phase for each task
Task Scheduler	A software procedure that implements the task plan
Scheduler Utility	A software application that generates a task plan from the set of human-generated task descriptors and a task
	plan descriptor
Compiled Task Descriptor	A task descriptor structure allocated in the binary image, used by Task Scheduler; includes constant task
	duration field, constant actual task body function entry address, and variable task status field
Step Descriptor	A data structure describing a single step of the Task Scheduler function (a single task call)
Compiled Task Plan	An array of step descriptors forming a full Task Plan
Scheduler Task Phase	A time period started by the task body call and limited by the timer interrupt; its duration is equal to the task
	body requested duration
Scheduler Post Delay Phase	A time period started at the task actual termination and limited by the timer interrupt; its duration is equal to
	the requested delay before the next task call

1. The time-triggered architecture of the Base Unit controller is implemented in two stages

    a. At the build time the Scheduler Utility generates the "C" source files from the human-generated task descriptor and task plan descriptor sources; further the output of the Scheduler Utility together with other application "C" sources is transferred to the GCC tool, which generates the microcontroller code image; the image contains the compiled task plan and task descriptor structures to be used at the run time

    b. At the run time the Task Scheduler component of the application code reads the task plan one item (step) after another, configures its timer according to requested task durations and calls a task using its start address from the task descriptor; the task that exceeded the requested duration is aborted (it is a hard error eliminated at the debugging stage); the task that finished earlier than requested moment, is followed by a so called post delay before the next task is called.

2. At the system initialisation stage, before the Task Scheduler is started, the tasks initialisation procedures are called sequentially for all tasks; the tasks may be parameterised at this early stage.

3. The user stack is cleaned up before each task call by the Task Scheduler, thus the stack overloading is not possible in the system.

4. The first call of the Task Scheduler is implemented through a call ExecuteTaskPlan(), which sets up and calls the first task in the requested plan; further initiation of the Task Scheduler functions is implemented only through the timer interrupts. Due to this fact and to the fact that no other activity, except of tasks called by the Task Scheduler, exists in the system, the controller represents a fully synchronous time triggered state machine.

5. Depending on the application the system may have multiple task plans, switching from one to another in dependence of the output environment or its own internal state. Additional Task Scheduler functions support these transitions. The programmer develops descriptors for each of plans to be implemented.

6. The Scheduler Utility uses effective data compression algorithms to save the memory usage for task and plan descriptors. Actually it is implemented through the search of identical branches within the task sequence, and implementation them as separate subtask plans (similar to subroutine calls).

7. The Scheduler Status utility is used to observe the state of the Task Scheduler and of each the current task plan task. Its output log received from the Base Unit serial number 0008-C1 is represented in a separate text file 0008-C1.txt

8. The Scheduler Status utility output table contains the following columns:

    a. ID – a unique task identifier

    b. TASK NAME – task identifier in a convenient human-readable format, used in other diagnostic software

    c. ADDRESS – is a the task body function entry address

    d. STATE – is the latest termination status of a task (TERM – terminated successfully, NSTRD – not started yet, ABRTD - aborted)

    e. COMPLETE – the status returned by a task at the termination (SUCCESS, for not started task it is UNKNOWN)

    f. ABORTS – the accumulated number of a given task aborts

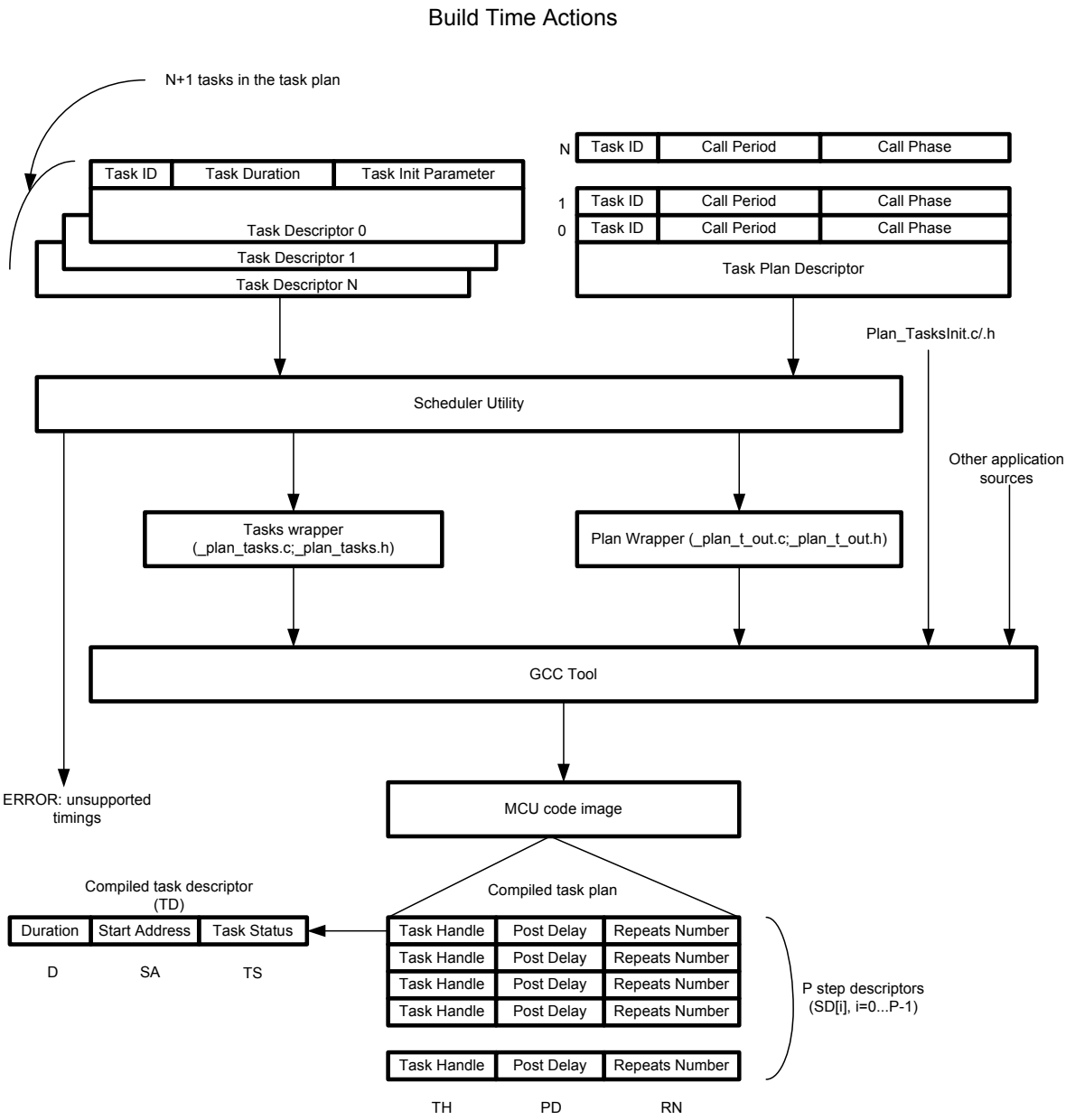
    g. MAX TIME (uSec) – maximum registered execution time of a task in microseconds

    h. MIN STK – the stack pointer value at the task abort, set to -1 if no aborts registered

i. LR ADDR – the link address register value at the task abort, set to -4 if no aborts registered

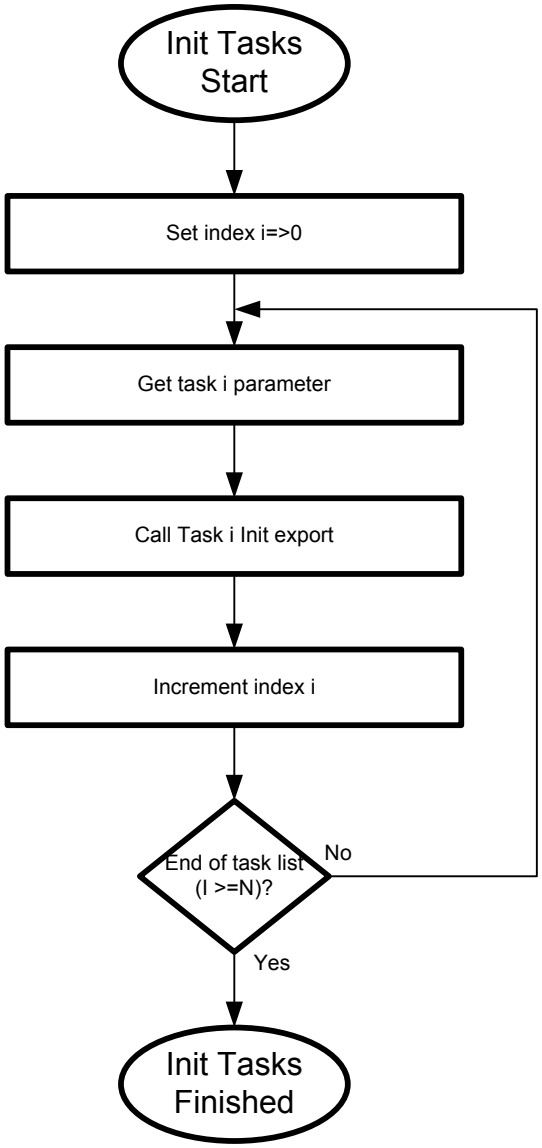
9. The Scheduler Status utility task plan descriptor output represents the compiled task plan converted to a human readable format. This part of the log is typically used for the analysis of Scheduler Utility, which is checked against the UML description for the task.

12.2 Build Time Actions



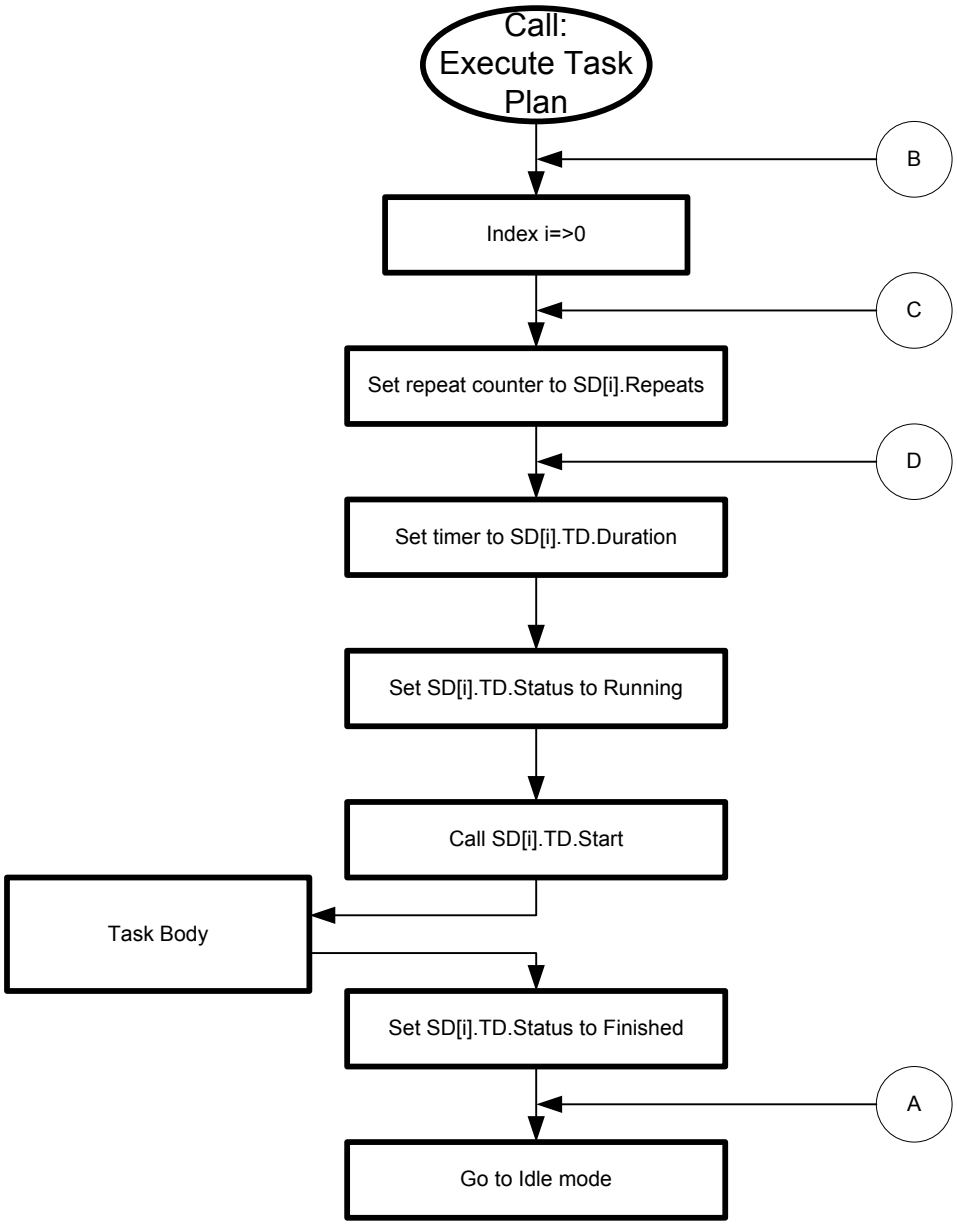
12.3 Runtime Initialisation

Runtime Init Actions



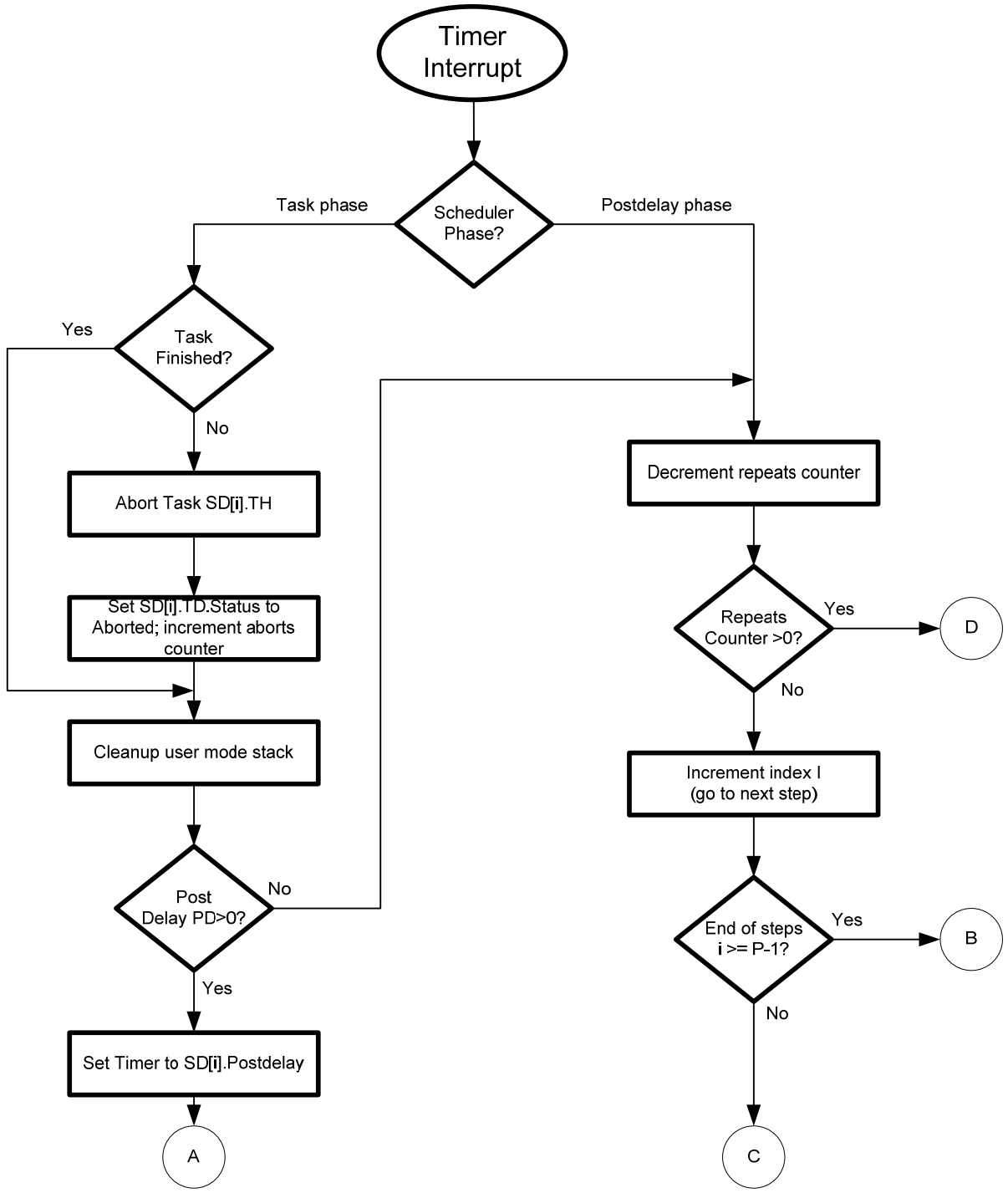
12.4 Runtime Task Body Setup and Control Actions

Runtime Task Body Setup Actions



12.5 Runtime Timer Interrupts

Runtime Timer Interrupt Procedure



13 FORMAL VERIFICATION

Formal verification is a formalisation of the user requirements, and a model of the system implementation, which is verified against each other using mathematical methods of a known rigor. Formal verification is applied at three main levels in the project:

1. Specification Modelling (User requirements versus a system model).
2. System Modelling (Implementation versus an independent system model)
3. Logic Modelling (Back annotated versus logic specification)

This is supported by empirical validation, which is a detailed test and characterisation of system components: these are called Design Verification reports, but in fact are the results from empirical validation.

13.1 Specification Modelling

Two specification languages were considered for this project: Temporal Z[4] and Matlab.

The team has experience of Z, but considered the limitations of finite data domain and the lack of simulation tools to be too severe in this application where data is virtually continuous (a relatively large number of sensors, with very high digitising resolution).

The team is also experienced in Matlab. The advantages of Matlab over Z include:

1. Versatile toolset widely available, to write, run and test the simulations
2. Ease of reading so peer review is much more likely
3. Ability to execute a simulation by combining a specification of the environment with the specification of the equipment.
4. Ability to perform mixed real and synthetic simulations, where a sample of any module in the equipment can be connected to an ADC/DAC card, and run. This allows the performance of modules to be assessed using Monte Carlo methods, as well as tested under simulated fault conditions at an early phase.

The fundamental difference between Matlab and Z approaches, is that in Z specifications are proven to be equivalent to an implementation by mathematical theorem proving methods, but in Matlab the specification is executed and compared using samples. The number of samples can be extremely large: Deep Life operate simulation processor farms for this purpose. Given the input data is in fact continuous and not discrete, then the coverage of the two methods can be similar, by using hierarchy to define, test and compare each submodule. By substituting modules with actual hardware, unexpected features such as hysterisis in an injector, are identified very quickly, and the effect on the rest of the system assessed.

After weighing these different advantages and disadvantages of the two specification environments, Matlab was chosen for this project.

The entire specification of the base controllers and the operating environment is expressed as a Matlab model, that has been published as part of this project and received peer review, including use by third parties.

None of the Matlab code is used to implement the system: it is a specification only, broken down by stepwise refinement into detailed module operations which have been compared systematically with the actual implementations and hardware samples.

---

[4] **Specification and verification of multi-agent applications using temporal Z** Regayeg, A.; Hadj Kacem, A.; Jmaiel, M. Intelligent Agent Technology, 2004. (IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on Volume , Issue , 20-24 Sept. 2004 Page(s): 260 - 266 Digital Object Identifier 10.1109/IAT.2004.1342953

The Matlab environment and rebreather model has been published, and taken up some third parties. It is available from www.deeplife.co.uk/or_models.php. This environment has been used very extensively throughout the project, and the environment extended where required, such as to assess the operation of powered breathing assistance: See DV_PABA_WOB_061006.pdf. The model allows breathing curves to be predicted at an early specification stage, even with details such as counterlung centroid movement ("Elastance" in NEDU parlance, and "Compliance" for CE SC7) in RB_formal_model_for_elastance_080131.pdf, as well PPO2, PPCO2, Loop volumes, injector positions etc: the model is extremely detailed covering every meaningful aspect of the system and its environment from a functional standpoint.

13.2 Logic Modelling: Processors and State Machines

The Design Authority of the ARM processor has stated that formal theorem proving has been applied to the ARM 7 processor, with comparison of back annotated netlist captured from the silicon layout to the logic specification for the device, using Synopsys Formality tools.

Other groups independent of the manufacturer or Design Authority, have also formally verified the ARM 6[5] and ARM 7 processors[6], and designers have been advised of and corrected processor bugs identified as a result[7].

The logic in the FPGAs is verified using a proven ASIC design flow within Deep Life Ltd, including back extraction of the netlist, equivalence checking and NetVS. Synopsys Formality has not been used in this instance, due to the layout itself being the intellectual property of Actel Corporation.

---

[5] A. Fox. "Formal specification and verification of ARM6." In D. Basin and B. Wolff, editors, *TPHOLs '03*, volume 2758 of *LNCS*, pages 25-40. Springer, 2003.

[6] SUSANTO Kong Woei and MELHAM Tom, "An AMBA-ARM7 formal verification platform", Formal methods and software engineering : (Singapore, 5-7 November 2003) ICFEM 2003 : international conference on formal engineering methods N°5, Singapore , 2003, vol. 2885, pp. 48-67, **ISBN** 3-540-20461-X ;

[7] V.A. Patankar, A. Jain, R.E. Bryant, "Formal Verification of an ARM Processor," *vlsid*, p. 282, 12th International Conference on VLSI Design - 'VLSI for the Information Appliance', 1999

14 UML

UML is a primary software design tool applied by the software development team.

UML descriptions exist for all functional blocks in the ARM processor. These diagrams are extensive.

The UML tools with direct compilation are not be coupled with a verified compiler, or a compiler which is deemed equivalent in accord with Deep Life's QA procedures. For this reason, the UML is maintained manually, with group reviews of the code against the UML, and automated Monte-Carlo simulation of the resulting code against the top level formal specification and environment model.

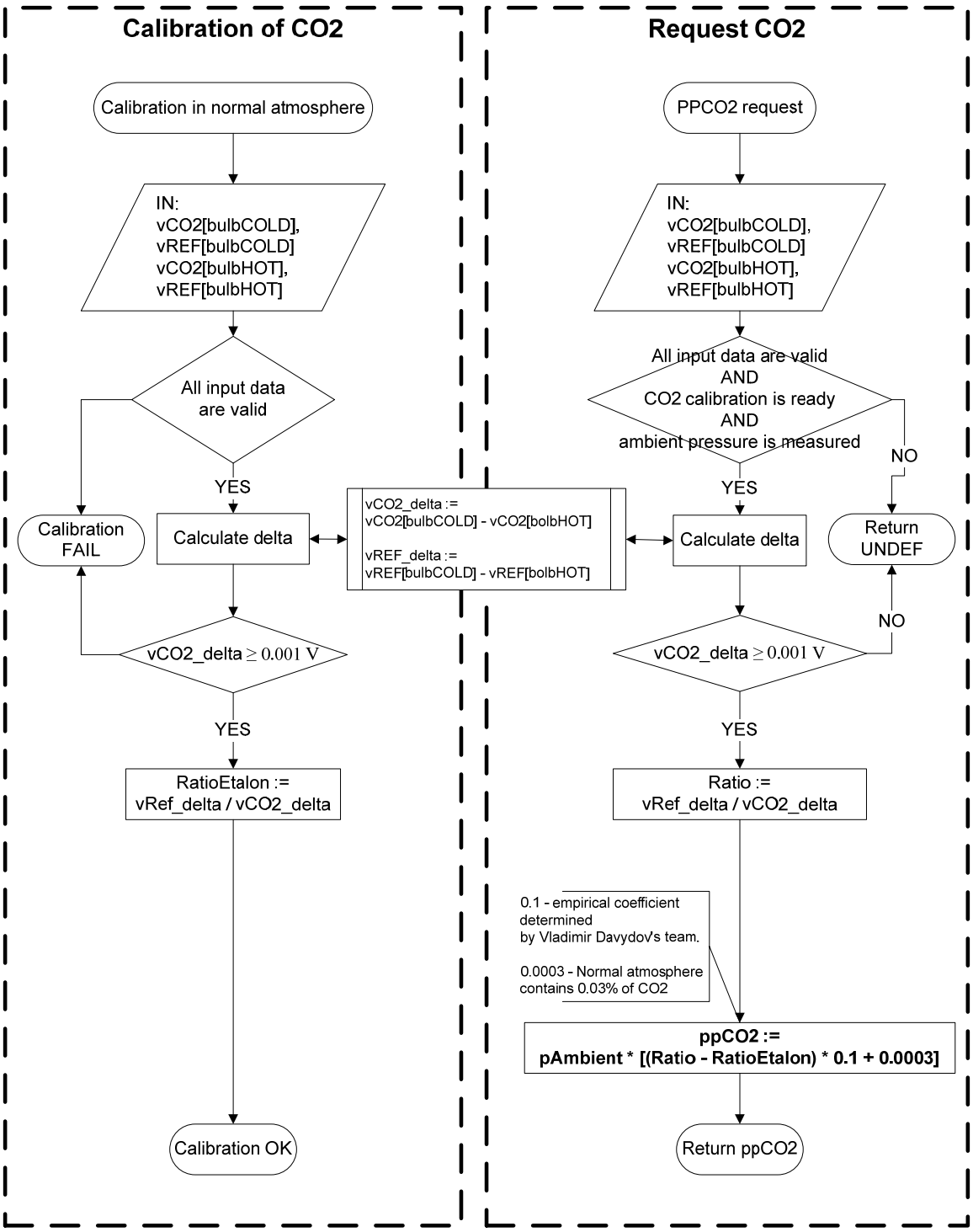
Examples of the UML descriptions, particularly Activity Diagrams and State Diagrams, are reproduced in the following sections (Capture date 16th June 2008), for the most safety critical functions:

- Oxygen sensor calibration
- Carbon Dioxide sensor calibration
- Oxygen sensor screening
- Switching on and off states (Sleep, On state)
- Oxygen Injector control

14.1 Activity Diagram: CO2 Sensor Calibration

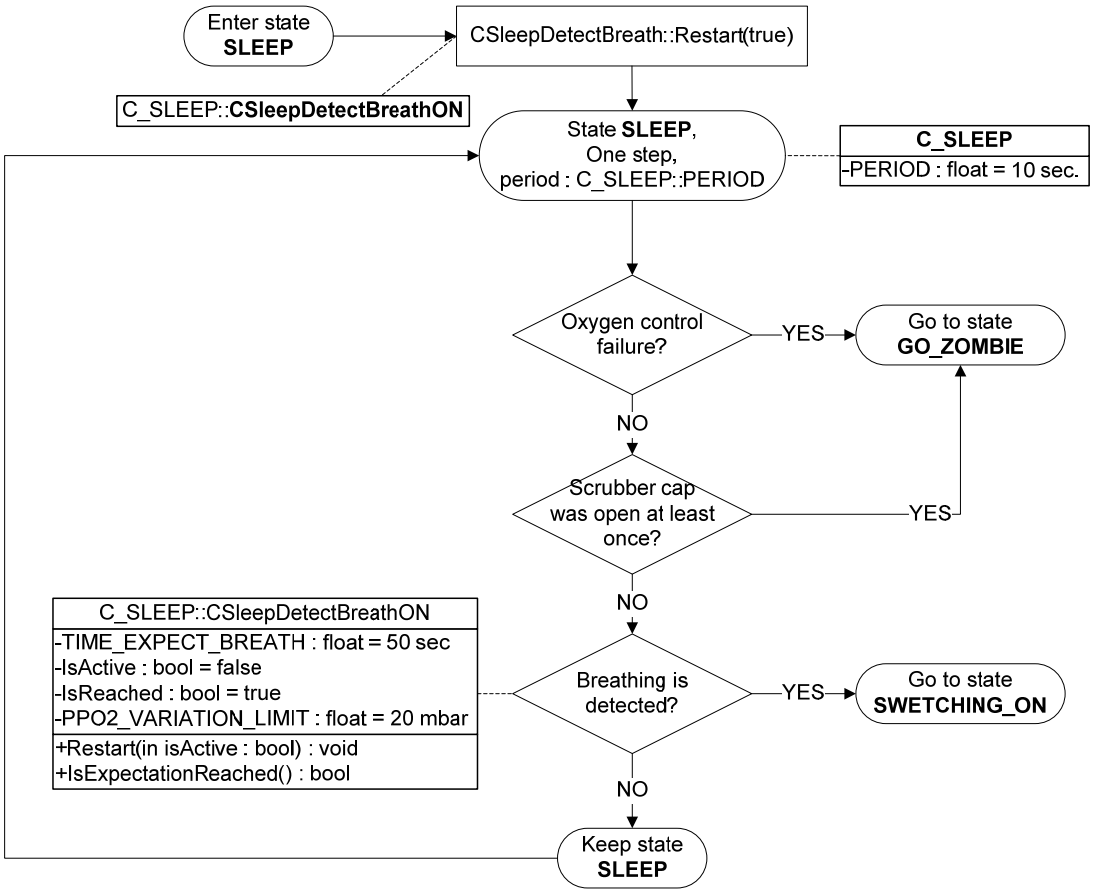
DOCUMENTATION:
Vladimir Davydov

IMPLEMENTATION:
svn.deeplife.co.uk/svn/REBREATHER\CONTROLLER\FIRMWARE\DESIGN.4\BASEUNIT\ALGORITHMS\src\task_calibrate.c
svn.deeplife.co.uk/svn/REBREATHER\CONTROLLER\FIRMWARE\DESIGN.4\BASEUNIT\VIEWPOINTS\src\viewpointCalibPPCO2.c



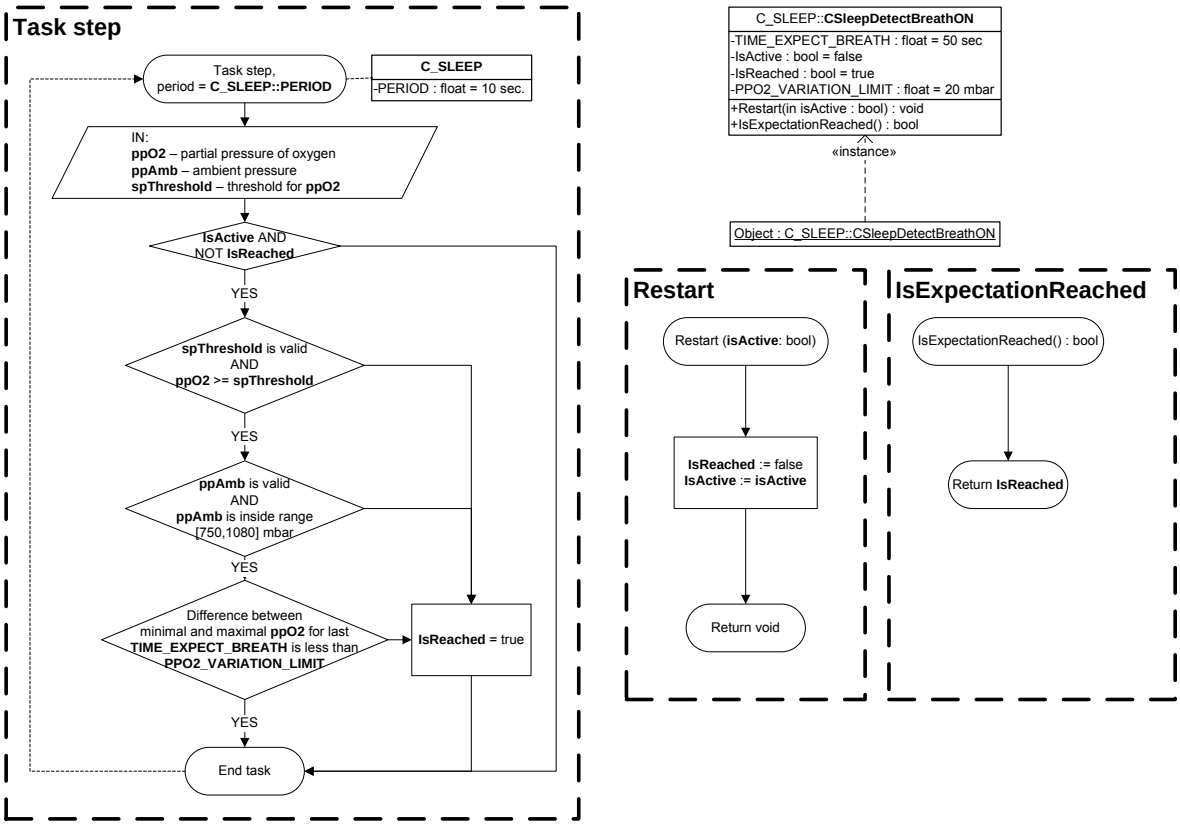
14.2 Activity Diagram: Sleep

IMPLEMENTATION:
svn.deeplife.co.uk/svn/REBREATHER\CONTROLLER\FIRMWARE\DESIGN.4\BASEUNIT\ALGORITHMS\src\autom_state_f_SLEEP.c



14.3 Activity Diagram: Sleep Detection

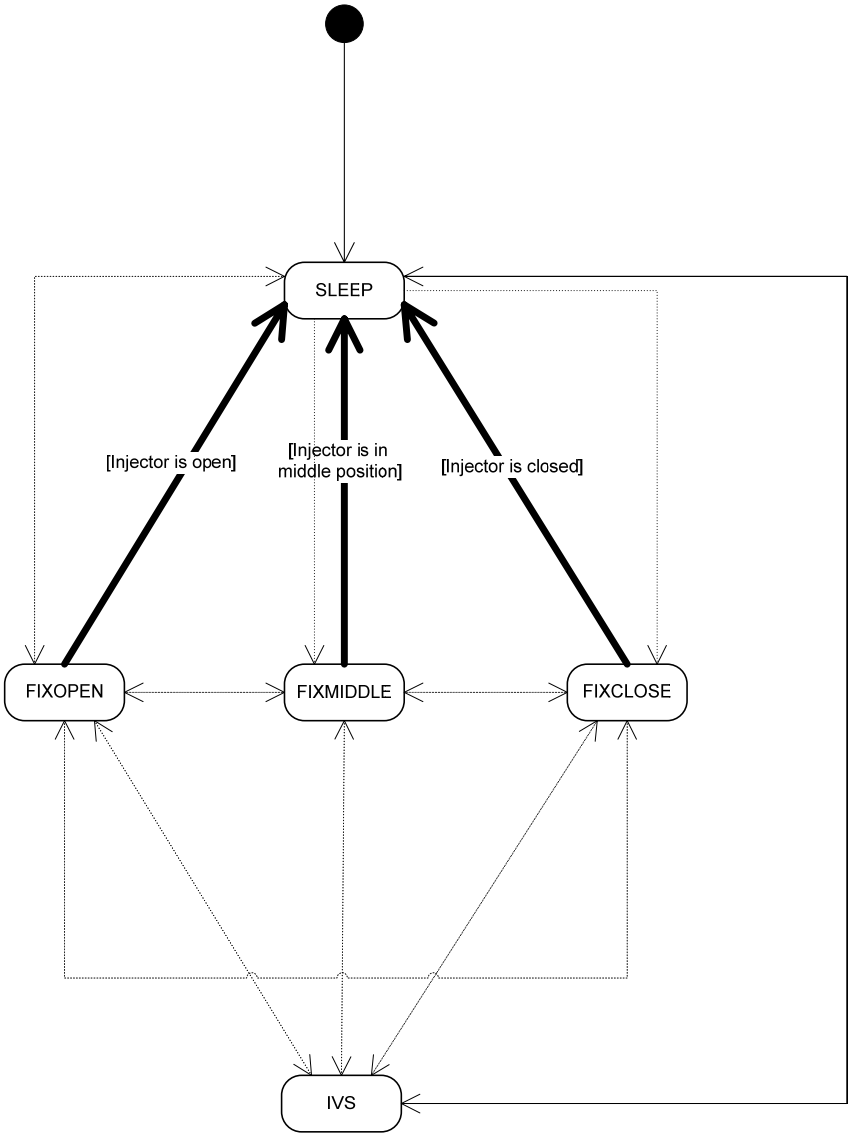
IMPLEMENTATION:
svn.deeplife.co.uk/svn/REBREATHER\CONTROLLER\FIRMWARE\DESIGN.4\BASEUNIT\ALGORITHMS\src\task_detectbreathON.c
svn.deeplife.co.uk/svn/REBREATHER\CONTROLLER\FIRMWARE\DESIGN.4\BASEUNIT\ALGORITHMS\src\pressurelimits.c



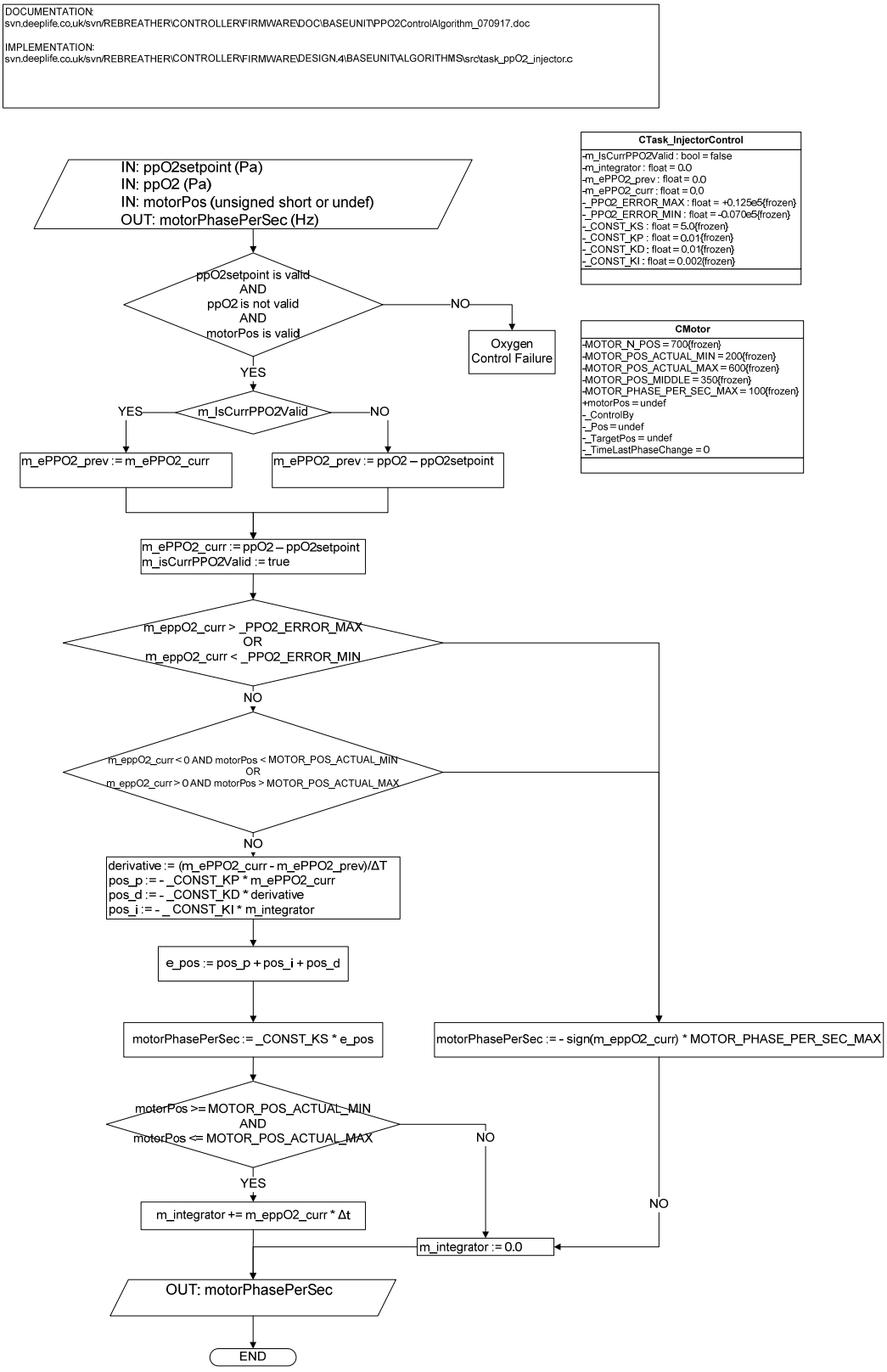
14.4 State Chart: Oxygen Injector States

DOCUMENTATION:

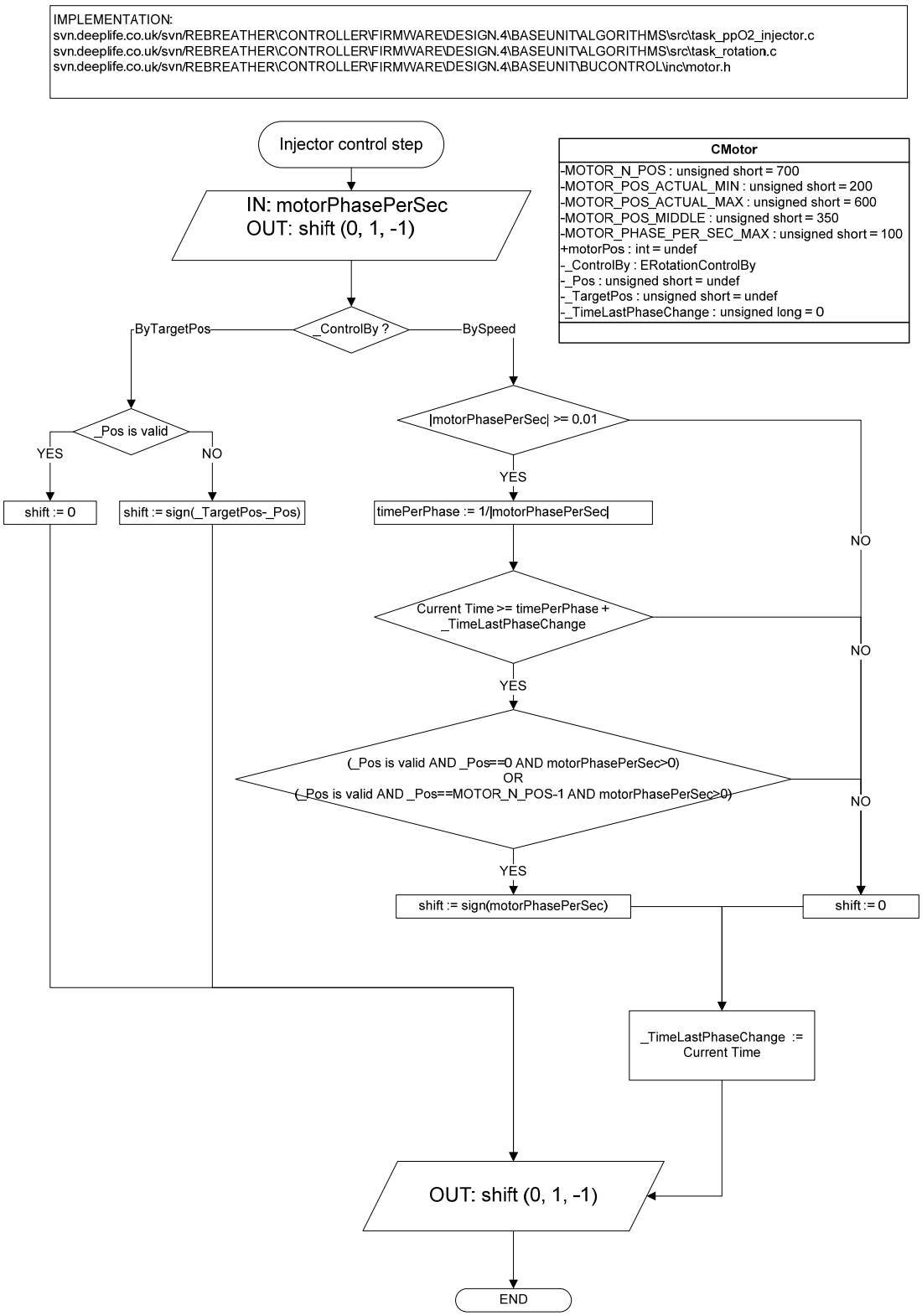
IMPLEMENTATION:
svn.deeplife.co.uk/svn/REBREATHER\CONTROLLER\FIRMWARE\DESIGN.4\BASEUNIT\ALGORITHMS\src\task_ppO2_injector.c



14.5 Activity Diagram: Oxygen Injector IVS State



14.6 Activity Diagram: Injector Control



15 ALARM MATRIX

The alarm hierarchy has been documented and reviewed in the form of an alarm matrix comprising an Excel file with a worksheet per functional unit and a top level structure categorising the alarms and response. This matrix gives the alarm and responses for each of the functional units that can express alarms within the system.

The latest version of that document at time of writing is RB_SSUBA_arlarm_matrix_20080816.xls and is maintained on the SVN under source control, and an extract from that document is reproduced below with capture date 16th June 2008. Refer to the SVN for updates to that Excel file.

Event Group	Event ID	Relative Priority (0 is the highest)	Channel Event		Event Source	Response Tag in						Comment
			Description	Feature		Dive Supervisor Console (DSC)	Top Side Termination LEDs (TSL)	Top Side Termination buzzer (TSB)	Helmet LEDs (PFL)	Helmet Voice (PFV)	Base Unit LEDs (BUL)	
General	1.1		No events detected		X	DSC0	TSL0	OFF	PFL0	PFV0	BUL0	
System	2.1		Base Unit: microcontroller section failed		Base Unit telemetry	DSC-A1	X	OFF	PFL1	PFV1	BUL2	
	2.2		Base Unit: FPGA section failed		Base Unit telemetry	DSC-A2	X	OFF	PFL1	PFV1	BUL2	
	2.3		Base Unit: O2 injector failed		Base Unit telemetry	DSC-A3	X	OFF	PFL1	PFV1	BUL2	
	2.4		Base Unit: calibration in progress or failed		Base Unit telemetry	DSC-A4	X	X	X	X	BUL3	
	2.5		Telemetry data is not updated in the database (telemetry stream is lost)		SQL server	DSC-A5	X	X	X	X	X	
Data Link	3.1		Base Unit: telemetry timeout		Telemetry stream	DSC-A6	X	OFF	X	X	X	
	3.2		Helmet (PFD): telemetry timeout		Telemetry stream	DSC-A6	X	X	X	X	X	
	3.3		UT (MUX) : telemetry timeout		Telemetry stream	DSC-A6	X	X	X	X	X	
	3.4		Base Unit: command acknowledge timeout		Command acknowledge stream	DSC-A7	X	X	X	X	X	
	3.5		Diver Side UT: command acknowledge timeout		Command acknowledge stream	DSC-A7	X	X	X	X	X	

	3.6		Top Side UT: command acknowledge timeout		Command acknowledge stream	DSC-A7	X	X	X	X	X	
	3.7		Helmet (PFD): command acknowledge timeout		Command acknowledge stream	DSC-A7	X	X	X	X	X	
	3.8*		Diver side umbilical data link failed		Bell Termination telemetry	DSC-A8	X	OFF	X	X	X	Requires Bell side UT box firmware implementation
	3.9*		Top side umbilical copper data link failed		Top Side Termination hardware and telemetry	DSC-A9	TSL1	OFF	X	X	X	Requires Top side UT firmware upgrade
	3.10*		Top side umbilical fibre data link failed		Top Side Termination hardware and telemetry	DSC-A10	TSL2	OFF	X	X	X	Requires Top side UT firmware upgrade
	3.11*		Diver side umbilical video link failed		Bell Termination telemetry	DSC-A11	X	OFF	X	X	X	Requires Bell side UT box firmware implementation
Parametric	4.1.1	1	PPO2 level average	>Max. Alarm	Base Unit telemetry	DSC-A12H	X	OFF	X	X	BUL4	
	4.1.2			>Max. Warn.		DSC-W1H	X	X	X	X	X	
	4. 1.3			Normal		X	X	X	X	X	X	
	4. 1.4			<Min. Warn.		DSC-W1L	X	X	X	X	X	
	4. 1.5	1										
				<Min.		DSC-A12L	X	ON	X	X	X	

			Alarm								
4.1.6	1		≤ 0.2		DSC-A12L	X	ON	PFL2	PFV2	BUL4	
4.1.7	1		Undefined		DSC-A13	X	ON	PFL2	PFV3	BUL4	
4.2.1	2	PPCO2 level	≥ 0.065	Base Unit telemetry	DSC-A14	X	ON	PFL2	PFV3	BUL4	
4.2.2	2		>Max. Alarm		DSC-A14	X	ON	X	X	X	
4.2.3			>Max. Warn.		DSC-W2	X	X	X	X	X	
4.2.4			Normal		X	X	X	X	X	X	
4.2.5			<Min. Warn.		X	X	X	X	X	X	
4.2.6			<Min. Alarm		X	X	X	X	X	X	
4.2.7	2		Undefined		DSC-A15	X	ON	PFL2	PFV3	BUL4	
4.3.1	3	PPCO level	≥ 2ppm	Base Unit telemetry	DSC-A16	X	OFF	PFL2	PFV3	BUL4	
4.3.2	3		Max. Alarm		DSC-A16	X	X	X	X	X	
4.3.3			Max. Warn.		DSC-W3	X	X	X	X	X	
4.3.4			Normal		X	X	X	X	X	X	
4.3.5			Min. Warn.		X	X	X	X	X	X	
4.3.6			Min. Alarm		X	X	X	X	X	X	
4.3.7	3		Undefined		DSC-A17	X	X	PFL2	PFV3	BUL4	
4.4.1		PPHe level	Max. Alarm	Base Unit telemetry	DSC-A18	X	X	X	X	X	Not used except for air dives
4.4.2			Max. Warn.		DSC-W4	X	X	X	X	X	to signal narcosis risks
4.4.3			Normal		X	X	X	X	X	X	

	4.4.4			Min. Warn.		DSC-W4	X	X	X	X	X	
	4.4.5			Min. Alarm		DSC-A18	X	X	X	X	X	
	4.4.6			Undefined		DSC-A19	X	X	X	X	X	
	4.5.1		Scrubber life level	Normal	Base Unit telemetry	X	X	X	X	X	X	
	4.5.2			Min. Warn.		DSC-W5	X	X	X	X	X	
	4.5.3			Min. Alarm		DSC-A20	X	X	PFL2	PFV4	X	
	4.5.4			Undefined		DSC-A21	X	X	PFL2	PFV4	BUL3	
	4.6.1		Scrubber health level	Normal	Base Unit telemetry	X	X	X	X	X	X	
	4.6.2			Min. Warn.		DSC-W6	X	X	X	X	X	
	4.6.3			Min. Alarm		DSC-A22	X	X	PFL2	PFV4	X	
	4.6.4			Undefined		DSC-A23	X	X	PFL2	PFV4	BUL3	
	4.7.1		Relative humidity level	Max. Alarm	Base Unit telemetry	DSC-A24	X	X	X	X	X	
	4.7.2			Max. Warn.		DSC-W7	X	X	X	X	X	
	4.7.3			Normal		X	X	X	X	X	X	
	4.7.4			Min. Warn.		DSC-W7	X	X	X	X	X	
	4.7.5			Min. Alarm		DSC-A24	X	X	X	X	X	
	4.7.6			Undefined		DSC-A25	X	X	X	X	X	
	4.8.1		Bail out pressure level	Max. Alarm	Diver side UT telemetry	DSC-A26	X	X	X	X	X	
	4.8.2			Max. Warn.		DSC-W8	X	X	X	X	X	

	4.8.3			Normal		X	X	X	X	X	X	
	4.8.4			Min. Warn.		DSC-W9	X	X	X	X	X	
	4.8.5			Min. Alarm		DSC-A27	X	ON	PFL1	PFV5	X	
	4.8.6			Undefined		DSC-A28	X	ON	PFL1	PFV5	X	
	4.9.1		Umbilical pressure level	Max. Alarm	Diver side UT telemetry	DSC-A29	X	ON	X	X	X	
	4.9.2			Max. Warn.		DSC-W10	X	X	X	X	X	
	4.9.3			Normal		X	X	X	X	X	X	
	4.9.4			Min. Warn.		DSC-W11	X	X	X	X	X	
	4.9.5			Min. Alarm		DSC-A30	X	X	PFL2	PFV6	X	
	4.9.6			Undefined		DSC-A31	X	X	PFL2	PFV6	X	
	4.10.1		Loop temperature level	Max. Alarm	Base Unit telemetry	DSC-A32	X	X	X	X	X	
	4.10.2			Max. Warn.		DSC-W12	X	X	X	X	X	
	4.10.3			Normal		X	X	X	X	X	X	
	4.10.4			Min. Warn.		DSC-W12	X	X	X	X	X	
	4.10.5			Min. Alarm		DSC-A32	X	X	X	X	X	
	4.10.6			Undefined		DSC-A33	X	X	X	X	X	
	4.11		Parameters except of listed: undefined data received		Telemetry stream	DSC-W13	X	X	X	X	X	
	4.12		Flood detected		Base Unit telemetry	DSC-A34	X	ON	PFL2	PFV7	BUL5	

Power	5.1		Umbilical power loss		Base Unit sensors; Top Side UT telemetry	DSC-A35	TSL1+TSL2	ON	PFL1	PFV8	X	
	5.2.1		Battery life level	Normal	Base Unit telemetry	X	X	X	X	X	X	
	5.2.2			Min. Warn.		DSC-W14	X	X	X	X	X	
	5.2.3			Min. Alarm		DSC-A36	X	X	PFL1	PFV9	X	
	5.2.4			Undefined		DSC-A37	X	X	PFL1	PFV9	BUL2	
Diver state	6.1		Long term no breathing		Base Unit internal event	DSC-W15	X	X	X	X	BUL1	
	6.2.1		Respiratory rate level	Max. Alarm	Base Unit telemetry	DSC-A38	X	ON	X	X	X	
	6.2.2			Max. Warn.		DSC-W16	X	X	X	X	X	
	6.2.3			Normal		X	X	X	X	X	X	
	6.2.4			Min. Warn.		DSC-W16	X	X	X	X	X	
	6.2.5			Min. Alarm		DSC-A38	X	X	X	X	X	
	6.2.6			Undefined		DSC-A39	X	X	X	X	X	
	6.3		Helmet open		Helmet telemetry	DSC-W17/DSC-A40	TSL3	X	X	X	X	
	6.4	0	Diver alert switch pressed		Helmet telemetry	DSC-A41	TSL4	ON	X	X	X	

There are 8 other worksheets that form this alarm matrix, all linked together by the above top level.

16 MODULE VERIFICATION

All software and firmware modules are verified by comparison with specification using automated validation methods, and by execution of a test bench at a module and system level.

Firmware modules are simulated before layout and after back extraction.

17 UNIFIED COMMUNICATION PROTOCOL

The rebreather system comprises a series of discrete functional modules:

1. Peripheral Field Display and Alarm Annunciator
2. Rebreather (2 in parallel in the commercial diving configuration)
3. Diver side Umbilical Terminator
4. Top Side Unit

To maximise diagnostic access and to minimise the verification requirements, a Unified Communication Protocol was designed that is used by all these subsystems. This protocol was designed specifically for high integrity safety systems.

This protocol supports half duplex and full duplex links across any transmission medium or transmission speed, bandwidth backoff as the medium starts to fail, fully redundant communication independent of latency across different channels and CRCs on each frame which are used to select the lowest error rate channel.

The Unified Communication Protocol is described by its own Colour Book Specifications, and is supported by tools that support systematic interrogation of functional units, frame by frame logging and full diagnostics.

18 DIAGNOSTIC UTILITIES AND LOGGING

Testing is only viable if there are good diagnostics.

In addition to the module level diagnostics, system level diagnostic tools have been developed.

18.1 TestNode Utility

This software utility is designed for use at the devices production stage, as well as for on-line diagnostics of the hardware and software. The utility is implemented as a GUI application and is supported under two platforms, Windows (x86), and Linux (x86). The utility has the following features:

- Connection to the rebreather hardware through USB ports, as well as through devices serial ports using third-party USB/RS-485 interface convertor
- Selection of the device for connection by its type and its serial number
- Full access to the peripheral devices, including reading state and writing required control words in single and periodic mode

- Function-specific dialogues, providing complicated peripheral devices control functions
- Unified data transmission protocol test functions with CRC error checks
- MCU firmware upgrade function
- Calibration data access for reading and writing
- Device identification data access for reading and writing

The advanced version of the TestNode utility contains the interpreter engine, thus providing easy, fast and flexible adaptation of this software to new or updated devices in the system. This adaptation can be implemented using a large library of scripts provided with the utility, as well by a user

18.2 FPGA Tool Utility

This console application is designed to provide FPGA firmware upgrade in all rebreather devices. In the Rebreather Base Unit the FPGA firmware is upgraded through the USB port, using the MCU controller functions managing the FPGA JTAG port. In other devices the firmware is upgraded through the device USB port, using FPGA functions managing serial EEPROM port. The utility is supported for Windows (x86) platform. It is configured through the command line parameters.

18.3 Scheduler Status Utility

This console application provides access to the MCU's Task Scheduler data structures to determine the current status of tasks execution, including for each task maximum and actual execution time, number of abort events, and the address where the last abort was initiated. The utility is supported under Windows (x86) platform. It is configured through the command line parameters.

18.4 Framer Tool Utility

This console application is specially designed for diagnostics of data transfers between devices in accordance with Unified Data Transmission Protocol. The utility provides receiving of the data stream from a device and transmission of the command stream to the device. The frames transmitted and received are logged, and the error rate parameters are calculated. The utility is supported under Windows (x86) platform. It is configured through the command line parameters.

18.5 Frame Log Parser Utility

This console application provides parsing of the telemetry stream from devices and representation in the XML format. Logs in the XML format are then transferred to the Excel application to represent the data in a table form or as a graph. The utility is supported under Windows (x86) platform. It is configured through the command line parameters.

19 REVIEWS

There are 5 levels of review applied to this project, of which 3 include a thorough review of all documentation and at least 2 levels are applied to the level of detail of all source code. The documentation documents the code, and the design tools enable the equivalence of the implemented code and the specified code to be tested.

Each of these review levels are described below.

19.1 Team Review

All software is reviewed by multiple members of the software team.

Software submitted for review has to be accompanied by a suitable test suite and documentation.

19.2 Antagonist Review

All firmware and software is submitted to a group maintained by Deep Life that is independent of all design activities other than building specification and environment models which they use to test the code submitted.

The validation group run the specification against the implementation with extensive data logging, which are checked automatically. This is applied at a module level and at a whole system level.

The Antagonist Review team have access to test chambers and facilities that are not available to the code developers, to prevent the code developers testing their code using the same or similar test conditions as the Antagonist Review team.

19.3 Independent Review

All primary documents, including all safety documents, are submitted to a team expert in the application and in diving technology for review. The results of those reviews are minuted.

The Independent Review team has no connection whatsoever with Deep Life other than it is funded by clients of Deep Life Ltd.

The Independent Review team has carried out quality audits, inspections, tests and general appraisals of the design work with a wide scope and remit. There have been more than 8 inspection visits, each of which involved a multi-disciplinary team.

19.4 Master Review

A team at the Norwegian Underwater Institute in Bergen expert in rebreather and dive technology was funded by a group of end users, particularly Statoilhydro AS, to carry out a thorough review and audit of the complete rebreather.

Members of the Master Review team participated in some of the HAZOP reviews carried out at key junctures in the project.

19.5 Accredited Body Review

Mark Dawes at SGS Ltd in the UK is UKAS accredited as a Notified Body for Complex PPE, and was commissioned by Deep Life to act as a Notified Body in respect of this equipment. The Notified Body has inspected the documentation, methods and will continue to be involved in audit of the equipment design, function and performance, and compliance with all standards that are claimed in the Technical File.

SIRA Certification Ltd is UKAS accredited to audit and certify compliance to EN61508 under the CASS Scheme. Deep Life has commissioned SIRA Certification to inspect and audit the life cycle processes used by Deep Life for the management of Safety Critical Products to SIL 4: the most onerous SIL level. This project is used as the primary example safety project in those audits.

20 EDA TOOLS

20.1 Specification development

Matlab and Simulink are used to describe all aspects of the specification: that is the product and its environment.

That environment and specification is applied throughout the design process to ensure the correctness of specification interpretations, and the implementations that are produced from them.

20.2 Code Modelling

All firmware code is described in Verilog and synthesised, with simulation of extracted (back annotated) netlists.

All software code is described using UML using a UML Add-on to MS-Visio.

20.3 Automatic Code Generation

All firmware code is generated automatically using Synopsys and Mentor synthesis tools.

20.4 Formal Equivalence Proving

There is inadequate FPGA data to support extraction and formal equivalence proof (layout to Verilog source), using tools such as Cadence Formality: the FPGA manufacturers do not release the FPGA layout at a silicon level. However, extraction of the netlist does allow a full layout versus schematic check to be carried out at a netlist level and this was performed.

20.5 Test generation and coverage

The coverage of test sets is measured using appropriate EDA tools: Mentor Graphics and Synopsys ASIC design toolkits for the firmware and compiler trace analysis for the software.

The firmware design team have extensive experience of using this code, with success in its application in ASIC design where many millions of gates have to work first time.

The software element of the system that makes up the independent monitoring function is considerably easier to debug, and with far greater coverage, than conventional code because it uses TTA: it can be treated as a set of completely isolated modules.

21 VERIFIED COMPILER

Whilst a fully verified ARM 7 processor was used, there was no fully verified compiler. The second route was adopted, namely a stable Open Source compiler, that has been subject to partial formal verification, and has been used very extensively.

The output of the compiler was verified using decision control equivalence checking to the specification, by Monte Carlo execution of each as well as by executing both using carefully considered test plans.

22 DEFENSIVE PROGRAMMING

Defensive programming is used throughout the software codes using for safety monitoring functions.

23 STATE AND TRUTH TABLES

State diagrams exist for all firmware, as well as timing diagrams. These are recorded on SVN, and in the Green Book for the firmware functionality.

Truth tables are an implementation level description, prone to error. They are not used in the primary design process: Verilog HDL code is used, with simulation. However, truth tables are used widely in the code verification process to assess coverage and accuracy of the test suite.

UML State diagrams exist for all safety functions implemented in software (monitoring functions.

24 CASS CHECK LISTS

The project was used to test the CASS Software Templates, for EN61508 compliance. These templates are designed to provide coverage to SIL 2 level, though may be applied to higher SIL level systems as a basic safety check.

A full compliance matrix with the CASS Software Templates has been produced for the project[8], and found to be fully compliant. This will be audited by SIRA and by members of the 61508 Association Software Working Group.

25 NASA GUIDELINES AND CHECKLIST

The NASA Guidelines[9] were the most comprehensive guide to Safety Critical Software that was used in this project. These comprise a formal method that is well structured and

[8] Document "CASS 61508 compliance DL_A1_080511.pdf" issued for independent review on 11th May 2008.

comprehensive. It is supported with extensive checklists. Unlike CASS, it is highly prescriptive, setting down the state of the art for best practice software safety management.

Each applicable section of the NASA Guidelines is considered.

25.1 Chapter 2 Recommendations

25.1.1 Chapter 2.1: Hazardous and Safety Critical Software

The firmware and the software of this project is safety critical software in the meaning of the NASA Guidelines.

The software controls the hazards, by providing monitoring functions and by intervention in certain fault modes to maintain life support where without that intervention the user would die, such as from a lack of oxygen.

In Section 2.1.2 of the NASA Guidelines, consideration is given to how the software itself can be hazardous. In this application the software can be hazardous by both failing to carry out an intervention action, and by activating an intervention action at an inappropriate time. These hazards are controlled by layering and by redundancy, where a higher layer in firmware (FPGA Verilog), does not permit the software to move the most critical parameters outside safe limits, and provides intervention in the case where the software fails to provide the functionality to keep the life support system within safe limits. In this sense, a relationship has been established in accord with NASA Guideline 2.1.5 between software and hardware to maintain a safe operating envelope.

The fault tolerance of the system has been considered in accord with Section 2.1.7 of the NASA Guideline, the system is designed to withstand two critical failures, and the MTBCF for those failures have been calculated and exposed to peer review by open publication.

25.1.2 Chapter 2.2: The System Safety Program (*sic)*

A full Hazard Analysis has been carried out and published. This comprises:

1. A bottom up analysis forming FMECA Volumes 2, 3 and 4
2. A top down analysis forming FMECA Volumes 6, 7 (a Fault Tree Analysis of the same)
3. An interface analysis forming FMECA Volume 8

The process described in Figure 2.1 of the NASA Guidelines has been applied, and continues to be applied with accident investigation and analysis.

The "Safety Plan Programme" is in the form of Deep Life Quality Procedures QP20 to QP25.

The independence of the System Safety Processes in Section 2.2.1 is described above, in Section 19 of this document.

Concurrent engineering has been applied throughout the project.

The timeline in Figure 2.2 of the NASA Guidelines is incorporated in and expanded upon in Deep Life Quality Procedure QP-20 which has been applied to this project: the project has been run in strict accord with that procedure.

[9] NASA Software Safety Guidebook, NASA-GB-8719.13, 31st March 2004

25.1.3 Chapter 2.3: Safety Requirements and Analysis

Considerable effort was expended in understanding the safety requirements. The top down FMECA (Volumes 6 and 7 of this series) is the most comprehensive FMECA that is reported to have been developed for a piece of diving equipment. In addition to that, the project was the first to collate all known rebreather accidents, publish a database of these, and a causal analysis. The project has led to a persistent rebreather accident group being formed, and this moving to an appropriate management body.

The risks listed in Section 5 have been assessed using the levels set down in EN61508, with a formal SIL assessment. The EN61508 risk levels differ from the NASA Guideline levels, and cover risk, severity and frequency rather than just risk. A translation between the levels for these two standards as it relates to this project is set down below:

SIL 4 = NASA Risk Level 2 to 4. The rebreather function is NASA Risk Level 3

SIL 2 = NASA Risk Level 6. The communications functions are NASA Risk Level 6

All hazards are eliminated where possible. Careful HAZOPs and safety reviews have been carried out to determine that this is the case.

Safety devices are incorporated where a risk has been identified.

Cautions and warnings are provided in the form of layered alarms, including voice announced alarms.

Administrative procedures and training have been developed and will continue to be developed for this system.

The following software safety analyses have been carried out:

- Fault Tree Analysis, forming Volume 7 of this FMECA.
- Software Failure Modes and Criticality Analysis, forming Section 7 of this document
- Specification analysis, forming a complete Matlab and Simulink model published on the web site of Deep Life Ltd, and a series of Colour Book Specifications for every major functional subsystem.

25.2 Chapter 3 Recommendations

25.2.1 Software Safety Planning

Software safety planning is managed in accord with Deep Life Quality Procedures QP-20 to QP-25. It encompasses all requirements identified in Chapter 3 of the NASA Guidelines.

The software identified as SIL 4 through a formal Safety Integrity Level assessment, corresponds to MIL-STD-882C category IA.

The software identified as SIL 2 through a formal Safety Integrity Level assessment, corresponds to MIL-STD-882C category IIB.

The top side software corresponds to MIL-STD-882C category IIIB.

The software risk matrix forms part of the SIL Assessment, using EN61508 nomenclature.

The FULL software safety effort has been applied to the whole of the firmware and software of the SIL 4 subsystem.

The MODERATE software safety effort has been applied to all other related firmware.

The LOW to MODERATE software safety effort has been applied to the top side software.

The Degree of Oversight requirements are met.

No COTS (off-the-shelf software) is used in any part of the system from the Top Side Unit to the Diver.

A full IV & V Evaluation has been carried out, as well as an IA review.

25.2.2 Software Development

No single event or action is allowed to initiate a potentially hazardous situation.

When an unsafe condition or command is detected, the system:

- Inhibits the potentially hazardous event sequence. An example is how the FPGA over-rides the MCU PPO2 optimisation if the PPO2 is outside safe limits.
- Initiates procedures or functions to bring the system to a predetermined safe state. An example is the interrogation of MCU and FPGA in the event of data being incorrect, and another example is the shut-down with the oxygen injector in a safe state upon sudden power loss.

A structured development environment is used for all development. This is assessed to ISO 9001:2000. All staff involved have received EN61508 training, using both external residential courses organised by specialist educational organisations, and by internal courses run by qualified staff that have held University lecturing positions teaching to MSc and PhD level. The project leader is a Fellow of the IEE.

All staff on the project are professionally qualified to Masters degree level in their respective field of specialism (i.e. software engineering, control engineering or electronics design engineering).

The entire software and firmware team have reviewed and have been involved with all pertinent specifications. These are maintained under source control using appropriate EDA tools.

25.3 Chapter 4 Recommendations

25.3.1 Lifecycle Models

Section 4.2.1 of the NASA Guidelines considers the different lifecycle models that may be applied.

A **Waterfall development model** was used as the primary development process, controlled by clearly defined phase reviews and documentation. This process is described in QP-20 and has been applied to all parts of this project.

Formal specification modelling followed by rapid prototyping, along with multiple feasibility studies and user focus groups were used during the early stages of the project to ensure the requirements were properly captured and understood, following by both unmanned and manned tested again with user focus group review of the developed product.

A **Spiral model** was used as the primary safety review process, overlapping the waterfall development with safety reviews, continuous risk analysis, accident studies, accident

analysis and requirement analysis. This led to major changes being introduced after the initial prototype was demonstrated, and after a batch of 5 units were demonstrated. All changes were applied retrospectively with total recall and update. In the application of the Spiral model in this way, it encompasses the benefits of the Evolutionary model but offers a faster execution of the project.

25.3.2 Development Models

Section 4.2.2 of the NASA Guidelines considers some of the different development techniques that are recommended. These are called methodologies for some reason in the NASA document: we stick to the plain English.

Traceability to customer requirements is embodied in the Colour Books that control each phase of the design, from capturing the market requirement through to the final implementation.

Of the development techniques that are listed, all have been applied: they are not exclusive. This is to be expected, as full safety verification effort was applied.

25.3.2.1 Structured Analysis and Design

Structured Analysis and Design was applied. The functional decomposition is described in the Colour Books, as is the Data flow (with TTA, software and firmware data flow is very simple), information modelling was performed both conventionally with a Unified Data Protocol being the outcome, and is covered by formal modelling. Structured Analysis and Design is folded into UML that was used for every functional module and operation. Evidence of this is in the Colour Books.

25.3.2.2 Object Oriented Design

Object oriented design was applied to all software development, regardless of whether C or C++ was used for coding each section. All software staff have very extensive experience using Object Oriented programming; it is not a case of applying just "book knowledge". Evidence of this is in the Colour Books.

25.3.2.3 Formal Methods

Formal methods were applied extensively, using mathematical modelling and formal logic to represent the safety specification, the environment and to model the implementation. Verification was by both module level and system level Monte Carlo methods, due to the signal space being unbounded. Evidence of this is in the Formal Models referred to in Section 13 of this document.

25.3.2.4 UML

All software is described using UML, and UML was the primary reference linking the formal models to the implementation. Evidence of this is in the Colour Books and extracts in Section 14 of this document.

25.3.2.5 Model Based Software Development

Model based software development was applied from the outset and encompasses the entire project. A formal method was used, referred to above, capturing the safety specification, the functional specification, the environment model and the implementation model.

25.3.3 Management

Section 4.3 of the NASA Guidelines considers some of the different development management processes that may be applied. The management process applied by Deep

Life is that documented in QP-20 to QP-25 in particular, and the whole of its Quality System in general.

25.3.4 Airlie Council Recommendations

The project management implements all of the steps recommended as Best Practice at the Airlie Council, an ambitious project which was established in 1992 by the Assistant Secretary of the Navy within of the U.S. Defence Department to identify proven industry and government software best practices. Namely:

1. Formal Risk Assessment is in the form of a formal SIL Assessment, listed in the Source Documents (Section 2).
2. Agreement on Interfaces is in the form of Colour Books, including the design and documentation of a Unified Data Protocol linking the major subsystems.
3. Formal inspections were carried out as listed in Section 19 of this document.
4. Metric Based Scheduling and Management was applied, using Merlin as the project projection and planning tool.
5. Quality Binary Gates at the Inch-Pebble Level was applied, as evidenced by the Design Verification reports that cover every subsystem that does not have its own Colour Book.
6. Programme Wide Visibility of Progress vs Plan was achieved by exporting Merlin plans to MS Project, and integrating these with the plans of the client.
7. Defect Tracking against Quality Targets is managed within the Design Verification processes. These were reviewed formally at the internal review management level, and provided to the client as well as key reports were published for the widest peer review.
8. Configuration Management is used, controlled by SVN source control.
9. People-aware Management Accountability has achieved a very low staff turnover throughout the project, and staffed the project with the highest level of expertise believed to be available for a project on this theme within any group. This team was supplemented by a client team that filled in missing skills, particularly those relating to the application and use of the system in saturation diving by bringing onto the project some of the most experienced saturation divers, dive supervisors and dive project managers in the industry.

25.3.5 Requirements

The requirement specifications were compiled from:

- Requirement analysis from experienced users
- Safety standards. The net of standards included within the scope of the project is as broad as reasonable possible to capture all the knowledge in those standards.
- FMECA Analysis of competitive equipment, including tests on that equipment.
- HAZID and HAZOP studies, such as summarised in FMECA Vol 6 in this series.
- Accident analyses, including the creating of an industry wide database of rebreather accidents and review of the 58 dive fatal accidents that have occurred since 1970 in the North Sea.
- Customer input, with regular reviews and user focus groups
- Generic system safety requirements.

- Best practice, encompassed in the Quality System and working procedures.
- Hardware and environmental constraints and interactions, including those identified using formal modelling processes.

The entire requirement management process described in Deep Life Quality Procedure QP-20 was applied.

25.4 Chapter 5 Recommendations

25.4.1 Design

The design process described in Deep Life Quality Procedures QP-20 to QP25 and QP5 to QP-8, were applied, as evidenced from the Colour Books and Safety Case documentation.

25.4.2 Implementation

The implementation process described in Deep Life Quality Procedures QP-20 to QP25 and QP5 to QP-8, were applied, as evidenced from the Colour Books and Safety Case documentation. This meets the recommendations in the NASA Guidelines Section 4.3.4.

25.4.3 Testing

Layered testing was applied: all sensory systems were tested, the modules were tested against the specification, as was the system, and the overall system was tested in conjunction with the client.

On top of this, an independent verification review was carried out to check the tests and the functionality, as well the standards compliance. The results of these tests, and the manned testing, was reviewed or witnessed by the client, and the client has appointed an independent contagonist group to examine the validity of key safety claims – very welcome move.

Evidence of this testing is in the form of Design Verification reports. This process meets the recommendations in the NASA Guidelines Section 4.3.5.

As "full" software effort was applied, the objective set was to test everything: this goes beyond the objective in the NASA Guidelines.

25.4.4 Products from the Development Process

The products from the development process are allocated in three groups:

1. Embedded software
    a. Base Unit MCU section software
2. Dive Supervisor software
    a. Daemon service
    b. Web server application code
    c. SQL server application code
3. Diagnostics and Test software utilities

The following sub-sections describe these products in more details.

25.4.4.1 Embedded Software: Base Unit MCU Section

This software is a core of the life critical control system. The code is developed using the "C" language. Several small time-critical routines are realised using assembler coding. The code mostly uses the OO principles, although being implemented in classic "C". The open source toolset GCC, version 4.2.2 has been used in the binary image builds.

25.4.4.2 Dive Supervisor Software: Daemon

This component implements the interface between Dive Supervisor client and the rebreather hardware. The Daemon is supported for two 32-bit x86 platforms: Windows and Linux. The code is developed using the C++ language only. The Microsoft C++ compiler toolset is used in builds for Windows platform. The GCC revision 4 is used for builds for Linux platform.

25.4.4.3 Dive Supervisor: Web Server Application Code

The primary role of the Web server code is in the management of the GUI at the client computer, and in accessing the telemetry data in the SQL database. The open source Web server – Apache - is used as a platform for this software component. The most of the application code is implemented on Python object-oriented language. The dynamic GUI objects have been designed Action script, running on the Macromedia Flash engine. These decisions provide the multi-platform implementation of the Web server: it may be installed on any platform supported by the Apache server.

25.4.4.4 Dive Supervisor: SQL Server Application Code

The SQL server stores all telemetry data supplied from the rebreather hardware, as well as control actions implemented by the supervisor. The open source SQL server – PostgreSQL – has been selected as a platform for this software component. The application code for the SQL server is implemented on the object oriented Python language. The SQL server is really a multi-platform implementation: it can be installed on any platform supported by the PostgreSQL server.

25.4.4.5 Diagnostics and Test Software Utilities

See section 18 for the description of these products.

25.4.5 Managing OO Projects

Conforming to strict software development technology rules is a main factor of the successful implementation of software OO projects. The application of these rules has been reviewed, and includes the following items:

- Implementation of colour books, according to Deep Life quality system, from the initial specification, to the specification of the technical decisions, and further to the specification of the software products, their features and user interface.
- Detailed definition of dependencies between software objects, their interfaces and functions; detailed definition and agreement of interfaces is a key to the effective implementation of even multi-component distributed software systems, like Dive Supervisor.
- Detailed definition of a role of each programmer within the software team; avoiding re-allocation of programmers to different software components effectively increases the quality of the software product
- Periodic review of the source code of each software component, in two models: by the project manager, and by team members (cross reviews)

25.4.6 Software Development Capability Framework

The software team and hardware team were both internal to a fully ISO 9001:2000 certified facility, certified for the development of hardware and associated software. The emphasis of that QA process is High Integrity Systems.

The QA Process and its application to this project were further audited by a specialist QA team from a lead client, Technip Norge AS.

The software processes are designed for compliance with all applicable European standards, particularly ISO/IEC 12207. This is a Software Capability Maturity Model, as evidenced in Deep Life's quality procedures.

25.4.7 Metrics

The metrics are documented in the Design Verification reports and in the Project Progress Reports (weekly, fortnightly and monthly depending on the phase of the project).

25.4.8 Software Configuration Management

SCM is managed using the SVN source control system. This provides identification, control, and links the documentation on the status accounting, audits and reviews as well as the specification and design documentation. An experienced Documentation Control Manager was appointed to the project.

25.4.9 Change Control

Change control is by ECOs and a Mantis controlled process.

25.4.10 Versioning

Version control is applied, with positive identification of all code and subsystems.

25.4.11 Status Accounting and Defect Tracking

Status accounting is primarily through the Mantis system, as is defect tracking.

25.4.12 Pitfalls for Real-Time Software Developers

The NASA Guidelines lists features that should **not** be present. Those listed in the NASA Guidelines are considered in turn below:

- Delays implemented as empty loops: a global table is used for very short loops where it is not possible to jump to the RTC, with a warning if the compiler version is changed.
- Interactive and incomplete test programs. All test programs are scripted, however there additionally a user diagnostic for access to logs and communication channels as part of the fault observation process and as a harness to call the test scripts.
- One big loop: There is no big loop, as TTA is used, with a task scheduler.
- No analysis of hardware peculiarities: there are detailed Design Verification reports on all hardware features.
- Fine-grain optimising during first implementation: The code is not optimised.

- Too many inter-component dependencies: the phrase "too many" is subjective, but in principle inter-dependencies are not permitted because communication is via a task scheduler.
- Only a single design diagram: there are UML diagrams for each operational mode, each function block and the overall system. Formal modelling uses Simulink diagrams, covering all function blocks and the environment.
- Error detection and handling is an afterthought: this is not the case, as TTA is used with a scheduler managing errors and processes as the foundation for the software.
- No memory analysis: a detailed memory map is in the software Green Book
- Documentation written after implementation: this is not the case as the full Colour Book set is available (Clear - requirement, White – functional specification, Blue – design proposal, Green - implementation).
- Indiscriminate use of interrupts: Not the case, as TTA is used so only the scheduler uses interrupts, and even then it is just a single interrupt source from a timer.
- No measurements of execution time: Not the case, as TTA demands knowledge of the expected timing and the TTA scheduler measures the time taken by each call of each task.

25.4.13 Software Risk Management Recommendations

The NASA Guidelines lists the following from Wood's "Software Risk Management for Medical Devices":

- Check variables for reasonableness before use: these are termed Assertions, and are applied immediately each primary subroutine is called. Sensors are all checked for reasonable values and sensor masking is applied for those outside acceptable ranges (A status word giving the confidence level for each sensor is maintained).
- Use execution logging: this is intrinsic to TTA.
- Come-from checks: this is managed by the TTA scheduler.
- Test for memory leakage: there can be no memory leakage in a TTA. The only memory allocated is the stack, which is cleaned up by the scheduler after each task finishes or is aborted.
- Use read-backs to check values: this is only appropriate for writes to remote devices: the MCU and FPGA only read and write to local memory and to local peripherals. All peripherals have read-back where the time constant allows (e.g. gas injector, via the Hall sensor).
- Use a simulator or ICE: an ICE is used, and a software simulator is also used.
- Reduce complexity: very simple (default in TTA).
- Design for weak coupling: this is forced by TTA.
- Consider stability of the requirements: the requirements are managed by the Colour Book process in accord with Deep Life QP-20.
- Consider compiler optimisation carefully: optimisation is switched off, deliberately.

- Careful using multi-threaded programs: TTA forces single threading.
- Dependency Graph: this is captured in UML.
- Two person rule. This is followed, and is natural when there are detailed team reviews.
- Prohibit program patches: No patches are permitted.
- Keep interface control documents up to date: this is done, with evidence from the Green Book revision dates for firmware, software and Unified Data Protocol documents.
- Create a list of possible hardware failures: this is documented in FMECA Vols 2 and 3.

The NASA Guidelines takes the following recommendations from SSP 50038, "Computer Based Control System Safety Requirements for the International Space Station Programme."

- Separate authorisation and separate control functions to initiate a critical or hazardous function. This is implemented by using the FPGA firmware to screen the MCU functions, and the MCU monitors the FPGA operation: i.e. cross-monitoring.
- Do not use I/O ports for both critical and non-critical functions. I/O ports are dedictated.
- Sufficient difference in addresses between critical and non-critical ports. These are well separated in the hardware.
- Ensure interrupt priorities and responses are defined. Interrupts are barred: TTA is applied, so the only interrupt use can be the TTA scheduler, and those interrupts are from a timer only. The complex inter-relationships that interrupts create in a real-time system are considered inappropriate for SIL3 and SIL 4 systems.
- Provide for orderly shutdown upon detection of unsafe conditions. Corrective action is taken to unsafe conditions, as is alarm annunciation and signalling. Power shutdown processes are active, and MCU or FPGA failure results in a controlled handover.
- Out-of-sequence transmission, it is prevented using TTA.
- Initialise all unused memory to a known pattern: all memory is initialised at start up.
- Hazardous sequences should not be initiated by a single keyboard entry. There is no user keyboard, or user intervention allowed, that can change the environment into an unsafe state. Attempts to set PPO2 for example outside the safe range, are rejected.
- Prevent inadvertent entry into a critical routine: this is a feature of TTA, and is managed by the scheduler.
- Don't use a stop or halt instruction: no stop instruction is used – the MCU only enters a short term sleep mode, waking up every 10s to check if the PPO2 status has changed, that would indicate a diver is breathing from the system, in which case it wakes up automatically and manages the breathing loop. If there is no breathing from the loop for 15 minutes, then the system powers down into the Sleep state, not to an off state.

- Put safety-critical operational software instructions in non-volatile read-only memory. All code is stored in non-volatile ROM, but this is FLASH memory, which becomes volatile after 10 years. The code does have CRC checks, which is checked periodically by the TTA scheduler.
- Don't use scratch files: there are no scratch files – there are no disks.
- Safety interlock bypass is not allowed by the TTA.
- Critical data communicated from one CPU to another should be verified prior to operational use: the transfers from MCU and FPGA are verified, and all transfers between functional units are verified using CRC.
- Dedicated status flags: flags are not used but the concept of dedicated resources is managed as this is intrinsic to the TTA scheduler.
- Verify critical commands: the user is not allowed to send any critical command.
- Ensure all flags are unique and single purpose: flags are not used as they are viewed by the software team as being hazardous.
- Put safety-critical decisions and algorithms in a single or a few software development components: a single application is used, with a single binary in the FPGA – there are no imports or external-links.
- Decision logic should not be based on all 1s or all 0s: full sensor fusion techniques are applied, with a sensor confidence figure calculated. All decisions are based on sensor readings: there are no stored states that lead to changes in decision paths – those are eliminated by TTA.
- Safety critical components should have only one entry point and one exit point: this is a feature of TTA, and is also intrinsic to the structured programming style that was used in code development.
- Perform reasonableness checks on all safety-critical inputs: this has been covered earlier, using the cross-checking between FPGA and MCU.
- Status check: this is handled using sensor confidence data, as well as self tests.
- Initialise the software into a known safe state: this is done.
- Don't allow the operator to change safety critical time limits in decision logic: the operator is not allowed to change any time limits.
- Provide the current system configuration to the operator: this is done, using logs and telemetry.
- Safety-critical routines should include "come-from" checks to verify that they are being called from a valid program or routine: this is intrinsic to the TTA scheduler.

26 CHECKLIST FROM QP24

The checklist below is taken directly from QP24 and applied to the OR project's. A tick mark means direct application and implementation of the requirements to the OR software and the evidence given for compliance against each section.

The check list is as follows:

26.1 Preparatory Phase

- ☑ The software project must identify clearly the hazards from any possible failure. This is normally part of the SIL Assessment, and will form a volume of the FMECA (a Top Down HAZID), and separate bottom up FMECA documents, but the FMECA for the software is considering any additional hazard that may be posed by the operation or failure of operation of the software: *current document FMECA vol.5*.
- ☑ The software developer has been familiar with the NASA Guide and this procedure, which has precedence. The training log should show evidence of that familiarity, including any refresher course. The Project Leader checks the training log and note that in the Master Log Book for that project: *recorded into the Master Log Book.*
- ☑ A SIL assessment has been carried out and the different modules assigned a SIL rating by using the process in QP-23 and this has been referred to in the Colour Books, FMECA and DV reports associated with the project: *recorded in the SIL assessment document SA_SIL_Assessment_070308.doc.*
- ☑ Confirm that the software program is required and that an ALARP solution is not available without a processor. Note that no project that is SIL 3 or above may depend on any number of processors of the same type: for SIL3 and above, the primary safety functions should be managed by an FPGA where at all possible with the software acting as a monitor for the FPGA or providing optimisation but not direct control unless the FPGA has failed: *see Colour Books for OR project.*
- ☑ Document how the software is to run on complies with each of the Processor Environment Requirements below: *full information is recorded into the Log Books.*

26.2 Processor Environment

All software and processors depend on an interface to the physical world. This interface is of overwhelming importance: if it fails, then the software fails. All software projects cover the following dependencies specifically in the FMECA and also in the electronics design:

26.2.1 Power Supplies

- ☑ All software runs on hardware with adequate brown out circuits on all power supplies connected to a component with a stored state: *all power supplies are equipped with brown out circuits as recorded in the Colour Books.*
- ☑ The power supplies has been designed and characterised so to have a clear demarcation where the supply operates and where it does not. The supply is not

permitted to have intermittent brown outs or noise at a level that is material to the operation of the system: *characterisation is recorded in the Colour Books.*

26.2.2 Brown Out Circuit

- ☑ The brown out circuit has been characterised for the trip voltage and the time required under that voltage, and this compared with the power supply regulator characteristics to ensure it will trigger by any brown out that can affect the contents of any register, bus or dynamic storage, holding a state: *the trip voltage and the time are recorded in logs and in the Colour Books.*

26.2.3 Watch Dog Timer

- ☑ All software runs on hardware with a watchdog timer with the watchdog interval not more than one ninth the interval between a failure occurring and the failure having potential safety critical consequences: *watchdog timer is implemented into the hardware.*
- ☑ Any such watchdog event has been flagged clearly and appropriate action taken automatically: *all events are recorded in the Colour Books and alarm matix.*

26.2.4 Electro-Magnetic Susceptibility

- ☑ The Electro-Magnetic Susceptibility (EMS) of the system has been tested in accord with European guidelines and regulations and adequate margins provided, appropriate to the environment in which the equipment works. Diligent enquiry or tests should be carried out to confirm the maximum EMS insult the system may be presented with: *there is a report on EMS testing of the system from independent testing laboratory to all applicable European and US standards.*

26.2.5 Clocks

When a clock fails, the processor fails.

- ☑ The clock circuit has been fully characterised, particularly for jitter, stability over a wider power supply range than the processor and for the absence of glitches: *the clock circuit was characterised and results are recorded in the Colour Books.*
- ☑ Where multiple processors are used, the clocks are independent. This was taken into account and implemented.

26.2.6 Reset Circuits

When a reset circuit fails, the processor again fails.

- ☑ The reset circuit has been fully characterised, particularly for jitter, stability over a wider power supply range than the processor and for the absence of glitches: *the jitter was measured under different set of power supply range.*

Power may be applied momentarily due to contact problems or the user switching the system on and off rapidly.

- ☑ The reset circuit operates correctly under all plausible conditions: *a set of conditions was defined in the Colour Books and reset circuit tested.*

26.2.7 CPU Self Test

If the CPU becomes damaged, then it cannot be relied upon.

- ☑ The software performs a CPU self check on power up. If the test fails, then the CPU is faulty and the software can move to a safe state: *CPU self test on power up is implemented and control functions are defined to handle the results.*

26.2.8 Bus Noise

When a processor has an external bus, if the data eye is closed at any time, then the processor will execute corrupted instructions or read/write corrupted data.

- ☑ To avoid this, all buses has been schmoo tested so the probability of synchronisation error over the bus is quantified: *the buses were schmoo tested with a different frequency steps.*

Careful distinction should be made between random jitter on the bus and the deterministic jitter.

The eye opening is not been less than the clock minus 40 sigma of the random jitter and 32 sigma of the deterministic jitter: this equates to a bus failure rate of10 sigma because the same jitter is on both edges of the clock, and on the data. The jitter has been from DC to at least 4 times the maximum clock frequency.

26.2.9 Hardware Integrity

- ☑ "Clocked" processors, meaning processors operating with a clock rate above that certified by the manufacturer of the processor are not been used in this safety system. Partial memories neither used other than in Flash and Hard Disk storage systems: *no "clocked" processors are used and manufacture's requirements are strictly followed.*

26.2.10 CRC

- ☑ The program memory contains a CRC code, and the first action that has been performed after start up is to check the entire program against the CRC code. If the CRC code check fails, the system should try and fail in a safe state and alert the user: *the CRC code is kept permanently in the memory and the program is checked against it at the start up.*

26.2.11 Memory Checks

- ☑ On startup, a March X, March Y and March C+ test of the RAM has been applied, and all programmable memory locations that are not used: *all March tests are implemented to the RAM at startup.*

26.2.12 "Empty" Memory

- ☑ Unused program memory locations has been filled with code to restart the program as an abnormal restart, similar to a watchdog timer: *the restarting code is used in all unused program memory locations.*

26.2.13 Removing Operating System Dependencies

- ☑ If it can be achieved within ALARP, then all operating system code has been eliminated and the software developed as an embedded application: *the software is implemented as embedded application.*

Where this is not practicable, then only completely open source operating systems may be used for safety critical software, such as embedded Linux, Debian Linux compiled with only

full release status code modules, or a RTOS certified to the SIL assigned to the project. All modules not required should be excluded from the Kernel compilation. This is done for Kernel.

- ☑ All drivers to safety critical hardware has been written by the project and not use standard drivers, such as standard USB drivers. The driver has been subject to special scrutiny: *a validation report exists for that driver. All drivers are not 'out of shelf' and were written by software team and validation recorded into the reports.*

Any closed code is not used in any safety critical application, such as Microsoft Windows products or Oracle databases. *That is strictly implemented.*

- ☑ The design minimises the safety-related part of the software. The smaller the safety related part the less likely it is to contain errors and the easier it should be to test: *a part of the software that is considered as the safety-related is reduced based on the ALARP principle.*

If the software contains both safety and non-safety parts, for example a display driver may be deemed a non safety part, the non-safety part should be treated with the same regard as the safety part. This may involve applying defensive techniques and integrity checks on the non-safety part. The intention of this is to ensure as far as practically possible that the non-safety part does not fail. This is because failure of the non-safety part may have a detrimental effect on the safety part.

- ☑ Where multiple safety functions are supported the design has been implemented to the rigour of the highest SIL level. This ensures that all safety functions are designed to the required maximum level: *software design is implemented to the highest possible SIL level.*

26.3 Methods not to be used

This is not a complete list.

- ☑ Delays implemented as program loops are not re-usable or portable. If they are used at all they are placed in a utilities module and not found in any other code: *no delays are implemented as program loops.*
- ☑ Interactive test are not used: *all tests are planned and scripted. Implemented.*
- ☑ There has been no reuse of code not intended for re-use. *Strictly followed.*
- ☑ One big loop. A single large loop forces all software to operate at the same rate. This is usually undesirable. An interrupt driven action architecture is likely to be more efficient but priorities must be assigned carefully to prevent a high rate of interrupts interfering with critical tasks: *Strictly followed.*
- ☑ Documentation written after implementation: the opposite approach is required: *the specification documents are written before any start of work and all other documents are written alongside of coding and testing.*
- ☑ Program patches are not allowed. *Strictly followed.*
- ☑ Hazardous sequences are not initiated by a single keyboard entry. *Strictly followed.*
- ☑ There are no stop or halt instructions. *Strictly followed.*

26.4 Specification of software safety requirements and integrity

- ☑ The software safety requirements have been recorded in the Colour Books for the project, with a SIL assigned and a concise description of the requirement. This references the SIL rating for the EUC, which in turn is a report showing how it was determined: *recorded in the SIL assessment document SA_SIL_Assessment_070308.doc*
- ☑ The safety requirements has been stated in the Colour Books, in the FMECA and also were coded into the formal model. *Implemented in the Colour Books, the FMECA and formal model.*
- ☑ The diagnostics requirements for each safety function has been defined: recorded in the Colour Books.
- ☑ The specification requirement describes unequivocally the problem domain of the system and the required safety functionality and integrity: *recorded into the Colour Books, by formal modelling and by UML.*
- ☑ The safety function requirements should fully specify the required system functionality, including timing criticality, failure modes, inputs and responses. The safety integrity requirements should fully specify how the safety function is to be maintained and how the software handles internal and external conditions that compromise maintaining the safety function. This information should be in the Colour Book and in the formal model: *recorded into the Colour Books and implemented in the formal model, and by UML.*
- ☑ Each requirement of the system identified during requirements elicitation and analysis is precisely stated once in the Colour Book and also was in a formal specification. In the Colour Book, clear and consistent language has been used, including the use of definitive words, such as 'shall': *recorded into the Colour Books according to the QA requirements.*
- ☑ The requirement specified should allow for traceability throughout the development and should be individually testable, by the creation of a suitable environment around the formal model: *Implemented.*
- ☑ The processor environment requirements in the previous subsection of this procedure, has been addressed specifically in each of the software Colour Books: *Strictly followed.*

26.5 Formal Modelling: Imperative for Safety Critical Software

26.5.1 Specification Modelling

A formal specification model exists for all safety critical software projects. This is intrinsic to QP-20, and applies to software just as much as it does for hardware or Verilog.

Where the inputs to the software are completely discrete then Object Z is considered the most appropriate language, though a case can be made for using Matlab and Simulink.

Where inputs to the software are continuous, such as from analogue sensors, the most appropriate modelling tool are Matlab and Simulink. The Matlab model describes the whole of the environment, including primary fault modes, as well as the function of the hardware and software using blockwise refinement to enable every software module or hardware module to be verified using Monte Carlo methods.

Notes: The Matlab model is used for formal modelling. It is regularly updated and supported by independent formal modelling/testing team.

Software for the implementation shipped to the client or with the product has not included any software that is generated automatically from the Matlab. Strictly followed.

The Matlab team are not involved in software development for the application: they are to be an independent review team. They transmit requirements to the software team and review the result, including examination of UML and testing of the embedded code. *Strictly followed*.

26.5.2 UML

All software modules for a safety critical system are described using UML.

It is hazardous to generate code directly from UML. UML has been used to describe the requirement and to guide the development of the test bench around the software.

The use of UML and Matlab has been described in the FMECA for the software. *Implemented and described in the current document.*

26.6 Software safety requirements reviews

- ☑ The review of the requirements has been minuted in the Log Books of each of the software developer(s) responsible for implementing them, with notes of the reviewers from outside the development group who were present. *Strictly followed, see the Log Books.*
- ☑ There are a variety of review methods and an appropriate review technique should be employed for the project. The technique should be appropriate to the SIL level. For SIL 1 a peer review is likely to be sufficient. For higher level SILs structured review techniques has been employed, including the results of formal verification of the requirements and their implementation: *Independent review is used by software team members and team leader.*
- ☑ The review should be planned and the reviewers specified within this plan. The results of the review should be documented along with any actions raised. The names of the developers who reviewed the safety requirements specification should include those who design and develop the source code and at least one person from outside that department: *followed, see the Log Books.*

26.7 Software Configuration Management (SCM)

It is very unlikely that “safe” software can be produced without SCM. This should be managed within QP-06. Note that the scope of SCM is much wider than simply controlling version numbers, so has been applied in the widest context of specifications, formal design documents, source code, builds, scripts, formal models, verification plans, test benches, design verification reports, bug tracking and packages.

- ☑ The SCM process meets the requirements of the NASA Guidebook, Section 4.5. That Guidebook has been reviewed as part of the software review. Implemented by using the SVN system for software development. *The NASA Guidebook requirements review is recorded in the current document.*

26.8 Software safety requirements traceability

QP-06 requires that all software development be traceable. For safety critical systems, that traceability extends to the software safety requirements.

- ☑ All software safety documents and formal models has been controlled using the SVN process defined in QP-06: *Strictly followed.*
- ☑ The traceability from the final documents, analysis and development back to the original requirements elicitation and specification phases has been to the Colour Book system of QP-20, using the reverse date codes that apply to all project documentation (See QP-05): *recorded into the Colour Books.*
- ☑ Each safety requirement has been contained in a formal fault model that is used to verify the project. Those formal models has been uniquely identified and traceable throughout the software development: *implemented in the system formal model.*

26.9 Validation planning

A Validation Plan has been created for all software modules, in the same manner as a Design Test Plan then Verification Plan is created for all safety critical hardware modules in QP-20.

The software validation plan should describe the validation strategy (not the validation methods) and the validation activity considerations. The validation plan should provide evidence that the validation activity has been determined in advance of the validation process and that the planned validation is sufficient to validate the software safety functionality and integrity to the SIL assigned to the project. The Validation Plan then becomes a DV report with the method and results of implementing that plan: implemented through the DV reports for each software module.

The software validation plan contains the following information:

- ☑ When the validation is going to be performed: *implemented*.
- ☑ Who is going to perform the validation task and their competencies: *implemented*.
- ☑ Where the validation is to take place: *implemented*
- ☑ The validation strategy has been considered and evidence that it is adequate: *implemented*.
- ☑ Policies relating to the validation (corporate, industrial, legislative, health and safety etc): *implemented*.
- ☑ A pass / fail criteria has been determined for the validation: *implemented*.
- ☑ The memory analysis: how much, and ensure no memory leaks: *implemented*.
- ☑ The execution time should be fully defined: *implemented*.

As a minimum the persons conducting the validation should differ from those who were responsible for its development. The validation activity has been performed by a separate department or institution.

For high SIL applications, a layered review process has been used, with independent teams, outside experts, and protagonists in other organisations. Where reasonably possible, public review should be sought, with open publication of key safety data: all OR system information is intended to be openly published for public review.

The validation plan has been reviewed in the same manner as a Safety Case in QP-20, that is, including client representatives and independent reviewers: representatives from customer were included into a review team.

26.10 Programming tools

All software has been considered as flawed: including compilers and design tools. The validation plan should ensure that a loop exists to verify the output of any tool. This said, some software is more flawed than others, and the rules below has been applied in selecting programming tools.

- ☑ The programming tools has been open, or has been verified, or there has been a closed loop verification such as is used in Verilog hardware design (with synthesis of gate level code from RTL, then use of tools such as Synopsys Formality to prove the equivalence between the extract netlist and the RTL). *Implemented*.
- ☑ There has been evidence that the programming tools are suitable and have a proven in use history of reliability. *Implemented and based on software team experience.*
- ☑ The code editors used support syntax highlighting and enforce compliancy to an industry standard e.g. ANSI. Code analysers are used that enforce the standard strictly, such as Lint: *implemented.*
- ☑ There has been evidence that the manufacturers of the programming tool have listed all known issues with the programming tool and have provided evidence of proven in use history. When there are known issues with the programming tool there should be evidence that appropriate 'work arounds' have been adopted by the developers: *implemented*.
- ☑ When the development tools support multiple levels of warning the maximum level has been selected for the development of safety systems: *implemented*.
- ☑ Furthermore there should be evidence that no warnings or errors are produced when the code is built. Where warnings exist there has been justification for these: *implemented*.
- ☑ When the development tool supports optimisation it should be disabled. Optimisers have the habit of removing or altering the operation of code that appears to have either no function or a more efficient function. This code could be a trap or defensive programming measure: *optimisation is not used.*

26.11 Reuse of design components

- ☑ The reuse of design components has been considered and logged: *implemented and recorded into the Colour Books.*

There have been many fatal accidents caused by reuse of software. An example is the Therac-25 X-ray facility, where software was reused but the mechanical interlocks that had rendered it safe previously were missing in the new application.

When software is reused, the same rigour in verifying it should be applied as to new software.

Ideally domain analysis should be performed during the design process. This activity should identify which parts of the design already exist in some form of component. Where components are identified that exist, have test evidence, proven in use history and can be used without modification, the design should incorporate these. These components should be incorporated into the design architecture during the design phase and fully verified even if they were verified previously.

The reuse of design components may also include frameworks and patterns that have been previously used and proven.

26.12 Testability of design

- ☑ The EUC has been designed to be tested: *implemented from specification*.
- ☑ The design allows functional and data items to be identified, isolated and tested: *Strictly followed.*
- ☑ The test coverage has been 100%, verified using tools to show that each command has executed: *implemented*.
- ☑ The design allows system behaviour to be tested. If the design is traceable to the software safety requirements then it will be easier to test: *implemented*.

26.13 Design method

The method stipulated in QP-20 is that there has been:

- ☑ A formal specification and a English specification written in precise language: *recorded into the Colour Books.*
- ☑ Formal verification or Monte Carlo formal validation against the formal model: *implemented*.
- ☑ Code has been written manually and verified. **QP-20 does not allow code to be generated automatically from the formal model.** That is, **the formal specification tools can generate C Code and Verilog. That facility is NOT to be used**, as otherwise there is no independence between the formal model and the implementation: there would be no opportunity to find an error in the formal specification itself. *Strictly followed.*
- ☑ An object orientated programming has been used, regardless of the programming language: *implemented*.

The design methods use the NASA Software Safety Guidebook, document NASA-GB-8719.13, with the following exceptions:

Within the NASA Guide context, formal verification is documented widely. References [7] and [8] from QP24 should be familiar to all developers working on the project. Within the context of QP-20, Monte Carlo testing of a formal model, usually in Matlab, against the actual code is the preferred method. However, for high SIL rated software, theorem proving methods should be considered as an additional safeguard to errors from compiler tools.

26.14 Architectural Design

- ☑ The software design has been described using the Colour Book system in QP-20: *recorded into the Colour Books.*
- ☑ At a Blue Book level, the architectural description should capture the software developer's solution to abstracting functionality, information representation and system behaviour from the software safety requirements: *implemented*.
- ☑ An appropriate method has been used to obtain a design architecture that is structured and cohesive: *a water fall method is used to the software development.*

- ☑ The design architecture provides an overall blueprint for the software whilst providing increasing levels of abstraction that eventually facilitate coding of the design: *implemented*.
- ☑ The design architecture documentation enables communications between all stakeholders interested in the development of the system: *reflected in the Colour Books.*
- ☑ It also allows early design decisions to be highlighted that have a profound impact on the software development and the required safety function: *see the Log Books.*

26.15 Detailed design and development

- ☑ **At least two people has been intimately familiar with all code:** this may be the developer and the software project leader if the software project leader meets with the developer daily and reviews the code written in enough detail.
- ☑ The development engineer places a dated and signed entry into his Log Book that he has **examined the appropriate Colour Book** giving the requirement and safety specification of the system or module to be designed.
- ☑ The development engineer places a dated and signed entry into his log book that he has **examined and agrees with the SIL Assignment** for the project: any differences has been resolved by discussion with the overall Project Leader.
- ☑ The detailed design of the software of the safety system has been a solution that is **modular and structured. T***his recommendation is strictly followed.*
- ☑ The use of **software interlocks to prevent hazardous sequences executing inappropriately**, has been used where possible.
- ☑ **All input parameters have predicate checks applied**, and this is especially the case with data communicating from one processor to another. *Notes: no communication exists between processors.*
- ☑ There should be **strict error checking**: *implemented*.
- ☑ Provide **for an orderly shut down into a safe state on detection of unsafe conditions.** The system can revert to a known, predictable and safe condition on detecting any anomaly: *strictly followed.*
- ☑ Prevent **inadvertent entry into a critical routine.** Detect if such entry occurs and revert to a safe state: *implemented*.
- ☑ **The design should be appropriately decomposed** (functionally / objects) at the Green Book stage to an appropriate level of abstraction that readily facilitates the generation of code: *implemented*.
- ☑ **The design produces modules / objects that have minimum coupling and maximum cohesion.**
- ☑ **Credit has been given whenever a CASE tool has been used to generate the design.** CASE tools automatically check the design to some level. Note CASE tools are not permitted to be run from the formal specification to generate the application automatically other than for verification purposes. As the CASE tools are commercial software and has been limited in use in safety critical systems,

then the tools used only for testing and verification purposes when a project leader considers it as useful and appropriate.

- ☑ **The software should be designed to facilitate testing and easy modification.** The design should lead to the creation of structured code that is easy to understand by other engineers.

26.16 Code Implementation

- ☑ All source code has been written in accordance with the coding standard in QP-06. *That is strictly followed.*
- ☑ The code has been reviewed independently of the author for compliance with the coding standard and good coding style. *Implemented with records in the Log Books.*
- ☑ Each source code module has been reviewed and a Green Book report written describing its structure. The review ensures that the code implements the requirements specified in safety requirements specifications. *Implemented by periodical reviews by project leader.*
- ☑ There has been evidence that the code has been approved post review and that the configuration management system indicates this. This evidence could be a change in revision number / type of the modules or archiving of the reviewed modules into a controlled / managed system. *Implemented through the SVN system with keeping of all module revisions.*

26.17 Software safety validation

- ☑ Each software module has been validated prior to integration of that module to the software system: *strictly followed.*
- ☑ A test bench exists for every module and for the system level: *implemented with saving test results in the SVN.*
- ☑ The operation has been compared with the UML description of the module: *implemented.*
- ☑ Modules should be kept as short and simple as reasonably practical to enable thorough validation: *implemented.*
- ☑ When software has been validated (tested) in accordance with the software safety validation plan, the plan and the results form a Design Validation report as per QP-20: *implemented*. *See DV reports at the SVN.*
- ☑ There is a Design Validation report for each module and for the overall system: *see DV reports at the SVN.*

The validation activity documentation, for each safety function specified, includes:

- ☑ A date and time (chronological record) indicating when the test was performed in the Log Book of the engineer who performed the plan: *see DV reports at the SVN.*
- ☑ The safety validation plan used and any variations to the plan that may have been taken.
- ☑ Identification of the safety function being tested.

- ☑ A list of the tools and equipment used for the test, including serial numbers / asset numbers and calibration dates for the equipment: *see DV reports at the SVN.*
- ☑ The results of the validation activity: *see DV reports at the SVN.*
- ☑ Any discrepancies between the expected and actual result, with any justifications or non-compliance reports: *see DV reports at the SVN.*

26.18 Recording of Safety Requirements being Met

- ☑ Safety requirements have a unique identification in UML (name or reference): *implemented through specification.*
- ☑ At each verification phase it has been possible to check that the uniquely identified requirement has been addressed i.e. that is has been designed into the system / module, has been coded and has been tested: *implemented*.
- ☑ The verification activity ensures that the inputs to the development phase have been satisfactorily addressed and that corresponding outputs exist that allow the next development phase to be accomplished: *see DV reports at the SVN.*
- ☑ The verification evidence should show that checks have been made to look for inconsistencies between the software design and the software safety requirements specification and software test specification: *see DV reports at the SVN.*

26.19 Modification of design

- ☑ The design facilitates easy modification: *implemented by applying module design and OOP methods.*
- ☑ The design is therefore be structured and modular: *implemented by applying module design and OOP methods.*
- ☑ The change control process in QP-06 has been applied to all changes: this requires ECOs and active bug logging. *Implemented through using of the SVN system.*

27 METHODS USED ARE APPROPRIATE TO SIL

The table below is taken from QP24, based on NASA Software Safety Guidebook NASA-GB-8719.13. This is additional to the EN61508 requirements that are addressed using the CASS Templates: See the CASS Compliance Matrix for this product. In general, the NASA Guidelines are a superset of the EN61508 requirements, and they address high SIL applications: CASS Software Templates presently cover only to SIL 2.

The methods used in the table below use the water-fall software life-cycle starting from requirements, moving through software design, implementation, testing, operations and maintenance. The sign ☑ indicates application of the method and references are made to a specific documentation with give evidence of their implementation. Where is a blank square it means the method has not been applied, for reasons described in the comments.

Method	NASA-GB-8719.13		Applied ☑ - Yes	Comments and Cross reference to evidence in this document (FMECA V5). Links are in italics.
	Section	Page		
Requirements Management Requirements Specification	6.4 6.4.1	107 107	☑	*Safety Requirements* *Chapter 2.3: Safety Requirements and Analysis* *Software Safety Planning* The details are in the Colour Books for the project.
Requirements Traceability and Verification	6.4.2	111	☑	*Chapter 2.3: Safety Requirements and Analysis*
Requirements Change Management	6.4.3	112	☑	*Chapter 2.3: Safety Requirements and Analysis* Implemented in accord with QP05 Design Control through Colour Books.
Development of SW Safety Requirements Generic SW Safety Requirements	6.5. 6.5.1	113 113	☑	*Safety Requirements* *Chapter 2.3: Safety Requirements and Analysis*
Fault and Failure Tolerance	6.5.2	114	☑	*Chapter 2.3: Safety Requirements and Analysis* *Software Failure Modes, Effects and Criticality Analysis (SFMECA)*
Hazardous Commands	6.5.3	115	☑	*Software Failure Modes, Effects and Criticality Analysis (SFMECA)* *Chapter 2.1: Hazardous and Safety Critical Software*
Timing, Sizing and Throughput Considerations, including Time to Criticality	6.5.4	116	☑	*Software Failure Modes, Effects and Criticality Analysis (SFMECA)* *Chapter 2.1: Hazardous and Safety Critical Software*
Formal Inspections of Software Requirements	6.5.5	117	☑	*Chapter 2.3: Safety Requirements and Analysis* *Software Failure Modes, Effects and Criticality Analysis (SFMECA)*
Test Planning	6.5.6	119	☑	*Specification Modelling* *Module Verification* *Software Safety Planning*
Checklist and cross references	6.5.7	119	☑	*Checklist from QP24*
SW Safety	6.6	120		

Requirements Analysis SW Safety Requirements Flow-down Analysis	6.6.1	120	☑	*Safety Requirements* *Formal Verification* *Chapter 2.3: Safety Requirements and Analysis*
Requirements Criticality Analysis	6.6.2	121	☑	*Safety Requirements* *Software Failure Modes, Effects and Criticality Analysis (SFMECA)*
Specification Analysis	6.6.3	124	☑	*Formal Verification* *Software Failure Modes, Effects and Criticality Analysis (SFMECA)*
Formal Methods – Specification Development	6.6.4	126	☑	*Formal Verification* *UML*
Model Checking	6.6.5	127	☑	*Formal Verification*
Timing, Throughput and Sizing Analysis	6.6.6	129	☑	*Specification Modelling*
SW Fault Tree Analysis	6.6.7	130	☑	*Safety Requirements* *See also details in the Fault Tree Analysis in FMECA Volume 7*
SW Failure Modes and Effects Analysis	6.6.8	131	☑	*Software Failure Modes, Effects and Criticality Analysis (SFMECA)*
Design of Safety Critical Software Language Restrictions and Coding Standards	7.4 7.4.5	137 145	☑	Done according to the procedure QP07 which sets detailed Coding Style requirements and restrictions; *Foundations*
Defensive Programming	7.4.6	146	☑	*Defensive Programming*
Reducing the Complexity of SW	7.4.8	148	☑	*Safety Architecture* Done in accord to QP07 Coding Style requirements.
Design Analysis Update Previous Analyses	7.5 7.5.1	153 154	☑	Done according to the procedure QP06 Software Specific QA Procedures: 6.4 Software review *Reviews* *Formal Verification*
Design Safety Analysis	7.5.2	156	☑	*Safety Requirements* *Formal Verification*
Independence Analysis	7.5.3	157	☑	Done according to the procedure QP06 Software Specific QA Procedures: 6.4 Software review *Independent Review*
Formal Inspections of Design Products	7.5.4	158	☑	*Formal Verification* *Independent Review*
Design Logic	7.5.6	159	☑	*Code Modelling*

Analysis (DLA)				
Design Data Analysis	7.5.7	159	☑	*Reviews*
Design Interface Analysis	7.5.8	159	☑	*Safety Architecture* *Formal Verification*
Design Traceability Analysis	7.5.9	160	☑	*Module Verification* *Formal Verification*
Software Element Analysis	7.5.10	161	☑	*Module Verification*
Rate Monotonic Analysis (RMA)	7.5.11	162	☑	*Specification Modelling* *Formal Verification*
Dynamic Flowgraph Analysis	7.5.12	162	☑	*Activity Diagram: CO2 Sensor Calibration* *Activity Diagram: Sleep Detection* *Activity Diagram: Injector Control*
Markov Modelling	7.5.13	163	☑	*Specification Modelling* *Formal Verification*
Requirements State Machines (RMS)	7.5.14	163	☑	*Logic Modelling: Processors and State Machines*
Software Development Techniques	8.4			Done according to the procedure QP06 Software Specific QA Procedures
Coding Checklists and Standards	8.4.1	168	☑	Done according to the procedure QP07 Coding Style requirements
Unit Level Testing	8.4.2	169	☑	*Module Verification*
Programme Slicing	8.4.4	171	☑	*Time Triggered Architecture Implementation*
Code Analysis	8.5			
Code Logic Analysis	8.5.1	171	☑	Done according to the procedures QP07 Coding Style requirements and QP06 Software Specific QA Procedures: 6.4 Software review *Team Review* *Automatic Code Generation*
Code Data Analysis	8.5.2	172	☑	Done according to the procedure QP06 Software Specific QA Procedures: 6.4 Software review *See also Log Books records.* *Automatic Code Generation*
Code Interface Analysis	8.5.3	172	☑	Done according to the procedure QP06 Software Specific QA Procedures: 6.4 Software review *See also Log Books records.* *Formal Equivalence Proving*
Unused Code Analysis	8.5.4	173	☑	Done according to the procedures QP07 Coding Style requirements and QP06 Software Specific QA Procedures: 6.4 Software review See also Log Books records.
Interrupt Analysis	8.5.5	173	☑	Done according to the procedure QP06 Software Specific QA Procedures: 6.4 Software review

				See also Log Books records. *Formal Equivalence Proving*
Test Coverage Analysis	8.5.6	174	☑	Done according to the procedure QP06 Software Specific QA Procedures: 6.4 Software revie*w* See also Log Books records. *Test generation and coverage*
Formal Inspection of Source Code	8.5.7	175	☑	Done according to the procedure QP06 Software Specific QA Procedures: 6.4 Software review See also Log Books records.
Final Timing, Throughput and Sizing Analysis	8.5.9	176	☑	Done according to the procedure QP06 Software Specific QA Procedures: 6.4 Software review See also Log Books records. *Formal Equivalence Proving*
SW Integration and Testing	9.4	183		
Testing Techniques and Considerations	9.4.1	183	☑	According to the procedure QP06 Software Specific QA Procedures; *Code Modelling* *Test generation and coverage*
Testing Environment	9.4.2	188	☑	*Test generation and coverage*
Integration Testing	9.4.3	189	☑	*Test generation and coverage* Done in accord with QP06 Software Specific QA Procedures. See test reports on the SVN.
Integrating Object-Oriented Software	9.4.4	189	☑	Done in accord with QP06 Software Specific QA Procedures. See test reports on the SVN.
System Testing	9.4.5	190	☑	*Test generation and coverage* Done in accord with QP06 Software Specific QA Procedures. See test reports on the SVN.
Regression Testing	9.4.6	192	☑	*Test generation and coverage* Done in accord with QP06 Software Specific QA Procedures. See test reports on the SVN.
Software Safety Testing	9.4.7	193	☑	*Test generation and coverage* *Safety Architecture* *Diagnostic Utilities and Logging*
Test Witnessing	9.4.8	193	☑	Done according to the procedure QP06 Software Specific QA Procedures;
Use of Computer Models for System Verification	9.4.9	194	☑	*Formal Verification* *UML* *EDA Tools*
Test Analysis	9.5	194		
Test Coverage Analysis	9.5.1	195	☑	Done according to the procedure QP06 Software Specific QA Procedures; 6.6 Release Testing
Formal Inspection	9.5.2	196	☑	Done according to the procedure QP06 Software

of Test Plan and Procedures				Specific QA Procedures;
Reliability Modelling	9.5.3	196	☑	*Formal Verification* *UML*
Test Results Analysis	9.5.4	197	☑	Done according to the procedure QP06 Software Specific QA Procedures; 6.6 Release Testing
Operations and Maintenance COTS Software	10.5 10.5.1	201 201	□	No COTS Software is used in safety critical systems.
Software Change Impact Analysis	10.5.2	203	☑	Done according to the procedure QP06 Software Specific QA Procedures; *Master Review* *Accredited Body Review*

28 RELEASE TO 61508 ASSOCIATION AND PUBLIC

The section of the Safety Review Panel internal to DL, approved the release of this document to the EN61508 CASS Auditor, to the 61508 Association's CASS SoftwareTemplate team with all supporting documents, and this document itself with the templates to the Company's web site to enable open peer review.