Software Design Specification (SDS)

Monaco

Controls Group

Revision Number A

Author: Cornelius Smits

Tennant Company
701 North Lilac Drive
P.O. Box 1452
Minneapolis, Minnesota 55422

Revision History
Revision A

Change #	Description of Change	Source of Change	Date	Author
A	• Initial starting document		09/02/2016	CS
	•			

Table of Contents

1. Introduction

The System Requirements Specification (SRS) is generated to outline the system requirements. The System Design Specification (SDS, or this document) explains the details of the software implementation to achieve those requirements. New requirements pop up from time to time that may drive a new design. This SDS is the first to take on a new approach to document management.

First, this document and many others are stored and versioned within the NPD Share Point website. This allows for a common document name and yet still manage version control options.

The second and bigger change is the document is now divided into multiple documents. The SDS contains a lot of information regarding how the system works to document for the controls engineer. It became clear that there was large portions of information that was useful to other audiences. This included functional operation, fault codes and corresponding conditions as well as general specifications and machines states. Therefore, a new structure was created to make the relevant information more available to other cross functional groups. Not only do the documents save explanation time, they provide a conversation piece for feedback on operation design decisions that were may not have been requirements.

The documents are divided into two categories: functional guides and software documentation. These two folders reside within the Controls folder on share point for easy access.

1.1 0114 – Functional Guides

The following documents are grouped in this folder since they contain the most commonly requested information during development. By grouping these documents, a link to the folder can be sent out for others to find and locate as needed.

Documents	Description	Audience
Display Fault Guide	Defines the available fault codes, how they display and information a *customer* may do to fix the fault.	Customer testimonials Assembly Marketing Program Managers Tech Pubs Service (high level)
Controls Specifications	Defines all the measureable limits and thresholds for machine operation. Highly detailed information for reference.	Development Engineers Current Product Engineers Service Tech (manuals) Development Engineers Test Engineers
User Display Guide	Explains how the buttons and LED's or display functions operate. This includes blink patterns and special modes that are available to aid in user manuals and testing. Multiple of these files may be necessary for different user interfaces.	Tech Pubs Service Manual Development Engineers Current Product Engineers Test Engineers Program Managers
Operational Reference	This covers high level modes of operation.	Current Product Engineers Development Engineers Program Managers

1.2 0115 – Software Documentation

This folder contains information that is highly technical and not likely to need accessed regularly. Other subfolders exist within this folder to manage information neatly and appropriately.

Documents	Description	Audience
Software Design Spec	This document that begins as a starting place and links the other documents together.	Controls Engineer
Configuration Parameters	Defines the parameters that are saved in non-volatile memory. These parameters should be accessible using the Galileo tools.	Controls Engineer
Fault Table	This is an Excel document that expands in much greater detail the fault codes and other unexpected conditions. Details are provided for conditions to check that cause fault codes to aid in troubleshooting.	Controls Engineer Tech Docs Service Manufacturing
CAN Dictionary Folder	Provides a storage location for Excel files used to master the CAN dictionary objects for all modules in the project. EDS files are stored in Subversion.	Controls Engineer Test Fixture Engineer
Context Diagram Folder	Stores the software module relationship diagrams for each software project.	Software Engineer Engineering Manager Systems Engineer
Design Parameters Folder	Stores interface layer design behavior documents for each hardware device.	Hardware Engineer Software Engineer System Engineer

2. Monaco Scrubber Overview

The Monaco SRS outlines the system requirements. This document explains the details of the software implementation and key items of note during the project design process. The software design is modular to support component carry-over into new designs. Similar to previous software designs that were constructed via layers, this design will use a layered architecture to link the different modules together.

2.1 Firmware Development Standard

This project will be the designed according to the NPD guidelines in the Firmware Development Standard document:

Controls Software Tools\Documents\Firmware Development Standard.docx

2.2 Software Modularity Standard

The software is designed using the NPD defined layered modular architecture. This promotes clean, structured, modular, reusable, and reviewable code. The modular definition is in the following file.

Controls Software Tools\Documents\Modular Software Guidelines.docx

2.3 Version Control

All code shall be submitted to version control for traceability and saving. All attempts will be made to not check in broken or erroneous code.

2.4 Share Point

This program utilizes Microsoft SharePoint to save design documents under revision control. The current home for the Monaco project location is below. For best user experience, open the site with Internet Explorer to allow easy check-in/checkout experiences.

http://tcoweb/npd/Monaco

2.5 Compatibility

This is a new program with new control boards and does not require backwards compatibility with a previous design.

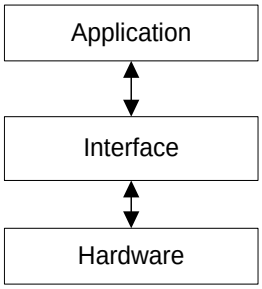
2.6 Source Code Documentation

All modules are written with Doxygen syntax commenting. A Rich Text Document and HTML output provide documentation for all the source code and is updated intermittently during development. It is able to map out calling functions and caller functions as well.

2.7 Context Diagrams

2.7.1 General

NPD software development follows a layered architecture. It decouples the hardware and application specific files such that the code is portable and reusable as outlined in the modularity document.



2.7.2 Layer Overview

This section describes modules in each layer. The interface files graphically show the include stack below each module for reference only. It may not be inclusive. Complete descriptions of each module are available as source code is developed and documented in the Doxygen outputs mentioned in Section 2.6.

2.7.2.1 *Hardware Layer*

This block contains all the hardware specific code needed to support the interface layer. The single hardware layer interface file (hw.h) is designed to be as simple as possible to perform the hardware function. If the interface is too complex, there may be functions defined that cannot be supported on another micro-controller. The implementation of the functions can do whatever is necessary to perform the operation since its complexity is hidden from the higher layers.

2.7.2.2 *Interface Layer*

This block contains interfaces that needed to support the application layer. The functions in the interface layer are not application specific but create components to be used by the application layer. The interface is not dependent on what microcontroller is used, because the hardware layer handles all the operation of the hardware.

2.7.2.3 *Application Layer*

This block contains all the code modules that are application specific. It knows about the components available within the interface layer and customizes the interface layer modules to behave in a manner that meets the design goals. Modules in this layer handle specific protocol data (such as CAN, USB, serial formats), and provide the connection between interface layers.

2.7.2.4 *RTOS*

A real-time operating system from Salvo is used to manage module tasks. This is the same OS used on the previous programs which is already configured to run on the Tiva line of micros making implementation straight forward.

3. Design Parameters

There are some limitations with the software design that put restrictions beyond the System Requirement Specification (SRS). These design parameters provide guidelines to remain compatible with previous design decisions or as a reminder of an implementation restriction.

Each interface layer that uses common hardware design contains certain design rules and guidelines for software development. Since low-side drivers, half bridges and H bridges are different, each interface layer rules are contained in their own document. Monaco contains the following common interfaces. The referenced specification value is located in the Operational Specifications section found later in this document.

Folder Location	/0115.3 - Design Parameters/
Actuator Control	/0115.3 - Design Parameters/Actuator Control Design Parameters.doc
Actuator	/0115.3 - Design Parameters/Actuator Design Parameters.doc
Detergent Pump	/0115.3 - Design Parameters/Detergent Pump Design Parameters.doc
Input Peripheral	/0115.3 - Design Parameters/Input Peripheral Design Parameters.doc
Scrub Motor	/0115.3 - Design Parameters/Scrub Motor Design Parameters.doc
Solution Flow	/0115.3 - Design Parameters/Solution Flow Design Parameters.doc
Vac Motor	/0115.3 - Design Parameters/Vac Motor Design Parameters.doc
Valve	/0115.3 - Design Parameters/Valve Design Parameters.doc

3.1 Module Code Revision

This section explains the version tracking of code changes to the overall module.

Parameter Number	Description
3.1.1	Code version tracking will be implemented as early as possible in the development.
3.1.2	Revision format shall be two bytes in length with a major dot minor.
3.1.3	Software Revision shall be obtainable from a CAN command from any module (including master).
3.1.4	Hardware Revision shall be queried from any module via CAN. (including Master)

3.2 Module Power Management

This section describes the power handling by the module.

Parameter Number	Description
3.2.1	Battery voltage for operation is determined locally.

3.3 Module Display and LED Operation

The module contains 7 LED's.

Parameter Number	Description
3.3.1	The heartbeat LED (D4) shall blink on for ½ second and off for ½ second when module has power.
3.3.2	The CAN LED blinks when the module is talking to a CAN master controller.
3.3.3	The module does not display any other user indication.

3.4 Module Serial Communication Ports

This section describes the serial communication ports.

Parameter Number	Description
3.4.1	The single serial port is for receiving diagnostic and test commands.
3.4.2	Serial port baud rate is at 115200 baud and configured 8-N-1.
3.4.3	An ASCII protocol is used by the monitor interface for commands.
3.4.4	A machine level USB communication shall support the following Galileo interface and protocols: • System • Dashboard • Control • Firmware • IDrive • Configuration

3.5 Memory and Data Storage

This section describes the non-volatile memory.

Parameter Number	Description
3.5.1	Serial flash memory is not present on the design.

4. Design

4.1 Theory of Operation

This module will be implemented in the 'C' programming language using TI's Code Composer IDE. The design is broken up into multiple modules according to the modular software design reference. The CANopen stack and RTOS are third party components. The Application Layer handles the combining of the interface Layer modules. The Hardware Layer consists of hardware specific routines within the micro.

4.2 Related Programs

The following subsystems are integrated into the Monaco scrubber. See the SDS links in the table for further information.

Subsystem	Program	SDS Link
Automated Battery Watering	Rhine ABW	ABW Module SDS.docx
ecNanoClean	Chimney Swift	Chimney Swift SDS.docx
Telemetry	Knight Rider	Knight Rider Module SDS.docx

4.3 Context Diagrams

This section expands on the general layers to depict the different modules required by this design. The black solid lines represent the breaks between the general application, interface and hardware layers. The diagrams are a starting point for software development and are subject to change.

As desirable as it is to have the entire design developed up front, it is just not possible. Therefore the detailed context diagrams for the individual firmware images is a living document outside this design specification. The links below provide quick access to the different diagrams for the machine modules.

4.3.1.1 *Boot-loader*

A special boot loader shall pull the firmware from the upper micro memory locations.

4.3.1.2 *Monaco Context Diagrams*

The latest version of the User Interface diagram is available in this file:
Monaco UI Context Diagram.vsd

The latest version of the Pascal LCD diagram is available in this file:
Monaco Pascal Context Diagram.vsd

The latest version of the Scrub Controller diagram is available in this file:
Monaco SC Context Diagram.vsd

4.4 Module Directory Structure

There are two major machine variants: base and standard (premium). The control systems are considerably different due to the standard machine contains an additional scrub controller board to provide electronic control of the motors. The base machine uses relays instead. The firmware design started out as two different projects but were able to be combined into one firmware file by using configuration parameters in firmware to handle the differences in operation.

4.4.1 Build Versions

There are three build options for most projects: Hardware, Bench and Production. Each build is set up in the IDE with a preceding "B-" to keep the project organized and easily identifiable as a build folder.

4.4.2 Hardware Build

The Hardware build focuses on the hardware folder and being able to exercise the hardware design with microcontroller control. There is likely a special monitor (terminal) menu to perform this task when loaded onto a PCB. The interface modules must be disabled to not take over the hardware making it difficult to test.

4.4.3 Bench Build

The bench build allows for the project to be built and run on a desk or lab environment. Since a full machine isn't always available, the bench build can disable certain features such as propel controllers. Commonly turned off is the watchdog for debugging and the code is loaded without a boot loader.

4.4.4 Production Build

The production build is intended for standalone running on a machine. This build enables the watchdog and starts the code location at the offset of the boot loader. With a boot loader in the micro, firmware can be updated with other tools rather than the debugger/programmer.

4.5 Project Directories

There should be significant alignment between the project directory structure and the context diagrams.

4.6 Project Repository

These directories contain all files that are necessary to build the software.
[http://cxsubver100:9891/development/Monaco/Scrub_Controller/trunk/]
[http://cxsubver100:9891/development/Monaco/Membrane_UI/trunk/]
[http://cxsubver100:9891/development/Monaco/Touchscreen_UI/trunk/]
[http://cxsubver100:9891/development/Monaco/Touchscreen_UI_GUI/trunk/]

4.7 Project Documentation

The Doxygen output file for source code documentation is located in the following places:

Scrub Controller Module
6-Code Documentation\Scrub_Controller\rtf\refman.rtf

Membrane UI Module
6-Code Documentation\Membrane_UI\rtf\refman.rtf

Touchscreen UI Module
6-Code Documentation\Touchscreen_UI_LCD\rtf\refman.rtf

Touchscreen UI GUI Module
6-Code Documentation\Touchscreen_UI_GUI\rtf\refman.rtf

5. Configuration Parameters

The configuration parameters allow for selection and monitoring of features and components of the module. Configuration parameters are tied to Tennant Variant Configuration options to be turned on or off when present on the machine. The configuration parameters are also expanded to include options that may be desirable or needed to change in the future.

All parameters are defined through a memory manager module that controls how to read and write to the specific parameter. The memory manager looks up the parameters and determines what memory holds the parameter value. Examples of locations include internal EEPROM, external EEPROM, external Serial flash or FRAM. It also works with a file system.

Outside machine access is granted through the Configuration Galileo USB protocol. This protocol is used to access the memory manager read and write functions. Internally, an interface exists for querying remote node CAN modules. The protocol for that is defined later in this section. Remote CAN modules may accept configuration parameters across the CAN network.

Each parameter is assigned a unique identifier and includes a verification CRC. Some identifiers are constant and may not change from machine to machine. Others are machine specific and therefore the identifier only matters locally. Additionally, some configuration parameters are used only for internal use and are not intended to be externally assigned values.

Since the parameter list can change through development, the following document may be modified with new design information.

Monaco Configuration Parameters.xlsx

The ecH2O parameters document is contained in the Chimney Swift documentation. The telemetry parameters document is contained in the Knight Rider documentation.

5.1 CAN Protocol

A general CAN object provides access to the memory manager on the slave nodes. Object Index 0x2000 has two sub-indexes: one for reading and one for writing. It is important to not confuse the read and write properties of the sub-index with the read and write of a memory parameter. This process is multi-step and does not support simultaneous reads or writes.

CAN Index	CAN Sub-index	Object Contents

The CAN packets are segmented transfers to allow parameters longer than 4 bytes.

5.1.1 Packet Definition

The data contents of the CAN packet is a fixed format whether the read or write sub-index is used.

Byte #	Data Name	Description	Size
1	Error Code	Returns an error code value if there is a problem processing the request. The following error codes are possible: 0 – **None** 1 – **Memory Access Error**: The hardware access to the location of the parameter has failed the read or write process. 2 – **CRC Error**: The read could not be completed since the CRC for the parameter did not match expected. 3 – **Size Error**: The specified size of the parameter field does not match the actual field size. 4 – **Parameter Invalid Error**: The specified parameter number is not recognized or valid.	1
2	Write	Boolean flag to indicate whether the process is a read or write operation. 0 – **Read**: Get the value of a specific parameter from memory. 1 – **Write**: Save a value to a parameter in memory.	1
3	Parameter Number	Specific parameter to read or write. Valid values can be from 0 – 65535.	2
4			
5	Parameter Size	Size of the parameter field. A field may be of any size up to 108 bytes long.	4
6			
7			
8			
9	Data[0]	Data fields can be from 1 to 108 bytes long. Any type of data can be stored in a single parameter including, but not limited to, bytes, shorts, integers, structures, enumerations or strings.	Variable
10	Data[1]		
…	…		
x+9	Data[x]		

5.1.2 Protocol

The protocol for both reading and writing parameters is similar. The following sections lay out the process.

5.1.2.1 *Write*

To write a configuration parameter to memory, follow the steps below.

1. Determine the parameter number.
2. Obtain the length of that parameter.
3. Fill in the CAN packet contents as defined above with Write = 1.
4. Write CAN object 0x2000 sub-index 0x02 (Parameter_Write) with data contents.
5. Read CAN object 0x2000 sub-index 0x01 (Parameter_Read) for reply.
6. The read packet contains the same format as the write. Verify the Error Code in the returned data packet is not greater than 0 indicating the write was successful.

5.1.2.2 *Read*

To read a parameter over the CAN communication, follow the steps below.

1. Determine the parameter number to read.
2. Fill in the packet definition contents. Parameter size does not have to be known. Set Write = 0.
3. Write CAN object 0x2000 sub-index 0x02 (Parameter_Write) with data contents.
4. Read the CAN object 0x2000 sub-index 0x01(Parameter_Read).
5. The returned data in the read object contains the same packet definition. Read the error code to ensure there were no issues reading.
6. Read the parameter number, parameter size and data for results.

6. Operation

The information for high level operation, including the state diagram, is located in the Operational Reference document.

/0110 - Controls/0114 - Functional Guides/T350 Operational Reference.docx

6.1 User Display and Input Operation

A simple guide for user display and input operation is defined in the following document.

/0110 - Controls/0114 - Functional Guides/T350 Standard Membrane User Guide.docx

6.2 Operational Specifications

There are many defined constants and configurable values that allow the machine to operate in a desired fashion. Some settings are determined once the machine performance has been evaluated while other values come from the SRS. The following document defines many of the measurable machine specifications including fault or warning trip conditions, timing for normal operation and some parameter threshold limits.

/0110 - Controls/0114 - Functional Guides/T350 Controls Specifications.doc

6.3 Operation Details

This section contains software design details of the development not captured elsewhere. This section is written as development continues and contains data that an engineer may need or want to know about the operation.

6.3.1 IDrive Reset

The iDrive is a propel module product developed by Curtiss Wright (P-G Drives). The On/Off switch for controlling power is located on pin 5 of the iDrive connector. The machine design is set up such that this line is controlled through the Scrub controller board such that it can be turned on, off, or power cycled as needed through firmware. The primary reason for this is to not allow the propel module to be turned on after an Estop event and the Estop is reset.

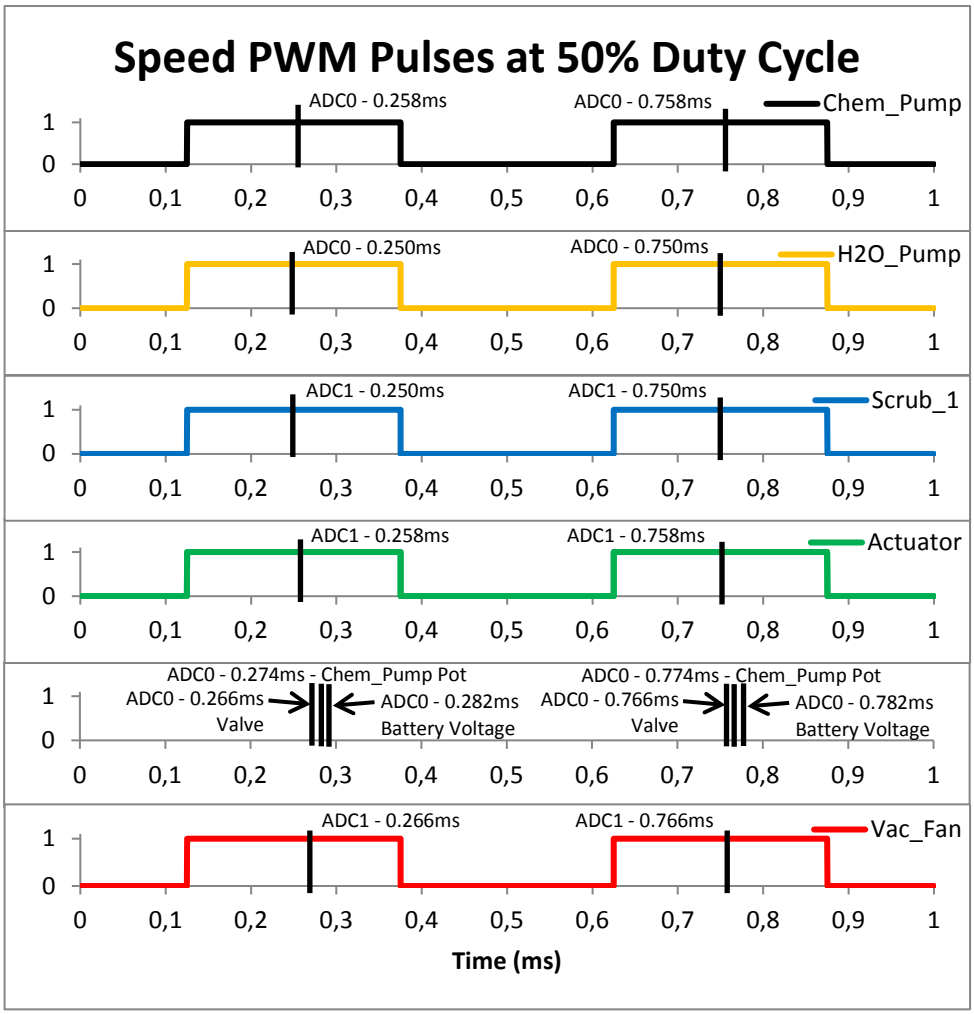
The iDrive also requires a power cycle after programming the operating parameters. The iDrive takes about 3 seconds to begin operation when powered on before use. This delay would cause the serial communication between the iDrive and user interface board to fault out and therefore should be reset with the iDrive.

The easiest method to ensure no machine faults are generated is to perform a key cycle on the entire machine. This cannot be accomplished on base machines. The iDrive is always powered and therefore needs a physical key cycle. On standard machines with a scrub controller board, it can be done electronically. By commanding all the Tennant modules to reset using the CAN reset node command within a small time window, the machine can essentially see a user key cycle. The scrub controller board during reset will de-activate the iDrive on/off switch cutting power. In order for the iDrive to capture the off logic, the line must be held low for at least 40 ms.

Below is a scope capture showing the software reset command and the resulting logic on the iDrive on/off switch line. The requirement is met in that the line is low for approximately 100 ms. This was captured with firmware revision 0.64 (toward end of development).

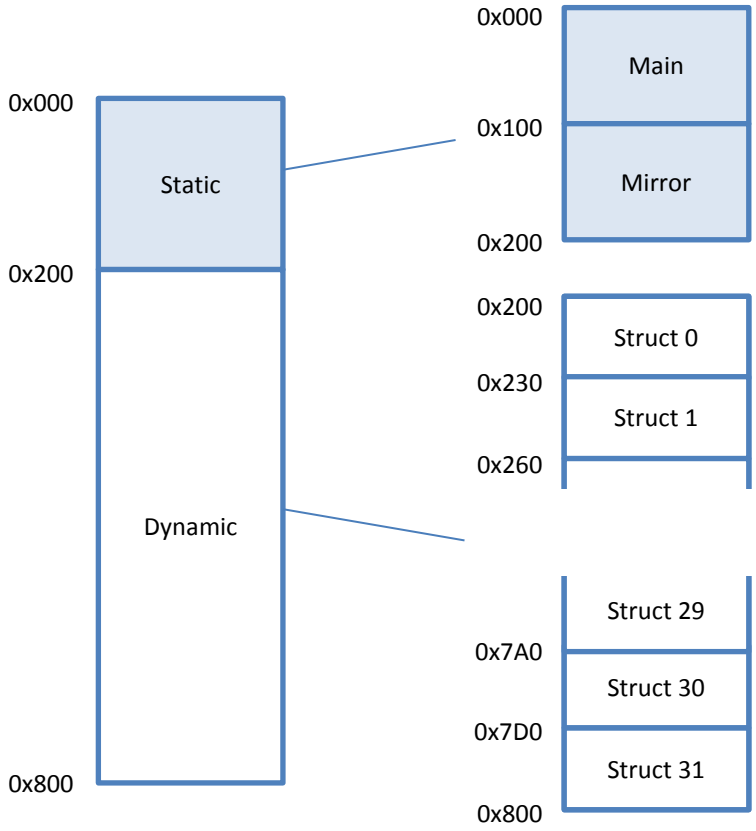
6.3.2 ADC and PWM pulse timing

This section describes the timing of the PWM pulses for motor speed control and the PWM trigger for the ADC module read. The trigger occurs in the center of each PWM cycle.



6.3.3 EEPROM Layout

In order to save frequently changing data in non-volatile memory, the onboard micro EEPROM is divided into two sections which are called static and dynamic. The EEPROM is limited in size so the design has to balance read/write cycles with save intervals. The following diagram maps the regions.



6.3.3.1 *Static Region*

Static memory contains the data that isn't expected to change or will perhaps only change a few times over the life of the machine. This section is divided into the main memory and a mirror section. The mirror is written right after any data is written to the main section. Each parameter is saved with a CRC for verification. Both the main and the mirror contain the CRC. Only if a read fails out of main, is the mirror then read. If the mirror CRC also fails, then a default value or error is returned depending on the function. Situations where this occurs may be a write upon power down that doesn't complete.

6.3.3.2 *Dynamic Region*

Dynamic memory is reserved to save many blocks or chunks of data. Parameters that need to be saved often such as user settings or timers or hour meters is saved in this manner. A block of memory is reserved for a structure that contains all the data objects. The entire structure is saved with a CRC. Upon power up, the memory module starts at the beginning and reads each structure from the region looking specifically at a counter value to locate the latest save. That then defines the next region to save data and holds the most current data. When a parameter is updated, the structure is updated and marked for save on the next periodic interval. The interval is designed such that the EEPROM should not wear out within the 10,000 hours expected life of a controller.

6.3.4 Down Pressure Auto-Adjust

The down pressure on an actuated machine is capable of automatically reducing the user selected down force when the motor current exceeds a threshold. The threshold is defined in the Controls Specifications document and the user observed behavior is defined in the Membrane Guide. The LCD may or may not display the feature depending on firmware version and implementation.

7. Fault Codes & Diagnostics

The machine tracks faults in many of the software modules. These faults are processed at the application layer and translated into a global application fault which assigns a fault code value. The values are two-byte or 4 hex digits allowing over 65,000 possible fault codes. Many codes are standardized such that they can remain the same across different machines. This helps particularly common codes be recognized more easily in the future.

Through development and testing, the controls group may document certain fault conditions with more detail. This may include what conditions caused the fault and the resulting action to take to clear the fault.

The fault code list is another design document that changes or is modified throughout the development process. Therefore, the list is kept in an Excel table in the following location. This also provides the opportunity to sort the fault codes using sorting options in Excel and then exported. The document is located in the following place:

/0110 - Controls/0115 - Software Documentation/Monaco Fault Table.xlsx

A second document for fault codes is also created for quick identity. It does not provide all the specific details for troubleshooting but is useful for some users inside and outside the organization.

/0110 - Controls/0114 - Functional Guides/T350 Display Fault Guide.docx