Software Development for a Low Voltage Load Switching Board

Jaine Elizabeth Perotti, Florida Institute of Technology

Mentor: Chad Kecy

Summer 2012

Keywords: software, embedded, electronics, FOCE, load switching, instrumentation

ABSTRACT

Over the course of ten weeks, software was developed to establish functionality of a low-voltage load switching board. Designed to protect scientific instruments deployed on the FOCE (Free Ocean Carbon Enrichment experiment) platform, the load switcher's hardware automatically trips a circuit breaker when certain fault conditions are present (over current, over voltage, and under voltage). Then, the software switches the associated instrument's relay off in order to provide full galvanic isolation. The software is also able to set the thresholds for each of the fault conditions, as provided by the user.

INTRODUCTION

Rising ocean acidity is emerging as one of the greatest scientific concerns of our time. Ocean acidification is the result of increased anthropogenic CO2 emissions. Human combustion of fossil fuels and land use practices in recent decades have lead to an unprecedented rate of increasing atmospheric CO2 concentrations. The current rate of uptake of CO2 in the world's oceans is far greater than that experienced by marine organisms for at least 20 million years. [1] That ocean acidification is occurring is an easily observable fact; between 1751 and 1994, surface ocean pH has decreased from approximately 8.25 to 8.14, which is an increase of almost 30%. [2]

Marine scientists are increasingly concerned about the potential effects on marine organisms and plant life. Impacts could have serious implications for the health of the global food web and human

economies. As a result, ocean acidification research has now become an increasingly important field of study. Although experiments in the lab are useful to researchers, a need existed for the ability to conduct in-situ ocean acidification experiments. In the lab, it is difficult to replicate all of the variables that exist in real ocean environments.

In response to this need, MBARI researchers devised an underwater science laboratory called "FOCE", which stands for the Free Ocean Carbon Enrichment Experiment. The FOCE is essentially a test chamber that makes precisely controlled, small changes to pH, and is deployed in the open ocean. Between 2005 and 2011, the first FOCE experiment was deployed in the Monterey Bay in about 900 meters of water, where it is still operational today as part of the MARS observatory. [3]

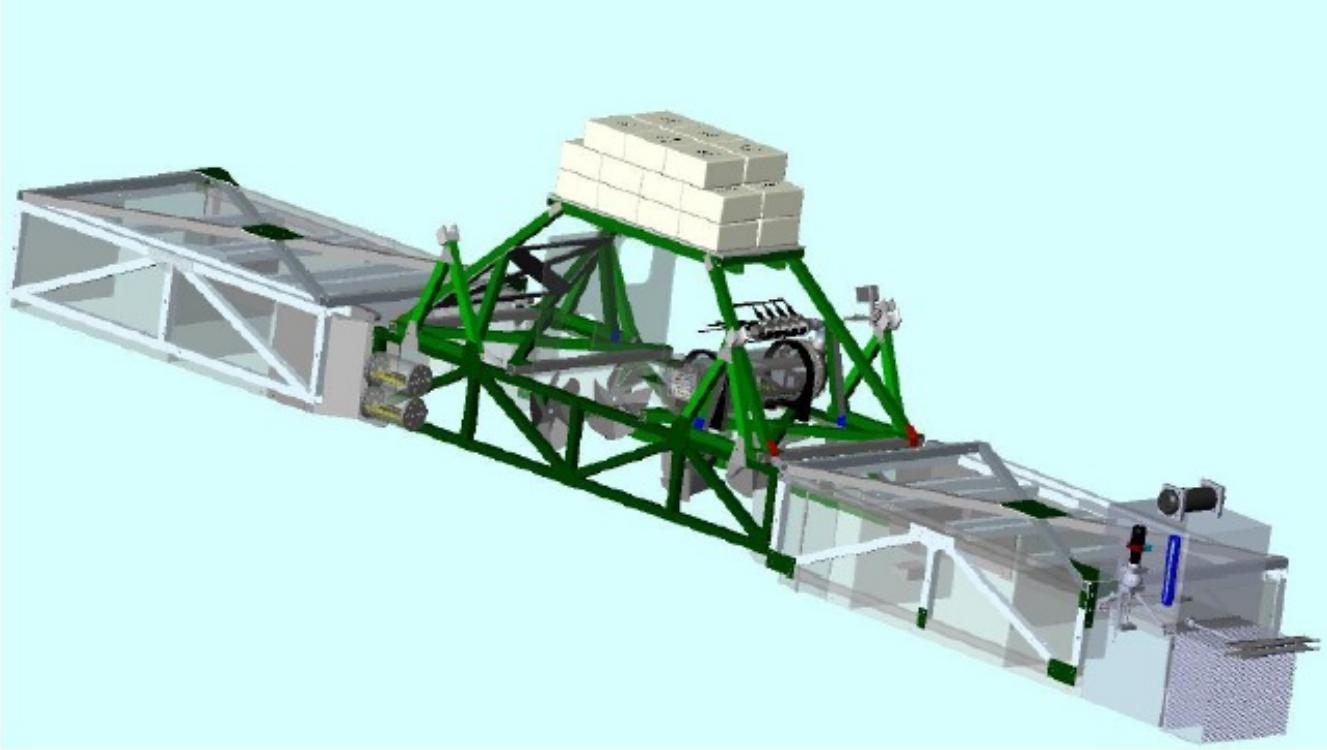


Figure 1: FOCE Drawing

As a result of the success of the deep water FOCE, researchers from all over the world showed an interest in adapting the technology to suit their own experiments. One of the first collaboration efforts resulted in the design and deployment of the Coral Prototype FOCE, which was installed at Heron Island, off the coast of Northeast Australia. The success of this effort gave rise to the concept of the "Exportable FOCE": an open-source version of the system that is intended to be inexpensive, easily modified and easily installed for use in a variety of different applications. The Shallow-Water FOCE is the first version of the Exportable FOCE to be developed.[3]

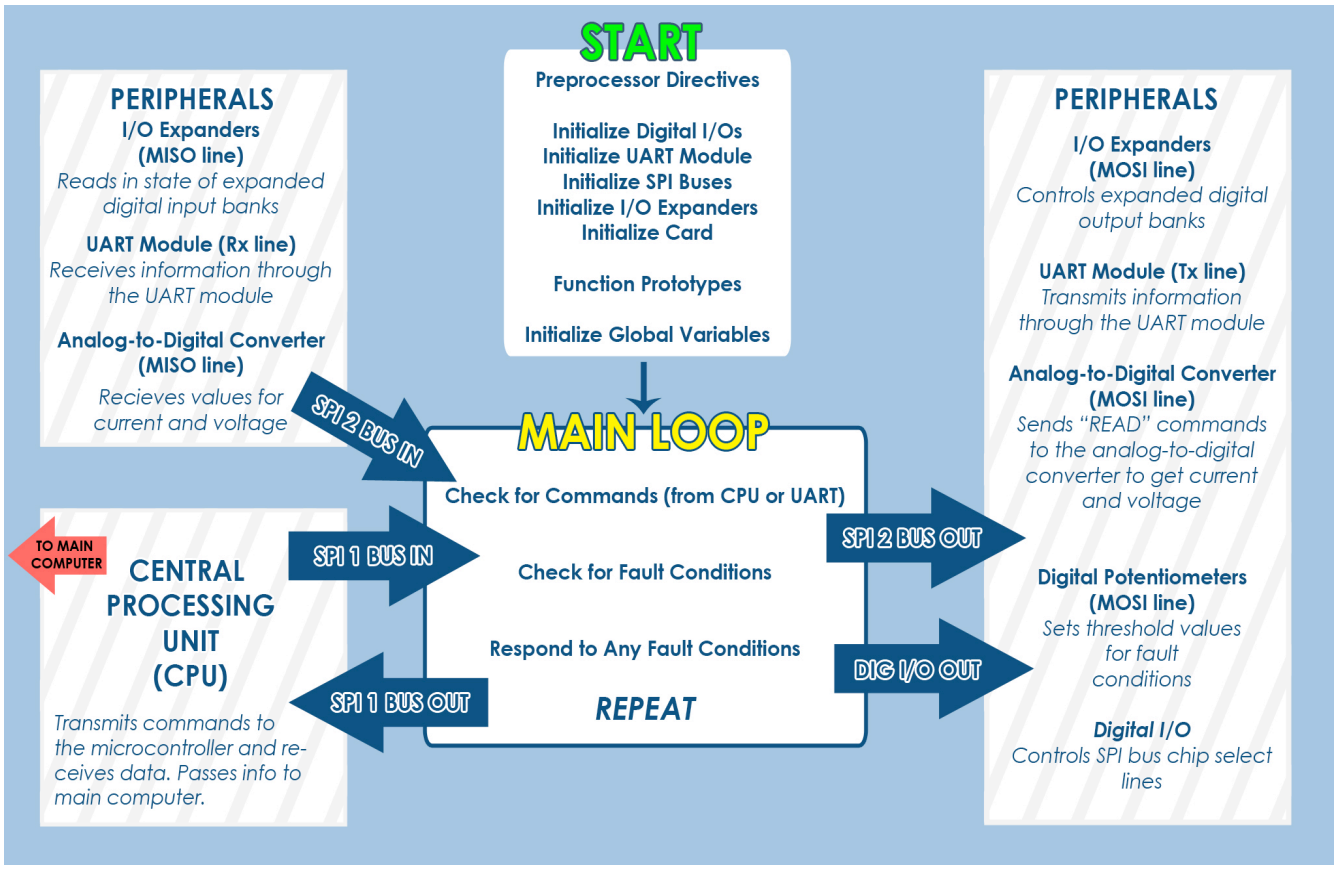
The focus of work this summer was on developing software for the Shallow Water FOCE's low voltage load switching board. The load switcher protects the FOCE's scientific instruments from being damaged by a number of different fault conditions, including over current, over voltage, and under voltage. Software needed to perform the following operations: digital input and output, reading voltage and current information, setting thresholds for fault conditions, and communication with the various devices on the board, as well as serial communication to the user.

MATERIALS AND METHODS

MPLAB IDE v.8.86 was the development environment that was used to write the actual code. The C30 Compiler was selected to "translate" the higher-level C programming language into the lower-level machine language that the PIC 24FJ256GA106 is able to understand. The MPLAB ICD 3 programmer physically connected the computer to the microcontroller via a USB programming cable.

RESULTS

PROGRAM FLOW OF EXECUTION



The program starts by setting preprocessor directives and initializing the controllers digital I/O

banks, UART module, and SPI module. It then initializes the digital I/O expanders and card information. Function prototypes for main() are then initialized, as are all global variables.

Within the main loop, the program checks for commands from either the CPU or UART (debugging station). If the controller has received a command, the program will be interrupted and the command will be executed. The program also continuously polls for fault conditions. If a fault condition is present, the program responds by checking for that condition three more times. If the condition remains, the instrument will be permanently shut down until a reset of the low voltage load switching board itself occurs.

DISCUSSION

INSTRUMENT CHASSIS

This is a diagram illustrating the way the FOCE manages it's scientific instruments. It is important for the shallow water FOCE to be able to monitor the operating conditions of each instrument, as well as to supply or cut off power to them.

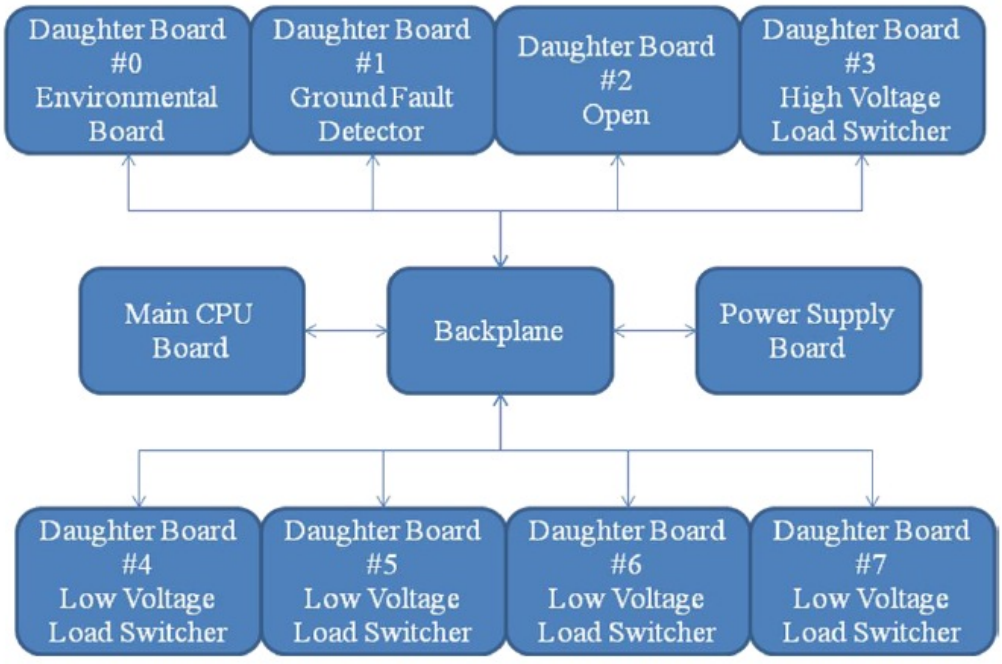


Figure 2: FOCE Chassis Block Diagram

Power and communication is distributed to all of the different system components through the backplane. The power supply board makes all of the power conversions needed for the components on

the various boards (12 vdc, 5 vdc, and 3.3 vdc). The main CPU board receives information from the individual daughter boards, consolidates that information, and sends it to another, more powerful computer that talks to the surface. The CPU board can also send various commands to its daughter boards.

The environmental daughter board is used to monitor conditions inside the pressure housing, such as temperature, pressure, and humidity. This ensures that the scientific instruments aren't compromised by their environment. The ground fault detection board is used to detect the presence of sea water in the pressure housing. Finally, the load switching boards are used to monitor the amount of current and voltage each instrument is pulling. If there is too much current, too much voltage, or too little voltage, the load switching board will automatically shut off power to that instrument, preventing that instrument from damaging itself.

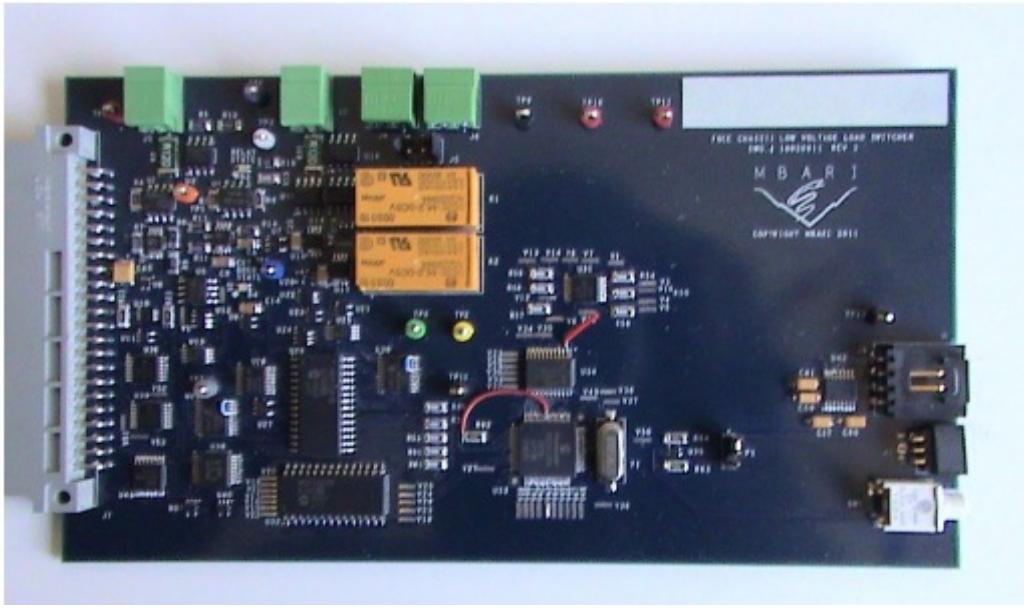


Figure 3: Low Voltage Load Switching Board

The load switcher uses two different components to turn power on/off to the scientific instruments: a circuit breaker that is automatically tripped by hardware in the presence of a dangerous condition, and a relay which is subsequently turned off to provide full galvanic isolation (small amounts of current can pass through the circuit otherwise).

The "brain" of the load switcher, so to speak, is a PIC 24FJ256GA106 microcontroller. It monitors current and voltage flowing to the instruments on board, and detects hardware generated signals which indicate the presence of over current, over voltage, and under voltage conditions. The controller is also used to adjust the thresholds for these conditions. In addition to these tasks, the

controller also performs certain actions in response, such as resetting the circuit breaker and turning the relays on and off.

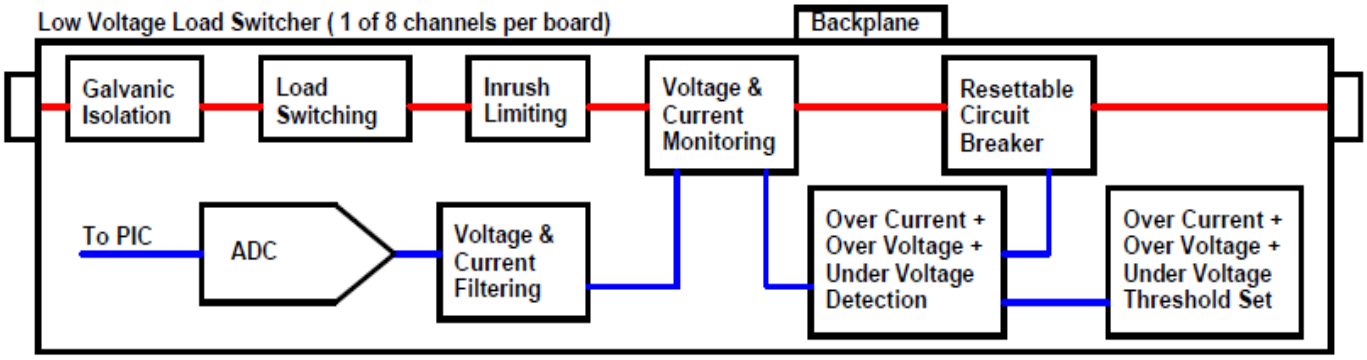


Figure 3: Low-Voltage Load Switching Board Block Diagram

SYSTEM DEFINITIONS

Digital I/Os

Table 1: Digital Outputs

Name	**Value**
PIC_DIO_01_RST	LATFbits.LATF4
PIC_DIO_02_RST	LATFbits.LATF5
PIC_DIO_03_RST	LATFbits.LATF6
PIC_DIO_04_RST	LATGbits.LATG2
PIC_DIO_05_RST	LATEbits.LATE6
PIC_ADC_01_SHDN	LATFbits.LATF0
PIC_ADC_02_SHDN	LATFbits.LATF1
PIC_ADC_03_SHDN	LATFbits.LATF2
PIC_ADC_04_SHDN	LATFbits.LATF3
PIC_DIO_05_CS	LATEbits.LATE5
CS_ADD_EN	LATEbits.LATE4
CS_ADD_A0	LATEbits.LATE3
CS_ADD_A1	LATEbits.LATE2
CS_ADD_A2	LATEbits.LATE1
CS_ADD_A3	LATEbits.LATE0
BD_ADD_0	LATDbits.LATD0
BD_ADD_1	LATDbits.LATD1
BD_ADD_2	LATDbits.LATD2

PIC_INT_OE	LATDbits.LATD11

Table 2: Digital Inputs

Name	**Value**
PIC_DIO_01_INT	PORTDbits.RD3
PIC_DIO_02_INT	PORTDbits.RD4
PIC_DIO_03_INT	PORTDbits.RD5
PIC_DIO_04_INT	PORTDbits.RD6
PIC_DIO_05_INT	PORTEbits.RE7

Daughterboard Types

Chip Select

I/O Expanders

Digital Potentiometers

Instrument Channels

Analog-to-digital Converters

PERIPHERAL DEVICES

The Low Voltage Load Switching board features a number of different peripheral devices and microcontroller modules. Peripheral devices included digital I/O expander banks, analog-to-digital converters, digital potentiometers, resettable circuit breakers, and relays. The microcontroller utilized its digital I/O, UART (Universal Asynchronous Receive and Transmit), and SPI (Serial Peripheral Interface) modules in order to interface with these peripheral devices.

Table 1: Peripherals and Associated Source Files

Peripheral Name	**Source File Name(s)**
Analog-to-digital converter	adcnew.c
Circuit breaker	breaker.c

Digital I/O expander	expander.c
Digital potentiometer	potentiometer.c
Relay	relay.c

Table 2: PIC Modules and Associated Source Files

Module Name	**Source File Name(s)**
Digital I/O	dig_io.c
UART	serial.c
SPI	spi.c, spi2.c

PIC Modules

The microcontroller's onboard digital I/O bank was used mainly to set up chip select lines to the digital I/O expander banks, as well as to control the addressing of the 4-to-16 line multiplexer (which handled the chip select lines of the other peripherals). The digital I/O expanders handled the remaining digital I/O tasks (turning relays on/off, resetting circuit breakers, reading in fault conditions).

The UART module was used to send and receive serial data to/from a PC user for debugging purposes. HyperTerminal, or some other terminal emulation program, may be used for this purpose. The program responds to these inputs in the same way it would if they originated from the CPU via the SPI1 bus.

The SPI module (SPI2) was utilized for communication to virtually all of the board peripherals, including the digital I/O expanders, analog-to-digital converter, and digital potentiometers. The SPI module (SPI1) was also used for communication between the CPU and the low-voltage load switching board, as well as between the CPU and its other daughter boards.

Digital I/O Expander Banks

The digital I/O expander banks constitute an extremely important part of the functionality of the board. Their outputs control the board's relays and circuit breakers, as well as controlling the write-protect and reset pins on the digital potentiometers. Their inputs read in the states of the relays and circuit breakers, as well as the presence of over current, over voltage, and under voltage fault conditions. On the current iteration of the board, there are two I/O expanders which serve two different instrument channels. On future versions of the instrument, additional I/O expanders will be added to accommodate eight separate instrument channels. The digital I/O expander bank is accessed by the microcontroller using the SPI2 bus.

Table 3: Digital I/O Expander Bank – Inputs and Outputs

Expander #	Inputs	Outputs
DIO_01	RELAY_01_STATE	RELAY_01_ON
	CKT_BRKR_01_STATE	RELAY_01_OFF
	CHAN_01_OC	CKT_BRKR_01_RST
	CHAN_01_OV	RELAY_02_ON
	CHAN_01_UV	RELAY_02_OFF
	RELAY_02_STATE	CKT_BRKR_02_RST
	CKT_BRKR_02_STATE	
	CHAN_02_OC	
	CHAN_02_OV	
	CHAN_02_UV	
DIO_05	POT_01_RST	
	POT_01_WP	
	POT_02_RST	
	POT_02_WP	

Table 4: Digital I/O Expander Function List

Function Name	Description
expInit()	Initializes the LVLS board's I/O expanders by setting pins as either inputs/outputs, and initializing outputs as either high or low.
expReset(int device_select)	Resets the digital I/O expanders.
expClr(int device_select, int regstr)	Clears the value (sets to 0x00) of a GPIO register in the LVLS board's digital I/O Expanders , as well as clearing the slot in the array that holds the value of that register (if that register is a GPIO).
expRead(int device_select, int regstr)	Performs a simple read of any register in the LVLS board's digital I/O expanders.
expWrite(int device_select, int regstr, int data)	Performs a simple write of any register in the LVLS board's digital I/O expanders.
expSetHigh(int device_select, int regstr, int val)	Sets an individual bit high in the digital I/O Expanders' GPIO registers, without changing the value of the other bits in that register.
expSetLow(int device_select, int regstr, int val)	Sets an individual bit low in the digital I/O Expanders' GPIO registers, without changing the value of the other bits in that register.
expReadBit(int device_select, int regstr, int val)	This function performs a read of an individual bit within a register in the LVLS board's digital I/O expanders. Returns true if bit is set, false if bit is not set.
expDigInit()	This function initializes the values of each of the I/Os associated with

	each expander, in a manner analogous to that done in digInit(). Output initialization routines will be added in later versions of the board for DIOs 02, 03, & 04.

Analog-to-digital Converters

The analog-to-digital converters perform the important task of reading in voltage and current information to the controller. Like the digital I/O expanders, they communicate with the microcontroller through the SPI2 bus. The function adcRead() is optimized for the addition of more instrument channels and ADCs. Currently, ADC_01 is the only analog-to-digital converter installed on the board.

Table ?: Analog-to-digital Converter Quantity List

ADC #	**Name**	**ADC Channel**
ADC_01	CHAN_01_VOLT	CH0
	CHAN_01_CURR	CH1
	CHAN_02_VOLT	CH2
	CHAN_02_CURR	CH3
ADC_02	CHAN_03_VOLT	CH0
	CHAN_03_CURR	CH1
	CHAN_04_VOLT	CH2
	CHAN_04_CURR	CH3
ADC_03	CHAN_05_VOLT	CH0
	CHAN_05_CURR	CH1
	CHAN_06_VOLT	CH2
	CHAN_06_CURR	CH3
ADC_04	CHAN_07_VOLT	CH0
	CHAN_07_CURR	CH1
	CHAN_08_VOLT	CH2
	CHAN_08_CURR	CH3

Table ?: Analog-to-digital Converter Function List

Function Name	**Description**
adcRead(int quantity)	Reads a 16-bit value from the analog to digital converter. ADC_02, ADC_03 and ADC_04 do not yet exist on current board.

Digital Potentiometers

The digital potentiometers are used to set the thresholds for the over current, over voltage, and under voltage fault conditions. Like the digital I/O expanders and ADCs, the digital potentiometers are read and written to using the SPI2 bus. In this software, writes and reads are made to non-volatile memory locations (EEPROM) so that settings are not lost when board is turned off.

Table ?: Digital Potentiometer Quantity List

Potentiometer #	**Name**	**Wiper**
POT_01	OVER_CURR_01	POT_01_P0W
	OVER_VOLT_01	POT_01_P1W
	UNDER_VOLT_01	POT_01_P2W
	INRUSH_LIM_01	POT_01_P3W
POT_02	OVER_CURR_02	POT_02_P0W
	OVER_VOLT_02	POT_02_P1W
	UNDER_VOLT_02	POT_02_P2W
	INRUSH_LIM_02	POT_02_P3W

Table ?: Digital Potentiometer Function List

Function Name	**Description**
potReset(int pot_select)	Resets the digital potentiometers. The argument pot_select will take one of two possible defs, POT_01_RST, or POT_02_RST, as set up in sys_defs.h. More pot defs may be added in later iterations of board with more than two digital potentiometers.
potReadWrite(int pot_select, int cmd, int wiper, int value)	Formats a command using bit shifts and a logical "or" into two 8-bit pieces, then writes them to one of the board's digital potentiometers. When using this function to read, just set the argument 'value' equal to 0x00.
potIncrDecr(int pot_select, int cmd, int wiper, int cycles)	This function increments or decrements the wiper using an 8-bit command. Does not work with non-volatile memory locations. Not used.

Relays

Circuit Breakers

COMMAND ARCHITECTURE

CONCLUSION/RECOMMENDATIONS

[REMOVE "I" LANGUAGE]

First, a couple of hardware fixes need to be made. A problem with the inrush limiting circuit repeatedly destroyed one of the digital potentiometers I was working with while testing my code. That will need to be changed. Also, while the current software has the board continuously monitoring for fault conditions, I would like to change the software so that it only looks at fault conditions when the hardware signals it to do so - thus "interrupting" the program instead of bogging it down with constant commands. Communication also needs to be established between the board and the CPU. This aspect of the project was something that I did not have time to test. The next version of the hardware will also include 8 channels for scientific instruments, instead of just two.

REFERENCES

[4] http://www.mbari.org/mars/

[3] http://www.mbari.org/news/homepage/2012/cpfoce/cpfoce.html

[1]: http://www.epoca-project.eu/index.php/what-do-we-do/outreach/rug/oa-questions-answered.html

[2]: Jacobson, M.Z. (2005). "Studying ocean acidification with conservative, stable numerical schemes for nonequilibrium air-ocean exchange and ocean equilibrium chemistry". Journal of Geophysical Research – *Atmospheres* **110**: D07302.

ACKNOWLEDGEMENTS

APPENDICES