OCEANOGRAPHIC INSTRUMENT SIMULATOR

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Amy Yenyu Chen

March 2016

© 2016
Amy Yenyu Chen
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Oceanographic Instrument Simulator

AUTHOR: Amy Yenyu Chen

DATE SUBMITTED: March 2016

COMMITTEE CHAIR: Vladimir Prodanov, Ph.D.

Associate Professor of Electrical Engineering

COMMITTEE MEMBER: Bridget Benson, Ph.D.

Assistant Professor of Electrical Engineering

COMMITTEE MEMBER: Chad Keczy, M.S.

Monterey Bay Aquarium Research Institute

ABSTRACT

Oceanographic Instrument Simulator

Amy Yenyu Chen

The Monterey Bay Aquarium Research Institute (MBARI) established the Free Ocean Carbon Enrichment (FOCE) experiment to study the long-term effects of decreased ocean pH levels by developing in-situ platforms [1]. Deep FOCE (dpFOCE) was the first platform, which was deployed in 950 meters of water in Monterey Bay. After the conclusion of dpFOCE, MBARI developed an open source shallow water FOCE (swFOCE) platform located at around 250 meter of water to facilitate worldwide shallow water experiments on FOCE [1][2]. A shallow water platform can be more ubiquitous than a deep-water platform as shallow water instruments are less expensive (as it does not have to be designed to withstand the pressure at deep ocean depths) and more easily deployed (they can be deployed right along the coast).

The swFOCE experiment is an open source platform, and MBARI has made the plans available online to anyone interested in studying shallow water carbon enrichment. There is a gateway node what is connected to four sensor nodes within the swFOCE. In order to test the sensor node individually, an idea of designing an Oceanographic Instrument Simulator is purposed. The Oceanographic instrument simulator (OIS), described in this paper provides the means for MBARI engineers to test the swFOCE platform without attaching the numerous and expensive oceanographic instruments. The Oceanographic Instrument Simulator simulates the various scientific instruments that could be deployed in an actual experiment.

The Oceanographic Instrument Simulator (OIS) system includes the designed circuit board, Arduino Due and an SD Card shield. The designed circuit board will be connected to a computer through a USB cable, and be connected to MBARI's swFOCE sensor node through a serial connection. When a query is given from the sensor node, the Arduino Due will parse the data given from the sensor node, search through the pre-installed data in the SD card and return the appropriate data back to the sensor node. A user can also manually set up the input current through a computer terminal window to control the simulated signals from the PCB.

Key Words: oceanographic; electronic load; sensors; MBARI; open source; simulator

ACKNOWLEDGMENTS

I would like to thank Dr. Vladimir Prodanov for directing and guiding me throughout the development of the oceanographic instrument simulator. He was always there to guide me and direct me to the right track whenever I needed to troubleshoot the simulator. Without his patience and guidance, I would not have been able to successfully troubleshoot the simulator.

I would like to thank Dr. Bridget Benson for pushing me to go above and beyond my expectations. She pushed me to write an abstract and publish a paper for IEEE Ocean's 15 and give a talk on the project during graduate seminar. She showed me all the possibilities I could have as I worked throughout the project.

I would like to thank Chad Kecy for the opportunity to work on this project. I would not have been able to work on the project without the opportunity that he provided. I would also like to thank MBARI for supporting the project and providing the summer internship opportunity for me to work on the OIS during the summer of 2015.

I would like to thank Matthew Mitchell and Richard Bae for working on the original OIS software programming. I was able to follow, modify, and improve the code.

Finally, I'd like to thank my family for their support and encouragement. Without their support, I would not be able to continue my education.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vii
LIST OF FIGURES	viii
CHAPTER 1: INTRODUCTION.....	1
CHAPTER 2: BACKGROUND AND PREVIOUS WORK	2
CHAPTER 3: PROJECT DESIGN	5
3.1 HARDWARE	8
3.1.1 DIGITALLY-CONTROLLED ELECTRONIC LOAD	10
3.1.2 CURRENT SENSE CIRCUITRY	17
3.1.3 ADC INTERFACE	19
3.1.4 4-20MA CURRENT LOOP TRANSMITTER	19
3.2 SOFTWARE	21
3.2.1 MANUAL MODE /SENSOR COMMANDS	24
3.2.2 RS232 SERIAL PORT COMMUNICATIONS/ TRANSCEIVER CHIP	27
3.2.3 SD CARD.....	29
CHAPTER 4: INTEGRATION TESTING	30
4.1 HARDWARE INTERFACE	32
4.1.1 DAC & ELECTRONIC LOAD INTERFACE	32
4.1.2 CURRENT SENSE CIRCUIT & ADC INTERFACE	38
4.1.3 DAC & ELECTRONIC LOAD & CURRENT SENSE CIRCUITRY & ADC INTERFACE	39
4.1.4 DAC & 4-20MA CURRENT LOOP TRANSMITTER INTERFACE	41
4.2 SOFTWARE AND HARDWARE SYSTEM INTEGRATION	42
4.2.1 PARALLEL MANUAL MODE TESTING	43
4.2.1.1 SINGLE 2A LOAD	44
4.2.1.2 THREE 2A LOADS	44
4.2.1.3 SINGLE 6A LOAD	45
4.2.2 PARALLEL SENSOR DATA/ SENSOR MODE	45
4.2.3 THERMAL TESTING	46
CHAPTER 5: CONCLUSION	50
REFERENCES	51
APPENDICES	
APPENDIX A – TESTING RESUTLS	52
APPENDIX B – OCEANOGRAPHIC INSTRUMENT SIMULATOR SCHEMATICS	54
APPENDIX C – ARDUINO CODE	59
INSTR_SIM_DRIVER.H	59
INSTR_SIM_DRIVER.INO.....	62

LIST OF TABLES

Table	Page
Table 1: Selection of DAC	11
Table 2: DAC value conversion	12
Table 3: The section of the Resistance value	13
Table 4: The section of the MOSFET	13
Table 5: The selection of the Op-amp	14
Table 6: ADC value conversion	19
Table 7: Current Sense Circuit vs. ADC	39
Table 8: Power Budget	48

LIST OF FIGURES

Figure	Page
Figure 1: Block Diagram of swFOCE Implementation [1]	3
Figure 2: Single 2A Load Protoboard	4
Figure 3: Oceanographic Instrument Simulator Block Diagram	6
Figure 4: Hardware Block Diagram	9
Figure 5: DAC Controlled Electronic Load Schematic	10
Figure 6: DAC interfaced schematic	12
Figure 7: Cooling Fan	15
Figure 8: Relative fan surface compare to ois	16
Figure 9: Electronic load test results	16
Figure 10: Block diagram for single 2A and Load and Single 6A load	17
Figure 11: LT1999 Block Diagram [8]	18
Figure 12: Current sense circuitry	18
Figure 13: 4-20mA Transmitter loop schematic [9]	20
Figure 14: 4-20mA Transmitter loop testing result	21
Figure 15: Main meau screen on hyperterminal window	22
Figure 16: Software flow Diagram of instrument simulator	23
Figure 17: pH Sensor and Manual Mode selection	24
Figure 18: Example data format for pH sensor [2]	25
Figure 19: Arduino Due connected to a computer through a USB connection [2]	26
Figure 20: Transceiver Schematic	27
Figure 21: RS232 Transceiver set up [2]	28
Figure 22: Arduino Due connected to RS232 connector [2]	28
Figure 23: Arduino Due with SD Card Shield Attached [2]	29
Figure 24: OIS Testing Block Diagram	31

Figure 25: Input of 1Vpp (yellow); Output of gate voltage (blue)	32
Figure 26: Electronic Load with Small Signal Capacitors	33
Figure 27: Conceptual Electronic Load Control Loop	34
Figure 28: Low Pass Filter with Op-amp Feedback	35
Figure 29: Electronic Load with Additional Resistor	36
Figure 30: Schematic Diagram of One-Half of LM358 [5]	36
Figure 31: Input of 1Vpp (yellow); Output of gate voltage (blue)	37
Figure 32: DAC Controlled Electronic Load Schematic	37
Figure 33: Voltage Input from the DAC vs. Current Output to the 12V Supply Sensor Node	38
Figure 34: Current Sense Circuit and ADC Schematic	39
Figure 35: DAC and Electronic Load and Current Sense Circuit and ADC Schematic	40
Figure 36: Integration Testing Result	40
Figure 37: DAC 4-20mA current loop transmitter	41
Figure 38: DAC and 4-20mA Current Loop Transmitter Result	41
Figure 39: The OIS Circuit Board	42
Figure 40: Single Load Test	44
Figure 41: Parallel Testing for Three Loads at the same time	44
Figure 42: Parallel Testing for Load #4	45
Figure 43: pH Sensors Command	46
Figure 44: Thermal Test without Fan	47
Figure 45: Thermal Test with Fan	48
Figure 46: Current output vs. Power limits	49
Figure 47: Electronic Load (without current sense circuitry) Testing Data	53

CHAPTER 1: INTRODUCTION

The Oceanographic Instrument Simulator (OIS), the topic of this paper, was designed to give test engineers the ability to test the functionality of the sensor node module within the swFOCE without the need for having the actual expensive oceanographic instruments connected to the system. The OIS simulates up to four oceanographic instruments by providing an equivalent electrical load and data stream as the real instrument. Thus, scientists and engineers can connect the OIS to the swFOCE, configure the OIS to simulate the oceanographic instruments they plan to deploy, and test the system to ensure the power limits of the system are not exceeded and the data from the ‘sensor’ accurately flows through the various modules of the system.

The rest of this paper is organized as follows: In chapter 3, the project design including the hardware board and the software user interface will be introduced and explained. The sub-chapters within chapter 3 will cover the design of both hardware and software. Chapter 4 will include the interface testing from hardware to software. Chapter 5 will then conclude with relating the project to the oceanographic community and describing possible future work.

CHAPTER 2: BACKGROUND AND PREVIOUS WORK

For the swFOCE experiment, sensors will be used to measure various characteristics of the shallow ocean water such as pH level, oxidation level, etc. The sensors will be connected to swFOCE sensor node, which is connected to the swFOCE gateway node. The gateway node is then connected to swFOCE device's main electrical housing, which is connected to the central node of the swFOCE experiment system, shown in Figure 1. Each of the four sensor nodes, shown at the bottom of the figure, allows the user to connect four different sensors (three with a current limit of 2A and one with a current limit of 6A). The sensor node gathers readings from each of its attached sensors and sends the data back to a shore station through the gateway node. The swFOCE experiment could generate a large amount of data from a variety of instruments, including pH sensors, CTDs, O₂ sensors, water flow-meters, and ADCPs. This data propagates through a number of subsea housings prior to being stored on a local shore computer and disseminated over the web.

The sensor node module, designed by MBARI, is used to interface a wide range of scientific instruments. The sensor node provides power and serial communication to a maximum of three instruments. The third channel can be switched to interface a 4-20mA instrument. The fourth channel is designed to have high current, up to 6A, for devices such as lights or simple motors. The sensor node can provide a maximum of 75 watts and can be configured for 12 Volts, 24Volts, or 48Volts.

FIGURE 1: BLOCK DIAGRAM OF SWFOCE IMPLEMENTATION [1]

Prior to the development of the OIS in this thesis, there was a single 2A prototype built for the senior project for the OIS. The single 2 Amperes electronic load has a user interface controlled by the Arduino Due software program. The user can either manually control the 2A single load or use pH sensor, the sensor mode to simulate pH sensor behavior.

FIGURE 2: SINGLE 2A LOAD PROTOBOARD

This thesis will include the complete integration of the OIS, the PCB board and the Arduino Due, that simulates up to 4 sensors: three 2 Amperes loads and one 6 Amperes load. The software program of the OIS will be able to simultaneously test all 4 sensors through either manual mode or sensor mode. This thesis includes the write-up of the hardware and software design and the integration test results.

CHAPTER 3: PROJECT DESIGN

The Oceanographic Instrument Simulator (OIS) is designed to be a simple portable test system for swFOCE platform. It can be used in a lab or office without additional oceanographic instruments. The OIS will be connected directly to the sensor node from the swFOCE to simulate sensor behaviors. There are four channels on the sensor node. Three of the four oceanographic instruments have a current limit of 2 Amperes and one of the four oceanographic instruments has a current limit of 6 Amperes. The specifications of the OIS are specified according to the sensor node form the swFOCE [1]. The overall power rating of the sensor node is 75 Watts therefore the OIS board cannot exceed 75 Watts. The OIS is designed to run on 12 Volts, 24 Volts, or 48 Volts, because those are commonly used power supplies for oceanographic instruments.

The OIS will be connected to the computer for the user to run the OIS with the terminal. A terminal will appear after the USB port from the OIS successfully connects to a computer. The software program initializes the OIS and ensures the connection between the sensor node and the OIS.

FIGURE 3: OCEANOGRAPHIC INSTRUMENT SIMULATOR BLOCK DIAGRAM

The OIS block diagram and its connection to the swFOCE sensor node is depicted in Figure 3. Electronic Loads #1, #2, and #3 simulate the three instruments with a current limit of 2 Amperes and Electronic Load #4 simulates an instrument with a current limit of 6 Amperes. Note that Port #3 can also be switched to interface a 4-20mA instrument per the sensor node's design. The three transceivers depicted below electronic loads #1, #2, and #3 emulate serial data transfer from each of the simulated sensors. The high current port #4 was not designed for serial interface as it is meant to interface to LEDs or motors. The Micro SD shield is connected to an SD card, which stores simulated sensor data. The user can configure each load and each communication port to simulate a sensor of interest (either manually or from a drop down list of sensors such as the SeaBird SBE18s pH sensor [9]). Once the OIS is configured, the user can attach the OIS to the sensor node of the swFOCE for full system testing.

The Arduino Due sets up the configuration, shows the instructions and monitors the OIS through a terminal window. The USB Port on the Arduino Due connects to the personal computer (PC) for the user to have access to the terminal. The Arduino Due was selected because unlike other Arduino boards, it contains four UART ports. The Arduino board sets the output voltage of the DAC, takes in feedback from the current sense circuit via the ADC and verifies the functionality of the UART ports.

The overall design of the OIS can be categorized into hardware and software. The following sections will go into detail on the hardware design and software design. The hardware design will contain the detailed schematic and testing of each hardware component used in the OIS. The software design will contain the detailed functionality of the main menu and the power protection function that turns off the OIS to prevent damage of the hardware.

3.1 HARDWARE

The hardware design of the OIS includes the DAC design, the electronic load design, the current sense circuit design, the 4-20mA current loop transmitter design and the ADC design. All these designs were built onto a single PCB, as shown in Figure 4. The components on the OIS printed circuit board (PCB) are shown in solid lines and the Arduino Due and the power supply are the other hardware components that are connected to the PCB.

There are a total of five DACs: four of the DACs are used to simulate three 2A loads and one 6A load and the other DAC is used for the 4-20mA current loop transmitter. In order to simulate each individual port, each instrument simulator port includes an electronic load, current sense circuit and an ADC. The Arduino Due contains the software user interface program. The power supply that supplies the OIS will be replaced with the sensor node from the swFOCE once connected to the sensor node.

FIGURE 4: HARDWARE BLOCK DIAGRAM

3.1.1 DIGITALLY-CONTROLLED ELECTRONIC LOAD

The electronic loads are designed to connect to the sensor nodes from the swFOCE. The OIS consists of four electronic loads. Channels 1, 2, and 3 have an electronic load of 2A and Channel 4 has an electronic load of 6A. Each individual electronic load consists of a DAC, an op-amp, MOSFET and a resistor. The purpose of the 2A electronic load is to sink in current from 0A to 2A, as shown in Figure 5. The sensor node will provide voltages that vary from 12V, 24V and 48V to supply the electronic load.

Figure 5 shows the DAC controlled electronic load. The Arduino Due will control the DAC and the electronic load by inputting a digital value to the DAC. The digital value will be converted to an analog voltage at the output of the DAC. The voltage input to the op-amp will control the voltage at the gate and source, which causes the MOSFET to sink in current varying from 0A to 2A.

FIGURE 5: DAC CONTROLLED ELECTRONIC LOAD SCHEMATIC

The DAC is controlled by the Arduino Due to convert the digital voltage signal to an analog signal in order to input an analog value to the electronic load. The DAC provides a fixed voltage

to the positive terminal of the op-amp. The selection of the DAC based on design requirements is shown in Table 1.

TABLE 1: SELECTION OF DAC

	Design	Selected (MCP4921)
Resolution	12bits	12bits
Supply voltage	3.3V	2.7V~5.5V
Output voltage range	Un-determined	Up to 3V

The MCP4921 is selected for the DAC, based on the supply voltage of 3.3V, which is compatible to the output voltage from the Arduino Due [3]. The Arduino Due is selected based on its four UART port capability. The Arduino Due can only handle up to 3.3V for each of the onboard pins. The DAC selection was also based on the 3.3V due to the 3.3V supply specification from the Arduino Due. The input voltage of the op-amp is limited to 3V based on the 3.3V supply rail. According to the datasheet, the MCP4921 is a DAC that operates from 0V to 3V to control the op-amp [3].

The DAC utilizes serial communication, which means it sends one bit at a time sequentially. The resolution of the DAC is 12 bit, which means it has a resolution of 4096. The purpose of the DAC is to convert digital inputs from the Arduino Due to analog output in order for the electronic load to function. MCP4921 is the DAC used for the OIS as shown in Figure 6 [3].

FIGURE 6: DAC INTERFACED SCHEMATIC

According to the resolution of the DAC, the voltages would correspond to the bits of the digital signal. As shown in Table 2, the digital bits correspond to the voltages ranging from 0V to 3A.

TABLE 2: DAC VALUE CONVERSION

Digital Input (From Arduino Due)	Analog voltage output (Power Supply)
4095	3
3413	2.5
2730	2
2048	1.5
1233	1
683	0.5

According to Figure 5, the op-amp creates a negative feedback loop from the gate of the MOSFET via the resistor then back to the negative terminal of the op-amp. Depending upon the input voltage from the DAC, the current conducted by the MOSFET and resistor varies. The output voltage from the Arduino goes up to 3.3V; therefore the range of the DAC output is set to vary from 0V to 3V. To sink in 0A to 2A, a resistor with value 1.5Ω is selected. The selection of the resistance value for the resistor is based on the input voltage from the DAC, MCP4921. The

input voltage will range from 0V to 3V. The current sink current will be from 0A to 2A. In order to sink in current up to 2A, the 1.5 Ω resistor is selected. The power rating is calculated to be $R \times I^2 = 1.5 \times (2^2) = 6W$. In order to increase the range, a 10W power-rating resistor was selected as shown in Table 3.

TABLE 3: THE SECTION OF THE RESISTANCE VALUE

	Design	Selected (71-RS)
Resistor value	1.5 Ω	1.5 Ω
Power rating	6W	10W

Base of the specification shown in Table 3 the FD050AN is selected for the MOSFET. The intended mode of operation of the MOSFET is at the continuous conduction mode at saturation mode. As shown in Table 4, at 125°C, the MOSFET can handle up to 73.5W [4]. The voltage threshold at 2A is around 5V, according to Figure 7 on the datasheet [4]. Shown in the testing data, the V_{GS} will be within 4V [4]. There are further temperature testing discussion and solutions presented in the “4.2.3 Thermal Testing” section. The testing showed that the OIS would require a fan to withstand the high current condition. The MOSFET on the OIS can possibly reach 125°C as an extreme condition [4]. However, it is obvious that at room temperature, the OIS can take much more heat before it overheats.

TABLE 4: THE SECTION OF THE MOSFET

	Design	Selected (FD050AN)
V_{DS}	>48V	60V
I_D	>2A	80A
P_{AVG}	>96W	245W @ $T \leq 25^\circ C$
		73.5W @ $T \leq 125^\circ C$

Table 5 shows the selection process of the op-amp, LM358. With the maximum input of 3V from the DAC, V_{DD} of the op-amp is calculated through the maximum input of the op-amp, 3V, added to the maximum V_{GS} , 5V, of the LM358 which is selected due to the wide range of supply and the under 0V GND sensing. $3V+5V=8V$ [5]. However, to expand the supply range of the op-amp, 2V was added to secure the functionality of the op-amp. $8V + 2V = 10V$. A supply voltage of 12V is selected for the op-amp to increase the stability of the electronic load.

TABLE 5: THE SELECTION OF THE OP-AMP

	Design	Selected (LM358)
V_{DD}	$\geq 10V$	3V~32V
Input GND Sense	$< 0V$	-0.3V
Output GND Sense	$\sim 2V$	5mV

According to the LM358 datasheet, the ground sensing capability, Input common mode voltage range starts from -0.3V [5]. This shows that the op-amp would still be on continuously even when the input of the op-amp is at 0V. It is crucial for the electronic load to be operating when the input from the Arduino Due is 0V. According to the LM358 datasheet, the output GND sense will be running continuously at 5mV, which ensures the op-amp running at the output when the input voltage is at 0V [5].

LM358 is selected based on the specification provided by MBARI [1]. The sensor node will take in voltage inputs: 12V, 24V or 48V. Due to the voltage inputs from the sensor node, the MOSFET of the OIS should be able to withstand up to 48V and the maximum current that the MOSFET should conduct at 2A. Therefore, the requirement of the MOSFET will have to be able to handle 2A. The power limitation of the MOSFET will be 75W. Specification provided by MBARI requires the OIS to handle up to 75W, therefore each individual sensor has to be able to handle up to 75W [1].

A cooling fan was selected after testing the electronic load. As shown in the Figure 7, the electronic load can heat up to 145°C without the fan. A fan was selected due to the power supply at 12V. The 12V power supply can be used to supply the op-amp and the fan. The fan was selected for the surface that the fan covers, 92x92x25 mm. The fan will cover the vertical surface of all the heat sinks, which are mounted onto the MOSFET, and dissipate the heat on the heat sink, as shown in Figure 8.

FIGURE 7: COOLING FAN

FIGURE 8: RELATIVE FAN SURFACE COMPARE TO OIS

FIGURE 9: ELECTRONIC LOAD TEST RESULTS

Figure 9 shows the result of the stable input from the power supply varying from 0V to 3V and the output of the electronic load from 0A to 2A. By varying voltages from 0V to 3V, the electronic load is able to sink in current ranges from 0A to 2A.

FIGURE 10: BLOCK DIAGRAM FOR SINGLE 2A AND LOAD AND SINGLE 6A LOAD

In order to achieve the fourth 0-6A load, three 2A electronic loads are connected in parallel, as shown in Figure 10. When the three 2A load are connected in parallel the current multiplies for channel 4 to sink in 6A load, as shown in the equation below.

$$6A = 3 \times 2A$$

The load will sink into the electronic load and will be the output current from the sensor node.

3.1.2 CURRENT SENSE CIRCUITRY

The current sense circuit is used to monitor the output current of the electronic load and shut down the OIS when the overall power of the OIS exceeds 75W. The LT1999 is selected due to its simplicity and practicality. To increase the resolution of the current sense circuit, the offset of the V_{REF} is set at 1V, which provides a resolution of 2V, 1volt to 3volt, for the ADC. The current

sense circuit can sense voltages ranging from 1V to 3V due to the offset of the V_{REF} . The original application of the LT1999 chip has an offset of 2.5V at V_{REF} , shown in Figure 11. In order to lower the V_{REF} , an additional 52.3k resistor is added, in parallel with the 160k Ω connected to ground, to lower the V_{REF} to 1V, as shown in Figure 12.

FIGURE 11: LT1999 BLOCK DIAGRAM [8]

FIGURE 12: CURRENT SENSE CIRCUITRY

The modified current sense circuit shown in Figure 12 will obtain 2V when the output current is 1A, the offset of 1V added to the 1V that the current sense circuit obtained. When the

output current is 2A, the current sense circuit will obtain 3V, the offset of 1A added to the 2V that the current sense obtained.

3.1.3 ADC INTERFACE

The ADC interfaces the current sense circuit and the Arduino Due. The onboard ADC is selected. The onboard ADC has a resolution of 12 bits, which provides resolution of 4096. The onboard ADC will take in the voltage from the current sense circuitry, 0volt to 3volts, and convert the analog value into digital value and the Arduino Due will calculate the overall power to ensure the board does not exceed 75W.

To ensure the functionality of the ADC, the voltage measured by the multi-meter from the current sense circuit should match the voltage read from the ADC, as shown in Table 6.

TABLE 6: ADC VALUE CONVERSION

Voltage from current sense resistor	Digital value
3	4095
2.5	3413
2	2730
1.5	2048
1	1233

3.1.4 4-20MA CURRENT LOOP TRANSMITTER

A single 4-20mA current loop transmitter is designed for only load #3, channel 3, on the OIS to communicate with 4-20mA instruments on the sensor node. The 4-20ma current loop transmitter, XTR117 [9], shown in Figure 13, from Texas Instruments is selected. According to the datasheet,

the chip is built around Eq. (1). According to the DAC voltage range from 0V to 3V, Eq. (1) is used to calculate the resistance of 15k.

$$I_o = \frac{100V_{in}}{R_{in}} \quad (1)$$

FIGURE 13: 4-20MA TRANSMITTER LOOP SCHEMATIC [9]

The 4-20mA will take in voltages ranging from 0V to 3V and convert the voltages to 0mA to 20mA. The following Figure 13 shows the test results of the 4-20mA current loop transmitter. The voltage to the current loop transmitter is supplied by the power supply and the receiving end of the current loop is represented by the 250Ω.

FIGURE 14: 4-20MA TRANSMITTER LOOP TESTING RESULT

3.2 SOFTWARE

The purpose of the software program is to provide a user interface platform for the user to command the OIS and monitors the current consumption on all active ports to ensure the OIS board does not exceed the maximum power of 75W. The three main hardware components for the software platform is the Arduino Due, the SD shield and the USB Port. The Arduino Due is used to initialize the OIS, to command the electronic load, and to calculate the overall power to ensure it does not exceed 75W. The SD shield includes the SD card. The SD card can store any sensor data and command sensor data. The USB port is used for testing purposes. In a complete design, the USB port will be replaced by the RS232 communication.

```
Main Menu
Please choose an option:
  1 - Select DC/DC converter used in Sensor Node
  2 - Configure individual port
  3 - Get port Configuration
  4 - Test individual port load
  5 - Test individual port communication
  6 - Run/Stop port
  7 - Read pH command
Select: █
```

FIGURE 15: MAIN MEAU SCREEN ON HYPERTERMINAL WINDOW

The main menu screen shown in Figure 15 is the terminal program that operates the OIS. Option 1 on the terminal program allows the user to specify the voltage of the sensor node, which could be 12V, 24V, or 48V. Option 2 allows the user to specify the desired configure channel and the desired functionality. The desired functionality could either be manual simulation or instrument simulation (described in the next subsection). Option 3 allows the user to check the status of each channel, when the program exceeds the power limit of 75W, the device will shut down immediately to protect the other hardware components on the OIS. Option 4 can test each individual port's electronic load. Option 5 can test each port's serial communication. Option 6 can run or stop individual ports. Option 7 is currently used for testing individual commands to simulate the SBE18s pH sensor. In the future, these commands will be incorporated into Option 5 for testing the serial communication.

FIGURE 16: SOFTWARE FLOW DIAGRAM OF INSTRUMENT SIMULATOR

Figure 16 is the software flow diagram built within the main menu shown in Figure 15. DC/DC conversion gives the user the option to select power input from the sensor node, 12V, 24V or 48V. Configure ports gives the user the option to select either manual mode or sensor mode. Get port configuration shows the port data. Test port load will obtain the port data. Test port communication allows each port to be tested in sensor mode. Run/Stop ports run and stop individual ports. For now, there is three different pH commands, for each command. Read pH_1 command can be read when the first port is used. Read pH_2 command can be read when the second port is used. Read pH_3 command can be read when the third port is used. The pH_1, pH_2, and pH_3 can be replaced with other sensors and conduct different simulations.

3.2.1 MANUAL MODE /SENSOR COMMANDS

The main menu of the OIS will appear on the computer screen after connecting the USB port to the computer. In order to start the OIS, the user will go through the main menu and configure each port in numerical order as shown in Figure 15.

After configuring the voltage level of the sensor node in option 1, there are two selections for option 2: the manual mode selection and the sensor command selection, as shown as Figure 17.

```
Select Instrument to be emulated on Port 1
  1 - pH Sensor
  2 - Manual Mode
  3 - Go back
Select: █
```

FIGURE 17: PH SENSOR AND MANUAL MODE SELECTION

The manual mode option will allow the user to configure the exact current that the user wants to simulate for the sensor behavior. The sensor command option, will allow the user to simulate sensor behaviors. As for now, the OIS only has pH sensor data stored. When pH sensor is selected, the OIS software program streams simulated data from the SD card. The three serial channels can simultaneously run the pH sensor, for each channel there will be a pH sensor read function in order to read the pH data as shown in Figure 18.

Figure 18 shows an example of the OIS responding to the sensor node's command of "gethd" directed toward the SBE18s pH sensor.

```

[plugged in instrument]
<POWERON/>
S> gethd<HardwareData DeviceType='SBE18S' SerialNumber='01801065'>
  <Manufacturer>Sea-Bird Electronics, Inc</Manufacturer>
  <FirmwareVersion>V1.2.0</FirmwareVersion>
  <FirmwareDate>Oct 21 2013</FirmwareDate>
  <PCBAsembly PCBID='64217' AssemblyNum='41819D' />
  <PCBAsembly PCBID='62222' AssemblyNum='41817' />
  <MfgDate>May 2 2013</MfgDate>
  <FirmwareLoader>SBE 18S FirmwareLoader V 1.0</FirmwareLoader>
  <PCBType>0</PCBType>
  <InternalSensors>
    <Sensor id='Water pH'>
      <type>PH0</type>
      <SerialNumber>01801065</SerialNumber>
    </Sensor>
  </InternalSensors>
_ </HardwareData>

```

FIGURE 18: EXAMPLE DATA FORMAT FOR PH SENSOR [2]

FIGURE 19: ARDUINO DUE CONNECTED TO A COMPUTER THROUGH A USB CONNECTION [2]

Figure 19 shows the Arduino Due connected to the computer through a USB connection.

3.2.2 RS232 SERIAL PORT COMMUNICATIONS/ TRANSCEIVER CHIP

MAX3323E is powered by 3.3V, supplied from the Arduino Due. RS232 is a commonly used protocol for the oceanographic instruments, which utilizes serial communication. The 1uF capacitors are used to reduce power supply noise. Figure 20 shows the schematic of the transceiver. Figure 21 shows the setup of the RS232 transceiver chip on the breadboard prior to the layout of the complete OIS board. Figure 22 shows the complete setup of the RS232 along with the Arduino Due and the SD card.

FIGURE 20: TRANSCEIVER SCHEMATIC

FIGURE 21: RS232 TRANSCEIVER SET UP [2]

FIGURE 22: ARDUINO DUE CONNECTED TO RS232 CONNECTOR [2]

3.2.3 SD CARD

The purpose of the SD card is to store up the sensor data so the OIS can acquire data when the sensor node requests data. Any sensor data can be stored in the SD card. The SD card is connected through the SD shield to the Arduino Due as shown in Figure 23.

FIGURE 23: ARDUINO DUE WITH SD CARD SHIELD ATTACHED [2]

CHAPTER 4: INTEGRATION TESTING

Integration testing is essential to ensure the functionality of the OIS board. The following test cases are designed to ensure the OIS board meets the specification and can withstand harsh conditions. The tests include the hardware integration, the hardware and software integration (parallel testing), and the thermal testing.

The purpose of the hardware integration testing is to ensure the hardware board is working properly prior to the integration with the software on the Arduino Due. Same for the software integration, in order to ensure the completion of the OIS, software was tested separately. Finally, after the integration of the hardware and software, thermal testing is conducted to ensure the OIS is protected from overheating. Figure 24 shows the testing diagram of the OIS. There are three transceivers used to test each serial port. Each transceiver will be connected to the USB serial converter then the UART port.

FIGURE 24: OIS TESTING BLOCK DIAGRAM

4.1 HARDWARE INTERFACE

The hardware interface includes the interface between the DAC and the electronic load, the current sense circuit and the ADC, the integration of the DAC, electronic load, current sense circuit and ADC, finally the DAC and the 4-20mA current loop transmitter. The hardware interface focuses on any inaccuracy, oscillation, and noise issue.

4.1.1 DAC & ELECTRONIC LOAD INTERFACE

The DAC controlled programmable current sink circuit is shown in Figure 5. A square wave is inputted to the op-amp to test the accuracy of the electronic load. As shown in Figure 25, there are oscillations at the gate of the MOSFET. The scope capture shown in Figure 25 has an input of 1Vpp square wave with an offset at 500mV. The yellow waveform represents the input square wave to the op-amp and the blue waveform represents the gate of the MOSFET. The gate voltage would vary due to the feedback loop and the input voltage. When the circuit is in ON-state of the square wave, high-frequency oscillation develops. The oscillations are seen here as a “thick band” in the gate voltage waveform, shown in Figure 25.

FIGURE 25: INPUT OF 1VPP (YELLOW); OUTPUT OF GATE VOLTAGE (BLUE)

Further research was done to analyze the frequency oscillation that develops within the closed

loop feedback, shown in Figure 26. Figure 25 shows the three capacitors within the small signal model of the MOSFET: C_{GD} , C_{GS} , and C_{DS} . C_{GS} is neglected due to the unity gain from the output of the op-amp to the source of the MOSFET. C_{DS} is relatively small compare to C_{GD} , therefore it has no affect compare to C_{GD} .

FIGURE 26: ELECTRONIC LOAD WITH SMALL SIGNAL CAPACITORS

The output resistance of the op-amp is approximately 25Ω according to the LM358 datasheet [5].

According to the FD050AN datasheet, the capacitance input is 3900pF and the capacitance output is 750pF [4].

FIGURE 27: CONCEPTUAL ELECTRONIC LOAD CONTROL LOOP

The presence of C_{GD} , approximately 3900pF, forms a low pass filter with the internal resistor from the op-amp LM358, 25Ω , as shown in Figure 11. When feedback is included as shown in Figure 27, the phase shift and gain will cause the oscillation to occur. The MOSFET at the output of the op-amp is shown to be DC grounded, therefore the MOSFET appears to be unity gain. The capacitor shown in Figure 27 is the C_{GD} from Figure 26. The resistor within the op-amp is the internal resistor of LM358. Figure 26 shows the conceptual electronic load control loop and shows the instability of the control through the low pass filter formed by the internal resistance within the op-amp and the capacitive load from the MOSFET. This issue could be easily prevented if the selection process of the op-amp was more careful. The instability within the feedback loop occurs due to the reduced phase margin caused by the parasitic pole.

FIGURE 28: LOW PASS FILTER WITH OP-AMP FEEDBACK

There are many ways to compensate for the instability caused by the feedback loop [8]. The approach chosen is to increase the bias current of the output stage thus lowering the output resistance of the op-amp and pushing the parasitic pole at higher frequency. The resistor R_{BIAS} is inversely proportional to the biased current, in other words R_{BIAS} decreases when there is higher current.

The resistor R_{BIAS} , $1k\Omega$ in the Figure 28 increases the bias current of the output stage of the LM358, op-amp. The large bias current caused the small output resistance at the op-amp, which removes the oscillation caused by the feedback loop. The functional block diagram of the LM358 [5] is shown in Figure 29.

FIGURE 29: ELECTRONIC LOAD WITH ADDITIONAL RESISTOR

The Darlington Pair of the push-pull stage shown in Figure 29 results with the biased current at 50uA. The output resistance of the last stage of the op-amp is estimated to be no less than 1k Ω .

FIGURE 30: SCHEMATIC DIAGRAM OF ONE-HALF OF LM358 [5]

As shown in Figure 30, Figure 30 the addition of the 1k Ω bias resistor resolves the oscillation issue.

FIGURE 31: INPUT OF 1VPP (YELLOW); OUTPUT OF GATE VOTLAGE (BLUE)

After testing the DAC, the DAC is integrated with the tested electronic load, shown in Figure 32.

FIGURE 32: DAC CONTROLLED ELECTRONIC LOAD SCHEMATIC

FIGURE 33: VOLTAGE INPUT FROM THE DAC VS. CURRENT OUTPUT TO THE 12V SUPPLY SENSOR NODE

Figure 33 shows the voltage input from the DAC verses the current output to the 12V supply sensor node. As shown in Figure 33, the input voltage ranging from 0V to 3V corresponds to the output current ranging from 0A to 2A.

4.1.2 CURRENT SENSE CIRCUIT & ADC INTERFACE

The current sense circuitry and the ADC are tested after integration, as shown in Figure 34. There is a ground voltage at 0 volt and a voltage reference which lowered to 1V with voltage division as shown below. Due to the different voltages, the troubleshooting took a while during this integration. Table 7 shows the voltage relationship between the ADC voltage and the current sense circuit current, the experimental data matches the expected data.

FIGURE 34: CURRENT SENSE CIRCUIT AND ADC SCHEMATIC

TABLE 7: CURRENT SENSE CIRCUIT VS. ADC

ADC [V]	Current Sense Circuit [A]
1.0	0
1.5	0.5
2.0	1
2.5	1.5
3.0	2.0

4.1.3 DAC & ELECTRONIC LOAD & CURRENT SENSE CIRCUITRY & ADC INTERFACE

The testing set up for the hardware integration is shown in Figure 35. The Arduino Due is used to control the input to the DAC and the drain of the MOSFET is connected to the 12V power supply. The result is shown in Figure 36, which displays the theoretical data, the experimental data reading from the power supply and the ADC data from the Arduino Due. The reading from the Arduino Due matches the theoretical data and the actual data result on the power supply, which represents the sensor node.

FIGURE 35: DAC AND ELECTRONIC LOAD AND CURRENT SENSE CIRCUIT AND ADC SCHEMATIC

FIGURE 36: INTEGRATION TESTING RESULT

4.1.4 DAC & 4-20mA CURRENT LOOP TRANSMITTER INTERFACE

4-20mA transmitter is a unique communication protocol commonly used in oceanographic instruments. In order to include this protocol into OIS, the 4-20mA transmitter is designed and implemented, as shown in Figure 37. Other than the increase in noise, the DAC and 4-20mA current loop transmitter interface went well. The result is verified and shown in Figure 38.

FIGURE 37: DAC 4-20mA CURRENT LOOP TRANSMITTER

FIGURE 38: DAC AND 4-20mA CURRENT LOOP TRANSMITTER RESULT

4.2 SOFTWARE AND HARDWARE SYSTEM INTEGRATION

After testing each individual part of the hardware board, the layout schematic was designed and implemented, as shown in Figure 39. Finally, the Software program on the Arduino Due was then integrated with the hardware board to do further testing.

There are two major modes of the OIS: The manual mode and the sensor mode. The manual mode is previous discussed in the set-up of the OIS. The sensor mode allows the user to select whichever sensor desired by the user. Finally the thermal testing ensures the board does not get burned when misused.

FIGURE 39: THE OIS CIRCUIT BOARD

4.2.1 PARALLEL MANUAL MODE TESTING

In order to show each channel does not interfere from each other, there are testing done separately on single 2A load, three 2A loads and the single 2A load on the OIS. The manual mode allows the user to simulate each channel up to full load (2A) for testing purposes. Manuel mode allows the user to run different channels at the same time within the total power limitation of 75W. (Within the power limitation of 75W, the user can input any desired current output and expect the entered current to be outputted.)

The parallel testing of multiple sensors is necessary to ensure the accuracy of the system. As shown in Figure 40, Figure 41, and Figure 42, the parallel testing result demonstrates the consistency of the accuracy of the OIS. The expected current is the current the user would input to the OIS (the expected output current from the sensor node) and the output current is the current that is displayed on the power supply (which should match the expected current). Figure 39, Figure 40, and Figure 41 show that the overall OIS system remains linear and accurate when all loads are running.

The following single 2A load, three parallel 2A load and single 6A load show the stability of the device. The expected output load matches the actual output load, which would be the output from the sensor node connected to the OIS. Therefore the figures shown below confirmed the stability of the device.

4.2.1.1 SINGLE 2A LOAD

FIGURE 40: SINGLE LOAD TEST

Figure 40 shows the linear relationship between the expected current and the output current.

4.2.1.2 THREE 2A LOADS

FIGURE 41: PARALLEL TESTING FOR THREE LOADS AT THE SAME TIME

Figure 41 shows the linear relationship between the expected current and the output current.

4.2.1.3 SINGLE 6A LOAD

FIGURE 42: PARALLEL TESTING FOR LOAD #4

Figure 42 shows the linear relationship between the expected current and the output current.

4.2.2 PARALLEL SENSOR DATA/ SENSOR MODE

In order to test the sensor data, all three channels with 2A load extract the pH sensor data from the SD card. Figure 17 shows the pH sensor format. The feedback of the pH sensor format confirms the working sensor mode. When three channels are running at the same time, there will be three displays on the computer screen to show the functional sensor mode, as shown in Figure 43. Similar to the parallel testing, the three pH sensors can run at the same time and do not interfere with each other. The pH sensor can also be replaced with an Oxygen sensor or Conductivity, Temperature, and Depth (CTD) sensor.

```
Main Menu
Please choose an option:
  1 - Select DC/DC converter used in Sensor Node
  2 - Configure individual port
  3 - Get port Configuration
  4 - Test individual port load
  5 - Test SD communication
  6 - Run/Stop port
  7 - Read pH_1 command
  8 - Read pH_2 command
  9 - Read pH_3 command
Select: █
```

FIGURE 43: PH SENSORS COMMAND

4.2.3 THERMAL TESTING

The thermal testing is required to better manage the temperature of the OIS and ensure the OIS board does not overheat and burn the components. The specification given by MBARI requires the board to not exceed 75W. Therefore when the board reaches 75W, the Arduino Due will automatically turn off.

The purpose of the thermal test is to ensure proper functionality of the OIS under a fully loaded condition. Excessive heat can cause instability of the system or even overheat the components used in the simulator. A sudden and excessive rise in temperature could cause failure of the electronic load. Thermal noise issues can be further reduced after adding a fan as well. Without the fan, a load up to 6A would go over 100°C in less than 7 minutes, as shown in Figure 44, result in the need of forced-air cooling. A load up to 2A would go over 80°C within 5 minutes. Two fans were selected to accommodate the anticipated heat. Each fan is 92x92x25mm, covering the surface of the heat sinks.

FIGURE 44: THERMAL TEST WITHOUT FAN

In the process of selecting the fan, the 12V supply is considered due to the simplification of the power supply needed for the OIS.

The 12V supply is recommended to be turned-on when the sink in current is greater than 1A. Currently, there is no control button or monitor to control the fan. The purpose of the fan will be ensuring the OIS is constantly under 45°C. With the fan on, as shown in Figure 45, a load up to 6A would remain around 45°C over a long period, thus providing the necessary stability of the entire system. There is currently no on/off control for the fan, but it is recommended when the current simulation ranges above 1A would need a fan. In the future, the fan can be monitored and be turned on when the temperature is above 45°C.

FIGURE 45: THERMAL TEST WITH FAN

TABLE 8: POWER BUDGET

Circuit	Part number	Voltage [V]	Current [mA]	# Used
DAC	MCP4921	3.3	0.35	5
Current-Sense Amplifier	LT1999	5	1.55	4
Dual Op-amp	LM358	12	6	5
Interface Circuit	Max3323	3.3	1	4
Fan	259-1464-ND	12	165	2
LED	WP710A10CGC K	3.3	20	1
	Estimated Value [mA]		Actual Value [mA]	
Total Current from 12V	360		378.47	
Total Current from 5V	6.2		6.17	
Total Current from 3.3V	25.75		28.5	

Table 8 shows the estimated power budget for the OIS. The OIS will be supplied with 3.3V and 5V from the Arduino Due and supplied with 12V from an external power supply (Hewlett Packard 6028A DC Power Supply 0-60V/0-10A, 200W). The estimated power budget matched the experimental power supply, shown in Table 8. The additional R_{BIAS} added to electronic load increases the power consumption of the op-amps. With the current rating around 50uA, the current draw by the $1k\ \Omega$ would cause an increase of 3uW, with four $1k\Omega$ there will be 12uW. The 12V HP power supply is used to represent the sensor node. Therefore with the sensor node present, there will be no need of the HP power supply.

FIGURE 46: CURRENT OUTPUT VS. POWER LIMITS

Figure 46 shows the power limitation on the OIS and when the power exceeds 75W, the Arduino Due will automatically shut down the device to ensure the hardware components do not overheat. The shut down feature is tested on the Arduino Due prior to the integration of the OIS board.

CHAPTER 5: CONCLUSION

Currently, the hardware and software design of the OIS are working properly and the OIS is providing MBARI engineers with an inexpensive means to test the swFOCE platform. The hardware design of the OIS could be improved to further increase the stability of the electronic load output and the software design could be improved with a graphical user interface. After the improvements, the OIS can then be interfaced with the swFOCE platform.

Although the OIS was designed specifically to test the swFOCE platform, the OIS can be used as a general oceanographic instrument simulator for other applications as it can be manually configured to emulate many different electronic loads and serial data streams. The OIS can serve as an excellent testing platform for the entire oceanographic community.

REFERENCES

- [1] Kechy, C.D.; Peltzer, E.T.; Walz, P.M.; Headley, K.; Herlien, R.A.; Maughan, T.; Matsumoto, G.I.; O'Reilly, T.C.; Salamy, K.A.; Shane, F.; Lovera, C.; Schofield, J.; Kirkwood, W.; Barry, J.P.; Brewer, P.G., "Open source instrumentation nodes for the greater oceanographic community," *Oceans - San Diego, 2013* , vol., no., pp.1,7, 23-27 Sept. 2013
- [2] A. Y. Chen, M. Mitchell, and R. Bae, "MBARI swFOCE Shallow Water Free Ocean Carbon Enrichment Experiment: Oceanographic Instrument Simulator," California Polytechnic State University, San Luis Obispo, Senior Project, Dec. 2014.
- [3] Microchip, "12-bite DAC with SPI Interface," MCP4921 datasheet, 2007
- [4] Fairchild Semiconductor, "N-Channel Power Trench MOSFET," FDP050AN06A0 datasheet, Oct. 2013
- [5] ON Semiconductor, "Single Supply Dual Operational Amplifiers," LM358 datasheet, Jan. 2015
- [6] Sunon, "MagLev Motor Fan", ME92251V1-000U-A99, 2014
- [7] Texas Instruments, "Implementation and Applications of current sources and current receivers," Application Bulletin March. 1999
- [8] S. Bendaoud, and G. Marino, "Practical Techniques to Aviod Instability Due to Capacitance Loading," Ask The Application Engineer – 32, June. 2012
- [9] Texas Instruments, "4-20mA Current-Loop Transmitter," XTR117 datasheet, May. 2012
- [10] Linear Technology, " High Voltage, Bidirectional Current Sense Amplifier," LT1999 datasheet, 2010
- [11] Sea-Bird Electronics, "SBE 18 pH Sensor," SBE 18 pH Sensor, 2015

APPENDICES

APPENDIX A – TESTING RESULTS

Given Value		Actual Testing Data				Expected Data		% error
V(DAC)	V _r	V _g	V _{gs}	I _{out} (0A-2A)	P (12V supply)	I _r =I _{mosfet}	P expected (12V)	
0.1000	0.9800	3.2900	2.3100	0.0650	0.7800	0.0667	0.8000	2.5000
0.2000	0.1980	3.4800	3.2820	0.1310	1.5720	0.1333	1.6000	1.7500
0.3000	0.2980	3.6300	3.3320	0.1980	2.3760	0.2000	2.4000	1.0000
0.4000	0.3980	3.7600	3.3620	0.2640	3.1680	0.2667	3.2000	1.0000
0.5000	0.4980	3.8800	3.3820	0.3310	3.9720	0.3333	4.0000	0.7000
0.6000	0.5980	4.0000	3.4020	0.3970	4.7640	0.4000	4.8000	0.7500
0.7000	0.6980	4.1100	3.4120	0.4640	5.5680	0.4667	5.6000	0.5714
0.8000	0.7970	4.2200	3.4230	0.5300	6.3600	0.5333	6.4000	0.6250
0.9000	0.8970	4.3300	3.4330	0.5960	7.1520	0.6000	7.2000	0.6667
1.0000	0.9960	4.4300	3.4340	0.6620	7.9440	0.6667	8.0000	0.7000
1.1000	1.0900	4.4500	3.3600	0.7300	8.7600	0.7333	8.8000	0.4545
1.2000	1.1900	4.5500	3.3600	0.7960	9.5520	0.8000	9.6000	0.5000
1.3000	1.2900	4.6600	3.3700	0.8620	10.3440	0.8667	10.4000	0.5385
1.4000	1.3900	4.7600	3.3700	0.9280	11.1360	0.9333	11.2000	0.5714
1.5000	1.4900	4.8500	3.3600	0.9950	11.9400	1.0000	12.0000	0.5000
1.6000	1.5900	4.9500	3.3600	1.0610	12.7320	1.0667	12.8000	0.5313
1.7000	1.6900	5.0300	3.3400	1.1270	13.5240	1.1333	13.6000	0.5588
1.8000	1.7900	5.1200	3.3300	1.1940	14.3280	1.2000	14.4000	0.5000
1.9000	1.8900	5.2100	3.3200	1.2600	15.1200	1.2667	15.2000	0.5263
2.0000	1.9900	5.3000	3.3100	1.3260	15.9120	1.3333	16.0000	0.5500
2.1000	2.0900	5.4100	3.3200	1.3940	16.7280	1.4000	16.8000	0.4286

2.2000	2.1900	5.4400	3.2500	1.4600	17.5200	1.4667	17.6000	0.4545
2.3000	2.2900	5.5300	3.2400	1.5260	18.3120	1.5333	18.4000	0.4783
2.4000	2.3900	5.6200	3.2300	1.5900	19.0800	1.6000	19.2000	0.6250
2.5000	2.4900	5.7300	3.2400	1.6560	19.8720	1.6667	20.0000	0.6400
2.6000	2.5900	5.8300	3.2400	1.7250	20.7000	1.7333	20.8000	0.4808
2.7000	2.6800	5.9100	3.2300	1.7920	21.5040	1.8000	21.6000	0.4444
2.8000	2.7800	6.0200	3.2400	1.8580	22.2960	1.8667	22.4000	0.4643
2.9000	2.8800	6.1400	3.2600	1.9250	23.1000	1.9333	23.2000	0.4310
3.0000	2.9800	6.2600	3.2800	1.9910	23.8920	2.0000	24.0000	0.4500

FIGURE 47: ELECTRONIC LOAD (WITHOUT CURRENT SENSE CIRCUITRY)
TESTING DATA

MONTEREY BAY AQUARIUM RESEARCH INSTITUTE
7700 SANDHOLDT ROAD
MOSS LANDING, CA 95039
(831) 775-1700 www.mbari.org

Title: Instrument Simulator		Number: <Doc>
Size: Engineer Amy Chen	Layout: <Layout Eng>	Filename: <Filename>
Date: Tuesday, January 19, 2016	Sheet: 3 of 5	Row: A

Tin/Rout : TTL CMOS
Tou/Rin : ELA232

MONTEREY BAY AQUARIUM RESEARCH INSTITUTE
7700 SANDHOLDT ROAD
MOSS LANDING, CA 95039
(831) 775-1700 www.mbari.org

Title: Instrument Simulator		Number: <Doc>	
Size: B	Engineer: Amy Chen	Layout: <Layout Eng>	Filename: <Filename>
Date: Tuesday, January 19, 2015	Sheet: 4	of: 5	Rev: A

MONTEREY BAY AQUARIUM RESEARCH INSTITUTE
 7700 SANDHOLDT ROAD
 MOSS LANDING, CA 95039
 (831) 775-1700 www.mbari.org
 Copyright MBARI 2000

Title Instrument Simulator			Number <Doc>	
Size	Engineer Amy Chen	Layout <Layout Eng>	Filename <Filename>	Rev
A				A
Date: Tuesday, January 19, 2016			Sheet 5 of 5	

APPENDIX C – ARDUINO CODE

Instr_Sim_Driver.h

```
// ----- MBARI OIS Project ----- */
// Instr_Sim_Driver.h
// Author: Amy Chen, Richard Bae, Matt Michell
//
// -----
#include <SD.h>
#include <SPI.h>

#define PH_DAC_ON 0.082 //1420
#define LENGTH 100 // default string length

#define CS_1 52
#define CS_2 50
#define CS_3 48
#define CS_4 46
#define SD_CS 44

#define ADC_1 A8
#define ADC_2 A9
#define ADC_3 A10
#define ADC_4 A11

#define VDD 3.3
#define DIGITAL_MAX 4095.0
#define VREF 0.95 // base voltage from the current sensing circuit
#define RESISTOR 1.5
#define NUM_SAMPLES 100

// Struct for each port configuration
typedef struct Port {
    String curInst; // current instrument
    double loadA;
    double loadV;
    String configName;
    boolean isRunning;
};

typedef struct DCConverter {
    int DCV;
    int DCW;
};

Port port1, port2, port3, port4; // Declaration of port objs
DCConverter convert; // DC/DC converter obj
int curCom; // Stores current user command input
String command; // user command in String object
String phCommand = ""; // string for ph serial data commands
File dataFile; // file that contains ph info
Sd2Card card; // sd card reader

/**
 * Initializes the SD card
 */
void Initialize_SD();

/**
 * Gets pH sensor command from pH sensor hyperterminal.
```



```

* The function then compares received string to a set
* of implemented commands. If the string matches a command,
* the function transmits static data from SD card to
* sensor hyperterminal.
*/
void Ph_Serial_Read();

/**
* Prompts user to input a number selection for the current menu.
* If the number is recognized, a menu option will be selected.
* If it is not recognized, the user will be prompted again.
* "return" key submits user's input. Both the 'd' and "return" keys delete the current
* input and re-prompts the user.
*
* @return an int representing the user's input selection
*/
int readNumInput();

/**
* Initializes each of the 4 port structs to 0.
*/
void Initialize_Ports();

/**
* Runs the DC/DC power menu. Allows the user to select
* DC voltage provided by sensor node and max allowed power
* consumption.
*/
void runDCSetPwrMenu(); // begins DC power menu

/**
* Stores DC power information after submitted by
* the user. Displays available sensor node output
* voltage options based upon max power selection
*
* @param pwrInfo power option selected by user in previous menu
*/
void DCSetVoltMenu(int pwrInfo);

/**
* Select instrument (sensor). Allows user to choose between
* available instruments for a specific port. User also has the
* option of selecting "Manual Mode" where they can specify
* their own instrument name and current.
*
* @ param thisPort port to attach selected instrument to
*/
void chooseInst(Port* thisPort);

/**
* Runs manual mode selection menu. User specifies what name
* and current consumption for current manual mode port.
*
* @param thisPort port to attach configure manual instrument to
*/
void runManualMode(Port* thisPort);

/**
* Select port menu.
*
* @return a pointer to the selected port struct
*/

```

```

Port *selectPort();

/**
 * Test port load menu. User selects running port to test load on.
 * Calculate drawn current and power consumption on selected port.
 */
void runTestPortLoad();

/**
 * Tests SD card communication. Confirms that sensor attached to port
 * can access serial data from SD card. Prints a test string from SD card
 * to confirm communication.
 */
void runTestPortComm();

/**
 * Menu to run or a stop a previously configured port. The function will
 * report an error is the selected port has not yet been configured.
 */
void runStopPort();

```

Instr_Sim_Driver.ino

```
// ----- MBARI OIS Project ----- */
// Instr_Sim_Driver.ino
// Author: Amy Chen, Richard Bae, Matt Michell
//
// -----

#include "Instr_Sim_Driver.h"
#include <SD.h>
#include <SPI.h>

void setup() {
  Serial.begin(9600); //Turning the Serial Protocol on
  Serial.println("Press any key to start");
  initDAC(); // get on board DACs
  Initialize_SD();
  Initialize_Ports();
}

void loop() {
  if (Serial.available()) {
    runMainMenu();
  }
}

void initDAC() {

  analogWriteResolution(12); // DAC 12-bit resolution
  analogReadResolution(12);

  SPI.setBitOrder(MSBFIRST);
  pinMode(CS_1, OUTPUT);
  SPI.begin(CS_1);
  SPI.setClockDivider(CS_1, 21); // 4 MHz clk
  pinMode(CS_2, OUTPUT);
  SPI.begin(CS_2);
  SPI.setClockDivider(CS_2, 21); // 4 MHz clk
  pinMode(CS_3, OUTPUT);
  SPI.begin(CS_3);
  SPI.setClockDivider(CS_3, 21); // 4 MHz clk
  pinMode(CS_4, OUTPUT);
  SPI.begin(CS_4);
  SPI.setClockDivider(CS_4, 21); // 4 MHz clk
}

void Initialize_SD() {

  // Ph_1 Sensor Hyperterminal Initialization
  Serial1.begin(9600);
  if(Serial1) {
    Serial1.println("[plugged in instrument]");
    Serial1.println("<POWERON/>");
  }
  // Ph_2 Sensor Hyperterminal Initialization
  Serial2.begin(9600);
  if(Serial2) {
    Serial2.println("[plugged in instrument]");
    Serial2.println("<POWERON/>");
  }
  // Ph_3 Sensor Hyperterminal Initialization
  Serial3.begin(9600);
  if(Serial3) {
```

```

    Serial3.println("[plugged in instrument]");
    Serial3.println("<POWERON/>");
}

pinMode(SD_CS, OUTPUT);
SPI.begin(SD_CS);
SPI.setBitOrder(MSBFIRST);
SPI.setDataMode(SD_CS, 2);
SPI.setClockDivider(SD_CS, 21); // 4 MHz clk

SD.begin();
card.init(SPI_HALF_SPEED, SD_CS);
digitalWrite(SD_CS, HIGH);
}

void Initialize_Ports() {
    convert.DCV = 0;
    convert.DCW = 0;

    port1.loadA = 0;
    port1.loadV = 0;
    port1.configName = "";
    port1.isRunning = false;

    port2.loadA = 0;
    port2.loadV = 0;
    port2.configName = "";
    port2.isRunning = false;

    port3.loadA = 0;
    port3.loadV = 0;
    port3.configName = "";
    port3.isRunning = false;

    port4.loadA = 0;
    port4.loadV = 0;
    port4.configName = "";
    port4.isRunning = false;
}

int readNumInput() {

    byte byteRead;

    Serial.print("Select: ");

    while (1) {
        byteRead = Serial.read();

        if ((byteRead >= 48 && byteRead <= 57) || byteRead == 10 ||
            byteRead == 13 || byteRead == 127 || byteRead == 100) { // Only take number inputs plus some characters

            if(byteRead == 127 || byteRead == 100) { // 100 = d (for delete)
                command = "";
                Serial.println();
                Serial.print("Select: ");
            }
            else {
                Serial.write(byteRead);
                command.concat(byteRead);
            }
        }
    }
}

```

```

if (byteRead == 10 || byteRead == 13) {
    Serial.println();

    if (command == "4813" || command == "4810") { // Command is '0' + CR or '0' + NL
        command = "";
        return 0;
    }
    else if (command == "4913" || command == "4910") { // Command is '1' + CR or '1' + NL
        command = "";
        return 1;
    }
    else if (command == "5013" || command == "5010") { // Command is '2' + CR or '2' + NL
        command = "";
        return 2;
    }
    else if (command == "5113" || command == "5110") { // Command is '3' + CR or '3' + NL
        command = "";
        return 3;
    }
    else if (command == "5213" || command == "5210") { // Command is '4' + CR or '4' + NL
        command = "";
        return 4;
    }
    else if (command == "5313" || command == "5310") { // Command is '5' + CR or '5' + NL
        command = "";
        return 5;
    }
    else if (command == "5413" || command == "5410") { // Command is '6' + CR or '6' + NL
        command = "";
        return 6;
    }
    else if (command == "5513" || command == "5510") { // Command is '7' + CR or '7' + NL
        command = "";
        return 7;
    }
    else if (command == "5613" || command == "5610") { // Command is '8' + CR or '8' + NL
        command = "";
        return 8;
    }
    else if (command == "5713" || command == "5710") { // Command is '9' + CR or '9' + NL
        command = "";
        return 9;
    }
    else {
        Serial.println("Wrong input. Please choose a number.");
        Serial.print("Select: ");
        command = "";
    }
}
}
}

void runMainMenu() {
    Serial.println();
    Serial.println("Main Menu");
    Serial.println("Please choose an option:");
    Serial.println(" 1 - Select DC/DC converter used in Sensor Node");
    Serial.println(" 2 - Configure individual port");
    Serial.println(" 3 - Get port Configuration");
}

```

```

Serial.println(" 4 - Test individual port load");
Serial.println(" 5 - Test SD communication");
Serial.println(" 6 - Run/Stop port");
Serial.println(" 7 - Read pH_1 command");
Serial.println(" 8 - Read pH_2 command");
Serial.println(" 9 - Read pH_3 command");

curCom = readNumInput();

if (curCom == 1) {
    curCom = 0;
    runDCSetPwrMenu();
}

else if (curCom == 2) {
    curCom = 0;
    Port curPort;
    if(convert.DCV != 0 && convert.DCW != 0){
        configPort();
    }
    else{
        Serial.println("Please configure DC/DC converter first from main menu");
        runMainMenu();
    }
}

else if (curCom == 3) {
    curCom = 0;
    runGetPortConfig();
}

else if (curCom == 4) {
    curCom = 0;
    runTestPortLoad();
}

else if (curCom == 5) {
    curCom = 0;
    runTestPortComm();
}

else if (curCom == 6) {
    curCom = 0;
    runStopPort();
}

else if (curCom == 7) {
    // Ph Data
    curCom = 0;
    Serial.print("Read Ph_1 command: ");
    Ph_Serial_Read_1();
    runMainMenu();
}

else if (curCom == 8) {
    // Ph Data
    curCom = 0;
    Serial.print("Read Ph_2 command: ");
    Ph_Serial_Read_2();
    runMainMenu();
}

```

```

else if (curCom == 9) {
    // Ph Data
    curCom = 0;
    Serial.print("Read Ph_3 command: ");
    Ph_Serial_Read_3();
    runMainMenu();
}

else {
    Serial.println("Invalid command. Please enter a valid command from the menu");
    runMainMenu();
}
}

struct Port* selectPort() {
    Serial.println("Choose a port:");
    Serial.println(" 1 - Port 1");
    Serial.println(" 2 - Port 2");
    Serial.println(" 3 - Port 3");
    Serial.println(" 4 - Port 4");

    curCom = readNumInput();

    if (curCom == 1) {
        curCom = 0;
        return &port1;
    }

    else if (curCom == 2) {
        curCom = 0;
        return &port2;
    }

    else if (curCom == 3) {
        curCom = 0;
        return &port3;
    }

    else if (curCom == 4) {
        curCom = 0;
        return &port4;
    }

    else {
        curCom = 0;
        Serial.println("Please choose correct port number.");
        selectPort();
    }
}

void runDCSetPwrMenu() {
    Serial.println("Please choose a DC/DC converter.");
    Serial.println("Select Power");
    Serial.println(" 1 - 10W");
    Serial.println(" 2 - 15W");
    Serial.println(" 3 - 20W");
    Serial.println(" 4 - 50W");
    Serial.println(" 5 - 75W");
    Serial.println(" 6 - Go back");

    curCom = readNumInput();
    if (curCom == 1) {

```

```

    curCom = 0;
    convert.DCW = 10;
    DCSetVoltMenu(1);
}
else if (curCom == 2) {
    curCom = 0;
    convert.DCW = 15;
    DCSetVoltMenu(2);
}
else if (curCom == 3) {
    curCom = 0;
    convert.DCW = 20;
    DCSetVoltMenu(3);
}
else if (curCom == 4) {
    curCom = 0;
    convert.DCW = 50;
    DCSetVoltMenu(4);
}

else if (curCom == 5) {
    curCom = 0;
    convert.DCW = 75;

    DCSetVoltMenu(5);
}

else if (curCom == 6) {
    curCom = 0;
    convert.DCW = 0;
    runMainMenu();
}

else {
    Serial.println("Please choose correct wattage.");
    runDCSetPwrMenu();
}
}

void DCSetVoltMenu(int pwrInfo) {
    Serial.println("Select Output Voltage");
    Serial.println(" 1 - 12V");
    Serial.println(" 2 - 15V");
    if (pwrInfo == 1 || pwrInfo == 2) {
        Serial.println(" 3 - Go back");

        curCom = readNumInput();

        if (curCom == 1) {
            curCom = 0;
            convert.DCV = 12;
        }
        else if (curCom == 2){
            curCom = 0;
            convert.DCV = 15;
        }
        else if (curCom == 3){
            curCom = 0;
            convert.DCV = 0;
            runDCSetPwrMenu();
        }
        else{

```



```

    Serial.println("Please choose correct Voltage from the menu");
    DCSetVoltMenu(1);
}
}
else if (pwrInfo == 3){
    Serial.println(" 3 - 24V");
    Serial.println(" 4 - Go back");
    curCom = readNumInput();

    if (curCom == 1){
        curCom = 0;
        convert.DCV = 12;
    }
    else if (curCom == 2){
        curCom = 0;
        convert.DCV = 15;
    }
    else if (curCom == 3){
        curCom = 0;
        convert.DCV = 24;
    }
    else if (curCom == 4){
        curCom = 0;
        convert.DCV = 0;
        runDCSetPwrMenu();
    }
    else{
        Serial.println("Please choose correct Voltage from the menu");
        DCSetVoltMenu(3);
    }
}

else if (pwrInfo == 4){
    Serial.println(" 3 - 24V");
    Serial.println(" 4 - 28V");
    Serial.println(" 5 - 48V");
    Serial.println(" 6 - Go back");

    curCom = readNumInput();

    if (curCom == 1){
        curCom = 0;
        convert.DCV = 12;
    }
    else if (curCom == 2){
        curCom = 0;
        convert.DCV = 15;
    }
    else if (curCom == 3){
        curCom = 0;
        convert.DCV = 24;
    }
    else if (curCom == 4){
        curCom = 0;
        convert.DCV = 28;
    }
    else if (curCom == 5){
        curCom = 0;
        convert.DCV = 48;
    }
    else if (curCom == 6){
        curCom = 0;

```

```

        convert.DCV = 0;
        runDCSetPwrMenu();
    }
    else{
        Serial.println("Please choose correct Voltage from the menu");
        DCSetVoltMenu(4);
    }
}

else if (pwrInfo == 5){
    Serial.println(" 3 - 24V");
    Serial.println(" 4 - 48V");
    Serial.println(" 5 - Go back");
    curCom = readNumInput();

    if (curCom == 1){
        curCom = 0;
        convert.DCV = 12;
    }
    else if (curCom == 2){
        curCom = 0;
        convert.DCV = 15;
    }
    else if (curCom == 3){
        curCom = 0;
        convert.DCV = 24;
    }
    else if (curCom == 4){
        curCom = 0;
        convert.DCV = 48;
    }
    else if (curCom == 5){
        curCom = 0;
        convert.DCV = 0;
        runDCSetPwrMenu();
    }
    else{
        Serial.println("Please choose correct Voltage from the menu");
        DCSetVoltMenu(5);
    }
}

else{
    Serial.write(pwrInfo);
    Serial.println(" is not supposed to be here.... Something is wrong :(");
    runMainMenu();
}
runMainMenu();
}

void configPort() {
    Port *curPort;
    curPort = selectPort();
    chooseInst(curPort);
}

void chooseInst(Port* thisPort) {

    Serial.print("Select Instrument to be emulated on Port ");

    if(thisPort == &port1){
        Serial.println("1");
    }

```

```

    }
    else if(thisPort == &port2){
        Serial.println("2");
    }
    else if(thisPort == &port3){
        Serial.println("3");
    }
    else if(thisPort == &port4){
        Serial.println("4");
    }
    else{
        Serial.println("You shouldn't be here... weird port number");
    }

    Serial.println(" 1 - pH Sensor");
    Serial.println(" 2 - Manual Mode");
    Serial.println(" 3 - Go back");

    curCom = readNumInput();

    Serial.print("Current Instrument: ");

    if (curCom == 1){
        curCom = 0;
        Serial.println("pH Sensor");
        thisPort->curInst = "pH Sensor";
        thisPort->loadA = PH_DAC_ON;
        runMainMenu();
    }
    else if (curCom == 2){
        curCom = 0;
        Serial.println("Manual Mode");
        thisPort->curInst = "Manual";
        runManualMode(thisPort);
        runMainMenu();
    }
    else if (curCom == 3){
        curCom = 0;
        configPort();
    }
    else{
        Serial.println("Please choose correct instrument from the menu");
        chooseInst(thisPort);
    }
}

void runManualMode(Port* thisPort) {

    byte byteRead;
    String configName = "";

    Serial.println("Please set load current in A");
    Serial.print("( '0.04' = 0.04A, value will not be displayed until 'return' key is pressed): ");
    Serial.setTimeout(100000);
    thisPort->loadA = Serial.parseFloat();
    Serial.read();
    Serial.println(thisPort->loadA, 3);
    if(thisPort == &port4){
        thisPort->loadA = ((thisPort->loadA) /3);
    }
    Serial.print("Please set name for this configuration: ");
    while(1) {

```

```

byteRead = Serial.read();
if(byteRead >= 33 && byteRead <= 126) {
    configName.concat((char)byteRead);
    Serial.write((char)byteRead);
}
else if(byteRead == 10 || byteRead == 13)
    break;
}

thisPort->configName = configName;
runMainMenu();
}

void runGetPortConfig() {
    Serial.println("DC/DC converter configuration");
    Serial.print("Current DC/DC converter max power setting: ");
    Serial.print(convert.DCW);
    Serial.println(" W");
    Serial.print("Current DC/DC converter output voltage setting: ");
    Serial.print(convert.DCV);
    Serial.println(" V");
    Serial.println();

    Serial.println("Current Port Settings");

    Serial.print("Port 1: ");
    if(port1.isRunning)
        Serial.println("running");
    else
        Serial.println("stopped");
    if(port1.curInst.compareTo("Manual") == 0){
        Serial.print("Port Configuration Name: ");
        Serial.println(port1.configName);
        Serial.print("Port Load Current: ");
        Serial.print(port1.loadA, 3);
        Serial.println(" A");
    }
    else{
        Serial.print("Current Port Instrument Mode: ");
        Serial.println(port1.curInst);
    }

    Serial.println();
    Serial.print("Port 2: ");
    if(port2.isRunning)
        Serial.println("running");
    else
        Serial.println("stopped");
    if(port2.curInst.compareTo("Manual") == 0){
        Serial.print("Port Configuration Name: ");
        Serial.println(port2.configName);
        Serial.print("Port Load Current: ");
        Serial.print(port2.loadA, 3);
        Serial.println(" A");
    }
    else{
        Serial.print("Current Port Instrument Mode: ");
        Serial.println(port2.curInst);
    }

    Serial.println();
    Serial.print("Port 3: ");

```

```

if(port3.isRunning)
    Serial.println("running");
else
    Serial.println("stopped");
if(port3.curInst.compareTo("Manual") == 0){
    Serial.print("Port Configuration Name: ");
    Serial.println(port3.configName);
    Serial.print("Port Load Current: ");
    Serial.print(port3.loadA, 3);
    Serial.println(" A");
}
else{
    Serial.print("Current Port Instrument Mode: ");
    Serial.println(port3.curInst);
}

Serial.println();
Serial.print("Port 4: ");
if(port4.isRunning)
    Serial.println("running");
else
    Serial.println("stopped");

if(port4.curInst.compareTo("Manual") == 0){
    Serial.print("Port Configuration Name: ");
    Serial.println(port4.configName);
    Serial.print("Port Load Current: ");
    Serial.print(port4.loadA*3, 3);
    Serial.println(" A");
}
else {
    Serial.print("Current Port Instrument Mode: ");
    Serial.println(port4.curInst);
}
runMainMenu();
}

void runTestPortLoad() {

    Port *temp;
    float readVal;
    int curADC;

    temp = selectPort();

    // check if port has been configured
    if(!temp->isRunning) {
        Serial.println("Please configure & run port before testing load");
        runMainMenu();
        return;
    }

    Serial.println();
    Serial.print("Load on ");
    if(temp == &port1) {
        Serial.print("Port 1");
        curADC = ADC_1;
    }
    else if(temp == &port2) {
        Serial.print("Port 2");
        curADC = ADC_2;
    }
}

```

```

else if(temp == &port3) {
    Serial.print("Port 3");
    curADC = ADC_3;
}
else {
    Serial.print("Port 4");
    curADC = ADC_4;
}
Serial.println(": ");

readVal = 0;
for(int i = 0; i < NUM_SAMPLES; i++) {
    readVal += analogRead(curADC) + 10.0; // the 10.0 accounts for some voltage loss in ckt
}
readVal = readVal / (1.0 * NUM_SAMPLES);

temp->loadV = ((float)(readVal) / DIGITAL_MAX * VDD - VREF);
//Serial.println(readVal);
Serial.print("  Current: ");
Serial.print(temp->loadV * 2 - VREF, 3);
Serial.println(" A");
Serial.print("  Power: ");
Serial.print((temp->loadV * 2 - VREF) * convert.DCV);
Serial.println(" W");

    runMainMenu();
}

void runTestPortComm() {

    byte byteRead;
    char nextChar;
    bool validCmd = false;

    // following 2 lines needed to re-initialize SD card
    SD.begin();
    card.init(SPI_HALF_SPEED, SD_CS);

    dataFile = SD.open("test.txt");

    if(dataFile) {

        while((nextChar = dataFile.read()) != '~') {
            if(nextChar == '\n')
                Serial.println();
            else
                Serial.print(nextChar);
        }
        Serial.println();
    }
    else {
        Serial.println("Error: could not open communication test file on SD card");
    }

    Serial1.println();
    //Serial2.println();

    SPI.end(SD_CS);
    runMainMenu();
}

```

```

void runStopPort() {

    Port *temp;
    byte sentByte1, sentByte2;
    unsigned short value;
    int curCS;
    byte msg1, msg2;

    Serial.println("Current port run/stop status:");

    Serial.print("Port 1 is currently ");
    if(port1.isRunning) {
        Serial.println("running");
    }
    else {
        Serial.println("stopped");
    }

    Serial.print("Port 2 is currently ");
    if(port2.isRunning) {
        Serial.println("running");
    }
    else {
        Serial.println("stopped");
    }

    Serial.print("Port 3 is currently ");
    if(port3.isRunning) {
        Serial.println("running");
    }
    else {
        Serial.println("stopped");
    }

    Serial.print("Port 4 is currently ");
    if(port4.isRunning) {
        Serial.println("running");
    }
    else {
        Serial.println("stopped");
    }

    Serial.println();
    Serial.println("Please choose a port to run/stop");

    temp = selectPort();
    if(temp->curInst == "") {
        Serial.println("Configure port before running/stopping");
        runMainMenu();
    }
    if(temp == &port1) {
        curCS = CS_1;
    }
    else if(temp == &port2) {
        curCS = CS_2;
    }
    else if(temp == &port3) {
        curCS = CS_3;
    }
    else {
        curCS = CS_4;
    }
}

```

```

value = (unsigned short)(DIGITAL_MAX / VDD * (temp->loadA * RESISTOR));

// for debugging digital value
//Serial.print("Digital value: ");
//Serial.println(value);

msg2 = (byte)value;
msg1 = (byte)(value >> 8);
msg1 = msg1 | 0b01110000;

if(temp->isRunning) {
    temp->isRunning = false;
    Serial.println("Port has stopped");
}
else {
    temp->isRunning = true;
    Serial.println("Port is running");
}

if(temp->isRunning) {
    digitalWrite(curCS, LOW);
    sentByte1 = SPI.transfer(curCS, msg1, SPI_CONTINUE);
    sentByte2 = SPI.transfer(curCS, msg2, SPI_LAST);
    digitalWrite(curCS, HIGH);
}
else {
    value = 0;
    msg2 = (byte)value;
    msg1 = (byte)(value >> 8);
    msg1 = msg1 | 0b01110000;

    digitalWrite(curCS, LOW);
    sentByte1 = SPI.transfer(curCS, msg1, SPI_CONTINUE);
    sentByte2 = SPI.transfer(curCS, msg2, SPI_LAST);
    digitalWrite(curCS, HIGH);
}

runMainMenu();
}

void Ph_Serial_Read_1() {

    byte byteRead;
    char nextChar;
    bool validCmd = false;

    Serial1.print("S> ");

    while (!Serial1.available())
        ;

    byteRead = Serial1.read();

    for(int i = 0; byteRead != 10 && byteRead != 13; i++) {

        if ((byteRead >=97 && byteRead <=122) ||(byteRead >= 48 && byteRead <= 57)
            || byteRead == 10 || byteRead == 13 || byteRead == 127) {

            phCommand.concat((char)byteRead);
            Serial1.write((char)byteRead);
            Serial.write((char)byteRead);
        }
    }
}

```



```

    }
    byteRead = Serial1.read();
}

// following 2 lines needed to re-initialize SD card
SD.begin();
card.init(SPI_HALF_SPEED, SD_CS);

if(phCommand.substring(0, 5) == "gethd") {
    //Serial.println("Valid text file");
    validCmd = true;
    dataFile = SD.open("ph_gethd.txt");
}
else if(phCommand.substring(0, 5) == "getcd") {
    validCmd = true;
    dataFile = SD.open("ph_getcd.txt");
}
else if(phCommand.substring(0, 5) == "getsd") {
    validCmd = true;
    dataFile = SD.open("ph_getsd.txt");
}
else {
    validCmd = false;
    Serial.println(" Invalid pH_1 command");
}

if(validCmd) {
    while((nextChar = dataFile.read()) != '~') {
        if(nextChar == '\n')
        {
            Serial1.println();
        }
        else
        {
            Serial1.print(nextChar);
        }
    }
}

Serial1.println();

phCommand = "";
SPI.end(SD_CS);
}

void Ph_Serial_Read_2() {

    byte byteRead;
    char nextChar;
    bool validCmd = false;

    Serial2.print("S> ");

    while (!Serial2.available())
    ;

    byteRead = Serial2.read();

    for(int i = 0; byteRead != 10 && byteRead != 13; i++) {

        if ((byteRead >=97 && byteRead <=122) ||(byteRead >= 48 && byteRead <= 57)
            || byteRead == 10 || byteRead == 13 || byteRead == 127) {

```

```

    phCommand.concat((char)byteRead);
    Serial2.write((char)byteRead);
    Serial.write((char)byteRead);
}
byteRead = Serial2.read();
}

// following 2 lines needed to re-initialize SD card
SD.begin();
card.init(SPI_HALF_SPEED, SD_CS);

if(phCommand.substring(0, 5) == "gethd") {
    //Serial.println("Valid text file");
    validCmd = true;
    dataFile = SD.open("ph_gethd.txt");
}
else if(phCommand.substring(0, 5) == "getcd") {
    validCmd = true;
    dataFile = SD.open("ph_getcd.txt");
}
else if(phCommand.substring(0, 5) == "getsd") {
    validCmd = true;
    dataFile = SD.open("ph_getsd.txt");
}
else {
    validCmd = false;
    Serial.println(" Invalid pH_2 command");
}

if(validCmd) {
    while((nextChar = dataFile.read()) != '~') {
        if(nextChar == '\n')
        {
            Serial2.println();
        }
        else
        {
            Serial2.print(nextChar);
        }
    }
}

Serial2.println();

phCommand = "";
SPI.end(SD_CS);
}

void Ph_Serial_Read_3() {

    byte byteRead;
    char nextChar;
    bool validCmd = false;

    Serial3.print("S> ");

    while (!Serial3.available())
    ;

    byteRead = Serial3.read();

```

```

for(int i = 0; byteRead != 10 && byteRead != 13; i++) {

    if ((byteRead >=97 && byteRead <=122) ||(byteRead >= 48 && byteRead <= 57)
        || byteRead == 10 || byteRead == 13 || byteRead == 127) {

        phCommand.concat((char)byteRead);
        Serial3.write((char)byteRead);
        Serial.write((char)byteRead);
    }
    byteRead = Serial2.read();
}

// following 2 lines needed to re-initialize SD card
SD.begin();
card.init(SPI_HALF_SPEED, SD_CS);

if(phCommand.substring(0, 5) == "gethd") {
    //Serial.println("Valid text file");
    validCmd = true;
    dataFile = SD.open("ph_gethd.txt");
}
else if(phCommand.substring(0, 5) == "getcd") {
    validCmd = true;
    dataFile = SD.open("ph_getcd.txt");
}
else if(phCommand.substring(0, 5) == "getsd") {
    validCmd = true;
    dataFile = SD.open("ph_getsd.txt");
}
else {
    validCmd = false;
    Serial.println(" Invalid pH_3 command");
}

if(validCmd) {
    while((nextChar = dataFile.read()) != '~') {
        if(nextChar == '\n')
        {
            Serial3.println();
        }
        else
        {
            Serial3.print(nextChar);
        }
    }
}

Serial3.println();

phCommand = "";
SPI.end(SD_CS);
}

```