MBARI swFOCE

Shallow Water Free Ocean Carbon Enrichment Experiment

Oceanographic Instrument Simulator

Senior Project

By Amy Chen, Byungjin Bae and Matt Mitchell

Advisor: Dr. Bridget Benson and Dr. Vladimir Prodanov

Sponsor: Chad Kecy from MBARI

ELECTRICAL & COMPUTER ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

Fall 2014

Table of Contents

Acknowledgement

We would like to give our special thanks to Dr. Bridget Benson and Dr. Vladimir Prodanov for advising the project. Throughout the entire project, we had weekly meetings with both Dr. Bridget Benson and Dr. Vladimir Prodanov to ensure we were on the right track. We would also like to thank Chad Kecy from MBARI who provided the project idea. Chad provided us with the parts we needed for our project and was available to meet on skype for video conferencing during the year.

Lastly, we would like to recognize all the Cal Poly faculty members for helping us through our undergraduate education.

List of Figures

List of Tables

Abstract

Our sensor node *Oceanographic Instrument Simulator* simulates various ocean sensors for MBARI's swFOCE (shallow water sensors node). The simulator system consists of an SD card, microcontroller board, and 4 electronic loads on a PCB that will be connected to the swFOCE sensor node. The current project can support a single node and simulates a pH sensor. 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 and search through the pre-installed data in 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.

1.0 Introduction

The Monterey Bay Aquarium Research Institute (MBARI) is known for their oceanographic research and building the equipment necessary to support the research. Since 2005, the engineers and the scientists at MBARI have been studying the problem of ocean oxidation due to increased absorption of CO_2.

MBARI established the Free Ocean Carbon Enrichment (FOCE) experiment to study the long-term effects of a decreased ocean pH by developing in-situ platforms. Deep FOCE (dpFOCE) was the first platform, which was deployed in 890m under water in Monterey Bay. After the conclusion of dpFOCE, MBARI deployed a shallow water FOCE (swFOCE) platform for shallow water experiments on FOCE.

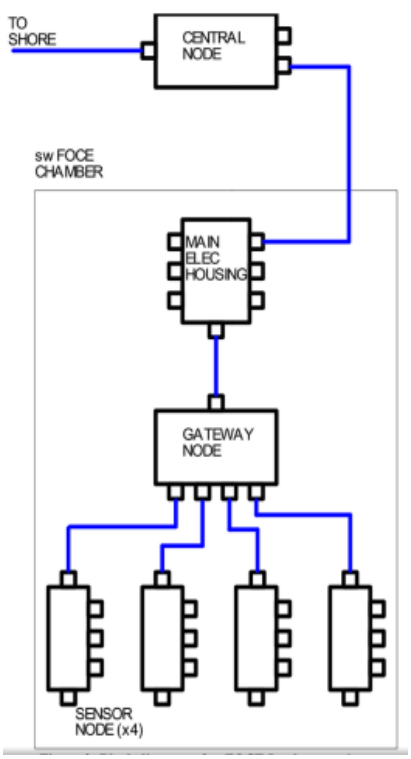


Figure 1: Overview of swFOCE device

1.1 Problem Statement

For the swFOCE experiment, a sensor will be used to measure various characteristics of the shallow ocean water such as pH level, oxidation level, etc. The sensor will be connected to swFOCE sensor node which is connected to 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. (See Figure 1) swFOCE sensor node is under development and a prototype board has been recently manufactured to test its compatibility with various oceanographic sensors for the experiment.

The main purpose of this project is to create a device that can simulate various oceanographic sensors to give the swFOCE project engineers a simple, easy-to-use, compact, and portable test system that they can use in their offices or benches instead of using actual sensors at a shallow water site.

The *Oceanographic Instrument Simulator* reduces the cost during the testing phase of research. When researchers test each individual sensor, they have to spend more money and time on purchasing and testing individual sensors. However, when all sensors are simulated a single PCB, the researchers can easily test the swFOCE system.

The *Oceanographic Instrument Simulator* will simulate various data outputs to test against the sensor node. The customers for the *Oceanographic Instrument Simulator* are usually scientists who study marine biology, marine geology, marine chemistry and marine technology.

1.2 Project Goal

The goal of this project is to develop simulated oceanographic instruments onto a single PCB. Instruments such as pH probes shall be observed and analyzed to develop a hardware simulator. A sensor node is a single PCB that functions as various sensors. The sensor node used in this project will be provided by MRARI, Monterey Bay Aquarium Research Institute. The *Oceanographic Instrument Simulator* is a single PCB board that will simulate data from each instrument and output the simulated data onto the sensor node. The *Oceanographic Instrument Simulator* PCB shall include simulated output data from each instrument and internal resistive behavior of each instrument. Through interfacing the instrument's data, the *Oceanographic Instrument Simulator* shall consist of 4 ports and have access to 4 ports simultaneously. The *Oceanographic Instrument Simulator* shall have manual mode. By entering the desired current, up to maximum allowed per port, the *Oceanographic Instrument Simulator* shall output an accurate data reading onto the sensor node.

2.0 Requirements and Specification

The following Table 1 shows the engineering specifications and marketing requirements for the *Oceanographic Instrument Simulator.*

Table 1: Oceanographic Instrument Simulator requirements and specifications

Marketing Requirements	Engineering Specifications	Justification
1	Simulate the electronic loads of following: Seabird SBE18s pH Sensor, Seabird SBE52 CTD Sensor, Seabird SBE43 DO Sensor, Teledyne Workhorse ADCP	The specific sensors and testing equipment are requested by the customer.
2	Interface 4 ports in order to have access to all 4 ports at the same time.	The user shall have access to 4 sensor ports simultaneously.
3	Prevent user from drawing more than 2A from port 1-3.	The current limit on port 1-3 on the sensor node is 2A.
3	Prevent user drawing more than 6A on port 4.	The current limit on port 4 on the sensor node is 6A.
3	Prevent user from drawing more than 75W total.	The power limit on the sensor node is 75 W.
4	Contain manual electronic load mode for user to enter desired current up to the maximum allowed current per port.	The customer request for manual mode. To prevent burned circuit, current above allowed current will not be provided while entering.
5	Ports 1-3 on the *Oceanographic Instrument Simulator* shall have serial communication.	The *Oceanographic Instrument Simulator* will be controlled by a microcontroller through serial communication.
6	Contain flashed front panel indicator	Front panel indicator shall be used to observe the voltage/current changes on the *Oceanographic Instrument Simulator*.
7	Power at 115VAC.	The *Oceanographic Instrument Simulator* will have 115VAC from sensor node.

Marketing Requirements

1. Contain variable electronics load for each port
2. Interface 4 sensor ports simultaneously
3. Have over voltage/ current protection circuitry
4. Include manual mode for user to input current values
5. Have serial communication
6. Functionality indicators
7. Use 115 VAC power supply

3.0 Functional Decomposition (Level 0 and Level 1)

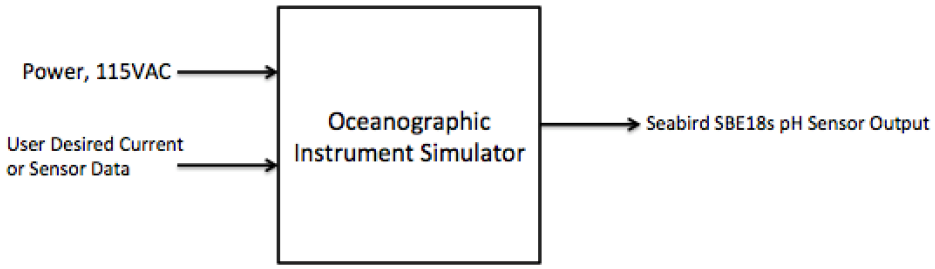


Figure 2: Oceanographic Instrument Simulator Level 0 Functional Decomposition Block Diagram

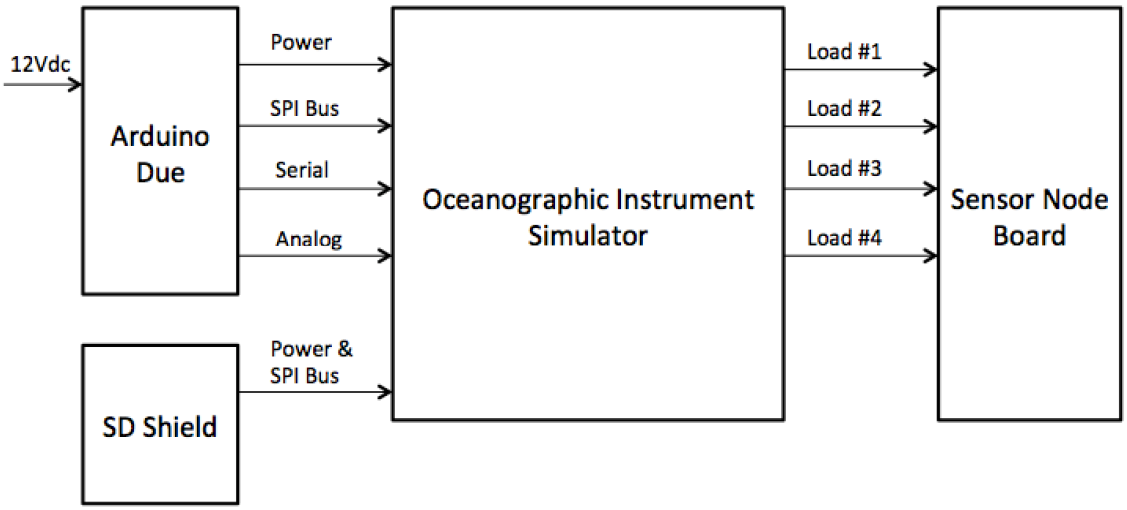


Figure 3: Oceanographic Instrument Simulator Level 1 Functional Decomposition Block Diagram

4.0 Project Design

4.1 Hardware Design

The hardware design for this project includes the design of the electronic load, DAC interface, current sense circuitry and the ADC interface.

4.1.1 Electronic Load

The electronic load design utilizes diode connected MOSFET with variableV_G. The op-amp will act like a voltage follower to prevent loading issue with its high impedance.

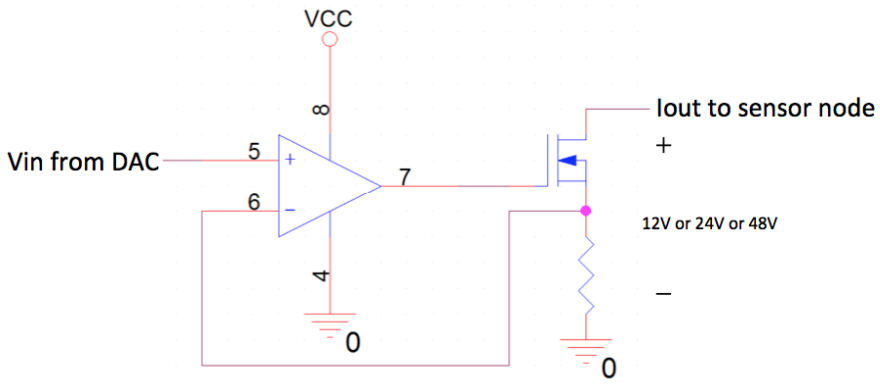


Figure 4: Electronic Load Schematic

Figure 4 shows the electronic load design. The op-amp will take in variable voltages from the DAC and supply the voltage to the gate of the MOSFET and the resistor. The current across the MOSFET and the resistor will vary depending on the input voltage from the DAC. The electronic load design will be used to simulate the output current for the pH sensor and output current varying from 0-2A, to the drain of the MOSFET.

When selecting the components used for the electronic load, we followed the requirements given from our sponsor, Chad, and determined the limits for each component. For example, the op-amp will take in 0-3V from the DAC. The LM358 op-amp was selected because the supply voltage ranges from 3V to 32V. The drain of the MOSFET will be supplied with variable voltages at 12V, 24V, or 48V. Due to the wide range of variable voltages at the drain of the MOSFET, LM358 was selected for the design.

We also need to select a power MOSFET that can handle voltage up to 48V. Due to the variable current input from 0-6A, the MOSFET needs to also take in up to 6A of current. Therefore, the specifications of the MOSFET include having 60V V_{DS} and over 6A I_D. FDP050AN06A0 was selected because of its high tolerance of I_D. FDP050AN06A0 can handle up to 80A.

The resistance of 1.5 Ω was selected due to the requirement of 0-2A. To utilize the maximum output from the DAC of 3V, due to Ohm's law. The 1.5 Ω resistor would be connected from the source of the MOSFET to the negative terminal on the op-amp, voltage follower, to ground.

$$V = IR$$

$$3V = (2A) \times R$$

$$R = 1.5\ \Omega$$

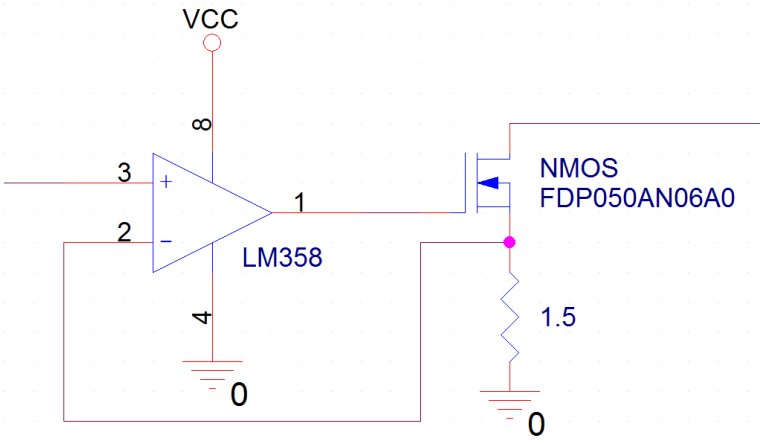


Figure 5: Electronic Load Schematic with selected components

Figure 5 shows the final electronic load design with a variable current output from 0-2A with the selected components part number and value.

4.1.2 DAC Interface

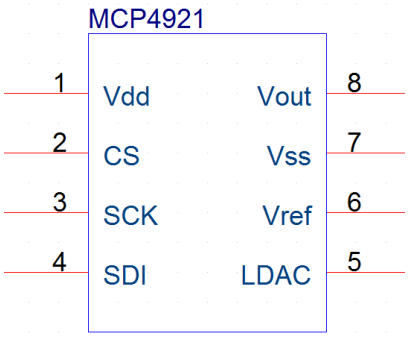


Figure 6: DAC Schematic

The selection of the DAC was difficult. Our original plan was to use a single DAC that can handle 4 ports. However, in the process, we had trouble interfacing the Arduino Due and the

single chip DAC with 4 ports. We then decided to select MCP4921 from Microchip, the DAC that we have previously used in class for other projects. MCP4921 is a single DAC. Therefore, we will need 4 DAC chips for 4 different loads.

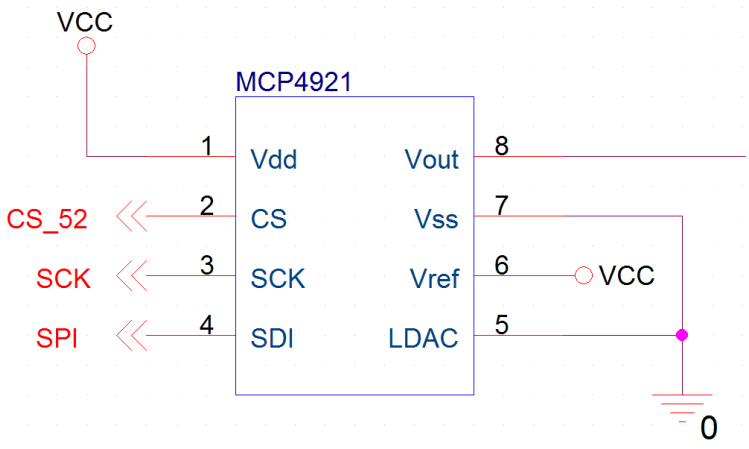
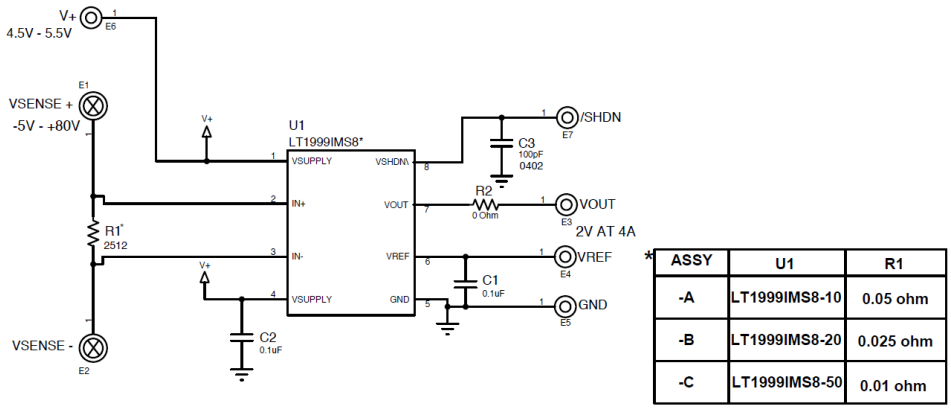


Figure 7: DAC interfaced schematic

Figure 7 shows the connections between DAC and the Arduino Due and the electronic load for the pH sensor. The DAC will take in signals from the Arduino Due from pin 52 for CS, pin SPI3 for SCK and pin SPI4 for SDI. The Vout will output 0V to 3V of voltage to the electronic load. The other 3 loads will share the same SCK and SPI with the pH sensor DAC, and depend on the load, different CS pins like CS_50, CS_48, and CS_46 will be assigned.

4.1.3 Current Sense Circuitry



ASSY	U1	R1
-A	LT1999IMS8-10	0.05 ohm
-B	LT1999IMS8-20	0.025 ohm
-C	LT1999IMS8-50	0.01 ohm

Figure 8: Current Sense Circuitry

Figure 8 is the current sense circuitry demo board schematic designed by Linear Technology. The chip will take in the current output to the sensor node and convert it to voltage with a ratio of 1:2. In our project, we utilize the current sense circuit to measure the output current and calculate the output power. One of the given requirements is to have the power limited to75W. In order to

ensure the power limit will be under 75W we did the calculations shown in Figure 9. The Arduino Due will then be programed to turn off the DAC before the power exceeds the limit.

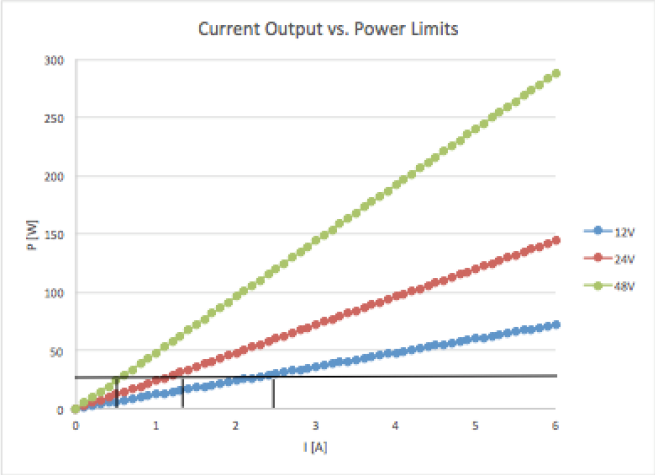


Figure 9: Current Output vs. Power Limits

Figure 9 shows the relationship between the current output to the sensor node and power limit for the *Oceanographic Instrument Simulator*. The 12V voltage input from the sensor node can handle any output current from 0-6A. The 24V voltage input from the sensor node can only handle up to 0-3.1A then the power will exceed 75W. The 48V voltage input from the sensor node can only handle up to 0-1.5A then the power will exceed 75W.

4.1.4 ADC Interface

To interface the ADC, we used the on board ADC on the Arduino Due. Pin numbers are A8, A9, A10 and A11. When interfacing the ADC with the current sense circuitry, we connected the onboard ADC, A8, to the Vout pin on the current sense circuitry. Pins A9, A10 and A11 will be used when additional sensors are used.

4.2 Software Design

The software design is also divided into 3 parts. We used the Arduino IDE for compiling the code which is a modified version of C++ that is used for programming Arduino microcontrollers. The IDE is supported on Windows, Mac, and Linux OS platforms. Since our program will be running on hyperterminal program, the operating system will not be a constraint for this project.

First, we have the *Oceanographic Instrument Simulator* set up program. With this program the user can set up the DC-DC converter, the type of simulated sensor (as of now the only supported sensor is the pH sensor), and the port that will be used. This program will run on a hyperterminal window in text only format (further explanation in 4.2.1).

The second part is the handling of sensor specific commands. The program will have a menu option that, when selected, will allow the Arduino to read in commands from a specific sensor. This process is emulated by a separate hyperterminal that sends a command to the main program and then receives sensor specific data from the onboard SD card. This communication is achieved through the use of RS232 cables and serial transceivers, one pair for each port (further explanation in 4.2.2).

Last is the load monitoring portion of the program. This program will run in the background to periodically check if the *Oceanographic Instrument Simulator* is running under the normal operation voltage and current range. If the sampled power is greater than maximum allowed power for the system, each port that was previously running will be stopped, and the user will be notified (further explanation in 4.2.3).

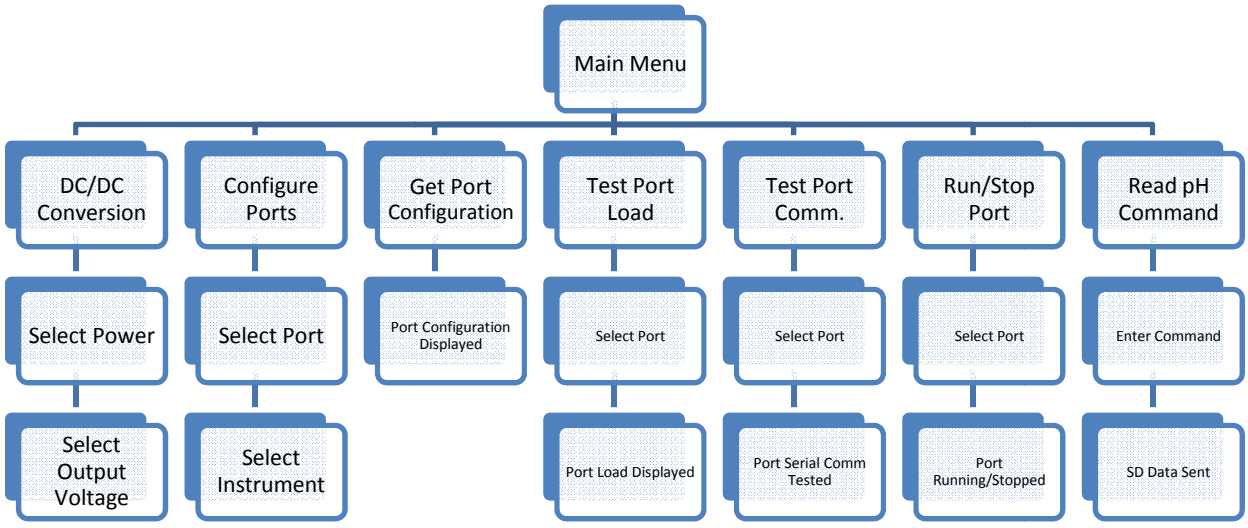


Figure 10: Software Flow Diagram of Instrument Simulator

4.2.1 Instrument Simulator Set-Up Program

This is main part of the program that the instrument simulator user will interact with. When the user connects the *Oceanographic Instrument Simulator* to his/her PC, and opens a hyperterminal window to communicate with the device, this set-up program will run. The flow of this program is represented in Figure 10.

This program will allow the user to set up DC to DC converter voltage and power, and then set up each of the 4 ports to any sensor that the user desires to simulate.

The user cannot configure ports unless he/she configures the DC to DC converter first. Also, without configuring the ports, user cannot run/stop ports, or test port load.

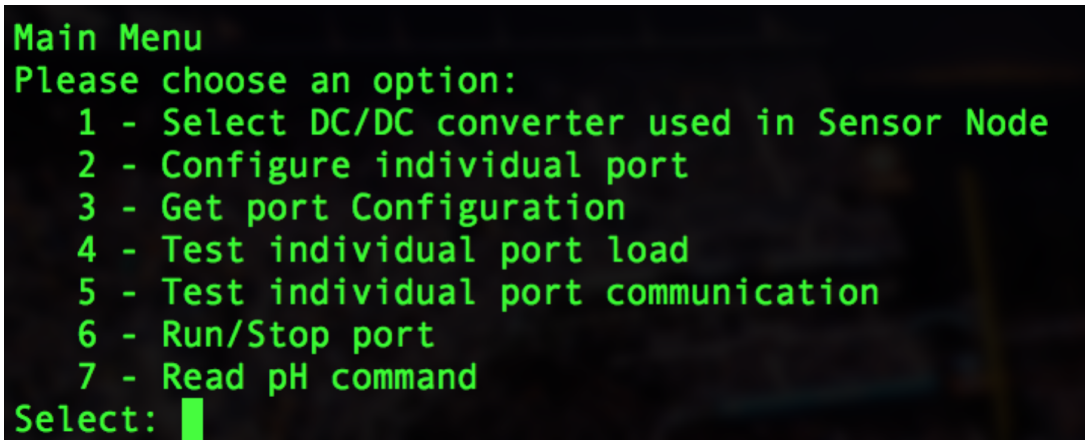


Figure 11: Main Menu Screen on hyperterminal window

4.2.2 Sensor Commands

This program will allow *Oceanographic Instrument Simulator* to act as a sensor instrument. When the appropriate main menu option is selected, option 7 in Figure 12 above for the pH sensor, the *Oceanographic Instrument Simulator* waits for a sensor command to arrive via the serial interface. This menu option prompts the sensor hyperterminal for a valid sensor command. Once the main program receives this command, a string compare function is run to check if the incoming command from the sensor is valid. Then, the program will check the SD card for the appropriate data and transmit the data back to the sensor hyperterminal.

For this, we assume that the user has proper simulation data for the sensors he/she wants to test stored in the SD card prior to using our instrument simulator. Figure 13 below shows the “gethd” command used by the pH sensor to get general hardware information about the pH sensor. This data should be stored inside the SD card.

```
[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>
   <PCBAssembly PCBID='64217' AssemblyNum='41819D'/>
   <PCBAssembly 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 12: Example data format for pH sensor

4.2.3 Load Monitoring

This portion of the main program is in charge of receiving current and voltage information from the load monitoring circuit and checks to see if the voltage and current level is under the limit of the system. If the current or voltage exceeds the limit, the program will shut down the port that exceeds the limit.

This program will make use of the Arduino's internal ADC and an external DAC to interface between the Arduino and the load monitoring circuit.

Through the DAC, the Arduino will open a port with a desired voltage from the user. The 12-bit DAC will convert 12-bit digital values, ranging from 0 - 4095, into an analog voltage, which range from 0V up to 3.3V. This voltage will go through the load monitoring circuit and the circuit will constantly return the current value back to Arduino through the ADC. The Arduino will read in the voltage value back from the ADC as a digital value from 0 - 4095 since the ADC is set to 12-bit precision.

4.2.4 External Components Interface

This project requires a board that can support up to 4 serial connections. Also, it should be able to have access to an SD card or other data storage system which will store the sensor data. With the requirement for the hardware in mind, we chose the Arduino Due. Arduino Due supports up to 4 serial connections including a USB connection with a computer. Also, Arduino Due has an SD card shield which we can interface easily with the microcontroller board by just attaching the SD card shield on top as in Figure 2.

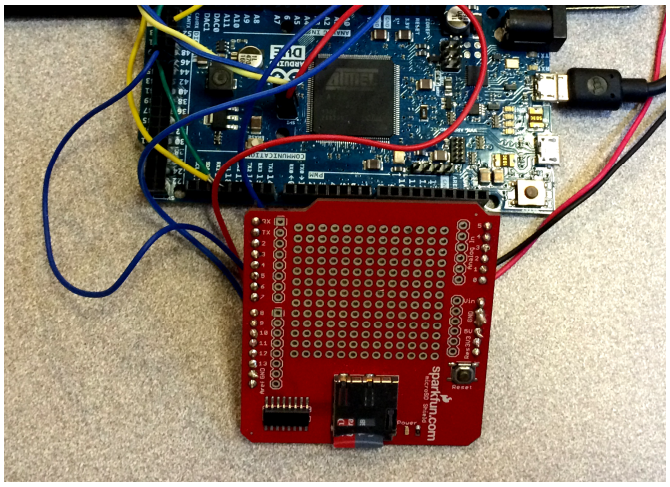


Figure 13: Arduino Due with SD card shield attached

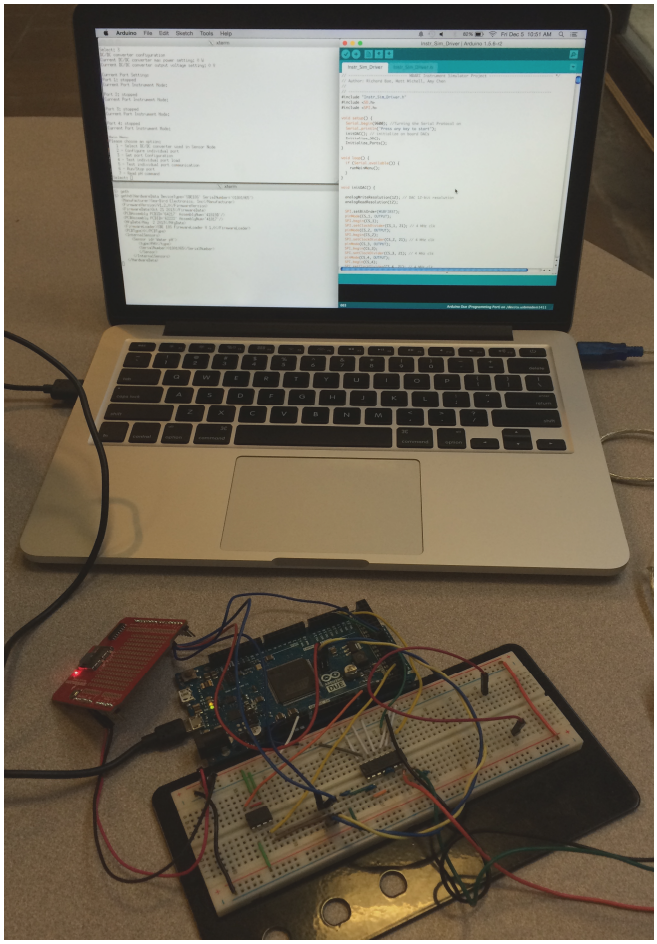


Figure 14: Arduino Due connected to a computer through a USB connection

Next is communication. Arduino Due does not come with any serial connector that we can use out of the box. So we decided to set up a serial connection by building a RS232 connector and attaching it to the microcontroller.

To add a RS232 connection to the Arduino Due, we used a RS232 transceiver chip, MAX3323EEPE to interface a RS232 female connector to the Arduino Due. This requires 4 100uF capacitors as well. Figure 4 shows the RS232 connector for Arduino Due.

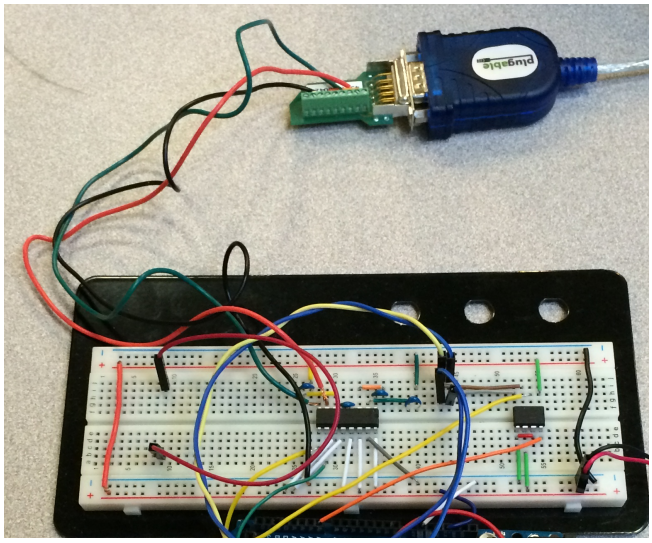


Figure 15: RS232 Transceiver set up

Another computer will be connected on the other side of the RS232. That computer will simulate a sensor that will send appropriate commands to our Arduino Due so our instrument simulator will fetch the requested data from the SD card.

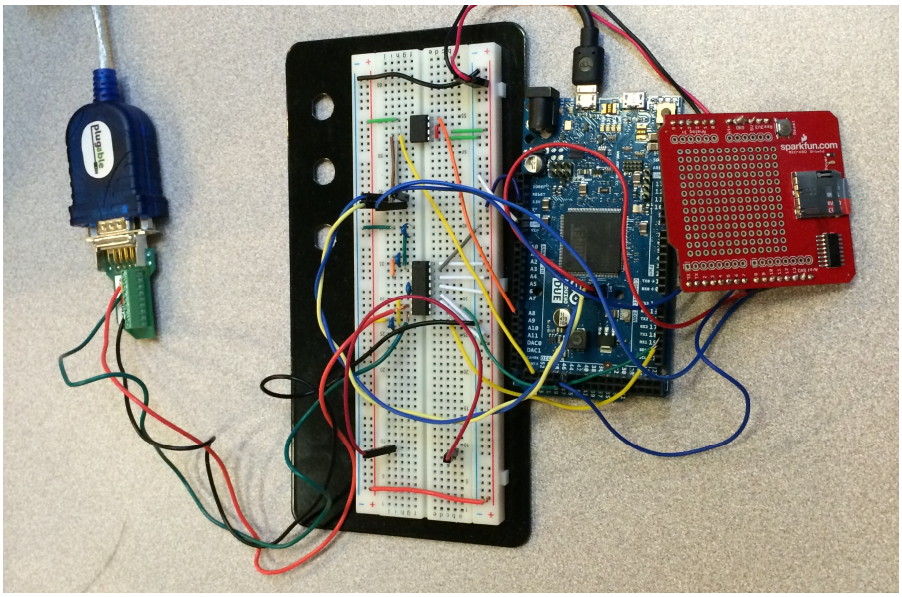


Figure 16: Arduino Due connected to RS232 connector

Lastly, we have a load monitoring circuit that will constantly check whether our device is exceeding the maximum current and voltage.

5.0 Testing

In this section, we will cover the testing methods and results of the *Oceanographic Instrument Simulator*.

5.1 Electronic Load Testing

The electronic load will take in 0V to 3V of input voltage from the power supply. According to Ohm's Law, we verified the output current. The output current should vary between 0-2A for the pH sensor. Figure 19 shows the theoretical current output vary from 0-3V divided by 1.5 Ω.

Sample Calculation:

$$V = IR$$

$$1.5V = I \times (1.5)$$

$$A = 1\ A$$

When testing the circuit, there are several times when the power supply to the op-amp was hitting the rail, and the current from the supply was hitting the rail. To solve the problem, we increased the supply voltage of the op-amp. We kept the supply at 13V instead of 5V, since the rail of the amplifier needs to be higher than $V_{GS,}$ and kept the current limit on the power supply at 2.1A instead of 2A.

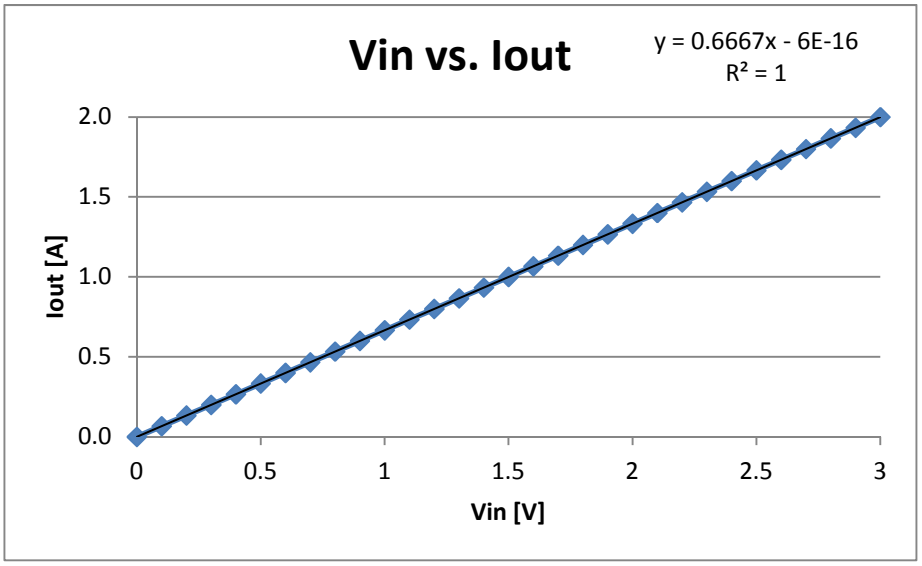


Figure 17: Voltage input from the power supply vs. theoretical current output to the sensor node

Figure 19 shows the relationship between the voltage inputs from the power supply, which will later be replaced by the DAC, and the theoretical current output to the sensor node with the input voltage of 12V from the sensor node.

5.2 DAC Testing

When testing the interface between the DAC and the electronic load, we first verified the output from the DAC with a multimeter. Through manually typing a value from the user interface, we obtain a similar result from the multimeter. After we verified the accuracy of the DAC chip, we interfaced the electronic load with the DAC and measured the current output obtained from the 12V power supply as shown in Figure 20.

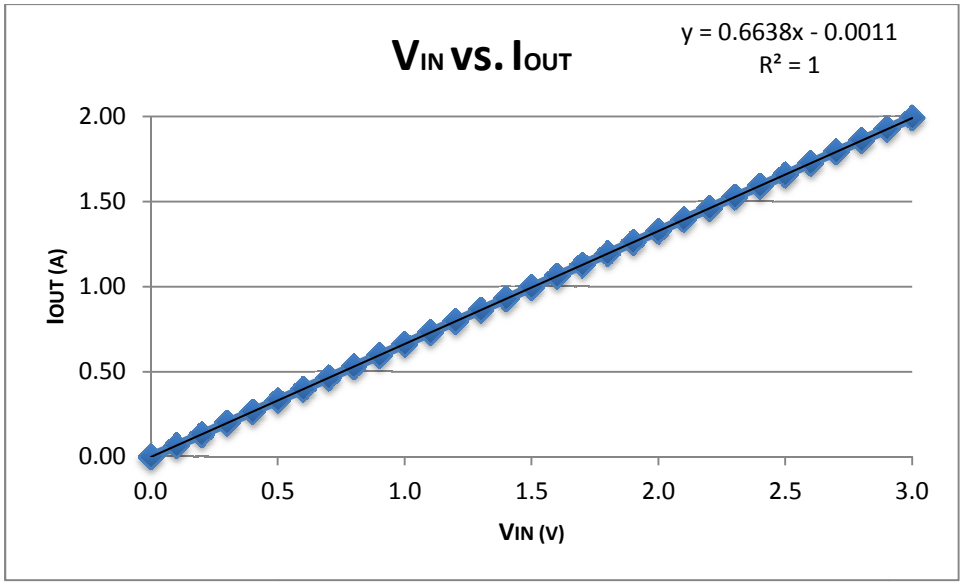


Figure 18: Voltage input from the DAC vs. current output to the 12V supply sensor node

Figure 20 shows the experimental data of the voltage input from the DAC and the current output to the 12V supply sensor node. We can see the equation shown in Figure 18 and Figure 19 are closely related, which verified the theoretical analysis of the electronic load and the functional DAC.

5.3 Current Sense Circuit Testing

The testing set up for the current sense circuit includes the DAC and the electronic load circuit.

After testing the DAC and electronic load, we then add on the current sense circuitry to the hardware design. The purpose of the current sense circuitry is to obtain the current output to the sensor node and use the current output to calculate the power limit, ensure the *Oceanographic Instrument Simulator* will be turned off when exceed the power limit. The current sense circuit has a voltage to current ratio of 1:2. Through the supply voltage from the DAC, the 12V power supply, which represents the sensor node, will display the current drawn from the power supply and the current sense circuit will obtain the voltage value with the 1:2 ratio. We will then verify the voltage obtained from the current sense circuit with the current drawn on the power supply to ensure the working current sense circuit.

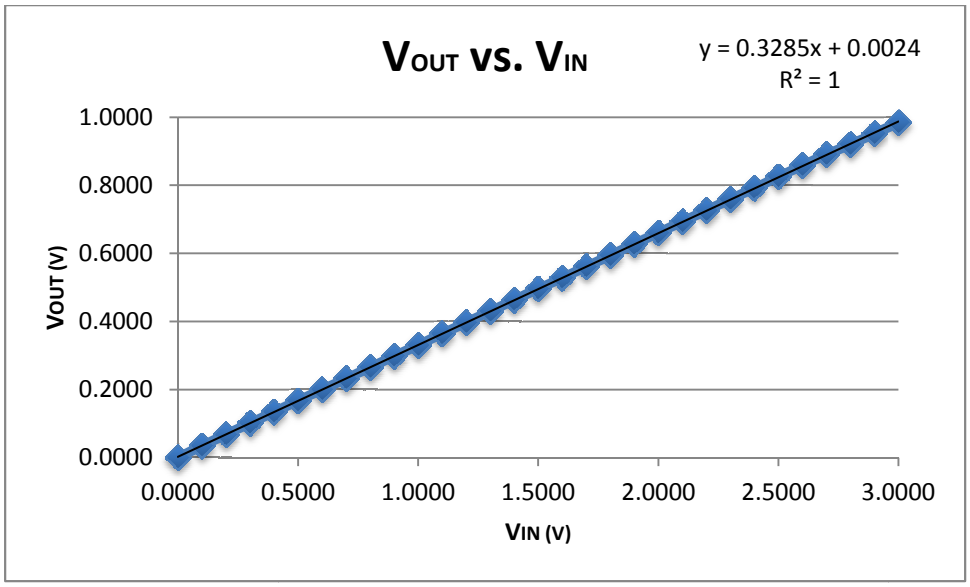


Figure 19: Electronic load (with current sense circuitry) result plot

Figure 21 shows the relationship between the input voltage from the DAC and the output voltage translated from the current sense circuit to the ADC. The input voltage is supplied by the DAC and the output voltage is measured from the multimeter from the current sense circuit with a 1:2 ratio of voltage to current. To measure the output current, we utilize the current sense chip provided by Linear Technology. We can measure the current through taking the output voltage from the sensor node. The output voltage will be taken in by the ADC and be used to calculate the power limit.

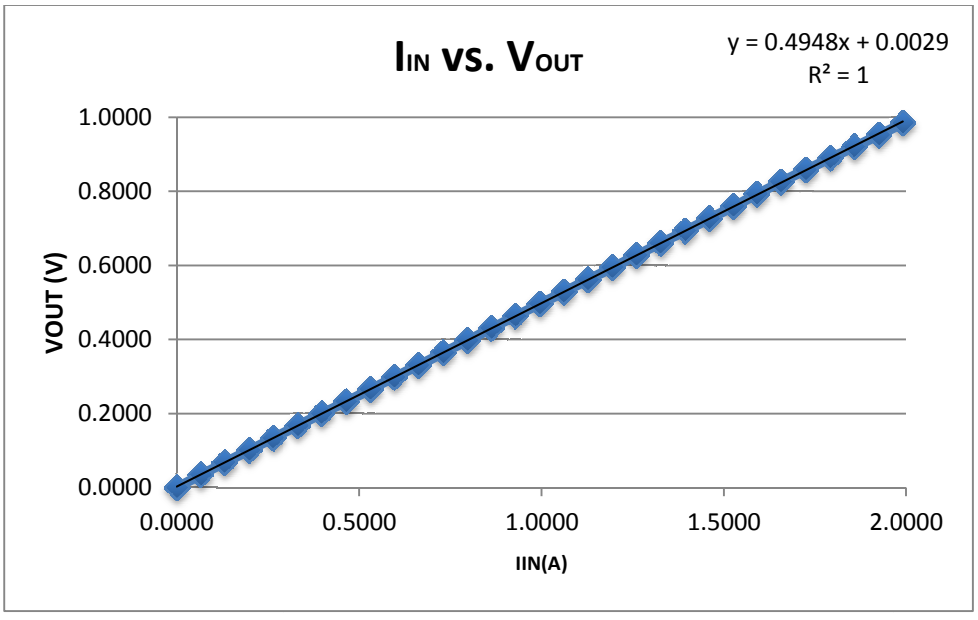


Figure 20: Current input from the power supply vs. voltage output from the multimeter reading

Figure 22 shows the relationship between the current input from the power supply, the sensor node and the voltage output from the multimeter reading, which will then be feedback to the ADC. The relationship between the current input from the power supply, the sensor node and the voltage output from the multimeter reading should be 1:2. Through the graph, we can see a 1:2 ratio relationship between the current and the voltage which verified the current sense circuit is working.

5.4 ADC Testing

After we verified the current sense circuitry, we interfaced the voltage output from the ADC to the Arduino and obtained the voltage from the user interface. When interfacing the ADC and the Arduino, we found out that the voltage output is offset at 2.5V. To solve the problem, we had to add a pull-down resistor to pull the Vref down to 1V. We decided to have the pull-down resistor with a resistance of 5.1k ohm to pull down 1.5V.

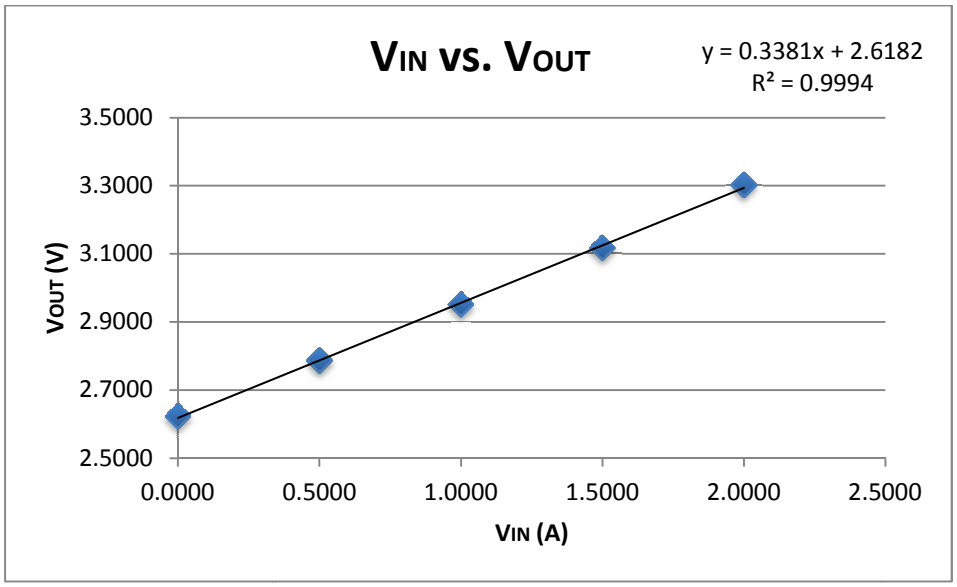


Figure 21: Current sense voltage output vs. DAC voltage input

Figure 23 shows the relationship between the voltage from the ADC, supplied from the DAC, ranges from 0V-2V and the Vout on the current sense circuitry. Due to the Vref offset at 2.6V and the DAC voltage limitation, we were only able to take measurements up to 3.3V. However, the equations displayed shows a fairly linear curve.

The final voltage offset on the output of the current sense circuit is 1V. To obtain the accurate current output to the sensor node, we will have to subtract the voltage output by 1V and multiple the voltage by 2 to get the current output, due to the offset of 1V and the 1:2 voltage output to current output on the current sense circuit. The calculations are shown below.

5.5 Communication Testing

For RS232 connectors, we used another computer with a hyperterminal window to confirm communication functionality. We opened a hyperterminal window on our main user computer, and then we connected the other side of RS232 connector to another computer.

5.7 Integration Testing

To make sure all parts of the programs work, we combined all the codes into our final project code and ran it on Arduino. We conducted same tests we used in previous unit tests to make sure that they still work in one integrated program.

For the current phase of the project, we have the Arduino Due to have another serial connection which can be connected to another serial device and be able to communicate between the main user terminal and the added serial device. We also have set up the ADC and DAC so they can receive analog voltage value from the load monitoring circuit and send analog voltage to the load monitoring circuit.

We will soon have Arduino Due read data from an SD card and receive voltage from the load monitoring circuit. More ports will be supported and we will have port 3 to support 4-20mA analog communication and port 4 to support up to 6A current to meet the requirements.

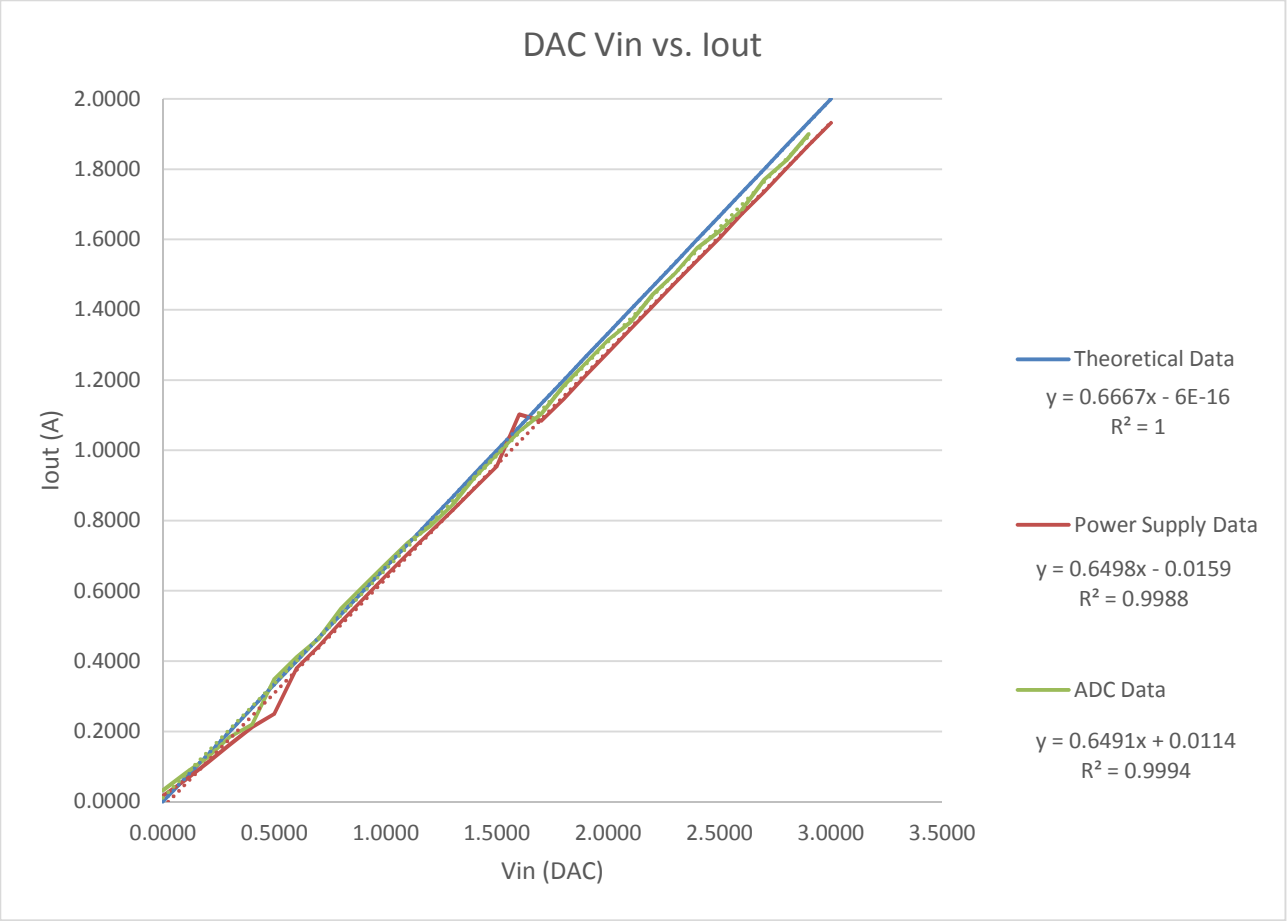


Figure 22: Integration testing result

In Figure 24 above, shows the relationship between the input voltage from DAC and the output current from the simulator, the theoretical data matched closely with the experimental data despite the long wires we had. Vin represent the voltage input from the DAC. Iout represents the current output to the sensor node. Theoretical data is the calculated current values. Power supply data is the actual values shown on the power supply that supplies 12V for pH sensor. ADC data is the reading Iout from the Arduino Due.

6.0 Conclusion

Currently, we have interfaced the pH sensor between Arduino Due and the load monitoring circuit. The printed circuit board design at the end of the project includes three loads that can handle 0-2A and one that can handle 0-6A.

The pH sensor's data from SD card can now be retrieved and sent over to the serial connection. Also, we have a text-based user interface displayed on a hyperterminal window. The user can interact with the interface to configure the *Oceanographic Instrument Simulator* and simulate the SBE18s pH sensor.

We have built a prototype version of the *Oceanographic Instrument Simulator* that simulates the SBE18s pH sensor. In the future, the project can possibly expand the number of supported ports from one to four, and include a 4-20mA connection.

7.0 References

[1] Arduino Due, Available: http://arduino.cc/en/Main/arduinoBoardDue

[2] LM358NG, "Single Supply Dual Operational Amplifiers," ON Semiconductor, October 2013. Available: http://www.onsemi.com/pub/Collateral/LM358-D.PDF

[3] FDP050AN06A0, "N-Channel Power Trench MOSFET," Fairchild Semiconductor, 2013. Available: https://www.fairchildsemi.com/datasheets/FD/FDP050AN06A0.pdf

[4] LT1999, "Demo Manual DC 1698A-C: High Voltage Bidirectional Current Sense Amplifier," Linear Technology, October 2010. Available: http://cds.linear.com/docs/en/demo-board-manual/dc1698afa.pdf

[5] MCP4921, "12-Bit DAC with SPI Interface," Microchip, 2004. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21897a.pdf

[6] MAX3323, "RS-232- Transceiver for Multidrop Applications," Maxim, 2003. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21897a.pdf

[7] pH Sensors, "Seabird SBE18s pH Sensor," Seabird Electronics Inc, 2004. Available: http://www.seabird.com/pdf_documents/Datasheets/18brochureDec12.pdf

[8] CTD Sensors, "Seabird SBE52 CTD Sensor,' Seabird Electronics Inc, 2012. Available: http://www.seabird.com/pdf_documents/manuals/52_010.pdf

[9] DO Sensors, "Seabird SBE43 DO Sensor," Seabird Electronics Inc, 2012. Available: http://www.seabird.com/pdf_documents/Datasheets/43brochureSep13.pdf

[10] Chen, Ke-Horng. Chien, Chieh-Ching. Ho, Hsin-Hsin. Huang, Li-Ren. "Optimum Power-Saving Method for Power MOSFET Width of One-Cycle Control DC-DC Converters," Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1711780

[11] Wang, Hongfang. Wang, Fred. " A Self-powered Resonant Gate Driver for High Power MOSFET Modules," Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1620537

[12] Yoshida, Isao. Katsueda, Mineo. Maruyama, Yasuo. Kohjiro, Iwamichi. "A HIGHLY EFFICIENT 1.9-GHz Si HIGH POWER MOSFET," Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5436230

[13] Evmorfopoulos, Nestoras. Karampatzakis, Dimitris. Stamoulis, Georgios. "Precise Identification of the Worst-Case Voltage Drop Conditions in Power Grid Verification," Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4110162

[14] Kazerani, M. "A High-Performance Controllable DC Load," Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4374737

[15] Franco, S. "Design with Operational Amplifiers and Analog Integrated Circuits," McGraw-Hill, 2002, ISBN 978-0-07-232084-8

[16] Gray, Hurst, Lewis and Meyer, Wiley & Sons. "Analysis and Design of Analog Integrated Circuits, " New York.

[17] Wang, Jun. Xu, Shuming. Korec, Jacek. "Power MOSFET with integrated gate resistor and diode-connected MOSFET, " U.S. Patent 5939863, August 1999. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-bool.html&r=3&f=G&l=50&co1=AND&d=PTXT&s1=%22power+mosfet%22.TI.&OS=TTL/%22power+mosfet%22&RS=TTL/%22power+mosfet%22

8.0 Appendix A – Testing Results

Table 2 below shows the testing result of the electronic load circuitry without current sense circuitry.

Table 2: Electronic load data

Given	Experimental Data					Notes	Theoretical Data		% error
V(DAC)	Vr	Vg	Vgs	Iout (0A-2A)	P (12V)		Ir=Imosfet	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	* voltage start jumping around (due to power supply used)	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

Table 3 below shows the testing result of the electronic load circuitry with current sense circuitry.

Table 3: Electronic load with current sense circuitry data

V(DAC)	Iout (0A-2A) *read from power supply	Vout *predicted through 1:2 ratio on datasheet	Vout *measure from multimeter
0.00	0.0000	0.0000	0.0000
0.10	0.0650	0.0325	0.0343
0.20	0.1310	0.0655	0.0677
0.30	0.1980	0.0990	0.1009
0.40	0.2640	0.1320	0.1340
0.50	0.3310	0.1655	0.1670
0.60	0.3970	0.1985	0.2000
0.70	0.4640	0.2320	0.2320
0.80	0.5300	0.2650	0.2650
0.90	0.5960	0.2980	0.2980
1.00	0.6620	0.3310	0.3300
1.10	0.7300	0.3650	0.3640
1.20	0.7960	0.3980	0.3970
1.30	0.8620	0.4310	0.4300
1.40	0.9280	0.4640	0.4630
1.50	0.9950	0.4975	0.4960
1.60	1.0610	0.5305	0.5280
1.70	1.1270	0.5635	0.5610
1.80	1.1940	0.5970	0.5940
1.90	1.2600	0.6300	0.6270
2.00	1.3260	0.6630	0.6600
2.10	1.3940	0.6970	0.6930
2.20	1.4600	0.7300	0.7260
2.30	1.5260	0.7630	0.7590
2.40	1.5900	0.7950	0.7920
2.50	1.6560	0.8280	0.8250
2.60	1.7250	0.8625	0.8580
2.70	1.7920	0.8960	0.8900
2.80	1.8580	0.9290	0.9200
2.90	1.9250	0.9625	0.9520
3.00	1.9910	0.9955	0.9840

9.0 Appendix B – Arduino Code

Instr_Sim_Driver.h

```
// ------------------------- MBARI Instrument Simulator Project ---------------------------- */
// Instr_Sim_Driver.h
// Author: Richard Bae, Matt Michell, Amy Chen
//
// -------------------------------------------------------------------------------------------
#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 Instrument Simulator Project ---------------------------- */
// Instr_Sim_Driver.ino
//  Author: Richard Bae, Matt Michell, Amy Chen
//
// ------------------------------------------------------------------------------------------
#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 Sensor Hyperterminal Initialization
  Serial1.begin(9600);
  if(Serial1) {
    Serial1.println("[plugged in instrument]");
    Serial1.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 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 command: ");
   Ph_Serial_Read();
   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);
  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);
  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 * 2.0, 3);
   Serial.println(" A");
   Serial.print("   Power: ");
   Serial.print(temp->loadV * 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();

  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() {

  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") {
    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 command");
  }

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

```
  Serial1.println();

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

10.0 Appendix C – ABET Senior Project Analysis

Project Title: Oceanographic Instrument Simulator

Student's Name:	Amy Chen Byungjin Matt Mitchell	Advisor's Name:	Bridget Benson Vladimir Prodanov

1. Summary of Functional Requirements

The *Oceanographic Instrument Simulator* simulates the outputs of the Seabird SBE18s pH Sensor, Seabird SBE52 CTD Sensor, Seabird SBE43 DO Sensor and Teledyne Workhorse ADCP. The Arduino Due microcontroller will control the load simulation of various underwater sensors by using an external DAC to turn each port on and off. The signals to port 1-3 will go through a variable load with a current limit of 2A. The signal to port 4 will go through a variable load with a current limit of 6A. The power consumption of the 4 ports will be constantly monitored by the Arduino Due microconotroller. If the power consumption exceeds a predetermined amount, all 4 ports will be shut off.

2. Primary Constraints

There is a current limitation of 2A on ports 1-3 and a limitation of 6A on port 4. The limitations are a constraint of the sensor node, which the *Oceanographic Instrument Simulator* will be plugged into. Input power is also restricted to 115 VAC. This will allow the user to plug the device into the wall outlets.

3. Economic

The list below breaks down the manufacturing and operating costs of the project.

- Financial Capital
  - High Power MOSFET (4) - $1.19 per MOSFET → $4.76 total
  - Power Resistor (4) - $1.56 per resistor → $6.24 total
  - DAC (4) - $1.56 per resistor → $6.24 total
  - Arduino Due (1) - $49.9
  - Total Financial Capital - $67.14
- Manufactured or Real Capital
  - Solder stations - $2,000
  - Circuit Testing Machines - $10,000
  - Total Real Capital - $12,000
- Natural Capital
  - Electricity Bill - $100 per month

Table 4: Cost estimate per unit

Item	Description	Quantity	Individual Cost	Total Cost
High Power MOSFET	Drain to source voltage of 60V. Current of 3A.	4	$1.19	$4.76
Adruino Due Microcontroller	The Due has 4 UARTs (hardware serial ports) and 12 analog inputs.	1	$49.99	$49.9
Resistor	Resistor with 1% tolerance of values of 1-2 Ω.	4	$1.56	$6.24
DAC	Convert bits to analog signal for the high voltage op-amp.	4	$1.56	$6.24
SD Shield	Holds SD card. Medium of communication between SD card and microcontroller	1	$14.95	$14.95
Max3323 Serial Transceiver	Serial communication between microcontroller and PC	4	$2.99	$11.95
USB Serial Converter	Converts data from USB to Serial	4	$15.00	$60.00
			Total Parts Cost	$154.04
Labor	$20/hr * 1 laborer (300 hrs)	1	$20	$6000
			Total Cost	$7634.09

During the beginning of the development and manufacturing phase, purchasing prototype parts, design parts, and additional unforeseen parts will result with the highest cost.

The experiment requires labor to solder components. Testing against the overall design is also necessary.

The original estimated cost of component parts is approximately $100. This is because the prototype will be developed on a breadboard, not a fabricated PCB.

Most of the cost will be sponsored by Monterey Bay Aquarium Research Institute and the remaining cost will be paid by the school's product budget.

The estimated development time is 8 months, starting in March 2014, and ending December 2014. However, one of the group members, Amy Chen, might continue this project as her thesis in graduate school.

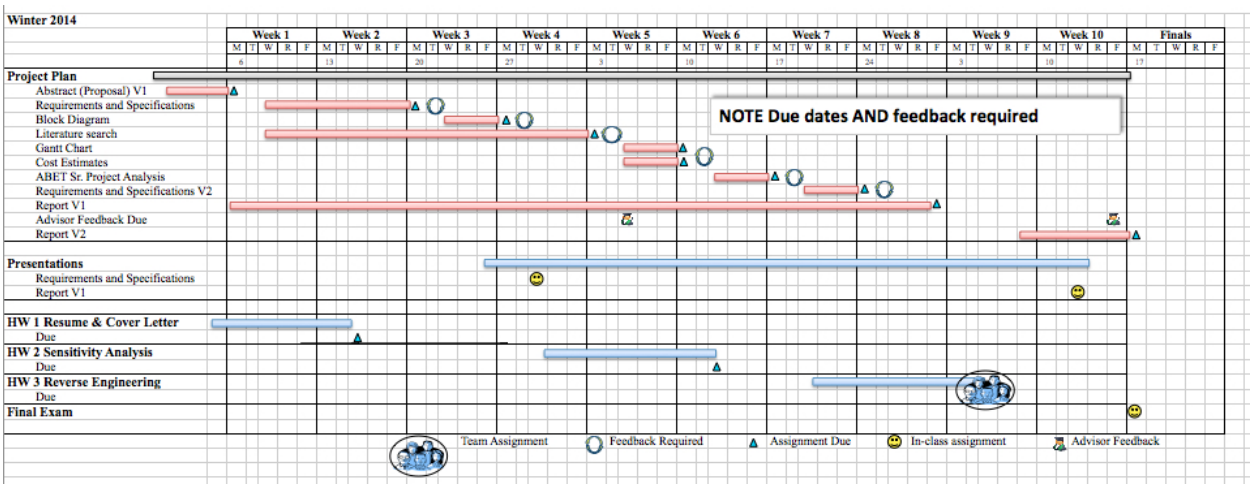


Figure 23: Winter 2014 Gantt Chart

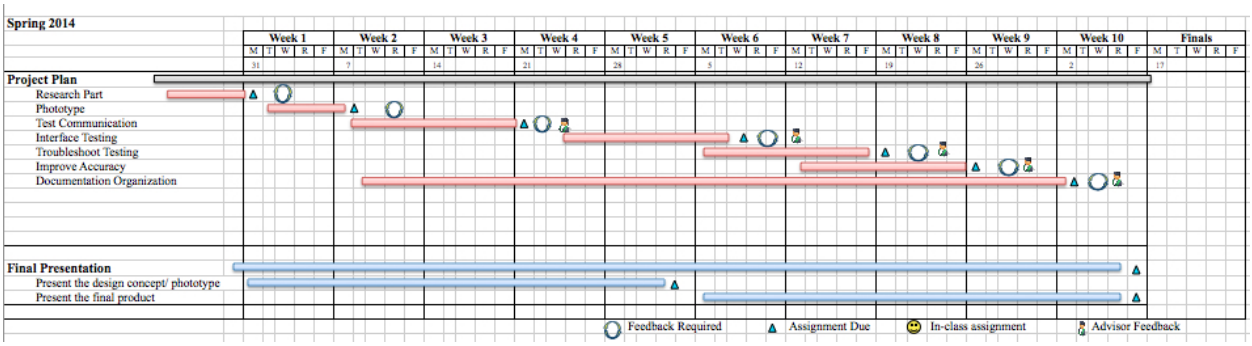


Figure 24: Spring 2014 Gantt Chart

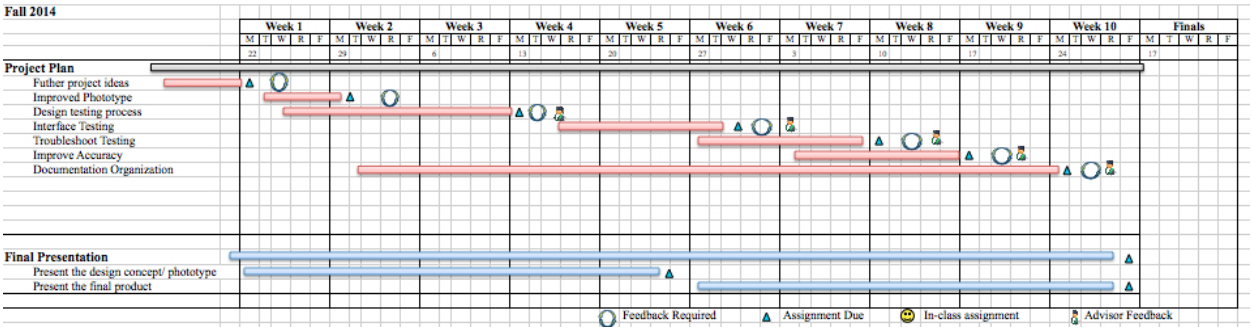


Figure 25: Fall 2014 Gantt Chart

4. Environmental

Electronics devices, such as PCBs, eventually become electronic waste and can damage ecosystems. The *Oceanographic Instrument Simulator* reduces the use of actual physical sensors, which has positive effects on the environment by reducing the use of electronic devices.

5. Manufacturability

A few challenges in manufacturing this product include:

- The distribution of both hardware and software since the software for checking current limitation status is also produced.
- Implementing DAC and ADC conversions
- Implementing a solder system for the PCB and peripheral components.

6. Sustainability

The PCB within the *Oceanographic Instrument Simulator* needs routine maintenance to keep the system from rusting and losing conductivity. Since the system is made of several electrical components, the connector to the terminal must be kept tidy to keep the connections intact and free from folding and stretching. The microcontroller may also require maintenance because there must be reliable communication between each port.

There are many upgrades that could potentially improve the design of the project. One key upgrade would be to focus more on the actual hardware circuit design rather than relying on microcontroller. When the circuit design is developed to restrict the limitation of current and voltages, it increases the stability of entire system.

7. Ethical

IEEE code of ethics states, “to improve the understanding of technology; its appropriate application, and potential consequences.”

The marine scientist will be able to benefit from the *Oceanographic Instrument Simulator*. By providing data and simulate the sensor node speeds up the simulating process for the scientist. The *Oceanographic Instrument Simulator* improves upon the provided sensor and makes it plausible to include multiple sensors onto one circuit board. Interfacing between multiple sensors, as opposed to a system that supports only 1 sensor, also reduces the resulting pollution of the entire system.

8. Development

While designing the circuitry, there are different power electronic components required to fit the current limit of 2A for port 1-3 and 6A for port 4. To simplify the circuit, the electronic load design will be the same for all 4 ports, but with the different configuration. For example, the electronic load design is used for 2A load, by cascading three 2A loads together; we will be able to obtain an electronic load design for 6A load.