MICROPROCESSOR MANAGEMENT OF ENERGY HARVESTING BUCK-BOOST CONVERTERS

By

Timothy O'Sullivan

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2014

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

Figures

Abstract

The project calls for the microprocessor management of a buck-boost converter with intermediate super capacitor storage. A buck-boost converter was designed with a 12V to 48V input with an adjustable current input limit and 12V to 48V output with adjustable current output limit. The driving specification is to produce a high current output power supply for the Liquid Robotics Wave Glider. The Wave Glider has a 12V, 0.5A power supply and the acoustic communications modem requires 24V to 32V at 1A. A microcontroller and firmware will be implemented that allows a user to manage a digital potentiometer, and analog to digital converter, and a digital input/out to control the output voltage of the design for the waveglider's internal devices. The complete design was not able to be accomplished, but individual components of the design were completed and tested.

I. Introduction

Motivation for this product stems from a current large-scale product currently manufactured by Liquid Robotics called the Wave Glider. [11] The Liquid Robotics Wave Glider website describes it as the first hybrid wave and solar propelled unmanned ocean robot. The website boasts that the Wave Glider's energy harvesting technology and its propulsion and energy systems, help customers such as the Monterey Bay Aquarium Research Institute, explore previously challenging regions of the ocean. The Wave Glider was first introduced in 2009 and has since distanced over 300,000 nautical miles while setting a record for the longest distance traveled by an autonomous vehicle. By taking advantage of the natural ocean wave and solar energy, the Wave Glider continuously transmits ocean data without the need for manpower or carbon emissions while tackling world-scale challenges such as hurricanes and tsunamis. [13] Multiple internal devices require power for the wave glider to operate and to transmit data collected. Each device does not need to constantly operate or consume power from the energy harvesting technology therefore a microprocessor management system regulates power to each device.

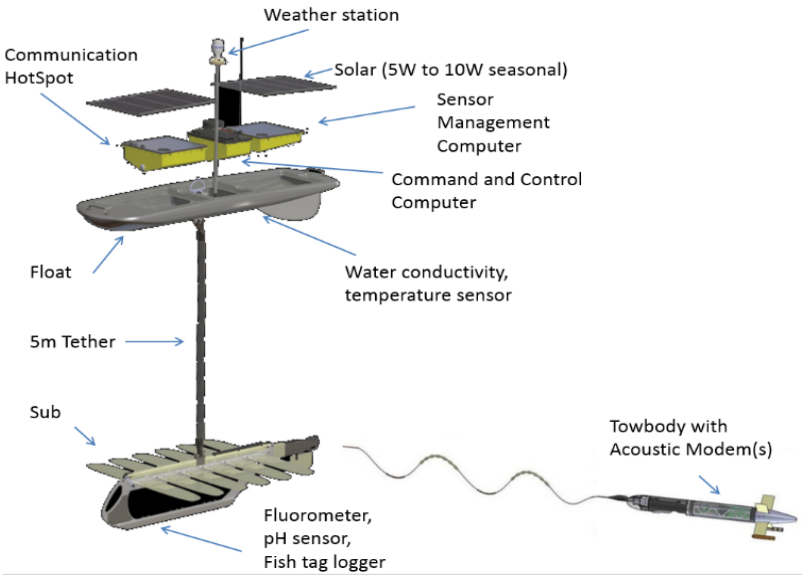


Figure 1: Liquid Robotics Wave Glider

Wave motion maximizes at the surface of water and decreases quickly as depth increases. When the wave rises, the float lifts, the sub rises and the wings on the sub are pushed down. This serves as an up and forward motion, pulling the float off the wave. The sub then drops, the wings flip and the sub moves down and forward. This process is then repeated multiple times. [12]

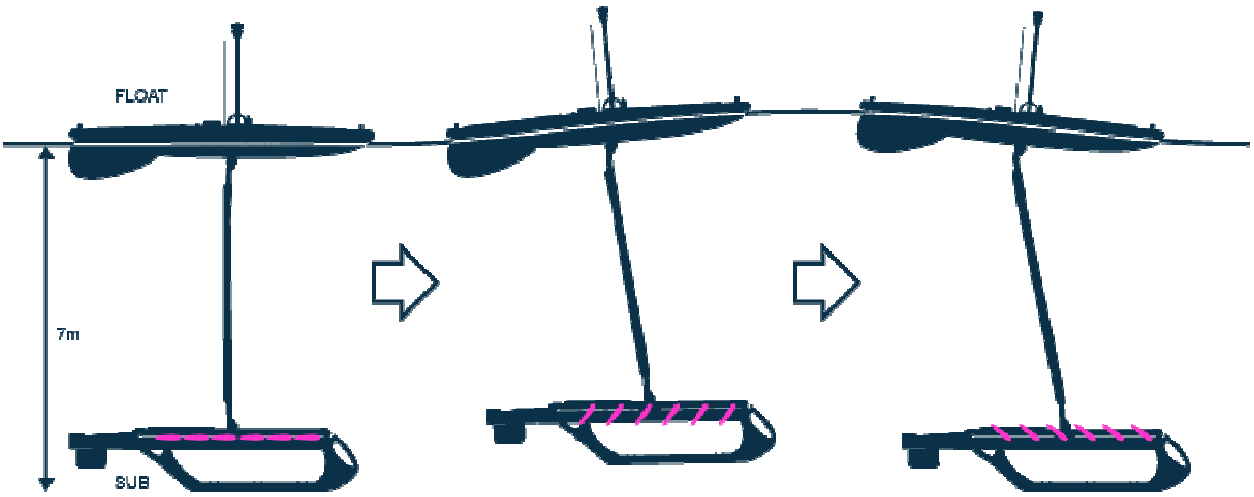


Figure 2: Wave motion to forward propulsion.

Customer Needs Assessment

The Liquid Robotics Wave Glider website describes it as the first hybrid wave and solar propelled unmanned ocean robot. The website boasts that the Wave Glider's energy harvesting technology and its propulsion and energy systems, help customers such as the Monterey Bay Aquarium Research Institute, explore regions of the ocean that were previously too challenging to explore. [11] The energy harvested by the super capacitor needs a management system to regulate the voltage and current outputted to other devices in the wave glider. The management system will regulate the operation of a digital potentiometer, analog to digital converter, and a digital input/output of a system designed to control output voltages to internal devices of the waveglider. These include the sensor management computer, acoustic modem, command and control computer and the communications hotspot. The control will be done based on a user interface which contains commands that regulate the operation of the three devices within the designed system.

II. Marketing Requirements, and Engineering Specifications

Requirements and Specifications

Customer needs and constraints of the overall system determine the marketing requirements and engineering specifications. The design has to fit inside of the Wave Glider while managing the electrical power supplied to each of the system's individual devices. The design should not require more voltage to operate than the output from the Wave Glider's power supply to maximize efficiency. [11] The project calls for the microprocessor management of a buck-boost converter with intermediate super capacitor storage. Buck-boost converters were designed with a 12V to 48V input with an adjustable current input limit and 12V to 48V output with adjustable current output limit. The driving specification is to produce a high current output power supply for the Liquid Robotics Wave Glider. The Wave Glider has a 12V, 0.5A power supply and the acoustic communications modem requires 24V to 32V at 1A. A microcontroller and firmware manages the super capacitor charge level to regulate the acoustic modem, the sensor management computer, the control and command computer and the communications hotspot.

TABLE I

MICROPROCESSOR MANAGEMENT OF AN ENERGY HARVESTING BUCK BOOST CONVERTER

Marketing Requirements	Engineering Specifications	Justification
1, 2, 5	1. Design must not draw more than the maximum allowable current from the Wave Glider sub power, or 12 V at 0.5 A	The management system should not consume power directly from the energy harvesting device.
5	2. Design a serial port driver to provide user control and status monitoring.	The user must be able to control the devices and be able to receive status of system.
5, 2	3. Must transmit a menu through the serial port to a computer running a serial terminal program.	The serial terminal program provides the user interface for control and receiving status.
1, 3, 4	4. Must output differing voltages ranging from 12V to 48V (1A to 4A).	Differing devices in the Wave Glider require varying voltages to operate correctly.
5, 3	5. Must design the I2C initialization for the microprocessor and I2C driver for the internal slave devices.	The internal devices must operate correctly based on the commands received from the user menu.
1, 5	6. UART must transmit at 9600 baud rate.	The serial port menu must respond no less than 100 microseconds.
Marketing Requirements 1. Low Power Consumption. 2. Must operate for years at sea. 3. Regulate Electrical Power. 4. Cost efficient. 5. Must provide user control.		

IV. Design

Level 0 Block Diagram

The level 0 block diagram in shown in Figure 1 below describes the high-level inputs and outputs of the design.

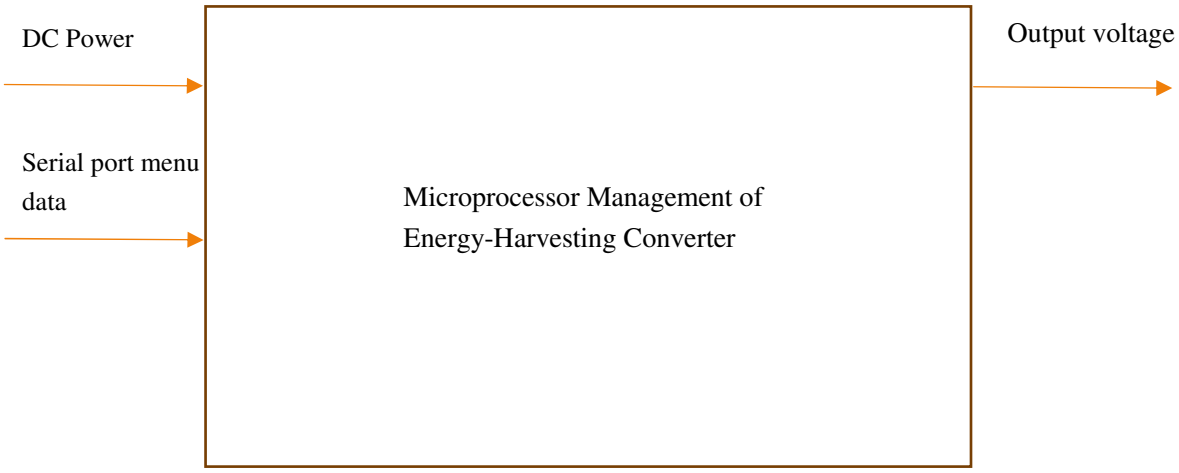


Figure 3: Level 0 Microprocessor Management Functionality

TABLE II

MICROPROCESSOR MANAGEMENT OF AN ENERGY HARVESTING BUCK BOOST CONVERTER

Module	Microprocessor Management of an Energy-Harvesting Buck-Boost Converter
Inputs	- DC Power (12V) - UART Menu data
Outputs	- Varying output voltage: 48V, 1A 24V, 2A 12V, 4A
Functionality	Reads Serial Port (UART) data selected from the menu and inputs the DC Power (12V). [7] The UART information that the system receives will determine the varying output voltage and drive different external devices. [3]

Level 1 Block Diagram

A MSP430F5438A microcontroller, the hardware selected and provided by MBARI, will be utilized to manage the I2C information. The level 1 block diagram shown in figure 4 below shows the three slave devices that the MSP430F microprocessor will be managing. Table III shows the inputs and outputs of the MSP430F microprocessor. The design calls for the control and status of the three slave devices mentioned in Tables IV, V, and VI. Once the MSP430F receives data from the menu, the code developed chooses the correct slave device to control and performs the necessary function that was specified from the menu data. For the digital input/output the inputs, ShortCkt and Charge are controlled and monitored while DigPotReset is only monitored. The design for the analog to digital converter controls and monitors V_{IN} registers 1 through 4 for IVINMON, V_{OUT}/20, ISMON, and PVIN/20. The digital potentiometer's registers W1-W4 for OVLO, Current_Ctrl, not connected, and Vout_Ctrl are both controlled and monitored.

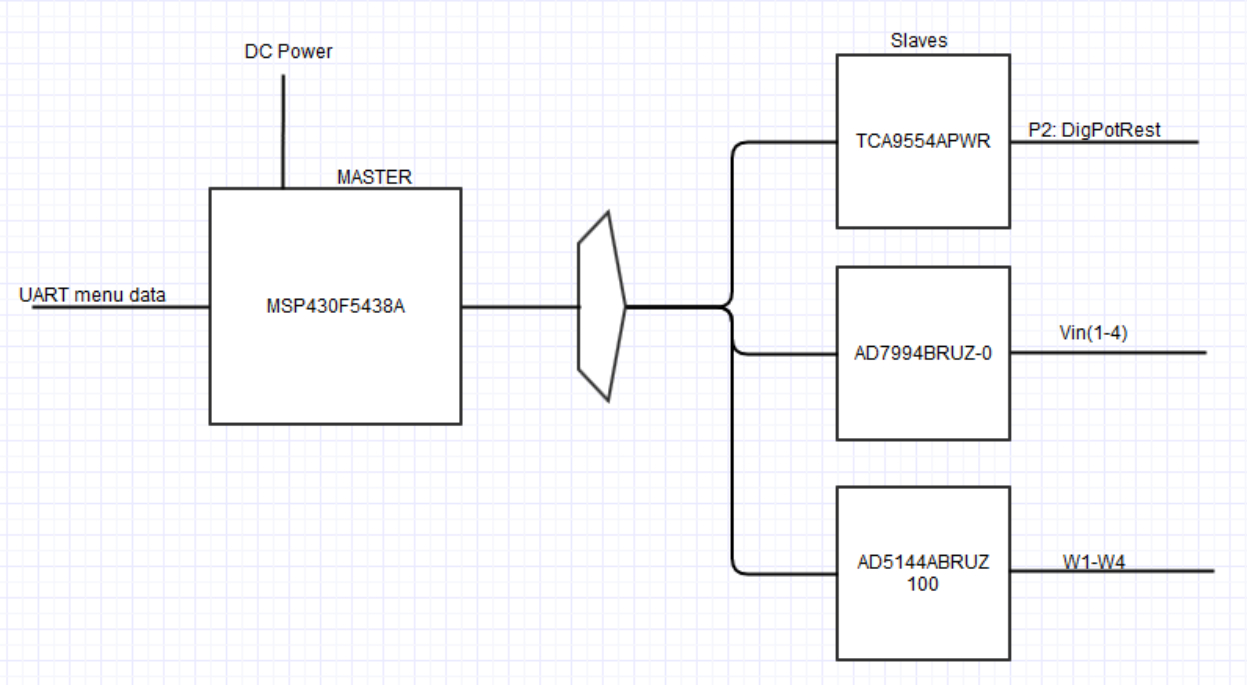


Figure 4: Level 1 Microprocessor Management Functionality

TABLE III

MSP430F MICROCONTROLLER

Module	MSP430F Microcontroller
Inputs	- DC Power - UART menu data
Outputs	- I2C slave device function
Functionality	Reads UART menu data input and inputs the DC Power. [7] The MSP430F microprocessor [3] interprets incoming data and determines which slave device to control. [8]

TABLE IV

DIGITAL INPUT/OUTPUT

Module	AD5144ABRUZ100
Inputs	- DC Power - P0: ShortCkt - P1: Charge
Outputs	- P2: DigPotReset
Functionality	The digital potentiometer receives I2C data from the MSP430F and performs necessary protocol based on user input from serial port. [4]

TABLE V

ANALOG TO DIGITAL CONVERTER

Module	AD7994BRUZ-0
Inputs	- DC Power - V_{IN1}: IVINMON - V_{IN2}: Vout/20 - V_{IN3}: ISMON - V_{IN4}: PVIN/20
Functionality	The analog to digital converter receives I2C data from the MSP430F and performs necessary protocol based on user input from serial port. [14]

TABLE VI

DIGITAL POTENTIOMETER

Module	TCA9554APWR
Inputs	- DC Power
Registers	- W1: OVLO - W2: Current_Ctrl - W3: not connected - W4: Vout_Ctrl
Functionality	The digital input/output receives I2C data from the MSP430F and performs necessary protocol based on user input from serial port. [15]

MSP430F5438A Microprocessor

Typical applications for the MSP430F5438A include analog and digital sensor systems, digital motor control, remote controls, thermostats, digital timers, and hand-held meters. The MS4P430F has a microcontroller configuration with three 16-bit timers, a high performance 12-bit analog-to-digital converter, up to four universal serial communication interfaces (USCIs), a hardware multiplier, DMA, a real-time clock module with alarm capabilities, and up to 87 I/O pins. [3]

Buck-Boost Slave Devices

The MSP430F5438A uses USCI_2 i2c ports to send data across a shared bus to communicate and control three slave devices on the buck boost circuit board. The devices are listed below:

1. Digital Potentiometer (Digital to Analog Converter)
   Part Number: AD5144ABRUZ100

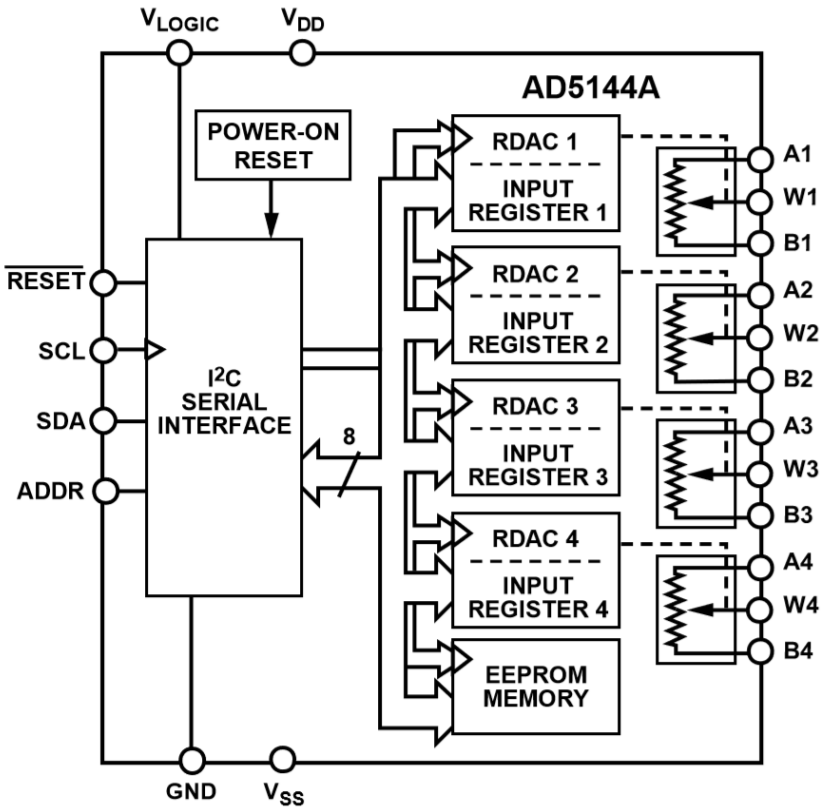


Figure 5: Functional Block Diagram of the AD5144A Digital Potentiometer [4]

Common applications of the AD5144A Digital Potentiometer include portable electronics adjustment level, LCD panel brightness and contrast controls, programmable filters, delays, time constants, and programmable power supplies. The AD5144A potentiometer provides a nonvolatile solution for 128-/256-position adjustment applications, offering guaranteed low resistor tolerance errors of ±8% and up to ±6 mA current density in the Ax, Bx, and Wx pins. [4]

2. Analog to Digital Converter
   Part Number: AD7994BRUZ-0

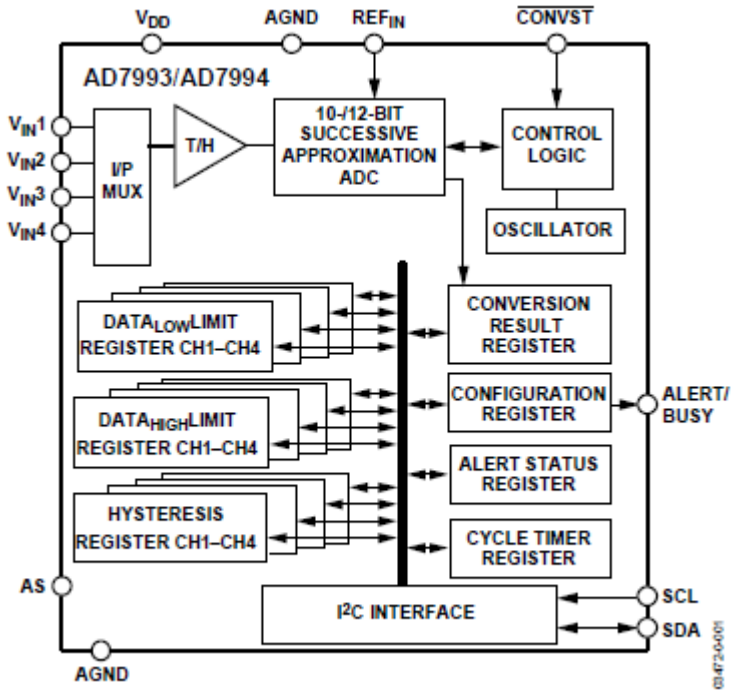


Figure 6: Functional Block Diagram of AD7994 Analog to Digital Converter [14]

Figure 6 represents the functional block diagram of the AD7794BRUZ-0 analog to digital converter. The AD7994 is a 4-channel 10 and 12 bit, low power, successive approximation ADC with an I2C compatible interface. The device operates from 2.7 V to 5.5 V Vcc and has a 2 microsecond conversion time, can handle input frequencies up to 11 MHz. The AD7994 requires an external reference that should be applied to the REF pin in the range of 1.2V to Vcc. This allows the widest dynamic input range to the ADC. [14]

3. Digital Input/Output
   Part Number: TCA9554APWR

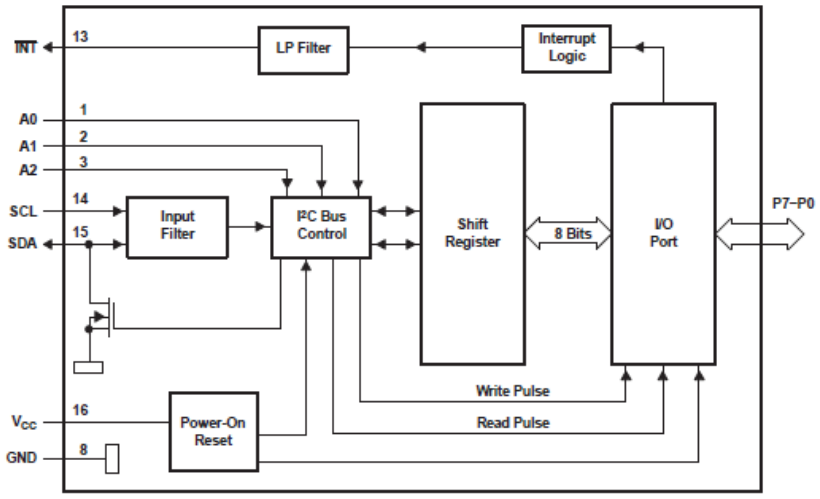


Figure 7: Functional Block Diagram of the TCA9554A Digital Input/Output. [15]

The TCA9554APWR Digital Input/Output is an 8-bit I/O expander for the two line bidirectional bus designed for 1.65 to 5.5 V Vcc operation. At power on, the I /Os are configured as inputs with a weak pullup to Vcc, however the master can set the I /Os as either by writing to the I/O configuration bits. The polarity of the input port register can be inverted with the polarity inversion register. All registers can be read by the system master. [15]

Serial Port Menu

The design calls for a serial port menu to provide user control and status monitoring. The goal was to transmit a menu through the serial port to a PC running a serial terminal program and then receive the command inputted by the user to control specific I2C devices. The string below was used to transmit to the serial port and the full code can be found in the code appendix. Figure 8 displays a screen capture of the serial port menu that was transmitted and analyzed using Tera Term.

```
4 const char string[200] = { "Welcome to the Buck-Boost Converter\r\n"
5                            "Enter a command to control the following: \r\n"
6                            "Digital Input/Output - D \r\n"
7                            "Analog to Digital Converter - A \r\n"
8                            "Digital Potentiometer - P \r\n"
```

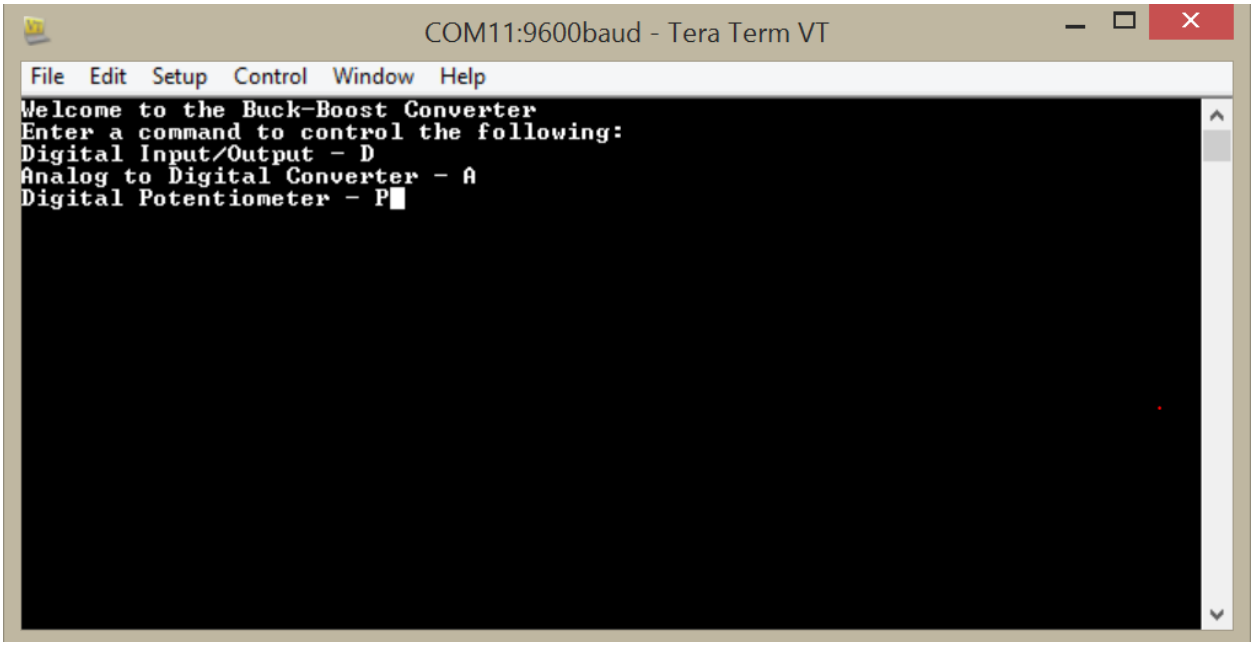
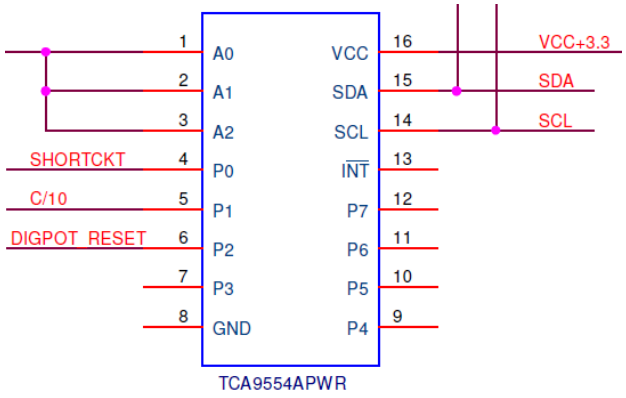


Figure 8: Screen capture of the menu transmitted to Tera Term serial port emulator.

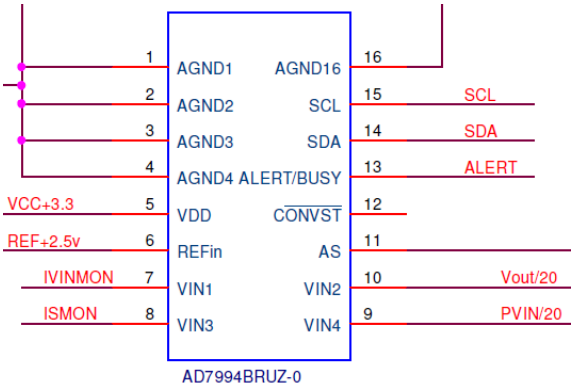
I2C Protocol

The design also calls for user control of the three slave devices. The goal for this project was to write the I2C initialization and setup for the MSP430F5438A microcontroller and I2C drivers for the slave devices on the buck-boost board. These drivers included:

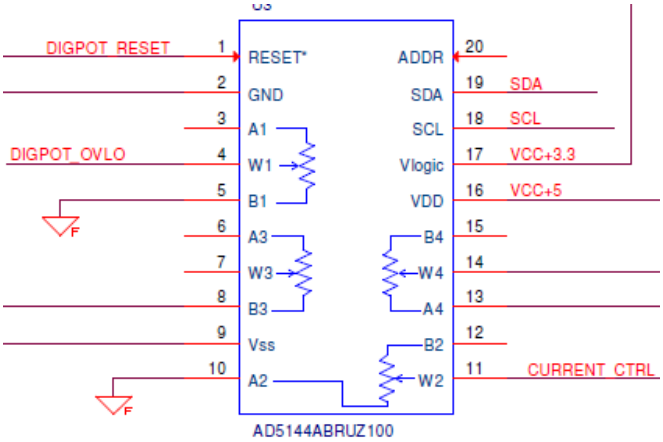
1. **TCA9554APWR Digital I/O**
   7-bit Slave Address: 010 0000
   - P0 Input: ShortCkt
   - P1 Input: Charge
   - P2 Output: DigPotReset



2. **AD7994BRUZ-0 Analog to Digital Converter**
   7-bit Slave Address: 010 0010
   - V_{IN1}: IVINMON
   - V_{IN2}: Vout/20
   - V_{IN3}: ISMON
   - V_{IN4}: PVin/20



3. **AD5144ABRUZ100 digital Potentiometer (Digital to Analog Converter)**
   7-bit Slave Address: 010 1010
   - W1: DIGPOT_OVLO
   - W2: Current_Ctrl
   - W3: not connected
   - W4: Vout_Ctrl



V. Testing

To test the microprocessor management design, first the input power for the system's hardware was obtained by testing for the input voltage and input power using a multimeter. Then a sample code to blink an LED was written to understand the schematic and hardware of the microprocessor. Next another sample code was created to initialize and test the functionality of the UART serial port. After, an AD5144 Evaluation kit was used to understand how the I2C transactions operated for the digital potentiometer. The I2C operations were viewed using a beagle protocol analyzer and the software associated with it. Finally after obtaining and understanding how the protocol functioned for the digital potentiometer, the code was recreated and tested in code composer studio using the MSP430F microprocessor.

Input Power

To verify that the design did not draw more than 0.5 amps at an input voltage of 12 volts, the input power, voltage, and current was first tested. This was done to meet the requirement in which the design would not draw more than the maximum allowable current from the waveglider sub power. The recorded data can be seen in table VII below as well as the graphs of input voltage vs. input current and input voltage vs. input power in figures 9 and 10.

TABLE VII
INPUT POWER

Input Voltage	**Maximum Input Current**	**Input Power**
12 V	*158.2 mA*	*1.90 W*
16V	*114.9 mA*	*1.84 W*
20 V	*89.5 mA*	*1.79 W*
24 V	*77.9 mA*	*1.86 W*

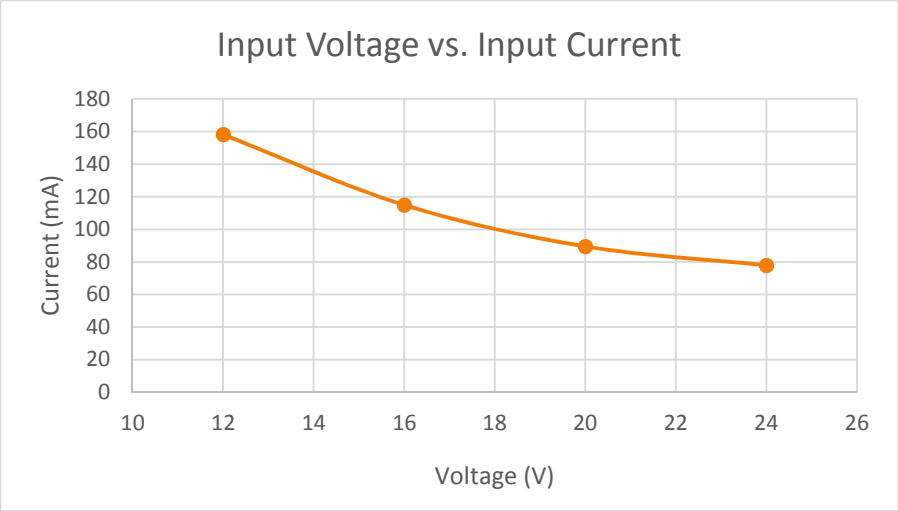


Figure 9: Input voltage vs. input current

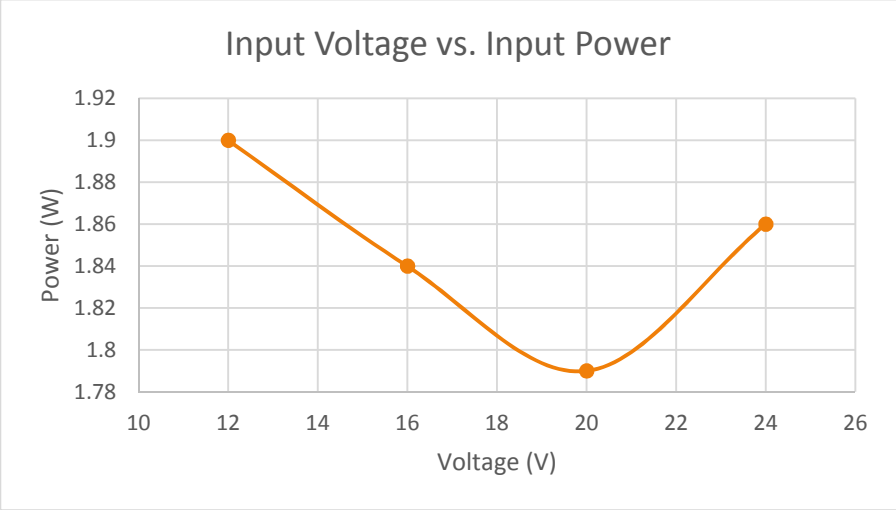


Figure 10: Input voltage vs. input power.

LED Toggle

In order to understand the schematic and the hardware, a sample code was written in code composer studio to blink LED D43 on the microprocessor circuit board. After initializing and setting port 3.0 as an output, the period at which D43 blinked was controlled using a timer A. Timer A's clock speed was set to SMCLK, or 8 MHZ. [3] Figure 9 below shows the connection of the microprocessor board at output port 3.0 to LED D43.

```
/*
 * Blink LED D43 Test
 */
#include <msp430.h>
#define LED_RED BIT0

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;        // Stop watchdog timer
    P3DIR |= LED_RED;                // Set P3.0 to the output direction
    P3OUT &= ~(LED_RED);             // Sets the D43 red LED off

    TA1CCTL0 = CCIE;
    TA1CCR0 = 50000;
    TA1CTL = TASSEL_2 + MC_2;        // Set the timer A to SMCLCK, Continuous

    __enable_interrupt();
    __bis_SR_register(LPM0 + GIE); // LPM0 with interrupts enabled
}

// Timer A0 interrupt service routine
#pragma vector=TIMER1_A0_VECTOR
__interrupt void TIMER1_A0_ISR (void)
{
P3OUT ^= (LED_RED);
TA1CCR0 += 50000;
}
```

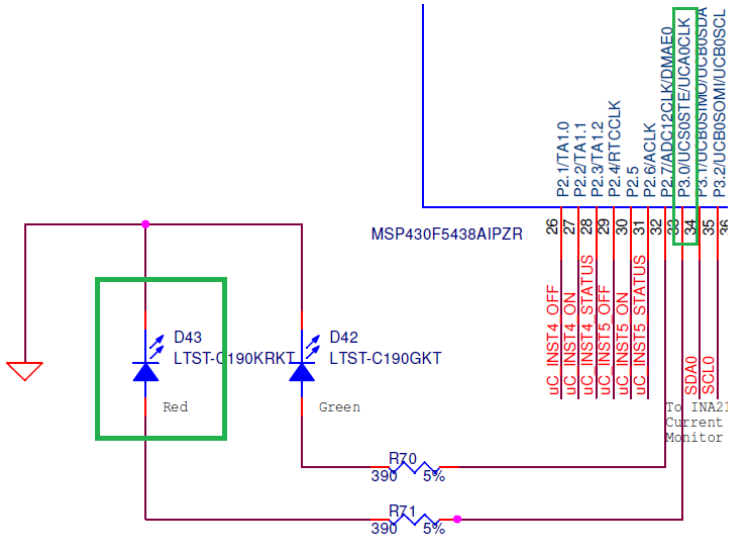


Figure 11: LED D43 and output port 3.0 on microprocessor board.

Serial Port UART

The design calls for a serial port driver to provide user control and status monitoring. A sample code was written in code composer studio to initialize and verify the functionality of the UART. A hexadecimal value of 55 was chosen to represent a square wave on the transmit buffer of the serial port. Using an oscilloscope, the wave form was obtained and showed 1 bit per 1/9600 seconds, or 9600 baud. Also, the serial port emulator Tera Term was used to observe the ASCII value, "U", of the hexadecimal value being transmitted. The code that was developed in code composer studio to test the serial port UART is shown below.

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P9SEL = 0x30;                             // P9.4,5 = USCI_A0 TXD/RXD
  UCA2CTL1 |= UCSWRST;                      // **Put state machine in reset**
  UCA2CTL1 |= UCSSEL_2;                     // SMCLK
  UCA2BR0 = 6;                              // 1MHz 9600 (see User's Guide)
  UCA2BR1 = 0;                              // 1MHz 9600
  UCA2MCTL = UCBRS_0 + UCBRF_13 + UCOS16;   // Modln UCBRSx=0, UCBRFx=0,
                                            // over sampling
  UCA2CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  UCA2IE |= UCTXIE;                         // Enable USCI_A2 RX interrupt

  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0, interrupts enabled
  __no_operation();                         // For debugger
}

#pragma vector=USCI_A2_VECTOR
__interrupt void USCI_A2_ISR(void)
{
  switch(__even_in_range(UCA2IV,4))
  {
  case 0:break;                             // Vector 0 - no interrupt
  case 2:                                   // Vector 2 - RXIFG
    while (!(UCA2IFG&UCTXIFG));             // USCI_A0 TX buffer ready?
    UCA2TXBUF = 0x55;                       // TX -> "U" to serial port
    break;
  case 4:break;                             // Vector 4 - TXIFG
  default: break;
  }
}
```

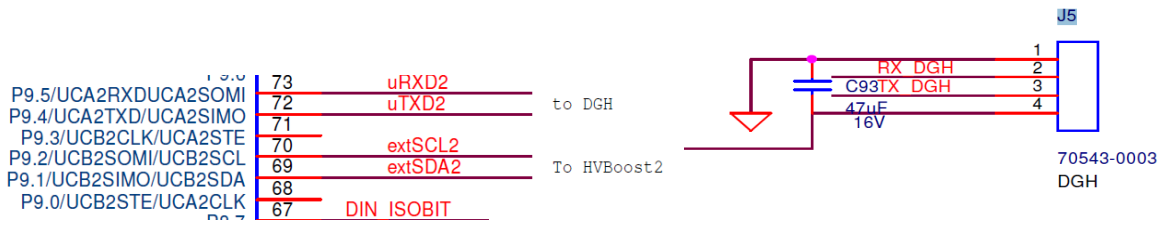


Figure 12: Ports 9.4,5 for receiving and transmitting to the UART. J5 connects the microprocessor board to a female DB-9 serial port.

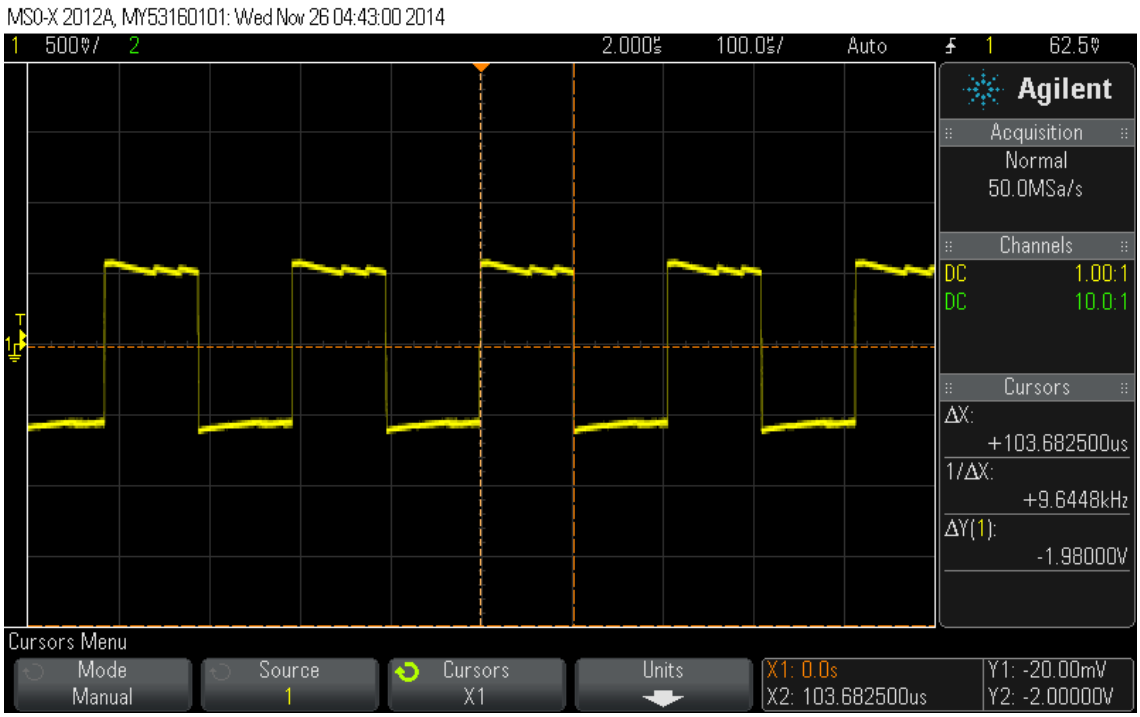


Figure 13: Oscilloscope capture displaying the UART Test at 9600 baud rate.

Figure 13 displays an oscilloscope capture of the UART transmitting the test code. The image shows a baud rate of 9600 which meets the requirements. The hardware provided originally by MBARI had an error with the serial port. The ground, transmit, and receive leads were soldered to the wrong output port from the microcontroller board.

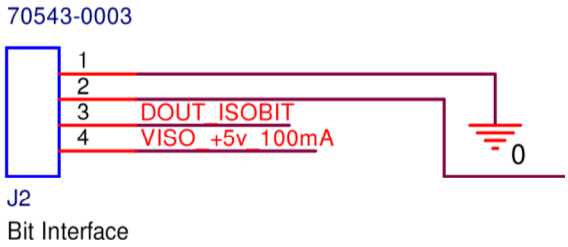


Figure 14: J2 output port from microcontroller board.

Figure 14 displays the output port J2 which is where the ground, transmit, and receive leads were soldered to on the board. This port outputs from a LT2882 Dual Isolated µModule Transceiver which does not have UART capabilities. The correct receive and transmit location can be seen in figure 12 which shows uRXD2 and uTXD2 connected to RX_DGH and TX_DGH at output port J5 on the microprocessor board.

AD5144A Evaluation

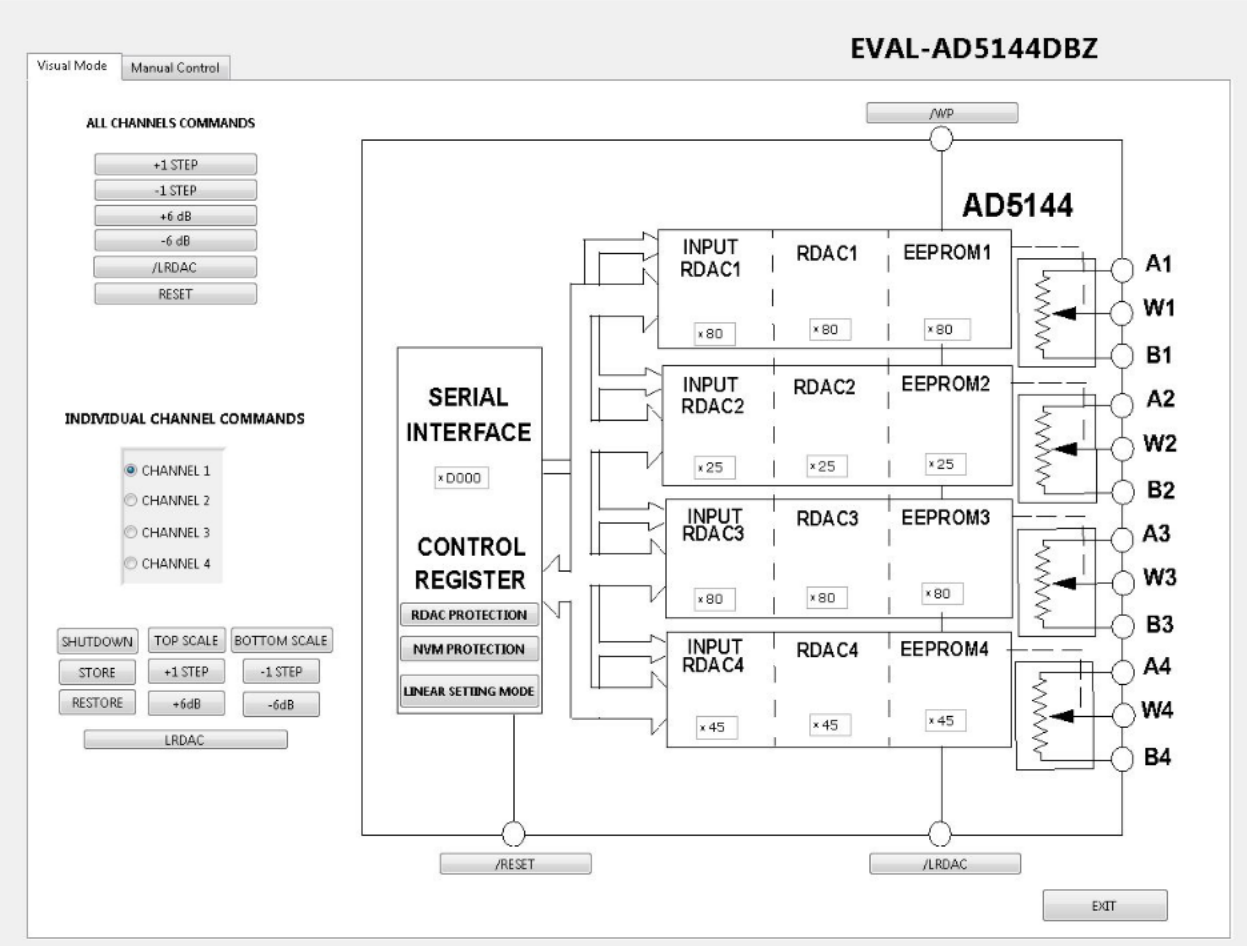


Figure 15: User Interface for the AD5144 Evaluation Software. [5]

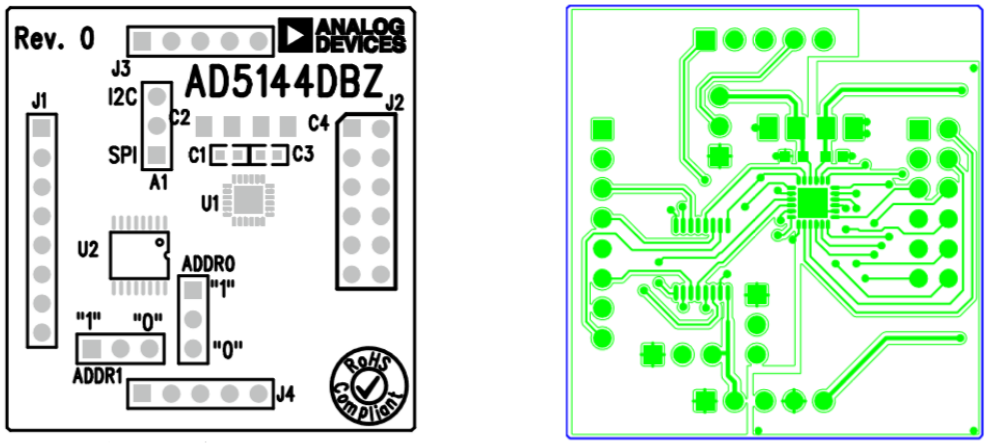


Figure 16: AD5144 Component side view of the daughter board and component placement drawing of daughter board. [5]

Figure 15 above displays the user interface for the AD5144 evaluation software. To understand the I2C transactions for this device a +1 step for all channels was performed using the evaluation software and board. This +1 step represents the I2C protocol, or the correct successive order of bytes transmitted and received, for the driver of the digital potentiometer that was to be designed. The software performed the necessary read and write transaction over the I2C bus to control the AD5144 digital potentiometer that were outlined in the datasheet. [4] Figure 16 represents the AD5144 daughter board on the evaluation mother board that was using to test the software and understand the I2C protocol for the device.

Beagle Protocol Analyzer

A total phase beagle I2c protocol analyzer was used in conjunction with the AD5144 evaluation hardware and software. The anlayzer was attached the the clock and data lines on the daughter board shown in figure 14.

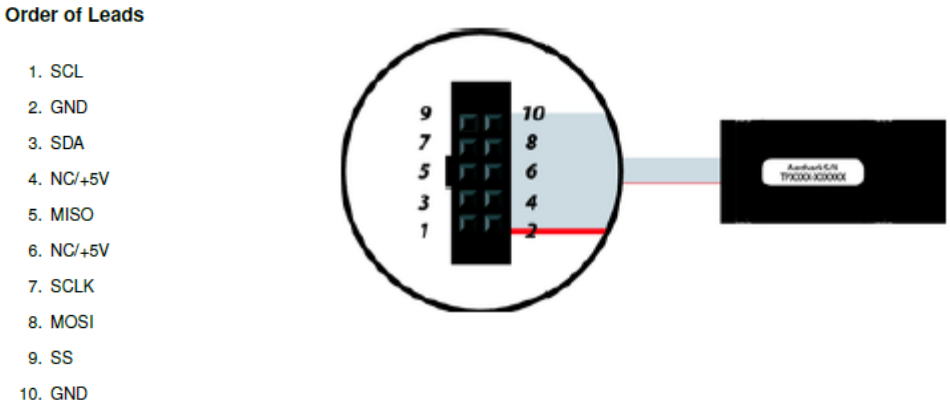


Figure 17: Order of leads for the beagle i2c analyzer.

Figure 17 above displays the order of leads for the beagle i2c analyzer that was attached to the bus of the AD5144 Evaluation board during the I2C testing phase. Data center, the software associated with the beagle analyzer was then used to view the read and write transactions from the evaluation softare described in the section above.

I2C Byte Testing

To test the I2C protocol, an I2C Beagle Analyzer in conjunction with an AD5144 Evaluation board was used. The AD5144 Evaluation software was used along with the software for the Beagle Analyzer to understand the read and write transactions involved in driving the digital potentiometer. Leads were soldered to the microcontroller board for SDA and SCL and the Evaluation board was powered using a 5V supply. Using Code composer studio and the MSP430F the protocol analyzed by the Beagle was recreated to drive the AD5144ABRUZ100 digital potentiometer on the buck-boost board. The I2C protocol was not able to be recreated using the microprocessor and buck-boost board. The I2C initialization sequence was determined to be correct from observing the voltage level of the clock and data line once the program from code composer studio was loaded onto the microprocessor. Both the clock and data lines were correctly initialized to a logic high level which corresponded to 3.3V set by the microprocessor. Using an oscilloscope, the clock and data lines were again observed and the hardware never performed a start condition, or transferred any bytes. A start condition is performed when the data line is pulled from a logic high level to a logic low level while the clock line remains at a logic high level. The byte that was on the I2C buffer was also transmitted to the serial port to test the ASCII values that each byte should transmit. It was found that the first byte was loaded into the buffer to be transmitted, but was never interpreted by the slave device due to the lack of an I2C start condition. Sample I2C codes provided by Texas Instruments for the MSP430F5438A were also tested and the same problem still existed. It can be concluded that the problem is likely a hardware manufacturing error. The sample array of bytes to be transmitted can be seen below.

```
26 const unsigned char TxData[] =              // Table of data to transmit for +1 Step on channel 1.
27 {
28   0x40,
29   0x01,
30   0x34,
31   0x03
32 };
```

Testing Conclusion

After testing the input power the design successfully drew less than 0.5 amps at an input of 12V. A menu, with a baud rate of 9600, was transmitted through the serial port to a computer running a serial terminal program for user control. The I2C initialization was completed based on the correct high logic value of the clock and data lines, but because of the lack of a start condition and the problem with the I2C byte testing the drivers could not be completed. Because the I2C drivers could not be completed, varying output voltages from 12V to 48V could not be obtained.

VI. Conclusion and Recommendations

As stated in the I2C testing section, the MSP430F5438A did not perform a start condition with both example code from Texas Instruments' and the recreated code to manage the digital potentiometer. Because of this, the design was not successful and did not function properly. Although communicating with multiple devices is more efficient using I2C, proper functionality of the slave devices proved to be much more difficult than using devices with SPI communication capabilities. The use of a serial port and emulator demonstrated a better way to test byte transfers on the I2C bus over viewing the waveform on an oscilloscope. Recommendations include the use of a different microprocessor, whether it be a different model manufactured by Texas Instruments that has same applications or a microprocessor from a different manufacturer that share similar uses. For this project that was not possible because the hardware was already fabricated based on MBARI's preferences, therefore no easy microprocessor change could be made. If fiscally possible, the use of evaluation hardware and software is recommended along with a Beagle protocol analyzer. These products proved to be great tools in understanding the I2C protocol outlined in the datasheet for managing the AD5144 digital potentiometer.

VII. References

[1] R. Ford and C. Coulston, *Design for Electrical and Computer Engineers*, McGraw-Hill, 2007, p. 37

[2] *IEEE Std 1233, 1998 Edition*, p. 4 (10/36), DOI: 10.1109/IEEESTD.1998.88826

[3] Texas Instruments, "Mixed Signal Microcontroller," MSP430F5438A datasheet, SLAS655D, Jan. 2010 [Revised Aug. 2013]

[4] Analog Devices, "Nonvolatile Digital Potentiometer,"AD5144A Data Sheet, 2012 [Revision A]

[5] Analog Devices, "Evaluating the AD5144 Digital Potentiometer," Evaluation Board User Guide, UG-469, Nov. 2012 [Revision 0]

[6] A. Bermak; C. Shi; M.K. Law. "Integrated Solar Energy Harvesting and Storage." U.S. Patent 8 629 386, Jan. 14, 2014.

[7] J.M Irazabal; S. Blozis. "I2C Manual" AN10216-01, Phillips Semiconductors, March 24, 2003.

[8] Texas Instruments. "MSP430x5xx and MSP430x6xx Family User's Guide." SLAU208M. June 2008. [Rev. Feb 2013]

[9] Texas Instruments. "MSP430 Optimizing C/C++ Compiler v 4.2 User's Guide" SLAU132H. June 2013.

[10] T. Noergaard. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers.* Elsevier Science. December 2012.

[11] Liquid Robotics. "SHARC SV3." Liquid Robotics Federal Product Specifications. 2013.

[12] Liquid Robotics. "Converting Wave Motion to Forward Motion." Internet: http://liquidr.com/technology/wave-motion.html [March 18, 2014]

[13] Liquid Robotics. "The Wave Glider SV Series" Internet: http://liquidr.com/technology/wave-glider.html [March 18, 2014]

[14] Analog Devices, "4-Channel ADC," AD7994 Data Sheet, 2004 [Revision 0]

[15] Texas Instruments, "Low Voltage I/O Expander," TCA9554A Data Sheet, SCPS196A, Dec. 2010 [Revised Mar. 2012]

A. ABET Analysis of Senior Project Design

Project Title: Microprocessor Management of an Energy Harvesting Buck-Boost Converter

Student's Name: Timothy O'Sullivan **Student's Signature:**

Advisor's Name: Bridget Benson **Advisor's Signature:**

Date:

Summary of Functional Requirements

The project calls for the microprocessor management of a buck-boost converter with intermediate super capacitor storage. A buck-boost converter was designed with a 12V to 48V input with an adjustable current input limit and 12V to 48V output with adjustable current output limit. The driving specification is to produce a high current output power supply for the Liquid Robotics Wave Glider. The Wave Glider has a 12V, 0.5A power supply and the acoustic communications modem requires 24V to 32V at 1A. A microcontroller and firmware will be implemented that allows a user to manage a digital potentiometer, and analog to digital converter, and a digital input/out to control the output voltage of the design for the waveglider's internal devices.

Primary Constraints

The first and most important constraint of the project is that it must fit inside of the Liquid Robotics waveglider. The design must manage this product's energy harvesting system and regulate internal devices. It must also withstand environmental factors such as temperature and pressure. Since the design goes into the waveglider and the ocean, it must function in cold temperatures and submerged water pressure. [11] A microcontroller must be utilized in the design of this product and because MBARI supplied the necessary hardware, the microcontroller constraints require the use of the MSP430F5438A. [3]

Economic

Economic impacts result from the costs associated with the development and manufacturing of the project. By managing the energy harvesting technology, Liquid Robotics and MBARI stand to gain economically from this. Most costs come from the purchasing of hardware and the labor through designing, testing, and implementing.

This design directly impacts four economic categories: human capital, financial capital, marketing capital, and natural capitol. This product has the capability to aid in the collection of more reliable data in a shorter amount of time. Because the design will operate within the waveglider, human capital is relatively effortless. Financial capital rests in the stockholders of Liquid Robotics while marketing capital involves companies that have ocean orientated missions and a need for this product. Natural capital stems from the necessary resources used to develop the waveglider. The product is self-sustaining, obtaining energy through solar panels and therefore does not deplete natural resources to operate.

Gantt Chart

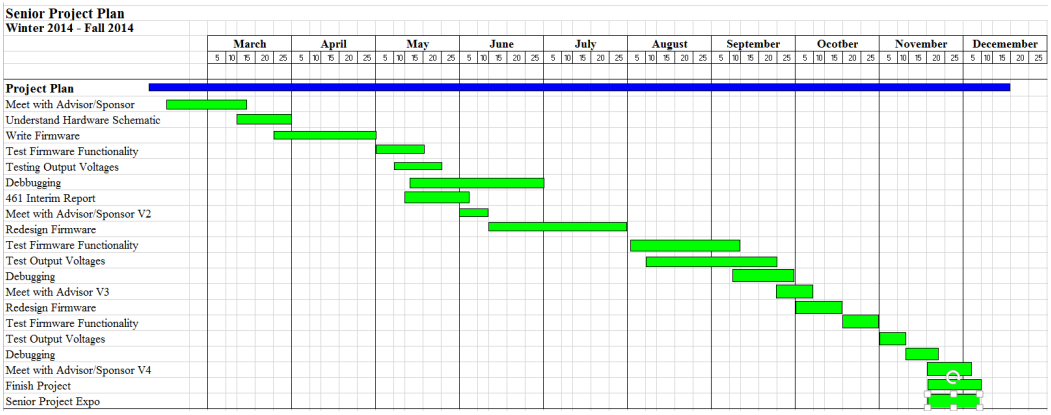


Figure 18: Gantt Chart and Design Time Estimates

The Gantt Chart above starts back in February while project planning with an advisor/sponsor meeting to discuss project goals and discuss deadlines. After interpreting the hardware schematic, the design begins with writing firmware. Next, implementation of the firmware and output voltages are tested. Then debugging of the design occurs. Then another design cycle starts with another advisor/sponsor meeting to discuss the functionality of current design and how to improve the system. The cycle follows the same steps in editing or redesigning the firmware. Project completes after a 4th advisor meeting to discuss the overall design.

Cost Estimates

TABLE VIII

COST ESTIMATES FOR INDIVIDUAL DESIGN COMPONENTS

Part	Cost
Texas Instruments MSP430 USB Debugging Interface	$115.00
Texas Instruments MSP430F543A Microcontroller	$11.76
LT3791-1 Buck Boost Converter	$7.00
Microcontroller Development Board	$80.00
Buck Boost Converter Board	$75.00
TOTAL:	$288.76

Labor

TABLE IX

LABOR AND TIME ESTIMATES FOR COMPLETE PROJECT

Time Estimates	Hours
Optimistic (ta)	**150**
Realistic (tm)	**300**
Pessimistic (tb)	**600**

$Te = \frac{Ta+4Tm+Tb}{6}$ = 325 Hours. Estimated labor costs at $20.00 per hour yields a total labor cost of $6500.

Commercial Estimates

The estimated number of devices sold per year is 5 because its development benefits research for the Monterey Bay Aquarium Research Institute.

After labor, integration, and hardware costs, the estimated cost for each design is $6500.

The estimated purchase price for the design is $7500 ± $500.

The estimated profit per year will be based on the revenue generated solely from selling the design, if a demand exists.

The device self-sustains, operating for months at sea inside the waveglider therefore no user costs exist.

Environmental

Except for the power used to implement and test the design, the project has no immediate environmental impacts. Environmental impacts arise with the purpose of the design. The waveglider collects and transmits data from the ocean and the waveglider implements the project. This product directly affects the Marine ecosystem and the waveglider may negatively affect species that habitat this ecosystem by interrupting feeding, mating, or quality of life within their habitat. [11] The design makes use of two fabricated circuit boards and requires silicon to function as a semiconductor, therefore if manufactured on a large commercial scale, will deplete some of the environments natural resources.

Manufacturability

The design functions mainly for research purposes, not as a consumer product. The implementation of hardware in the waveglider will be challenging to manufacture because it is very specific. Assembly challenges stem from the specific hardware that was fabricated by MBARI and their selection of the MSP430F5438A microcontroller. The firmware created for the hardware integration of the system is specific to the MSP430 libraries and may not operate correctly when using a different microcontroller. [8] The use of other microcontrollers will require the altering of code to satisfy the differing microcontrollers' peripherals.

Sustainability

The project manages the energy harvesting technology in the waveglider and will be self-sustaining, and the operation of the design functions off the energy harvested from the sun and waves therefore positively uses the environment's resources. [11] Microorganisms that inhabit the ocean may decompose the waveglider, and in turn the project design.

Ethics

Evaluating the product and its impact through the Psychological Egoism approach, the product aids in self-sustaining a device that provides efficient data for research that benefits the self-interest of humans. The waveglider self-sustains and the implementation of my design only furthers its overall sustainability and ability to benefit the self-interest of humans through research and data acquisition. The design will avoid real or perceived conflicts of interests when possible, and will disclose them to affected parties while being honest and realistic in stating claims based on available data. The microprocessor management system rejects bribery in all forms and improves the understanding of technology and technical competence. This project does not differentiate the race, religion, gender, disability, age, origin, sexual orientation. It also avoids injury to other, their property, and employment while taking the safety and health of the public along with its environmental impacts into account.

Health and Society

The waveglider may negatively affect organisms that it comes in contact with therefore the health of organisms that inhabit the ocean are at risk.

Social and Political

Direct Stakeholders:

- Liquid Robotics: design may improve the waveglider product.
- MBARI: Company sponsor, design is for their research.
- Cal Poly: Design and testing done at this location. Advisor is a professor at this university.

Indirect Stakeholders:

- Marine Wildlife: Ecosystems may be negatively affected by the project.
- Marine biologists: Project may bring data sooner, improve research.

Development

The use of logic analyzers and oscilloscopes provided a way to test hardware, engineering properties, and microcontroller functionality. Also, many varying approaches in programmable language C, will be explored to determine the most efficient procedure for proper operation. A terminal emulator is utilized to help the development of the firmware.

See References section above.

B. Circuit Diagrams and Schematics

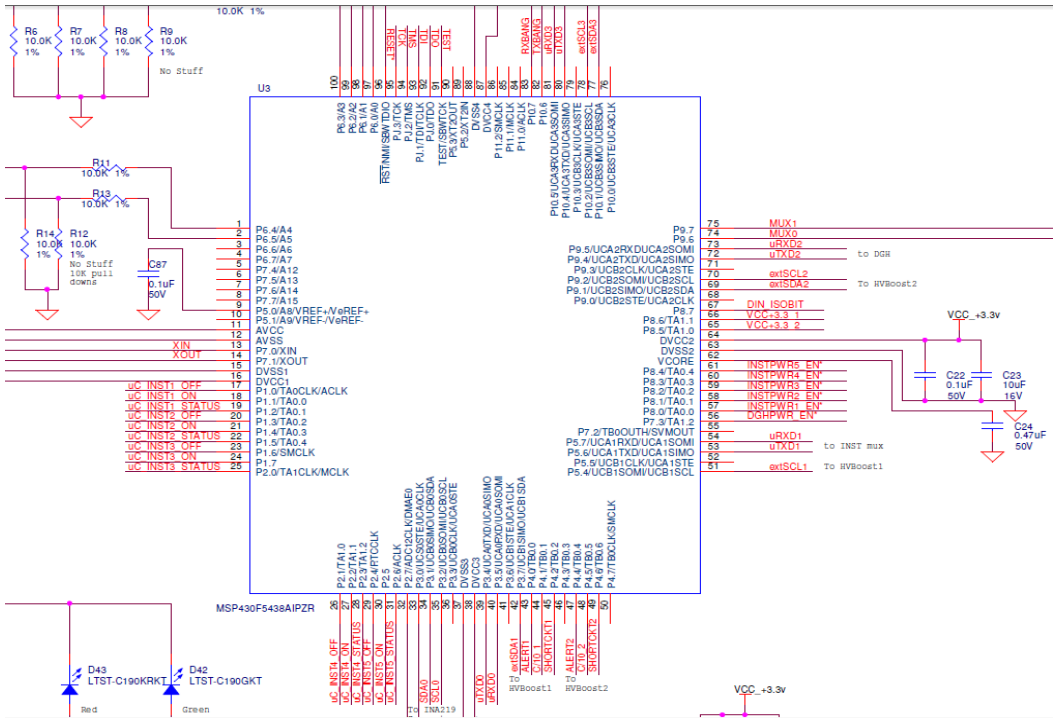


Figure 19: MSP430F5438A Microprocessor on circuit board 1.

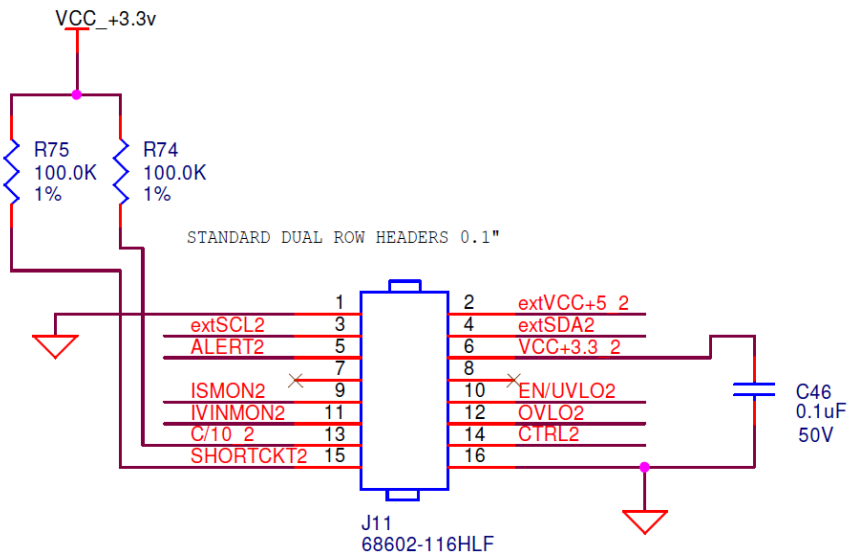


Figure 20: Connection from board 1 to board 2

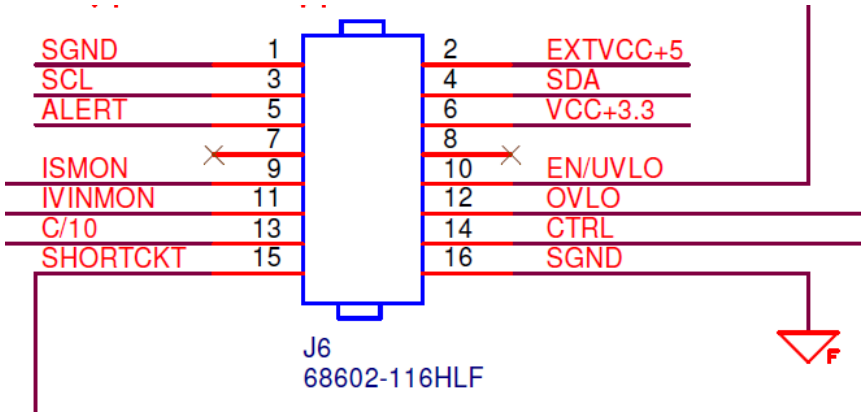


Figure 21: Connection from board 2 to board 1.

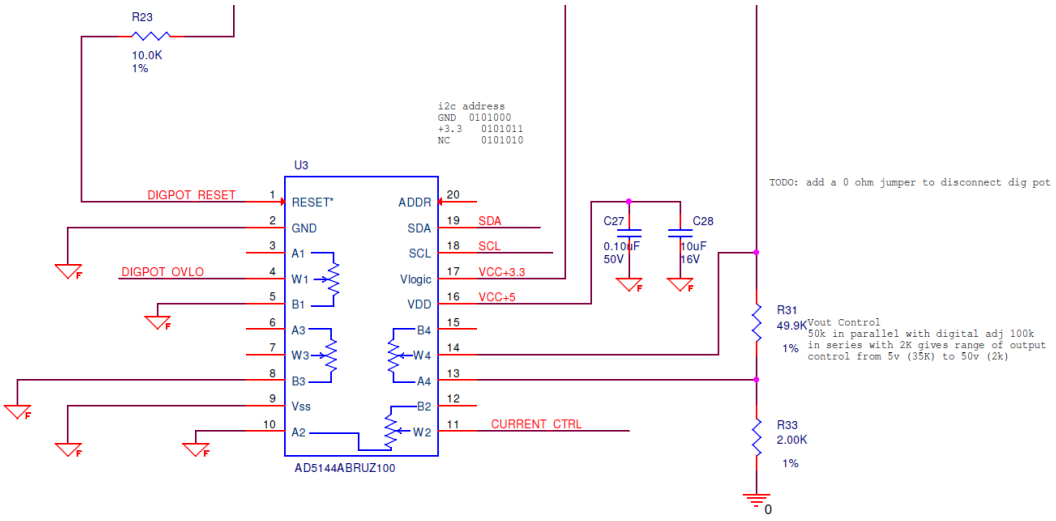


Figure 22: AD5144 Digital Potentiometer on circuit board 2.

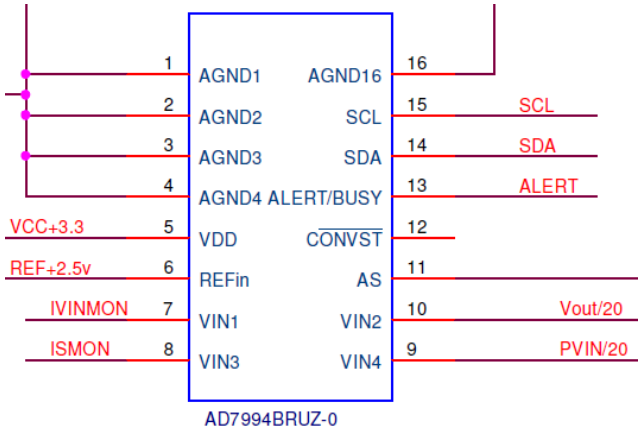


Figure 23: AD7994 Analog to Digital Converter on circuit board 2.

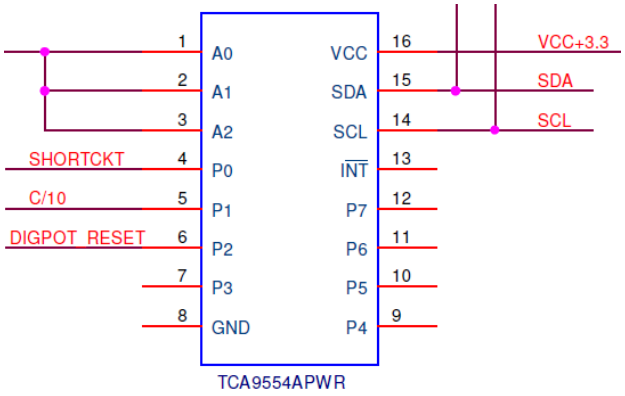


Figure 24: TCA9554 Digital Input/Output on circuit board 2.

C. Parts List and Costs

TABLE X

PARTS LIST AND COSTS

Part	Cost
Texas Instruments MSP430 USB Debugging Interface	$115.00
Texas Instruments MSP430F543A Microcontroller	$11.76
LT3791-1 Buck Boost Converter	$7.00
Microcontroller Development Board	$50.00
Buck Boost Converter Board	$50.00
Analog Devices AD5144 EVAL–MB–LV–SDZ Board	$108.00
Total Phase Beagle (TP320121) Protocol Analyzer	$330.00
Texas Instruments MSP430F5438A Experimenter's Board	$175.00
TOTAL:	$846.76

D. Basic Programs Listing

1. Texas Instruments Code Composer Studio v5.0
2. Tera Term v4.84
3. Total Phase Data Center v6.61 (I2C Protocol Analyzer)
4. Analog Devices AD5144 Evaluation Software

E. Hardware

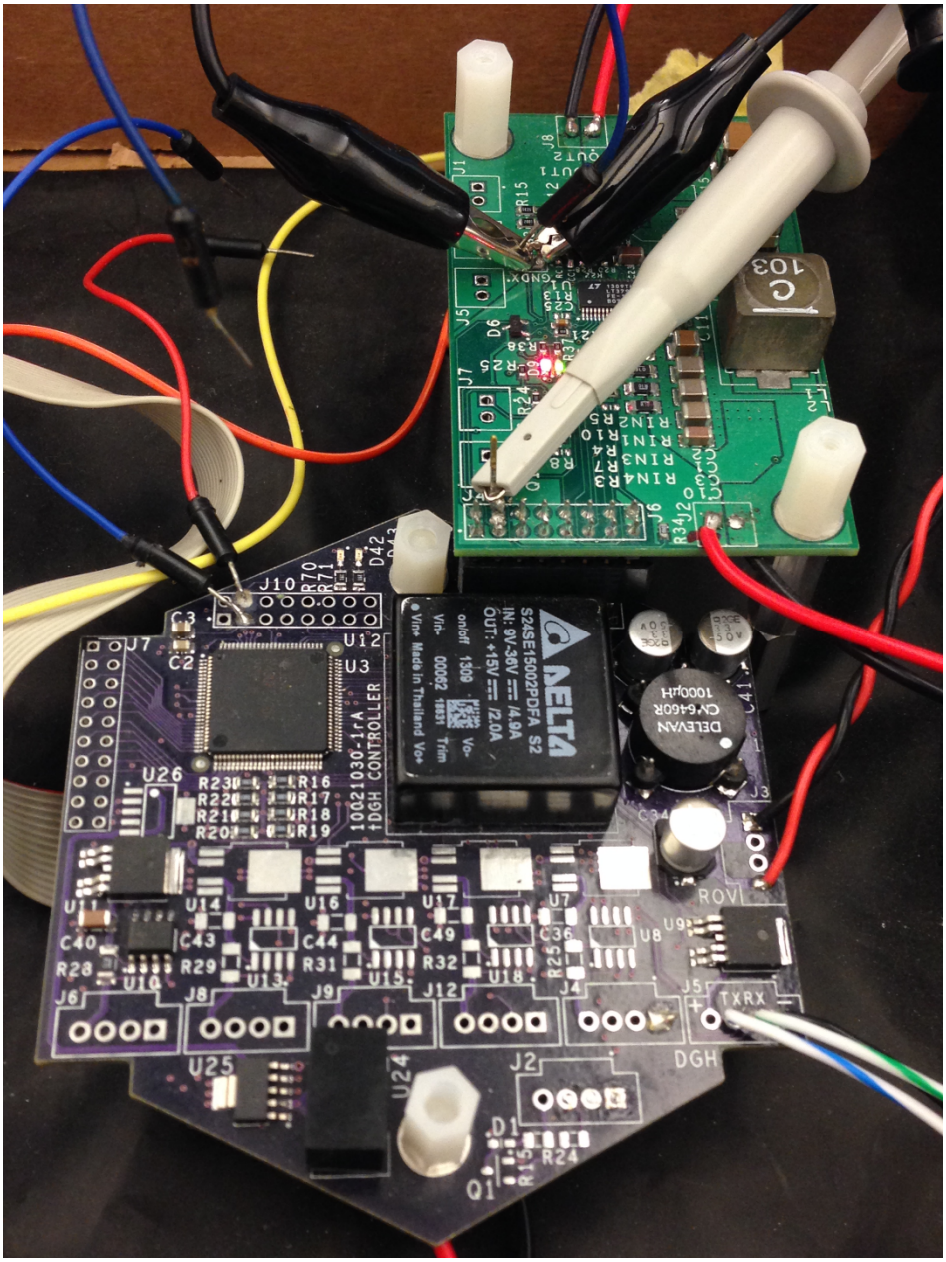


Figure 25: View of microprocessor board (black) and buck-boost board (green).

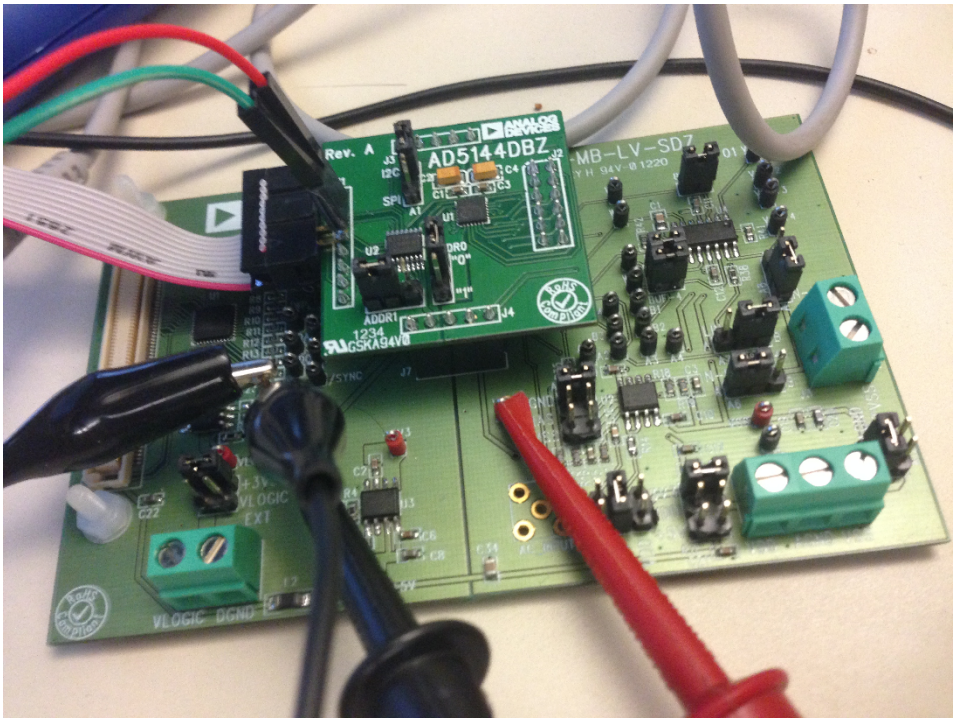


Figure 26: AD5144DBZ Evaluation board

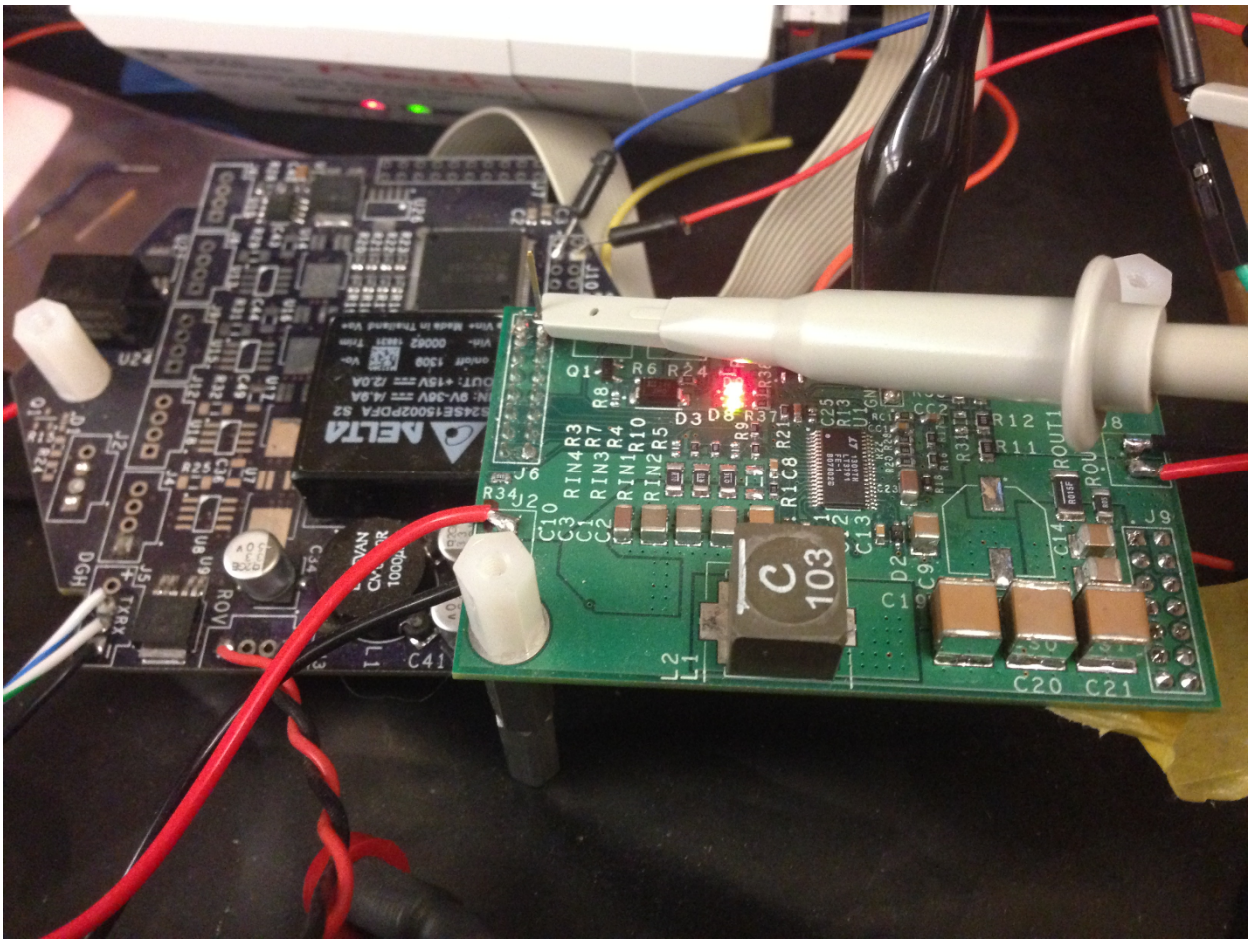


Figure 27: Microprocessor board on the left, buck-boost board on the right.

F. Code

LED Toggle

```
/*
 * Blink LED D43 Test
 */
#include <msp430.h>
#define LED_RED BIT0

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;       // Stop watchdog timer
    P3DIR |= LED_RED;               // Set P3.0 to the output direction
    P3OUT &= ~(LED_RED);            // Sets the D43 red LED off

    TA1CCTL0 = CCIE;
    TA1CCR0 = 50000;
    TA1CTL = TASSEL_2 + MC_2; // Set the timer A to SMCLCK, Continuous

    __enable_interrupt();
    __bis_SR_register(LPM0 + GIE); // LPM0 with interrupts enabled
      }

// Timer A0 interrupt service routine
#pragma vector=TIMER1_A0_VECTOR
__interrupt void TIMER1_A0_ISR (void)
{
      P3OUT ^= (LED_RED);
      TA1CCR0 += 50000;
}
```

UART Testing

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;            // Stop WDT
  P9SEL = 0x30;                        // P9.4,5 = USCI_A0 TXD/RXD
  UCA2CTL1 |= UCSWRST;                 // **Put state machine in reset**
  UCA2CTL1 |= UCSSEL_2;                // SMCLK
  UCA2BR0 = 6;                         // 1MHz 9600 (see User's Guide)
  UCA2BR1 = 0;                         // 1MHz 9600
  UCA2MCTL = UCBRS_0 + UCBRF_13 + UCOS16; // Modln UCBRSx=0, UCBRFx=0,
                                                 // over sampling
  UCA2CTL1 &= ~UCSWRST;             // **Initialize USCI state machine**
  UCA2IE |= UCTXIE;                          // Enable USCI_A2 TX
interrupt

  while(1){
__bis_SR_register(LPM0_bits + GIE); // Enter LPM0, interrupts enabled
  __no_operation();                          // For debugger

}
```

```
}

#pragma vector=USCI_A2_VECTOR
__interrupt void USCI_A2_ISR(void)
{
  switch(__even_in_range(UCA2IV,4))
  {
  case 0:break;                             // Vector 0 - no interrupt
  case 2:                                   // Vector 2 - RXIFG
    break;
  case 4:
    UCA2TXBUF = 0x55;
    break;                                // Vector 4 - TXIFG
  default: break;
  }
}
```

Serial Port Menu

```
#include <msp430.h>

const char string[200] = { "Welcome to the Buck-Boost Converter\r\n"
                         "Enter a command to control the following: \r\n"
                         "Digital Input/Output - D \r\n"
                         "Analog to Digital Converter - A \r\n"
                         "Digital Potentiometer - P \r\n"};

unsigned int  i = 0;//Counter
int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  P9SEL = 0x30;                            // P9.4,5 = USCI_A2 TXD/RXD
  UCA2CTL1 |= UCSWRST;                    // **Put state machine in reset**
  UCA2CTL1 |= UCSSEL_2;                     // SMCLK
  UCA2BR0 = 6;                              // 1MHz 9600 (see User's Guide)
  UCA2BR1 = 0;                                  // 1MHz 9600
  UCA2MCTL = UCBRS_0 + UCBRF_13 + UCOS16;   // Modln UCBRSx=0, UCBRFx=0
                                            // over sampling
  UCA2CTL1 &= ~UCSWRST;                     // **Initialize USCI state
machine**
  UCA2IE |= UCTXIE + UCRXIE;              // Enable USCI_A0 RX interrupt




  __bis_SR_register(LPM0_bits + GIE); // Enter LPM0, interrupts enabled

}


#pragma vector=USCI_A2_VECTOR
__interrupt void USCI_A2_ISR(void)
{
```

```
      switch(__even_in_range(UCA2IV,4))
      {
      case 0:break;                             // Vector 0 - no interrupt
      case 2:                                   // Vector 2 - RXIFG
            if (UCA2RXBUF == 'a') // 'a' received?
          {
            UCA2IE |= UCTXIE; // Enable USCI_A0 TX interrupt
            UCA2TXBUF = string[i++];
          }
      case 4:           // Vector 4 - TXIFG
/*
              UCA2TXBUF = string[i++];           // TX -> RXed character
              __bic_SR_register_on_exit(LPM0_bits);  */
              break;
      default: break;
      }
}
```

I2C Digital Potentiometer Byte Transmit

```
void i2c_init(void);
void uart_init(void);

#include <msp430.h>

unsigned char *PTxData;                     // Pointer to TX data
unsigned char TXByteCtr;

const unsigned char TxData[] =               // Table of data to transmit for
{                                            // +1 Step on channel 1.
  0x40,
  0x01,
  0x34,
  0x03
};

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                  // Stop WDT
  P9SEL |= 0x06;                             // Assign I2C pins to   USCI_B2
  P9REN |= 0x06;
  UCB2CTL1 |= UCSWRST;                       // Enable SW reset
UCB2CTL0 = UCMST + UCMODE_3 + UCSYNC;     // I2C Master, synchronous mode
 UCB2CTL1 = UCSSEL_2 + UCSWRST;            // Use SMCLK, keep SW reset
 UCB2BR0 = 12;                             // fSCL = SMCLK/12 = ~100kHz
 UCB2BR1 = 0;
 UCB2I2CSA = 0x2A;                         // Slave Address for Digital
Potentiometer
 UCB2CTL1 &= ~UCSWRST;                     // Clear SW reset, resume operation
 UCB2IE |= UCTXIE;                         // Enable TX interrupt

         /* UART Initialization Sequence */
    P9SEL = 0x30;                             // P9.4,5 = USCI_A2 TXD/RXD
    UCA2CTL1 |= UCSWRST;                      // **Put state machine in reset**
    UCA2CTL1 |= UCSSEL_2;                     // SMCLK
```

```
    UCA2BR0 = 6;                                // 1MHz 9600 (see User's Guide)
    UCA2BR1 = 0;                                // 1MHz 9600
    UCA2MCTL = UCBRS_0 + UCBRF_13 + UCOS16;     // Modln UCBRSx=0, UCBRFx=0,
                                                                    // over
       sampling
    UCA2CTL1 &= ~UCSWRST;                       // **Initialize USCI state
machine**
 // UCA2IE |= UCRXIE;                          // Enable USCI_A0 RX interrupt

  while (1)
 {
    __delay_cycles(50);                      // Delay required between
transaction
    PTxData = (unsigned char *)TxData;       // TX array start address
                                                    // Place breakpoint here to
       see each
                                                    // transmit operation.
    TXByteCtr = sizeof TxData;               // Load TX byte counter

    UCB2CTL1 |= UCTR + UCTXSTT;              // I2C TX, start condition

    __bis_SR_register(LPM0_bits + GIE);      // Enter LPM0, enable interrupts
    __no_operation();                        // Remain in LPM0 until all data
                                                    // is TX'd

    while (UCB2CTL1 & UCTXSTP);              // Ensure stop condition got sent
  }
}

//-------------------------------------------------------------------------
// The USCIAB2TX_ISR is structured such that it can be used to transmit any
// number of bytes by pre-loading TXByteCtr with the byte count. Also, TXData
// points to the next byte to transmit.
//-------------------------------------------------------------------------

#pragma vector = USCI_B2_VECTOR
__interrupt void USCI_B2_ISR(void)
{
  switch(__even_in_range(UCB2IV,12))
  {
  case  0: break;                            // Vector  0: No interrupts
  case  2: break;                            // Vector  2: ALIFG
  case  4: break;                            // Vector  4: NACKIFG
  case  6: break;                            // Vector  6: STTIFG
  case  8: break;                            // Vector  8: STPIFG
  case 10: break;                            // Vector 10: RXIFG
  case 12:                                   // Vector 12: TXIFG
if(TXByteCtr)
{
       UCB2TXBUF = *PTxData;               // Load TX buffer
      UCA2IE |= UCTXIE;
      UCA2TXBUF = UCB2TXBUF;           // Transmit i2c bus data to the UART
       TXByteCtr--;                          // Decrement TX byte counter
       *PTxData++;
}
else {
      UCB2CTL1 |= UCTXSTP;                   // I2C stop condition
      UCB2IFG &= ~UCTXIFG;                   // Clear USCI_B0 TX int flag
      __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
} break;
  default: break;
  }
}
```

```
void i2c_init()
{
        /* I2c Initialization sequence */
           P9SEL |= 0x06;  // Assign I2C pins to USCI_B2
           P9REN |= 0x06;
          // P9DIR |= 0x06;


           UCB2CTL1 |= UCSWRST;                        // Enable SW reset
           UCB2CTL0 = UCMST + UCMODE_3 + UCSYNC; // I2C Master, I2C mode,
synchronous mode
           UCB2CTL1 = UCSSEL_2 + UCSWRST;     // Use SMCLK, keep SW reset
           UCB2BR0 = 12; // fSCL = SMCLK/12 = ~100kHz  (Baud Rate Control
Register 0)
           UCB2BR1 = 0;


           UCB2I2CSA = 0x22;               // Slave Address (in hex)


           UCB2CTL1 &= ~UCSWRST;      // Clear SW reset, resume operation
           UCB2IE |= UCTXIE + UCRXIE;      // Enable TX and RX interrupt;
}

void uart_init()
{
        /* UART Initialization Sequence */
           P9SEL = 0x30;                        // P9.4,5 = USCI_A2 TXD/RXD
           UCA2CTL1 |= UCSWRST;         // **Put state machine in reset**
          UCA2CTL1 |= UCSSEL_2;                     // SMCLK
          UCA2BR0 = 6;                   // 1MHz 9600 (see User's Guide)
          UCA2BR1 = 0;                              // 1MHz 9600
          UCA2MCTL = UCBRS_0 + UCBRF_13 + UCOS16;   // Modln UCBRSx=0,
UCBRFx=0,
                                                    // over sampling
         UCA2CTL1 &= ~UCSWRST;     // **Initialize USCI state machine**
        // UCA2IE |= UCRXIE;                     // Enable USCI_A0 RX interrupt

}
```