Battery Monitoring for Electric Vehicle Battery Packs

June 5, 2016

Senior Project for

Matthew Hennes

with advisor

Jeffrey Gerfen

Contents

Introduction

Purpose

Battery packs containing multiple batteries in series must be kept balanced, with each individual battery at the same voltage in order to increase the longevity of the battery pack as a whole. If some batteries in the pack are a significantly higher voltage than others, those higher voltage batteries will charge more quickly, and will then be damaged by overcharging while the lower voltage batteries catch up. In lead acid batteries, overcharged batteries will gas, consuming electrolyte in the process, and will therefore need more frequent maintenance (watering) to be kept in working order. [1]

In order to keep battery packs balanced, individual batteries are must periodically be removed and charged individually, so that all batteries in the pack are at or near the same potential. To determine when this is necessary with a simple, “dumb” battery pack, a technician would typically open the battery pack and measure to potential across each individual battery by hand. This process could be greatly simplified if the measurement process happened automatically without the need to open the battery pack.

The purpose of this project is to develop a system capable of taking automated measurements of the voltages of the batteries in the battery pack for the Electric Vehicle Engineering Club (EVEC)’s electric van. The club recently needed to replace two battery packs because they were destroyed by severely unbalanced batteries. This large cost for the club could have been avoided if a system like this was being used to keep track of individual battery voltages.

Deliverable

The deliverable for this project will be a working prototype of a battery monitoring system that is capable of measuring the individual voltages of batteries in a battery pack. This system will be developed to work with the EVEC battery packs, and the final prototype and designs will be delivered to the club for final integration into their battery packs. The prototype will not be physically located in the battery pack, as the specific details of how the system should be

positioned within the battery pack is better left up to the club to determine based on their needs. However, all components used in the prototype will be capable of physically fitting inside the battery packs so that no redesign will be required of EVEC aside from laying out the locations of components within the battery packs. Complete system designs will also be made available to EVEC so that they can create additional battery monitoring systems in the future, or modify the design if their needs change.

Requirements

This battery monitoring system will be used to measure the individual voltages of batteries in a battery pack. While it may also be capable of monitoring other designs, the system will be specifically intended to monitor lead acid batteries wired in series to create a single, high voltage battery pack. As such, the system shall be capable of measuring the individual voltages of eighteen batteries. The maximum measurable voltage shall be 250V, which makes the system ideal for measuring the voltages of standard 12V batteries such as the ones used in the automotive and marine industries. The monitoring system shall provide sufficient information to make the manual measurement of battery voltages unnecessary. In order to achieve this, battery voltages shall be reported individually (rather than as combinations of batteries), and will provide sufficient detail to make informed decisions about when battery balancing is necessary. The system shall be capable of measuring battery voltage to within 2% of the true value, and shall be capable of taking a measurement of each battery at least once per minute, although it will generally be used at a lower frequency. The system shall pull less than 1mA from the battery pack during measurement, and shall connect batteries in such a way that they are isolated from each other in order to prevent short circuits, and will not interfere with regular usage of the battery pack. The system shall be capable of operating at temperatures of at least 50°C with no airflow, and shall withstand temperatures of up to 100°C while not in use.

System Specifications

Specification	Minimum	Typical	Maximum
Input voltage (V DC)		216	250
Number of individual batteries	1	18	18
ADC precision (bits, plus a sign bit)		12	
Voltage display precision (decimal places)		2	
Measurement accuracy (mV)	360	240	61
Measurement time (per battery) (ms)		150	
Measurement frequency (18 batteries, measurements per minute)		1	20
Auxiliary battery voltage (V)	6	9	12
System power consumption (from auxiliary battery, mA)		15	
Auxiliary battery life (standard 9V battery, hours)		650	
Measurement current draw (from target battery pack, at 216 V, µA)		853	
Operating temperature (°C)	0	25	70[1]
Storage temperature (°C)	-65		150
UART baud rate		9600	
Microcontroller clock frequency (MHz)		16	
Physical size (cm)		16.5 x 16.5	
Weight (g)		198	

[1] Note: It is not recommended to charge or discharge lead acid batteries above 50°C. [4]

System Architecture

System Overview

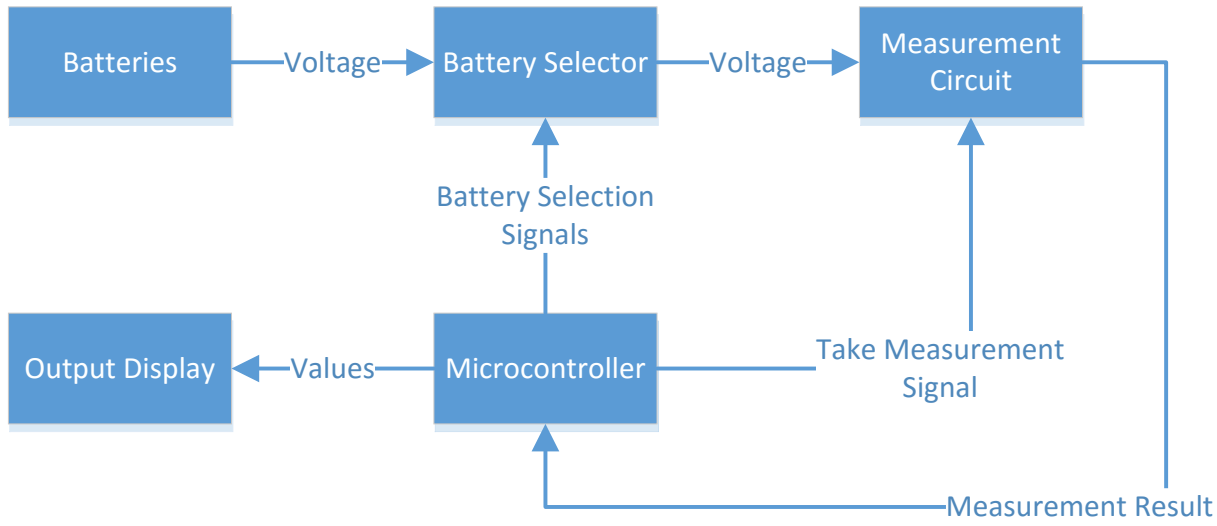

Figure 1: System Block Diagram

The battery monitoring system essentially consists of three main parts: the battery selector, the measurement circuit, and the microcontroller. These three components work together to read the voltages from the batteries and ultimately output those voltages to the output display. This process begins with the microcontroller sending a signal to the battery selector. This signal indicates to the battery selector which battery is currently being measured, and the battery selector then allows that battery's voltage to pass through to the measurement circuit. The microcontroller then sends a signal to the measurement circuit indicating that a measurement should be taken. The measurement circuit measures the voltage passed to it from the battery selector and returns the result to the microcontroller. Finally, the microcontroller repeats this process for each battery, storing the result of each measurement before outputting all measurements to the display.

Hardware Overview

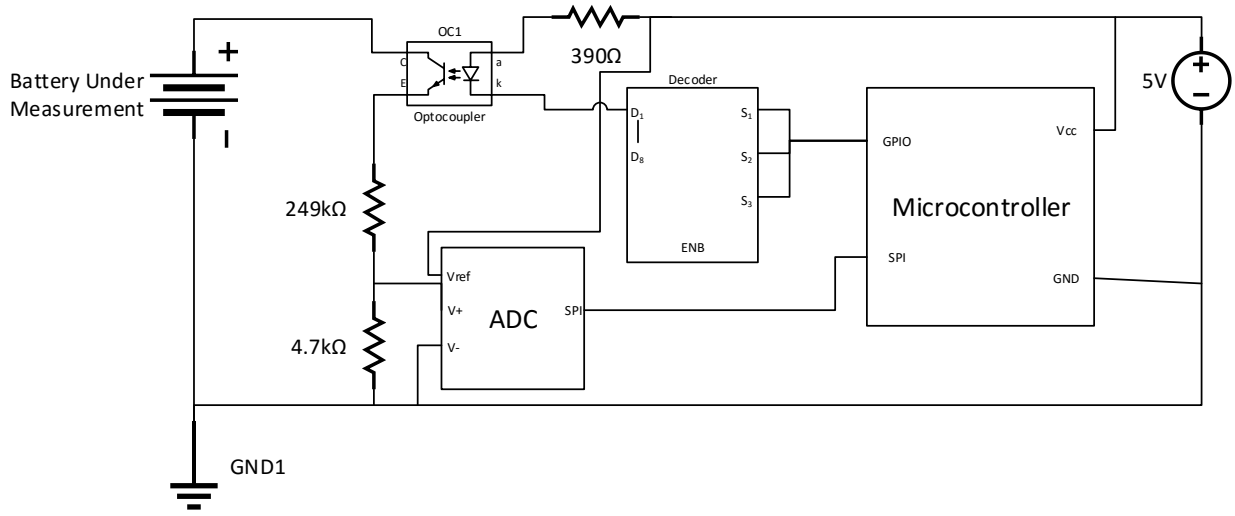

Figure 2: Hardware Overview - Illustrates how a single battery is connected to the system. In the full system, there are eighteen optocouplers, one for each battery. See Figure 3 for a full system schematic including all eighteen batteries/optocouplers.

The microcontroller portion of the system consists of an Adafruit Atmega32u4 breakout board. The battery selector portion of the system consist of eighteen optocouplers (one per battery) which can create either a short circuit or an open circuit, allowing individual batteries to either be passed through uninhibited or completely isolated. This allows a single battery to be measured. These optocouplers are driven by 3:8 decoders. The measurement circuit consists of a voltage divider and an analog to digital converter (ADC), which is interfaced to the microcontroller over a serial peripheral interface (SPI) connection.

Figure 2 shows how a single battery would be connected in order to be measured. Because the microcontroller has limited general purpose input/output (GPIO) pins, 3:8 decoders are used to interface the microcontroller to the eighteen individual optocouplers (one per battery). Each of these decoders requires three GPIO pins to address, plus one GPIO pin to select which decoder is currently in use, for a total of four pins per decoder. Each decoder is connected to six optocouplers, so a total of three decoders are needed. These three decoders use a total of 12 GPIO pins, as opposed to the 18 pins that would be required to directly drive the optocouplers from the microcontroller. Further optimization could be achieved by using the same GPIO pins

for the address signals to all three decoders, reducing the number of GPIO pins used to six, but this level of optimization is not necessary, so unique address pins are used for each decoder in order to improve code clarity. See Figure 3 for a full system schematic.

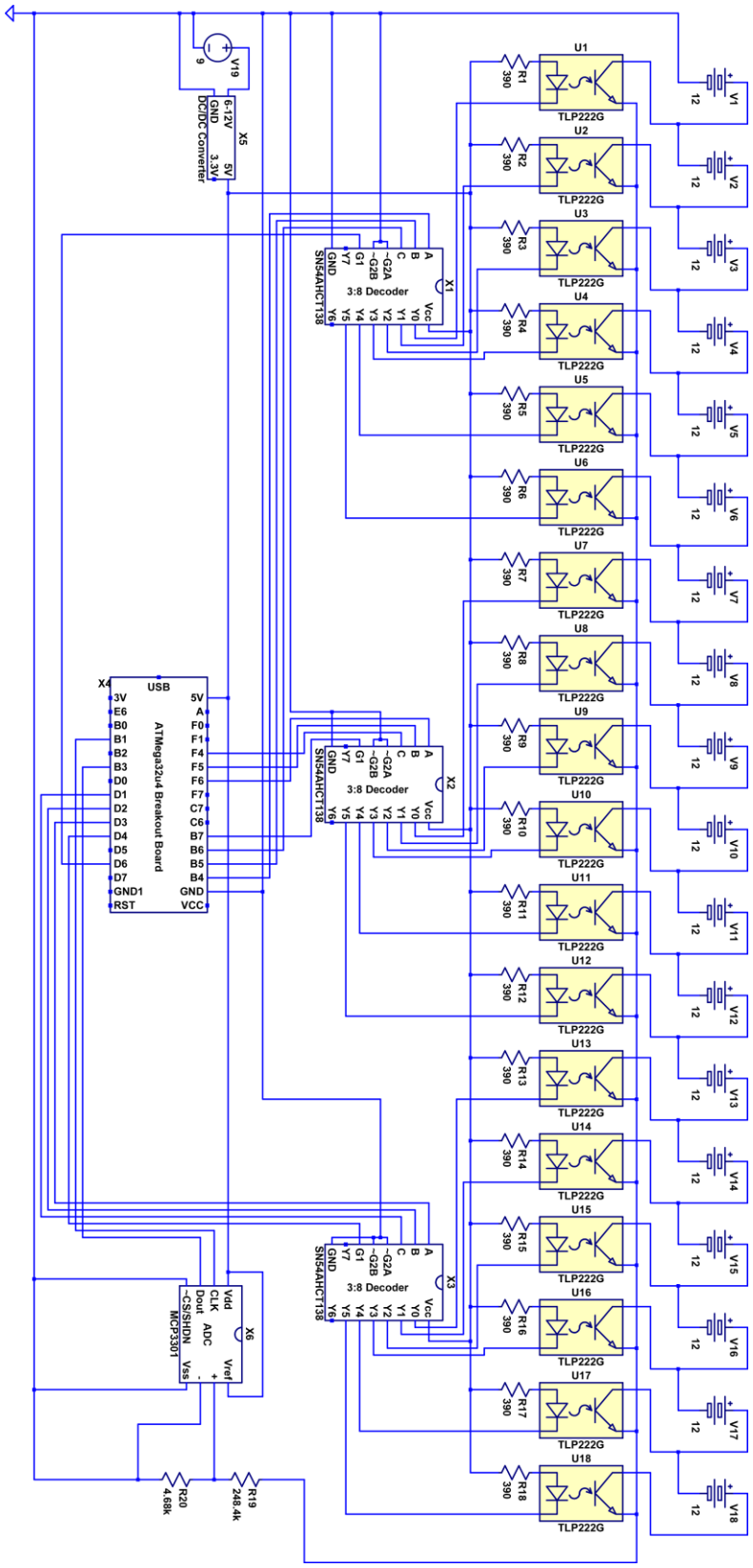

Figure 3: Full system schematic

The vast majority of microcontrollers and integrated circuits (such as ADCs) run on 3.3V or 5V, and will be destroyed by the types of voltages used in automotive battery packs. The full voltage of the battery pack (up to 250V) cannot be connected directly to the microcontroller or an external ADC, and therefore cannot be measured directly. To resolve this, a voltage divider is used, and voltage is measured across the smaller resistor. In this case we use a 249kΩ resistor and a 4.7kΩ resistor to form an equivalent resistance of 253.7kΩ. Voltage is measured across the 4.7kΩ resistor, which gives us a 53.98:1 gain. That is, each volt measured by the ADC is equivalent to 53.98 volts on the actual battery being measured. This is then compensated in software by multiplying the ADC's output by 53.98 to calculate the real voltage of the battery. While resistors with 1% tolerances on resistances were used, the exact resistance value of each resistor (as measured by a digital multimeter) was used to calculate these values. The values are automatically calculated in software based on resistance constants. Therefore, if more instances of this device were created, accuracy could be maintained despite slightly different resistance values by simply changing the two resistor value constants in the code to represent the true resistances of the resistors being used.

Most microcontrollers have onboard ADCs, but these onboard ADCs are generally limited to 10-bit or lower resolution. Ten-bit resolution (1,024 steps) would give us a step size of 244.1mV step size when measuring a 250V battery. This step size defines the maximum (worst case) accuracy with which we can measure battery voltage. This 244.1mV step size equates to 2.034% of the nominal voltage of a 12V battery, which is just outside the required accuracy of 2%. To achieve higher accuracy, this system uses a discrete ADC with a 12-bit resolution (4,096 steps). With 12-bit resolution, our worst case accuracy is improved to 61.04mV, or 0.5086% of the nominal voltage of a 12V battery, well within the requirement of 2%.

Software Overview

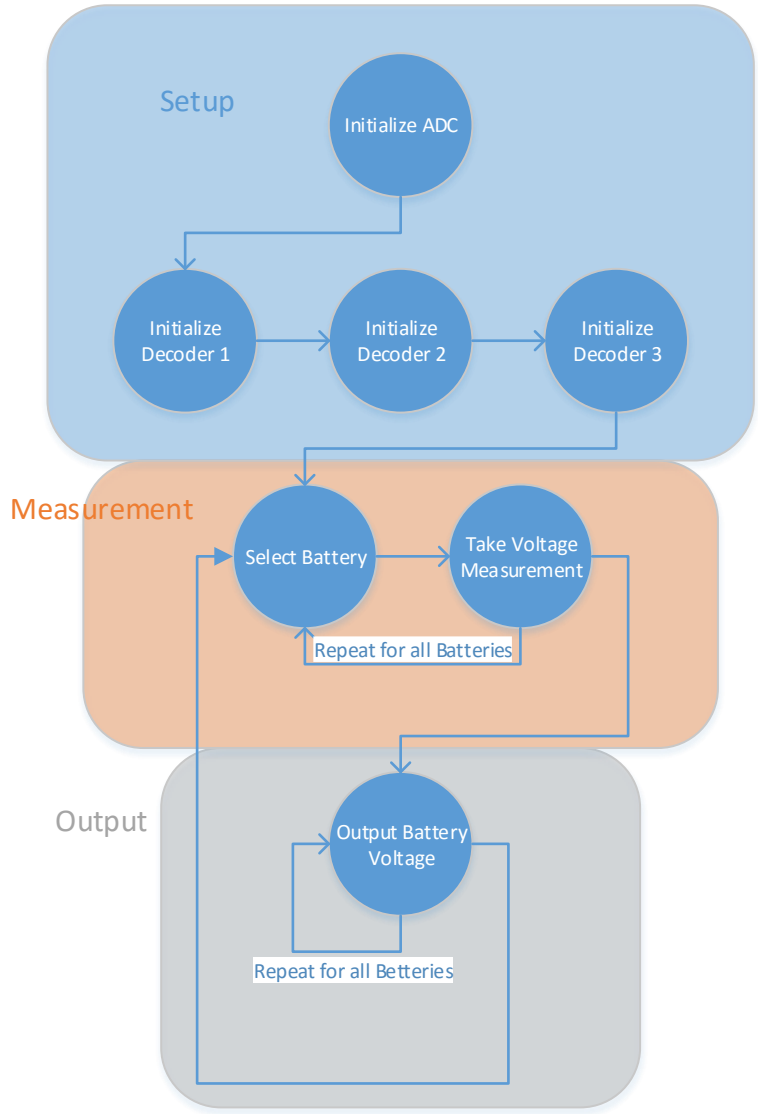

Figure 4: Software Flow

The microcontroller code for this system works in three primary steps: setup, measurement, and output. The setup step initializes all components as well as the microcontroller itself. It also starts up SPI (used to communicate with the ADC) and universal asynchronous receiver/transmitter (UART) (used for outputting results) on the microcontroller.

The measurement step runs in a loop until the voltage of each battery has been measured. First, it ensures that all optocouplers are disabled (to ensure that a short between different batteries

is not created). Then, the necessary signals are sent to the relevant decoder to turn on the optocoupler associated with the battery to be measured. After a short delay to allow transients to level off, the microcontroller then begins clocking the SPI clock signal, which initiates a measurement on the ADC. The ADC then returns the measured value to the microcontroller via SPI. The microcontroller stores this value and moves on to the next battery.

The output step iterates through all the recorded voltages for each battery, converting the raw ADC value to a voltage value, and sends it out via UART. Raw data comes from the ADC in the form of an unsigned integer between 0 and 4095. In order to obtain the voltage of each battery, the ADC's output value for each battery is subtracted from the preceding value. So for example, the measurement for the fourth battery is the value measured on the fourth battery minus the value measured on the third battery. The raw value is kept for the first battery, as its negative terminal is connected to ground, which is also the signal used for the ADC's negative input. The output of the ADC is an unsigned integer from 0 to 4,095, so once the potential difference for each battery has been calculated, it must be converted to a useful floating point representation. This is done by multiplying each battery's measurement by a constant, floating point value which translates the ADC's output into a meaningful voltage. See

Appendix: Microcontroller Code for the full source code.

Component Selection

Microcontroller

An Adafruit ATMega32u4 Breakout Board was used for the microcontroller (reference designator X4 in Figure 3). This board provides a through hole pinout to the normally surface mount Atmel ATMega32u4 microcontroller, as well as providing a regulator to supply 3.3V and 5V, a USB port for programming and UART communication, and a physical reset button. This particular microcontroller board was selected because it uses the same microcontroller chip as the Arduino Leonardo, making it fully compatible with the Arduino software library. [2] This board is, however, much smaller than the Arduino Leonardo, as well as being more readily obtainable, as the Arduino Leonardo is no longer in production.

Analog to Digital Converter

The ADC (reference designator X6 in Figure 3) used in the system was a Microchip MCP3301. This ADC provides 13-bit (12 data bits plus a sign bit) values, and is capable of taking up to 100 kilosamples per second. This particular ADC was selected for this system for its convenient SPI communication interface. In order to take a reading from this ADC, a user need only clock out 16 clock cycles on the SPI interface, just as if a 16-bit value was being read via SPI (16 bits is a common data size so send/receive over SPI). The first clock begins the process of taking the measurement, and the ADC then sends zero values for the first three clock cycles. By the fourth clock cycle, the measurement is complete, and the ADC sends out the value in the remaining 13 clock cycles. [3] In this way, it is possible to simply read a 16-bit value over SPI with no need to start a conversion separately.

Other Components

Other components used in this system are relatively standard models that provide no special features. These components, including decoders, resistors, and optocouplers, were selected primarily on the basis of cost and availability.

Bill of Materials

Part Description	Reference Designator	Distributor	Distributor Part Number	Quantity Required	Unit Price	Total Price
Atmega32u4 Breakout Board	X4	adafruit	296	1	$19.90	$19.90
249kΩ Resistor (CMF55249K00BEEB)	R19	Digi-Key	CMF249KHBCT-ND	1	$0.93	$0.93
4.7kΩ Resistor (MFP-25BRD52-4K7)	R20	Digi-Key	4.7KADCT-ND	1	$0.46	$0.46
Optocoupler (TLP222G(F))	U1-U18	Digi-Key	TLP222GF-ND	18	$1.013 (quantities of 10 or more)	$18.24
Analog to Digital Converter (MCP3301-CI/P)	X6	Digi-Key	MCP3301-CI/P-ND	1	$2.27	$2.27
3:8 Decoder (SN74AHCT138N)	X1-X3	Digi-Key	296-4666-5-ND	3	$0.39	$1.17
Total						**$42.97**

Testing

Test Setup

In order to test the system, the battery inputs were connected to the individual positive terminals of eighteen 12V batteries, which are wired in series to make up a single 216V battery pack. The battery monitoring system's ground was tied to the battery pack's ground (the negative terminal of the first battery) as shown in Figure 3. The batteries used were lead acid marine and RV batteries, and the true voltages of the batteries (as measured by a digital multimeter) ranged between 12.28V and 12.56V, and were fully charged (or nearly so) at the time of testing. The average voltage of the batteries was 12.47V, and the total voltage of the entire pack was 224.51V. A standard 9V alkaline battery was used as the auxiliary battery during testing, and results were printed via UART over USB to a laptop. The monitoring system was allowed to test all batteries and print the results. The system was then shut down (disconnected from power), and the battery voltages were measured manually. This process was repeated five times, for a total of 90 individual battery measurements.

Test Results

During testing, the average measurement error was 35mV above the actual voltage of the battery being tested, across all batteries. This equates to 0.28% error, well within the required 2%. Furthermore, this average error is well under the maximum accuracy of the ADC, 61mV, which suggests that the primary source of error is the inherent limitations of the ADC. In only one instance was the measured voltage more than one ADC step (61mV) different from the actual voltage. In that instance the measured voltage was 120mV (0.96%) away from the actual value, less than two ADC steps (122mV), and still well within the required 2% error. No single battery consistently showed a greater error than the others.

Future Work

While this system does a good job of measuring battery voltage, it currently requires a wired connection to read the results of the measurement via UART. While connecting a cable is still more convenient for a technician than disassembling the battery pack in order to measure voltages, it would be even better if the results of the measurement could be read wirelessly. This could best be done with either a discrete Bluetooth module or a discrete Wi-Fi module. The best way in which to implement this would be using a discrete Wi-Fi module so that data could be uploaded directly to a remote server without the need for a nearby receiver, such as would be needed with Bluetooth. At the time of writing, one such module that would work well is the Espressif Systems ESP8266EX (Digi-Key part number 1528-1438-ND). This module uses 802.11b/g/n (2.4GHz), and has a maximum data rate of 72.2Mbps, much higher than would be necessary for this system. The primary advantages of this module over others are cost ($6.95 at time of writing), and the convenient SPI interface. Because of the SPI interface, this module could be added to the system with only two additional pins used on the microcontroller. This would be accomplished by reusing the SPI data pins that are already used for the ADC, and simply adding chip select pins, one for the ADC and one for the Wi-Fi module. The microcontroller could then communicate with either the Wi-Fi module or the ADC at any given time (not both simultaneously). This would work because data would be uploaded after all measurements had been taken and the microcontroller was idling waiting to take the next measurement, so there would never be a situation in which it was necessary to communicate with the Wi-Fi module and the ADC simultaneously.

While voltage is the primary factor important for keeping batteries in good condition, other information could potentially be useful as well, especially if data could be read from the battery while charging, or while the vehicle is in operation. The most important factors that could be measured would be current draw from the entire battery pack and the temperature, either of each individual battery or of the pack as a whole. Temperature would be useful to a user because it is not recommended to charge batteries at extremely high or low temperatures, and an extremely warm battery during vehicle operation could be a sign of a problem with the battery

pack. Current would be useful to a vehicle operator because it would allow them to estimate power usage during driving (and regeneration during braking if the vehicle is equipped with that system), and would therefore allow them to estimate potential range. Temperature could be measured using a thermistor, and current draw could be measured using a Hall Effect current transducer such as the HASS 400-S (Digi-Key part number 398-1066-ND). This sensor would be ideal for measuring current because it does not need to be in direct contact with the circuit. Current flowing through the loop of the Hall Effect sensor is translated to a potential on the sensor's output pins. In this way, the large currents (up to 40A) pulled by the vehicle during operation could be measured by the system, with no power waste, such as would occur if current was measured by measuring the voltage across a resistor in series with the load. The output, an analog voltage from 0 to 5V, could be measured using one of the microcontroller's internal 10-bit ADCs for moderate precision (48.8mA per step), or a second ADC could be interfaced for 12-bit precision (12.2mA per step). If an external ADC was used, it could be interface in the same way discussed above for a Wi-Fi module, by adding chip select pins to the microcontroller and reusing the same SPI pins as the current ADC.

Finally, it would be desirable for the system to be entirely powered by the battery pack itself so that it would not be necessary for the battery to be replaced. This could be achieved by using a DC to DC converter to step down the battery pack voltage (216V nominal) to the voltage required for the electronics (5V). Because the measurement system uses less than a tenth of a watt, the system would be able to run indefinitely without having any significant effect on the battery pack if it were powered entirely from the battery pack itself. One drawback of this addition is that DC to DC converters capable of stepping down from 216V to 5V are relatively expensive (about $80 at time of writing, nearly double the cost of the entire current system). This cost could be significantly reduced (in exchange for added complexity) by using two DC to DC converters rather than a single one. In this case, the first converter would step down from 216V to 25-50V (depending on the exact device chosen), and the second to step down from the first converter's output to 5V. This combination approach can be built for about $40 at time of writing, much

more inexpensive than a single converter, but still comparable to the cost of the entire current system.

References

[1] C&D Technologies, "Charging Valve Regulated Lead Acid Batteries," C&D Technologies, Inc., Blue Bell, PA, 2012.

[2] l. ada, "Using with Arduino," 8 February 2016. [Online]. Available: https://learn.adafruit.com/atmega32u4-breakout/using-with-arduino. [Accessed 6 June 2016].

[3] Microchip, "MCP3301," December 2001. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21700E.pdf. [Accessed 6 June 2016].

[4] Battery University, "BU-410: Charging at High and Low Temperatures," Cadex Electronics Inc., 2 April 2016. [Online]. Available: http://batteryuniversity.com/learn/article/charging_at_high_and_low_temperatures. [Accessed 6 June 2016].

Appendix: Microcontroller Code

```
// Top resistor: 248.4 k
// Bottom resistor: 4.68 k
// Req: 253.08 k

#include <SPI.h>

const int ADC_VAL_BUFF_SIZE = 50;
const int NUM_BATTERIES = 18;

// Conversion constants - used to convert ADC values to voltages
const float V_REF = 5.06;
const float R_EQ = 253.08;
const float R_MEAS = 4.68;
const float STEP_SIZE = (2 * V_REF) / 8192;
const float R_MULTIPLIER = R_EQ / R_MEAS;
const float V_MULTIPLIER = R_MULTIPLIER * STEP_SIZE;

// Pin constants
//  value is Arduino GPIO pin number and comment is board pin label
const int ADC_DISABLE = 6;    //D7

const int DEMUX1_ENABLE = 12; //D6
const int DEMUX1_A = 8;       //B4
const int DEMUX1_B = 9;       //B5
const int DEMUX1_C = 10;      //B6

const int DEMUX2_ENABLE = 11; //B7
const int DEMUX2_A = 19;      //F6
const int DEMUX2_B = 20;      //F5
const int DEMUX2_C = 21;      //F4

const int DEMUX3_ENABLE = 4;  //D4
const int DEMUX3_A = 1;       //D3
const int DEMUX3_B = 0;       //D2
const int DEMUX3_C = 2;       //D1

// Setup the various components - this is run once at startup
void setup()
{
  Serial.begin(9600);

  // start the SPI library:
  SPI.begin();

  // Initialize ADC
  pinMode     (ADC_DISABLE, OUTPUT);
  digitalWrite(ADC_DISABLE, HIGH);

  // Initialize Demux 1
  pinMode     (DEMUX1_ENABLE, OUTPUT);
  digitalWrite(DEMUX1_ENABLE, LOW);

  pinMode     (DEMUX1_A, OUTPUT);
  digitalWrite(DEMUX1_A, LOW);

  pinMode     (DEMUX1_B, OUTPUT);
  digitalWrite(DEMUX1_B, LOW);

  pinMode     (DEMUX1_C, OUTPUT);
```

```
  digitalWrite(DEMUX1_B, LOW);

  // Initialize Demux 2
  pinMode     (DEMUX2_ENABLE, OUTPUT);
  digitalWrite(DEMUX2_ENABLE, LOW);

  pinMode     (DEMUX2_A, OUTPUT);
  digitalWrite(DEMUX2_A, LOW);

  pinMode     (DEMUX2_B, OUTPUT);
  digitalWrite(DEMUX2_B, LOW);

  pinMode     (DEMUX2_C, OUTPUT);
  digitalWrite(DEMUX2_C, LOW);

  // Initialize Demux 3
  pinMode     (DEMUX3_ENABLE, OUTPUT);
  digitalWrite(DEMUX3_ENABLE, LOW);

  pinMode     (DEMUX3_A, OUTPUT);
  digitalWrite(DEMUX3_A, LOW);

  pinMode     (DEMUX3_B, OUTPUT);
  digitalWrite(DEMUX3_B, LOW);

  pinMode     (DEMUX3_C, OUTPUT);
  digitalWrite(DEMUX3_C, LOW);
}

// Take measuremnts and output results - this is run in a continuous loop
void loop()
{
  unsigned int ADCResults[NUM_BATTERIES]; // Values read from ADC
  char value[ADC_VAL_BUFF_SIZE];    // Used to store string version of voltage
  char battery[ADC_VAL_BUFF_SIZE];  // Used to store string version of battery #
  char raw[ADC_VAL_BUFF_SIZE];      // Used to store string version of ADC value
  float voltage = 0;

  // Take measurements
  for (int i = 0; i < NUM_BATTERIES; i++)
  {
    ADCResults[i] = measureBattery(i);
    delay(100);
  }

  // Output results
  for (int i = 0; i < NUM_BATTERIES; i++)
  {
    if (i == 0) // first battery's voltage is already with respect to ground
      voltage = ADCResults[i] * V_MULTIPLIER;
    else // subtract previous battery's voltage for single battery voltage
      voltage = (ADCResults[i] - ADCResults[i - 1]) *
       V_MULTIPLIER;
    Serial.write("Voltage ");
    String(i).toCharArray(battery, ADC_VAL_BUFF_SIZE);      // Convert to string
    Serial.write(battery);
    Serial.write(": ");
    String(voltage).toCharArray(value, ADC_VAL_BUFF_SIZE);  // Convert to string
    Serial.write(value);
    Serial.write(" V\n");
  }
  Serial.write("\n");
```

```
  delay(1000);
}

// Take an ADC sample, and read it from SPI, returns battery voltage as read by
//  the ADC
unsigned int takeADCSample()
{
  unsigned int result = 0;

  SPI.beginTransaction(SPISettings(1700000, MSBFIRST,
   SPI_MODE0));                    // Initialize SPI at 1.7 MHz
  delay(100);
  digitalWrite(ADC_DISABLE, LOW);   // Turn the ADC on
  delay(100);
  // Transfer 16 bytes (begins measurement and receives data)
  result = SPI.transfer16(0);
  delay(100);
  SPI.endTransaction();
  digitalWrite(ADC_DISABLE, HIGH);  // turn the ADC off

  return result + 11;               // compensate for inherent offset in ADC
}

// Measure the specified battery's voltage - accepts parameter for which battery
//       enables (only) the correct optocoupler, takes a measurement on the ADC, and
//  returns the value measured by the ADC
unsigned int measureBattery(int battery)
{
  unsigned int ADC_Value;

  // Turn off all optocouplers (should already be off)
  digitalWrite(DEMUX1_ENABLE, LOW);
  digitalWrite(DEMUX1_A, LOW);
  digitalWrite(DEMUX1_B, LOW);
  digitalWrite(DEMUX1_C, LOW);
  digitalWrite(DEMUX2_ENABLE, LOW);
  digitalWrite(DEMUX2_A, LOW);
  digitalWrite(DEMUX2_B, LOW);
  digitalWrite(DEMUX2_C, LOW);
  digitalWrite(DEMUX3_ENABLE, LOW);
  digitalWrite(DEMUX3_A, LOW);
  digitalWrite(DEMUX3_B, LOW);
  digitalWrite(DEMUX3_C, LOW);

  // figure out which battery to turn on, and turn it on
  switch(battery)
  {
    case 0:
      digitalWrite(DEMUX1_ENABLE, HIGH);
      break;

    case 1:
      digitalWrite(DEMUX1_ENABLE, HIGH);
      digitalWrite(DEMUX1_A, HIGH);
      break;

    case 2:
      digitalWrite(DEMUX1_ENABLE, HIGH);
      digitalWrite(DEMUX1_B, HIGH);
      break;
```

```
case 3:
  digitalWrite(DEMUX1_ENABLE, HIGH);
  digitalWrite(DEMUX1_B, HIGH);
  digitalWrite(DEMUX1_A, HIGH);
  break;

case 4:
  digitalWrite(DEMUX1_ENABLE, HIGH);
  digitalWrite(DEMUX1_C, HIGH);
  break;

case 5:
  digitalWrite(DEMUX1_ENABLE, HIGH);
  digitalWrite(DEMUX1_C, HIGH);
  digitalWrite(DEMUX1_A, HIGH);
  break;

case 6:
  digitalWrite(DEMUX2_ENABLE, HIGH);
  break;

case 7:
  digitalWrite(DEMUX2_ENABLE, HIGH);
  digitalWrite(DEMUX2_A, HIGH);
  break;

case 8:
  digitalWrite(DEMUX2_ENABLE, HIGH);
  digitalWrite(DEMUX2_B, HIGH);
  break;

case 9:
  digitalWrite(DEMUX2_ENABLE, HIGH);
  digitalWrite(DEMUX2_B, HIGH);
  digitalWrite(DEMUX2_A, HIGH);
  break;

case 10:
  digitalWrite(DEMUX2_ENABLE, HIGH);
  digitalWrite(DEMUX2_C, HIGH);
  break;

case 11:
  digitalWrite(DEMUX2_ENABLE, HIGH);
  digitalWrite(DEMUX2_C, HIGH);
  digitalWrite(DEMUX2_A, HIGH);
  break;

case 12:
  digitalWrite(DEMUX3_ENABLE, HIGH);
  break;

case 13:
  digitalWrite(DEMUX3_ENABLE, HIGH);
  digitalWrite(DEMUX3_A, HIGH);
  break;

case 14:
  digitalWrite(DEMUX3_ENABLE, HIGH);
  digitalWrite(DEMUX3_B, HIGH);
  break;
```

```
    case 15:
      digitalWrite(DEMUX3_ENABLE, HIGH);
      digitalWrite(DEMUX3_B, HIGH);
      digitalWrite(DEMUX3_A, HIGH);
      break;

    case 16:
      digitalWrite(DEMUX3_ENABLE, HIGH);
      digitalWrite(DEMUX3_C, HIGH);
      break;

    case 17:
      digitalWrite(DEMUX3_ENABLE, HIGH);
      digitalWrite(DEMUX3_C, HIGH);
      digitalWrite(DEMUX3_A, HIGH);
      break;

        // If bad value passed in, turn off all optocouplers (should already be off)
    default:
      digitalWrite(DEMUX1_ENABLE, LOW);
      digitalWrite(DEMUX1_A, LOW);
      digitalWrite(DEMUX1_B, LOW);
      digitalWrite(DEMUX1_C, LOW);
      digitalWrite(DEMUX2_ENABLE, LOW);
      digitalWrite(DEMUX2_A, LOW);
      digitalWrite(DEMUX2_B, LOW);
      digitalWrite(DEMUX2_C, LOW);
      digitalWrite(DEMUX3_ENABLE, LOW);
      digitalWrite(DEMUX3_A, LOW);
      digitalWrite(DEMUX3_B, LOW);
      digitalWrite(DEMUX3_C, LOW);
      break;
  }

  delay(100);

  ADC_Value = takeADCSample(); // Take measurement on battery

  // Turn the optocoupler back off
  digitalWrite(DEMUX1_ENABLE, LOW);
  digitalWrite(DEMUX1_A, LOW);
  digitalWrite(DEMUX1_B, LOW);
  digitalWrite(DEMUX1_C, LOW);
  digitalWrite(DEMUX2_ENABLE, LOW);
  digitalWrite(DEMUX2_A, LOW);
  digitalWrite(DEMUX2_B, LOW);
  digitalWrite(DEMUX2_C, LOW);
  digitalWrite(DEMUX3_ENABLE, LOW);
  digitalWrite(DEMUX3_A, LOW);
  digitalWrite(DEMUX3_B, LOW);
  digitalWrite(DEMUX3_C, LOW);

  return ADC_Value;
}
```