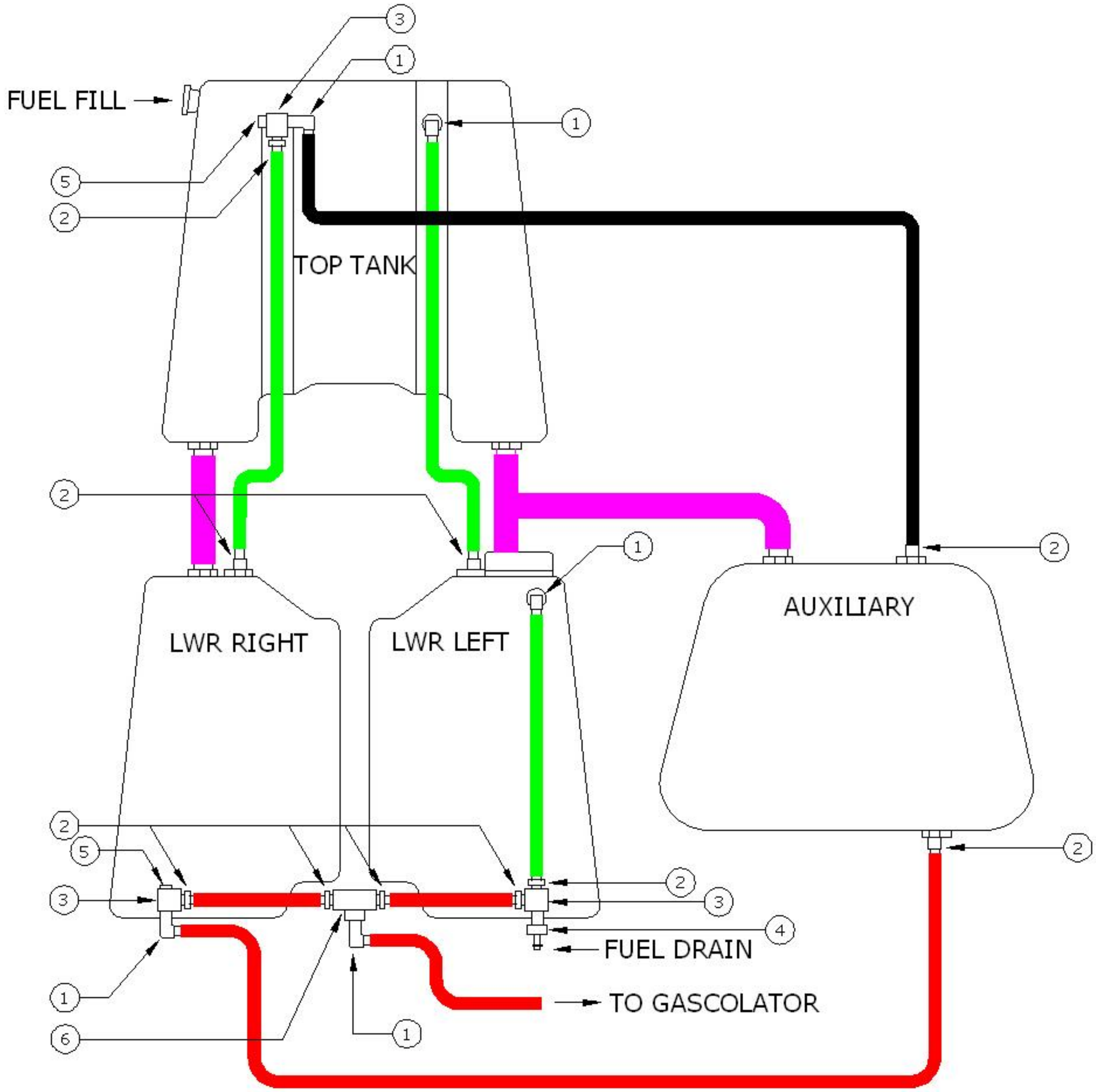
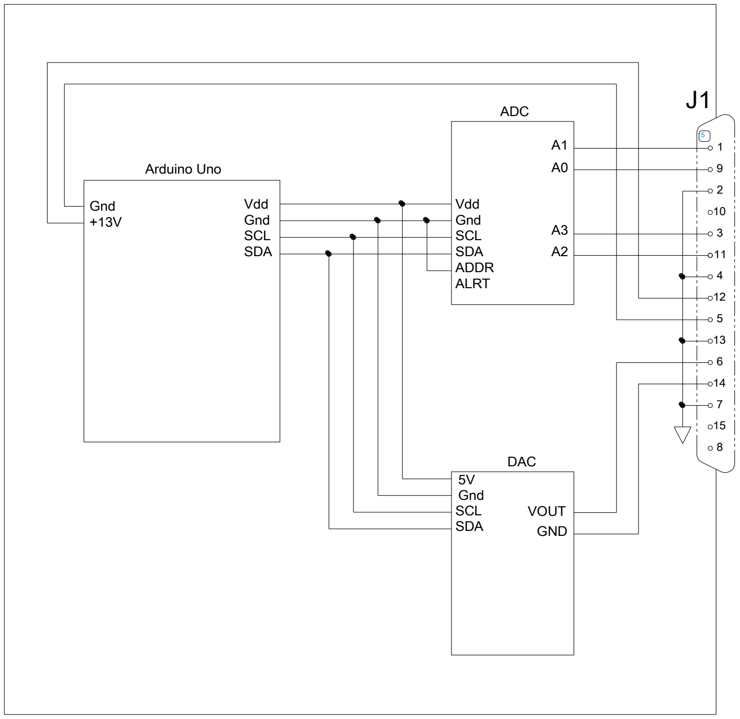
JUAN RIVERA HELICYCLE BUILDER LOG #2
FUEL QUANTITY TOTALIZER

19 October, 2014 -- There's an annoying deficiency in the stock fuel quantity indicator. It's driven by a capacitive probe in the lower/left tank, so the indicator reads "full" until the fuel is completely drained from the upper tank. In other words, your fuel is one third gone before the indicator comes off the "full" peg. That's a deficiency that is crying out for improvement, and I intend to do just that. Here's a schematic of the Helicycle fuel tanks. It includes the auxiliary tank that can be added if the pilot is not too heavy.



You can see that the probe in the lower/left tank will measure the fuel in all three of the lower tanks since they are all interconnected and they're all at the same level.

I'm going to install a second fuel probe in the upper tank and I've create a digital fuel totalizer that will take inputs from both probes and drive a 270-degree sweep UMA indicator. Since the processing will be done in the digital domain so it is going to be easy to compensate each probe for the geometry of the tanks[1]. I can also easily account for the presence or absence of an auxiliary tank with a simple jumper setting on my circuit board. The result will be a very accurate fuel quantity indicator for the first time.

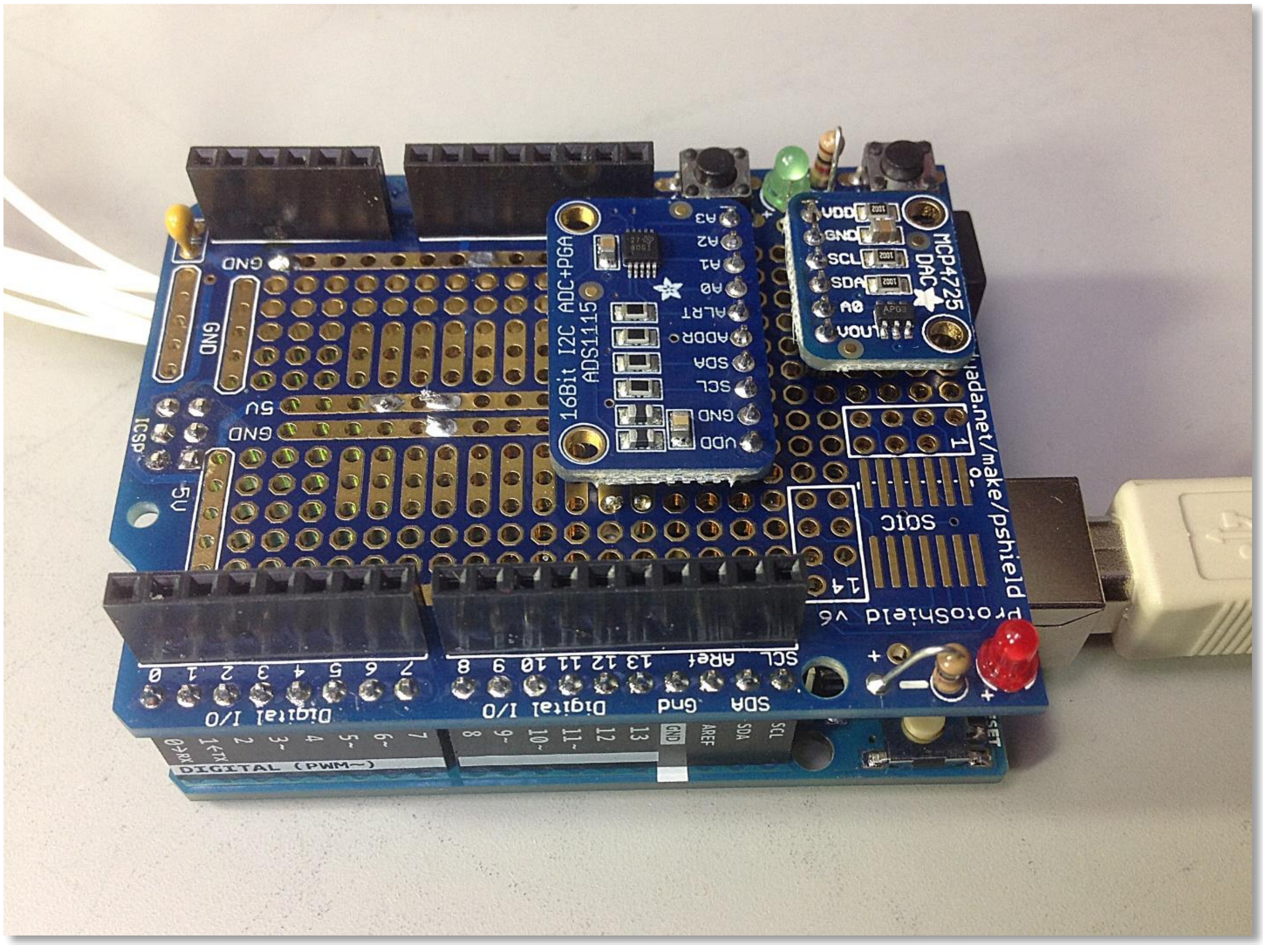


Here's a schematic of the hardware. I'm using a very inexpensive general purpose digital processor board called Arduino Uno (it's Italian.) Attached to the Uno board, on a daughter board, are a 16-bit analog to digital converter (ADC) with two differential input channels and a 12-bit digital to analog converter (DAC.)

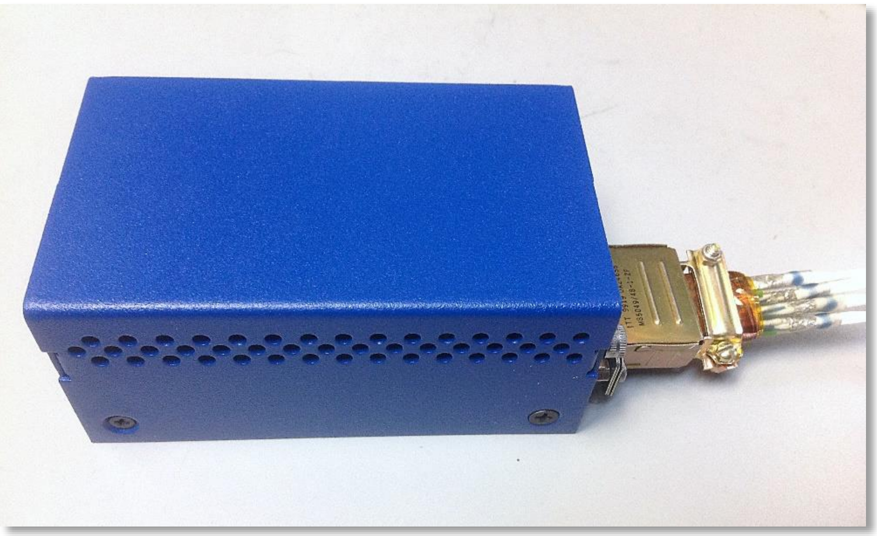
The analog voltages from the two probes are converted to digital values using the ADC and the calculated total is converted from digital back to an analog voltage to drive the indicator using the DAC.

Differential inputs look at the difference between the signal + and the signal – inputs and ignore their relationship to ground. This eliminates a big source of noise. All the cables to and from the totalizer will be shielded of course.

[1] Just because the fuel probe is half submerged doesn't mean that the tank is half full because of the complex geometry of the tanks. This error can be compensated for by using a lookup table or a polynomial equation. Once I get my tanks installed I'll collect data on the probe output voltages as I add measured amounts of fuel. Once I have the raw data I can decide how best to process it if needed. This part of the project will be fun.



Here's the Arduino board with the daughter board stacked on top. The two small boards mounted on the daughter board are the DAC and the ADC. The Arduino is programmed over the USB port that you see in the lower right. Once the program is loaded there is no need for the USB connection since the program code will reside in non-volatile memory on the Arduino.



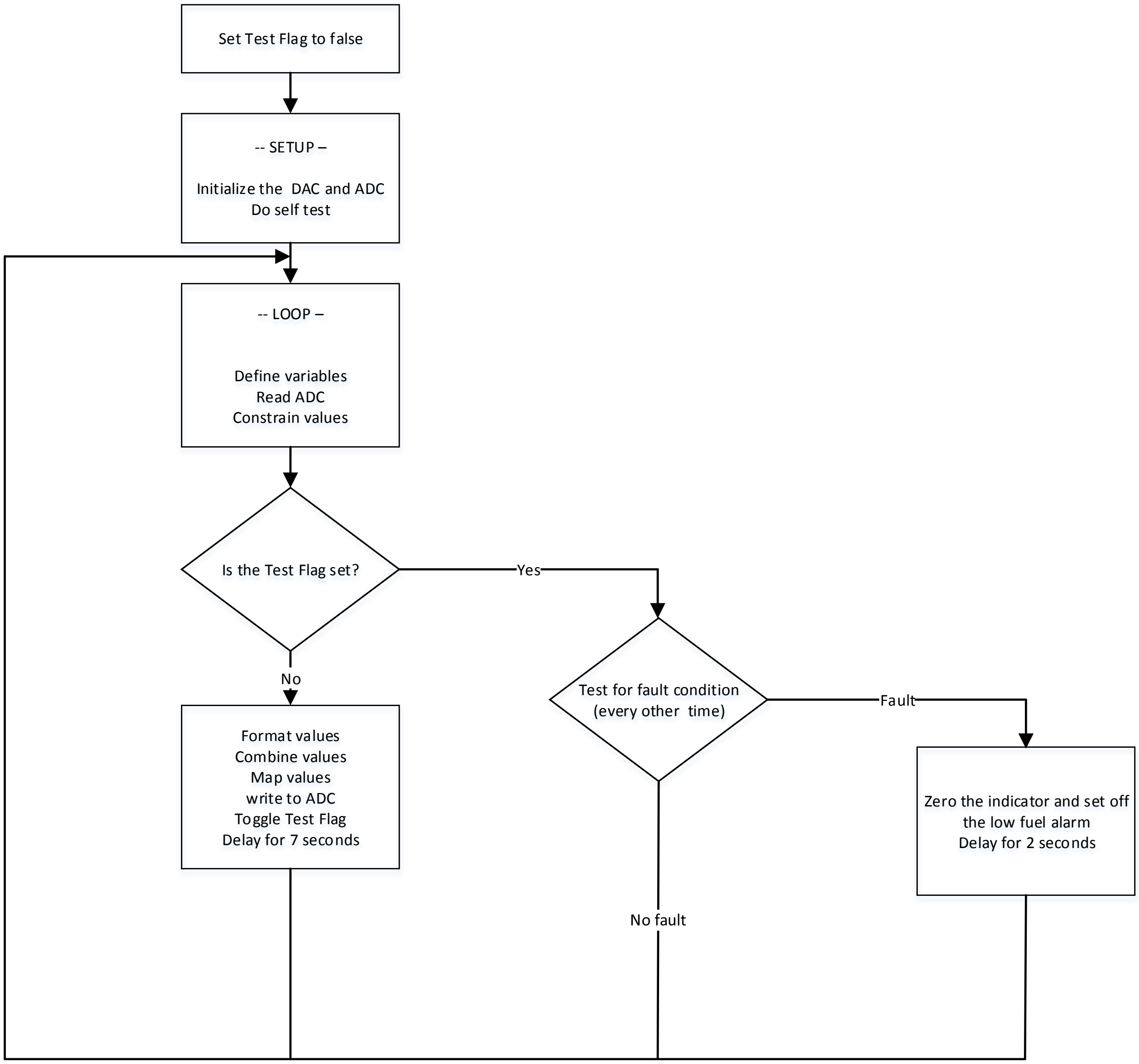
Here's the circuitry all packed in its enclosure with the interface cable attached, and ready to install.

This particular enclosure is made of steel, but I'll need some ballast in the front anyway, so this will be fine.

All of the Arduino products are open-source, meaning that everything about them is published and you are free to use it in any way you please. If I was to manufacture this circuit in quantities I could recreate the entire circuit in a single board and eliminate any parts that are not used in this application. Another benefit of this open-source Arduino family of products is that everything you see only cost about fifty dollars and it only took me a few hours to cobble a basic program together and debug it.

I now am able to input two separate 0-5V inputs that represent the outputs from the two fuel probes, weight the values to account for the capacity of the tanks, and drive my UMA fuel quantity indicator. It works perfectly – full scale on the bottom probe drives the indicator to two thirds full, and full scale on both inputs drives the indicator to full scale.

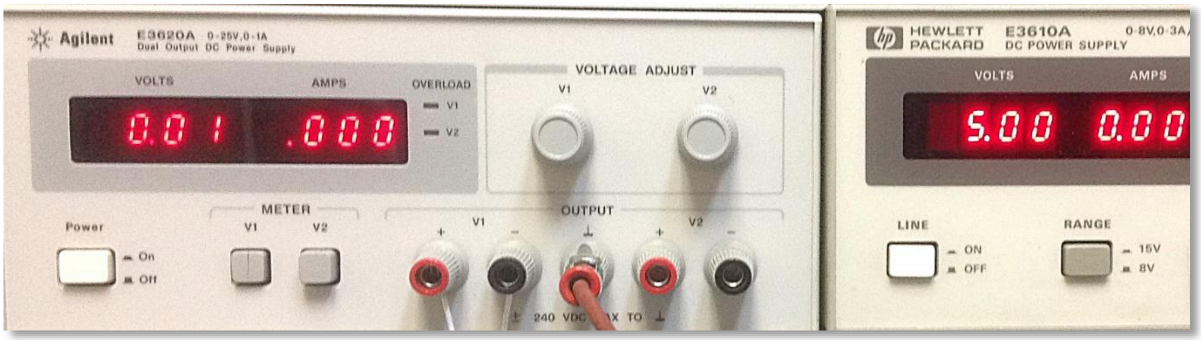
Now that I'm done with the development I'll find a nice box to mount the boards in and install a DB-15 connector. I've already checked the hardware for EMI in the aircraft VHF band and there is none to be seen. This is important since this will be installed in the instrument pod, near the radio.



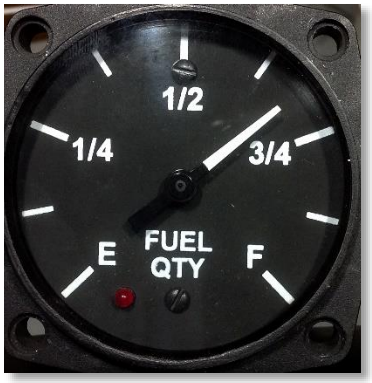
I'll describe a few areas of the program:

Self-Test – When power is applied the processor does a self-test by swinging the fuel quantity indicator from zero to full scale and then back to zero. This tests that the processor is able to execute the code, that the wiring to the indicator is intact, and that the indicator and low fuel alarm are both functional.

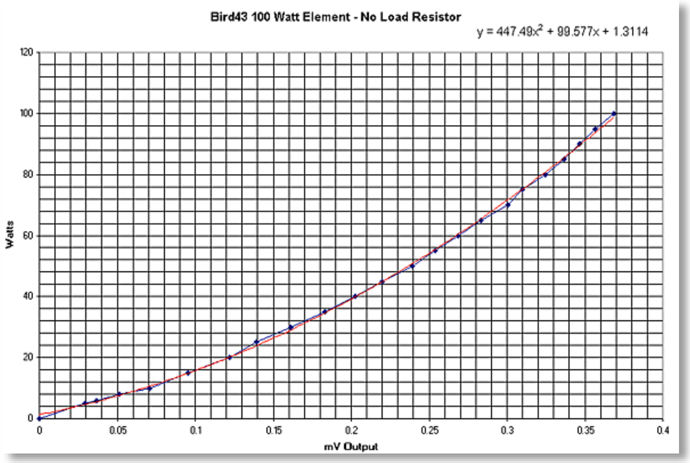
Probe Fault Check -- During operation I do a check to see if there may be a problem with the data from the probes. If the bottom tanks are not full, and there is any fuel in the top tank, I call that an error condition since it should never occur. If that condition exists I zero the indicator for 2 seconds and set off the low fuel alarm. Then the indicator will indicate fuel level for 7 seconds. This cycle repeats continuously as long as the fault condition exists. This way the pilot will be alerted to the possibility of an error in the fuel quantity reading but still be able to see what it says and decide whether to trust it or not. Between this repeating check and the initial self-test, this totalizer should seldom fool the pilot with unknown erroneous readings.



Here's my test setup – The left power supply represents the upper tank and its set to 0.0 volts which represents an empty tank. The right power supply is set to 5.00 volts, or a full tank.



And here's the UMA fuel quantity indicator indicating2/3 full. Once I have a working fuel system I'll measure the output of each probe as fuel is added in carefully measured increments. When I have that data I can tweak the code to account for the irregular shape and exact capacity of each tank. When I get to that part of the project the ability of the Arduino board to do complex math will make the choice of a digital approach obvious.



Here's an example of curve matching from a previous project I did. I used a third-order polynomial equation to match the non-linear curve of a wattmeter. You can see the equation in the upper right of the graph. The black curve is the actual measured output and the red curve was created simply by using that equation. I think this is a more elegant method than using a look-up table, so I'll try this first. Using this method, each probe will have its own equation and it can be implemented in about four lines of code.

Here's the current source code (anything in green is a comment and not part of the compiled code):

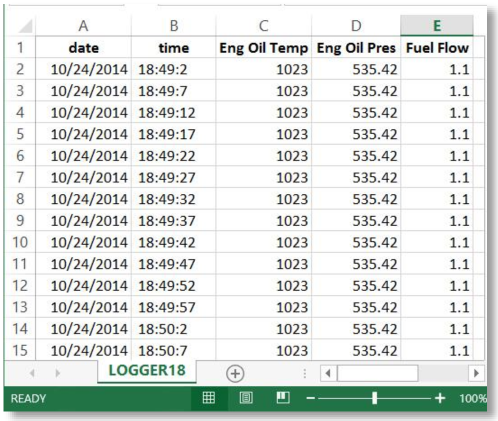
```
// 19 Oct 2014
// Written by Juan Rivera
//                                       HELICYCLE FUEL TOTALIZER
// The totalizer uses two capacitive fuel probes - one in the top tank and one in the lower/left tank.
// The raw outputs of the two probes are acquired using a dual-channel 16-bit differential
// analog-to-digital converter.  The two values are weighted based on the fuel capacity of the tanks,
// combined and used to drive a UMA fuel quantity indicator using a 12-bit digital-to analog converter.
// If fuel remains in the upper tank and the lower tank is not full (this should never happen) then the
// indicator will dip to zero every seven seconds and trigger an alarm condition to indicate a fault

// LINK TO LIBRARIES
#include <Wire.h>
#include <Adafruit_ADS1015.h>
Adafruit_ADS1115 ads;
#include <Adafruit_MCP4725.h>
Adafruit_MCP4725 dac;

boolean test=false; // Initialize the test flag

void setup(void)
{
        // ADC Range: +/- 6.144V (1 bit = 0.1875mV (Default setting)
        // Probe-1 (top tank):    Ain0=Sig, Ain1=Com
        // Probe-2 (bottom tank): Ain2=Sig, Ain3=Com

        // START THE ADC AND DAC
        ads.begin();
        dac.begin(0x62);

        // DO A SELF TEST OF THE PROCESSOR, DAC, UMA INDICATOR, AND THE LOW FUEL ALARM
        dac.setVoltage(400, false);
        delay(10000); // Output 0.500 Volts for 10 seconds (Zero)
        dac.setVoltage(3665, false);
        delay(10000); //Output 4.500 Volts for 10 seconds (Full Scale)
        dac.setVoltage(400, false);
        delay(5000); // Output 0.500 Volts for 5 seconds (Zero)
}
void loop(void)
{
        // DECLARE VARIABLES
        int16_t P1; // Probe-1 raw data
        int16_t P2; // Probe-2 raw data
        float p1;   // Convert to Floating point
        float p2;   // Convert to floating point
        float Total;//P1 and P2 combined
        int total;  // Convert to floating point

        // READ THE TWO DIFFERENTIAL ADC CHANNELS
        P1 = ads.readADC_Differential_0_1();  // The difference between pins Ain0 and Ain1
        P2 = ads.readADC_Differential_2_3();  // The difference between pins Ain2 and Ain3

        // CONSTRAIN VALUES TO BETWEEN 0.0 AND 5.0 VOLTS
        p1=constrain(P1,0,26667); // Limit p1 max value to 5.000 Volts
        p2=constrain(P2,0,26667); // Same for p2

        // The test flag is toggled at the end of the loop so it will be true every other pass,
        // causing the fault loop to be executed every other pass if there is a problem.
        if (test==true) // Check the test flag
        {
                // The test flag is set, so check to see if a fault exists
                if (p1>1000 and p2 < 26000) // Upper tank still have fuel yet lower tanks are not full?
                {
                        // Yes - we have a problem
                        dac.setVoltage (400, false); // Zero the indicator and set off the alarm
                        delay(2500); // Allow the UMA to trigger a low fuel alarm
                }
        }
        else // If the fault flag is not set then execute the reset of this loop.
            // There may be a fault but we'll catch it next time around.
        {
                test=!test; // Toggle the Test Flag
        }
        // The fuel system on my Helicycle consists of four tanks.  The upper tank has
        // a capacity of 7 gallons and is measured by Probe-1.  The three lower tanks
        // have a combined capacity of 14 gallons and they are measured by
        // Probe-2, so the outputs of the probes need to be weighed accordingly.

        p2=2*p2; // The bottom tanks contain twice as much fuel as the top tank

        // COMBINE BOTH TANKS, SCALE OUTPUT AND CONSTRAIN TO VALUES BETWEEN 0.5V AND 4.5V
        // Combining p1 and two times p2 will result in a value of 80,000 with full fuel
        // and o when empty.  The map function will translate that range to values between
        // 400 and 3665 for the digital-to analog converter and result in a voltage of
        // between 0.5 and 4.5 volts, which is what my UMA indicator expects for minimum
        // and maximum values.
        Total = map((p1+p2),0,80000,400,3665); // 3665 sets full scale for my indicator

        // CONVERT TO INTEGER AND SEND TO DAC
        total=Total;
        dac.setVoltage (total, false); // Write to DAC
        delay(7000); // Delay for 7 seconds and then repeat the loop
}
```

If you are interested in duplicating this project or learning more, feel free to contact me at Juan(at sign)Helicycles.Org. I'll be happy to send you a bill of materials and this source code.

I'll add more to this once I have my tanks installed…

---

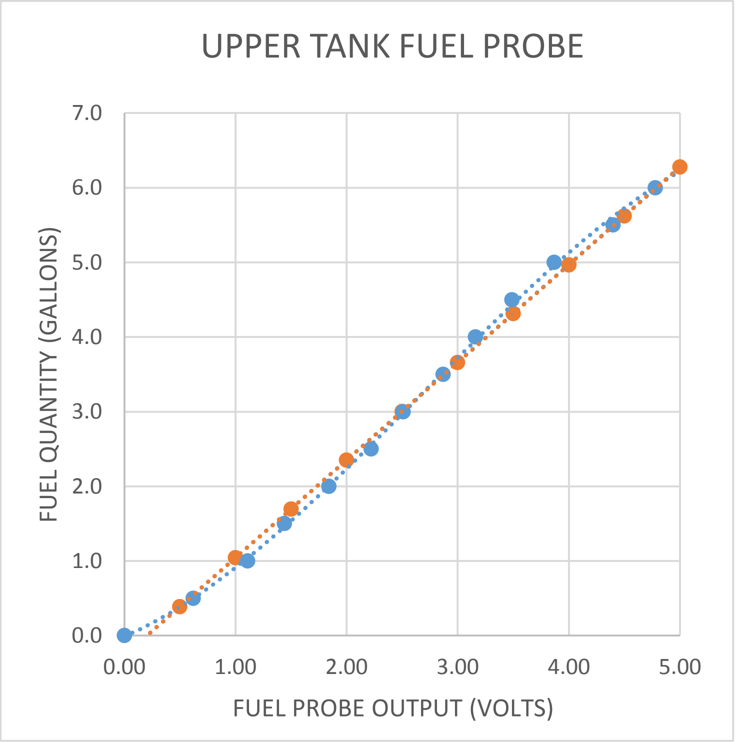
24 October – I liked the Arduino board so much that I picked up another one with a data logging and real-time clock on one inexpensive daughter board. The clock time-stamps each data frame with the date and time and runs off a small internal battery. The data is written to an SD memory card that can be removed and inserted into your card reader on your PC. The data is formatted for Excel and can be easily plotted. The Arduino Uno itself can accept several single-ended analog inputs, or I can add a few more of the 12-bit ADC modules that can each accept two differential analog inputs.



	A	B	C	D	E
1	date	time	Eng Oil Temp	Eng Oil Pres	Fuel Flow
2	10/24/2014	18:49:2	1023	535.42	1.1
3	10/24/2014	18:49:7	1023	535.42	1.1
4	10/24/2014	18:49:12	1023	535.42	1.1
5	10/24/2014	18:49:17	1023	535.42	1.1
6	10/24/2014	18:49:22	1023	535.42	1.1
7	10/24/2014	18:49:27	1023	535.42	1.1
8	10/24/2014	18:49:32	1023	535.42	1.1
9	10/24/2014	18:49:37	1023	535.42	1.1
10	10/24/2014	18:49:42	1023	535.42	1.1
11	10/24/2014	18:49:47	1023	535.42	1.1
12	10/24/2014	18:49:52	1023	535.42	1.1
13	10/24/2014	18:49:57	1023	535.42	1.1
14	10/24/2014	18:50:2	1023	535.42	1.1
15	10/24/2014	18:50:7	1023	535.42	1.1

I'm thinking of tapping into my interface chassis and bridging all of my sensors to one connector where I can attach this logger. This is another very low cost project. I'm into it for about $50 and have the unit running on code I downloaded and modified in about half an hour. Here's a sample run in Excel:

You can see the date/time stamp for each entry (I have it taking samples every five seconds but the sample rate can be much faster or much slower as needed. To the right are three column of dummy data that can be anything I want that can be converted into a voltage. And I can always add more channels…



8 February, 2015 - I now have my top tank mounted so I'm now able to characterize the probe's output curve (left.) Considering the shape of the tank, and the slope of the probe, it's a lot more linear than I expected. First a word about calibration – these probes can (and probably should) be calibrated for Jet-A. I think the factory does a calibration using aviation gas, but Jet-A is going to cause an error. The procedure is simple and covered in the data sheet that comes with the probe, so I won't cover it here – you set the empty and full output levels. Anyway, the graph shows Upper Tank probe output voltage on the horizontal axis (X) and fuel quantity on the vertical axis (Y.) To arrive at the math equation I need I measured the probe output voltage every half gallon from empty to full and plotted it into Excel (red line.) Then I had Excel create a linear trend line and cough up the formula it used. The formula was Y = (1.3084 * X) - .2667. I used that formula to plot the fuel quantity based on the probe voltage (blue line.) That will get me way more accuracy than I need. (Red is measured results and blue is calculated using that formula.)

My next step is to install my remaining tanks and do the same exercise for the lower tanks and probe. Once that's done I can tweak the math in the Arduino fuel quantity totalizer and I'll have an extremely accurate fuel quantity indicator. Since I don't yet have an aux tank that may be months away.

I'll add more as I get back to this project…