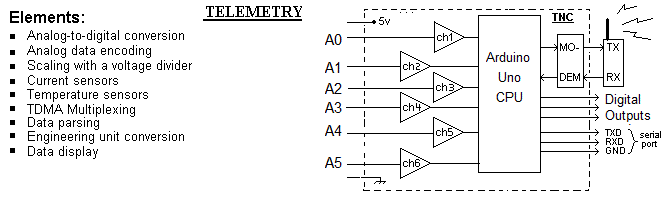
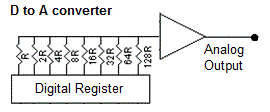
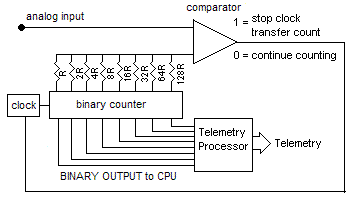
**EA-467 Telemetry Introduction (I)** (rev-d) Fall 2017

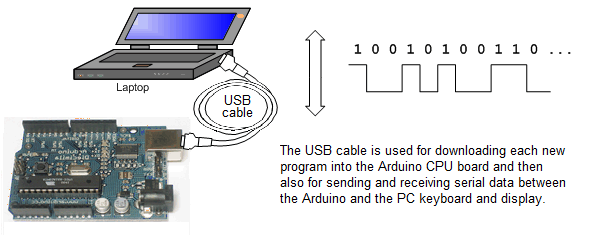


**Introduction:** A fundamental aspect of all spacecraft is telemetry communications from the spacecraft to the ground station. Telemetry’s function is to provide information on the state of every important system, parameter, voltage, current, and temperature onboard the satellite. To get this analog information into digital form, a fundamental component called an Analog to Digital converter (ADC or A/D converter) is used.

**DAC:** To understand an ADC you first have to understand the opposite, the digital-to-analog converter (DAC or D/A converter). The DAC circuit uses a simple resistor on each bit of a data register which is double the resistance of the one before it. This way a DAC can output an analog value from a digital value (stored in a register) in a very fast single CPU cycle.

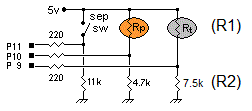
**ADC:** But in an A/D converter there is no simple circuit to get from Analog to Digital. Instead, all A/D converters are based on some form of sample-and-compare circuit. This circuit counts up a binary value in a register driving a DAC until the analog output of the DAC equals the unknown analog input as measured by a comparator as shown here. When equal, the comparator stops the counter and the count register is read. That is why A/D values are often called ***counts***.

It is important to note that the speed of the ADC is limited by the speed of the internal counter which has to go through hundreds if not thousands of cycles for each conversion. Thus the ADC is what limits the speed of A/D conversion and not the speed of the system processor.

**Arduino ADC’s:** This lab uses the Arduino’s internal ADCs to sample telemetry values, convert to digital, and multiplex them into a serial stream for transmission on a TDMA FSK channel. At the ground station, the telemetry counts are converted back to digital values, and then entered into “telemetry equations” for display of the original analog values and units. A typical simple telemetry format used on prior USNA satellite missions is shown here.

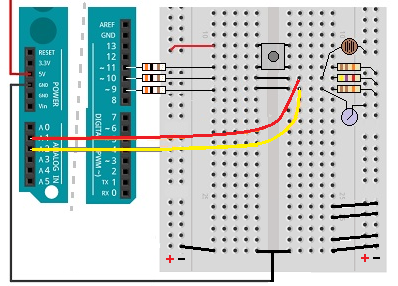
**Telemetry Format: LABSAT-X>ALL:T#002,132,138,159,131,213,11111111**

Each telemetry packet has a header consisting of the satellite callsign (LABSAT-X), and destination address (ALL). The data starts with a T#SSS packet serial number, then 5 comma-separated analog counts, and finally 8 discrete bit values. Each of the 5 A/D channels is encoded to 10 bit resolution, giving a count value from 0 to 1024.

****

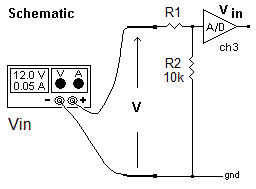
**Previous C&DH LAB Inputs:** The schematic to the right in the ***previous*** *C&DH lab* sensed the value of the photoresistor (Rp) and thermistor (Rt) connected as R1 in an R1/R2 voltage divider between 5v and ground. If the device had a low resistance, it pulled the input pin high. If it had a high resistance, R2 pulled the input low. The 220 ohm resistors were protection from shorts on the CPU pins and the 4.7k and 7.5k values of the R2’s were chosen to give a good threshold voltage between 5v and 0 over the useful range of the sensor but only to a resolution of one bit, either a 1 or 0, depending on whether the value on I/O pins P9 to P11 appeared high or low.

In this lab, instead, you will use A/D inputs to read these sensor inputs to 10 bit resolution. *Leave unused parts on the board so they don’t get lost, but trace all connections to be sure you know what is going on and now things are connected.*

**Part A. Light Level & Temperature:** The Arduino has 6 A/D converters labeled A0 to A5 that can resolve analog inputs to 10 bit resolution. We will use A1 to A5 for this experiment. Leave the 220 ohm resistors connected to the digital I/O pins from the last lab and the 2 short wire jumpers in the middle of the board and run the Red A1 analog input and Yellow A2 input to these two sensors on the Arduino as shown. Keep the existing 4.7K and 7.5k values for the R2 pull-down resistors.

The thermistor has a negative temperature coefficient so as temperatures go up, the resistance goes down. These are designed to be 10k at 25C. So a 10k value for R2 would center the A/D range on 25C. Using the 7.5k value of R2 from the last lab will center the range for higher temperatures. The switch remains on digital I/O Pin 11 since it is still just a “1/0” type of sensor.

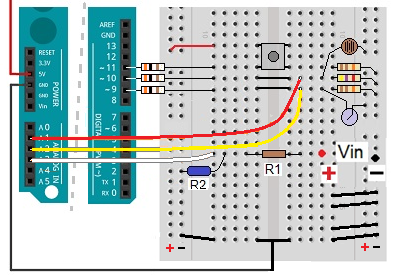
##### Part B: Voltage Divider

The 6 Analog inputs on the Arduino convert any input voltage from 0 to 5 volts into a 10 bit count from 0 to 1024 resolution. Your code takes this count and puts it together with four other telemetry channels into a single data packet for transmission. On receipt, the telemetry count, X, is converted back to original engineering units for display using a telemetry equation. A linear equation for example would yield a voltage measurement V from count X using the equation V = A\*X + B.

For any voltage input over 5 volts, you first have to scale input voltages into the 0-5v range using a voltage divider. The voltage divider is a pair of resistors R1 and R2 that give an output voltage VAD into the ADC that is less than the input voltage Vin by a ratio determined by the two resistors by the following equation:

VAD=Vin ( R2 /( R1+R2 ))

and for the ADC, the relationship:

Count/1024 = VAD /5v

Design a voltage divider to give the convenient voltage range of 0.0 to 102.4 volts over the full ADC count range of 0 to 1024 with a precision of 0.1 volt. For example, this will give a count of 120 for a voltage input (Vin) of 12.0 volts. To make that scaling, we know that 0-5v into the ADC results in a 0-1024 count at the output, calculate and record the VAD voltage input to the ADC that is necessary to give a count of 120.

Next, use the voltage divider rule with R2 = 10k to solve for R1 that produces that VAD voltage with a 12.0 V input and record the value. Verify your design by completing the following steps:

1. Install the 1% precision 10.0k resistor for R2 (dark blue) and calculated R1 (brown) on your board as shown. Note, 1% resistors do not use color codes but are printed with 4 digits, a 3 digit mantissa and a 1 digit decade multiplier. So 10.0K = 1002.

2. Connect the white wire A3 analog input to the R1/R2 voltage divider as shown above.

##### Part C: Measuring Current Telemetry

##### This section will add a current sensor to give an output count equal to the current in mA. We can measure current with an ADC as simply the voltage drop across a resistor that carries the current to be measured, as shown here. The voltage across that resistor is proportional to Iload x Rdrop according to Ohm’s law. In this circuit you will add Rdrop in series with the load resistor to report the load current on channel 4.

Since a count of 1024 results from an ADC input of 5 volts, calculate and record the value of R**drop** to give 5 volts with 1024 mA current flowing through it?

1. Ask the instructor for a resistor equal to your R**drop** and install it as shown.

2. Connect the large load resistor as shown and connect the junction with R**drop** to the Brown Analog input A4.

**Part D: ARDUINO CODE:** Write a program to read in the value of these four analog inputs and then output the values to the display terminal. First, declare all the required variables. Initialize integer variables for each one of your sensors. For example, initialize a variable “RpValue”, which will be used to accept the sensor value on the photoresistor:

int RpValue = 0;

Second, in your setup() function ensure you initialize your serial monitor at 9600 (just like the C&DH lab with the Serial.begin(9600); statement. Then use pinMode(xx,INPUT) function to define the pins you are using to read your sensor values as inputs. For example, to define your analog input on pin A1 (for the photoresistor as in the above schematic) then the statement reads:

pinMode(A1,INPUT);

Third, in your loop() function read in the sensor values using the analogRead() function. For example, to read the value on the photoresistor as “RpValue” use:

RpValue = analogRead(A1);

Add the statement delay(5) after each analogRead statement for pins A1 to A5 to account for the delay in the Analog to Digital Converter (The Arduino takes about one millisecond for this process, so 5 milliseconds ensures time to complete).

Repeat the above process for your thermistor on pin A2 (RtValue), Voltage on pin A3 (VinVal) and Current on pin A4 (IinVal), and a spare on pin A5 (V5Val).

Finally, output the values for your 5 A/D inputs using the Serial.print() function. Note that you will have to print commas “,” to separate the data in your telemetry stream or the numbers will flow together and be useless. Also, add a delay(1000) statement at the end of your loop to cause sensor readings to be taken and reported once per second. Test your code and make sure that you get five comma separated telemetry counts every second showing the telemetry values for the photoresistor, thermistor, Vin, Iin and V5:

**Part E. Telemetry Downlink Format**

The example telemetry format shown in the introduction section is used in many hand Radio and Automatic Packet Reporting System (APRS) telemetry applications and consists of 5 analog channels and then also the I/O state of 8 bits. Your goal now is to program your CPU to replicate the above telemetry format inserting the values of the 5 Analog channels and the first of the 8 status bits showing the state of the push button.

**Packet Callsign Header:** To complete the full format of the telemetry package, use a Serial.print statement to add a header consisting of the FROM and TO fields similar to the telemetry format shown in the introduction section. Here is an example: “TLMSAT-X>ALL:T#” (X is your particular Arduino CPU number, assigned by your seat). Add this to the front of the Serial.print statements in your loop() function.

**Packet Serial Number:** Add an int variable “SerNum” at the beginning of the code to act as the packet serial number that gets incremented (SerNum++) inside the loop() function to uniquely identify each telemetry packet. This serial number should be included in the packet right after the “T#” packet format identifier.

**Discrete 8 Bits:** Now add the final 8 discrete I/O bits on the end of the telemetry format. These bits represent the state on the 8 pins P2 through P12. For simplicity, we will just use one sensor, the existing switch on P11. Remember to add a variable defining the switch state as pin 11 and to accept the incoming data. In your setup() function, add a pinMode() statement for the switch. In your loop() function, add a digitalRead() statement to read in the status of the switch. Add a Serial.print command to the end of your packet string to print out the status of your switch (a 1 or a 0). Print an additional dummy set of seven 1’s and 0’s, e.g. “0001110” to fill-out the remaining 8-bit telemetry format. Test your code and make sure the format matches the above telemetry format and is coming for your numbered (X) Arduino board.

**Telemetry Format: LABSAT-X>ALL:T#002,132,138,159,131,213,10001110**

**Part F. Serial Pin OUTPUT:** Your code now prints all data to the attached PC via the USB port. But to send it to a transmitter, it needs to go out a serial pin. To configure your Arduino to also output data on digital Pin 3, add the appropriate lines from the code below to include the Software Serial library and configure it at the start of your code. This implements a serial port driver on any desired digital pin. To setup and use this serial pin driver, add the ‘mySerial.begin(9600);’ in your setup() function. Also, add the following code before the setup function.

#include <SoftwareSerial.h>

SoftwareSerial mySerial(12,3,true); //configure (RX Pin12, TX Pin3, true=RS232)

Add the following commands in the loop function but place them so that the text will be printed in their respective locations (Arduino IO monitor or Putty)

Serial.println(“This goes out the USB port”);

mySerial.println(“This goes out on Pin3”);

*Change the delay to 2 seconds* then duplicate, one-for-one each of your existing Serial.print() statements with ‘mySerial.print();’ statements so that a copy of your telemetry also goes out pin 3.

When you think this is working, use the two-wires connected to a 9 pin serial port connector to your PC with the dark wire to ground and the yellow or lighter color wire to take the output of your pin 3 over to the serial input of your PC on its pin 2 of the serial port. Open PuTTY, select serial button, and OPEN. It defaults to 9600 baud. You should see your output so you can check if it is formatted correctly.

**NOTE**: If the lab is using a patch panel, you will need to change the delay(8000) so that your telemetry only updates once every 8 seconds. With this low duty cycle, you can share the data channel with your partners LAB sat from his serial port and all the other Labsats if we paralleled them onto the same channel.

**Part G. Data Decoding and Engineering Unit Conversion.**

At this point, your telemetry packet with 5 analog variables and 8 bits are only relative counts, and on the ground they need to be converted to actual units such as Temperature, Light level, volts, etc. This process is called ***engineering unit conversion*** and is usually done on the ground in the telemetry receive system where changes are easier to make as calibrations may change during spacecraft testing.

(**Caution**, Above 14 volts, take your values quickly to the next step at the higher voltages or you will smoke the load resistor!)

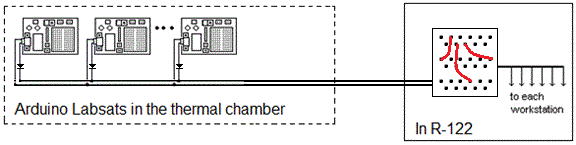
**Voltage and Current:** Connect the power supply to the top of the load resistor Vin and to the ground end of the Rdrop resistor ***and adjust from 2 to 16 volts every 2 volts***. Record and Enter the Vin and Iin telemetry counts and associated voltages and currents as shown on the power supply into Excel. The voltage counts should be about 10 times the voltage and the current counts should be about the same as the current in Milliamps.

Have Excel calculate a linear trend line equation of the form (V = A \* X + B ) for converting the X counts to actual volts and current on the ground. The offset B should pretty much be zero in these cases, so what are your two engineering unit conversion equations for this Voltage and Current Telemetry. Record the equation for later use.

**Light Sensor:** Calibrate the photoresistor by placing the lab’s light meter next to it. Shadow the light level with a piece of paper to get a range of light value counts between about 1 and 999 for the light sensor while also recording the light meter level in lux. Record and enter the data into Excel, using an X-Y plot of light level (Y axis) versus count(X axis). Calculate and record the trend line equation to use on the ground to convert these counts back to light level?

**Thermistors:** You can calibrate the thermistors for temperature by placing the Arduino telemetry system into the thermal chamber and vary the temperature while recording the telemetry data. A serial port cable from the thermal chamber runs to the R122 patch panel where the data can be patched to your PuTTY serial port as shown below.

Table . Temp Data



219, 4.5

328, 6.8

436, 15.8

525, 23.1

556, 26.3

666, 36.9

720, 43

760, 48

778, 50.5

832, 58.5

845, 60

Figure . Thermal Chamber Setup

Unfortunately, running a dozen temperatures from -40 to +60C and letting the system stabilize at each one takes several hours.  ***So we will simply give you data previously recorded in Table 1.*** The count is on the left and the temperature in C is on the right. Enter them into Excel to plot temperature (Y) versus telemetry count(X). Calculate and record a 3rd order trend line equation for Temperature that can be used for engineering unit conversion of counts back to the original temperature on the ground?

**Part H. Flight Code Calibration:** Now that you have developed Engineering Unit Conversion equations to convert raw counts back into actual Light levels, Temperatures, Voltages and Currents, modify your telemetry transmitting code to apply these conversions to the A1, A2, A3 and A4 analog data so that you can transmit the resulting Lux, Temperature, Voltage and Milliamps directly.

Don’t forget to declare these variables and the output as float. To do the math, you will have to call the math library in your SETUP code using the following script #include<math.h>. To raise an equation to the third power us the script pow(x,3) where x is the variable you wish to cube.

To avoid unnecessary decimal digits, limit your data to only a single decimal point. You can do this in the Serial.print(X,1) statement by using a “1” instead of “DEC” to force the formatting to only one decimal point.

**Lab Report:** The report for this lab will be included with Lab Report 1 as part of the Telemetry portion. You will include your Arduino code as an Appendix.