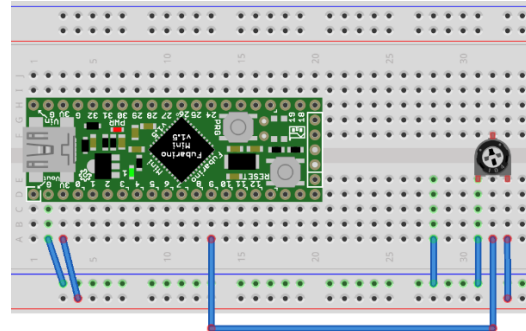
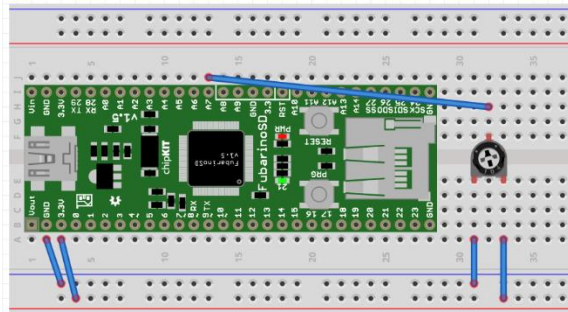


## Lab 6 - ADC - Functions and Program Flow

### Part 1 - ADC Simple Example

In this lab we will learn how to use the analog input capability of the Fubarino. This first sketch is a simple sketch to demonstrate this principle. Connect the potentiometer with one side to 3.3V and the other at GND with the middle (wiper) going to the A7 input of the board.



Upload and the program and open the serial terminal and look at the output.

```
void setup() {
  // Configure Analog input A6 and A7
  pinMode(A6, INPUT);
  pinMode(A7, INPUT);

  Serial.begin(115200); // Open the serial port
  while (!Serial);     // Wait for the PC to open it's serial port
  delay(100);          // Small delay before transmitting data.
  Serial.println("A6,A7"); // Print the header information
}

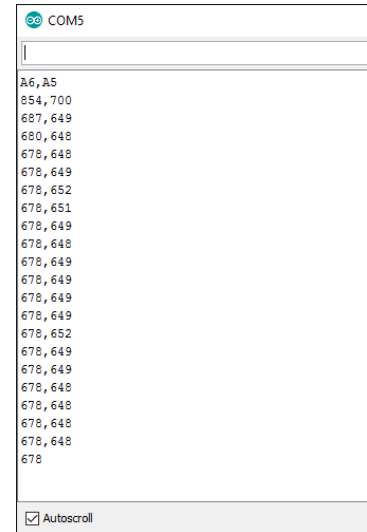
void loop() {
  uint16_t value;

  value = analogRead(A6); // Read analog input A6
  Serial.print(value, DEC);
  Serial.print(",");
  value = analogRead(A7); // Read analog input A6
  Serial.print(value, DEC);
  Serial.println("");
  delay(100);
}
```

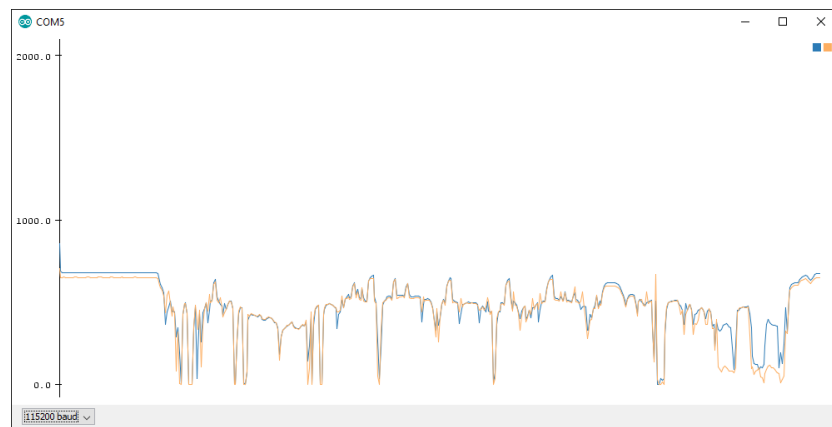
Notice that the string “A6,A7” is not printed until the serial port has been opened on the PC. This is a nice trick so that no output is missed, but it hangs your board in an endless loop while its waiting for the PC to open its serial port. Once the serial terminal is open adjust the pot and notice that the measured voltage of the ADC is displayed on the serial terminal.

You may also notice that the A6 values change as well even though there is nothing connected to this input. This is due to the multiplexed nature of the ADC within the PIC32 microcontroller. Floating ADC inputs will tend to read values close to those inputs that were measured previously to their current reading, in this case A6 is close to A7.

Next, close the serial terminal and reset your board (but don't go back into the bootloader). From the Arduino IDE Tools menu select the “Serial Plotter”. This will open a graph that will display the sensed values on a graph rather than in textual format.



```
COM5
A6, A5
854, 700
687, 649
680, 648
678, 648
678, 649
678, 652
678, 651
678, 649
678, 648
678, 649
678, 649
678, 649
678, 649
678, 652
678, 649
678, 648
678, 648
678, 648
678
Autoscroll
```



### Check off

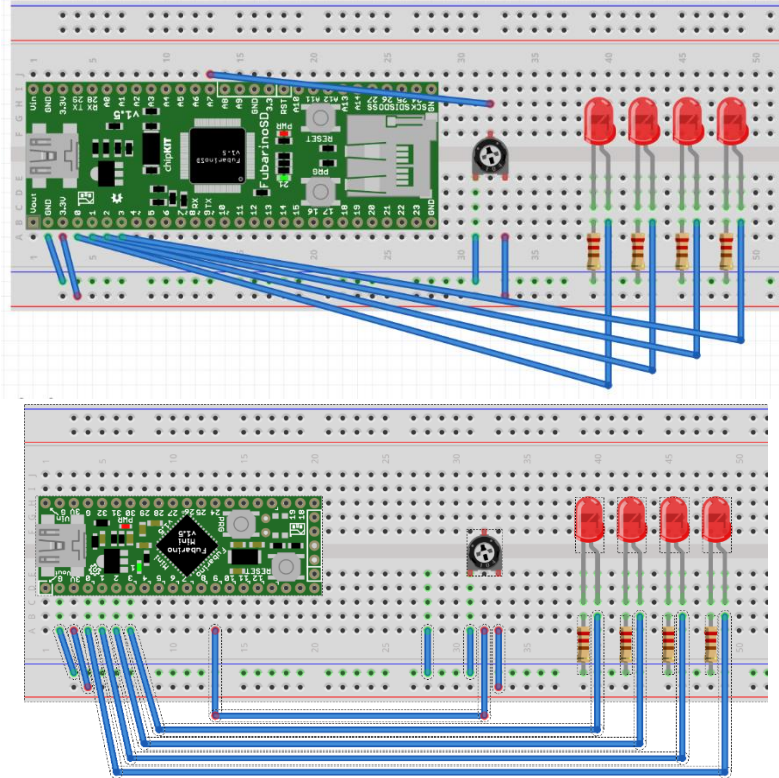
Call the instructor over for check off when you have values being displayed in the serial plotter.

Why does the value from A6 mirror the value from A7.

Compare and contrast what happens if you touch the analog pins A6 and A7 with your finger?

## Part 2 - Use ADC to control display speed

**Create a simple standalone program** that does a binary count sequence on the LEDs (such as we did in lab 2) and use the ADC value from the pot to control the counting speed. This can be accomplished by using the value obtained from an analog read in the delay function.



### Check off

Call the instructor over for check off when you get it working.

## Part 3 - Use ADC to control display speed – 10 Seconds

Alter the program from part 2 so that the delay goes from 0 to 10 seconds rather than from 0 to 1 second.

Hint: The \* character in the C++ programming language is the multiply operator.

### Check off

Call the instructor over for check off when you get it working.

## Part 4 - ADC to LEDs

**Start off this section by using the code from the lab 5 part 8** and the circuit from part 2/3. Augmenting the program from lab 5 with a single line of code that allows us to read pin A7 as an analog value. If you have not completed lab 5 part 8, you can start with lab 5 parts 2-7.

Save the previous lab and save it with a new name (like lab6\_part2).

Add the pinMode command in the setup function:

```
void setup()
{
  pinMode(A7, INPUT);
  nibbleToPinsInit(0,1,2,3);
  ...
}
```

Add the nibbleToPins line in your setup function:

```
void loop()
{
  nibbleToPins(analogRead(A7)/64,0,1,2,3);
  if(receive_function(command,sizeof(command)))
  {
  }
}
```

Four LED's should still be hooked up to pins 0-3.

### Analysis:

```
analogRead(A7)
```

This function returns a value that indicates the voltage on the analog pin A7. The range of values is 0 for GND and 1023 for the ADC reference voltage.

### Check off

Call the instructor over for check off when you get it working be prepared to explain why the divide by 64 is there and why it is outside "if(receive\_function(command,sizeof(command)))"

## Part 5 - Sensor

### Starting with the program from the previous part of this lab.

Add a command to the parser **adc** that reads an analog pin and prints out the value in decimal.

```
Serial.println(analogRead(A7), DEC);
```

Call it several times while adjusting the pot.

#### **Analysis:**

This allows you to see the values as numbers instead of just lights.

#### **Check off**

Call the instructor over for check off when you get it working.

## Part 6 - Serial Voltmeter

### Starting with the program from the previous part of this lab.

Add a command called **volts** to the parser that returns the voltage from the pot. (Remember full range on the ADC is 1024 and maximum input voltage is 3.3V)

```
float value;  
value = (float)analogRead(A7);  
Serial.println(value / 1023.0 * 3.3, 2);
```

When using `Serial.println(value, decimal_places)` is used to print out a floating point value the first parameter is the floating point value and the second parameter is the number of decimal places to print. The following example will print the string "3.1415". The "9" is not printed because it is the fifth decimal place and we only asked for four places to be printed.

```
Serial.println(3.14159,4); //output a four place decimal to Serial Monitor
```

Verify the functionality with a volt meter by measuring the values at the ADC pin with a volt meter and issuing a **volts** command.

#### **Analysis:**

Processing an input into a more familiar unit.

#### **Check off**

Call the instructor over for check off when you get it working.

## Part 7 - Thermister

### Starting with the program from the previous part of this lab.

Create a voltage divider with a 10K resistor and the 10K thermistor included in the kit between 3.3V and GND with the thermistor on the GND side.

Tap between the resistor and thermistor and connect it to an analog pin.

Use the `adc` command to get the values for ambient and with your fingers holding it.

Try switching the thermistor and resistor and test again, note how the values change.

Try to display the value on the LEDs so that room temperature is 0 and holding your finger on it is 15 in binary.

### **Analysis:**

This helps you to see how an input that is not full range can still be used to control something that does go from zero to a max value.

### **Check off**

Call the instructor over for check off when you get it working.

Be prepared to answer which way the thermistor's resistance changes as it gets warmer.

## Part 8 - CdS Cell (Light Sensor)

Change the thermistor from the previous section to a light sensor.

Try to display the value on the LEDs so that ambient light is 0 LEDs and holding your hand over it is 15 in binary.

### **Analysis:**

Same as Part 5 but with a different sensor type.

### **Check off**

Call the instructor over for check off when you get it working.

Be prepared to answer which way the light sensor's resistance changes as it gets more light.

## Part 8 - Light Sensor Revisit

Starting with the program from part 8 of this lab. Comment out the analog read command at the top of the loop function:

```
//nibbleToPins (analogRead (A7) /64, 0, 1, 2, 3) ;
```

Replace the code with code that will light the LEDs from the ADC with “if statements” that will light up 0,1,2,3 or 4 lights depending on how bright the light is that is hitting the sensor (the brighter the light, the more lights on).

### Analysis:

A more human friendly way of displaying the brightness data on the LEDs.

Should help you learn how to use a analog signal to control stuff, those outputs could be connected to relays that turn on exterior lights as it gets darker at your house.

### Check off

Call the instructor over for check off when you get it working.

## Part 9 - Light Meter Calibration

Add two Commands to the parser "calbright" and "caldark". Each command will store the current value of the light sensor when entered. This way you can expose the sensor to the brightest source available and issue the “calbright” command and the program will remember this value. Next you will obscure the sensor casting a shadow and making the sensor detect as little light as possible. Issue the “caldark” command and save the values in a different variable. Use these two stored values to scale the values sent to the LED’s from the previous part in this lab. You now can adjust it to different lighting conditions without recompiling the code.

### Analysis:

Sometimes you need to be able to change a device for conditions without reprogramming it.

### Check off

Call the instructor over for check off when you get it working.

## Part 10 - Light Meter Graphing

Print out the scaled values from the calibrated meter in your main loop. Open the Serial Plotter and plot the values as you obscure the light hitting the sensor.

### Check off

Call the instructor over for check off when you get it working.