OPEnS Pyranometer: IoT Solar Radiation Sensor
Apogee Silicon Pyranometer Documentation

Garen Porter, Brooke Weir, Alejandro Tovar

Abstract

This is our documentation of using the Apogee Silicon Pyranometer. The document discusses its use and its specifications. It goes over the data it reads, its conversion rating, and how we integrate with the feather M0.

1 Background

The Pyranosaurs team at OPEnS lab at Oregon State University is building a solar radiation sensor. One prototype that the team is exploring is a lux sensor (Adafruit TSL2591) that can measure either temperature in degrees Celsius. It can be pointed at an object and it will measure the temperature. The lux sensor can be wired to a multiplexer that is connected to a Feather M0 and Adalogger. Using LOOM code, the lux sensor readings can be put onto a micro SD card and then transferred to a computer via a SD card. The data can either be a text file or a csv file. After building the lux sensor, we have to determine it's accuracy. Using a similar device like the Apogee Silicon Pyranometer can be used to test the lux sensor

2 Specifications

1. Model: SP-110
2. Power Source: Self Powered (Any external power source damages the sensor)
3. Output Range: 0 - 250 mV
4. Conversion Rating: 1 mV = 5 W/m^2
5. Spectral Range: 360 - 1120 nm
6. Data it Reads: Visible and Short Wave Radiation

3 Wiring

The Apogee SP-110 model has 3 wires. They are the shield, the low signal, and the high signal. To properly connect this sensor, the shield and low are jumpered into ground. Then the high goes to an analog input. This can be used with a multimeter where you connect ground to the shield and low. Then connect high to VCC. The multimeter should be set to read mV in DC voltage. Then you should be able to read the SP-110 in mV.

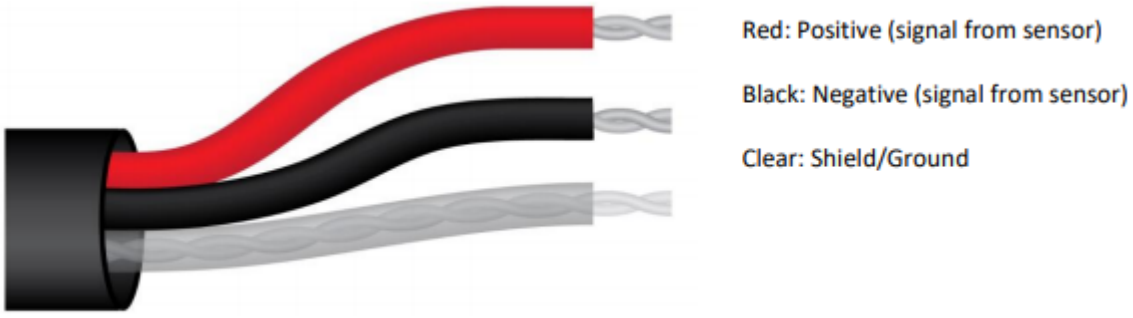

Figure 1: Apogee Wiring

4 Integration With M0

We integrated the apogee sensor to the M0 board by using an Adafruit ADC called ADS1115. It is an I2C chip so it is possible to integrate similarly to other I2C boards. The ADS1115 is integrated into the LOOM system and the LOOM multiplexer(tca9548A). For the Apogee, the two analog pins will be connected to two analog inputs from the ADS1115. The shield will be connected to ground. For the ADC, there are 5 pins. They are VDD, GND, SDA,

SCL, and ADDR. The VDD is for power. The SDA and SCL pins are for I2C communication to the LOOM system. The ADDR pin is wired depending on the address location you declare in code. We used the default address 0x48, meaning the ADDR pin needs to be connected to ground. The four previous pins connect to the multiplexer as shown in figure 2.

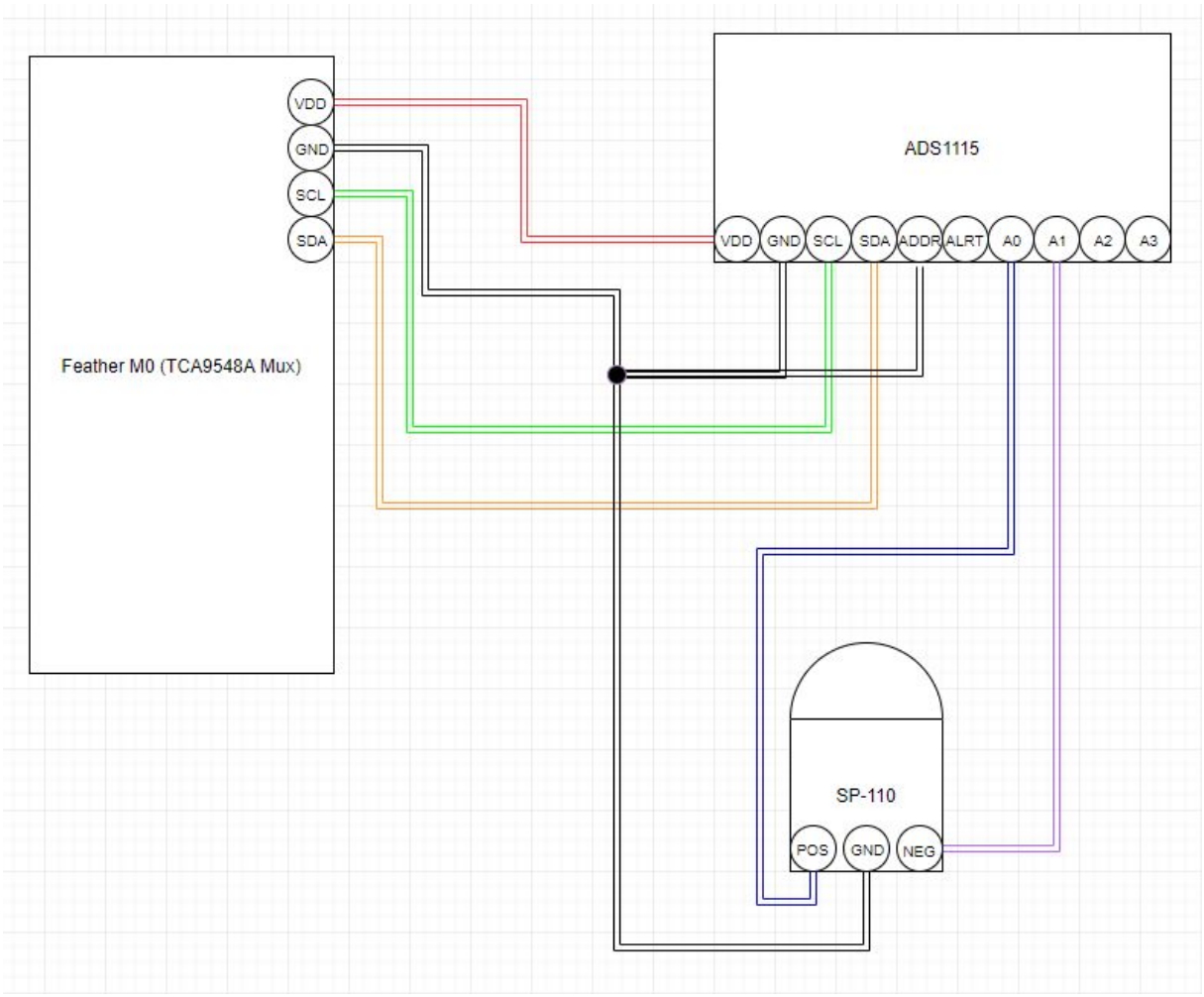

Figure 2: Circuit Diagram to Integrate Apogee with Feather m0

5 Code

The code below is the version that is integrated with LOOM. It is available online on Github.
Github Link: Loom Pyranometer

`File name : loom_ads1115.h`

```
// ================================================================
// ===                         LIBRARIES                        ===
// ================================================================
#include "ADS1115.h"
#include <Wire.h>
```

```
// ================================================================
// ===                        DEFINITIONS                       ===
// ================================================================


// ================================================================
// ===                         STRUCTURES                       ===
// ================================================================

struct config_ads1115_t {
  uint16_t delay;
};

struct state_ads1115_t {
  ADS1115 inst_ads1115;
  double radiance;
};

// ================================================================
// ===                    GLOBAL DECLARATIONS                   ===
// ================================================================

struct config_ads1115_t config_ads1115;
struct state_ads1115_t  state_ads1115;

// ================================================================
// ===                    FUNCTION PROTOTYPES                   ===
// ================================================================

bool setup_ads1115();
void package_ads1115(OSCBundle *, char [], uint8_t port);
void measure_ads1115();

// ================================================================
// ===                           SETUP                          ===
// ================================================================
//
// Runs any startup for ads1115 that should occur on device startup
//
// @return  Whether or not sensor initialization was successful
//

bool setup_ads1115()
{
  //Setup Here
 // Serial.println("Initializing ADS1115...");
  bool is_setup;

  state_ads1115.inst_ads1115 = ADS1115(0x48);    // creating address location
  state_ads1115.inst_ads1115.initialize();                          // initialize the device
  state_ads1115.inst_ads1115.setMode(ADS1115_MODE_CONTINUOUS);      // set mode
```

```
  if(state_ads1115.inst_ads1115.testConnection()){                  // test connection
    is_setup = true;
    config_ads1115.delay = 4000;
    LOOM_DEBUG_Println("Initialized ads1115");
  }
  else{
    is_setup = false;
    LOOM_DEBUG_Println("Failed to initialize ads1115");
  }

    //Serial.print("FINISHED SETTING UP ADS1115\n");
  return is_setup;
}

// ================================================================
// ===                         FUNCTIONS                        ===
// ================================================================

// --- PACKAGE ads1115 --- (Multiplexer Version)
//
// Adds OSC Message of most recent sensor readings to a provided OSC bundle
//
// @param bndl                  The OSC bundle to be added to
// @param packet_header_string  The device-identifying string to prepend to OSC messages
//   if I2C multiplexer sensor, then also
// [@param port                  Which port of the multiplexer the device is plugged into]
//

void package_ads1115(OSCBundle *bndl, char packet_header_string[], int port)
{
    //Serial.print("// --- PACKAGE tmp007 --- (Multiplexer Version)
//
// Adds OSC Message of most recent sensor readings to a provided OSC bundle
//
// @param bndl                  The OSC bundle to be added to
// @param packet_header_string  The device-identifying string to prepend to OSC messages
//   if I2C multiplexer sensor, then also
// [@param port                  Which port of the multiplexer the device is plugged into]
//PACKAGING BUNDLE\n");
  char address_string[255];
  sprintf(address_string, "%s%s%d%s", packet_header_string, "/port", port, "/ads1115/data");

  OSCMessage msg = OSCMessage(address_string);

  msg.add("W/m^2").add((int32_t)state_ads1115.radiance);




  bndl->add(msg);
}

#if is_multiplexer != 1
```

```
void package_ads1115(OSCBundle *bndl, char packet_header_string[])
{
  char address_string[255];

  //W/m^2
  sprintf(address_string, "%s%s%s%s", packet_header_string, "/", ads1115_0x48_name, "apogee");
  bndl->add(address_string).add((int32_t)state_ads1115.radiance);

}
#endif

// --- MEASURE ads1115 ---
//
// This function uses the ADC to read a voltage differential between two analog inputs.
// It reads in a voltage and will be converted into mV.
//

void measure_ads1115(){
    double mV;
    int counts;
    counts = state_ads1115.inst_ads1115.getConversionP0N1();    // counts up to 16 bits
    mV = counts * state_ads1115.inst_ads1115.getMvPerCount();   // converts to mV
    state_ads1115.radiance = mV * 5;                            // multiply by conversion factor:
                                                                // SP-110 Radiance = 5 W/m^2 per mV
    #if LOOM_DEBUG == 1
      Serial.print(F("[ ")); Serial.print(millis()); Serial.print(F(" ms ] "));
      Serial.print(F("mV ")); Serial.print(mV); Serial.print(F("  "));
      Serial.print(F(" W/m^2: ")); Serial.print(state_ads1115.radiance); Serial.println(F("  "));
    #endif
    delay(config_ads1115.delay);

}
```

Here is the code for the specific line:

```
    counts = state_ads1115.inst_ads1115.getConversionP0N1();    // counts up to 16 bits
```

It shows how the function .getConversionP0N1() works. It is in the Loom Library under the node file.
File name: ADS1115.c

```
/** Read differential value based on current MUX configuration.
 * The default MUX setting sets the device to get the differential between the
 * AIN0 and AIN1 pins. There are 8 possible MUX settings, but if you are using
 * all four input pins as single-end voltage sensors, then the default option is
 * not what you want; instead you will need to set the MUX to compare the
 * desired AIN* pin with GND. There are shortcut methods (getConversion*) to do
 * this conveniently, but you can also do it manually with setMultiplexer()
 * followed by this method.
 *
 * In single-shot mode, this register may not have fresh data. You need to write
 * a 1 bit to the MSB of the CONFIG register to trigger a single read/conversion
 * before this will be populated with fresh data. This technique is not as
 * effortless, but it has enormous potential to save power by only running the
 * comparison circuitry when needed.
 *
```

```
 * @param triggerAndPoll If true (and only in singleshot mode) the conversion trigger
 *        will be executed and the conversion results will be polled.
 * @return 16-bit signed differential value
 * @see getConversionP0N1();
 * @see getConversionP0N3();
 * @see getConversionP1N3();
 * @see getConversionP2N3();
 * @see getConversionP0GND();
 * @see getConversionP1GND();
 * @see getConversionP2GND();
 * @see getConversionP3GND);
 * @see setMultiplexer();
 * @see ADS1115_RA_CONVERSION
 * @see ADS1115_MUX_P0_N1
 * @see ADS1115_MUX_P0_N3
 * @see ADS1115_MUX_P1_N3
 * @see ADS1115_MUX_P2_N3
 * @see ADS1115_MUX_P0_NG
 * @see ADS1115_MUX_P1_NG
 * @see ADS1115_MUX_P2_NG
 * @see ADS1115_MUX_P3_NG
 */
int16_t ADS1115::getConversion(bool triggerAndPoll) {
    if (triggerAndPoll && devMode == ADS1115_MODE_SINGLESHOT) {
      triggerConversion();
      pollConversion(I2CDEV_DEFAULT_READ_TIMEOUT);
    }
    I2Cdev::readWord(devAddr, ADS1115_RA_CONVERSION, buffer);
    return buffer[0];
}

/** Get AIN0/N1 differential.
 * This changes the MUX setting to AIN0/N1 if necessary, triggers a new
 * measurement (also only if necessary), then gets the differential value
 * currently in the CONVERSION register.
 * @return 16-bit signed differential value
 * @see getConversion()
 */
int16_t ADS1115::getConversionP0N1() {
    if (muxMode != ADS1115_MUX_P0_N1) setMultiplexer(ADS1115_MUX_P0_N1);
    return getConversion();
}
```