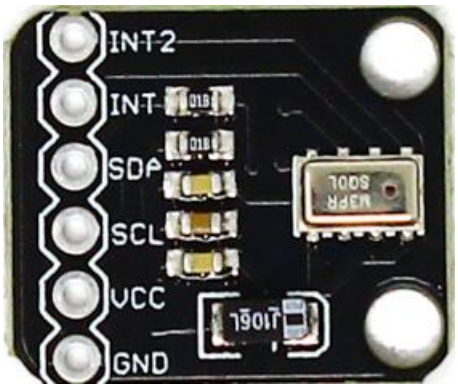
SmartElex Altitude Pressure Sensor Breakout Board - MPL3115A2

The MPL3115A2 is a low-cost, low power, highly accurate barometric pressure sensor. Use this sensor to detect changes in barometric pressure (weather changes) or for altitude (UAV controllers and the like). The sensor is *very* sensitive and capable of detecting a change of only 0.05kPa which equates to a 0.3m change in altitude.



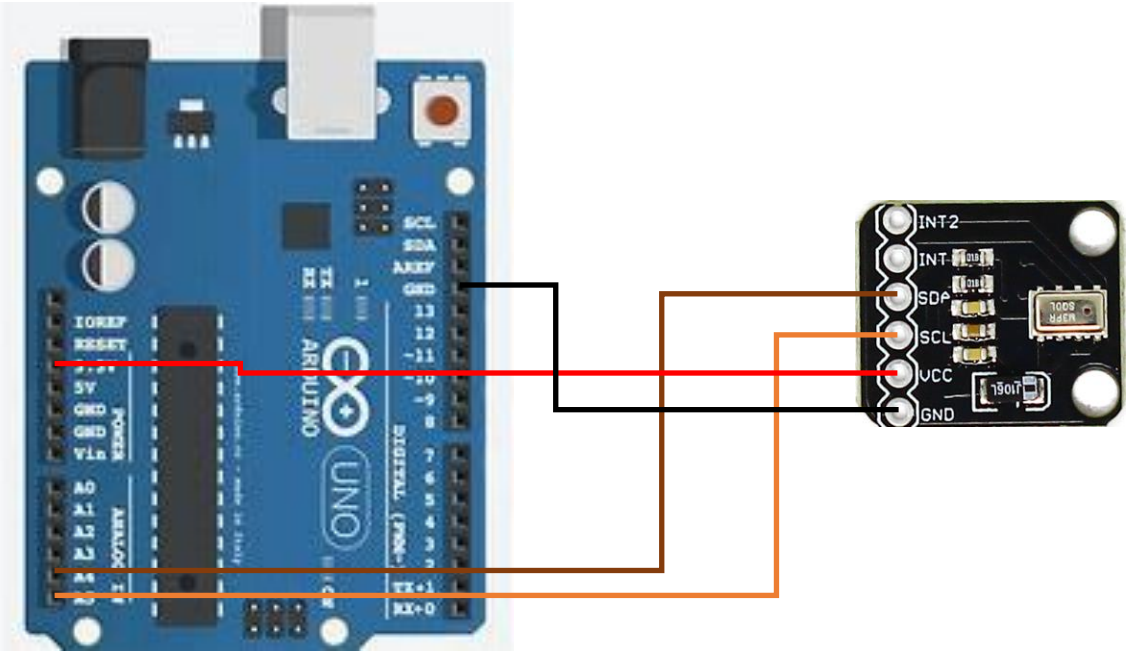
Things you should know about this sensor:

- Uses the I^2C interface
- Only one sensor can reside on the I^2C bus
- Uses the I^2C repeated start condition. Arduino supports this, check if you're using a different microcontroller.
- Typical pressure accuracy of ±0.05kPa
- Typical altitude accuracy of ±0.3m
- Typical temperature accuracy of ±3C
- 3.3V sensor - use inline logic level converters or 330 Ohm resistors to limit 5V signals

This sensor is ideal for environmental sensing, a weather station, or datalogging. It is a worthy replacement for the BMP085 and is more sensitive than the MPL115A1.

Wiring

Connecting the MPL3115A2 to Arduino:



Arduino	MPL3115A2
SCL(A5)	SCL
SDA(A4)	SDA
3.3v	VCC
GND	GND

Arduino Code

The following Arduino example will get your sensor up and running quickly, and will show you current pressure in Pascals.

```
#include <Wire.h> // for IIC communication

#define STATUS     0x00

#define OUT_P_MSB  0x01

#define OUT_P_CSB  0x02
```

```
#define OUT_P_LSB  0x03

#define OUT_T_MSB  0x04

#define OUT_T_LSB  0x05

#define DR_STATUS  0x06

#define OUT_P_DELTA_MSB  0x07

#define OUT_P_DELTA_CSB  0x08

#define OUT_P_DELTA_LSB  0x09

#define OUT_T_DELTA_MSB  0x0A

#define OUT_T_DELTA_LSB  0x0B

#define WHO_AM_I   0x0C

#define F_STATUS   0x0D

#define F_DATA     0x0E

#define F_SETUP    0x0F

#define TIME_DLY   0x10

#define SYSMOD     0x11

#define INT_SOURCE 0x12

#define PT_DATA_CFG 0x13

#define BAR_IN_MSB 0x14

#define BAR_IN_LSB 0x15

#define P_TGT_MSB  0x16

#define P_TGT_LSB  0x17

#define T_TGT      0x18

#define P_WND_MSB  0x19

#define P_WND_LSB  0x1A

#define T_WND      0x1B
```

```
#define P_MIN_MSB  0x1C

#define P_MIN_CSB  0x1D

#define P_MIN_LSB  0x1E

#define T_MIN_MSB  0x1F

#define T_MIN_LSB  0x20

#define P_MAX_MSB  0x21

#define P_MAX_CSB  0x22

#define P_MAX_LSB  0x23

#define T_MAX_MSB  0x24

#define T_MAX_LSB  0x25

#define CTRL_REG1  0x26

#define CTRL_REG2  0x27

#define CTRL_REG3  0x28

#define CTRL_REG4  0x29

#define CTRL_REG5  0x2A

#define OFF_P      0x2B

#define OFF_T      0x2C

#define OFF_H      0x2D


#define MPL3115A2_ADDRESS 0x60 // 7-bit I2C address


long startTime;

void setup()

{

  Wire.begin();        // join i2c bus
```

```
  Serial.begin(57600);  // start serial for output
  if(IIC_Read(WHO_AM_I) == 196)
    Serial.println("MPL3115A2 online!");
  else
    Serial.println("No response - check connections");
  // Configure the sensor
  setModeAltimeter(); // Measure altitude above sea level in meters
  //setModeBarometer(); // Measure pressure in Pascals from 20 to 110 kPa

  setOversampleRate(7); // Set Oversample to the recommended 128
  enableEventFlags(); // Enable all three pressure and temp event flags
}
void loop()
{
  startTime = millis();
  float altitude = readAltitude();
  Serial.print("Altitude(m):");
  Serial.print(altitude, 2);
  //altitude = readAltitudeFt();
  //Serial.print(" Altitude(ft):");
  //Serial.print(altitude, 2);
  /*float pressure = readPressure();
   Serial.print(" Pressure(Pa):");
   Serial.println(pressure, 2);*/
  //float temperature = readTemp();
```

```
  //Serial.print(" Temp(c):");
  //Serial.print(temperature, 2);
  //float temperature = readTempF();
  //Serial.print(" Temp(f):");
  //Serial.print(temperature, 2);
  Serial.print(" time diff:");
  Serial.print(millis() - startTime);
  Serial.println();
  //delay(1);
}
//Returns the number of meters above sea level
float readAltitude()
{
  toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading
  //Wait for PDR bit, indicates we have new pressure data
  int counter = 0;
  while( (IIC_Read(STATUS) & (1<<1)) == 0)
  {
    if(++counter > 100) return(-999); //Error out
    delay(1);
  }
  // Read pressure registers
  Wire.beginTransmission(MPL3115A2_ADDRESS);
  Wire.write(OUT_P_MSB);  // Address of data to get
  Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not supported before Arduino V1.0.1!
```

```
  Wire.requestFrom(MPL3115A2_ADDRESS, 3); // Request three bytes

  //Wait for data to become available
  counter = 0;
  while(Wire.available() < 3)
  {
    if(counter++ > 100) return(-999); //Error out
    delay(1);
  }

  byte msb, csb, lsb;
  msb = Wire.read();
  csb = Wire.read();
  lsb = Wire.read();

  toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading

  // The least significant bytes l_altitude and l_temp are 4-bit,
  // fractional values, so you must cast the calulation in (float),
  // shift the value over 4 spots to the right and divide by 16 (since
  // there are 16 values in 4-bits).
  float tempcsb = (lsb>>4)/16.0;

  float altitude = (float)( (msb << 8) | csb) + tempcsb;

  return(altitude);
}

//Returns the number of feet above sea level
float readAltitudeFt()
{
  return(readAltitude() * 3.28084);
```

```
}

//Reads the current pressure in Pa

//Unit must be set in barometric pressure mode

float readPressure()

{

 toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading

 //Wait for PDR bit, indicates we have new pressure data

 int counter = 0;

 while( (IIC_Read(STATUS) & (1<<2)) == 0)

 {

    if(++counter > 100) return(-999); //Error out

    delay(1);

 }

 // Read pressure registers

 Wire.beginTransmission(MPL3115A2_ADDRESS);

 Wire.write(OUT_P_MSB);  // Address of data to get

 Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not supported before Arduino V1.0.1!

 Wire.requestFrom(MPL3115A2_ADDRESS, 3); // Request three bytes

 //Wait for data to become available

 counter = 0;

 while(Wire.available() < 3)

 {

  if(counter++ > 100) return(-999); //Error out

  delay(1);

 }
```

```
  byte msb, csb, lsb;
  msb = Wire.read();
  csb = Wire.read();
  lsb = Wire.read();
  toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading
  // Pressure comes back as a left shifted 20 bit number
  long pressure_whole = (long)msb<<16 | (long)csb<<8 | (long)lsb;
  pressure_whole >>= 6; //Pressure is an 18 bit number with 2 bits of decimal. Get rid of decimal portion.
  lsb &= 0b00110000; //Bits 5/4 represent the fractional component
  lsb >>= 4; //Get it right aligned
  float pressure_decimal = (float)lsb/4.0; //Turn it into fraction
  float pressure = (float)pressure_whole + pressure_decimal;
  return(pressure);
}
float readTemp()
{
  toggleOneShot(); //Toggle the OST bit causing the sensor to immediately take another reading

  //Wait for TDR bit, indicates we have new temp data
  int counter = 0;
  while( (IIC_Read(STATUS) & (1<<1)) == 0)
  {
     if(++counter > 100) return(-999); //Error out
     delay(1);
  }
```

```
 // Read temperature registers

 Wire.beginTransmission(MPL3115A2_ADDRESS);

 Wire.write(OUT_T_MSB);  // Address of data to get

 Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not supported before Arduino V1.0.1!

 Wire.requestFrom(MPL3115A2_ADDRESS, 2); // Request two bytes

 //Wait for data to become available

 counter = 0;

 while(Wire.available() < 2)

 {

  if(++counter > 100) return(-999); //Error out

  delay(1);



 byte msb, lsb;

 msb = Wire.read();

 lsb = Wire.read();


 // The least significant bytes l_altitude and l_temp are 4-bit,

 // fractional values, so you must cast the calulation in (float),

 // shift the value over 4 spots to the right and divide by 16 (since

 // there are 16 values in 4-bits).

 float templsb = (lsb>>4)/16.0; //temp, fraction of a degree

 float temperature = (float)(msb + templsb);

 return(temperature);
```

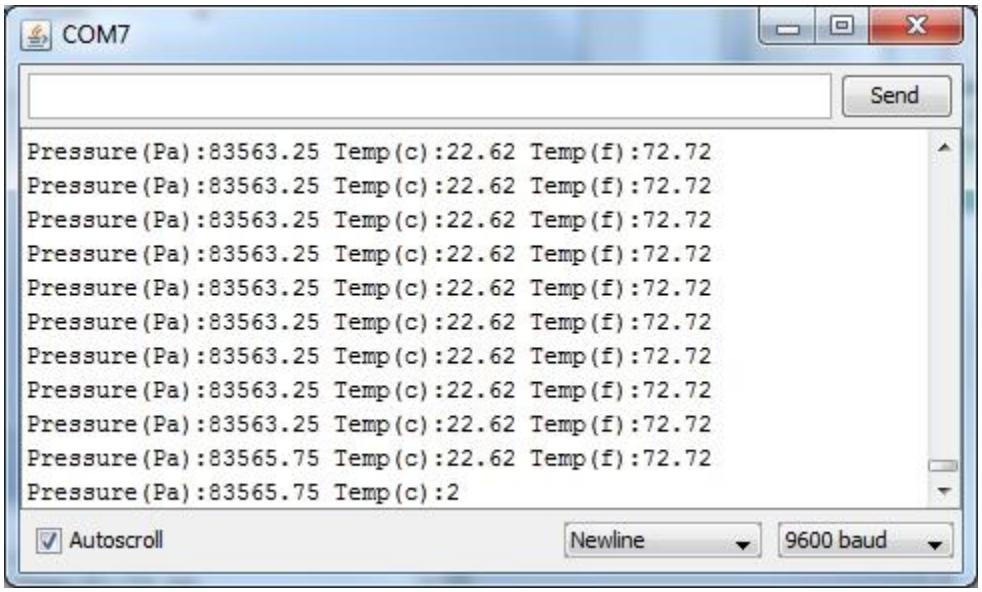
```
}

//Give me temperature in fahrenheit!

float readTempF()

{

  return((readTemp() * 9.0)/ 5.0 + 32.0); // Convert celsius to fahrenheit

}

//Sets the mode to Barometer

//CTRL_REG1, ALT bit

void setModeBarometer()

{

  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings

  tempSetting &= ~(1<<7); //Clear ALT bit

  IIC_Write(CTRL_REG1, tempSetting);

}

//Sets the mode to Altimeter

//CTRL_REG1, ALT bit

void setModeAltimeter()

{

  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings

  tempSetting |= (1<<7); //Set ALT bit

  IIC_Write(CTRL_REG1, tempSetting);

}

//Puts the sensor in standby mode

//This is needed so that we can modify the major control registers

void setModeStandby()
```

```
{
  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
  tempSetting &= ~(1<<0); //Clear SBYB bit for Standby mode
  IIC_Write(CTRL_REG1, tempSetting);
}

//Puts the sensor in active mode
//This is needed so that we can modify the major control registers
void setModeActive()
{
  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
  tempSetting |= (1<<0); //Set SBYB bit for Active mode
  IIC_Write(CTRL_REG1, tempSetting);
}
//Setup FIFO mode to one of three modes. See page 26, table 31
//From user jr4284
void setFIFOMode(byte f_Mode)
{
  if (f_Mode > 3) f_Mode = 3; // FIFO value cannot exceed 3.
  f_Mode <<= 6; // Shift FIFO byte left 6 to put it in bits 6, 7.
  byte tempSetting = IIC_Read(F_SETUP); //Read current settings
  tempSetting &= ~(3<<6); // clear bits 6, 7
  tempSetting |= f_Mode; //Mask in new FIFO bits
  IIC_Write(F_SETUP, tempSetting);
}
```

```
//Call with a rate from 0 to 7. See page 33 for table of ratios.
//Sets the over sample rate. Datasheet calls for 128 but you can set it
//from 1 to 128 samples. The higher the oversample rate the greater
//the time between data samples.
void setOversampleRate(byte sampleRate)
{
  if(sampleRate > 7) sampleRate = 7; //OS cannot be larger than 0b.0111
  sampleRate <<= 3; //Align it for the CTRL_REG1 register
  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
  tempSetting &= 0b11000111; //Clear out old OS bits
  tempSetting |= sampleRate; //Mask in new OS bits
  IIC_Write(CTRL_REG1, tempSetting);
}
//Clears then sets the OST bit which causes the sensor to immediately take another reading
//Needed to sample faster than 1Hz
void toggleOneShot(void)
{
  byte tempSetting = IIC_Read(CTRL_REG1); //Read current settings
  tempSetting &= ~(1<<1); //Clear OST bit
  IIC_Write(CTRL_REG1, tempSetting);
  tempSetting = IIC_Read(CTRL_REG1); //Read current settings to be safe
  tempSetting |= (1<<1); //Set OST bit
  IIC_Write(CTRL_REG1, tempSetting);
}
```

```
//Enables the pressure and temp measurement event flags so that we can

//test against them. This is recommended in datasheet during setup.

void enableEventFlags()

{

  IIC_Write(PT_DATA_CFG, 0x07); // Enable all three pressure and temp event flags

}

// These are the two I2C functions in this sketch.

byte IIC_Read(byte regAddr)

{

  // This function reads one byte over IIC

  Wire.beginTransmission(MPL3115A2_ADDRESS);

  Wire.write(regAddr);  // Address of CTRL_REG1

  Wire.endTransmission(false); // Send data to I2C dev with option for a repeated start. THIS IS NECESSARY and not supported before Arduino V1.0.1!

  Wire.requestFrom(MPL3115A2_ADDRESS, 1); // Request the data...

  return Wire.read();

}

void IIC_Write(byte regAddr, byte value)

{

  // This function writes one byto over IIC

  Wire.beginTransmission(MPL3115A2_ADDRESS);

  Wire.write(regAddr);

  Wire.write(value);

  Wire.endTransmission(true);

}
```

Once the library is installed, open Arduino, and expand the examples menu. You should see the MPL3115A2_Pressure sub-menu. Load the "Pressure" example onto the Arduino. Open the serial terminal at 9600bps. You will see the current barometric pressure and temperature in the room!



Load the `BarometricHgInch` example for an example that coverts pressure from Pascals to inches of mercury, altimeter setting adjusted. This type of pressure reading is used in the USA on Wunderground for home weather stations and aircraft.
Load the `Altimeter` example for an example that coverts pressure to current altitude in feet (or meters).

Explanation of Functions

The library and example code demonstrate the most popular functions supported by the MPL3115A2. Here is an explanation of all the available functions in the library:

- **myPressure.begin()** gets sensor on the I^2C bus.
- **myPressure.readAltitude()** returns a float with meters above sea level. Ex: 1638.94
- **myPressure.readAltitudeFt()** returns a float with feet above sea level. Ex: 5376.68
- **myPressure.readPressure()** returns a float with barometric pressure in Pa. Ex: 83351.25
- **myPressure.readTemp()** returns a float with current temperature in Celsius. Ex: 23.37

- **myPressure.readTempF()** returns a float with current temperature in Fahrenheit. Ex: 73.96
- **myPressure.setModeBarometer()** puts the sensor into Pascal measurement mode.
- **myPressure.setModeAltimeter()** puts the sensor into altimetry mode.
- **myPressure.setModeStandy()** puts the sensor into Standby mode. Required when changing CTRL1 register.
- **myPressure.setModeActive()** starts taking measurements!
- **myPressure.setOversampleRate(byte)** sets the # of samples from 1 to 128. See note below *
- **myPressure.enableEventFlags()** sets the fundamental event flags. Required during setup.

When you call the readAltitude, readAltitudeFt, readPressure, or readTemp you will get a float with the sensor reading or an error code:

- 1638.94 is an example of a valid reading.
- -999 indicates that I2C timed out (512ms max). Check your connections.

Table 59. System Output Sample Rate Selection

OS2	OS1	OS0	Oversample Ratio	Minimum Time Between Data Samples
0	0	0	1	6 ms
0	0	1	2	10 ms
0	1	0	4	18 ms
0	1	1	8	34 ms
1	0	0	16	66 ms
1	0	1	32	130 ms
1	1	0	64	258 ms
1	1	1	128	512 ms

Oversample settings

- **setOversampleRate(byte)** receives a value from 0 to 7. Check table 59 above. Allows the user to change sample rate from 1 to 128. Increasing the sample rate significantly decreases the noise of each reading but increases the amount of time to capture each reading. A oversample of 128 will decrease noise to 1.5Pa RMS but requires 512ms per reading. The datasheet recommends oversample of 128 for basic applications.

The MPL3115A2 has a large number of features. Checkout the datasheet for more info. This library covers the fundamentals.

Pressure vs Altimeter Setting

If you grabbed a few pressure readings and became confused when you checked your local weather conditions, you're not alone. The absolute pressure that the MPL3115A2 pressure sensor outputs is not the same as what weather stations refer to as pressure. Weather stations report pressure in lots of different units:

- millimeters Mercury (mmHg)
- inches Mercury (inHg)
- millibars or hectopascals (hPa)
- pounds per square inch
- atmospheres (Atm)
- kilogram per centimeter
- inches of water

In barometer mode, the MPL3115A2 outputs pressure readings in Pascals. This is most closely related to millibars or hectopascals. But, why does the sensor not agree with the station around the corner? This is because many stations report pressure in a few different formats. Have a look at all these numbers for the Boulder/Denver area. The key is that your local weather station is probably reporting the *Altimeter setting*.

Thank you National Oceanic and Atmospheric Administration (NOAA)! Did you know they're headquartered here in Boulder, CO?

- **Station pressure** - This is the pressure that is observed at a specific elevation and is the true barometric pressure of a location.
- **Altimeter setting** - This is the pressure reading most commonly heard in radio and television broadcasts. It is not the true barometric pressure at a station. Instead it is the pressure "reduced" to mean sea level using the temperature profile of the "standard" atmosphere, which is representative of average conditions over the United States at 40 degrees north latitude.
- **Mean sea level pressure** - This is the pressure reading most commonly used by meteorologists to track weather systems at the surface. Like the altimeter setting, it is a "reduced" pressure, which uses observed conditions rather than "standard" conditions to remove the effects of elevation from pressure readings.

The calculation to get from Pascals to 'Altimeter setting' is a bit gnarly:

$$h_m = 0.3048 \times h_{ft}$$

$$Alt = (P_{mb} - 0.3) \times \left(1 + \left(\left(\frac{1013.25^{0.190284} \times 0.0065}{288}\right) \times \left(\frac{h_m}{(P_{mb} - 0.3)^{0.190284}}\right)\right)\right)^{\frac{1}{0.190284}}$$

Formula to convert Pascal pressure to Altimeter setting

Grab the full formula here and give this great Altimeter setting calculator a try. This formula relies on two things: knowing the current pressure in milibars and knowing the height above sea level that the pressure was read. We recommend you capture altitude using a local survey point or a GPS receiver.

If you installed the MPL3115A2 library, you should also have the *BarometricHgInch* example sketch under the Examples->MPL3115A2_Pressure menu under the Arduino IDE. We didn't build this calculation into the library because it could potentially chew up a lot of RAM and code space calculating all the floating point math. But, if you're doing home weather station calculations, this should get you started.