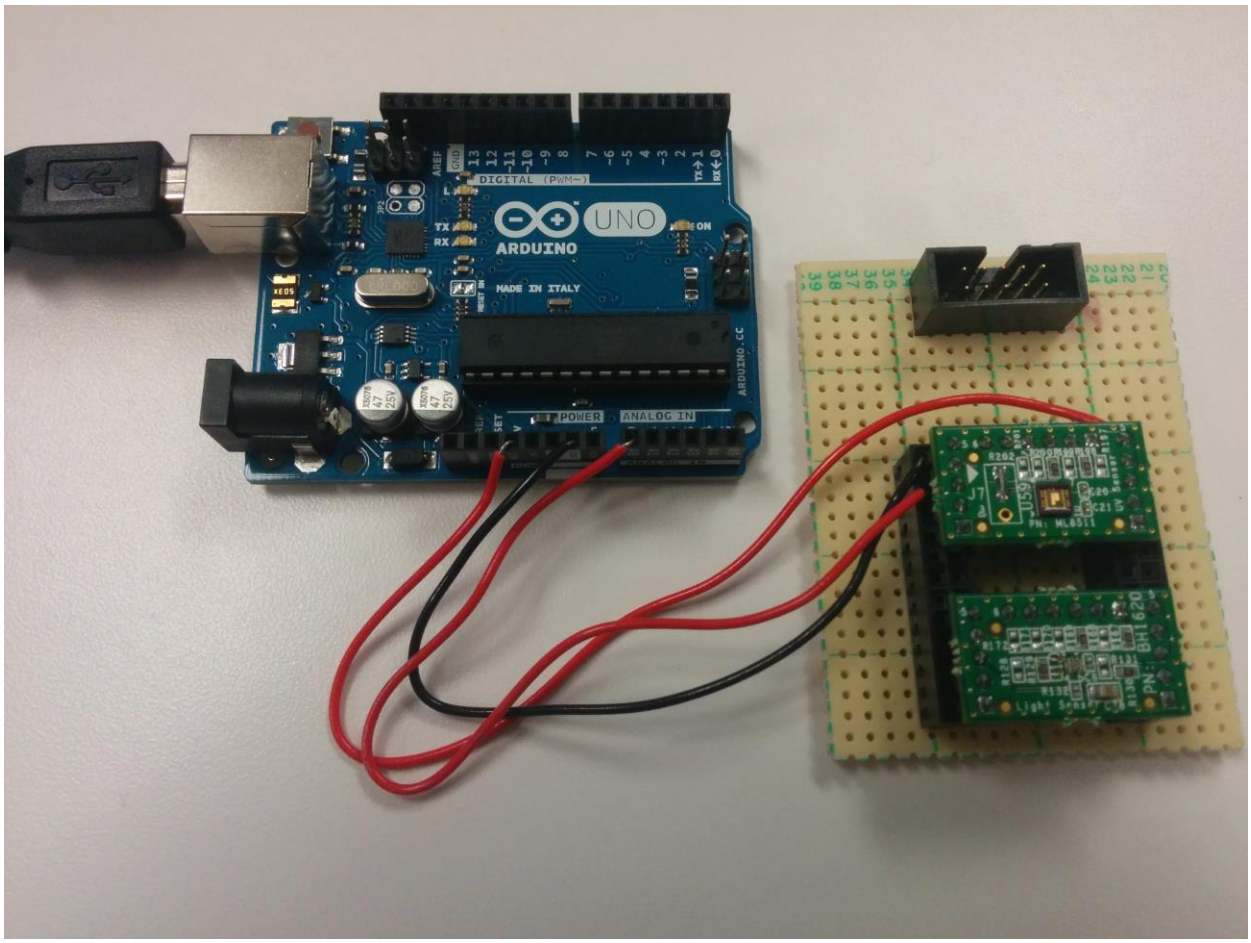
Connecting the Sensor Platform Kit to the Arduino UNO Platform



Above: ROHM Sensor Platform breakout boards connected to the Arduino Uno

Table of Contents

Copyright and License

The following except is copied directly from the Arduino FAQ (http://arduino.cc/en/Main/FAQ)

"What do you mean by open-source hardware?" - Open-source hardware shares much of the principles and approach of free and open-source software. In particular, we believe that people should be able to study our hardware to understand how it works, make changes to it, and share those changes. To facilitate this, we release all of the original design files (Eagle CAD) for the Arduino hardware. These files are licensed under a Creative Commons Attribution Share-Alike license, which allows for both personal and commercial derivative works, as long as they credit Arduino and release their designs under the same license.

The Arduino software is also open-source. The source code for the Java environment is released under the GPL and the C/C++ microcontroller libraries are under the LGPL.

Please note that all references to ROHM's Sensor Platform Kit are also shared under open source guidelines under the GNU General Public License, Version 3. Details can be found at the following link: https://github.com/ROHMUSDC/ROHMSensorPlatformEVK.

Revision History

Date	Description	Revision ID
12 January 2015	First Draft	A

Introduction

The following document was written to provide a brief connection guide and starting point for using ROHM's Sensor Platform Kit with the Arduino Uno. This guide assumes that the user has basic functional knowledge of both the Sensor Platform Kit and the Arduino itself. If this is not correct, please see the following links for other guides and information on these products.

ROHM's Sensor Platform Kit: https://github.com/ROHMUSDC/ROHMSensorPlatformEVK

Arduino: http://arduino.cc/

Please note that the Arduino platform is a microcontroller platform; thus, this document will explain and show examples of how to connect to and convert values for the ROHM Sensor Kit Sensors. This includes the following:

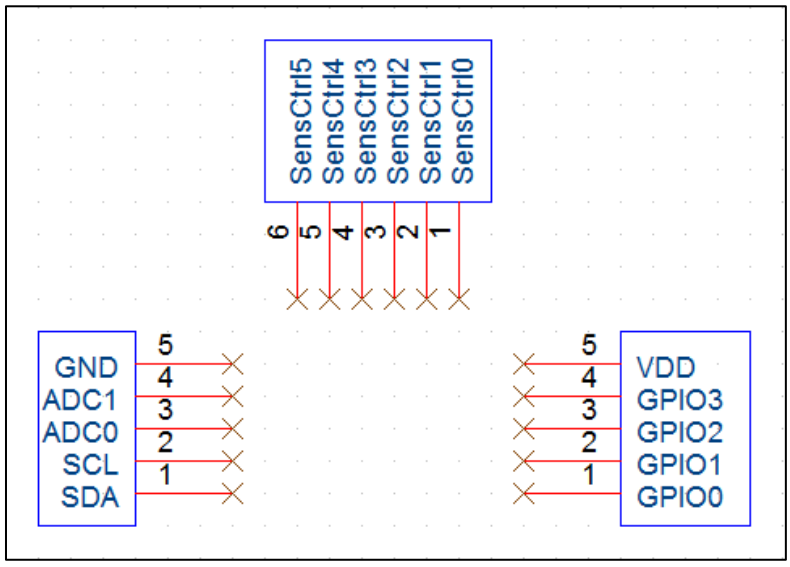
- ROHM BH1721 – Digital Ambient Light Sensor
- ROHM BH1620 – Analog Ambient Light Sensor
- ROHM BU52011HFV – Hall Switch Sensor
- ROHM BDE0600G – Analog Temperature Sensor
- LAPIS ML8511 – Analog UV Sensor
- Kionix KMX061 – Digital Accelerometer and Magnetometer

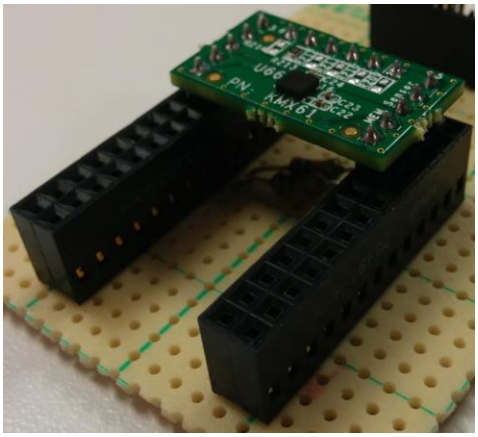
Getting Started

1. Initial Setup
    a. ROHM Sensor Platform Kit
        i. For this guide, we will be referring to the breakout boards included with the ROHM Sensor Platform Kit. Thus, the base board will not be used in this setup
    b. The following are recommended for using the Arduino Uno for this guide
        i. PC with Arduino IDE
        ii. USB Cable to power and monitor serial communication
        iii. Download the example file "***ROHM_SensorKitBreakoutBoardConnect_1-13-2015.ino***" from https://github.com/ROHMUSDC/ROHMSensorPlatformEVK. This document will refer to this code to help explain how to connect and properly calculate sensor data.

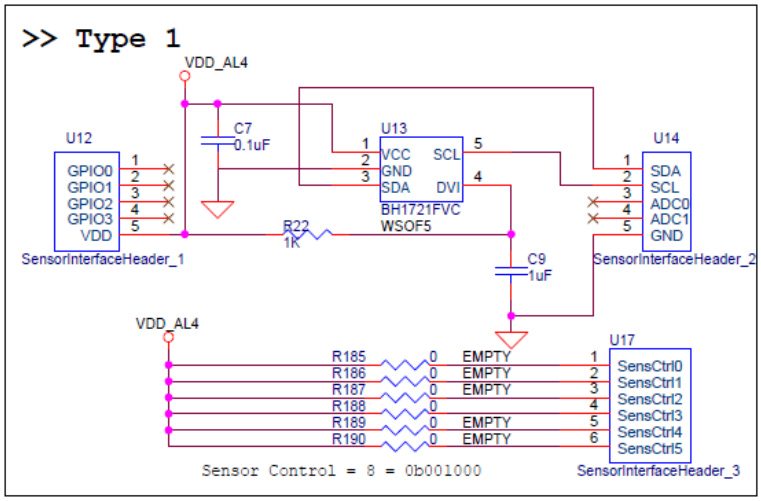
Connecting ROHM Breakout Boards to the Arduino UNO

General Information

- ROHM's Sensor Platform Breakout Boards have the same pinout across all the different sensors. If you are looking at the board from the top, the breakout board pinout will be as follows:
  - 

- The top row of pins is in place to "key" the sensor onto the base boards; however, they **do not** have any use when connecting to different platforms. We recommend mounting the breakout board on standoffs to avoid connections to these pins
  - 
- Please note that breakout board schematics can also be found at the following site:
  - https://github.com/ROHMUSDC/ROHMSensorPlatformEVK

Hardware Connection for ROHM BH1721 Digital ALS to the Arduino Uno

- ROHM BH1721 – Digital Ambient Light Sensor Breakout Board Schematic
  - 

- As seen in the schematic above, this device connects 4 pins.
  - VDD – Connect directly to Arduino 3.3V output pin on power connector
  - GND – Connect directly to Arduino GND pin on the power connector
  - I2C Connection
    - Note: User can specify the I2C pins, but in the example application, I2C is connected as follows…
      - SDA – Pin A4
      - SCL – Pin A5
    - Note: There are no pull-up resistors on the breakout board or the Arduino Uno. Thus, please be sure to add pull-up resistance to both SDA and SCL pins. 10kohm resistors were used when writing this example code

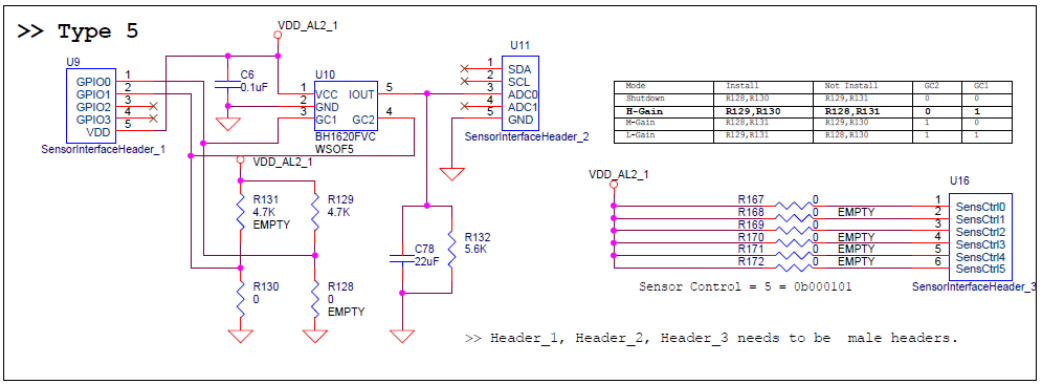
Software Explanation for ROHM BH1721 Digital ALS to the Arduino Uno

- Code Segments pertaining to this sensor can be found by defining and seeing code within the "***DigitalALS***" #ifDef statements
- Pseudo-Code Explanation
  1. Define Relevant Variables
  2. Begin setup()
     1. Initialize the I2C output
        - Note: this sensor uses the "wire" library built into Arduino to read and write send the I2C commands (http://arduino.cc/en/reference/wire)
     2. Configure the Digital ALS once
        - Send the Power ON Byte (0x01) to ALS device address (0x23)

- Send Continuous Auto-Resolution Mode Setting (0x10) to ALS device address (0x23)

3. Begin loop()
    1. Read 2 Bytes from Sensor, [0][1]
    2. Format the two bytes into an INT
        - Sensor Out = [0]<<8 | [1]
    3. Convert to Lux = Sensor Out / 1.2
    4. Format Serial Output and Return information

Hardware Connection for ROHM BH1620 Analog ALS to the Arduino Uno

- ROHM BH1620 – Analog Ambient Light Sensor Breakout Board Schematic
    - 

- As seen in the schematic above, this device connects 5 pins.
    - VDD – Connect directly to Arduino 3.3V output pin on power connector
    - GND – Connect directly to Arduino GND pin on the power connector
    - ADC0 – ADC Output for Temperature Readings
        - Note: User can specify which ADC pin, but in the example application, this part is connected as follows…
            - ADC0 – Pin A0
    - GPIO0, GPIO1
        - These pins are used to program the different mode gain options. These pins are not used in the example application.

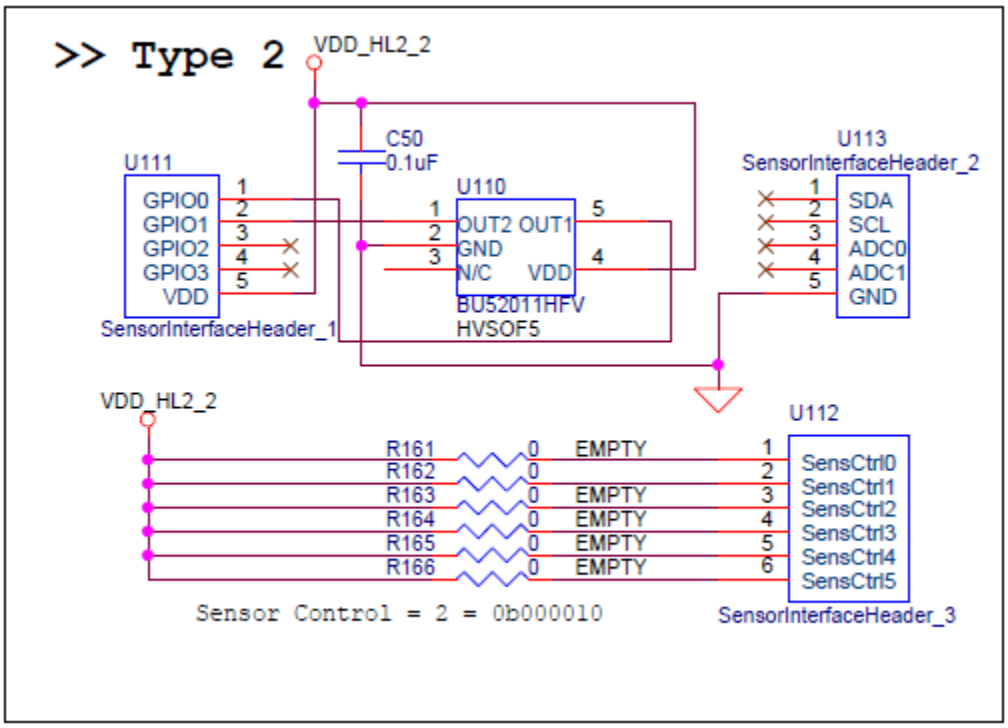
Software Explanation for ROHM BH1620 Analog ALS to the Arduino Uno

- Code Segments pertaining to this sensor can be found by defining and seeing code within the "***AnalogALS***" #ifDef statements
- Pseudo-Code Explanation
    1. Define Relevant Variables
    2. Begin loop()
        1. Read back the Analog Sensor Output from Pin A0

- Note: Default Arduino Reference voltage is 5V; however we are supplying 3.3V to the sensor. Take these values into account when performing your conversions!

2. Convert Analog Sensor Output to V, then to Lx
   - Convert to V
     - ADC_Voltage = (sensorValue / 670) * 3.3V
     - ADC_Voltage = sensorValue * (3.3V/670)
     - ADC_Voltage = sensorValue * 0.004925
   - Convert to Ev
     - Ev = ADC_Voltage/((0.57x10^-6)*R1), R1 = 5600
     - Ev = ADC_Voltage*(1/(0.57x10^-6*5600))
     - Ev = ADC_Voltage * 313.28
     - Ev = sensorValue * 0.004925 * 313.28
     - Ev = sensorValue * 1.543
3. Format Serial Output and Return information

Hardware Connection for ROHM BU52011 Hall Sensor to the Arduino Uno

- ROHM BU52011HFV – Hall Switch Sensor
  - 

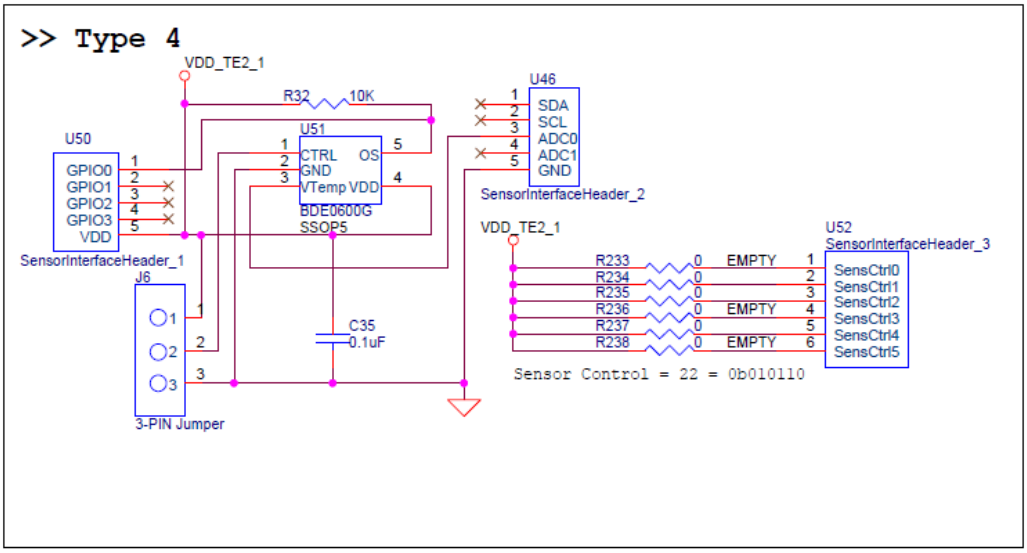
- As seen in the schematic above, this device connects 4 pins.
  - VDD – Connect directly to Arduino 3.3V output pin on power connector
  - GND – Connect directly to Arduino GND pin on the power connector

- GPIO0, GPIO1 – Indicate Hall Switch Output
  - Note: User can specify which digital input pins are used, but in the example application, this part is connected as follows…
    - GPIO0 – pin 7 (on the Arduino digital header)
    - GPIO1 – pin 8 (on the Arduino digital header)

Software Explanation for ROHM BU52011 Hall Sensor to the Arduino Uno

- Code Segments pertaining to this sensor can be found by defining and seeing code within the "***HallSen***" #ifDef statements
- Pseudo-Code Explanation
  1. Define Relevant Variables
  2. Begin setup()
     1. Setup Arduino Pins 7 and 8 and Input pins
  3. Begin loop()
     1. Perform digital reads on pins 7 and 8
        - Output will be either 1 or 0 and will indicate presence of north or south magnetic fields
     2. Format Serial Output and Return Information

Hardware Connection for ROHM BDE0600G Temp Sensor to the Arduino Uno

- ROHM BDE0600G – Analog Temperature Sensor
  - 

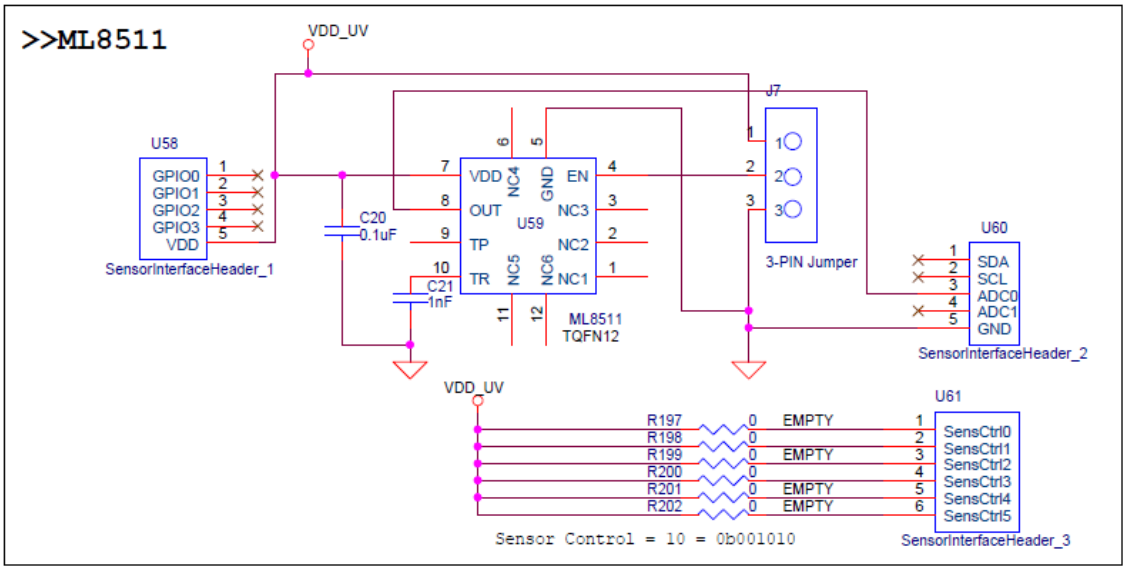
- As seen in the schematic above, this device connects 4 pins.
  - VDD – Connect directly to Arduino 3.3V output pin on power connector
  - GND – Connect directly to Arduino GND pin on the power connector
  - ADC0 – ADC Output for Temperature Readings

        - Note: User can specify which ADC pin, but in the example application, this part is connected as follows…
            - ADC0 – Pin A1
    - GPIO0
        - This pin is used to trigger the thermostat output function. This pin is not used in the example application.

Software Explanation for ROHM BDE0600G Temp Sensor to the Arduino Uno

- Code Segments pertaining to this sensor can be found by defining and seeing code within the "***AnalogTemp***" #ifDef statements
- Pseudo-Code Explanation
    1. Define Relevant Variables
    2. Begin Loop()
        1. Read back the Analog Sensor Output from Pin A1
            - Note: Default Arduino Reference voltage is 5V; however we are supplying 3.3V to the sensor. Take these values into account when performing your conversions!
        2. Convert value to V, then to Temperature reading
            - Known Values
                - Temperature Sensitivity = -10.68mV/degC
                - Temperature Sensitivity = -0.01068V/degC
                - Temp Known Point = 1.753V @ 30 degC
            - Calculation
                - ADC_Voltage = (sensorValue / 670) * 3.3V
                - ADC_Voltage = sensorValue * (3.3V/670)
                - ADC_Voltage = sensorValue * 0.004925
                - Temperature (in deg C) = (ADC_Voltage - 1.753)/(-0.01068) + 30
        3. Format Serial Output and Return Information

Hardware Connection for LAPIS ML8511 UV Sensor to the Arduino Uno

- LAPIS ML8511 – Analog UV Sensor
  - 

- As seen in the schematic above, this device connects 3 pins.
  - VDD – Connect directly to Arduino 3.3V output pin on power connector
  - GND – Connect directly to Arduino GND pin on the power connector
  - ADC0 – ADC Output for UV Sensor Output
    - Note: User can specify which ADC pin, but in the example application, this part is connected as follows…
      - ADC0 – Pin A2
  - GPIO0
    - This pin is used to trigger standby mode for the UV sensor. This pin is not used in the example application.

Software Explanation for LAPIS ML8511 UV Sensor to the Arduino Uno

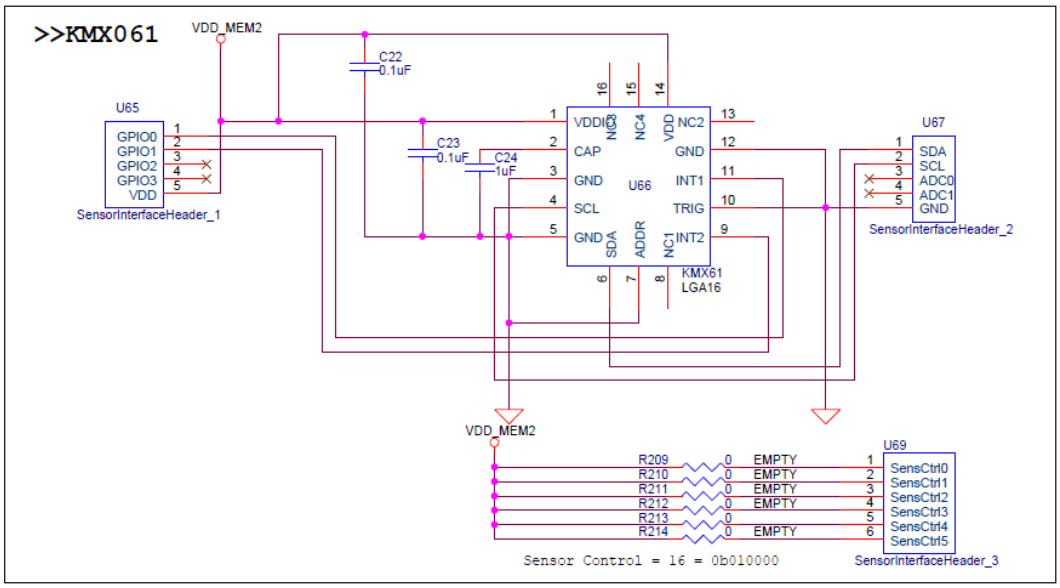
- Code Segments pertaining to this sensor can be found by defining and seeing code within the "***AnalogUV***" #ifDef statements
- Pseudo-Code Explanation
  1. Define Relevant Variables
  2. Begin Loop()
     1. Read back the Analog Sensor Output from Pin A2
        - Note: Default Arduino Reference voltage is 5V; however we are supplying 3.3V to the sensor. Take these values into account when performing your conversions!
     2. Convert value to V, then to UV Intensity
        - Known Values

- UV Sensitivity = 0.129 V/(mW/cm^2)
- Temp Known Point = 2.2V @ 10mW/cm^2

- Calculation
  - ADC_Voltage = (sensorValue / 670) * 3.3V
  - ADC_Voltage = sensorValue * (3.3V/670)
  - ADC_Voltage = sensorValue * 0.004925
  - UV Intensity (in mW/cm^2)= (ADC_Voltage - 2.2)/(0.129) + 10

3. Format Serial Output and Return Information

Hardware Connection for Kionix KMX061 Accel+Mag Sensor to the Arduino Uno

- Kionix KMX061 – Digital Accelerometer and Magnetometer
  - 

- As seen in the schematic above, this device connects 6 pins.
  - VDD – Connect directly to Arduino 3.3V output pin on power connector
  - GND – Connect directly to Arduino GND pin on the power connector
  - I2C Connection
    - Note: User can specify the I2C pins, but in the example application, I2C is connected as follows…
      - SDA – Pin 4 (on Arduino Digital Header)
      - SCL – Pin 5 (on Arduino Digital Header)
    - Note: There are no pull-up resistors on the breakout board or the Arduino Uno. Thus, please be sure to add pull-up resistance to both SDA and SCL pins. 10kohm resistors were used when writing this example code
  - GPIO0/GPIO1

- This pin is used monitor the interrupt pins on the KMX61. These pins are not used in the example application.

Software Explanation for Kionix KMX061 Accel+Mag Sensor to the Arduino Uno

- Code Segments pertaining to this sensor can be found by defining and seeing code within the "***DigitalMEMs***" #ifDef statements
- Pseudo-Code Explanation
    1. Define Relevant Variables
    2. Begin setup()
        1. Initialize the I2C output
            - Note: this sensor uses the "SoftI2CMaster" library to read and write send the I2C commands (http://arduino.cc/en/reference/wire). This was required because the "wire" library does not support the "repeated start" condition which is required for I2C reads from the KMX061
            - This Library is not built into the Arduino IDE. Resources for this can be found at the following address:
                - http://playground.arduino.cc/Main/SoftwareI2CLibrary
                - https://github.com/felias-fogg/SoftI2CMaster
        2. Configure the Accel/Mag Sensor once by performing the following reads
            - 1. Standby Register (0x29), write 0x03 (Turn Off)
            - 2. Self Test Register (0x60), write 0x00
            - 3. Control Register 1 (0x2A), write 0x13
            - 4. Control Register 2 (0x2B), write 0x00
            - 5. ODCNTL Register (0x2C), write 0x00
            - 6. Temp EN Register (0x4C), write 0x01
            - 7. Buffer CTRL Register 1 (0x78), write 0x00
            - 8. Buffer CTRL Register 2 (0x79), write 0x00
            - 9. Standby Register (0x29), write 0x0u
            - Note: Please see the KMX061 Datasheet for additional information on these registers
    3. Begin loop()
        1. Read back the Accelerometer output by reading 6 Bytes starting from address 0x0A. [0][1]...[5]
        2. Format Each of the X, Y, and Z axis acceleration
            - Xout = ([1]<<6) | ([0]>>2)
            - Yout = ([3]<<6) | ([2]>>2)
            - Zout = ([5]<<6) | ([4]>>2)

- *Note:* to save some time with conversions, please note that these outputs are meant to be unsigned 14bit values. Thus, to make it easier to convert we recommend the following…
    - Xout = (float)(([1]<<8) | ([0]))/4
    - Yout = (float)(([3]<<8) | ([2]))/4
    - Zout = (float)(([5]<<8) | ([4]))/4

3. Convert Returned Value to G
    - Axis_ValueInG = MEMS_Accel_axis / 1024
4. Read back the Mag Sensor output by reading 6 Bytes starting from address 0x12. [0][1]...[5]
5. Format Each of the X, Y, and Z axis mag data
    - Xout = ([1]<<6) | ([0]>>2)
    - Yout = ([3]<<6) | ([2]>>2)
    - Zout = ([5]<<6) | ([4]>>2)
    - *Note:* to save some time with conversions, please note that these outputs are meant to be unsigned 14bit values. Thus, to make it easier to convert we recommend the following…
        - Xout = (float)(([1]<<8) | ([0]))/4
        - Yout = (float)(([3]<<8) | ([2]))/4
        - Zout = (float)(([5]<<8) | ([4]))/4
6. Convert Returned Value to uT
    - Axis_ValueInuT = MEMS_Mag_axis / 0.146
7. Format Serial Output and Return Information

```
//Appendix A: Arduino Code

/*-----------------------------------------------------------------
 ROHM Sensor Platform Breakout Board Sensor Return Application

 This program reads the value of the connected ROHM Sensors and
 returns the sensor output using the serial port

 Created 13 January 2015
 by ROHM USDC Applications Engineering Team
-----------------------------------------------------------------*/

// ----- Debugging Definitions -----
#define AnalogALS
#define AnalogTemp
#define  AnalogUV
#define  HallSen
#define DigitalALS
#define DigitalMEMs

// ----- Included Files -----
#include <Wire.h>         //Default I2C Library

#define     SCL_PIN     4//Note that if you are using the Accel/Ma
#define   SCL_PORT   PORTD//install the "SoftI2CMaster" as "Wire" d
#define     SDA_PIN     5//References:
#define   SDA_PORT   PORTD//  http://playground.arduino.cc/Main/So
#include  <SoftI2CMaster.h>//  https://github.com/felias-fogg/Sof
#define  I2C_TIMEOUT  1000// Sets Clock Stretching up to 1sec
#define   I2C_FASTMODE   1// Sets 400kHz operating speed

// ----- Globals -----

//ADC Globals - Analog ALS, Temp, UV
int ADCpin_AnalogALS = A0;
int ADCpin_AnalogTemp = A1;
int ADCpin_AnalogUV = A2;
int sensorValue = 0;
```

```
float sensorConvert = 0;

//Digital Input Globals - Hall Sensor
int Hall_Out0 = 0;
int Hall_Out1 = 0;

//I2C globals (using wire library) - Digital ALS
int DigitalALS_DeviceAddress = 0x23;
unsigned char ALS_highByte = 0;
unsigned char ALS_lowByte = 0;
unsigned int ALSReturn = 0;

//I2C globals (using SoftI2CMaster libary) - MEMs Kionix Sensor
int I2C_check = 0;
int  DigitalMEMS_DeviceAddress  =  0x1C;//this is the 8bit address,

  //Accel Portion
int MEMS_Accel_Xout_highByte = 0;
int MEMS_Accel_Xout_lowByte = 0;
int MEMS_Accel_Yout_highByte = 0;
int MEMS_Accel_Yout_lowByte = 0;
int MEMS_Accel_Zout_highByte = 0;
int MEMS_Accel_Zout_lowByte = 0;
int MEMS_Accel_Xout = 0;
int MEMS_Accel_Yout = 0;
int MEMS_Accel_Zout = 0;
float MEMS_Accel_Conv_Xout = 0;
float MEMS_Accel_Conv_Yout = 0;
float MEMS_Accel_Conv_Zout = 0;

  //Mag Sensor Portion
int MEMS_Mag_Xout_highByte = 0;
int MEMS_Mag_Xout_lowByte = 0;
int MEMS_Mag_Yout_highByte = 0;
int MEMS_Mag_Yout_lowByte = 0;
int MEMS_Mag_Zout_highByte = 0;
int MEMS_Mag_Zout_lowByte = 0;
int MEMS_Mag_Xout = 0;
```

```
int MEMS_Mag_Yout = 0;
int MEMS_Mag_Zout = 0;
float MEMS_Mag_Conv_Xout = 0;
float MEMS_Mag_Conv_Yout = 0;
float MEMS_Mag_Conv_Zout = 0;

void setup()
{
  Wire.begin();        // start I2C functionality
  Serial.begin(9600);  // start serial port at 9600 bps
  while (!Serial) {
     ;// wait for serial port to connect. Needed for Leonardo only
  }

  pinMode(13, OUTPUT);   //Setup for the LED on Board
  pinMode(7, INPUT_PULLUP);
  pinMode(8, INPUT_PULLUP);

 //----- Start Initialization for BH1721 Digital ALS -----
 // 1. Send 1 Power On Byte (0x01u)
 // 2. Send 1 Continuous Auto-Resolution Mode Byte (0x10u or 0x20u)
 #ifdef DigitalALS
 Wire.beginTransmission(DigitalALS_DeviceAddress);
 Wire.write(0x01u);
 Wire.endTransmission();
 delay(100);

 Wire.beginTransmission(DigitalALS_DeviceAddress);
 Wire.write(0x10u);
 Wire.endTransmission();
 delay(100);
 #endif
 //----- END Initialization for BH1721 Digital ALS -----

 //----- Start Initialization for KMX061 Digital Accel/Mag Sensor -
 #ifdef DigitalMEMs
  // 1. Standby Register (0x29), write 0x03 (Turn Off)
  // 2. Self Test Register (0x60), write 0x00
```

```
// 3. Control Register 1 (0x2A), write 0x13
// 4. Control Register 2 (0x2B), write 0x00
// 5. ODCNTL Register (0x2C), write 0x00
// 6. Temp EN Register (0x4C), write 0x01
// 7. Buffer CTRL Register 1 (0x78), write 0x00
// 8. Buffer CTRL Register 2 (0x79), write 0x00
// 9. Standby Register (0x29), write 0x0u

I2C_check = i2c_init();
if(I2C_check == false){
   while(1){
     Serial.write("I2C Init Failed (SDA or SCL may not be pulled
     Serial.write(0x0A); //Print Line Feed
     Serial.write(0x0D); //Print Carrage Return
     delay(500);
   }
}

 i2c_start(DigitalMEMS_DeviceAddress);//This needs the 8 bit add
i2c_write(0x29);
i2c_write(0x03);
i2c_stop();

 i2c_start(DigitalMEMS_DeviceAddress);//This needs the 8 bit add
i2c_write(0x60);
i2c_write(0x00);
i2c_stop();

 i2c_start(DigitalMEMS_DeviceAddress);//This needs the 8 bit add
i2c_write(0x2A);
i2c_write(0x13);
i2c_stop();

 i2c_start(DigitalMEMS_DeviceAddress);//This needs the 8 bit add
i2c_write(0x2B);
i2c_write(0x00);
i2c_stop();
```

```
   i2c_start(DigitalMEMS_DeviceAddress);//This needs the 8 bit add
  i2c_write(0x2C);
  i2c_write(0x00);
  i2c_stop();

   i2c_start(DigitalMEMS_DeviceAddress);//This needs the 8 bit add
  i2c_write(0x4C);
  i2c_write(0x01);
  i2c_stop();

   i2c_start(DigitalMEMS_DeviceAddress);//This needs the 8 bit add
  i2c_write(0x78);
  i2c_write(0x00);
  i2c_stop();

   i2c_start(DigitalMEMS_DeviceAddress);//This needs the 8 bit add
  i2c_write(0x79);
  i2c_write(0x00);
  i2c_stop();

   i2c_start(DigitalMEMS_DeviceAddress);//This needs the 8 bit add
  i2c_write(0x29);
  i2c_write(0x00);
  i2c_stop();

 #endif
 //----- END Initialization for KMX061 Digital Accel/Mag Sensor ---
}

void loop()
{

  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage
  delay(250);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the volta
  delay(250);              // wait for a second

  //---------- Start Code for Reading BH1620FVC Analog Ambient Ligh
```

```
 #ifdef AnalogALS

//----- Start ADC Read from Port A0 ----

//Notes on Arduino ADC
//Arduino uses an 5V ADC Reference Voltage; thus, we will need t
//to 3.3V Levels to safely operate the sensors
//5V = 1024, 3.3V = 670
 sensorValue =analogRead(ADCpin_AnalogALS);
Serial.write("ADC Raw Value Return = ");
Serial.print(sensorValue);
Serial.write(0x0A); //Print Line Feed
Serial.write(0x0D); //Print Carrage Return

//Calculations for Analog ALS - BH1620FVC
// Note: Device's default configuration is in "High Gain Mode"
// Note: Default R1 is 5.6kohm
// Math:  ADC_Voltage = (sensorValue / 670) * 3.3V             /
//       ADC_Voltage = sensorValue * (3.3V/670)
//       ADC_Voltage = sensorValue * 0.004925
//       Ev = ADC_Voltage/((0.57x10^-6)*R1), R1 = 5600        /
//       Ev = ADC_Voltage*(1/(0.57x10^-6*5600))
//       Ev = ADC_Voltage * 313.28
//       Ev = sensorValue * 0.004925 * 313.28
//       Ev = sensorValue * 1.543

sensorConvert = (float)sensorValue*1.543;
Serial.write("Analog ALS Converted Return = ");
Serial.print(sensorConvert);
Serial.write(" lx");
Serial.write(0x0A); //Print Line Feed
Serial.write(0x0D); //Print Carrage Return
#endif
//---------- End Code for Reading BH1620FVC Analog Ambient Light

//---------- Start Code for Reading BDE0600G Analog Temperature S
#ifdef AnalogTemp
```

```
//----- Start ADC Read from Port A1 ----
//Notes on Arduino ADC
//Arduino uses an 5V ADC Reference Voltage; thus, we will need t
//to 3.3V Levels to safely operate the sensors
//5V = 1024, 3.3V = 670
 sensorValue =analogRead(ADCpin_AnalogTemp);
Serial.write("ADC Raw Value Return = ");
Serial.print(sensorValue);
Serial.write(0x0A); //Print Line Feed
Serial.write(0x0D); //Print Carrage Return

//Calculations for Analog Temp Sensor - BDE0600G
// Temperature Sensitivity = -10.68mV/degC
// Temperature Sensitivity = -0.01068V/degC
// Temperature Sensitivity = -93.63degC/V
// Temp Known Point = 1.753V @ 30DegC

// Math:  ADC_Voltage = (sensorValue / 670) * 3.3V             /
//        ADC_Voltage = sensorValue * (3.3V/670)
//        ADC_Voltage = sensorValue * 0.004925
//        Temperature = (ADC_Voltage - 1.753)/(-0.01068) + 30

sensorConvert = (float)sensorValue * 0.004925;
sensorConvert = (sensorConvert - 1.753)/(-0.01068)+30;
Serial.write("Analog Temp Sensor Converted Return = ");
Serial.print(sensorConvert);
Serial.write(" degC");
Serial.write(0x0A); //Print Line Feed
Serial.write(0x0D); //Print Carrage Return
#endif
//---------- End Code for Reading BDE0600G Analog Temperature Sen

//---------- Start Code for Reading ML8511 Analog UV Sensor -----
#ifdef AnalogUV

//----- Start ADC Read from Port A2 ----
//Notes on Arduino ADC
//Arduino uses an 5V ADC Reference Voltage; thus, we will need t
```

```
//to 3.3V Levels to safely operate the sensors
//5V = 1024, 3.3V = 670
 sensorValue =analogRead(ADCpin_AnalogUV);
Serial.write("ADC Raw Value Return = ");
Serial.print(sensorValue);
Serial.write(0x0A); //Print Line Feed
Serial.write(0x0D); //Print Carrage Return

//Calculations for UV Sensor - ML8511
//Known Point = 2.2V @ 10mW/cm2
//Rate = 0.129

// Math:  ADC_Voltage = (sensorValue / 670) * 3.3V             /
//        ADC_Voltage = sensorValue * (3.3V/670)
//        ADC_Voltage = sensorValue * 0.004925
//        UV Intensity = (ADC_Voltage - 2.2)/(0.129) + 10   //Con

sensorConvert = (float)sensorValue * 0.004925;
sensorConvert = (sensorConvert - 2.2)/(0.129)+10;
Serial.write("Analog UV Sensor Converted Return = ");
Serial.print(sensorConvert);
Serial.write(" mW/cm2");
Serial.write(0x0A); //Print Line Feed
Serial.write(0x0D); //Print Carrage Return
#endif
//---------- End Code for Reading ML8511 Analog UV Sensor -------

//---------- Start Code for Reading BU52011HFV Hall Sensor ------
#ifdef HallSen
//Hardware Connection
//For the Hall Sensor, we only need to monitor digital inputs for
//Connect Pin1 of Hall Sensor Header U111 to Arduino Pin7
//Connect Pin2 of Hall Sensor Header U111 to Arduino Pin8
 Hall_Out0 =digitalRead(7);
 Hall_Out1 =digitalRead(8);
Serial.write("HallOut0 = ");
Serial.print(Hall_Out0);
Serial.write(0x0A); //Print Line Feed
```

```
Serial.write(0x0D); //Print Carrage Return
Serial.write("HallOut1 = ");
Serial.print(Hall_Out1);
Serial.write(0x0A); //Print Line Feed
Serial.write(0x0D); //Print Carrage Return
#endif
//---------- End Code for Reading BU52011HFV Hall Sensor --------

//---------- Start Code for Reading BH1721 Digital Ambient Light
#ifdef DigitalALS
// -- Notes on Arduino I2C Connection --
//I2C is built in using using the "wire" library
//Uno, Ethernet        A4 (SDA), A5 (SCL) [UNO - THIS is the plat
//Mega2560             20 (SDA), 21 (SCL)
//Leonardo             2  (SDA), 3  (SCL)
//Due                 20 (SDA), 21 (SCL), SDA1, SCL1

// -- Notes on ROHM BH1721 Digital ALS --
//Device Address = 0x23u
//Intialization Routines (See Setup Function Above)
// 1. Send 1 Power On Byte (0x01u)
// 2. Send 1 Continuous Auto-Resolution Mode Byte (0x10u or 0x20u
//Main Loop Routines
// 1. Read 2 Bytes from Sensor, [0][1]
// 2. Sensor Out = [0]<<8 | [1]
// 3. Convert to Lux = Sensor Out / 1.2

Wire.requestFrom(DigitalALS_DeviceAddress, 2);
 ALS_highByte =Wire.read();  // First Byte Read back is the high
 ALS_lowByte =Wire.read();   // Second Byte Read back is the low

 ALSReturn = ALS_highByte<<8 | ALS_lowByte;//Store char values i
Serial.write("Digital ALS Raw = ");
Serial.print(ALSReturn);
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return

 ALSReturn = ALSReturn / 1.2;//Convert to Lx
```

```
Serial.write("Digital ALS Converted Value = ");
Serial.print(ALSReturn);  // print the character
Serial.write(" Lx");
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return
#endif
//---------- End Code for Reading BH1721 Digital Ambient Light S

//---------- Start Code for Reading KMX61 Kionix Accelerometer+M
#ifdef DigitalMEMs
// -- Notes on Arduino I2C Connection --
//I2C is built in using using the "wire" library
//Uno, Ethernet        A4 (SDA), A5 (SCL) [UNO - THIS is the plat
//Mega2560             20 (SDA), 21 (SCL)
//Leonardo             2  (SDA), 3  (SCL)
//Due                 20 (SDA), 21 (SCL), SDA1, SCL1

// -- Notes on ROHM KMX61 Accel/Mag Sensor --
//Device Address = 0x0Eu
//12 Bit Return Value

//Intialization Routines (See Setup function above)
// 1. Standby Register (0x29), write 0x03 (Turn Off)
// 2. Self Test Register (0x60), write 0x00
// 3. Control Register 1 (0x2A), write 0x13
// 4. Control Register 2 (0x2B), write 0x00
// 5. ODCNTL Register (0x2C), write 0x00
// 6. Temp EN Register (0x4C), write 0x01
// 7. Buffer CTRL Register 1 (0x78), write 0x00
// 8. Buffer CTRL Register 2 (0x79), write 0x00
// 9. Standby Register (0x29), write 0x0u

//Main Loop Routines
// 1. Read 6 Bytes starting from address 0x0A.  These will be the
// 2. Xout = ([1]<<6) | ([0]>>2)
// 3. Yout = ([3]<<6) | ([2]>>2)
// 4. Zout = ([5]<<6) | ([4]>>2)
// 5. Read 6 Bytes starting from addres 0x12. These will be the
```

```
 // 6. Xout = ([1]<<6) | ([0]>>2)
 // 7. Yout = ([3]<<6) | ([2]>>2)
 // 8. Zout = ([5]<<6) | ([4]>>2)

 // Start Getting Data from Accel
i2c_start(DigitalMEMS_DeviceAddress);
 i2c_write(0x0A);
  i2c_rep_start(DigitalMEMS_DeviceAddress | 1);// Or-ed with "1"
 MEMS_Accel_Xout_lowByte = i2c_read(false);
 MEMS_Accel_Xout_highByte = i2c_read(false);
 MEMS_Accel_Yout_lowByte = i2c_read(false);
 MEMS_Accel_Yout_highByte = i2c_read(false);
 MEMS_Accel_Zout_lowByte = i2c_read(false);
 MEMS_Accel_Zout_highByte = i2c_read(true);
 i2c_stop();

 //Note: The highbyte and low byte return a 14bit value, dropping
 //     However, because we need the signed value, we will adjust
MEMS_Accel_Xout = (MEMS_Accel_Xout_highByte<<8) | (MEMS_Accel_Xou
MEMS_Accel_Yout = (MEMS_Accel_Yout_highByte<<8) | (MEMS_Accel_You
MEMS_Accel_Zout = (MEMS_Accel_Zout_highByte<<8) | (MEMS_Accel_Zou

 //Note: Conversion to G is as follows:
 //     Axis_ValueInG = MEMS_Accel_axis / 1024
 //     However, since we did not remove the LSB previously, we n
 //     Thus, we will divide the output by 4095 (1024*4) to conve
MEMS_Accel_Conv_Xout = (float)MEMS_Accel_Xout/4096;
MEMS_Accel_Conv_Yout = (float)MEMS_Accel_Yout/4096;
MEMS_Accel_Conv_Zout = (float)MEMS_Accel_Zout/4096;

 /*  //Uncomment if you want to see the Raw Sensor Output
 Serial.write("Digital MEMS Raw Accel Xout = ");
  Serial.print(MEMS_Accel_Xout);
 Serial.write(0x0A);  //Print Line Feed
 Serial.write(0x0D);  //Print Carrage Return
 Serial.write("Digital MEMS Raw Accel Yout = ");
  Serial.print(MEMS_Accel_Yout);
 Serial.write(0x0A);  //Print Line Feed
```

```
 Serial.write(0x0D);  //Print Carrage Return
 Serial.write("Digital MEMS Raw Accel Zout = ");
Serial.print(MEMS_Accel_Zout);
 Serial.write(0x0A);  //Print Line Feed
 Serial.write(0x0D);  //Print Carrage Return
 */

 Serial.write("Digital MEMS Accel Xout = ");
 Serial.print(MEMS_Accel_Conv_Xout);
 Serial.write(0x0A);  //Print Line Feed
 Serial.write(0x0D);  //Print Carrage Return
 Serial.write("Digital MEMS Accel Yout = ");
 Serial.print(MEMS_Accel_Conv_Yout);
 Serial.write(0x0A);  //Print Line Feed
 Serial.write(0x0D);  //Print  Carrage  Return
 Serial.write("Digital MEMS Accel Zout = ");
 Serial.print(MEMS_Accel_Conv_Zout);
 Serial.write(0x0A);  //Print Line Feed
 Serial.write(0x0D);  //Print  Carrage  Return

 // Start Getting Data from Mag Sensor
i2c_start(DigitalMEMS_DeviceAddress);
 i2c_write(0x12);
  i2c_rep_start(DigitalMEMS_DeviceAddress | 1);// Or-ed with "1"
 MEMS_Mag_Xout_lowByte = i2c_read(false);
 MEMS_Mag_Xout_highByte = i2c_read(false);
 MEMS_Mag_Yout_lowByte = i2c_read(false);
 MEMS_Mag_Yout_highByte = i2c_read(false);
 MEMS_Mag_Zout_lowByte = i2c_read(false);
 MEMS_Mag_Zout_highByte = i2c_read(true);
 i2c_stop();

 //Note: The highbyte and low byte return a 14bit value, dropping
 //     However, because we need the signed value, we will adjust
 MEMS_Mag_Xout = (MEMS_Mag_Xout_highByte<<8) | (MEMS_Mag_Xout_lowB
 MEMS_Mag_Yout = (MEMS_Mag_Yout_highByte<<8) | (MEMS_Mag_Yout_lowB
 MEMS_Mag_Zout = (MEMS_Mag_Zout_highByte<<8) | (MEMS_Mag_Zout_lowB
```

```
 //Note: Conversion to G is as follows:
 //      Axis_ValueInG = MEMS_Accel_axis / 1024
 //      However, since we did not remove the LSB previously, we n
 //      Thus, we will divide the output by 4095 (1024*4) to conve
MEMS_Mag_Conv_Xout = (float)MEMS_Mag_Xout*0.146;
MEMS_Mag_Conv_Yout = (float)MEMS_Mag_Yout*0.146;
MEMS_Mag_Conv_Zout = (float)MEMS_Mag_Zout*0.146;

 /*  //Uncomment if you want to see the Raw Sensor Output
 Serial.write("Digital MEMS Raw Mag Xout = ");
  Serial.print(MEMS_Mag_Xout);
 Serial.write(0x0A);  //Print Line Feed
 Serial.write(0x0D);  //Print Carrage Return
 Serial.write("Digital MEMS Raw Mag Yout = ");
  Serial.print(MEMS_Mag_Yout);
 Serial.write(0x0A);  //Print Line Feed
  Serial.write(0x0D);   //Print Carrage Return
 Serial.write("Digital MEMS Raw Mag Zout = ");
Serial.print(MEMS_Mag_Zout);
 Serial.write(0x0A);  //Print Line Feed
 Serial.write(0x0D);  //Print Carrage Return
 */

 Serial.write("Digital MEMS Mag Xout = ");
 Serial.print(MEMS_Mag_Conv_Xout);
 Serial.write(0x0A);  //Print Line Feed
 Serial.write(0x0D);  //Print Carrage Return
 Serial.write("Digital MEMS Mag Yout = ");
 Serial.print(MEMS_Mag_Conv_Yout);
 Serial.write(0x0A);  //Print Line Feed
 Serial.write(0x0D);  //Print  Carrage  Return
 Serial.write("Digital MEMS Mag Zout = ");
 Serial.print(MEMS_Mag_Conv_Zout);
 Serial.write(0x0A);  //Print Line Feed
 Serial.write(0x0D);  //Print  Carrage  Return

 #endif
 //---------- END Code for Reading KMX61 Kionix Accelerometer+M
```

```
  Serial.write(0x0A); //Print Line Feed
  Serial.write(0x0D); //Print Carrage Return
}

void I2C_CheckACK()
{
  if(I2C_check == false){
     while(1){
       Serial.write("No ACK!");
       Serial.write(0x0A); //Print Line Feed
       Serial.write(0x0D); //Print Carrage Return
       delay(500);
     }
  }
}
```