3 Degrees Accelerometer using MMA8452Q

https://www.sparkfun.com/products/10955

This breakout board makes it easy to use the tiny MMA8452Q accelerometer in your project. The MMA8452Q is a smart low-power, three-axis, capacitive micro-machined accelerometer with 12 bits of resolution. This accelerometer is packed with embedded functions with flexible user programmable options, configurable to two interrupt pins. Embedded interrupt functions allow for overall power savings relieving the host processor from continuously polling data.

The MMA8452Q has user selectable full scales of ±2g/±4g/±8g with high pass filtered data as well as non filtered data available real-time. The device can be configured to generate inertial wake-up interrupt signals from any combination of the configurable embedded functions allowing the MMA8452Q to monitor events and remain in a low power mode during periods of inactivity.

This board breaks out the ground, power, I2C and two external interrupt pins.

Not sure which accelerometer is right for you? Our Accelerometer and Gyro Buying Guide might help!

Various type of Accelerometers:
https://www.sparkfun.com/pages/accel_gyro_guide

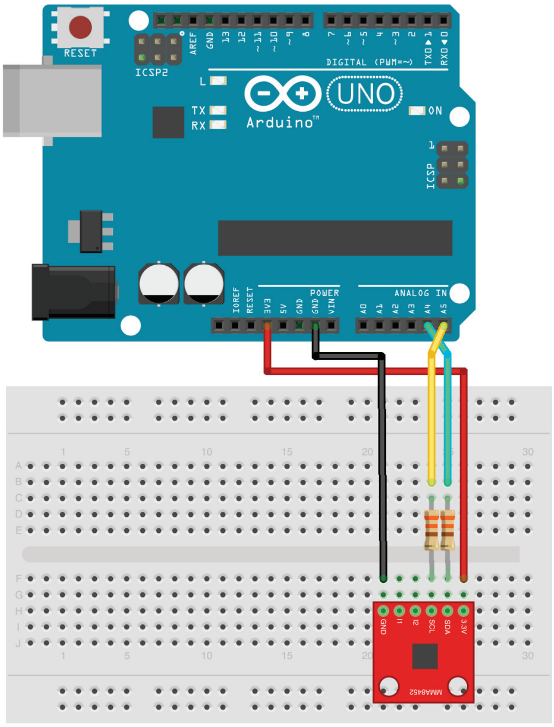
HARDWARE REQUIRED

Arduino Board
(1) MMA8452 Accelerometer
2 330 Ohm Resistors

CIRCUIT

Hardware setup:
MMA8452 Breakout ----------------------- Arduino
3.3V -- 3.3V
SDA -----------^^(330 Resistor)^^-------- A4
SCL ------------^^(330 Resistor)^^-------- A5
GND -- GND

IMAGE



Made with Fritzing.org

ARDUINO COOKBOOK
by Angelos Floros

CODE

```
/*
 MMA8452Q Basic Example Code
 Nathan Seidle
 SparkFun Electronics
 November 5, 2012

 License: This code is public domain but you buy me a beer if you use this and we meet someday (Beerware license).

 This example code shows how to read the X/Y/Z accelerations and basic functions of the MMA5842. It leaves out
 all the neat features this IC is capable of (tap, orientation, and inerrupts) and just displays X/Y/Z. See  the advanced example code to see more features.

 Hardware setup:
 MMA8452 Breakout ------------ Arduino
 3.3V --------------------- 3.3V
 SDA -------^^(330)^^------- A4
 SCL -------^^(330)^^------- A5
 GND ---------------------- GND

 The MMA8452 is 3.3V so we recommend using 330 or 1k resistors between a 5V Arduino and the MMA8452 breakout.

 The MMA8452 has built in pull-up resistors for I2C so you do not need additional pull-ups.
 */
```

```
#include <Wire.h> // Used for I2C

// The SparkFun breakout board defaults to 1,
// set to 0 if SA0 jumper on the bottom of the board is set

#define MMA8452_ADDRESS 0x1D  // 0x1D if SA0 is high, 0x1C if low

//Define a few of the registers that we will be accessing on the MMA8452
#define OUT_X_MSB 0x01
#define XYZ_DATA_CFG  0x0E
#define WHO_AM_I   0x0D
#define CTRL_REG1  0x2A
#define GSCALE 2 // Sets full-scale range to +/-2, 4, or 8g. Used to calc real g values.

void setup()
{
  Serial.begin(57600);
  Serial.println(“MMA8452 Basic Example”);

  Wire.begin(); //Join the bus as a master
  initMMA8452(); //Test and intialize the MMA8452
}

void loop()
{
  int accelCount[3];  // Stores the 12-bit signed value
  readAccelData(accelCount);  // Read the x/y/z adc values

  // Now we’ll calculate the accleration value into actual g’s
  float accelG[3];  // Stores the real accel value in g’s
  for (int i = 0 ; i < 3 ; i++)
  {
    accelG[i] = (float) accelCount[i] / ((1<<12)/(2*GSCALE));
// get actual g value, this depends on scale being set
  }  for (int i = 0 ; i < 3 ; i++)   // Print out values
  {
    Serial.print(accelG[i], 4);  // Print g values
    Serial.print(“\t”);  // tabs in between axes
  }=
  Serial.println();
  delay(10);  // Delay here for visibility
}

void readAccelData(int *destination)
{
  byte rawData[6];  // x/y/z accel register data stored here
  readRegisters(OUT_X_MSB, 6, rawData);  // Read the six raw data registers into data array
    for (int i = 0; i < 3 ; i++) // Loop to calculate 12-bit ADC and g value for each axis
  {
    int gCount = (rawData[i*2] << 8) | rawData[(i*2)+1];  //Combine the two 8 bit registers into one 12-bit number
    gCount >>= 4; //The registers are left align, here we right align the 12-bit integer

    // If the number is negative, we have to make it so manually (no 12-bit data type)

    if (rawData[i*2] > 0x7F)
    {
      gCount = ~gCount + 1;
      gCount *= -1;  // Transform into negative 2’s complement #
    }
    destination[i] = gCount; //Record this gCount into the 3 int array
  }
}
```

```
// Initialize the MMA8452 registers
// See the many application notes for more info on setting all of these registers:
// http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MMA8452Q

void initMMA8452()
{
  byte c = readRegister(WHO_AM_I);  // Read WHO_AM_I register
  if (c == 0x2A) // WHO_AM_I should always be 0x2A
  {
    Serial.println(“MMA8452Q is online...”);
  }
  else
  {
    Serial.print(“Could not connect to MMA8452Q: 0x”);
    Serial.println(c, HEX);
    while(1) ; // Loop forever if communication doesn’t happen
  }

  MMA8452Standby();  // Must be in standby to change registers

  // Set up the full scale range to 2, 4, or 8g.
  byte fsr = GSCALE;
  if(fsr > 8) fsr = 8; //Easy error check
  fsr >>= 2; // Neat trick, see page 22. 00 = 2G, 01 = 4A, 10 = 8G
  writeRegister(XYZ_DATA_CFG, fsr);

  //The default data rate is 800Hz and we don’t modify it in this example code

  MMA8452Active();  // Set to active to start reading
}

// Sets the MMA8452 to standby mode. It must be in standby to change most register settings
void MMA8452Standby()
{
  byte c = readRegister(CTRL_REG1);
  writeRegister(CTRL_REG1, c & ~(0x01)); //Clear the active bit to go into standby
}

// Sets the MMA8452 to active mode. Needs to be in this mode to output data
void MMA8452Active()
{
  byte c = readRegister(CTRL_REG1);
  writeRegister(CTRL_REG1, c | 0x01); //Set the active bit to begin detection
}

// Read bytesToRead sequentially, starting at addressToRead into the dest byte array
void readRegisters(byte addressToRead, int bytesToRead, byte * dest)
{
  Wire.beginTransmission(MMA8452_ADDRESS);
  Wire.write(addressToRead);
  Wire.endTransmission(false); //endTransmission but keep the connection active

  Wire.requestFrom(MMA8452_ADDRESS, bytesToRead); //Ask for bytes, once done, bus is released by default

  while(Wire.available() < bytesToRead); //Hang out until we get the # of bytes we expect

  for(int x = 0 ; x < bytesToRead ; x++)
    dest[x] = Wire.read();
}

// Read a single byte from addressToRead and return it as a byte
byte readRegister(byte addressToRead)
{
  Wire.beginTransmission(MMA8452_ADDRESS);
  Wire.write(addressToRead);
  Wire.endTransmission(false); //endTransmission but keep the connection active

  Wire.requestFrom(MMA8452_ADDRESS, 1); //Ask for 1 byte, once done, bus is released by default

  while(!Wire.available()) ; //Wait for the data to come back
  return Wire.read(); //Return this one byte
}

// Writes a single byte (dataToWrite) into addressToWrite
void writeRegister(byte addressToWrite, byte dataToWrite)
{
  Wire.beginTransmission(MMA8452_ADDRESS);
  Wire.write(addressToWrite);
  Wire.write(dataToWrite);
  Wire.endTransmission(); //Stop transmitting
}
```