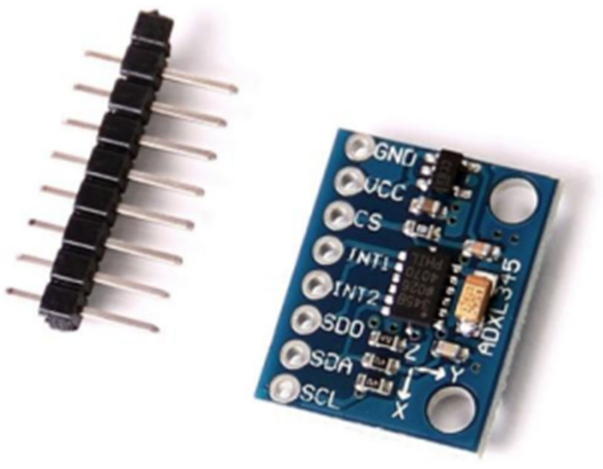
ARDUINO SENSOR ACCELEROMETER

User Manual



Description:

The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 g. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I2C digital interface. The ADXL345 is well suited to measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 mg/LSB) enables measurement of inclination changes less than 1.0°. Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion and if the acceleration on any axis exceeds a user-set level. Tap sensing detects single and double taps. Free-fall sensing detects if the device is falling. These functions can be mapped to one of two interrupt output pins. An integrated, patent pending 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor intervention. Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

Features:

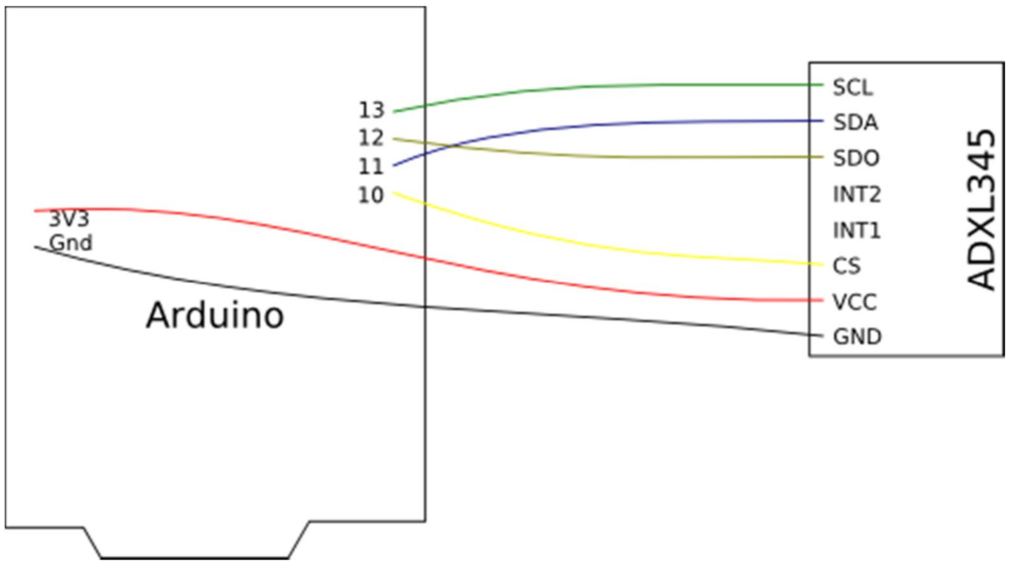
•2.0-3.6VDC Supply Voltage
•Ultra Low Power: 40uA in measurement mode, 0.1uA in standby@ 2.5V
•Tap/Double Tap Detection
•Free-Fall Detection
•SPI and I2C interfaces

Hooking it Up:

This section of the guide illustrates how to connect an Arduino to the ADXL345 breakout board. The following is a table describing which pins on the Arduino should be connected to the pins on the accelerometer:

Arduino Pin	ADXL345 Pin
10	CS
11	SDA
12	SDO
13	SCL
3V3	VCC
Gnd	GND

Here is a diagram in case you like pictures!



Beginner Sample Code:

Let's look at a sketch to get the ADXL345 up and running with an Arduino. You can download the sketch in its entirety here. Below we'll examine what the different sections of this code does.

```
//Add the SPI library so we can communicate with the ADXL345 sensor
#include <SPI.h>

//Assign the Chip Select signal to pin 10.
int CS=10;

//This is a list of some of the registers available on the ADXL345.
//To learn more about these and the rest of the registers on the ADXL345, read the datasheet!
char POWER_CTL = 0x2D;    //Power Control Register
char DATA_FORMAT = 0x31;
char DATAX0 = 0x32;   //X-Axis Data 0
char DATAX1 = 0x33;   //X-Axis Data 1
char DATAY0 = 0x34;   //Y-Axis Data 0
char DATAY1 = 0x35;   //Y-Axis Data 1
char DATAZ0 = 0x36;   //Z-Axis Data 0
char DATAZ1 = 0x37;   //Z-Axis Data 1

//This buffer will hold values read from the ADXL345 registers.
char values[10];
```

```
//These variables will be used to hold the x,y and z axis accelerometer values.
int x,y,z;
```

This is the initialization section of the sketch. It's pretty basic really. Here's what's happening.

1. The SPI.h library is added to the sketch. SPI is a communication protocol used by Arduino to communicate to the ADLX345.
2. A variable named CS is created to store the pin number of the Chip Select signal.
3. Variables are created for several of the ADXL345 registers. These variables store the address of the indicated registers and are used to set and retrieve values from the accelerometer. The datasheet shows all of the registers available on the ADXL345 and their addresses.
4. Variables are created that will hold information that has been retrieved from the accelerometer.

```
void setup(){
 //Initiate an SPI communication instance.
 SPI.begin();
 //Configure the SPI connection for the ADXL345.
 SPI.setDataMode(SPI_MODE3);
 //Create a serial connection to display the data on the terminal.
 Serial.begin(9600);

 //Set up the Chip Select pin to be an output from the Arduino.
 pinMode(CS, OUTPUT);
 //Before communication starts, the Chip Select pin needs to be set high.
 digitalWrite(CS, HIGH);

 //Put the ADXL345 into +/- 4G range by writing the value 0x01 to the DATA_FORMAT register.
 writeRegister(DATA_FORMAT, 0x01);
 //Put the ADXL345 into Measurement Mode by writing 0x08 to the POWER_CTL register.
 writeRegister(POWER_CTL, 0x08);  //Measurement mode
}

void loop(){
 //Reading 6 bytes of data starting at register DATAX0 will retrieve the x,y and z acceleration values f
rom the ADXL345.
 //The results of the read operation will get stored to the values[] buffer.
 readRegister(DATAX0, 6, values);
 //The ADXL345 gives 10-bit acceleration values, but they are stored as bytes (8-bits). To get the full value, two bytes must be combined for each axis.
 //The X value is stored in values[0] and values[1].
 x = ((int)values[1]<<8)|(int)values[0];
 //The Y value is stored in values[2] and values[3].
 y = ((int)values[3]<<8)|(int)values[2];
 //The Z value is stored in values[4] and values[5].
 z = ((int)values[5]<<8)|(int)values[4];

 //Print the results to the terminal.
 Serial.print(x, DEC);
 Serial.print(',');
 Serial.print(y, DEC);
 Serial.print(',');
 Serial.println(z, DEC);
 delay(10);
}
```

The main section of the sketch is split into two parts: the setup and the loop. The setup section is used to configure different aspects of the Arduino to communicate with the ADXL345. At the end of the setup section two functions are used to put the accelerometer into the mode we want to use (+/-4g detection, and enable measurement mode). The functions will be discussed in the next section of the sketch.

The loop does most of the work in the sketch. Here's what's going on in the loop:

1. Arduino reads 6 register values from the ADXL345 using the readRegister function. This command reads the x, y and z acceleration values from the accelerometer and stores them in the buffer named "values" that was created in the initialization section of the sketch.
2. When values are read from the accelerometer they are returned in bytes. A byte is only 8 bits, but the accelerometer reports acceleration with up to 10 bits! In order to see the correct acceleration value we have to concatenate two bytes together. The x, y and z acceleration values are determined by concatenating the bytes from the values buffer.
3. The acceleration values are printed to the serial terminal
4. The sketch waits for 10ms before executing the loop again. This will allow the loop to run at roughly 100 hz.

```
//This function will write a value to a register on the ADXL345.
//Parameters:
```

```
//  char registerAddress - The register to write a value to
//  char value - The value to be written to the specified register.
void writeRegister(char registerAddress, char value){
  //Set Chip Select pin low to signal the beginning of an SPI packet.
  digitalWrite(CS, LOW);
  //Transfer the register address over SPI.
  SPI.transfer(registerAddress);
  //Transfer the desired register value over SPI.
  SPI.transfer(value);
  //Set the Chip Select pin high to signal the end of an SPI packet.
  digitalWrite(CS, HIGH);
}


//This function will read a certain number of registers starting from a specified address and store thei
r values in a buffer.
//Parameters:
//  char registerAddress - The register addresse to start the read sequence from.
//  int numBytes - The number of registers that should be read.
//  char * values - A pointer to a buffer where the results of the operation should be stored.
void readRegister(char registerAddress, int numBytes, char * values){
  //Since we're performing a read operation, the most significant bit of the register address should be
set.
  char address = 0x80 | registerAddress;
  //If we're doing a multi-byte read, bit 6 needs to be set as well.
  if(numBytes > 1)address = address | 0x40;


  //Set the Chip select pin low to start an SPI packet.
  digitalWrite(CS, LOW);
  //Transfer the starting register address that needs to be read.
  SPI.transfer(address);
  //Continue to read registers until we've read the number specified, storing the results to the input bu
ffer.
  for(int  SPI.transfer(0x00);
  }
  //Set the Chips Select pin high to end the SPI packet.
  digitalWrite(CS, HIGH);
}
```

The readRegister command is used to read values from the ADXL345. When using the readRegister function three variables must be given to the function: the registerAddress where reading should start from, the number of registers that should be read in sequence, and the buffer where the values should be stored. When reading a register from the ADXL345 the address must be modified so that the 8th bit is set high. If more than 1 register is to be read the 7th bit must also be set high. The function accomplishes this task in the first two lines. After that the function is very similar to the writeRegister function. The CS pin is set low to start an SPI sequence, then the address is transferred. After that a loop is used to read the specified number of registers from the accelerometer and the values are stored in the specified buffer. Finally the CS pin is set high to end the SPI sequence and the function is complete.

Once you've hooked the ADXL345 up to an Arduino as specified in the Hook Up section of the guide, download the sketch. After opening the sketch in the Arduino IDE just load it to your board and open the terminal from Arduino. You'll see the X,Y and Z acceleration values start scrolling in the terminal window.

Advanced Sample Code:

Once you understand how to read and write from registers to the ADXL345 there are some really cool functions that you might want to experiment with. In this section we'll look at a sketch that can detect single and double taps, and will convert the x,y and z acceleration values to Gs (a unit of measurement corresponding to 1g). This project will use both an Arduino sketch to interface with the ADXL345 and a Processing sketch so that we can interpret the data. You can download the Arduino sketch here and the Processing sketch here. **In order for the sketch to work properly you'll need to add a wire from the INT1 pin on the ADXL345 to pin D2 on the Arduino.** Let's look at some of the special parts of this sketch.

```
//Create an interrupt that will trigger when a tap is detected.
  attachInterrupt(0, tap, RISING);


  //Put the ADXL345 into +/- 4G range by writing the value 0x01 to the DATA_FORMAT register.
  writeRegister(DATA_FORMAT, 0x01);


  //Send the Tap and Double Tap Interrupts to INT1 pin
  writeRegister(INT_MAP, 0x9F);
  //Look for taps on the Z axis only.
  writeRegister(TAP_AXES, 0x01);
  //Set the Tap Threshold to 3g
  writeRegister(THRESH_TAP, 0x38);
  //Set the Tap Duration that must be reached
  writeRegister(DURATION, 0x10);
```

```
//100ms Latency before the second tap can occur.
writeRegister(LATENT, 0x50);
writeRegister(WINDOW, 0xFF);

//Enable the Single and Double Taps.
writeRegister(INT_ENABLE, 0xE0);

//Put the ADXL345 into Measurement Mode by writing 0x08 to the POWER_CTL register.
writeRegister(POWER_CTL, 0x08);  //Measurement mode
readRegister(INT_SOURCE, 1, values); //Clear the interrupts from the INT_SOURCE register.
```

This code has been added to the setup() section of the sketch. It's a bit different from the setup() section of the ADXL345_Basic example because we're setting the ADXL345 up to detect a single and double tap event. When the accelerometer detects a tap or a double tap we want the INT1 pin to go high, then when the event is handled the INT1 pin will go back low.

When the INT1 pin goes high on the ADXL345 we want to the Arduino to generate an interrupt. In order to do this we use the attachInterrupt() function; we tell the function that we want an interrupt to occur on the Arduino Interrupt 0 (pin D2) when the pin goes from low to high, and we want the Arduino to execute the tap function when this occurs. We'll look at the tap() function later on.

After the interrupt is created we need to configure the ADXL345 to recognize single and double tap events. In order to make this configuration we must change the values in the following registers: INT_MAP, TAP_AXES, THRESH_TAP, DURATION, LATENT and WINDOW. You can read the 'TAP DETECTION' section of the datasheet to see why these values were assigned to their registers. Basically, though, the ADXL345 has been configured so that if a single or double tap is detected the INT1 pin will go from low to high; once the interrupt is handled the INT1 pin will go back low.

```
//Convert the accelerometer value to G's.
 //With 10 bits measuring over a +/-4g range we can find how to convert by using the equation:
 // Gs = Measurement Value * (G-range/(2^10)) or Gs = Measurement Value * (8/1024)
 xg = x * 0.0078;
 yg = y * 0.0078;
 zg = z * 0.0078;
```

In the final project we'll want two types of outputs. If a single tap is detected the Arduino will output the raw accelerometer values for x,y and z; just like in the ADXL345_Basic example. However if the ADXL345 detects a double tap we want the Arduino to output the actual G values of the x,y and z axis. In order to find the G values from the raw accelerometer data we must scale the original values. To find the scale we need to know two things: the number of bits used to represent the original values and the range of acceleration values that can be represented. The ADXL345 is set up to measure the values with 10 bits. We've set the range to +/- 4 g, or an 8g range. The scale is found by dividing the total range by the number that can be represented in 10 bits.

$$Scale = \frac{8}{2^{10}} = \frac{8}{1024} = 0.0078$$

Once we know the scale all we have to do is multiply the raw accelerometer data by the scale to find the number of gs. The variables xg, yg and zg are float variables so that they can hold decimal numbers.

```
if(tapType > 0)
 {
  if(tapType == 1){
   Serial.println("SINGLE");
   Serial.print(x);
   Serial.print(',');
   Serial.print(y);
   Serial.print(',');
   Serial.println(z);
  }
  else{
   Serial.println("DOUBLE");
   Serial.print((float)xg,2);
   Serial.print("g,");
   Serial.print((float)yg,2);
   Serial.print("g,");
   Serial.print((float)zg,2);
   Serial.println("g");
  }
  detachInterrupt(0);
  delay(500);
  attachInterrupt(0, tap, RISING);
  intType=0;
 }
```

When the tap() interrupt function runs (we'll cover this next) it assigns a value to the tagType variable. The variable is assigned '1' if a single tap was detected and '2' if a double tap was detected. If a single tap was detected the Arduino will write the word "SINGLE" to the terminal followed by the x,y and z accelerometer values. If a double tap was detected Arduino will write the word "DOUBLE" to the terminal followed by the g values for the x,y and z axis. After printing the values to the terminal we disable the interrupt for a little while; this prevents the Arduino from detecting any 'echoed' interrupts that may occur while the ADXL345 is still vibrating from the tap event.

```
void tap(void){
  //Clear the interrupts on the ADXL345
  readRegister(INT_SOURCE, 1, values);
  if(values[0] & (1<<5))tapType=2;
  else tapType=1;;
}
```

Because of the way the attachInterrupt() function was called the tap() function will be used whenever Arduino sees an interrupt occur on pin D2. This function is fairly straightforward: Arduino reads the INT_SOURCE register on the ADXL345 and stores the value in the 'values' buffer. The INT_SOURCE register tells us if the interrupt came from a single tap or a double tap event. Depending on which kind of tap set of the interrupt we assign the tapType variable with a 1 or a 2.

Download the Arduino sketch and the Processing sketch. After you've connected the ADXL345 to the Arduino as specified in the Hooking It Up section, and added a wire from the INT1 pin on the ADXL345 to pin D2 on the Arduino, open Arduino and download the sketch to your board. Then open processing and run the ADXL345_Advanced processing sketch. You may have to change the serial port in the sketch to reflect which port your Arduino is plugged into. If everything has been done correctly the Processing window will output the words "Single Tap" to the screen along with the raw accelerometer values when you tap the ADXL345; likewise "Double Tap" and the G values will be displayed when you double tap the ADXL345.