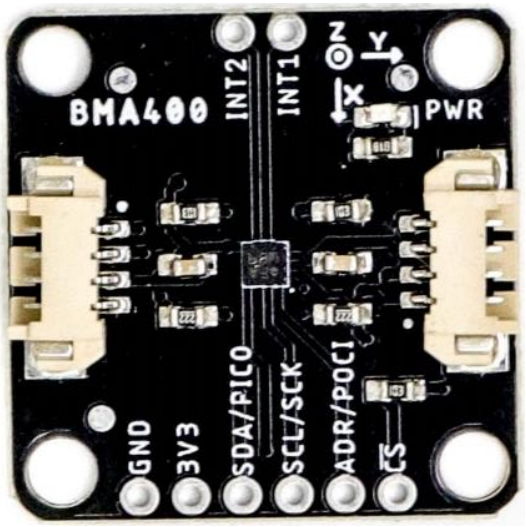
SmartElex Triple Axis Accelerometer Breakout - BMA400

The SmartElex Triple Axis Accelerometer Breakout - BMA400 offer a 3-axis acceleration sensor perfect for ultra low-power applications on a easy to use breakout boards. The breakout includes 0.1"-spaced PTH pins connected to the sensor's communication interface and interrupt pins for applications that require a traditional soldered connection.



The BMA400 from Bosch Sensortech© has a full scale acceleration range of ±2/±4/±8/±16g with exceptionally low current consumption of < 14.5µA while operating at its highest performance settings. The sensor also includes a complete feature set for on-chip interrupts including step counter, activity recognition, orientation detection and tap/double tap.

This guide will go into detail on the features of the BMA400 and hardware on these boards as well as how to integrate it into a circuit using the SparkFun BMA400 Arduino Library.

Hardware Overview

BMA400 3-Axis Accelerometer

The BMA400 3-axis accelerometer from Bosch Sensortec is an ultra-low power motion sensor with a host of interrupt features making it ideal for battery-powered activity monitoring applications.

The BMA400 features four user-selectable ranges of ±2/±4/±8/±16g and consumes just 14.5µA when measuring acceleration data at the highest performance settings. The BMA400 also includes a robust interrupt feature set:

- Auto Wakeup
- Auto Low Power
- Step Counter
- Activity Recognition (Running, Walking, Standing Still)
- Orientation Detection
- Tap and Double Tap

The sensor has two interrupt pins available for all of the above interrupt conditions. The table below outlines a few of the BMA400's sensing and operating parameters. For a full overview of the BMA400, refer to the datasheet.

Parameter	Min.	Typ.	Max.	Units	Notes
Supply Voltage	1.72	1.8	3.6	V	Breakouts run BMA400 at 3.3V.
Normal Mode Current Draw		14.5		µA	Oversampling Rate set to 3
Low-Power Mode Current Draw		0.850			Oversampling Rate set to 0
Sleep Mode Current Draw		0.160			
Acceleration Range		±2		g	Sensitivity of 1024LSB/g
		±4			Sensitivity of 512LSB/g
		±8			Sensitivity of 256LSB/g

		±16			Sensitivity of 128LSB/g
Output Data Rate (ODR)	12.5		800	Hz	ODR in Low-Power mode is 25Hz

Communication Interfaces - I²C & SPI

The breakout communicates over I²C by default but also supports communicating using SPI.The standard size routes the I²C interface to a pair of connectors as well as a 0.1"-spaced PTH header for users who prefer SPI or a traditional soldered connection. The standard-size breakout routes both INT1 and INT2 to a pair of PTH pins.

The BMA400's I²C address is **0x14** by default. Adjust the ADR jumper to change to the alternate address (**0x15**) or open it completely to use the SPI interface. More information on this jumper in the Solder Jumpers section below.

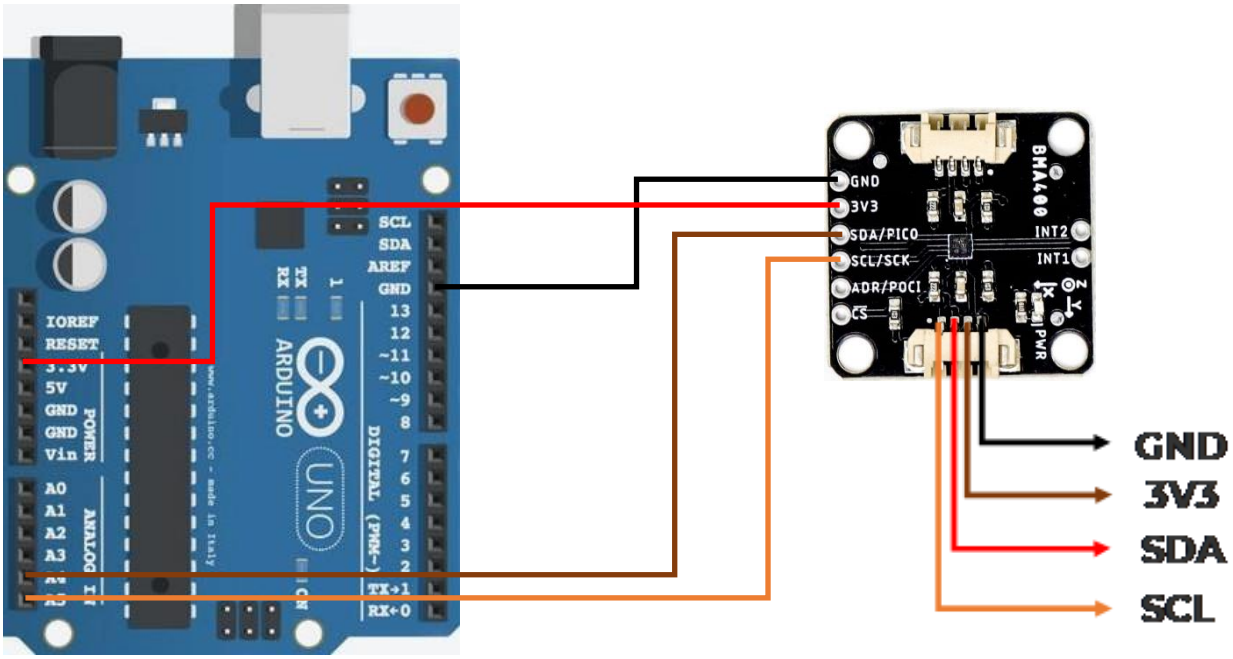
Solder Jumpers

The breakout has four solder jumpers labeled **PWR**, **ADR**, **I2C**, and **CS**. The table below outlines the jumpers' labels, default state, functionality, and any notes regarding their use.

Label	Default State	Function	Notes
PWR	CLOSED	Completes the Power LED circuit.	Open to disable the Power LED.
ADR	SEE NOTE	Sets the sensor's I²C address.	Address is 0x14 by default. Switch to 0x15 by severing the trace between the "Center" and "Left" pads and connecting the "Center" and "Right" pads. Leave open entirely to use the BMA400 over SPI (Standard size only.)
I2C	CLOSED	Ties the SDA/SCL lines to VCC (3.3V) through a pair of 2.2kΩ resistors.	Open the jumper to disable the pull-up resistors on the I2C lines.

CS*	CLOSED	Pulls the BMA400's chip select (CS) pin to VCC (3.3V) through a 10kΩ resistor.	*Standard size only. Open to disable the pull-up on CS.

Wiring:



Arduino	BMA400
A5(SCL)	SCL
A4(SDA)	SDA
3.3V	3V3
GND	GND

SPI Assembly

Setting the breakout up to communicate with the sensor over SPI requires completely opening the ADR jumper and we recommend soldering to the PTH header to make the connections.

BMA400 Arduino Library

The SparkFun BMA400 Arduino Library is based off the API for the sensor from Bosch to let users get started reading data from the sensor and using the various interrupt options. Install the library through the Arduino Library Manager tool by searching for **"SparkFun BMA400"**.

Library Functions

The list below outlines and describes the functions available in the SparkFun BMA400 Arduino Library. For detailed information on the available parameters and use of all functions, refer to the .cpp file in the library.

Sensor Initialization and Configuration

- `int8_t BMA400::beginI2C(uint8_t address, TwoWire& wirePort);` - Initialize the sensor over I²C at the specified address and port.
- `int8_t BMA400::beginSPI(uint8_t csPin, uint32_t clockFrequency);` - Initialize the sensor over SPI with the specified Chip Select pin and clock frequency.
- `int8_t setMode(uint8_t mode);` - Set the power mode.
- `int8_t getMode(uint8_t* mode);` - Returns the setting for power mode.
- `int8_t setAutoWakeup(bma400_auto_wakeup_conf* config);` - Enable Auto Wakeup.
- `int8_t setAutoLowPower(bma400_auto_lp_conf* config);` - Enable Auto Low Power
- `int8_t setRange(uint8_t range);` - Set the measurement range.
- `int8_t getRange(uint8_t* range);` - Returns the value stored for measurement range.
- `int8_t setODR(uint8_t odr);` - Set the output data rate.
- `int8_t getODR(uint8_t* odr);` - Returns the value stored for output data rate.
- `int8_t setOSRLP(uint8_t osrLP);` - Set the output data rate for Low Power.
- `int8_t getOSRLP(uint8_t* osrLP);` - Returns the value stored for output data rate for Low Power.
- `int8_t setDataSource(uint8_t source);` - Set the data source filter. Options are Default with variable ODR, 100Hz ODR, and 100Hz with 1Hz bandwidth.
- `int8_t getDataSource(uint8_t* source);` - Returns the value for data source.
- `int8_t setFilter1Bandwidth(uint8_t bw);` - Set the low pass filter bandwidth.
- `int8_t getFilter1Bandwidth(uint8_t* bw);` - Returns the value set for low pass filter bandwidth.
- `int8_t getStepCount(uint32_t* count, uint8_t* activityType);` - Returns the number of steps measured along with activity type (Still, Walk, and Run).
- `int8_t selfTest();` - Runs self test to determine if the sensor is behaving correctly.

Sensor Data

- `int8_t getSensorData(bool sensorTime = false);` - Requests acceleration data from the BMA400. This must be called to update the data structure.
- `int8_t getTemperature(float* temp);` - Returns the temperature data recorded in °C.

Interrupt Control and Feature Selection

- `int8_t setInterruptPinMode(bma400_int_chan channel, uint8_t mode);` - Set the interrupt pin mode. Options are push/pull and active high/low.
- `int8_t enableInterrupt(bma400_int_type intType, bool enable);` - Enable interrupt condition.
- `int8_t getInterruptStatus(uint16_t* status);` - Returns interrupt status flags for the various interrupt conditions (wakeup, activity, step, etc.).
- `int8_t setDRDYInterruptChannel(bma400_int_chan channel);` - Select the interrupt pin for the data ready interrupt condition.
- `int8_t setGeneric1Interrupt(bma400_gen_int_conf* config);` - Set the generic 1 interrupt configuration.
- `int8_t setGeneric2Interrupt(bma400_gen_int_conf* config);` - Set the generic 2 interrupt configuration.
- `int8_t setOrientationChangeInterrupt(bma400_orient_int_conf* config);` - Set the orientation change interrupt configuration.
- `int8_t setTapInterrupt(bma400_tap_conf* config);` - Set the tap interrupt configuration.
- `int8_t setStepCounterInterrupt(bma400_step_int_conf* config);` - Set the step counter interrupt configuration.
- `int8_t setActivityChangeInterrupt(bma400_act_ch_conf* config);` - Set the activity change interrupt configuration.
- `int8_t setWakeupInterrupt(bma400_wakeup_conf* config);` - Set the device wakeup interrupt configuration.

FIFO Buffer Control

- `int8_t setFIFOConfig(bma400_fifo_conf* config);` - Enable and configure the FIFO buffer.
- `int8_t getFIFOLength(uint16_t* numData);` - Set the number of samples to store in the FIFO buffer.
- `int8_t getFIFOData(BMA400_SensorData* data, uint16_t* numData);` - Pull data stored in FIFO buffer.
- `int8_t flushFIFO();` - Flush the FIFO buffer.

Example Code: Basic Readings I^{2}C

Example 1 demonstrates how to set the BMA400 up to communicate basic motion data over I^{2}C. Open the example by navigating to **File > Examples > SparkFun BMA400 Arduino Library > Example01_BasicReadingsI2C**. Select your Board and Port and click Upload. Open the serial monitor after the upload completes with the baud set to **115200** to watch motion data print out.

If you have adjusted the ADR jumper to change to the alternate I^{2}C address for the BMA400 comment/uncomment the line with the correct value:

```
uint8_t i2cAddress = BMA400_I2C_ADDRESS_DEFAULT; // 0x14
//uint8_t i2cAddress = BMA400_I2C_ADDRESS_SECONDARY; // 0x15
```

The example attempts to initialize the sensor with default settings in I^{2}C at the specified address. If it cannot initialize properly, the code prints out an error in over serial:

If you see this error, double check the sensor is connected properly and set to the correct I^{2}C address and reset the development board or re-upload the code.

The loop polls the BMA400 for acceleration data every 20ms using the 'getSensorData();' function and prints out acceleration for all three axes in g's:

```
#include <Wire.h>
#include "SparkFun_BMA400_Arduino_Library.h"

// Create a new sensor object
BMA400 accelerometer;

// I2C address selection
uint8_t i2cAddress = BMA400_I2C_ADDRESS_DEFAULT; // 0x14
//uint8_t i2cAddress = BMA400_I2C_ADDRESS_SECONDARY; // 0x15

void setup()
{
    // Start serial
    Serial.begin(115200);
    Serial.println("BMA400 Example 1 - Basic Readings I2C");

    // Initialize the I2C library
    Wire.begin();

    // Check if sensor is connected and initialize
    // Address is optional (defaults to 0x14)
    while(accelerometer.beginI2C(i2cAddress) != BMA400_OK)
    {
        // Not connected, inform user
        Serial.println("Error: BMA400 not connected, check wiring and I2C address!");

        // Wait a bit to see if connection is established
        delay(1000);
    }

    Serial.println("BMA400 connected!");
}

void loop()
{
    // Get measurements from the sensor. This must be called before accessing
    // the acceleration data, otherwise it will never update
    accelerometer.getSensorData();

    // Print acceleration data
    Serial.print("Acceleration in g's");
```

```
    Serial.print("\t");
    Serial.print("X: ");
    Serial.print(accelerometer.data.accelX, 3);
    Serial.print("\t");
    Serial.print("Y: ");
    Serial.print(accelerometer.data.accelY, 3);
    Serial.print("\t");
    Serial.print("Z: ");
    Serial.println(accelerometer.data.accelZ, 3);

    // Print 50x per second
    delay(20);
}
//////////////////////////////////////////////////////END//////////////////////////////////////////////////////
```

Move the sensor around in different directions and watch the acceleration data change with the motion.