

Using the SPOT Accelerometer

Ron Goldman



> *The Sun SPOT demo sensor board includes a 3-axis accelerometer that can be used to measure the orientation or motion of the SPOT. This application note describes how to use and calibrate the accelerometer.*

A low-power, three-axis linear accelerometer is mounted on the demo sensor board of the Sun SPOT. The accelerometer can be used to measure the motion of the SPOT. It can also measure the SPOT's orientation with respect to gravity. The Z-axis is perpendicular to the Sun SPOT boards. The X-axis is parallel to the row of LEDs on the sensor board. The Y-axis is parallel to the long edge of the sensor board.

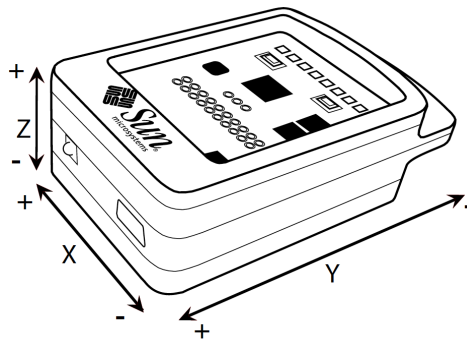


Figure 1. Accelerometer X, Y and Z axes

In Figure 1, the plus (+) on the end of an axis indicates that when the device's acceleration vector increases in that direction, the associated accelerometer readings will grow larger. If the SPOT is sitting flat on a table then the acceleration due to the Earth's gravity will be $1g$ along the positive Z-axis, and $0g$ along the X and Y axes. Note that while gravity is pointing down (along the negative Z-axis) this is equivalent to a uniform upwards acceleration of $1g$ according to the Einstein equivalence principle even though the SPOT is not moving.

The accelerometer consists of a Micro-Electro-Mechanical System (MEMS) sensor element that is displaced from its nominal position when a linear acceleration is applied, causing an electrical imbalance that is read via an analog-to-digital converter. The raw voltage value is then converted to g-force units. The first version of the Sun SPOT used a LIS3L02AQ accelerometer that can be set to measure accelerations over a scale of either $\pm 2g$ or $\pm 6g$.¹ The current version of the SPOT (rev 8) uses a MMA7455L accelerometer that can be set to measure accelerations over a scale of either $\pm 2g$, $\pm 4g$ or $\pm 8g$.²

The SPOT library includes the `IAccelerometer3D` interface that defines the basic methods that any three-axis accelerometer should support. `IAccelerometer3D` extends `ITransducer` which in turn extends `IResource`. The `LIS3L02AQAccelerometer` and the `MMA7455LAccelerometer` classes implement the `IAccelerometer3D` interface along with methods specific to the LIS3L02AQ or MMA7455L. The classes implementing each accelerometer also implement the `IMeasurementRange` and `IMeasurementInfo` interfaces that provide the ability to change the range being sensed.

- 1 For a full description of the technical specifications of the LIS3L02AQ accelerometer please refer to the STMicroelectronics documentation available at <http://www.st.com/stonline/products/literature/od/9321.pdf>.
- 2 For a full description of the technical specifications of the MMA7455L accelerometer please refer to the Freescale Semiconductor documentation available at http://www.freescale.com/files/sensors/doc/data_sheet/MMA7455L.pdf.

The Basic IAccelerometer3D API

The basic methods used to read the current acceleration in G's along each axis are `getAccelX()`, `getAccelY()` and `getAccelZ()`. A fourth method, `getAccel()`, returns the magnitude of the current total acceleration, $|\vec{a}|$. This is the vector sum of the acceleration along the X, Y & Z axes, $|\vec{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2}$.

Here is a code fragment that will loop until the acceleration along the X-axis exceeds 1/4g:

```
import com.sun.spot.resources.Resources;
import com.sun.spot.resources.transducers.IAccelerometer3D;
import com.sun.spot.util.Utils;

IAccelerometer3D acc = (IAccelerometer3D)Resources.lookup(IAccelerometer3D.class);

while (true) {
    double ax = acc.getAccelX();
    if (ax >= 0.25) {
        System.out.println("X acceleration above threshold: " + ax);
        break;
    }
    Util.sleep(250); // check every 1/4 second
}
```

Note that this code will work on either new or old SPOTs. Whether the actual accelerometer is a LIS3L02AQ or MMA7455L does not matter—both accelerometers implement the `IAccelerometer3D` interface.

Here's another fragment to detect when the SPOT is in motion by checking for the total acceleration to deviate from the 1g of gravity:

```
public boolean isMoving() throws IOException {
    double mag = acc.getAccel();
    return Math.abs(mag - 1.0) >= 0.1;
}
```

Measuring tilt

Another set of methods, `getTiltX()`, `getTiltY()` and `getTiltZ()`, use the acceleration along an axis in order to compute the inclination, or tilt, of that axis with respect to the total acceleration the SPOT is experiencing. If the SPOT is at rest then the tilt is measured with respect to gravity.

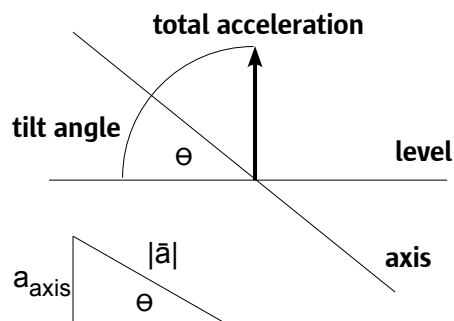


Figure 2. Computing the tilt

As shown in Figure 2 the tilt angle can be computed as $\Theta = \arcsin\left(\frac{a_{\text{axis}}}{|\vec{a}|}\right)$ where θ is the tilt angle measured in radians (with a range of $-\frac{\pi}{2}$ to $+\frac{\pi}{2}$), a_{axis} is the acceleration measured along the given axis, and $|\vec{a}|$ is the total acceleration.

Here is a code example to measure the tilt of the SPOT and display the tilt in the LEDs like a bubble in a level:

```
public void demoBubbleLevel() {
    for (int i = 0; i < 8; i++) {
        leds[i].setOff();           // turn off all LEDs
        leds[i].setColor(LEDColor.BLUE); // set them to be blue when lit
    }

    while (true) {
        try {
            int tiltX = (int)Math.toDegrees(acc.getTiltX()); // returns [-90, +90]
            int offset = -tiltX / 15; // so bubble goes to higher side [6, -6]
            if (offset < -3) offset = -3; // clip angle to range [3, -3]
            if (offset > 3) offset = 3;
            leds[3 + offset].setOn(); // use 2 LEDs to display "bubble"
            leds[4 + offset].setOn();
            Utils.sleep(50); // update 20 times per second
            leds[3 + offset].setOff(); // clear display
            leds[4 + offset].setOff();
        } catch (IOException ex) {
            System.out.println("Error reading accelerometer: " + ex);
        }
    }
}
```

The above example is included in the Sun SPOT SDK and can be found in:

Demos/CodeSamples/AccelerometerSampleCode

For most SPOT programs the need to measure acceleration the functionality defined by `IAccelerometer3D` will be all that is needed.

The `IMeasurementRange` API

The `IMeasurementRange` interface allows reading and changing the scale used by a sensor. The method `setCurrentRange(int)` is used to select the desired scale. The number of possible ranges can be determined by calling `getNumberRanges()`. For any specific range the maximum and minimum values can be determined by calling `getMaxValue(int)` or `getMinValue(int)`. Likewise the accuracy and resolution can be read with `getAccuracy(int)` and `getResolution(int)`.

Here is a code snippet to use the largest scale possible and to print the selected range:

```
import com.sun.spot.resources.Resources;
import com.sun.spot.resources.transducers.IAccelerometer3D;
import com.sun.spot.resources.transducers.IMeasurementRange;

IAccelerometer3D acc = (IAccelerometer3D)Resources.lookup(IAccelerometer3D.class);
IMeasurementRange macc = (IMeasurementRange) acc;
macc.setCurrentRange(macc.getNumberRanges() - 1);
System.out.println("Accelerometer now using the " + macc.getMaxValue() + "g scale");
```

where `getMaxValue()` is from the `IMeasurementInfo` interface and returns the maximum for the currently set scale. If you knew that you had a rev8 SPOT with a MMA7455L accelerometer then this code could be used to switch to the 8g scale:

```
((IMeasurementRange)acc).setCurrentRange(MMA7455LAccelerometer.SCALE_8G);
```

The `MMA7455LAccelerometer` API

Accessing the low-level details of the LIS3L02AQ or MMA7455L accelerometer is generally only needed for calibrating how the raw values read by the accelerometer are converted to actual G values. Both accelerometers use similar methods to offset and scale the raw reading.

The MMA7455L accelerometer has a built in analog-to-digital converter to read the electrical signal from the MEMS sensor element. For each axis the MMA7455L then subtracts a drift offset from the raw digital value and that is what is returned when reading the MMA7455L over the I2C bus on the SPOT's sensor board. The method `getRawAccelValues()` returns an integer array of the x, y and z raw values.

To convert the raw values to G's requires dividing them by the *gain*, where the gain depends on both the axis being converted and the measurement scale the accelerometer is set to.

$$a = \frac{\text{raw}}{\text{gain}}$$

The routine `getGains()` returns a two-dimensional array of doubles, [3][3], giving the gain value for each axis at each scale. The method `getDriftOffsets()` returns an array of doubles giving the zero offset value for each axis; the same value is used for all scales. These two arrays can be set by the methods `setDriftOffsets(double[3])` and `setGains(double[3][3])`. The code to convert from a raw value into G's looks like:

```
int scale = ((IMeasurementRange)acc).getCurrentRange();
double[][] gains = acc.getGains();

double accX = acc.getRawAccelValues()[0] / gains[scale][0];
```

The method `saveCalibration()` takes the current arrays of gains and drift offsets and saves them to the flash memory on the demo board. When the accelerometer code is initialized it tries to read the calibration values back in – if they do not exist it will default to using the nominal values. The drift offsets are written to the MMA7455L accelerometer chip.

The LIS3L02AQAccelerometer API

The LIS3L02AQ accelerometer has similar methods. `getRawAccelValues()` is used to get an array of the raw voltage values read from each axis of the accelerometer by the ADT7411 analog-to-digital converter on the SPOT sensor board. The values range from 0 to 1023. To convert this value to G's requires two steps: subtracting off the *zero offset* value for that axis which will shift the range to *-offset* to *+offset*, and then dividing by the *gain* for the axis to scale the value to G's.

$$a = \frac{\text{raw} - \text{zeroOffset}}{\text{gain}}$$

The nominal zero offset value is 465.5 and the nominal gain for the 2G scale is 186.2, while for the 6G scale it is 62.

The routines `getZeroOffsets()` and `getGains()` each return a two-dimensional array, [2][3], giving the values for each axis at each scale. The code to convert from a raw value into G's looks like:

```
int scale = acc.getCurrentScale();
double[][] gains = acc.getGains();
double[][] zeroOffsets = acc.getZeroOffsets();

double accX = (acc.getRawX() - zeroOffsets[scale][0]) / gains[scale][0];
```

The zero offset and gain of each axis can differ by $\pm 10\%$ from one accelerometer to the next, or between axes of the same accelerometer. To get the greatest accuracy it is important to calibrate each SPOT's accelerometer as described below. Once the proper gains and zero offset values have been determined they can be passed to the accelerometer code with the methods `setZeroOffsets(double[2][3])` and `setGains(double[2][3])`.

The method `saveCalibration()` takes the current arrays of gains and zero offsets and saves them to the flash memory on the demo board. When the accelerometer code is initialized it tries to read the calibration values back in—if they do not exist it will default to using the nominal values.

Calibrating the accelerometer

A simple way to determine the correct drift/zero offset and gain for a given axis of the accelerometer is to take two readings, one with the axis pointing straight upwards and one with it pointing straight downwards. Since both readings are aligned with gravity the average of the two readings is the zero offset, while one half of the difference between the two readings is the proper gain. This needs to be done for all scales.

The drift/zero offset value can also be computed by taking a reading when an axis is perpendicular to the Earth's gravity—when one axis is pointing up or down the other two axes will read zero acceleration.

An application to do this calibration is part of the Sun SPOT SDK. It can be found in the `SPOT-Utilities` directory in the `CalibrateAccelerometer` folder. This application will calibrate a SPOT with either a LIS3L02AQ or MMA7455L accelerometer. If you deploy it to a SPOT and run it you will be able to calibrate the SPOT's accelerometer. The program uses 6 LEDs to indicate which orientations need to be calibrated (red) and which already have (green). When the SPOT's orientation lines up with an uncalibrated orientation the corresponding LED will turn white. To make a reading along the current orientation press switch 1 (just below the LEDs on the left). The SPOT will signal that it is about to take a reading (flashing LEDs) and then for each axis take the average of 50 readings at each scale. If the SPOT's print output is being displayed, then the minimum, maximum and average values for each axis and scale will be printed. When readings have been taken along all 6 orientations the computed drift/zero offsets and gains values will be printed.

This calibration process is done twice. The second time uses the values from the first to tighten the constraints on what is an acceptable orientation.

After the calibration process the accelerometer performs a self test (blue LEDs)—push switch 1 to start. If it passes then the calibration values, will be stored as persistent properties in the demo sensor board's Flash memory and used whenever a SPOT application uses the accelerometer in the future.

Using the accelerometer to determine position

By definition integrating acceleration yields velocity and integrating velocity yields distance traveled. Therefore one should be able to add up the acceleration measured over time and determine how far the SPOT has traveled. Unfortunately errors in measuring the acceleration quickly add up and make the distance estimate meaningless. Even if the SPOT is sitting still the value read from the accelerometer will vary from one reading to the next, so integrating the readings would result in a constantly increasing estimate of the distance the stationary SPOT has moved. For a longer discussion on the difficulties of dead reckoning using the SPOT's accelerometer please read the blog posting “Location, Location, Location” available at http://blogs.sun.com/roger/entry/location_location_location_accelerometer.

A word on sampling rates

The rate at which an application needs to read the accelerometer will vary greatly and depends on the type of accelerometer data being measured. For an application concerned with measuring the SPOT's orientation a sample rate of 10-20 readings per second will usually suffice; likewise for gesture recognition. For measuring the motion of the SPOT when mounted in a toy slot car, a rate of 100 readings per second was more than adequate. If one wishes to measure vibrations then the Nyquist-Shannon sampling theorem tells us that the

sampling frequency needs to be greater than twice the signal bandwidth. The question then is how high a sample rate is possible.

Figures 3 and 4 show the various components involved in reading accelerometer values on a SPOT.

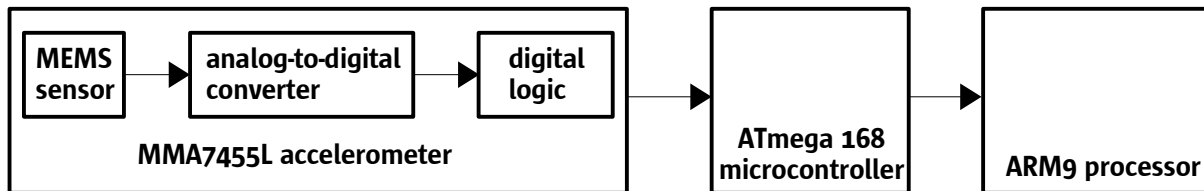


Figure 3. Components involved in reading the MMA7455L accelerometer

The MMA7455L accelerometer can only be set to sample at either 125 or 250 Hz. Use the method `setSampleFilter(boolean)` to select the high (`true`) or low (`false`) sample rate.

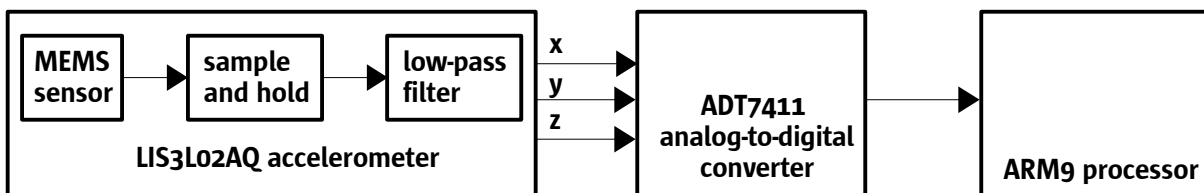


Figure 4. Components involved in reading the LIS3L02AQ accelerometer

Reading the LIS3L02AQ accelerometer is more complicated. The MEMS sensors of the LIS3L02AQ accelerometer are capable of measuring accelerations over a maximum bandwidth of 4.0 KHz for the X and Y axis and 2.5KHz for the Z axis. They are sampled internally at 66KHz. To implement low-pass filtering for antialiasing and noise reduction each output pin from the accelerometer has a capacitor to provide a simple, single-pole low-pass filter. The capacitor used on the SPOT demo sensor board is 0.01 μ F which results in a cutoff frequency of 160Hz. Replacing this capacitor one can raise the cutoff frequency to 2.5KHz.

The ADT7411 analog-to-digital converter can operate in either a round-robin or single channel mode. Reading a single channel it can perform a conversion every 45 microseconds, a sampling rate of 22.2KHz. In round-robin mode the a-to-d converts each of its 8 analog inputs every 579 microseconds, yielding a sampling rate of 1.7KHz.

The SPOT application running on the ARM9 processor takes about 124 microseconds to read a single raw accelerometer value from the a-to-d, which gives a maximum sampling rate of 8.1KHz. To also convert these to G's takes 151 microseconds per reading for a maximum sampling rate of 6.6KHz. Reading raw values for all three axes requires 370 microseconds, a sampling rate of 2.7KHz. Converting to G's takes 460 microseconds for all three axes, or 2.2KHz. These maximum sample rates are only possible if the SPOT application is 100% dedicated to reading the accelerometer. If any other computation is performed in other threads that will decrease the sample rate and introduce jitter in when samples are taken and possibly even cause missed samples.

Putting all of these factors together implies that the maximum sample rate is 320Hz, or slightly higher, to match up with the cutoff frequency of 160Hz. This means taking a set of readings every 3 .125 milliseconds,

which is a rate that can be sustained and still allow for other computation to be done—including real-time analysis of the incoming accelerometer values. If radio packets are sent while reading the accelerometer, it might cause a sample period to be missed as it takes about 4 milliseconds to send a packet. Sample readings may also be lost due to garbage collection, though since a generational collection scheme is used by the Squawk Java VM, the expected delays from GC are only about 20-30 milliseconds.

Replacing the capacitor to raise the cutoff frequency to 850Hz would allow all three axes to be read at a maximum rate of 1.7KHz. If only a single axis needs to be read then it should be possible to read one channel at a rate of 5KHz after raising the cutoff frequency to 2.5KHz—though at this rate samples could probably only be taken for a limited time.

Note: To learn how to take samples at regular intervals please refer to the Sun SPOT Application Note on *Using the AT91 Timer/Counters*.

About Sun Labs

Established in 1990 as part of Sun Microsystems, Sun Labs is now the applied research and advanced development arm of Oracle Corporation. With locations in California and Massachusetts. Sun Labs is one of the ways Oracle invests in the future, and is responsible for many of the technology advancements—including asynchronous and high-speed circuits, optical interconnects, 3rd-generation Web technologies, sensors, network scaling and Java technologies. Although many companies have R&D groups, Sun Labs can claim one of the highest rates of technology transfer in the industry.

SOFTWARE. HARDWARE. COMPLETE.