



Kionix, Inc. Demo Software

User's Manual

08/19/2004

Table of Contents

I. Software Installation.....	1
II. Configuration.....	1
III. Demos.....	2
1. OpenGL Ball.....	3
2. OpenGL Cube.....	3
3. OpenGL Virtual Oscilloscope.....	4
4. Simple Data Display.....	4
5. Free-fall Detector.....	4
6. Data Sampler.....	5
7. Mouse Cursor.....	5
Appendix A: Programmer's Manual.....	6
1. Perl API.....	6
2. Serial Communication.....	8
Appendix B: Basic Concepts of Motion.....	9
1. Calculating Velocity and Distance From Acceleration.....	9
2. Calculating Angle of Tilt From Acceleration.....	10
3. Free-fall Detection.....	10
4. Limitations of These Methods.....	11
Appendix C: Application Notes.....	13
Integrating the Kionix KXM52 Tri-Axis Accelerometer with the Texas Instruments MSP430F149 Microprocessor to Measure Tilt and Other Motions.....	13
Appendix D: Algorithm Reference.....	18
1. Communicating Through the Serial Port.....	18
2. Reading Acceleration.....	19
3. Converting From Acceleration to Tilt.....	20
4. Controlling a Rolling Ball by Tilting.....	21
5. Detecting Free-fall.....	23
6. Jolt Detection.....	24

I. Software Installation

1. Connect the Demo or Development Board to your PC.
2. Run the “KionixDemos.exe” program from the CD.
3. Run the Configuration program from Start Menu -> Kionix Demos to configure and calibrate the sensor. (See “Configuration” below.)

II. Configuration

The configuration program, shown in the figure below, will allow you to configure communication with the device.

1. **Interface** – The method used to communicate with the device. Currently the method is Serial, the RS232 COM port of the computer.
2. **Device** – The driver specific to the Kionix part being used. Because a tri-axis 2g part is used in this application, the driver named TriAxis2g is used.
3. **Serial Port** – This selects the serial port to which the device is connected. Press “Scan” to limit the list to ports that can actually be accessed. A label to the right of the button will appear showing how many ports were found to be available. Then select the port to which the device is connected from the address pull-down above, and press “Test” to check that the device is connected properly. If it is, a label will appear to the right of the Test button confirming that the test was a success.
4. **Performance** – These boxes show the adjustments made to readings received. You may enter these manually if you wish, but it is easiest to lay the device flat and press “Calibrate”.
5. **Adjust X/Y/Z** – These are the amounts by which each reading on the relevant axis is adjusted, in g's, before it is returned. This is to account for any slight variations in center the device might have.
6. **Dead Zone X/Y/Z** – This is the minimum absolute value a reading must reach before it is registered. If it is below this value, it will be read as zero. Having a dead zone can filter out noise, but at the cost of losing real readings if they are very small.

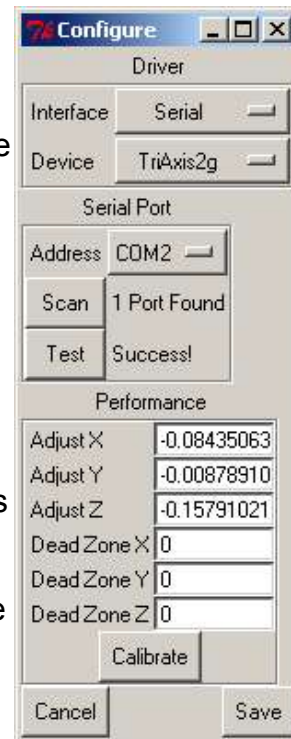


Figure Number range
Figure - configure.pl

When you are finished adjusting the settings, press “Save” to save and exit the program.

III. Demos

Various demos are included with this distribution to demonstrate the capabilities of the device and to get you started making your own applications. They are written in the Perl scripting language, which is an easy-to-learn and extensible language for anything from simple tasks to complex programs and web applications. For more information on Perl, visit www.perl.org.

Several of these demos also use the OpenGL graphics system. OpenGL is widely used for programming graphics-intensive games and is supported by most major video card vendors. For more information on OpenGL, visit www.opengl.org.

1. OpenGL Ball

The OpenGL Ball demo is a simple demonstration of accelerometer-based controls in video games. By tilting the device, the user is able to roll the ball around the board and roll over the red target. Additionally, a strong bump applied to the z axis of the device will cause the ball to bounce into the air. A game development team could use this unique control scheme to add a new level of playability to games like the classic Marble Madness by Atari Games, or to create an entirely new game of their own. Other games which this could be used with include motorcycle, racing, snowboarding, skateboarding, and jet fighter games.



Figure Number range Figure - glball.pl

2. OpenGL Cube

The OpenGL Cube demo shows the ability of the device to detect and react to certain levels of tilt. As the device is rotated, the multicolored cube matches its rotation on the screen. This kind of information could be used in the creation of devices such as digital levels for construction, automatic leveling in photography, and portrait/landscape detection for hand-held devices.

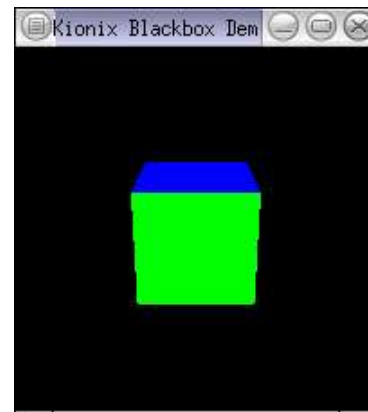


Figure Number range Figure - glcube.pl

3. OpenGL Virtual Oscilloscope

The OpenGL Virtual Oscilloscope demo graphs data in a simple visual format, to give the viewer a general idea of the patterns present in the motion of the device. By recognizing these patterns, such a device could become integral in several kinds of applications. A free-fall detector could be used to protect important data by spinning down a hard drive before it hits the ground. A jolt detector could create a record of package mishandling during shipping. A vibration detector could be placed on a piece of machinery to issue an alert if the pattern of movement changes significantly, indicating the possible need for maintenance.

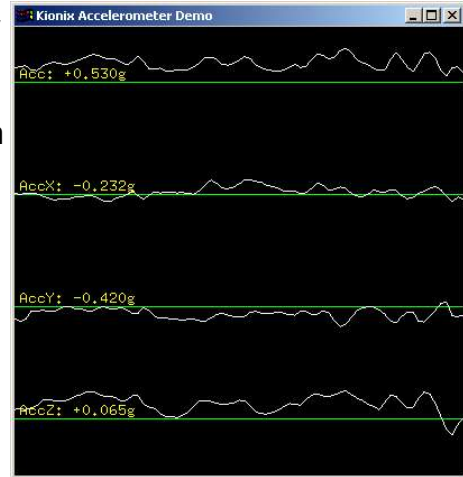


Figure Number range Figure -
glscope.pl

4. Simple Data Display

The Simple Data Display demo shows the current readings of the device in the most raw form as is possible. For those interested in pure data, a device could be attached to a piece of memory to store a running log of all readings. The data could be retrieved later for analysis. The actual collection and processing of accelerometer data does not require very much processor power, and the sample rate of the device is very high, so an accelerometer can be added to almost any application while creating minimal overhead.

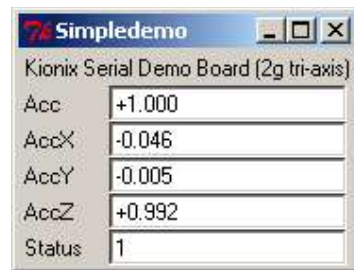


Figure Number range Figure -
simpledemo.pl

5. Free-fall Detector

The Free-fall Detector demo is an example of the free-fall detection algorithm described in Appendix B. It registers when it has been in free-fall for half a foot, then reports how far it fell when it reaches the bottom of its fall. Logging can be temporarily paused with the "Stop" button, or the log can be saved to a CSV (comma-separated values) file with the "Save" button. For more details on the algorithm, see Free-fall Detection in Appendix B.

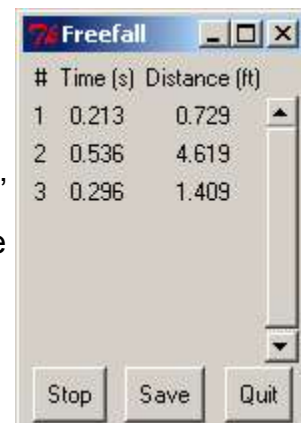


Figure Number range
Figure - freefall.pl

6. Data Sampler

The data sampler takes a constant stream of readings from the device and graphs them in real-time to the screen.

Sampling is done by setting the time you wish to sample, the rate at which you wish to sample, and pressing “Go”. Note that the sample rate you specify is the target sample rate. If you specify a very high number, the program may read less samples than you expect. If you wish to save the data you collected, press “Save”

to write the data to a CSV (comma-separated values) file. You can then use Excel to analyze and graph the data.



Figure Number range Figure - sampler.pl

7. Mouse Cursor

The mouse cursor is a simple demonstration of the idea of a tilt mouse. After opening the program, the mouse cursor can be moved by tipping the device.

Appendix A: Programmer's Manual

1. Perl API

The Perl API for the Kionix demo board provides an easy, object-oriented interface for a Perl programmer to access acceleration data. It is invoked in much the same way as any other Perl module, and returns an object which can be used to interact with the sensor.

Usage

Create a new Sensor object from scratch:

```
use Sensor;
my $Sensor = Sensor->new('Serial', 'TriAxis2g', 'Port' =>
'COM1');
$Sensor->open or warn "Failed to open sensor";
```

Create a Sensor object from an existing configuration file:

```
use Sensor;
my $Sensor = Sensor->newFromConfig('default.ini');
$Sensor->open or warn "Failed to open sensor";
```

Sensor Methods

The following methods are universal to all Sensor objects.

Sensor->new(\$Interface, \$Device, %Options)

Create a new object based on the interface driver `$Interface` and the device driver `$Driver`. The `%Options` hash will be used to override default options if it is included.

Sensor->newFromConfig(\$File, %Override)

Create a new object, reading configuration options from `$File`. The `%Override` hash can be used to override options read from the configuration file.

\$Sensor->loadConfig(\$File)

Load configuration options from `$File` and apply them to `$Sensor`.

\$Sensor->configure(\$Key, \$Value)

Sets a configuration option for the object. Valid options are detailed in "Configuration Options" below.

\$Sensor->open()

Opens the sensor and prepares it for reading. Returns true or false, depending on if the open was successful.

\$Sensor->close()

Closes the sensor. Returns true or false, depending on if the close was successful.

`$Sensor->canMeasure()`

`$Sensor->canMeasure($Reading)`

`$Sensor->canMeasure(@Readings)`

When called with no parameters, `canMeasure` returns a list of all the readings a sensor can return. When called with one parameter, `canMeasure` returns a true or false indicating if that reading can be returned. When called with multiple parameters, `canMeasure` returns a true or false value indicating if all of the readings can be returned.

`$Sensor->status()`

Returns the status of the device, 1 indicating a ready status and 0 indicating a bad status. If there is no way to determine the status of the device, this command returns `undef`.

`$Sensor->$Reading()`

Return the desired reading, as indicated by `$Reading`. `$Reading` can be any of the measurements returned by `$Sensor->canMeasure()`. For an example, see “Measurements Available From a TriAxis2g Sensor” below.

Measurements Available From a TriAxis2g Sensor

`$Sensor->AccX()`

`$Sensor->AccY()`

`$Sensor->AccZ()`

Returns acceleration on the X, Y, or Z axis. This value is in g's, the acceleration due to the Earth's gravity. That is, $1g = 9.8m/s^2$. For tilt calculations, $1g = 90^\circ$. More information on using the values returned by an accelerometer can be found in Appendix B: Basic Concepts of Motion.

`$Sensor->Acc()`

Unlike the other readings, `Acc` is a calculated value. It is based on `AccX`, `AccY`, and `AccZ`, using the Pythagorean Theorem in three dimensions¹. It is a measurement of the magnitude of the acceleration currently being applied to the accelerometer, without the direction. It is useful for applications such as jolt and free fall detection.

`$Sensor->AccAll()`

Returns an array of the X, Y, and Z accelerations. Note that because all three of these are sampled anyway each time any one reading is taken, using this method is three times faster than calling `AccX`, `AccY`, and `AccZ` in succession.

Configuration Options

The following options can be passed to the `configure()` method.

`Port`

This option is accepted by any object using the Serial interface driver. It

¹ Specifically, the calculation used to get overall acceleration is:

$$a = \sqrt{x^2 + y^2 + z^2}$$

specifies which COM port the sensor is plugged into.

`AdjustX`, `AdjustY`, `AdjustZ`

These options are accepted by sensors which can return `AccX`, `AccY`, and `AccZ` respectively. They are the amount by which the reading is adjusted to account for slight offsets in the “zero” position of the sensor. These are usually set during a calibration process, like the one in the `configure.pl` example script.

`DeadZoneX`, `DeadZoneY`, `DeadZoneZ`

These options are accepted by sensors which can return `AccX`, `AccY`, and `AccZ` respectively. They specify a minimum absolute value above which each reading must be. If the reading is within the dead zone for the axis, it is simply returned as zero. These are usually set to filter out noise when the device is level.

2. Serial Communication

For those who wish to communicate with the device in their own program or programming language of choice, this section details how communication with the demo board takes place.

Serial Port Setup

For communication to occur, the serial port must be configured with the correct parameters. The current demo board uses the following setup:

Baud Rate: 38400 bps

Parity: None

Data Bits: 8

Stop Bits: 1

Handshake: None

Flow Control: Off

Commands

The following commands can be issued to the demo board once communication has been established.

Code	Description	Return
X	Get X-axis acceleration.	X (2-byte integer)
Y	Get Y-axis acceleration.	Y (2-byte integer)
Z	Get Z-axis acceleration.	Z (2-byte integer)
A	Return all three axes.	XYZ (three 2-byte integers)
T	Echo a 'T', useful for checking board status.	T (1-byte character 0x54)

Return Values

The value returned for an axis reading is stored as a 2-byte integer in little-

endian (least significant byte first) order. That is, the value can be calculated with the following equation:

$$Reading = FirstByte + (SecondByte * 256)$$

The resulting value represents a number on an arbitrary scale set by the analog-to-digital converter taking the reading. To convert this value to millivolts, the following equation is used:

$$Millivolts = (Reading * 1000) / 1241.2121$$

Finally, to change the reading in millivolts into the acceleration in g's, subtract the 0g offset or center (usually $V_{dd}/2$, or 1650mv for a 3.3V part). Then, divide by the sensitivity rating of the part in mv/g. This demo board uses a device with a sensitivity of 660 mv/g, thus:

$$Acceleration = (Millivolts - Center) / 660$$

The resulting number is the acceleration value returned by the sensor in g's.

Appendix B: Basic Concepts of Motion

The concepts discussed in this section are widely available and are a part of any Physics course, but they have been reproduced here both as a refresher and as a quick reference useful to anyone working with accelerometer data. There is also a discussion of the limitations of these methods when used to determine the position or tilt of a device using a tri-axis accelerometer.

1. Calculating Velocity and Distance From Acceleration

Given an acceleration (a) and a period of time (t), it is possible to calculate the change in velocity during the relevant time period. If the original velocity is also available, the velocity at the end of the time period and the change in position over the time period can be calculated. Lastly, if the original position is available, the position at the end of the time period can be calculated. This can be done according to the steps below.

Calculating Change in Velocity

Given a, the acceleration applied on the axis.

Given t, the time period for which the acceleration was applied.

$$\Delta v = at$$

Results in Δv , the change in velocity during the time period.

Calculating Final Velocity

Given Δv from the previous equation.

Given v_0 , the velocity at the start of the time period.

$$v = v_0 + \Delta v$$

Results in v , the velocity at the end of the time period.

Calculating Change in Distance

Given v_0 , v , and t from previous equations.

$$\Delta d = \frac{(v_0 + v)}{2} * t$$

Results in Δd , the change in distance during the time period.

Calculating Final Distance

Given Δd from the previous equation.

Given d_0 , the distance at the start of the time period.

$$d = d_0 + \Delta d$$

Results in d , the distance at the end of the time period.

Conclusion

At the end of these equations, we have both the final velocity (v) and the final distance (d). This information can be used to determine the same values in the next time period, resulting in a continuous flow of acceleration, velocity, and distance data.

2. Calculating Angle of Tilt From Acceleration

The acceleration data can also be used to find how far the device is tilted. This can be done because the Earth is always pulling on the device with 1g of acceleration. If the device is put on a flat surface and is completely still, all of that acceleration is on the Z axis. The acceleration on the X and Y axes will be zero. If the object is put on its side, whichever axis is pointed toward the earth will read 1g. Getting the angle from the readings on an axis can be done two different ways. The simplest way, although it is not very exact, is to multiply the acceleration on the axis by 90:

$$rotation = acceleration * 90$$

For a much more exact value, take the inverse sine of the acceleration:

$$rotation = \text{asin}(acceleration)$$

3. Free-fall Detection

A tri-axis accelerometer can be used to detect when an object is in free-fall. The first step is to calculate the overall magnitude of the acceleration being applied to the object. This is done with the Pythagorean Theorem:

$$a = \sqrt{x^2 + y^2 + z^2}$$

If the object is in free-fall, the value will be very close to zero. Depending on the rotation of the object, however, it may be a somewhat higher number (see

“Limitations of These Methods” below for details). The distance the object fell can be calculated by using gravity ($g = 9.8 \text{ m/s}^2$) as acceleration (a) in the equation for calculating distance (d) from acceleration (a) and time (t):

$$d = .5 * at^2$$

Note that this calculation will produce the wrong number if the object has been thrown upward or downward. Again, see “Limitations of These Methods” below for more details.

4. Limitations of These Methods

While these equations are effective for many applications, there are several limitations and “gotchas” that one has to watch for when using the data returned by the accelerometer.

Noise In Acceleration -> Velocity -> Distance Calculations

All measurements contain a small amount of background noise. Unfortunately, in an acceleration reading, noise can disrupt the apparent velocity of the device. This difference will become more apparent over time as the “phantom” velocity pushes the distance measurements farther and farther from reality. This change will be relatively slow, however, due to the low-noise nature of the device. The difference can also be made less visible by taking an average of several readings instead of acting on each reading as it arrives. A dead zone can also be implemented, risking the possible loss of the actual readings if they are very small.

Differentiating Between Tilt and Motion

While the ability to measure either motion or tilt is very useful, it comes with the unfortunate disadvantage of not being able to easily differentiate between the two. The z-axis of this device may make this distinction possible on the x and y axes by watching the effect of the acceleration in question on the z axis, and the acquired data could be applied to future demos. Of course, associated equations will also be made available.

Rotation and Center of Mass in Free-fall Detection

Centripetal acceleration, present if the object is rotating, can throw off free-fall detection by causing the overall acceleration of the object to be significantly higher than zero even when the object is actually in free-fall. This effect can be reduced by putting the device in a heavy casing, positioning the accelerometer as close as possible to the device's center of mass, and allowing a range of values to represent free-fall. 0.0 to 0.5g is usually a safe range to use.

Free-fall Time When an Object is Thrown

An object is in free-fall as soon as no forces except for gravity are acting on it. That means that if you throw the device upward, it will be in free-fall even when it is “falling” upward. Similarly, being thrown downward will shorten the time the object is in free-fall before it hits the ground. Either of these events will throw off the equation presented in “Free-fall Detection” above, which assumes that the

object was released into free-fall with a velocity of zero. This limitation is not an issue for applications such as hard drive protection, as they are only concerned with the fact that the object has been dropped, but can adversely affect applications in which the distance the object fell is important. In these cases, it may be better to use an accelerometer with a higher range and implement impact detection instead of using a low-range accelerometer for detecting free-fall.

Appendix C: Application Notes

Integrating the Kionix KXM52 Tri-Axis Accelerometer with the Texas Instruments MSP430F149 Microprocessor to Measure Tilt and Other Motions.



ABSTRACT

This application report describes how to integrate the Kionix KXM52 tri-axis accelerometer with the MSP430F149 to capture and utilize the motion sensing capabilities of the KXM52. Example code is provided for capturing and using tilt sensing, vibration sensing and acceleration. Source code is provided for the MSP430F149 as well as source code for the demonstration software drivers and applications.

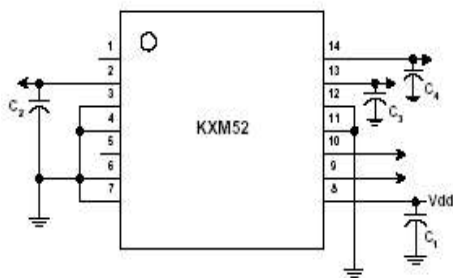
Introduction

Kionix linear accelerometers function on the principle of differential capacitance. Acceleration causes displacement of a silicon structure resulting in a change in capacitance. A signal-conditioning CMOS technology ASIC detects and transforms changes in capacitance into an analog output voltage which is proportional to acceleration. The output voltage is sent to the MSP430F149 ADC for conversion to a digital signal. This signal can then be sent to a serial port, an IR chip, a flash NAND for storage, a USB port, through SPI or I²C to another processor, etc. A serial port is used for the following examples and is also used on the Kionix Evaluation Board. A SDIO port is used for the Kionix SDIO Card.

In the examples that follow the MSP430F149 waits for user input, via the serial port, to retrieve the requested data from the accelerometer. The signals generated by the KXM52 are sent to the MSP430F149 ADC for conversion to a digital signal. The MSP430F149 moves the digital data to a buffer and then sends the data, via the USART, to the serial port. The data, now in numerical format, is received by the software driver (see example code), converted to actual g-force and then sent to the calling program (see example code).

The Kionix Tri-Axis Accelerometer KXM52L20

Application Schematic and Pin Function Table



Pin	Function
1	DNC
2	Output X
3	GND
4	Reserved
5	Parity
6	Reserved
7	Reserved
8	Vdd
9	PS
10	Self Test
11	Reserved
12	GND
13	Output Y
14	Output Z

Definitions

C_2, C_3, C_4 — An external capacitor is used to set the -3dB filter point for each sensor output.
DNC — Do not connect.
 f_{BW} — Sensor bandwidth frequency needed in application (typ. 10Hz to 1500Hz).
Parity — Checks EEPROM for parity error.
PS — Power shutdown pin. When the PS pin is connected to GND or left floating, the KXM52 is shutdown and drawing very little power. When the PS pin is tied to Vdd, the unit is fully functional.
Reserved — For factory use; recommend grounding.
Self Test — The output of a properly functioning part will increase by at least 1g when Vdd is applied to the self-test pin (#10).

Application Design Equations

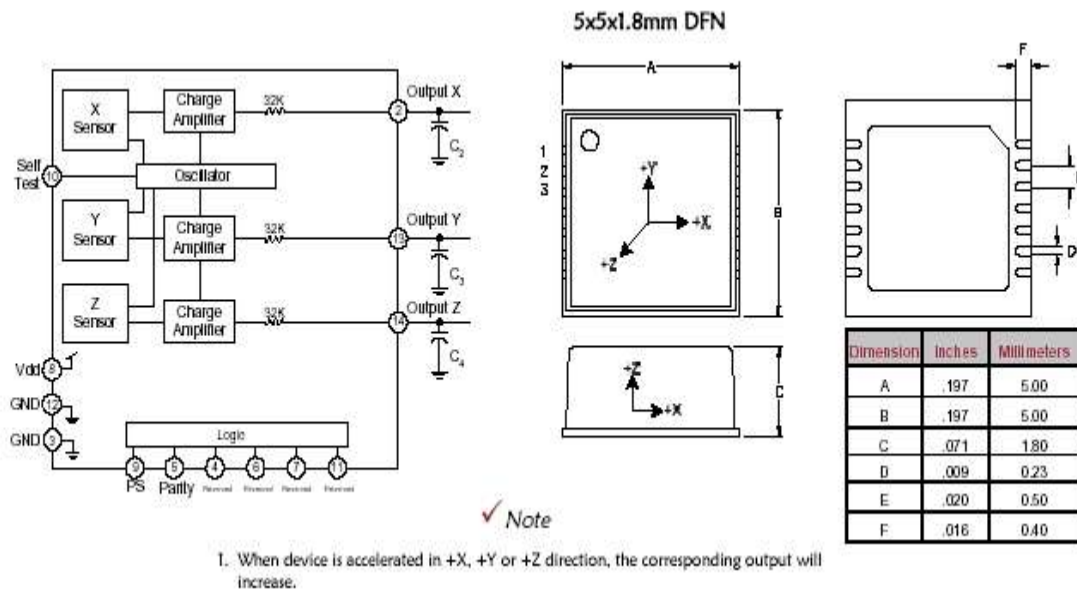
In a typical application, the desired bandwidth will be determined by the fastest signal needing to be measured. Use this equation to calculate C_2 and C_3 for the sensor:

$$C_2 = C_3 = C_4 = \frac{1}{2\pi \times 32000 \times f_{BW}}$$

Notes

1. Recommend using 0.1 μ F for decoupling capacitor C_1 .
2. An evaluation board is available upon request.

Packaging



Refer to the KXM52 Product and Data Sheet <http://www.kionix.com/Product%20Sheets/KXM52.pdf>

Software Implementation, Example Software Drivers and Demos

MSP430F149 Code

```

*****
;
; MSP-FET430P140 Demo - ADC12, Sequence of Conversions (non-repeated)
;
; Description: This program will show how to convert a non-repeated sequence
; of channels.
;
; This example shows how to perform A/D conversions on a sequence of channels.
; A single sequence of conversions is performed - one conversion each on
; channels A0, A1, A2, and A3. Each conversion uses AVcc and AVss for the
; references. The conversion results are stored in ADC12MEM0, ADC12MEM1,
; ADC12MEM2, and ADC12MEM3 respectively and are moved to R5, R6, R7, and R8
; respectively after the sequence is complete. Test by applying voltages to
; pins A0, A1, A2, and A3, then setting and running to a break point at
; "jmp Mainloop." To view the conversion results, open a register window in
; C-Spy and view the contents of R5, R6, R7, and R8.
;
; Note that a sequence has no restrictions on which channels are converted.
; For example, a valid sequence could be A0, A3, A2, A4, A2, A1, A0, and A7.
; See the MSP430x1xx User's Guide for instructions on using the ADC12.
;
; *Note* This example only functions on MSP production silicon, not PMS
; pre-production silicon. Production silicon will be noted on the chip as
; "M430F149" whereas pre-production silicon will be marked "P430F149."

```



```

DW   RESET           ;
ORG   0FFEEh         ; ADC12 Interrupt Vector
DW   ADC12ISR        ;
END

```

Software Driver

Example Pseudo-code For Retrieving An Acceleration Reading

```

SerialPort.send("x") // For this example, the requested axis
                    // will be X
Reading = SerialPort.recieve(2 bytes) // Value is sent as a 2-
                                     // byte integer
Voltage = Reading/4096 // This ratio is defined by the A/D
                    // converter
Acceleration = (Voltage*1000)/Sensitivity // Sensitivity is in
                                     // mv/g
Acceleration -= AdjustmentX // Center reading based on earlier
                          // calibration
return Acceleration // Acceleration is returned in g's

```

Software Demos for the Kionix Evaluation Board (Windows)

1. GL Ball
2. BallPoint
3. GL Cube
4. GL Scope

Hardware Implementation

The Kionix Evaluation Board

Circuit Description

Figure 1 shows the hardware configuration of the Kionix Evaluation Board.

Figure 1

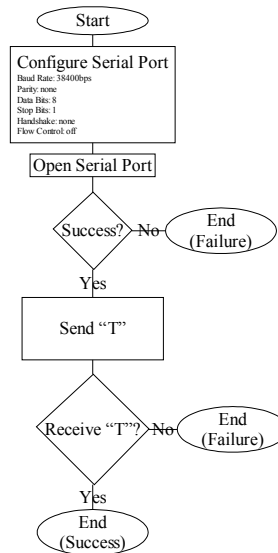
Potential Applications

Automotive	Consumer	Industrial/Scientific
<ul style="list-style-type: none">• Stability Systems	<ul style="list-style-type: none">• Game Controllers	<ul style="list-style-type: none">• Vibration Detection and Monitoring
<ul style="list-style-type: none">• ABS/Antiskid	<ul style="list-style-type: none">• Cell phones	<ul style="list-style-type: none">• Seismic Gas Shutoff
<ul style="list-style-type: none">• Inertial Navigation	<ul style="list-style-type: none">• PDA's	<ul style="list-style-type: none">• Machine Health
<ul style="list-style-type: none">• Headlight Leveling	<ul style="list-style-type: none">• Tilt Sensing	<ul style="list-style-type: none">• Shock Sensing
<ul style="list-style-type: none">• Security/Antitheft	<ul style="list-style-type: none">• Computer Peripherals	<ul style="list-style-type: none">• Equipment Tilt Sensing
<ul style="list-style-type: none">• Fleet Monitoring	<ul style="list-style-type: none">• Virtual Reality	<ul style="list-style-type: none">• Fork Lift Positioning
<ul style="list-style-type: none">• Seatbelt Tensioning	<ul style="list-style-type: none">• Head Tracking	<ul style="list-style-type: none">• Seat Damping
<ul style="list-style-type: none">• Active Suspension	<ul style="list-style-type: none">• Wearable Computing Inputs	<ul style="list-style-type: none">• Platform Leveling
<ul style="list-style-type: none">• Active Ride Control	<ul style="list-style-type: none">• Remote Control	<ul style="list-style-type: none">• Construction Leveling
<ul style="list-style-type: none">• Active Cruise Control	<ul style="list-style-type: none">• Digital Pens	<ul style="list-style-type: none">• UAV's
<ul style="list-style-type: none">• Electronic Park Brake	<ul style="list-style-type: none">• Laptop Anti-Theft	<ul style="list-style-type: none">• UMV's
<ul style="list-style-type: none">• Tilt Sensing	<ul style="list-style-type: none">• Laptop Drop Detection	<ul style="list-style-type: none">• Marine Navigation
<ul style="list-style-type: none">• Crash Detection	<ul style="list-style-type: none">• Disk Drive Monitoring	<ul style="list-style-type: none">• Antenna Alignment
<ul style="list-style-type: none">• Rollover Detection	<ul style="list-style-type: none">• Joysticks	<ul style="list-style-type: none">• Satellite Dish Alignment
<ul style="list-style-type: none">• Tire Motion	<ul style="list-style-type: none">• 3D and Motion-sensing Mice	
<ul style="list-style-type: none">• Misfire Detection	<ul style="list-style-type: none">• Camcorders	
<ul style="list-style-type: none">• Fuel Shutoff	<ul style="list-style-type: none">• Digital Cameras	
<ul style="list-style-type: none">• Battery Disconnect	<ul style="list-style-type: none">• Binoculars	
<ul style="list-style-type: none">• Airbag Deployment	<ul style="list-style-type: none">• Appliances	
<ul style="list-style-type: none">• Vehicle Performance testing	<ul style="list-style-type: none">• Out-of-Balance	
	<ul style="list-style-type: none">• Inertial Navigation	
	<ul style="list-style-type: none">• Handheld GPS	

Appendix D: Algorithm Reference

The following section briefly describes the algorithms used in the demonstration software. The algorithms are written in pseudo-code and visualized with flowcharts.

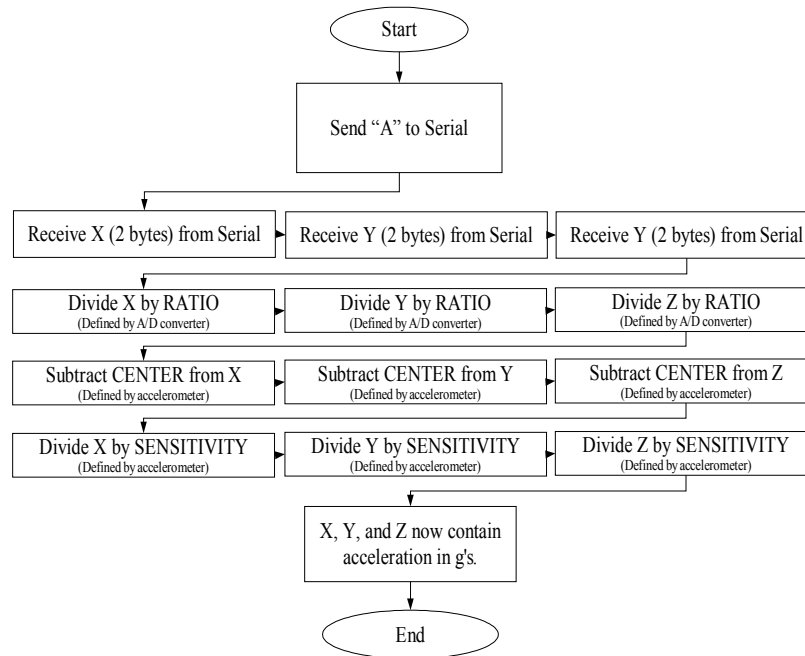
1. Communicating Through the Serial Port



```
// Configure the serial port
serial.baud_rate = 38400bps
serial.parity = none
serial.data_bits = 8
serial.stop_bits = 1
serial.handshake = none
serial.flow_control = off
```

```
// Open the serial port and check connectivity
serial.open()
wait 0.1 seconds // Waiting for the board to wake up
serial.send("T")
RESPONSE = serial.receive(1 byte)
if RESPONSE = "T" then
    // Device is ready
else
    // Device is not ready
    exit
end if
```

2. Reading Acceleration



```

// Read all three accelerations
serial.send("A")
SIGX = serial.receieve(2 bytes) // SIGX is a 2-byte integer
SIGY = serial.receieve(2 bytes) // SIGY is a 2-byte integer
SIGZ = serial.receieve(2 bytes) // SIGZ is a 2-byte integer

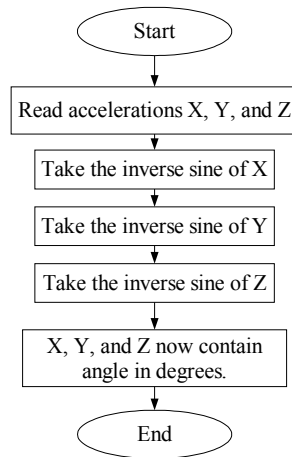
// Relevant constants for conversion
RATIO = 1241.2121 (units/v) // defined by A/D converter
SENSITIVITY = 0.660 (v/g) // defined by the sensor
CENTER = 1.650 (v) // defined by the sensor

// Conversion of SIGX to ACCX
VOLTSX = SIGX/RATIO
VOLTSX = VOLTSX - CENTER
ACCX = VOLTSX / SENSITIVITY

// Conversion of SIGY to ACCY
VOLTSY = SIGY/RATIO
VOLTSY = VOLTSY - CENTER
ACCY = VOLTSY / SENSITIVITY

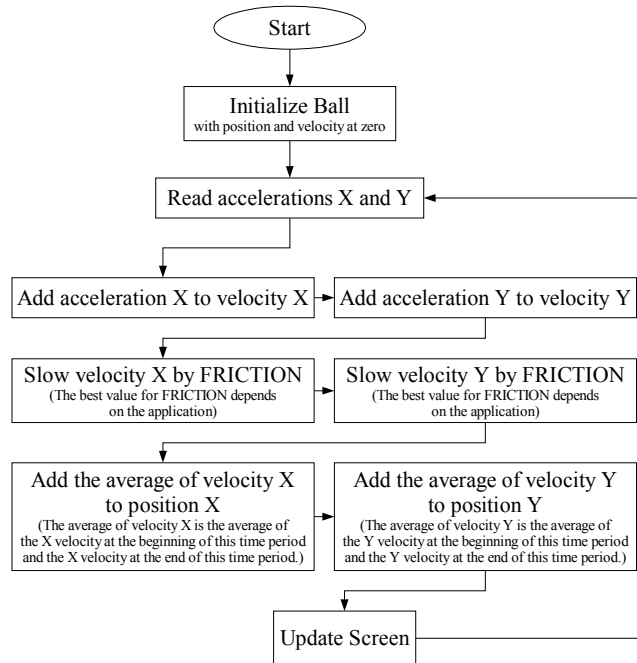
// Conversion of SIGZ to ACCZ
VOLTSZ = SIGZ/RATIO
VOLTSZ = VOLTSZ - CENTER
ACCZ = VOLTSZ / SENSITIVITY
  
```

3. Converting From Acceleration to Tilt



TILT = inverse_sine(ACCELERATION)

4. Controlling a Rolling Ball by Tilting

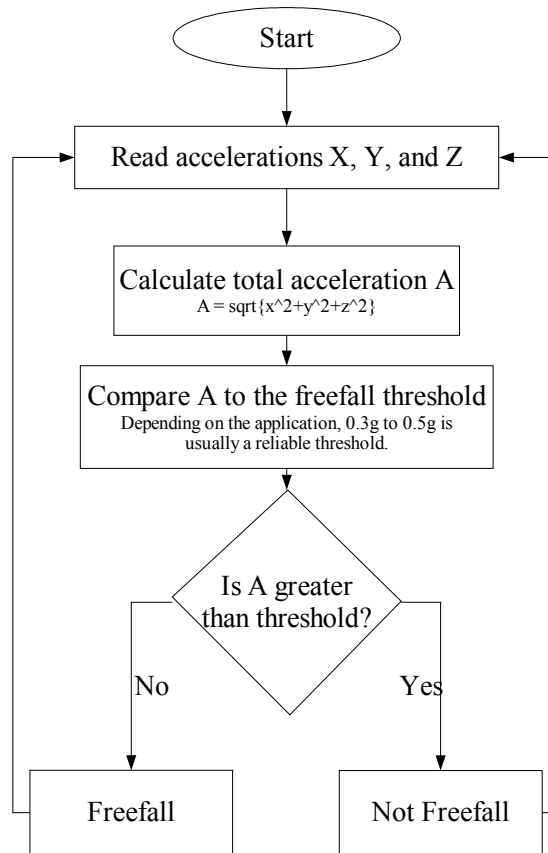


```
ball.positionX = 0
ball.positionY = 0
ball.velocityX = 0
ball.velocityY = 0
```

```
while running
    (ACCX, ACCY, ACCZ) = sensor.allreadings
    OLDVELX = ball.velocityX
    OLDVELY = ball.velocityY
    // Add acceleration to velocity
    ball.velocityX = ball.velocityX + ACCX
    ball.velocityY = ball.velocityY + ACCY
    // For a better feel, the following implements
    // friction. The best value for FRICTION depends on
    // the application.
    if absolute_value(ball.velocityX) < FRICTION then
        ball.velocityX = 0
    else
        if ball.velocityX > 0 then
            ball.velocityX = ball.velocityX - FRICTION
        else
            ball.velocityX = ball.velocityX + FRICTION
        end if
    end if
    if absolute_value(ball.velocityY) < FRICTION then
        ball.velocityY = 0
```

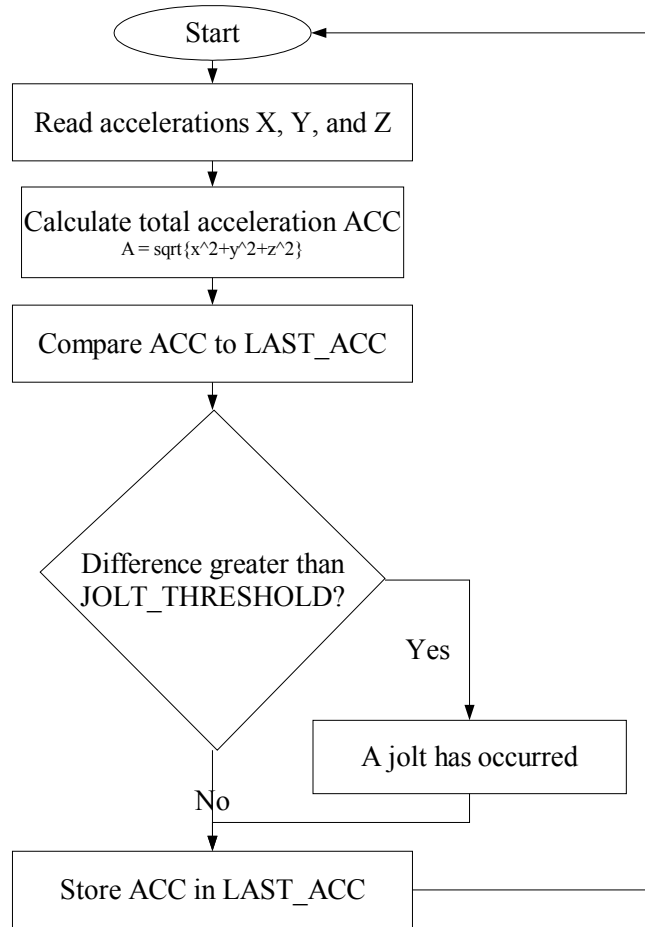
```
else
    if ball.velocityY > 0 then
        ball.velocityY = ball.velocityY - FRICTION
    else
        ball.velocityY = ball.velocityY + FRICTION
    end if
end if
// Add average velocity to position
ball.positionX = ball.positionX + average(
    ball.velocityX, OLDVELX)
ball.positionY = ball.positionY + average(
    ball.velocityY, OLDVELY)
end while
```

5. Detecting Free-fall



```
FREEFALL = 0.3 g // Threshold under which the object is  
                // considered to be in free-fall  
IN_FREEFALL = false // Stores whether or not the device is  
                    // falling  
while running  
    (ACCX, ACCY, ACCZ) = sensor.allreadings  
    TOTAL_ACC = square_root(ACCX ^ 2 + ACCY ^ 2 + ACCZ ^ 2)  
    if TOTAL_ACC < FREEFALL then  
        IN_FREEFALL = true  
    else  
        IN_FREEFALL = false  
    end if  
end while
```

6. Jolt Detection



```
JOLT_THRESHOLD = 1 g // Amount by which the overall  
                    // acceleration reading must change  
                    // between readings to be considered a  
                    // jolt
```

```
(ACCX, ACCY, ACCZ) = sensor.allreadings  
LAST_ACC = square_root(ACCX ^ 2 + ACCY ^ 2 + ACCZ ^ 2)
```

```
while running  
    (ACCX, ACCY, ACCZ) = sensor.allreadings  
    ACC = square_root(ACCX ^ 2 + ACCY ^ 2 + ACCZ ^ 2)  
    JOLT = absolute_value(ACC - LAST_ACC)  
    if JOLT >= JOLT_THRESHOLD then  
        // A jolt has occurred  
    end if  
    LAST_ACC = ACC  
end while
```