Introduction

This application note will help users quickly implement proof-of-concept designs using the KXP84 tri-axis accelerometer. Please refer to the KXP84 Data Sheet for additional implementation guidelines.

Circuit Schematics

This section shows recommended wiring schematics for the KXP84 when operating in both I^2C and SPI modes. It also describes an alternate method for resetting the KXP84 registers to 0x00 upon power up. Please refer to the KXP84 Data Sheet for all pin descriptions.

Note: these schematics are recommendations based on proven KXP84 operation. Your specific application may require modifications from these recommendations.

I^2C Schematic

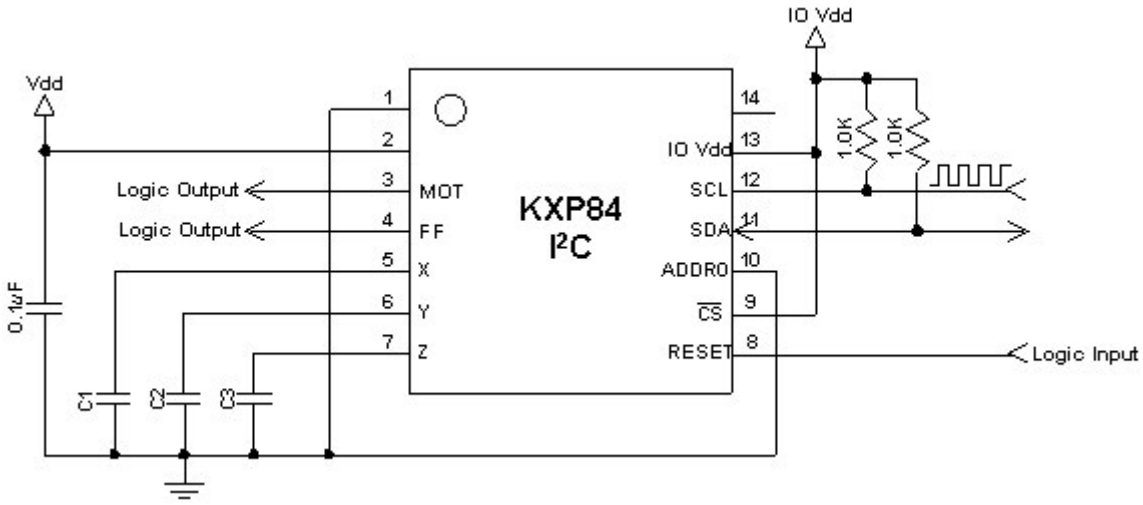

Figure 1. Schematic for I^2C Operation

36 Thornwood Dr. – Ithaca, NY 14850
tel: 607-257-1080 – fax: 607-257-1146
www.kionix.com - info@kionix.com

SPI Schematic

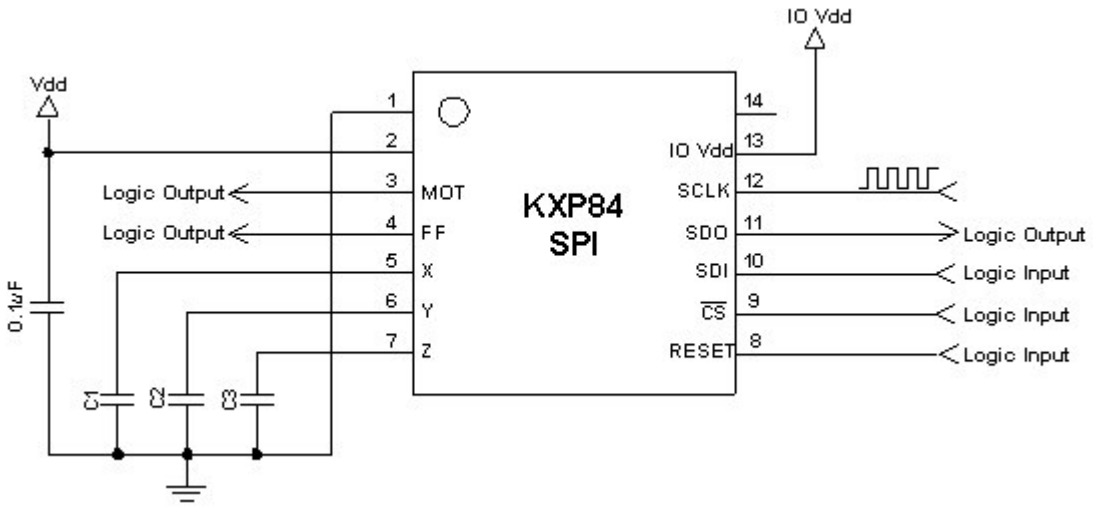

Figure 2. Schematic for SPI Operation

RC Reset (Optional)

A good design practice is to control the I^2C RESET pin with a microprocessor, but it is also possible to reset the KXP84 using the RC circuit shown in Figure 3. This circuit momentarily pulls the RESET pin high at power on, and immediately returns it low during operation. Note that the RC values are just recommendations, therefore the final schematic may differ based on application needs.

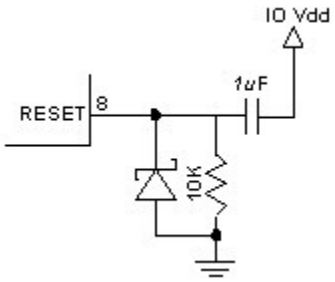

Figure 3. Schematic for RC Reset

Filter Cap Recommendations

To be effective in many applications, such as HDD protection, the KXP84 needs to respond to changes in acceleration as quickly as possible. The accelerometer's bandwidth, and in turn its response time, is largely determined by the external filter capacitance. Therefore, the filter capacitance should be as small as the application will allow. Table 1 shows several commonly used bandwidths and the associated capacitor values for C1, C2, and C3 in the circuits shown above. For most applications, 500Hz (0.01uF) should be a good starting point.

Bandwidth (Hz)	Capacitance (uF)
250	0.02
500	0.01

1000	0.005
1500	0.003

Table 1. Bandwidth (Hz) and Capacitance (uF)

Quick Start Implementation

The KXP84 offers the user a powerful range of operating options and features, mostly controlled by setting appropriate values in registers. This section is not a comprehensive guide to all of the options and features. Rather it is intended to guide the user to an implementation of the KXP84 that will get the device up and running as quickly as possible. Once up and running, the user should experiment with different setting and options to reach the optimum performance for their specific application.

The registers shown in Table 2 need to be set to get the KXP84 up and running:

Register Name	Address		Recommended Value	
	Hex	Binary	Hex	Binary
CTRL_REGB	0x0B	0000 1011	0x06	0000 0110
CTRL_REGC	0x0A	0000 1010	0x00	0000 0000
FF_INT	0x06	0000 0110	0x14	0001 0100
FF_DELAY	0x07	0000 0111	0x14	0001 0100
MOT_INT	0x08	0000 1000	0x4D	0100 1101
MOT_DELAY	0x09	0000 1001	0x14	0001 0100

Table 2. KXP84 Registers

For each register a set of initial recommended values is provided that will ensure the KXP84 comes up in a known operational state. Note that these conditions just provide a starting point, and the values should vary as users refine their application requirements.

Register Recommendations

CTRL_REGB

CLKhld	nENABLE	ST	X	X	MOTIen	FFIen	FFMOTI
0	0	0	0	0	1	1	0

Figure 4. Operational Starting Point for CTRL_REGB

CLKhld = 0: The KXP84 will not hold the I^2C clock during A/D conversions.
nENABLE = 0: The KXP84 is enabled/operational.
ST = 0: The KXP84 self-test function is not enabled/operational.
MOTIen = 1: High-g motion detection interrupt is enabled/operational.
FFIen = 1: Free-fall detection interrupt is enabled/operational.
FFMOTI = 0: High-g motion interrupt flag can be detected on pin 3, and the free-fall interrupt flag can be detected on pin 4.

CTRL_REGC

X	X	X	FFLatch	MOTLatch	X	IntSpd1	IntSpd0
0	0	0	0	0	0	0	0

Figure 5. Operational Starting Point for CTRL_REGC

IntSpd1 and IntSpd0 = 0: The interrupt sampling frequency is 250 samples/second. Therefore, the interrupt delay times can be calculated using Equation 1.

Free-fall Delay (sec) = FF_Delay (# of samples) / 250 (samples/sec)
High-g Motion Delay (sec) = MOT_Delay (# of samples) / 250 (samples/sec)

Equation 1. Free-fall and Motion Delay Calculations

MOTLatch = 0: The motion interrupt output will go high whenever the criterion for motion detection is met. The output will return low when the criterion is not met.

FFLatch = 0: The free-fall interrupt output will go high whenever the criterion for free-fall detection is met. The output will return low when the criterion is not met.

Thresholds and Delays

The following are some suggested acceleration thresholds and delay (or dwell) times appropriate to many drop detection applications. For each suggested parameter the appropriate register value is provided in binary and hexadecimal.

Free fall

Detection Threshold = 0.4g
FF_INT = (0001 0100 or 0x14)

Delay Time = 80mS
FF_DELAY = (0001 0100 or 0x14)

High-g Motion

Detection Threshold = 1.5g
MOT_INT = (0100 1101 or 0x4D)

Delay Time = 80mS
MOT_DELAY = (0001 0100 or 0x14)

Expected Results

When the registers are loaded with the recommended values, the KXP84 becomes armed for HDD protection. This means that the freefall and motion interrupts will be triggered if the accelerometer experiences an event that exceeds any of the above described thresholds and delays. In this case, FF_INT will go to "1" if all accelerometer axis (X, Y, and Z) simultaneously drop below 0.4g for 80mS or more and MOT_INT will go to "1" if any accelerometer axis (X, Y, or Z) go rise above 1.5g for 80mS or more. Figures 6 and 7 show how the KXP84 interrupts will react to a typical event.

By monitoring the interrupt pins 3 and 4, or repeatedly reading the status register, CTRL_REGA (0x0C), the user will be notified of a harmful event and must park/unload the HDD head for protection. Note that the interrupts are selected to be unlatched, so they will return low after the event has concluded. For additional information, please refer to the KXP84 Data Sheet.

Typical Motion Interrupt Example (Unlatched)

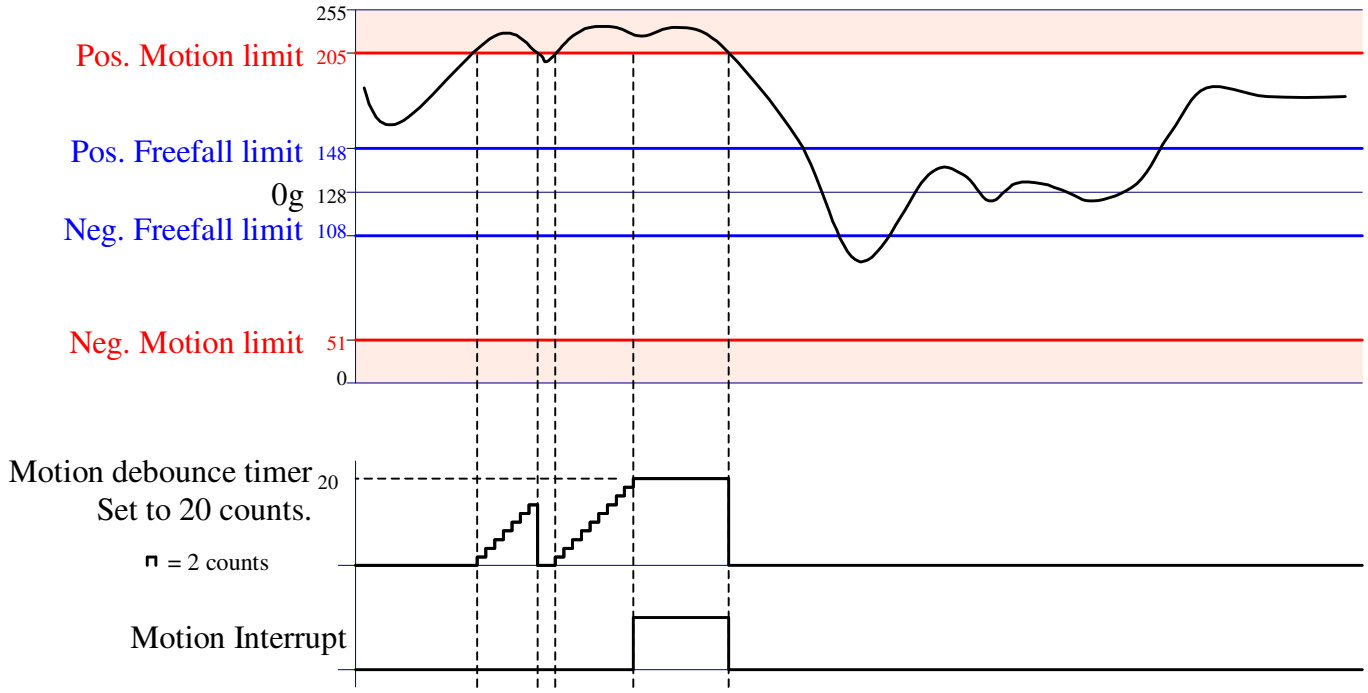

Figure 6. Typical Motion Interrupt Example (MOTLatch = 0)

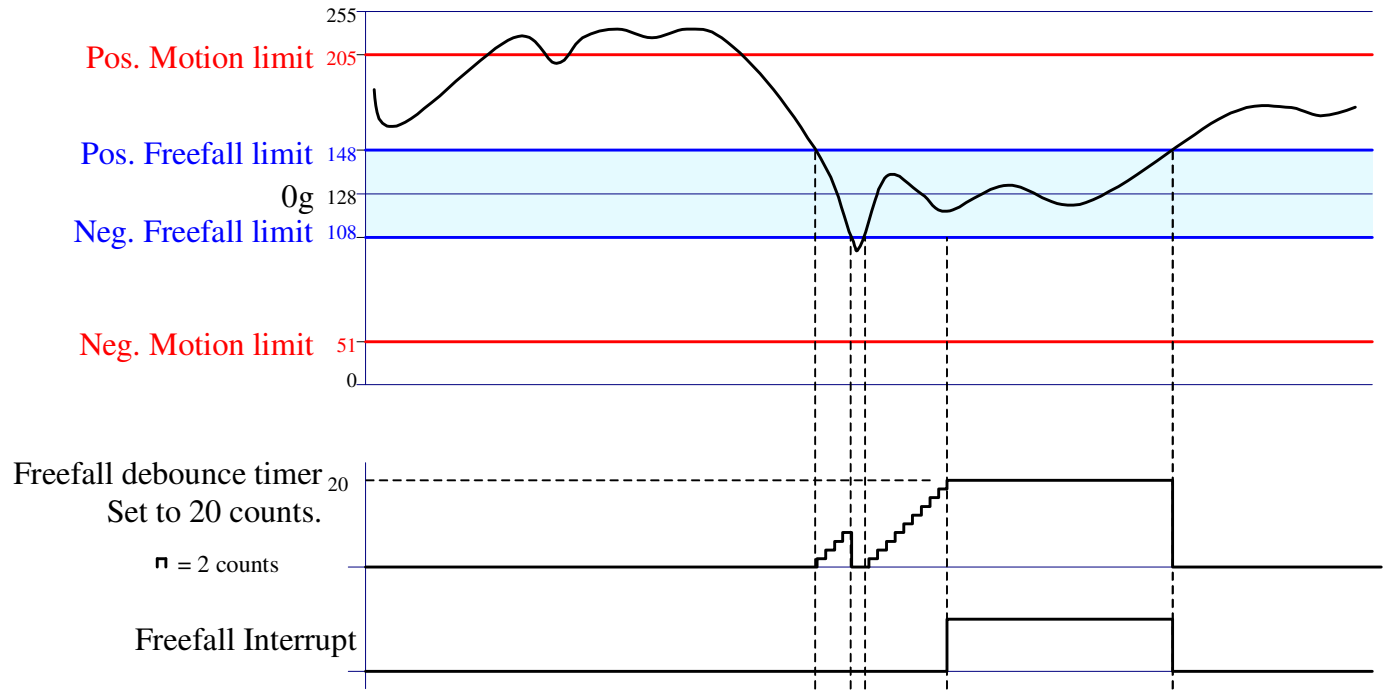

Figure 7. Typical Free-fall Interrupt Example (FFLatch = 0)

Software Implementation

The following source code is the software program used with the Kionix HDD Protection Kit. It was written in Python, an uncompiled language, for I^2C communication between a KXP84 Evaluation Board, Aardvark I2C/SPI Host Device, and a PC. Note that this device is polling the KXP84 status register, CTRL_REGA, every 2mS in order to determine if a HDD head park/unload is required.

```
#!/bin/env python
#============================================================================
#
#
#============================================================================
# IMPORTS
#============================================================================
import sys
import time

from aardvark_py import *

#============================================================================
# CONSTANTS
#============================================================================

port     = 0
bitrate = 100

# Device
DEVICE = 0x18
```

```
# Addresses
XOUT_H = 0x00
XOUT_L = 0x01
YOUT_H = 0x02
YOUT_L = 0x03
ZOUT_H = 0x04
ZOUT_L = 0x05
FF_INT = 0x06
FF_DELAY = 0x07
MOT_INT = 0x08
MOT_DELAY = 0x09
CTRL_REGC = 0x0A
CTRL_REGB = 0x0B
CTRL_REGA = 0x0C

# CTRL_REGA
Parity = 0x04
MOTI = 0x02
FFI = 0x01

# CTRL_REGB
CLKhld = 0x80
nEnable = 0x40
ST = 0x20
MOTDen = 0x10
FFDen = 0x08
MOTIen = 0x04
FFIen = 0x02
FFMOTI = 0x01


#====================================================================
# MAIN PROGRAM
#====================================================================

# Open the device
handle = aa_open(port)
if (handle <= 0):
    print "Unable to open Aardvark device on port %d" % port
    print "Error code = %d" % handle
    sys.exit()

# User Configuration

Sensitivity = (5.0/(2**12)) # g/count
SampleRate = 250.0 # Hz

sys.stdout.write("Freefall Threshold (g): ")
FF_Threshold = float(sys.stdin.readline())
FF_Threshold = round(FF_Threshold / (16 * Sensitivity))
print "FF_INT byte: 0x%x" % FF_Threshold
print "Freefall Threshold set to %0.3fg" % (16 * FF_Threshold * Sensitivity)

sys.stdout.write("Freefall Interrupt Delay (ms): ")
FF_Delay = float(sys.stdin.readline())/1000
FF_Delay = int(round(FF_Delay * SampleRate))
if FF_Delay < 0x01:
    FF_Delay = 0x01
```

```
if FF_Delay > 0xFF:
    FF_Delay = 0xFF
print "FF_DELAY byte: 0x%x" % FF_Delay
print "Freefall Interrupt Delay set to %dms" % ((FF_Delay / SampleRate) * 1000)
sys.stdout.write("Motion Threshold (g): ")
MOT_Threshold = float(sys.stdin.readline())
MOT_Threshold = round(MOT_Threshold / (16 * Sensitivity))
print "MOT_INT byte: 0x%x" % MOT_Threshold
print "Motion Threshold set to %0.3fg" % (16 * MOT_Threshold * Sensitivity)

sys.stdout.write("Motion Interrupt Delay (ms): ")
MOT_Delay = float(sys.stdin.readline())/1000
MOT_Delay = int(round(MOT_Delay * SampleRate))
if MOT_Delay < 0x01:
    MOT_Delay = 0x01
if MOT_Delay > 0xFF:
    MOT_Delay = 0xFF
print "MOT_DELAY byte: 0x%x" % MOT_Delay
print "Motion Interrupt Delay set to %dms" % ((MOT_Delay / SampleRate) * 1000)

# Ensure that the I2C subsystem is enabled
aa_configure(handle,  AA_CONFIG_SPI_I2C)

# Power the EEPROM using the Aardvark adapter's power supply.
# This command is only effective on v2.0 hardware or greater.
# The power pins on the v1.02 hardware are not enabled by default.
aa_target_power(handle, AA_TARGET_POWER_BOTH)

# Activate the pull-up resistor(s)
aa_i2c_pullup(handle,AA_I2C_PULLUP_BOTH);

# Setup the clock phase
#aa_spi_configure(handle, mode >> 1, mode & 1, AA_SPI_BITORDER_MSB)

# Set the bitrate
bitrate = aa_i2c_bitrate(handle, bitrate)
print "Bitrate set to %d kHz" % bitrate

# Configure the device
aa_i2c_write(handle,DEVICE,0,array('B',[CTRL_REGC, 0]))
aa_i2c_write(handle,DEVICE,0,array('B',[CTRL_REGB, 0]))
aa_i2c_write(handle,DEVICE,0,array('B',[CTRL_REGB, FFIen | MOTIen]))
aa_i2c_write(handle,DEVICE,0,array('B',[FF_DELAY, FF_Delay]))
aa_i2c_write(handle,DEVICE,0,array('B',[MOT_DELAY, MOT_Delay]))
aa_i2c_write(handle,DEVICE,0,array('B',[FF_INT, int(FF_Threshold)]))
aa_i2c_write(handle,DEVICE,0,array('B',[MOT_INT, int(MOT_Threshold)]))

print "Ready..."
while (1):
    aa_i2c_write(handle,DEVICE,0,array('B',[CTRL_REGA]))
    (ret, data, count) = aa_i2c_read_ext(handle,DEVICE,0,1)
    if len(data) < 1:
        sys.stdout.write("SENSOR READ ERROR. Park drive head.\n(Press Enter to resume
sampling or Q followed by Enter to quit.) ")
        userinput = sys.stdin.readline()
        if (userinput == "q\n" or userinput == "Q\n"):
            aa_close(handle)
            sys.exit()
    elif data[0] & FFI:
```

```
        sys.stdout.write("Freefall started. Park drive head.\n(Press Enter to resume
sampling or Q followed by Enter to quit.) ")
        userinput = sys.stdin.readline()
        if (userinput == "q\n" or userinput == "Q\n"):
            aa_close(handle)
            sys.exit()
        aa_i2c_write(handle,DEVICE,0,array('B',[CTRL_REGB, 0]))
        aa_i2c_write(handle,DEVICE,0,array('B',[CTRL_REGB, FFIen | MOTIen]))
    elif data[0] & MOTI:
        sys.stdout.write("High-g motion detected. Park drive head.\n(Press Enter to
resume sampling or Q followed by Enter to quit.) ")
        userinput = sys.stdin.readline()
        if (userinput == "q\n" or userinput == "Q\n"):
            aa_close(handle)
            sys.exit()
        aa_i2c_write(handle,DEVICE,0,array('B',[CTRL_REGB, 0]))
        aa_i2c_write(handle,DEVICE,0,array('B',[CTRL_REGB, FFIen | MOTIen]))

aa_close(handle)
```