PANDORA PRODUCTS

Accelerometer Recorder
gCorder

Jim Schimpf

Document Number: PAN-201410019
Revision Number: 0.9
12 December 2015

Pandora Products.
215 Uschak Road
Derry, PA 15627

Pandora Products.

215 Uschak Road

Derry, PA 15627

Phone: 724-539.1276

Email: jim.schimpf@gmail.com

Document Revision History

Version	Author	Description	Date
0.1	js	Initial Version	18-Oct-2014
0.2	js	LPC1114 Version	2-Nov-2014
0.3	js	Modify LPC1114 with computer cmd set	22-Nov-2014
0.4	js	Build C library to support computer cmd set	23-Nov-2014
0.5	js	Update Schematic & board pictures	21-Dec-2014
0.6	js	Minor corrections	24-Dec-2014
0.7	js	Connection pictures	6-Dec-2015
0.8	js	Gnuplot source	11-Dec-2015
0.9	js	Use X11 plotting	12-Dec-2015

Contents

List of Figures

1 Introduction

With the advent of inexpensive MEMS accelerometers like the MMA8451[3] it is now possible to build a very small accleration recorder that can be strapped to system to record it's motion. The rest of this paper details the design and constuction of the system.

2 Design

2.1 Criteria

There are a number of criteria for the system:

- Small and light, so it won't overburden the tested items.
- Rugged so it can withstand being dropped or thown

- Battery powered
- Inexpensive interface
- Data output in the simplest form possible
- Allow g range selection
- Allow specification of data collection period
- Allow setting a trigger g value for data collection

2.2 Initial Design

2.2.1 Hardware

The prototype design is done using an Arduino[1] and the Adafruit breakout board for the MMA8451. This is proof of concept for the software and hardware.

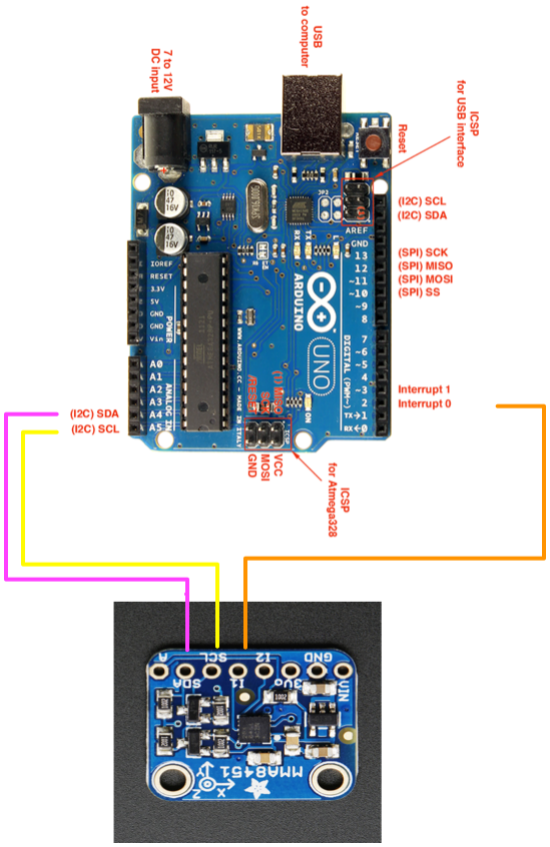

Figure 1: Prototype

This was used to develop the software and do initial testing for data collection and display.

2.2.2 Software

The software was written in the Arduino IDE using Adafruit's MMA8451 library. This library has code to initialize the chip, code to read and write the registers and defines of the needed chip registers. The code allows setup of the data collection, start a collection and readout the collection.

Major limitation of this version is the limited RAM available in the Arduino, we only have room for collection of ~100 points. So the time of data collection is limited to 100 * point collection time. Also the interrupt it used in a polling fashion rather than a true interrupt but as will be seen later this is not a real problem in actual use.

2.2.2.1 Startup When started the software shows:

```
G Recorder Ver[ Oct 19 2014 07:02:15]
-- Finding Acclerometer --
Setting EEPROM
-- Ready --
Main Menu
1 - Setup Run
2 - Capture Data
3 - Read Out Data
Choice:
```

The Setup Run choice allows these choices:

```
Shock Capture Setup
1 - Capture Rate (ms)
2 - Capture G Range (2,4,8)
3 - Set G trigger (% 8 g)
4 - Exit
Choice:
```

These choices allow the capture rate, sensor range and trigger value to be set. The trigger range because of the way it's being used is a % of 8 g. That is an input of 50 would set the trigger at 50% of 8 g or 4 g to trigger the board. This trigger setting is independent of the capture range chosen for the sensor. All of these values are stored in EEPROM so are available ever after power cycle or reset.

2.2.2.2 Capture After the setup is done Capture Data is picked the LED on the board goes out and the system waits for the trigger. This is done is a polling loop not an interrupt. The latency here seems quite low as shown in this picture of a capture.

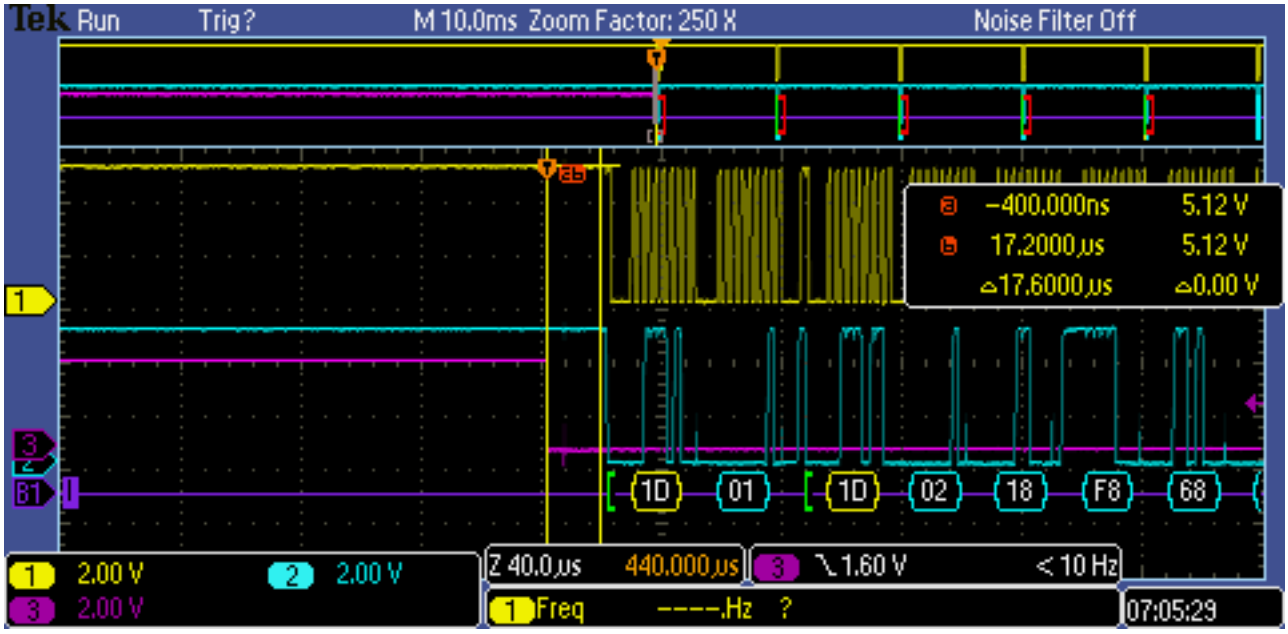

Figure 2: Capture trace

The purple trace is the interrupt line and as you can see data capture starts about 18 uSec after the trigger.

While the capture is running the LED is on steadly and after the capture is done it blinks rapidly. Also you cannot run another capture till the data is read out.

2.2.2.3 Read Out The readout dump is just 4 columns of data with ,'s between them. The time, x g value, y g value, z g value.

```
Choice: 3
ReadOut function
Range 8
Trigger 1
0,1087,474,3966   <-- Time, x, y, z G values as raw data
10,-703,587,1100
20,91,277,2776
30,-97,202,2051
40,-53,117,1400
```

This also gives the current range and trigger values. In this case range is +/- 8 g and the trigger is 1% of 8 g or 0.08 g. The values following are the raw sensor data.

2.2.3 Data Display

To display the data it is cut and pasted from the terminal window into a text file. This text file is then processed by an AWK script to convert the raw data to floating point g values and to convert the time to seconds:

```
# Convert to CSV
BEGIN {
range = 2 # Set default
}
/^range/ {
range = $2;
}
{
# Get the TIME X Y & Z
n = split($0,data,",")
# Convert time to seconds
data[1] = data[1]/1000
# Convert the DATA to G values
# 2 g = 4096
data[2] = range * data[2]/4096
data[3] = range * data[3]/4096
data[4] = range * data[4]/4096
# Put out T,x,y,x
printf("%f  %f  %f  %f\n",data[1],data[2],data[3],data[4])
}
```

This is written to a text file called plotdata.txt and then we run a gnuplot script to plot this data:

```
#
# Plot the G Recorder data
# Plot Time vs X,Y and Z g data
# Data is:  Time (sec) Xg(g's)....
#
set title 'G Recorder'  # This is the label for the entire graph
set xlabel 'Time (s)'   # This is the X Axis label
set ylabel 'Acceleration g' # This is the Y Axis lable
set grid     # This puts a grid on the graph
#
# These next lines do the heavy lifting and plot the data
# Note: The \' at the end of the lines, this is actually 1 very long
#       line the \' escape the end of line characters.  Don't type
#       anything after them but your RETURN
# plot  - Says plot the data from "data.txt" the file we created
```

```
#         using says use these columns of data for this line 1:2 says
#         use column 1 as X and column 2 as Y
#         with line says connect the points with a line
#         You can also say with point or with dot for different styles
#         title '<name>' gives a title to this line and puts that
#               name and color into a legend on the graph
plot "plotdata.txt"  using 1:2 title 'X g' with line,\
"plotdata.txt"  using 1:3 title 'Y g' with line,\
"plotdata.txt"  using 1:4 title 'Z g' with line
```

All of these operations are done by a script file:

```
#!/bin/sh
# Plot data
# Syntax plotdata.sh <data file>
awk -f dataprep.awk $1 > plotdata.txt
gnuplot gplot.txt
```

The output is from gnuplot is:

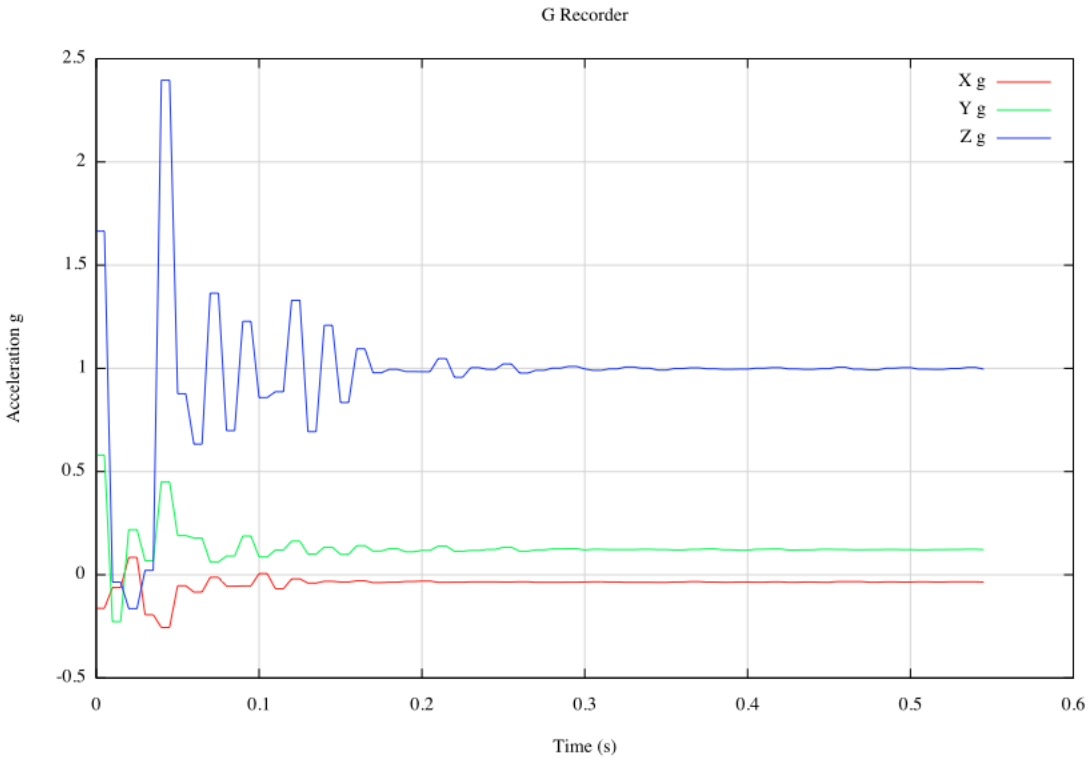

Figure 3: Run 1 plot

2.3 LPC1114 Design

2.3.1 Introduction

This design will be done with a LPC1114 chip instead of an Arduino. This can also use FreeRTOS for multithreading and mutexes. It will necessitate re-design of the I2C connection as the Arduino used the Wire library (One Wire) which is not available on the LPC1114.

2.3.2 Hardware

The schematic for the LPC1114 hardware is:

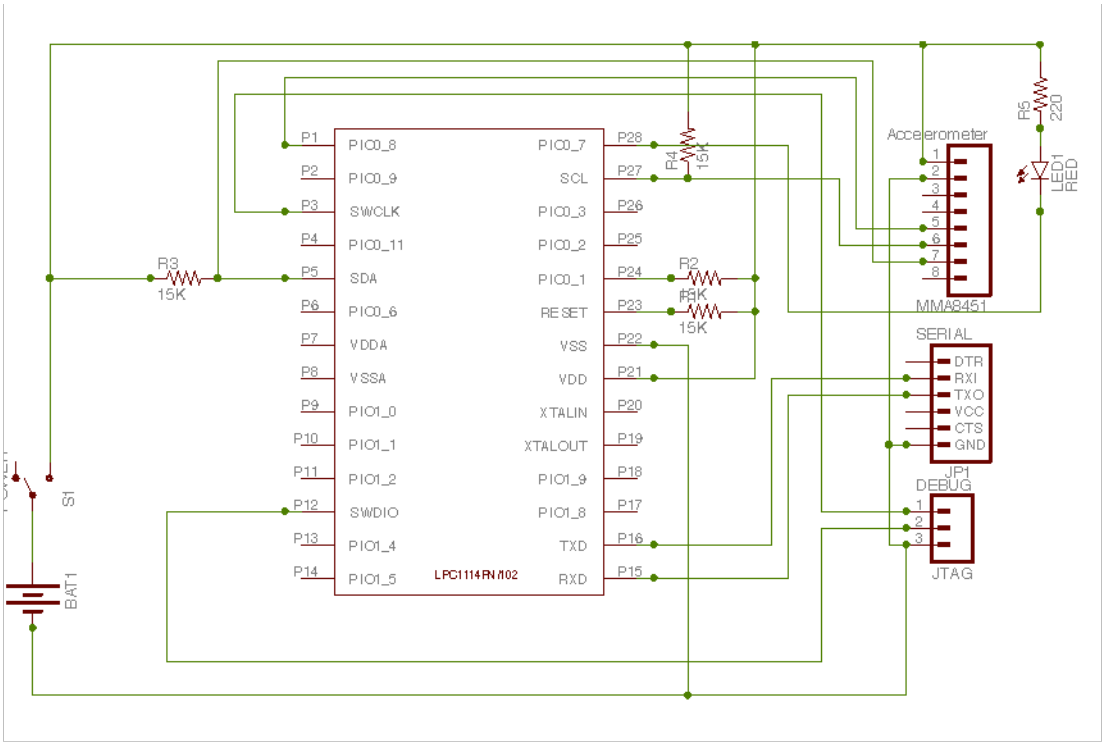

Figure 4: gCorder LPC1114 Schematic

The board design is:

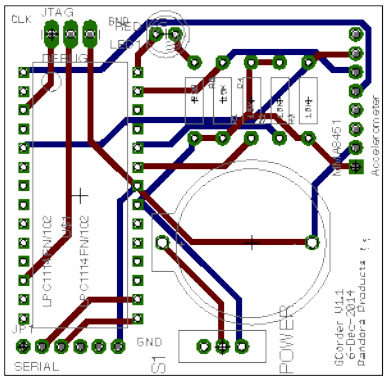

Figure 5: gCorder LPC1114 Board

As you can see we use a button cell to power the system and have a ON-OFF switch to power down when not in use.

A completed prototype is:

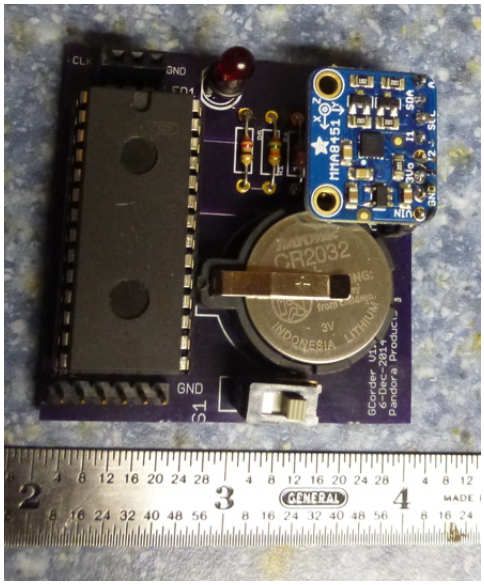

Figure 6: Prototype

2.3.3 MMA8451 Support

2.3.3.1 I2C Support The MMA8451 when reading a register needs to have the address written and register written then a read done to the address without an interposed STOP bit. This is called

Repeated Start. It is not the way the standard I2C code for the LPC library works. Shown below is a read of the ID register, note the first write of the address and the register (0x0D). Then read command of the one byte without a stop bit between the two.

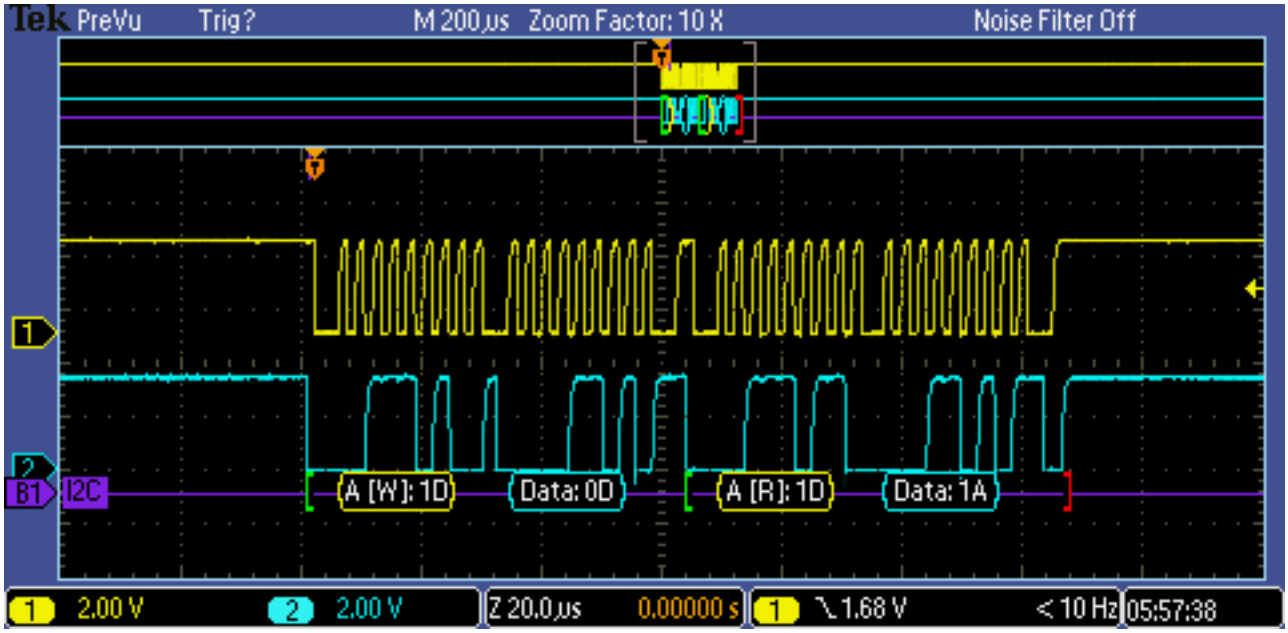

Figure 7: Read interaction

The write interaction is simpler just a specification of the address and register then the data.

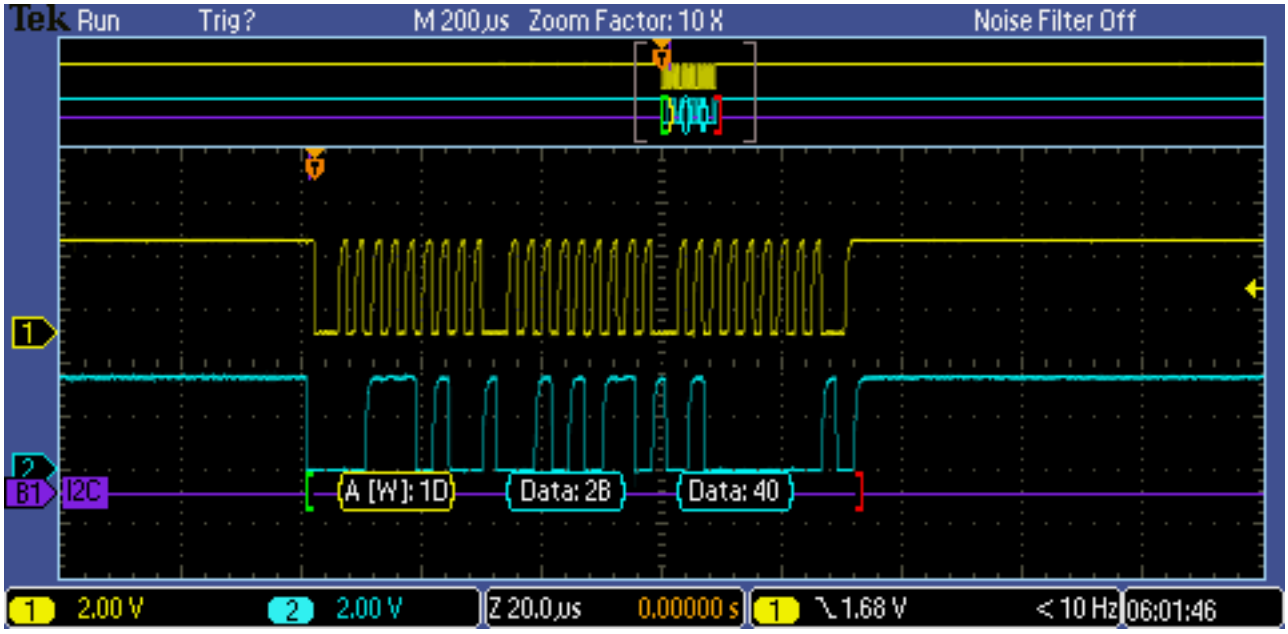

Figure 8: Write interaction

Multiple reads are done in the same way a single read is done in this case reading the x,y an z autoincrementing acceleration registers.

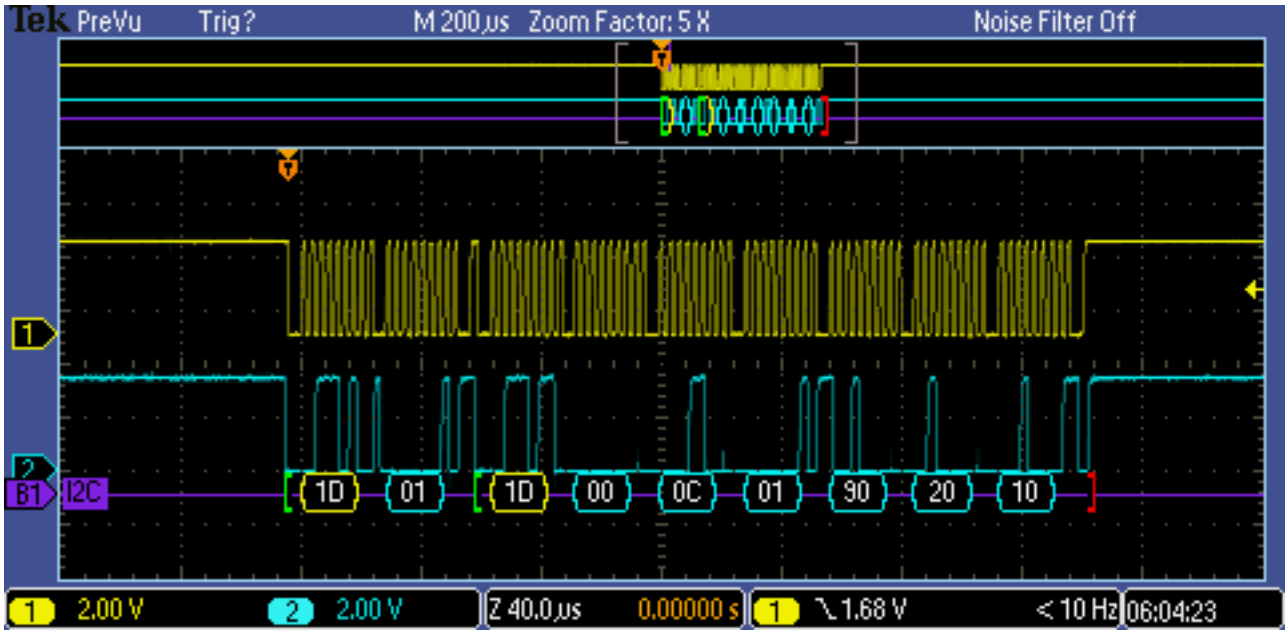

Figure 9: Multiple read

The code to do this is in two parts. Two additional functions were added to the I2C support code.

First the write was witten using the LPC1114 support library function Chip_I2C_MasterSend():

```
/**
 * @brief   Write I2C data
 * @param   size    # Bytes to write
 * @param   addr    I2C address
 * @param   reg     I2C register
 * @param   buf     Data here
 * @return  # bytes written
 */
int i2cCmdWrite(int size,int addr,int reg,unsigned char *buf)
{
    int rtnval;
    unsigned char buffer[20];
    int i;
    // Build the buffer with the data to send
    // but with the register first
    buffer[0] = reg;
    for( i=0;i<size; i++)
    {
        buffer[i+1] = buf[i];
    }
    rtnval = Chip_I2C_MasterSend(I2C0, addr, buffer, size + 1);
    rtnval = rtnval - 1;
    return rtnval;
}
```

Notice that the register is put into the data buffer and then all the rest of the data is added to the buffer following it. This increases the size of the data by 1 and we have to correct for that on transmission and return of the count of sent data.

The read of data is done via the Chip_I2C_MasterCmdRead().

```
/**
 * @brief   Read I2C data
 * @@param  size    # Bytes of data
 * @param   addr    I2C device addr
 * @param   reg     I2C device register
 * @param   buf     Data is here
 * @return  # Bytes read
 */
int i2cCmdRead(int size,int addr,int reg,unsigned char *buf)
{
    int rtnval;
```

```
    rtnval = Chip_I2C_MasterCmdRead(I2C0, addr,reg , buf, size);
    return rtnval;
}
```

2.3.3.2 MMA8451 Library With the read/write in I2c built then a library of functions to support the MMA8451 can be built. The first function checks for the presence of the MMA8451 and intializes it:

```
/*
 * @brief initMMA8451 Set up accelermeter
 *
 * @param addr  - Device address
 * @return 0 if OK, <> 0 if not
 */
int initMMA8451(int addr )
```

This uses the device address either 0x1D or 0x1C and returns 0 if it is present. You can read the source to see the setup but it intialized the device to some useful default settings. Then two other functions are added to read and write registers. These will be needed to program the device for generation of interrupts on acceleration changes. Both of thes are written using the new I2C functions discussed above.

```
/*
 * @brief readMMA8451Reg - Read a register @ address
 *
 * @param reg - Register to read
 *
 * @return register value or -1 if faillure
 */
int readMMA8451Reg(int reg)
/*
 * @brief writeMMA8451Reg - Write to a control register
 *
 * @param reg   - Register used
 * @param value - Value to put in register
 *
 * @return 0 OK, <> 0 problem
 */
int writeMMA8451Reg(int reg, int value)
```

The final function is used to read out the 6 bytes containing the acceleration data and return them as 3 shorts.

```
/*
 * @brief readMMA8451Acceleration Get acceleration value
 *
 * @param data  Acceleration data structure, filled out here
 *
 * @return 0 OK <> 0 problem
 */
int readMMA8451Acceleration(ACEL_DATA *data)
```

The data structure is:

```
typedef struct {
                short x;
                short y;
                short z;
} ACEL_DATA;
```

To keep things simple and small in the program we will keep all values as integer and only convert to floating point values after the data has been transferred to a PC. Thus the g values here will be as integer fractions of the chosen g range. In the PC and in plotting they will be converted to Earth g.

2.3.4 Operating program

The program has been built see gCorder that duplicates the functions of the Arduino program. The major difference is this program uses freeRTOS and thus can use threads and uses mutex to control program actions. Thus unlike the Arduino programs it does not need callbacks to run the LED flashes and does not use spin loops to wait for functions. With the additonal memory in the LPC1114 it can capture up to 200 points in a run.

The output is shown here:

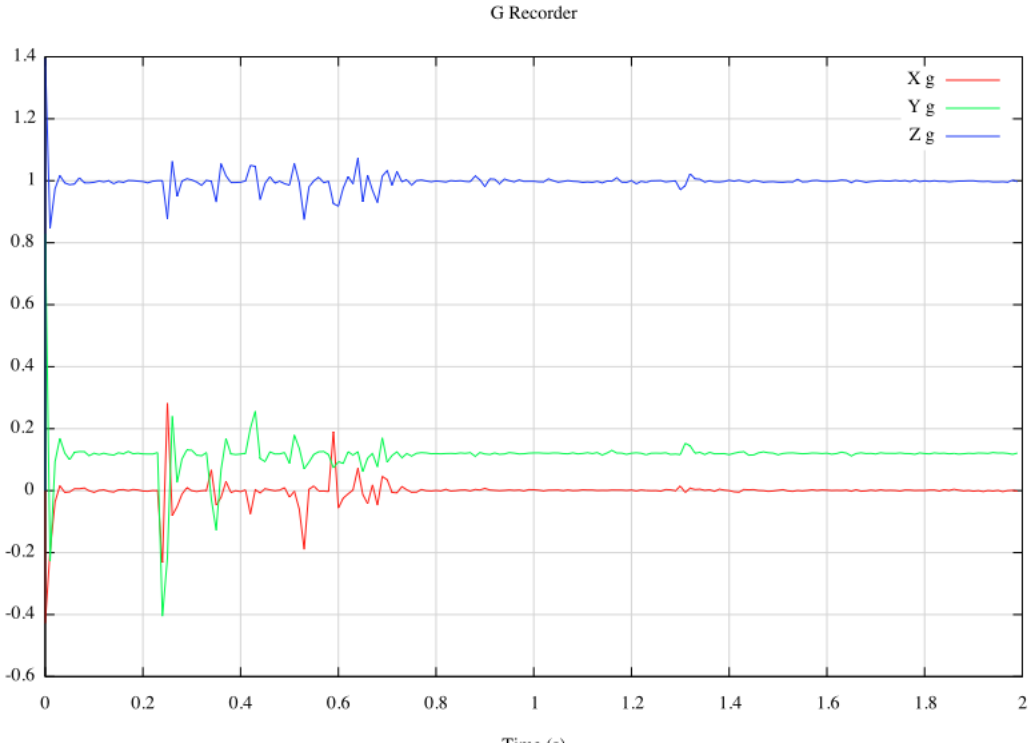

Figure 10: Dropped Bolt

NOTE: This graph is slightly different than the previous ones as output -> g calculations have been corrected.

2.4 gCorder - User Program

2.4.1 Introduction

The programs developed so far have been prototypes to exercise the MMA8451 and determine its characteristics. Now we will detail an end user program, it will be in two parts a “headless” program running on the LPC1114 board and a Mac GUI program for the user interface.

Dividing into two programs like this allows the LPC1114 program to be integer only and deal directly with the MMA8451 in its native units. The Mac program can then use engineering units and values familiar to the user.

2.4.2 LPC1114 User interface - Form

The interface will be in ASCII so it can be typed in directly but will be terse to easily support a computer interface.

The basic form will be:

```
<cmd>[<value>],<cmd>[<value>],....<cr>
```

- <cmd> - A single letter (see below for commands)
- [<value>] - An optional value which is a number.
- , - Command separator
- <cr> - Marks the end of command string

All commands or sequence of commands are followed by OK<cr> and the command prompt G>. If the command did not work then it is followed by BAD #<cr> and then the prompt # is the failing command in the string, 1 for the first and so on.

2.4.3 LPC1114 User interface - Commands

Command	Form	Use
S	S#	Collection time step (ms)
R	R#	g range set (g)
T	T#	Trigger value set (% range)
C	C	Arm for collection
D	D	Dump data
? or H	? or H	Help
V	V	Get version information

Table 1: Command Set

2.4.3.1 Step - Time step size

```
S####
```

The single character S followed by digits is the time step in ms. Time must be >= 10.

2.4.3.2 g Range - Sensor range

R#

The single character R followed by 2 (+/-2g) 4 (+/-4g) or 8 (+/-8g) range value. The range can only be these values.

2.4.3.3 Trigger g value

```
T##
```

The single character T followed by a % of the current range as the trigger value. 1-99 ok values.

2.4.3.4 C Collect data

```
C
```

The single character C will start data collection. NOTE This command must be alone or the last command in a set as after the return status no more data is available till after trigger.

2.4.3.5 D Dump data

```
D
```

Dump the current data.

```
Range 2
Trigger 10
0,-1226,90,4095
10,5813,-33,4085
20,475,384,4396
:
:
-- END RUN --
```

Data is <time>,<x g><y g>,<z g><cr>

2.4.3.6 ? or H Help

```
? or H
```

Will return the current command set to the user.

```
gCorder Control Program VER:[Dec  2 2014 04:35:50]
Input => <cmd>[number],<cmd>[number],....<cr>
CMD  Number  Meaning
 S   Step    Time step in ms Min 1 ms
 R   Range   2,4 or 8 only
 T   Trigger % of range 10-99
 C    -      Collect data
 D    -      Dump data
 V    -      Version
 H or ?      Print this help
```

2.4.3.7 V Version information Return software version information.

```
gCorder Control Program VER:[Dec  2 2014 04:35:50]
```

2.5 gCorder C library

2.5.1 Introduction

This library is being built so that we can run the gCorder through a GUI program and not have to deal directly with the gCorder hardware interface. This program will present the user with a dialog where they can set up the recorder, arm and then dump and plot the data.

The code will be in two parts, the first described here is a C libarry that interfaces to the gCorder giving an API to control it. The second part will be a Objective C Cocoa project for OS X with the GUI.

2.5.2 C Library API

2.5.2.1 gCORD struct The gCORD structure is returned by the init and must be passed to all the called routines.

```
#define DUMP_SZ 10000
typedef struct {
                    PORT_HANDLE *sp;
                    char dumpBuffer[DUMP_SZ];
                    int dbSize;

                    // Device Scanner
                    int scanRun;
                    pthread_mutex_t scanLock;
                    pthread_t scanThread;
                    int gCorderPresent;
                } gCORD;
```

The Device Scanner is a thread that runs every 2 seconds and verifies the presence or absence of a gCorder. The flag **gCorderPresent** is set to 1 when present and 0 when not.

2.5.2.2 gCorderInit

```
gCORD *gc = gCorderInit( int port );
```

Output	Parameter	Meaning
	port	Serial port of gCorder
gc		gCORD struct if open or NULL if not

This call opens the serial port to the gCorder. Port is item # of that port in the port list. (Run myterm.sh to see the list). This also starts the device scanner which runs in background while the gc is active.

2.5.2.3 gCorderCommand

```
gCORD *gc;
val = gCorderCommand( "T100R8T20\r",gc );
```

Output	Parameter	Meaning
	cmd	Command sequence to gCorder NOTE: terminated by \r
	gc	Open gCORD structure
status		0 Success <> 0 Failure

This call is used to send a command or commands to the gCorder (see 2.4.3 on page 15) . You can send any number of commands concatenated together. The example shows Step, Range and Trigger commands sent as one mesage. NOTE: You must put a \r at the end of all command strings. The command returns success if accepted by the gCorder, it fails if the gCorder is not present or the commands are bad.

2.5.2.4 gCorderDump

```
gCORD *gc;
val = gCorderDump(gc);
```

Output	Parameter	Meaning
	gc	Open gCORD structure
status		0 Success <> 0 failure

This command is used to dump the data from from the gCorder after collection. This will return failure if there is no data to read or the gCorder is not connected

2.5.2.5 gCorderClose

```
gCorderClose(gc);
```

Output	Parameter	Meaning
	gc	Open gCORD structure
NONE		0

This call is made to close the gc structure when finished

2.6 gReceiver OS X GUI program

2.6.1 Introduction

This is a program that runs on OS X and has a GUI interface using the library described above. It makes the sensor interface simple and will setup and run the sensor. In addition it can display the sensor data and save it in comma separate value files. The program is designed as single dialog box with no menu items.

2.6.2 Installation

2.6.2.1 Hardware In order to use the gCorder you must have a USB <-> Serial converter like this one from Sparkfun https://www.sparkfun.com/products/11814 or from AdaFruit ://www.adafruit.com/products/284 to plug into the gCorder from your computer.

You need the gCorder hardware, verify the battery is ok by turning it on and noting if the LED flashes.

2.6.2.2 Software Before installing the program you must install XCode, GNUPLOT and AQUA term OS X programs.

XCode Available from the Apple Store. This is needed for the command line programs.

XQUARTZ is available at http://www.xquartz.org.Use the installer.

GNUPLOT is at http://www.miscdebris.net/upload/gnuplot-4.2.5-i386.dmg Move the Gnuplot to applications.

Now copy the gRecorder.app program to Applications.

2.6.3 Device Connection

Connect up the gCorder with a USB<->Serial connector like this:

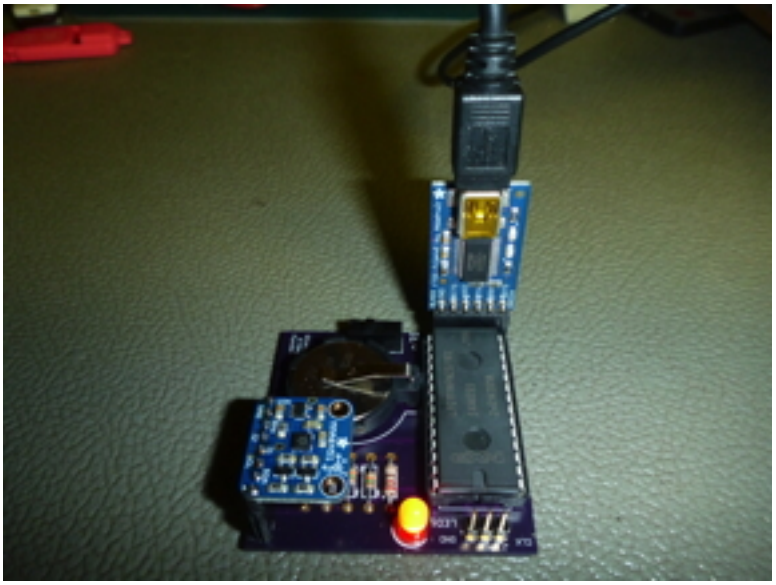

Figure 11: Connection

This shows the USB<->SERIAL with its GND pin aligned with GND marked on the board. Note S1 position turning the board on. (The LED should be flashing.

2.6.4 Use of the program

When you start the gReceiver.app it will show this.

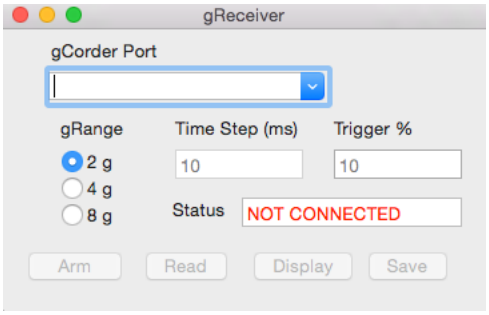

Figure 12: Startup

The first thing you must do is used the gCorder Port combo box to pick the serial port the sensor is using:

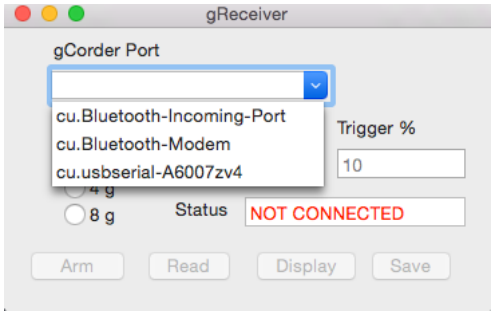

Figure 13: Pick Port

Once that is done and there is a MMA8451 sensor present the display becomes:

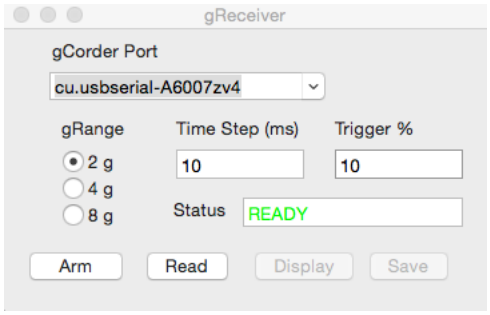

Figure 14: Sensor Ready

At this point you can change the collection parameters, like time step and trigger %. **Note if you change theses values hit enter and status line will show you changed them.**

- The trigger value is % of 8g (independent of range setting) so 10% above is a trigger value 0.8g
- Range is the sensitiviy of setting for the sensor.

After arming the sensor you can disconnect it from the USB<->SERIAL connector (gently...) and collect the data. When you arm it the sensor light does not blink, it's ready for data collection. When the collection is done the LED on the sensor blinks fast and when you connect it to the program it shows: (Note an easy way to test is to rap the table top sharply and it will trigger)

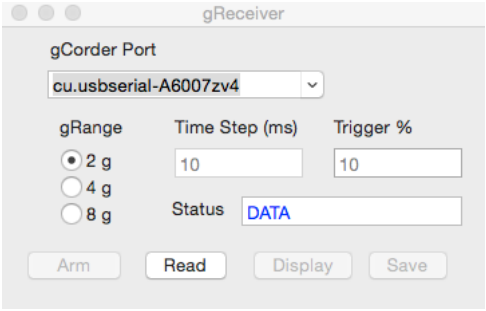

Figure 15: Data ready

Then you can press Read to read the data and from that point you can Display it as a graph (see above) or save it to a CSV file.

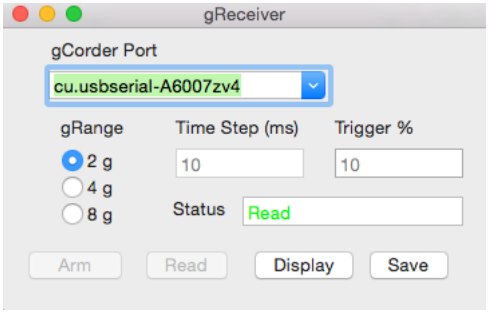

Figure 16: Display or Save

2.6.5 Output

This is the output from the setup above and dropping the unit from a few inches above the table top.

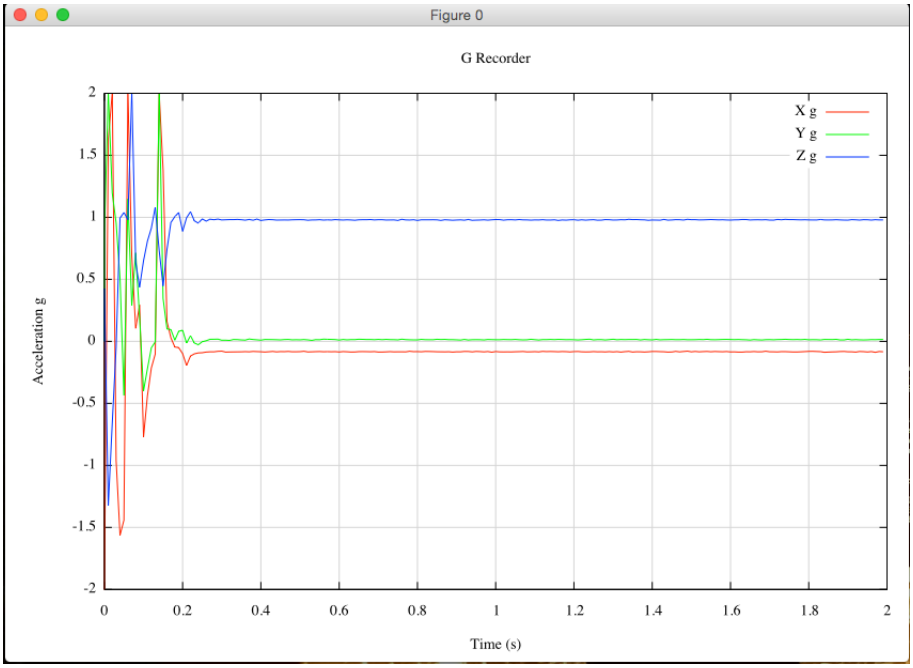

Figure 17: Results

You can see X,Y & Z acceleration. The MM8451 board shows the orientation of the sensor and Z axis is verticle in this case. It has as expected a value of 1g in the steady state case.

2.6.6 Program Internals

The program is a very conventional single window program. The major difference from standard is how the data is handled. We see from from the gRecorderInterface library that the data is read by it from sensor and stored in the library object.(2.5.2.4 on page 18). The data is a string that looks like:

```
Range 2
Trigger 10
0,32,470,4094
6,38,462,4094
12,40,466,4091
18,30,464,4115
24,35,460,4083
30,34,458,4094
    :
    :
```

Where the first column is the sensor time, the second is Xg, the third is Yg and the fourth is Zg. To use this data we must convert the g values from their _/- 4096 values to fractional g values. This is all done using AWK scripts[2]and command line operations.

The awk scripts and data (see above) are copied to /tmp and from there we run command scripts to either generate a CSV file or display the data using gnuplot.

For CSV generation we use the following script:

```
awk -f /tmp/csvprep.awk /tmp/data.txt > [Our CSV file]
```

For the display we use:

```
awk -f /tmp/dataprep.awk /tmp/data.txt > /tmp/plotdata.txt
/Applications/Gnuplot.app/Contents/Resources/bin/gnuplot /tmp/gplot.txt
```

The awk and gnuplot files are stored in the resources of the program as you can see in the XCode project.

References

[1] Arduino. Arduino home page.

[2] Bruce Barnett. *Awk - A Tutorial and Introduction*. Unix Tutorials, http://www.grymoire.com/Unix/Awk.html, November 2014.

[3] Freescale Semiconductor. *Xtrinsic MMA8451Q 3-Axis,Xtrinsic MMA8451Q 3-Axis,*, rev 8.1 edition, October 2014.