Portable Data Acquisition Device
Technical Report
Amy Reilly & Shawna Stuckey
A thesis submitted to the
Faculty of the Electrical & Computer Engineering Technology Program
of the University of Cincinnati

This document is a partial fulfillment of the
requirements for the degree of

Bachelor of Science

in Electrical and Computer Engineering Technology
at the College of Engineering & Applied Science

by

Amy Reilly
Shawna Stuckey

Bachelor of Science University of Cincinnati

April 2013

Faculty Advisor: James O. Everly

Acknowledgements:

Employees of Signalys for their support, expertise and funding, including: Neil Coleman, Phil Wilkin, Bob Coleman, Sam Luckett, Kyle Coleman, Rob Longbottom, and Calvin Mayes

Professors for the education that made this project possible, including: James Everly, Elvin Stepp, Michael Haas, Xuefu Zhou
TABLE OF CONTENTS

ABSTRACT................................................................................................................................................. 5

INTRODUCTION............................................................................................................................................... 6
  Problem .................................................................................................................................................. 6
  Solution .................................................................................................................................................. 7
  Credibility ................................................................................................................................................ 8
  Amy Reilly ........................................................................................................................................... 8
  Shawna Stuckey .................................................................................................................................... 8
  Goals and Methodology ......................................................................................................................... 8
  Goals ...................................................................................................................................................... 8
  Overview ............................................................................................................................................... 11

DISCUSSION.................................................................................................................................................. 11
  Concept .................................................................................................................................................. 11
  Design Objectives .................................................................................................................................. 11
    Accelerometer Requirements .................................................................................................................. 11
    Microcontroller .................................................................................................................................. 12
    Ethernet Capabilities .............................................................................................................................. 12
    Power .................................................................................................................................................... 12
    Sample Rate ...................................................................................................................................... 12
    Webpage Design .................................................................................................................................. 12
    Archiving Data ..................................................................................................................................... 12
    Portable ................................................................................................................................................ 12
    Storage ............................................................................................................................................... 12
  Technical Approach ................................................................................................................................ 13
    Accelerometer .................................................................................................................................... 13
    Microcontroller .................................................................................................................................. 14
    Communication .................................................................................................................................... 15
    SQL Database ...................................................................................................................................... 17
    Web Application ................................................................................................................................... 18
  Budget ..................................................................................................................................................... 20
  Problems Encountered ............................................................................................................................. 22
  Firmware .............................................................................................................................................. 22
Communication .................................................................................................................. 22
Speed .................................................................................................................................. 22
Future Recommendations ..................................................................................................... 23
Communication .................................................................................................................... 23
Data Storage ........................................................................................................................ 23
Weather Proof ....................................................................................................................... 23
Power ................................................................................................................................... 23

CONCLUSION ................................................................................................................... 24

SOURCES ............................................................................................................................ 25

APPENDIX A ....................................................................................................................... 26
MMA8453_n0m1 Header File ............................................................................................... 26
MMA8453_n0m1 Implementation File .................................................................................. 31
Arduino Sketch – Microcontroller Code ............................................................................. 38
ASP.NET Web Application Code .......................................................................................... 40
SQL Database Query Code .................................................................................................. 46

APPENDIX B ....................................................................................................................... 47
MMA8452Q Breakout Board Schematic .............................................................................. 47

APPENDIX C ....................................................................................................................... 48
Datasheets ........................................................................................................................... 48
ATmega328 .......................................................................................................................... 48
MMA8452Q .......................................................................................................................... 52
W5100 .................................................................................................................................. 53

FIGURES
Figure 1: Portable Data Acquisition Device at Tech Expo ................................................... 6
Figure 2: Flow Chart ............................................................................................................ 10
Figure 3: MMA8452Q Breakout Board [7] ......................................................................... 13
Figure 4: MMA8452Q Breakout Board Schematic [7] .......................................................... 14
Figure 5: Arduino Uno [8] ................................................................................................... 15
Figure 6: Arduino Ethernet Shield with PoE Module [9] ................................................. 16
Figure 7: Block Diagram .................................................................................................... 16
Figure 8: Gobetwino Log File Setup ............................................................................... 17
Figure 9: X-axis Time History Graph ............................................................................... 19
Figure 10: X-axis RMS vs. Time Graph ............................................................................ 19
Figure 11: Portable Data Acquisition Device ..................................................................... 24

TABLES
Table 1: Estimated Hardware Cost .................................................................................. 20
Table 2: Estimated Software Cost ................................................................................... 20
Table 3: Actual Hardware Cost ....................................................................................... 21
Table 4: Actual Software Cost ....................................................................................... 21
Table 5: Lab Cost ........................................................................................................... 21
ABSTRACT

There are various structures (buildings, bridges, etc) that, if monitored, people would be able to improve upon. Unfortunately, monitoring these structures would require sending someone to a site daily or weekly to take readings from a meter. Not every company has the funds to support this logic, and even if they do it would be an inefficient use of time and money. For this reason, it is necessary to create a portable data acquisition device that can take measurements from these structures and allow them to be monitored remotely. This paper discusses the methodology and functionality of the portable data acquisition device that has been created for use by Signalysis, Inc.
INTRODUCTION

Quality control is an important piece of engineering. One of the keys to measuring quality is collecting data for analysis. This is easily done when testing a stationary object, but what can be done to test the structure of a bridge? Or the sturdiness of a wind turbine? The Portable Data Acquisition Device, as shown in Figure 1 at the Technical Exposition, is a device that helps to solve this problem.

This technical report outlines the ten-month process involved in conceiving, conceptualizing, designing, prototyping, and completing the Portable Data Acquisition Device.

**Problem**

By monitoring structures – such as buildings, bridges, or wind turbines – it is possible to catch problems before they become disasters. Monitoring these structures, however, is easier said than done. Normally, it would require sending someone to a site daily or weekly to take readings from a meter. Not every company has the funds to support this logic, and even if they do it would be an inefficient use of time and money.

Currently, all devices that remotely monitor vibration waves are specifically aimed at one market. There are several companies that monitor water systems, such as Telog, Teledyne ISCO, and Hach. A company called Ludowici Meshcape provides a device
that monitors mining equipment via a smartphone application. This is fine if the ones that exist fit the market you need, but they are also very costly.

There is a company named Campbell Scientific that appears to handle a range of industries. They have several “structural monitoring applications” that appear to handle the same type of monitoring our device will provide. Although this company provides applications similar to what our device will do, their systems implement different devices for different structures (e.g., one device for monitoring bridges and a different device for monitoring building), whereas our device is universal.

The significance of resolving this problem is that companies need to monitor structure(s) to make sure that they are operating properly. If they are able to this, they will be able to catch problems early and fix them before any real damage is done. This will save them money and in the long run it will be well worth the cost.

This problem can be caused by various factors. It can stem from the lack of budget a company has to tend to the issues. The location of the structure needing to be monitored can play a factor. Also, not every company has the time or resources to put into solving a problem like this.

Solution
The solution to this problem is to build a portable data acquisition device that can be monitored remotely. The data is recorded by the device and then stored for archiving. Every time the display updates, the data is downloaded to a database and becomes available on a website where it can be viewed and analyzed by a user. The device is called a Portable Vibration Monitoring Device.

This is a solution because it is capable of taking data without the use of humans. The device holds power and is able to take readings as often as needed. This solution resolves the problem because it provides a way to collect the data, store it, and view the performance of the structure. The device will read the vibration data from the structure and record the data that can later be viewed and examined by the end user.

We implemented the solution by creating a device that can record the data. We also designed a webpage application that allows the user to monitor a threshold to determine if the measurements exceed the set limits. The website allows an end user to view the time history results and examine the data that was collected from the device.

As senior students in the programs of Electrical Engineering Technology and Computer Engineering Technology, our coursework has provided us with the knowledge necessary to solve a problem like this. We have also both worked as co-ops at Signalysis, who specializes in this type of analysis. We have the means to create a complete device that will handle the task without issues.
Credibility
Qualifications were key when completing this project. The collective knowledge gained through academic and professional experience between Shawna Stuckey and Amy Reilly spans 9 years.

Amy Reilly
Amy’s primary focus in this project was intended to be hardware with some assistance with software development. She has enjoyed hands on projects since her childhood. A course in electronics where she built a radio led her to pursue a bachelor’s degree in EET at the University of Cincinnati. In her coursework, she has taken classes in circuit analysis, electronics, computer interfacing, and embedded systems. Her co-op experience at Signalysis has introduced her to the science of accelerometer data analysis and given her experience in researching parts for specific projects. The coursework at UC along with co-op experience has led her to be successful in parts research, hardware design and microcontroller programming during this project.

Shawna Stuckey
Shawna’s focus for the project was the software portion. She was first introduced to Visual Studio and C++ programming upon entering the University of Cincinnati. The courses that were required led Shawna to be capable to produce the software for the project. The Computer Engineering Technology course series was the most helpful when creating the software to fit the needs of the project. Also, working at Signalysis has furthered her knowledge in C++ programming and has introduced the basics of the use of an accelerometer. With the software background she has obtained through coursework and coop experience, Shawna has completed all of the requirements to obtain a bachelor’s degree in Computer Engineering Technology. Shawna was able to interface the accelerometer with the microcontroller and successfully access the collected data to be analyzed.

Goals and Methodology

Goals
This device is intended to demonstrate that the microcontroller can read the values of vibration from an accelerometer and be used to monitor structures by simply attaching the unit. The unit collects data in real time.

Our main goals in completing this project were:
- To create a portable data acquisition device that can be monitored remotely
- To become familiar with microcontroller programming
- To become familiar with web design
- To learn how to process accelerometer data
Although the device requires the use of a serial cable, it does not have to be positioned right next to a computer; it can be as far away from the computer as the cable allows. Becoming familiar with microcontroller programming was inevitable because it was required in order to communicate with the accelerometer via I²C bus. The MMA8453_n0m1 library was discovered, which made it possible to read and process the accelerometer data over the I²C bus. Some web application design had been introduced during some coursework, but it was perfected by designing the webpage for this application.

Methodology
The flow chart shown in Figure 2 describes the flow of data through our system. Essentially, the data is retrieved by the accelerometer, which communicates with the Arduino microcontroller over I²C bus. Then, the Arduino sends each set of X/Y/Z measurements over serial communication to the local computer for storage before the web application moves the data into the SQL database. Each time the webpage is refreshed, new data is read in from the database and graphed as time history and RMS values.
Figure 2: Flow Chart
Overview
The remainder of this report describes the technical design of this senior capstone project and how it was completed. The following sections are included: design objectives, technical approach, budget, problems, and future recommendations.

DISCUSSION

Concept
The concept of this project was developed by Signalysis, Inc. Signalysis is a company that specializes in quality control by means of analyzing sound and vibration waves. This company expressed a need for a portable data acquisition device that can be accessed from a remote location. There are software programs that analyze vibrations and some devices out there that offer similar solutions, but those devices are more specialized and costly than this solution.

Signalysis needs this product because one of their customers has wind turbines that need to be monitored in central Ohio. As it is now, it is difficult to monitor the wind turbines to see if they are operating properly, so Signalysis asked for assistance to solve the problem. To make the product more versatile, it was decided to make this device available for any industry rather than just focus on wind turbines. Signalysis provides custom quality control systems to their customers, so tweaking small aspects of this product to fit the customer will be nothing new for them.

Design Objectives
An explanation of the design objectives are as follows:

Accelerometer Requirements
For the accelerometer, it was desired that it be a digital accelerometer. Digital accelerometers are more accurate than analog, and analog accels require additional circuitry to acquire the signal. Digital accelerometers interface very easily with microcontrollers once the I²C or SPI bus communication is established.

According to Professor Helmicki, an expert on the subject of structural analysis, bridge analysis deals with micro-G’s values (G’s are a unit of acceleration), so it was also required that the range of the digital accelerometer be as small as possible.

In the initial design, a BMA180 digital accelerometer was selected.
Portable Data Acquisition Device

**Microcontroller**
A microcontroller was required in order to retrieve the data from the accelerometer and send it to a computer efficiently. A Netduino Plus was selected because of capabilities of .NET programming and on-board Ethernet and SD card slot.

**Ethernet Capabilities**
Ethernet capabilities were desired in order to send data from the microcontroller to the local computer over the internet.

**Power**
For the scope of this project, Ethernet capabilities are already required so the Portable Data Acquisition Device will need to be powered via a Power over Ethernet module (PoE).

**Sample Rate**
In the world of Signal Analysis, sample rates should be as fast as possible. For the capabilities of the Netduino Plus microcontroller, this is approximately 18-20 samples per second.

**Webpage Design**
The webpage needs to include several graphs of data including time history and RMS values for each axis, X/Y/Z. For each axis, there will be a separate time history and RMS graph, resulting in six graphs total.

**Archiving Data**
The initial design intended data to be archived in text files. This was deemed acceptable per Signal Analysis requirements.

**Portable**
A portable device will be capable of being placed in a location that is not directly next to the computer it is communicating with.

**Storage**
It is intended that data will be stored on the SD card slot available on the microcontroller. Using this method, higher data rates can hopefully be achieved because data will be stored locally rather than constantly transmitted.
**Technical Approach**

During the implementation of the design objectives, some of the previously specified criteria were altered in order to create the best product in the allotted time. The technical approach taken during the design was in-depth and much research was done. All of the code from each program can be found in Appendix A, schematics in Appendix B, and specific details of each component are shown in the datasheets found in Appendix C.

**Accelerometer**

When it was discovered that the previously selected BMA180 was obsolete, research led to a new selection of the MMA8452Q digital accelerometer. This device can seem in *Figure 3*.

![Figure 3: MMA8452Q Breakout Board [7]](image)

The MMA8452Q has several selectable ranges available to use – the needs of this project require a small range, so the smallest range of ± 2G was chosen. Most digital accelerometers are capable of using SPI or I²C for digital communication, but the MMA8452Q is only able to use I²C. The schematic of the breakout board for this accelerometer is shown in *Figure 4* below. This breakout board is ideal because it includes the two 10KΩ pull-up resistors required for correct I²C operation.
When doing research, a library was found that allows an Arduino program to communicate with the MMA8452Q accel using simple commands. The code for this library can be found in Appendix A.

**Microcontroller**
Although a Netduino Plus was selected in the design phase of the project, there were issues once the accelerometer was upgraded to the MMA8452Q. Attempts were made to upgrade the firmware in order to accommodate the needs of the new accel, but all attempts were unsuccessful. Hardware was upgraded to the Netduino Plus 2 because the necessary firmware came pre-installed on this board, but communication was never successfully made.

A final microcontroller change was made when the Arduino Uno was selected (shown in *Figure 5*). Specifications for this microcontroller can be found in Appendix C. This microcontroller uses its own language and compiler; the language is very similar to C++ and the manufacturer’s webpage is very informative with examples on how to use it.
The program loaded onto the Arduino can be found in Appendix A. Essentially, this code waits for the interrupt bit to fire from the accel over the I²C bus. When it detects the interrupt bit, it reads the X/Y/Z values from the accel and stores them into local variables. The values received from the accel are a binary value from -256 to +256, so then the program maps that value to its appropriate value on the pre-determined ± 2G scale. Finally, the Arduino sends this data over serial communication to be written directly into a text file.

The sample rate for data taken by the accelerometer depends on the capabilities of the microcontroller and the connection speed. In this case, the best sample rate achieved was 15 samples per second, but the connection speed slowed this down significantly as time went on. For demonstration, only 2 samples per second were used in order for the results to display the most up-to-date information on the graphs. The code in Appendix A includes a 500 ms delay to ensure that only 2 samples per second are recorded.

Communication
Initially, it was desired to use the Arduino in a client/server application over the internet using the Ethernet Shield with Power over Ethernet Module (shown in Figure 6).
Because there was no Ethernet access available at the Technical Exposition and making this internet connection proved more difficult than anticipated, it was decided to use serial communication. A block diagram of the connections can be seen in Figure 7.

The program Gobetwino acts as the server in the serial communication application. This program allows the Arduino to perform tasks it could not do on its own — such as logging data directly to a text file. Another important feature of Gobetwino is that it is able to write a timestamp with each set of data it receives as it writes to the text file. This feature is key when graphing the data on the webpage and performing RMS calculations in the SQL database query. It was for that reason that the decision was made to log data directly to the text file rather
than first storing it on the SD card and then transferring the whole chunk. Figure 8 shows how to setup the Gobetwino application.

There are several different types of commands; “LGFIL,” or log file, is the desired command type for this application. The command name can be whatever the user wants, it just has to match the line in the Arduino Sketch. Next, the file to be written to must be selected. This file can be anywhere on the computer. This was ideal because in order for the web application to retrieve the data, it needed to be located in a specific location. Finally, the time stamp check box must be checked in order to keep track of time. The Arduino cannot keep time on its own, so this step is crucial.

![Gobetwino Log File Setup](image)

**Figure 8: Gobetwino Log File Setup**

**SQL Database**
The database used in this application is managed using SQL Server Management Studio. The database, called ArduinoData, archives all data retrieved from the MMA8452Q accel via the Arduino and Gobetwino applications.
Within the database there are two tables: AccelData (for the G-values of each axis with their timestamps) and RMS Data (for the RMS values with associated timestamp). The RMS Data table uses a query to calculate the RMS value for each second of data using the data in the AccelData table. The query code can be found in Appendix A.

**Web Application**

The web application is written using Visual Studios 2010 in ASP.NET. A screen shot of the webpage can be seen in *Figure 9* below. This application displays two graphs for each axis of the MMA8452Q: time history and RMS versus time.

The data from the text file is copied into the SQL database each time the webpage refreshes. Once the data is copied, all of that data is deleted from the text file so that the file does not get too big.

The webpage program reads data from the database and displays it in Time History graphs showing G-value vs. time for each axis. It does the same to display the RMS vs. time values in separate graphs; there are six graphs total.

Another capability of the webpage is that the user can set control limits based on what they will expect the data to be. When the control limits are set, as soon as the page is refreshed and new data is gathered, the graph turns red to alert the user of a problem if the values in the graph exceed the set limits. These values default to ±2 because the range of the accel is ± 2G so they values will never exceed them.

Screen shots of the graphs can be seen in *Figure 9* and *Figure 10*. The X-axis Time History is shown in *Figure 8* and an RMS graph can be seen in and *Figure 10*. The control limit settings can also be seen in *Figure 9*. 
Figure 9: X-axis Time History Graph

Figure 10: X-axis RMS vs. Time Graph


**Budget**

The projected budget is shown below in Table 1 and Table 2; the actual budget is displayed in Table 3 and Table 4.

### Table 1: Estimated Hardware Cost

<table>
<thead>
<tr>
<th>No.</th>
<th>Part Description</th>
<th>Part Number(s)</th>
<th>Manufacturer</th>
<th>Qty.</th>
<th>Est. Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accelerometer</td>
<td>BMA180</td>
<td>Bosch</td>
<td>1</td>
<td>$29.95</td>
</tr>
<tr>
<td>2</td>
<td>Microcontroller</td>
<td>Netduino Plus</td>
<td>Netduino</td>
<td>1</td>
<td>$60.00</td>
</tr>
<tr>
<td>3</td>
<td>Ethernet Connection</td>
<td>Netduino Plus</td>
<td>X</td>
<td></td>
<td>---</td>
</tr>
<tr>
<td>4</td>
<td>Product Case</td>
<td></td>
<td></td>
<td>1</td>
<td>$40.00</td>
</tr>
<tr>
<td>5</td>
<td>Power Supply</td>
<td></td>
<td></td>
<td>1</td>
<td>$100.00</td>
</tr>
</tbody>
</table>

**Total Estimated Cost:** $229.95

### Table 2: Estimated Software Cost

<table>
<thead>
<tr>
<th>No.</th>
<th>Part Description</th>
<th>Qty.</th>
<th>Est. Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Microsoft Visual Studio 2010</td>
<td>1</td>
<td>(Free)</td>
</tr>
<tr>
<td>2</td>
<td>SQL Server Management Studio</td>
<td>1</td>
<td>(Free)</td>
</tr>
<tr>
<td>3</td>
<td>Microsoft SQL Server 2008</td>
<td>1</td>
<td>(Free)</td>
</tr>
<tr>
<td>4</td>
<td>Server Computer</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Server OS Software</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Total Estimated Cost:** $0.00

Table 1 shows that the Netduino Plus was selected, and because it includes Ethernet capabilities, item 3 is technically a non-item. Table 2 shows the software and other parts that will be required to successfully complete this project. The first three items are software that are available for a free download because we are students. At the time, it was not clear yet if a server computer with a server operating system would be needed, but it was assumed that Signalysis would have an extra computer they could have provided us with if necessary.
Table 3: Actual Hardware Cost

<table>
<thead>
<tr>
<th>No.</th>
<th>Part Description</th>
<th>Part Number(s)</th>
<th>Manufacturer</th>
<th>Qty</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accelerometer</td>
<td>MMA8452Q</td>
<td>Freescale</td>
<td>1</td>
<td>$9.95</td>
</tr>
<tr>
<td>2</td>
<td>Microcontroller</td>
<td>Arduino</td>
<td>Arduino</td>
<td>1</td>
<td>$29.95</td>
</tr>
<tr>
<td>3</td>
<td>Ethernet Connection</td>
<td>Ethernet Shield w/PoE</td>
<td>Arduino</td>
<td>1</td>
<td>$59.95</td>
</tr>
<tr>
<td>4</td>
<td>Product Case</td>
<td>Crib for Arduino</td>
<td>Arduino</td>
<td>1</td>
<td>$29.95</td>
</tr>
<tr>
<td>5</td>
<td>Ethernet Faceplate</td>
<td>Ethernet Faceplate</td>
<td>Arduino</td>
<td>1</td>
<td>$4.95</td>
</tr>
</tbody>
</table>

Total Actual Cost: $134.75

Table 4: Actual Software Cost

<table>
<thead>
<tr>
<th>No.</th>
<th>Part Description</th>
<th>Qty</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Microsoft Visual Studio 2010</td>
<td>1</td>
<td>(Free)</td>
</tr>
<tr>
<td>2</td>
<td>SQL Server Management Studio</td>
<td>1</td>
<td>(Free)</td>
</tr>
<tr>
<td>3</td>
<td>Microsoft SQL Server 2008</td>
<td>1</td>
<td>(Free)</td>
</tr>
</tbody>
</table>

Total Actual Cost: $0.00

Table 3 and 4 show that the actual cost turned out to be less than the projected cost. The power supply that was included in the estimated cost was not needed, and the microcontroller, Ethernet capability, and accelerometer changed from the design phase to the implementation phase.

Table 5: Lab Cost

<table>
<thead>
<tr>
<th>No.</th>
<th>Part Description</th>
<th>Qty (Hrs)</th>
<th>Labor Rate</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Amy’s Hours</td>
<td>200</td>
<td>$25/hr</td>
<td>$5,000</td>
</tr>
<tr>
<td>2</td>
<td>Shawna’s Hours</td>
<td>200</td>
<td>$25/hr</td>
<td>$5,000</td>
</tr>
</tbody>
</table>

Total R&D Cost: $10,000

The final cost of all hardware, software, and labor comes out to be $10,134.75. The Netduino Plus 2 was not included in the budget even though the part was purchased. Although the Ethernet Shield with PoE Module was not used to its potential, it was included in Table 3.
Problems Encountered

Firmware
Issues with the firmware arose when the hardware was upgraded. Unlike the BMA180, the MMA8452Q required an I²C library that was capable of handling its repeated start (RS) bit.

The first solution was to upgrade the firmware on the original microcontroller, the Netduino Plus. Although the instructions were followed step by step, loading the new version of firmware to the Netduino Plus was unsuccessful.

The next solution was to upgrade the microcontroller to the Netduino Plus 2. This was a solution because this Netduino already had the updated firmware on it. Although the I²C library used on the Netduino Plus 2 supported the necessary RS bit for the MMA8452Q, unknown problems occurred and communication was never completed between the two devices.

The final solution was to alter the design and use the Arduino Uno as the microcontroller. Amy’s experience with this microcontroller made it the obvious next step for completing the project.

Communication
The Ethernet and SD card both use the same pins to communicate with the Arduino, so sending data from the SD card over Ethernet would have taken up a bit of valuable time. When this was realized, it was decided to write data directly over Ethernet as soon as it is acquired.

Unfortunately, attempts to create a server located on the local computer to receive the data over the internet were never successful. Data could be written from the Arduino directly to a webpage, but that was not in our design criteria. Because these issues were never resolved, it was decided to utilize the serial connection to the local computer to transfer data.

Speed
Speed was an issue because the accelerometer and microcontroller were capable of taking data faster than it could be processed. For this reason, the time delay in the microcontroller program had to be slowed down significantly so that all the data that was collected could be processed. Taking data at the rate that was used would not be ideal.
Future Recommendations

Communication
To improve the communication and fulfill the original design criteria, it would be ideal to establish communication over Ethernet. The Ethernet Shield should be used to transfer the data via Ethernet to a server, which timestamps the data and archives it to the database.

Once that communication is established, the device can become truly portable if wireless technology is implemented. That was beyond the scope of this project, but it would be a very useful addition to the project.

Data Storage
For speed, it would be best to use buffers and/or the SD card to store data. By doing so, the data can be transferred in one group over Ethernet to be processed and timestamped.

Weather Proof
Since the project was designed to measure structures, the packaging will have to be able to withstand various weather conditions. A simple water-proof case to enclose the microcontroller and circuitry should suffice as long as it is sealed and will not affect the free movement of the accelerometer.

Power
Although this was outside the scope of this project, once the device is completely wireless it will need to also be powered wirelessly. This can be achieved by using solar panels and/or batteries. This technology is developing and would likely be able to provide a power solution for this device.
CONCLUSION

Once the data is acquired and graphed, an end user can perform many types of analysis. With the data, a baseline can be determined and the variations from that line can be monitored for safety or general purposes. From there, it is possible to determine how the structure is affected by different conditions and forces. The X / Y / Z values can be evaluated through an FFT (Fast Fourier Transform) algorithm to determine the frequencies of the vibrations.

There is a need for a product like our Portable Vibration Monitoring Device because it will save companies time and money in several aspects of the business. We have demonstrated that we can build this product, and with Signalysis’s guidance and funding we were able to create a product that tests the capabilities of a microcontroller to do this type of analysis.

Figure 11: Portable Data Acquisition Device
SOURCES

10. Available from: http://www.mikmo.dk/gobetwino.html
APPENDIX A

MMA8453_n0m1 Header File

Obtained from source [6]

/***********************************************
*                                            *
* Name    : MMA8453_n0m1 Library              *
* Author   : Noah Shibley, Michael Grant, NoMi Design Ltd. http://n0m1.com *
* Date     : Feb 10th 2012                    *
* Version  : 0.1                             *
* Notes    : Arduino Library for use with the Freescale MMA8453Q via i2c.  *
*          : Some of the lib source from Kerry D. Wong                        *
*          : http://www.kerrywong.com/2012/01/09/interfacing-mma8453q-with-     *
*          : arduino/                                                          *
*                                            *
* This file is part of MMA8453_n0m1.        *
*                                            *
* MMA8453_n0m1 is free software: you can redistribute it and/or modify      *
* it under the terms of the GNU General Public License as published by      *
* the Free Software Foundation, either version 3 of the License, or         *
* (at your option) any later version.                                            *
*                                            *
* AtTouch is distributed in the hope that it will be useful,                 *
* but WITHOUT ANY WARRANTY; without even the implied warranty of             *
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the               *
* GNU General Public License for more details.                                *
*                                            *
* You should have received a copy of the GNU General Public License          *
* along with MMA8453_n0m1. If not, see <http://www.gnu.org/licenses/>.        *
*                                            *
*******************************************************************************/

#ifndef MMA8453_N0M1_H
#define MMA8453_N0M1_H

//uncomment PINCHANGE_INT to use INT pins other then arduino pins 2 & 3
//include the library from: http://code.google.com/p/arduino-pinchangeint/

//#define PINCHANGE_INT

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#ifdef PINCHANGE_INT
    #include <PinChangeInt.h>
    #include <PinChangeIntConfig.h>
#endif

#include <I2C.h>
const byte REG_STATUS = 0x00; // (R) Real time status
const byte REG_OUT_X_MSB = 0x01; // (R) [7:0] are 8 MSBs of 10-bit sample
const byte REG_OUT_X_LSB = 0x02; // (R) [7:6] are 2 LSBs of 10-bit sample
const byte REG_OUT_Y_MSB = 0x03; // (R) [7:0] are 8 MSBs of 10-bit sample
const byte REG_OUT_Y_LSB = 0x04; // (R) [7:6] are 2 LSBs of 10-bit sample
const byte REG_OUT_Z_MSB = 0x05; // (R) [7:0] are 8 MSBs of 10-bit sample
const byte REG_OUT_Z_LSB = 0x06; // (R) [7:6] are 2 LSBs of 10-bit sample
const byte REG_SYSMOD = 0x0b; // (R) Current system mode
const byte REG_INT_SOURCE = 0x0c; // (R) Interrupt status
const byte REG_WHO_AM_I = 0x0d; // (R) Device ID (0x3A)
const byte REG_XYZ_DATA_CFG = 0xe; // (R/W) Dynamic range settings
const byte REG_HP_FILTER_CUTOFF = 0x0f; // (R/W) cut-off frequency is set to 16Hz @ 800Hz
const byte REG_PL_STATUS = 0x10; // (R) Landscape/Portrait orientation status
const byte REG_PL_CFG = 0x11; // (R/W) Landscape/Portrait configuration
const byte REG_PL_COUNT = 0x12; // (R) Landscape/Portrait debounce counter
const byte REG_PL_BF_ZCOMP = 0x13; // (R) Back-Front, Z-Lock trip threshold
const byte REG_P_L_THS_REG = 0x14; // (R/W) Portrait to Landscape trip angle is 29 degree
const byte REG_FF_MT_CFG = 0x15; // (R/W) Freefall/motion functional block configuration
const byte REG_FF_MT_SRC = 0x16; // (R) Freefall/motion event source register
const byte REG_FF_MT_THS = 0x17; // (R/W) Freefall/motion threshold register
const byte REG_FF_MT_COUNT = 0x18; // (R/W) Freefall/motion debounce counter
const byte REG_TRANSIENT_CFG = 0x1d; // (R/W) Transient functional block configuration
const byte REG_TRANSIENT_SRC = 0x1e; // (R) Transient event status register
const byte REG_TRANSIENT_THS = 0x1f; // (R/W) Transient event threshold
const byte REG_TRANSIENT_COUNT = 0x20; // (R/W) Transient debounce counter
const byte REG_PULSE_CFG = 0x21; // (R/W) ELE, Double_XYZ or Single_XYZ
const byte REG_PULSE_SRC = 0x22; // (R) EA, Double_XYZ or Single_XYZ
const byte REG_PULSE_THSX = 0x23; // (R/W) X pulse threshold
const byte REG_PULSE_THSY = 0x24; // (R/W) Y pulse threshold
const byte REG_PULSE_THSZ = 0x25; // (R/W) Z pulse threshold
const byte REG_PULSE_TMLT = 0x26; // (R/W) Time limit for pulse
const byte REG_PULSE_LTCY = 0x27; // (R/W) Latency time for 2nd pulse
const byte REG_PULSE_WIND = 0x28; // (R/W) Window time for 2nd pulse
const byte REG_ASLP_COUNT = 0x29; // (R/W) Counter setting for auto-sleep
const byte REG_CTRL_REG1 = 0x2a; // (R/W) ODR = 800 Hz, STANDBY mode
const byte REG_CTRL_REG2 = 0x2b; // (R/W) Sleep enable, OS Modes, RST, ST
const byte REG_CTRL_REG3 = 0x2c; // (R/W) Wake from sleep, IPOL, PP_OD
const byte REG_CTRL_REG4 = 0x2d; // (R/W) Interrupt enable register
const byte REG_CTRL_REG5 = 0x2e; // (R/W) Interrupt pin (INT1/INT2) map
const byte REG_OFF_X = 0x2f; // (R/W) X-axis offset adjust
const byte REG_OFF_Y = 0x30; // (R/W) Y-axis offset adjust
const byte REG_OFF_Z = 0x31; // (R/W) Z-axis offset adjust
const byte FULL_SCALE_RANGE_2g = 0x0;  
const byte FULL_SCALE_RANGE_4g = 0x1;  
const byte FULL_SCALE_RANGE_8g = 0x2;

extern "C" void accelISR(void) __attribute__((signal));

class MMA8453_n0m1 {

public:
    friend void accelISR(void); //make friend so btnPressISR can access private var keyhit

    MMA8453_n0m1();

};
void setI2CAddr(int address);

void dataMode(boolean highRes, int gScaleRange);

int x();

int y();

int z();

void shakeMode(int threshold, boolean enableX, boolean enableY, boolean enableZ, boolean enableINT2, int arduinoINTPin);
* shake
* returns true if there is shaking (high pass filtered motion)
* *******************************************************************************/
boolean shake() { boolean shakeOut = shake_; shake_ = false; return shakeOut; }

/*********************************************************/
* * shakeAxisX
* returns true if there is shake on the x axis
* *******************************************************************************/
boolean shakeAxisX() { boolean shakeAxisOut = shakeAxisX_; shakeAxisX_ = false; return shakeAxisOut; }

/*********************************************************/
* * shakeAxisY
* returns true if there is shake on the y axis
* *******************************************************************************/
boolean shakeAxisY() { boolean shakeAxisOut = shakeAxisY_; shakeAxisY_ = false; return shakeAxisOut; }

/*********************************************************/
* * shakeAxisZ
* returns true if there is shake on the z axis
* *******************************************************************************/
boolean shakeAxisZ() { boolean shakeAxisOut = shakeAxisZ_; shakeAxisZ_ = false; return shakeAxisOut; }

/*********************************************************/
* * motionMode
* set to motion detection mode
* *******************************************************************************/
void motionMode(int threshold, boolean enableX, boolean enableY, boolean enableZ, boolean enableINT2, int arduinoINTPin);

/*********************************************************/
* * motion
* returns true if there is motion
* *******************************************************************************/
boolean motion() { boolean motionOut = motion_; motion_ = false; return motionOut; }

/*********************************************************/
* update
* update data values, or clear interrupts. Use at start of loop()
*/
void update();

/***********************************************************/
* * regRead
* *
***********************************************************/
void regRead(byte reg, byte *buf, byte count = 1);

/***********************************************************/
* * regWrite
* *
***********************************************************/
void regWrite(byte reg, byte val);

//-- Compatibility functions to match the api of the ADXL345 library
//-- http://code.google.com/p/adxl345driver/
//-- allows for more easy updates of code from the ADXL345 to the MMA8453
//--
void setRangeSetting(int gScaleRange) { gScaleRange_ = gScaleRange; } //call this before
setFullResBit()
void setFullResBit(boolean highRes) { dataMode(highRes,gScaleRange_); }
void readAccel(int *x, int *y, int *z) { xyz(*x,*y,*z); }

private:

    void xyz(int& x, int& y, int& z);
    void clearInterrupt();

    int x_, y_, z_;
    byte I2CAddr;

    boolean highRes_;  
    int gScaleRange_;  
    boolean dataMode_;  
    boolean shakeMode_;  
    boolean motionMode_;  

    boolean motion_;  
    boolean shake_;  
    boolean shakeAxisX_;  
    boolean shakeAxisY_;  
    boolean shakeAxisZ_;  

    volatile boolean ISRFlag;
```cpp
static MMA8453_n0m1* pMMA8453_n0m1; //ptr to MMA8453_n0m1 class for the ISR

```

```
MMA8453_n0m1 Implementation File

Obtained from source [6]

/**********************************************************************************
* Name    : MMA8453_n0m1 Library
* Author  : Noah Shibley, Michael Grant, NoMi Design Ltd. http://n0m1.com
* Date    : Feb 10th 2012
* Version : 0.1
* Notes   : Arduino Library for use with the Freescale MMA8453Q via i2c.
           
           Some of the lib source from Kerry D. Wong
           http://www.kerrywong.com/2012/01/09/interfacing-mma8453q-with-
           arduino/
* 
* This file is part of MMA8453_n0m1.
* 
* MMA8453_n0m1 is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
* 
* AtTouch is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
* 
* You should have received a copy of the GNU General Public License
* along with MMA8453_n0m1. If not, see <http://www.gnu.org/licenses/>.
* 
**********************************************************************************/

#include "MMA8453_n0m1.h"

MMA8453_n0m1* MMA8453_n0m1::pMMA8453_n0m1 = 0;

MMA8453_n0m1::MMA8453_n0m1()
{
    pMMA8453_n0m1 = this;
    dataMode_ = false;
    shakeMode_ = false;
    ISRFlag = false;
    shake_ = false;
    shakeAxisX_ = false;
    shakeAxisY_ = false;
    shakeAxisZ_ = false;
}
```

Portable Data Acquisition Device
I2CAddr = 0x1c;  //The i2C address of the MMA8453 chip. 0x1D is another common value.
gScaleRange_ = 2;  //default 2g

/***********************************************************
* setI2CAddr
*
***********************************************************/
void MMA8453_n0m1::setI2CAddr(int address)
{
    I2CAddr = address; I2CAddr;
}

/***********************************************************
* update
*
***********************************************************/
void MMA8453_n0m1::update()
{
    if(dataMode_)
    {
        xyz(x_,y_,z_);
    }
    if(shakeMode_ == true || motionMode_ == true)
    {
        clearInterrupt();
    }
}

/***********************************************************
* clearInterrupt
*
***********************************************************/
void MMA8453_n0m1::clearInterrupt()
{
    if(ISRFlag)
    {
        byte sourceSystem;
        I2c.read(I2CAddr,REG_INT_SOURCE,byte(1),&sourceSystem); //check which system fired the interrupt
        if((sourceSystem&0x20) == 0x20) //Transient
        {

}}
// Perform an Action since Transient Flag has been set
// Read the Transient to clear system interrupt and Transient
byte srcTrans;
shake_ = true;
I2c.read(I2CAddr, REG_TRANSIENT_SRC, byte(1), &srcTrans);

if (srcTrans & 0x02 == 0x02)
{
    shakeAxisX_ = true;
}
if (srcTrans & 0x08 == 0x08)
{
    shakeAxisY_ = true;
}
if (srcTrans & 0x20 == 0x20)
{
    shakeAxisZ_ = true;
}

ISRFlag = false;

if ((sourceSystem & 0x04) == 0x04) // FreeFall Motion
{
    byte srcFF;
    I2c.read(I2CAddr, REG_FF_MT_SRC, byte(1), &srcFF);
    motion_ = true;
}

**********************************************************
* xyz
* Get accelerometer readings (x, y, z)
* by default, standard 10 bits mode is used.
* This function also converts 2's complement number to
* signed integer result.
* If accelerometer is initialized to use low res mode,
* isHighRes must be passed in as false.
**********************************************************
void MMA8453_n0m1::xyz(int & x, int & y, int & z)
{
    byte buf[6];
    if (highRes_)
        
Portable Data Acquisition Device
I2c.read(I2CAddr, REG_OUT_X_MSB, 6, buf);
x = buf[0] << 2 | buf[1] >> 6 & 0x3;
y = buf[2] << 2 | buf[3] >> 6 & 0x3;
}
else
{
    I2c.read(I2CAddr, REG_OUT_X_MSB, 3, buf);
x = buf[0] << 2;
y = buf[1] << 2;
z = buf[2] << 2;
}
if (x > 511) x = x - 1024;
if (y > 511) y = y - 1024;
if (z > 511) z = z - 1024;

/***********************************************************
* dataMode
*
***********************************************************/
void MMA8453_n0m1::dataMode(boolean highRes, int gScaleRange)
{
    highRes_ = highRes;
gScaleRange_ = gScaleRange;
dataMode_ = true;
byte statusCheck;
byte activeMask = 0x01;
byte resModeMask = 0x02;

    //setup i2c
I2c.begin();

    //register settings must be made in standby mode
I2c.read(I2CAddr, REG_CTRL_REG1, byte(1), &statusCheck);
I2c.write(I2CAddr, REG_CTRL_REG1, byte(statusCheck & ~activeMask));
if (gScaleRange_ <= 3) { gScaleRange_ = FULL_SCALE_RANGE_2g; } //0-3 = 2g
else if (gScaleRange_ <= 5) { gScaleRange_ = FULL_SCALE_RANGE_4g; } //4-5 = 4g
else if (gScaleRange_ <= 8) { gScaleRange_ = FULL_SCALE_RANGE_8g; } //6-8 = 8g
else if (gScaleRange_ > 8) { gScaleRange_ = FULL_SCALE_RANGE_8g; } //boundary
I2c.write(I2CAddr, REG_XYZ_DATA_CFG, byte(gScaleRange_));

    //set highres 10bit or lowres 8bit
I2c.read(I2CAddr, REG_CTRL_REG1, byte(1), &statusCheck);
if (highRes) {
    I2c.write(I2CAddr, REG_CTRL_REG1, byte(statusCheck & ~resModeMask));
} else {
    I2c.write(I2CAddr, REG_CTRL_REG1, byte(statusCheck | resModeMask));
}

    //active Mode
void MMA8453_n0m1::shakeMode(int threshold, boolean enableX, boolean enableY, boolean enableZ, boolean enableINT2, int arduinoINTPin)
{
    #ifdef PINCHANGE_INT
        pinMode(arduinoINTPin, INPUT); digitalWrite(arduinoINTPin, HIGH);
        PCIntPort::attachInterrupt(arduinoINTPin, accelISR, FALLING);
    #else
        Serial.println("no INT on pin, define PINCHANGE_INT");
    #endif

    boolean error = false;
    byte statusCheck;

    //setup i2c
    I2c.begin();
    I2c.write(I2CAddr, REG_CTRL_REG1, byte(0x18)); //Set device in 100 Hz ODR, Standby
    byte xyzCfg = 0x10; //latch always enabled
    if(enableX) xyzCfg |= 0x02;
    if(enableY) xyzCfg |= 0x04;
    if(enableZ) xyzCfg |= 0x08;
    I2c.write(I2CAddr, REG_TRANSIENT_CFG, xyzCfg); //XYZ + latch 0x1E
    I2c.read(I2CAddr, REG_TRANSIENT_CFG, byte(1), &statusCheck);
    if(statusCheck != byte(threshold)) error = true;

    if(threshold > 127) threshold = 127; //8g is the max.
    I2c.write(I2CAddr, REG_TRANSIENT_THS, byte(threshold)); //threshold
    I2c.read(I2CAddr, REG_TRANSIENT_THS, byte(1), &statusCheck);
    if(statusCheck != byte(threshold)) error = true;
}

I2c.read(I2CAddr, REG_CTRL_REG1, byte(1), &statusCheck);
I2c.write(I2CAddr, REG_CTRL_REG1, byte(statusCheck | activeMask));
I2c.write(I2CAddr, REG_TRANSIENT_COUNT, byte(0x05)); //Set the Debounce Counter for 50 ms
I2c.read(I2CAddr, REG_TRANSIENT_COUNT, byte(1), &statusCheck);
if (statusCheck != 0x05) error = true;

I2c.read(I2CAddr, REG_CTRL_REG4, byte(1), &statusCheck);
statusCheck |= 0x20;
I2c.write(I2CAddr, REG_CTRL_REG4, statusCheck); //Enable Transient Detection Interrupt in the System
byte intSelect = 0x20;
if (enableINT2) intSelect = 0x00;
I2c.read(I2CAddr, REG_CTRL_REG5, byte(1), &statusCheck);
statusCheck |= intSelect;
I2c.write(I2CAddr, REG_CTRL_REG5, statusCheck); //INT2 0x0, INT1 0x20

I2c.read(I2CAddr, REG_CTRL_REG1, byte(1), &statusCheck); //Read out the contents of the register
statusCheck |= 0x01; //Change the value in the register to Active Mode.
I2c.write(I2CAddr, REG_CTRL_REG1, statusCheck); //Write in the updated value to put the device in Active Mode

if (error)
{
    Serial.println("Shake mode setup error");
    Serial.println("retrying...");
    delay(100);
    shakeMode(threshold, enableX, enableY, enableZ, enableINT2, arduinoINTPin);
}

shakeMode_ = true;

void MMA8453_m0n1::motionMode(int threshold, boolean enableX, boolean enableY, boolean enableZ, boolean enableINT2, int arduinoINTPin)
{
    if (arduinoINTPin == 2 || arduinoINTPin == 3)
    {
        arduinoINTPin = arduinoINTPin - 2;
        attachInterrupt(arduinoINTPin, accelISR, FALLING);
        //DataSheet pg40, When IPOL is ‘0’ (default value) any interrupt event will signaled with a logical 0
    }
    else
    {
        #ifdef PINCHANGE_INT
            pinMode(arduinoINTPin, INPUT); digitalWrite(arduinoINTPin, HIGH);
            PCintPort::attachInterrupt(arduinoINTPin, accelISR, FALLING);
        #else
            Serial.println("no INT on pin, define PINCHANGE_INT");
        #endif
    }
boolean error = false;
byte statusCheck;

//setup i2c
I2c.begin();

I2c.write(I2CAddr, REG_CTRL_REG1, byte(0x18)); //Set device in 100 Hz ODR,

Standby
byte xyzCfg = 0x80; //latch always enabled
xyzCfg |= 0x40; //Motion not free fall
if(enableX) xyzCfg |= 0x08;
if(enableY) xyzCfg |= 0x10;
if(enableZ) xyzCfg |= 0x20;

I2c.write(I2CAddr, REG_FF_MT_CFG, xyzCfg); //XYZ + latch + motion
I2c.read(I2CAddr, REG_FF_MT_CFG, byte(1), &statusCheck);
if(statusCheck != xyzCfg) error = true;

if(threshold > 127) threshold = 127; //a range of 0-127.
I2c.write(I2CAddr, REG_FF_MT_THS, byte(threshold)); //threshold
I2c.read(I2CAddr, REG_FF_MT_THS, byte(1), &statusCheck);
if(statusCheck != byte(threshold)) error = true;

I2c.write(I2CAddr, REG_FF_MT_COUNT, byte(0x0A)); //Set the Debounce Counter for 100 ms
I2c.read(I2CAddr, REG_FF_MT_COUNT, byte(1), &statusCheck);
if(statusCheck != 0x0A) error = true;

I2c.read(I2CAddr, REG_CTRL_REG4, byte(1), &statusCheck);
statusCheck |= 0x04;
I2c.write(I2CAddr, REG_CTRL_REG4, statusCheck);

//Enable Motion Interrupt in the System
byte intSelect = 0x04;
if(enableINT2) intSelect = 0x00;
I2c.read(I2CAddr, REG_CTRL_REG5, byte(1), &statusCheck);
statusCheck |= intSelect;
I2c.write(I2CAddr, REG_CTRL_REG5, statusCheck); //INT2 0x0, INT1 0x04

I2c.read(I2CAddr, REG_CTRL_REG1, byte(1), &statusCheck); //Read out the contents of the register
statusCheck |= 0x01; //Change the value in the register to Active Mode.
I2c.write(I2CAddr, REG_CTRL_REG1, statusCheck); //Write in the updated value to put the device in Active Mode

if(error)
{
    Serial.println("Motion mode setup error");
    Serial.println("retrying...");
    delay(100);
    motionMode(threshold, enableX, enableY, enableZ, enableINT2, arduinoINTPin);
}

Portable Data Acquisition Device
motionMode_ = true;

/***********************************************************
* regRead
* *
****************************************************/
void MMA8453_n0m1::regRead(byte reg, byte *buf, byte count)
{
    I2c.read(I2CAddr, reg, count, buf);
}

/***********************************************************
* regWrite
* *
****************************************************/
void MMA8453_n0m1::regWrite(byte reg, byte val)
{
    I2c.write(I2CAddr, reg, val);
}

/***********************************************************
* accelISR
* *
****************************************************/
void accelISR(void){
    MMA8453_n0m1::pMMA8453_n0m1->ISRFlag = true;
}

---

**Arduino Sketch – Microcontroller Code**

/********************
* Senior Design Project
* Portable Data Acquisition Device
* March 26, 2013
*
* Program to read accel data over I2C bus and
* write values to text file over serial comm port
* through Getbetwino program
*
* by Amy Reilly
********************/
#include <I2C.h>
#include <MMA8453_n0m1.h>

// Function prototype
void MapValue(double &X, double &Y, double &z);

// Instantiate object
MMA8453_n0m1 accel;

void setup()
{
    Serial.begin(9600);    // establish connection speed
    accel.setI2CAddr(0x1D); // set I2C address for accel
    accel.dataMode(true, 2); // set data mode

    // set motion mode to utilize interrupt
    * threshold [0-127] formula: 0.5g / 0.063g = 7.9 counts, round to 8 counts ,
    * enable X,
    * enable Y,
    * enable Z,
    * enable MMA8452Q INT pin 2 (false= pin 1),
    * arduino INT pin number */
    accel.motionMode(8, true, true, true, false, 2);

    // initialize CS pins for SD card and Ethernet
    pinMode(10, OUTPUT);
    pinMode(4, OUTPUT);
}

// end setup function

void loop()
{
    accel.update();    // read from I2C bus line

    if(accel.motion())    // if accel is in motion...
    {
        double X = accel.x();    // save X-value
        double Y = accel.y();    // save Y-value
        double Z = accel.z();    // save Z-value

        // map values to appropriate G value
        MapValue( X, Y, Z);

        // send command over Serial to Getbetwino
        // data gets written directly into a text file
        Serial.print("#S|ACCELDATA[");
        Serial.print(X);
        Serial.print(" ");
        Serial.print(Y);
        Serial.print(" ");
        Serial.print(Z);
        Serial.print(" ");
        Serial.println("#");
}
void MapValue(double &X, double &Y, double &Z)
{
    // if X values are out of range...
    if( X > 256 )
        X = 256;
    else if( X < -256 )
        X = -256;

    // if Y values are out of range...
    if( Y > 256 )
        Y = 256;
    else if( Y < -256 )
        Y = -256;

    // if Z values are out of range...
    if( Z > 256 )
        Z = 256;
    else if( Z < -256 )
        Z = -256;

    // map X values from +/- 256 counts to +/- 2 G's
    X = map( X, -256, 256, -2, 2);
    Y = map( Y, -256, 256, -2, 2);
    Z = map( Z, -256, 256, -2, 2);
} // end MapValue function

double map(double val, double in_min, double in_max, double out_min, double out_max)
{
    return (val - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
} // end map function

---

**ASP.NET Web Application Code**

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.IO;
using System.Data.SqlClient;
```

Portable Data Acquisition Device
namespace SeniorDesignWebsite
{
    public partial class _Default : System.Web.UI.Page
    {
    
        protected void Page_Load(object sender, EventArgs e)
        {
            TextBox1.Text = DateTime.Now.ToString(); //puts the current date and time in the textbox
            deleteData(); //runs the deleteData method
            createRMS(); //runs the createRMS method

            StreamReader sd = new StreamReader("example.txt"); //opens the file
            string text = sd.ReadLine(); //reads line by line what is in the file
            while (text != null)
            {
                string[] split = text.Split(';', ' '); //splits the text where the ';' is

                string[] ar = split[1].Split(' '); //takes the data after the space
                string x; //initializing variables
                string y; //initializing variables
                string z; //initializing variables
                string date; //initializing variables

                x = ar[0]; //giving the variables declaration in an array
                y = ar[1]; //giving the variables declaration in an array
                z = ar[2]; //giving the variables declaration in an array
                date = split[0]; //giving the variables declaration in an array

                //Use this code to put them in the Database.
                SqlConnection con = new SqlConnection("Server=(local);Database=ArduinoData;User ID=SigUser; Password=Signalys1s; ");
                SqlCommand Cmd = new SqlCommand();
                Cmd.Connection = con;
                con.Open();
                Cmd.CommandType = System.Data.CommandType.Text;
                Cmd.CommandText = "INSERT INTO AccelData VALUES (" + x + ", " + y + ", " + z + "," + date + ")";

                SqlDataReader reader = Cmd.ExecuteReader();
                con.Close();

                text = sd.ReadLine();

                if (TextBox2.Text != null)
                {
                    double xmax = Convert.ToDouble(TextBox2.Text.ToString()); //converts the integer to a string
                    if (xmax < Convert.ToDouble(x))
                    {
                    }
                }
            }
        }
    }
}
if (TextBox5.Text != null)
{
    double xmin = Convert.ToDouble(TextBox5.Text.ToString());  //converts the integer to a string
    if (xmin > Convert.ToDouble(x))
    {
    }
}
if (TextBox3.Text != null)
{
    double ymax = Convert.ToDouble(TextBox3.Text.ToString());  //converts the integer to a string
    if (ymax < Convert.ToDouble(y))
    {
    }
}
if (TextBox6.Text != null)
{
    double ymin = Convert.ToDouble(TextBox6.Text.ToString());  //converts the integer to a string
    if (ymin > Convert.ToDouble(y))
    {
    }
}
if (TextBox4.Text != null)
{
    double zmax = Convert.ToDouble(TextBox4.Text.ToString());  //converts the integer to a string
    if (zmax < Convert.ToDouble(z))
    {
    }
}
if (TextBox7.Text != null)
{
    double zmin = Convert.ToDouble(TextBox7.Text.ToString());  //converts the integer to a string
    if (zmin > Convert.ToDouble(z))
    {
    }
}
sd.Close(); //closes the file

File.WriteAllText(@"C:\Program Files (x86)\Common Files\Microsoft Shared\DevServer\10.0\example.txt", String.Empty); //empties the file

//Deletes the data from the SQL database
private void deleteData()
{
    SqlConnection con = new SqlConnection("Server=(local);Database=ArduinoData;User ID=SigUser; Password=Signalys1s; ");
    SqlCommand Comd = new SqlCommand();
    Comd.Connection = con;
    con.Open();
    Comd.CommandType = System.Data.CommandType.Text;
    List<string> time = new List<string>();
    Comd.CommandText = ("DELETE FROM [RMS Data]");
    SqlDataReader reader = Comd.ExecuteReader();
    reader.Close();
    con.Close();
}

//Creates the values from the SQL database for the RMS calculation
private void createRMS()
{
    SqlConnection con = new SqlConnection("Server=(local);Database=ArduinoData;User ID=SigUser; Password=Signalys1s; ");
    SqlCommand Comd = new SqlCommand();
    Comd.Connection = con;
    con.Open();
    Comd.CommandType = System.Data.CommandType.Text;
    List<string> time = new List<string>();
    Comd.CommandText = ("SELECT DISTINCT Time FROM AccelData");
    SqlDataReader reader = Comd.ExecuteReader();
    while (reader.Read())
    {
        time.Add(reader.GetString(0));
    }
    reader.Close();

    foreach (string timeVal in time)
    {
        Comd.CommandText = ("INSERT INTO [RMS Data] (XRMS,YRMS,ZRMS, DateTime) VALUES((SELECT SQRT((SUM(SQUARE(X))/COUNT(X))) FROM AccelData WHERE Time = "+ timeVal + ")", (SELECT SQRT((SUM(SQUARE(Y))/COUNT(Y))) FROM AccelData WHERE Time = "+ timeVal + ")", (SELECT SQRT((SUM(SQUARE(Z))/COUNT(Z))) FROM AccelData WHERE Time = "+ timeVal + ")");
    }
reader = Comd.ExecuteReader();
reader.Close();
}
con.Close();

protected void TextBox1_TextChanged(object sender, EventArgs e)
{
  TextBox1.Text = DateTime.Now.ToString();
}

protected void Button1_Click(object sender, EventArgs e)
{
  Response.Redirect(Request.RawUrl, true); //Refreshes the page
}

protected void TextBox2_TextChanged(object sender, EventArgs e)
{
}

protected void Chart1_Load(object sender, EventArgs e)
{
  Chart1.ChartAreas["ChartArea1"].AxisX.Title = "Date and Time"; //labels for the x axis
  Chart1.ChartAreas["ChartArea1"].AxisY.Title = "X value in G's"; //label for the y axis
  Chart1.Titles.Add("Time vs. X Value in G's"); //label for the title
}

protected void Chart2_Load(object sender, EventArgs e)
{
  Chart2.ChartAreas["ChartArea1"].AxisX.Title = "Date and Time"; //label for the x axis
  Chart2.ChartAreas["ChartArea1"].AxisY.Title = "Y value in G's"; //label for the y axis
  Chart2.Titles.Add("Time vs. Y Value in G's"); //label for the title
}

protected void Chart3_Load(object sender, EventArgs e)
{
  Chart3.ChartAreas["ChartArea1"].AxisX.Title = "Date and Time"; //label for the x axis
  Chart3.ChartAreas["ChartArea1"].AxisY.Title = "Z value in G's"; //label for the y axis
  Chart3.Titles.Add("Time vs. Z Value in G's"); //label for the title
}

protected void Chart4_Load(object sender, EventArgs e)
{
  Chart4.ChartAreas["ChartArea1"].AxisX.Title = "Date and Time"; //label for the x axis
  Chart4.ChartAreas["ChartArea1"].AxisY.Title = "X RMS Values"; //label for the y axis
  Chart4.Titles.Add("Time vs. X RMS"); //label for the title
```csharp
protected void Chart5_Load(object sender, EventArgs e)
{
    Chart5.ChartAreas["ChartArea1"].AxisX.Title = "Date and Time"; //label for the x axis
    Chart5.ChartAreas["ChartArea1"].AxisY.Title = "Y RMS Values"; //label for the y axis
    Chart5.Titles.Add("Time vs. Y RMS"); //label for the title
}

protected void Chart6_Load(object sender, EventArgs e)
{
    Chart6.ChartAreas["ChartArea1"].AxisX.Title = "Date and Time"; //label for the x axis
    Chart6.ChartAreas["ChartArea1"].AxisY.Title = "Z RMS Value"; //label for the y axis
    Chart6.Titles.Add("Time vs. Z RMS"); //label for the title
}

protected void Button2_Click(object sender, EventArgs e) //after button click convert the string to a double integer
{
    string strxmax = TextBox2.Text; //puts what is in the textbox to a string
double xmax = Convert.ToDouble(strxmax); //converts to a double integer
}

protected void Button8_Click(object sender, EventArgs e) //after button click convert the string to a double integer
{
    string strxmax = TextBox5.Text; //puts what is in the textbox to a string
double xmax = Convert.ToDouble(strxmax); //converts to a double integer
}

protected void Button3_Click(object sender, EventArgs e) //after button click convert the string to a double integer
{
    string strxmax = TextBox3.Text; //puts what is in the textbox to a string
double xmax = Convert.ToDouble(strxmax); //converts to a double integer
}

protected void Button9_Click(object sender, EventArgs e) //after button click convert the string to a double integer
{
    string strxmax = TextBox6.Text; //puts what is in the textbox to a string
double xmax = Convert.ToDouble(strxmax); //converts to a double integer
}

protected void Button4_Click(object sender, EventArgs e) //after button click convert the string to a double integer
{
    string strxmax = TextBox4.Text; //puts what is in the textbox to a string
double xmax = Convert.ToDouble(strxmax); //converts to a double integer
}

protected void Button7_Click(object sender, EventArgs e) //after button click convert the string to a double integer
{
    // Code for conversion
}
```
string strxmax = TextBox7.Text; //puts what is in the textbox to a string
double xmax = Convert.ToDouble(strxmax); //converts to a double integer

---

### SQL Database Query Code

```sql
--SELECT SQRT((SUM(SQUARE(X))/COUNT(X))) AS 'XRMS'
--FROM AccelData
--WHERE Time = '3/21/2013 1:10:23 PM'

INSERT INTO [RMS Data] (XRMS, YRMS, ZRMS, DateTime)
VALUES ((SELECT SQRT((SUM(SQUARE(X))/COUNT(X)))) FROM AccelData WHERE Time = '3/21/2013 1:10:23 PM'),
(SELECT SQRT((SUM(SQUARE(Y))/COUNT(Y)))) FROM AccelData WHERE Time = '3/21/2013 1:10:23 PM'),
(SELECT SQRT((SUM(SQUARE(Z))/COUNT(Z)))) FROM AccelData WHERE Time = '3/21/2013 1:10:23 PM'), ('3/21/2013 1:10:23 PM'))
```

---
APPENDIX B

MMA8452Q Breakout Board Schematic
Datasheets

ATmega328

**Features**
- High Performance, Low Power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions - Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of in-System Self-Programmable Flash program memory (ATmega48PA/88PA/168PA/328P)
  - 55/68/128/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  - 612/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C
- Optional Boot Code Section with Independent Lock Bits
- In-System Programming by On-chip Boot Program
- True Read-While-Write Operation
- Programming Lock for Software Security

**Peripheral Features**
- Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Real Time Counter with Separate Oscillator
- Six PWM Channels
- Eight-channel 10-bit ADC in TQFP and QFN/MLF package
  - Temperature Measurement
  - Eight-channel 10-bit ADC in PDIP Package
  - Temperature Measurement
- Programmable Serial USART
- Master/Slave SPI Serial Interface
- Byte-oriented 2-wire Serial Interface (Philips I²C compatible)
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Interrupt and Wake-up on Pin Change

**Special Microcontroller Features**
- Power-on Reset and Programmable Brown-out Detection
- Internal Calibrated Oscillator
- External and Internal Interrupt Sources
- Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby

**I/O and Packages**
- 23 Programmable I/O Lines
- 28-pin PDIP, 32-lead TQFP, 28-pad G/FN/MLF and 32-pad G/FN/MLF

**Operating Voltage:**
- 1.8 - 6.5V for ATmega48PA/88PA/168PA/328P

**Temperature Range:**
- 0°C to 70°C

**Speed Grade:**
- 0 - 20 MHz @ 1.8 - 6.5V

**Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:**
- Active Mode: 0.2 mA
- Power-down Mode: 0.1 μA
- Power-save Mode: 0.76 μA (including 32 kHz RTC)
1.1 Pin Descriptions

1.1.1 VCC
Digital supply voltage.

1.1.2 GND
Ground.

1.1.3 Port B (PB7:0) XTL1/XTAL2/TO5C1/TO5C2
Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the Inverting Oscillator amplifier and Input to the Internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the Inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TO5C2..1

Input for the Asynchronous Timer/Counter2 If the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in "Alternate Functions of Port B" on page 82 and "System Clock and Clock Options" on page 26.

1.1.4 Port C (PC5:0)
Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.1.5 PC6/RESET
If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset Input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in Table 25-3 on page 318. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in "Alternate Functions of Port C" on page 85.

1.1.6 Port D (PD7:0)
Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.
ATmega48PA/88PA/168PA/328P

The various special features of Port D are elaborated in "Alternate Functions of Port D" on page 68.

1.1.7 AV\textsubscript{CC}

AV\textsubscript{CC} is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to \textsubscript{VCC} even if the ADC is not used. If the ADC is used, it should be connected to \textsubscript{VCC} through a low-pass filter. Note that PC6..4 use digital supply voltage, \textsubscript{VCC}.

1.1.8 AREF

AREF is the analog reference pin for the A/D Converter.

1.1.9 ADC7:6 (TQFP and QFN/MLF Package Only)

In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.
2. Overview

The ATmega48PA/88PA/168PA/328P is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega48PA/88PA/168PA/328P achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram
Portable Data Acquisition Device

Freescale Semiconductor
Data Sheet: Technical Data
An Energy Efficient Solution by Freescale

MMA8452Q

3-Axis, 12-bit/8-bit Digital Accelerometer

The MMA8452Q is a small low-power, three-axis, capacitive micromachined accelerometer with 12 bits of resolution. This accelerometer is packed with embedded functions with flexible user programmable options, configurable to two interrupt pins. Embedded interrupt functions allow for overall power savings relieving the host processor from continuously polling data.

The MMA8452Q has user selectable full scales of ±2g/±4g/±8g with high pass filtered data as well as non filtered data available real-time. The device can be configured to generate inertial wakeup interrupt signals from any combination of the configurable embedded functions allowing the MMA8452Q to monitor events and remain in a low power mode during periods of inactivity. The MMA8452Q is available in a 3 mm × 3 mm × 1 mm QFN package.

Features
- 1.95 V to 3.6 V supply voltage
- 1.6 V to 3.6 V interface voltage
- ±2g/±4g/±8g dynamically selectable full-scale
- Output Data Rates (ODR) from 1.56 Hz to 800 Hz
- 99 µg/Hz noise
- 12-bit and 8-bit digital output
- I²C digital output interface (operates to 2.25 MHz with 4.7 kΩ pullup)
- Two programmable interrupt pins for six interrupt sources
- Three embedded channels of motion detection
  - Freefall or Motion Detection: 1 channel
  - Pulse Detection: 1 channel
  - Jolt Detection: 1 channel
- Orientation (Portrait/Landscape) detection with set hysteresis
- Automatic ODR change for Auto-WAKE and return to SLEEP
- High Pass Filter Data available real-time
- Self-Test
- RoHS compliant
- Current Consumption: 6 µA – 165 µA

Typical Applications
- eCompass applications
- Static orientation detection (Portrait/Landscape, Up/Down, Left/Right, Back/Front position identification)
- Notebook, eReader and Laptop Tumble and Freefall Detection
- Real-time orientation detection (virtual reality and gaming 3D user position feedback)
- Real-time activity analysis (pedometer step counting, freefall drop detection for HDD, dead-reckoning GPS backup)
- Motion detection for portable product power saving (Auto-SLEEP and Auto-WAKE for cell phone, PDA, GPS, gaming)
- Shock and vibration monitoring (mechatronic compensation, shipping and warranty usage logging)
- User interface (menu scrolling by orientation change, tap detection for button replacement)

ORDERING INFORMATION

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Temperature Range</th>
<th>Package Description</th>
<th>Shipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMA8452Q8T</td>
<td>-40°C to +85°C</td>
<td>QFN-16</td>
<td>Tray</td>
</tr>
<tr>
<td>MMA8452QR1</td>
<td>-40°C to +85°C</td>
<td>QFN-16</td>
<td>Tape and Reel</td>
</tr>
</tbody>
</table>

© Freescale Semiconductor, Inc., 2010, 2011. All rights reserved.
W5100 Datasheet

The W5100 is a full-featured, single-chip Internet-enabled 10/100 Ethernet controller designed for embedded applications where ease of integration, stability, performance, area and system cost control are required. The W5100 has been designed to facilitate easy implementation of Internet connectivity without OS. The W5100 is IEEE 802.3 10BASE-T and 802.3u 100BASE-TX compliant.

The W5100 includes fully hardwired, market-proven TCP/IP stack and integrated Ethernet MAC & PHY. Hardwired TCP/IP stack supports TCP, UDP, IPv4, ICMP, ARP, IGMP and PPPoE which has been proven in various applications for several years. 16Kbytes internal buffer is included for data transmission. No need of consideration for handling Ethernet controller, but simple socket programming is required.

For easy integration, three different interfaces like memory access way, called direct, indirect bus and SPI, are supported on the MCU side.

Target Applications

The W5100 is well suited for many embedded applications, including:

- Home Network Devices: Set-Top Boxes, PVRs, Digital Media Adapters
- Serial-to-Ethernet: Access Controls, LED displays, Wireless AP relays, etc.
- Parallel-to-Ethernet: POS / Mini Printers, Copiers
- USB-to-Ethernet: Storage Devices, Network Printers
- GPIO-to-Ethernet: Home Network Sensors
- Security Systems: DVRs, Network Cameras, Kiosks
- Factory and Building Automations
- Medical Monitoring Equipments
- Embedded Servers
Features

- Support Hardwired TCP/IP Protocols: TCP, UDP, ICMP, IPv4 ARP, IGMP, PPPoE, Ethernet
- 10BaseT/100BaseTX Ethernet PHY embedded
- Support Auto Negotiation (Full-duplex and half duplex)
- Support Auto MDI/MDIX
- Support ADSL connection (with support PPPoE Protocol with PAP/CHAP Authentication mode)
- Supports 4 Independent sockets simultaneously
- Not support IP Fragmentation
- Internal 16Kbytes Memory for Tx/Rx Buffers
- 0.18 μm CMOS technology
- 3.3V operation with 5V I/O signal tolerance
- Small 80 Pin LQFP Package
- Lead-Free Package
- Support Serial Peripheral Interface (SPI MODE 0, 3)
- Multi-function LED outputs (TX, RX, Full/Half duplex, Collision, Link, Speed)
Block Diagram