Summer 8-31-2011

Development of a vibration-powered impact recorder

William Contreras
New Jersey Institute of Technology

ABSTRACT

DEVELOPMENT OF A VIBRATION-POWERED IMPACT RECORDER

by
William Contreras

Currently, the U.S. Army stores a great deal of equipment for long periods of time. Often, this equipment is subjected to damaging vibrations. Given this, the army wants to be able to monitor the vibrations that are undergone by this equipment. Here, a battery-powered monitoring device would be undesirable because its batteries would need to be replaced. To solve this problem, an energy harvesting, vibration monitoring device has been developed. The device, which is known as a vibration-powered impact recorder (VPIR for short), uses a piezoelectric transducer to power a microcontroller, which uses the power to count the number of times a certain vibration threshold has been passed.

The VPIR device operates in the following way: When the device is vibrated, the piezoelectric transducer produces a voltage. If the vibration is strong enough, the voltage generated by the transducer becomes great enough to turn the microcontroller on. When turned on, the microcontroller adds ‘one’ to the value stored in a particular EEPROM register. A LabVIEW program running on a PC is used to read the value, or ‘count,’ held in the EEPROM register, and if necessary set the count value in the microcontroller to zero. The PC and microcontroller communicate via USB.

At this time, the device has been successfully built and tested on both a breadboard and a specially designed printed circuit board.

DEVELOPMENT OF A VIBRATION-POWERED IMPACT RECORDER

by
William Contreras

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

August 2011

APPROVAL PAGE

DEVELOPMENT OF A VIBRATION-POWERED IMPACT RECORDER

William Contreras

Dr. Gordon Thomas, Thesis Advisor Professor of Physics, Department of Physics, NJIT	Date
Dr. John Federici, Committee Member Professor of Physics, Department of Physics, NJIT	Date
Dr. Reginald Farrow, Committee Member Research Professor, Department of Physics, NJIT	Date

BIOGRAPHICAL SKETCH

Author:	William Contreras
Degree:	Master of Science
Date:	August 2011

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2011

- Bachelor of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2009

Major: Electrical Engineering

To My Parents, For Supporting Me In
My Academic Pursuits

ACKNOWLEDGEMENT

I would like to thank my thesis advisor and thesis committee for imparting a great deal of valuable knowledge to me. They have helped me develop an understanding as to what science and engineering are all about. I would like to thank them in particular for helping make my Master's thesis possible.

In addition to my advisor and committee members, I would like to thank the U.S. Army for providing the funding that helped make my Master's thesis possible.

Lastly, though certainly not least, I would like to thank my parents for supporting me in my academic pursuits.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of this thesis is to discuss the development of the vibration-powered impact recorder (VPIR for short). In the thesis, all aspects of the development of the VPIR system are discussed, beginning with the selection and optimization of the transducer used to convert vibrational energy into electrical energy. An overview of the operation of the VPIR system is provided, and the hardware and software used in the system are explained in detail. In addition, the printed circuit board designed for the VPIR system is examined. The major problems encountered in the debugging of the VPIR system, and the results observed in the final testing of the system are discussed.

1.2 Background Information

Currently, the U.S. Army stores a great deal of equipment for long periods of time. Often, this equipment is subjected to damaging vibrations. Given this, the army wants to be able to monitor the vibrations that are undergone by this equipment. Here, a battery-powered monitoring device would be undesirable because its batteries would need to be replaced. The VPIR device has been developed to provide a solution to this problem.

Essentially, the VPIR device harvests vibrational energy to monitor vibrations. In particular, it counts the number of times that it passes through a specific vibrational threshold. Although such information provides only a relatively low-resolution picture of the vibrations undergone by the device, it can be used to great avail by the Army to help determine the reliability of a piece of equipment.

CHAPTER 2

DESIGN OF THE VIBRATION-POWERED IMPACT RECORDER

2.1 THE TRANSDUCER

To harvest energy from vibrations, it was decided that a piezoelectric transducer should be used. A variety of piezoelectric sensors were tested to determine which of the sensors was the most sensitive. One sensor tested was the Phidgets 1104 vibration sensor. This sensor was found to have the greatest sensitivity of all of the sensors tested. As such, it was decided that the Phidgets sensor should be used for the VPIR project.

2.1.1 Sensor Construction

The Phidgets 1104 sensor consists of a circular mass of piezoelectric material sandwiched between an electric insulator and a metal back-plate. The metal back-plate provides an electrical connection to the bottom side of the piezoelectric material. The other electrical connection is provided by a small hole in the insulator leading to the top side of the piezoelectric material. The sensor can be seen in Figure 2.1.

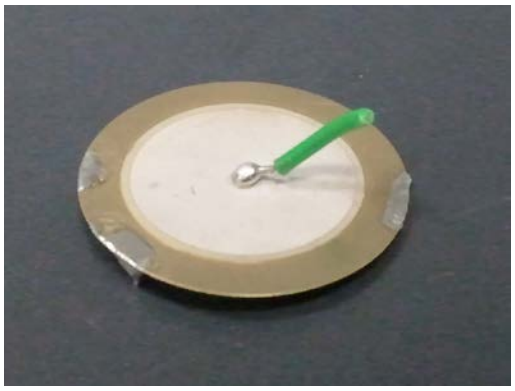

Figure 2.1 Image of a Phidgets 1104 piezoelectric sensor.

2.1.2 Sensor Mounting: Experimentation

Once it was decided that the Phidgets sensor should be used, the best way to mount the sensor on the VPIR device was sought after. In specific, it was desired to mount the Phidgets sensor such to get the greatest voltage response out of it. Toward this end, several different mounting schemes were tested. Finite element analysis, which is to be discussed later, was also performed on a couple of the mounting schemes.

In particular, four mounting schemes were tested. The names of the mounting schemes that were tested are as follows: ‘tall-cantilever mount’, ‘short-cantilever mount’, ‘fixed-middle mount’, and ‘two-fixed-ends mount’. Illustrations of the four mounting schemes can be seen in Figure 2.2. (The short-cantilever mounting scheme was tested with an inertial mass on the edge of the sensor, as seen in Figure 2.2)

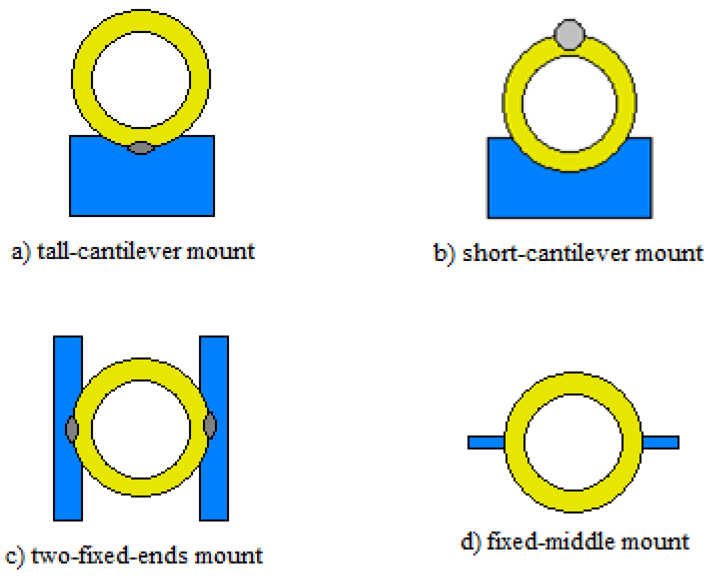

Figure 2.2 Illustrations of the different mounting schemes that were tested.

For testing, in each case, the substrate (the blue parts shown in Figure 2.2) of the mount was fixed to a subwoofer diaphragm. In particular, each sensor was mounted such that the speaker diaphragm would move in a direction normal to the plane formed by the sensor's back-plate. Once mounted, each sensor was shaken in a sinusoidal fashion at a particular frequency for several different amplitudes. For each amplitude trial, the diaphragm's maximum acceleration was calculated.

In the experiments, the speaker diaphragm moved in a straight line, which will be referred to as the 'x' direction. The diaphragm's position as a function of time is indicated by Eq. (2.1). The acceleration of the diaphragm is the second derivative of Eq. (2.1), which is shown in Eq. (2.2). The maximum value of Eq. (2.2) is Eq. (2.3). Eq. (2.3) was used to calculate the diaphragm's maximum acceleration.

$$x = A\sin(\omega t) \qquad \textbf{(2.1)}$$

$$a = -A\omega^2\sin(\omega t) \qquad \textbf{(2.2)}$$

$$a_{max} = A\omega^2 \qquad \textbf{(2.3)}$$

For each amplitude trial, the peak value of the sensor's voltage response was also recorded. By associating the peak output voltage with the peak acceleration for each trial, a voltage/acceleration relationship could be obtained for the mounting configuration being tested.

A function generator was used to supply the subwoofer with a sinusoidal signal in order to get the subwoofer's diaphragm to oscillate in a sinusoidal fashion. The frequency of the function generator’s signal, which translates to the frequency of the diaphragm’s movement, was measured with a multimeter.

Along with the sensor mount a small mirror was mounted to the speaker diaphragm. In specific, the mirror was mounted such that its reflective surface was perpendicular to the movement of the speaker diaphragm. Laser light was bounced off the mirror and onto a flat surface, which was mounted as seen in Figure 2.3.

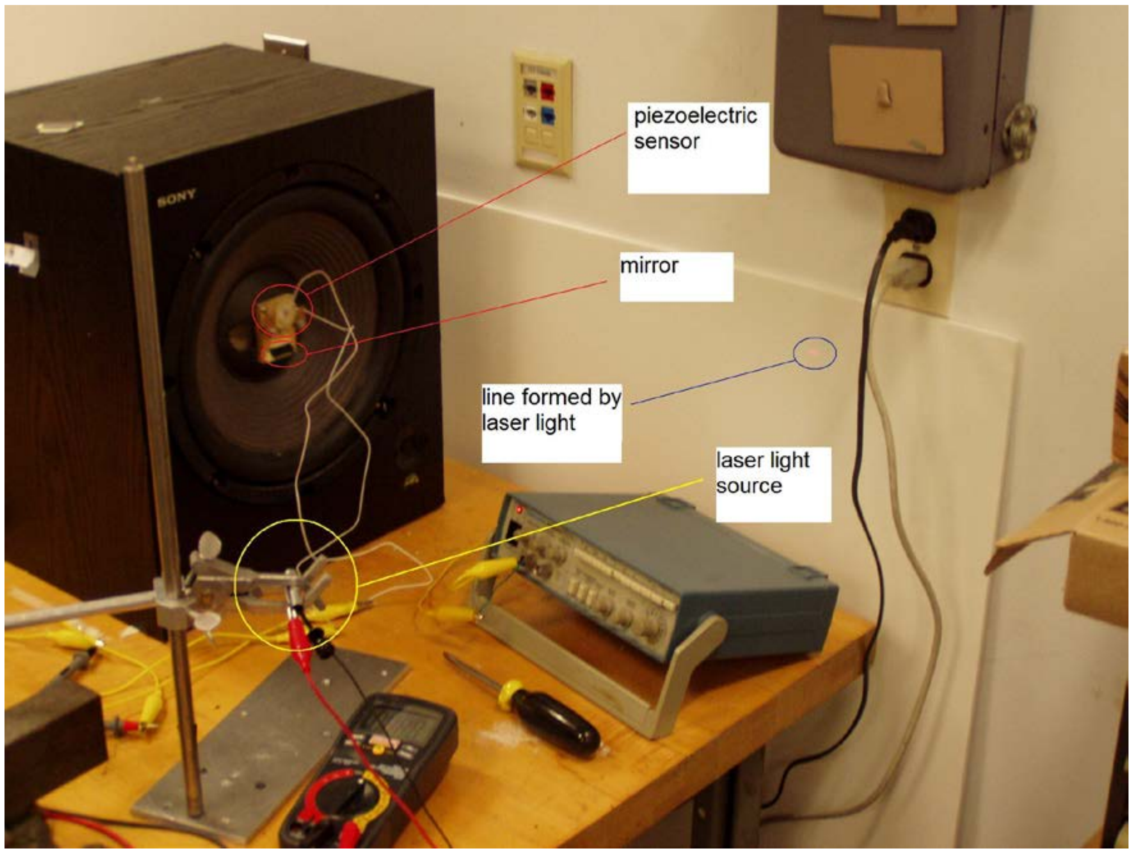

Figure 2.3 The experimental setup used to obtain voltage/acceleration data for the different mounting configurations.

As the speaker diaphragm moved back and forth during an amplitude trial, the laser light formed a line on the flat surface. The length of the line formed on the flat surface was twice the length of the diaphragm’s peak-to-peak movement. Given this relationship, the amplitude of the diaphragm’s movement, and thus the peak acceleration of the diaphragm’s movement, was determined.

The voltage/acceleration relationships that were obtained for the four different mounting schemes can be seen in Figures 2.4 - 2.7.

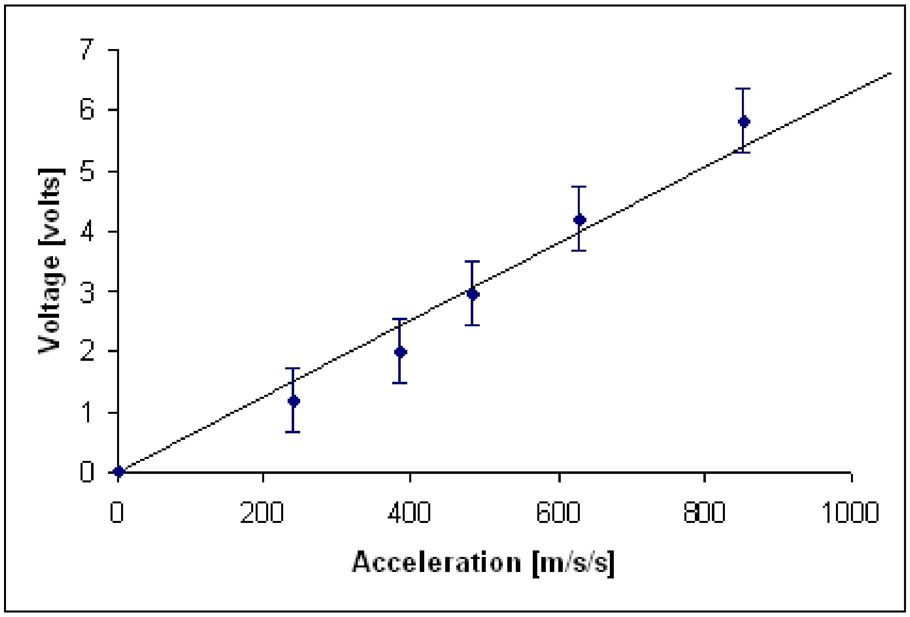

Figure 2.4 Voltage/acceleration data for the tall-cantilever mount.

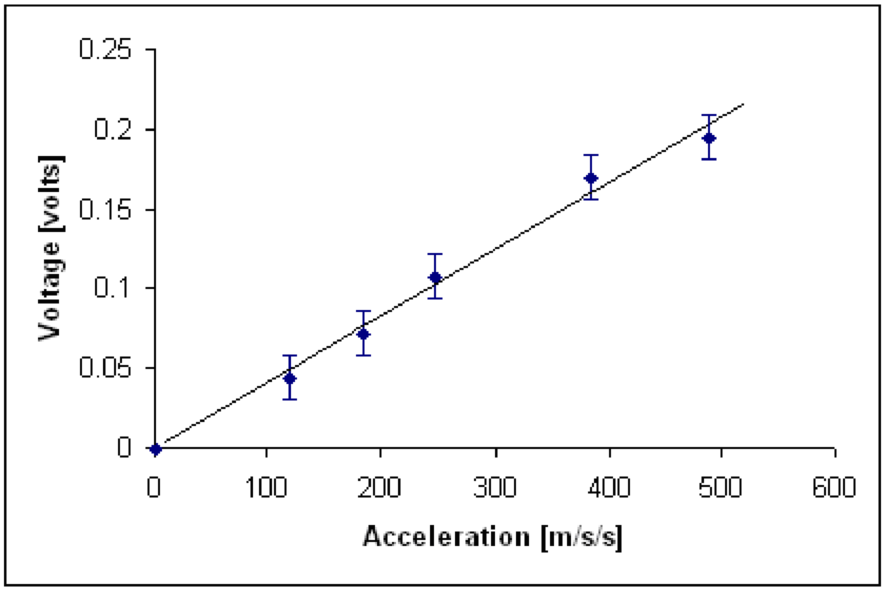

Figure 2.5 Voltage/acceleration data for the fixed-middle mount.

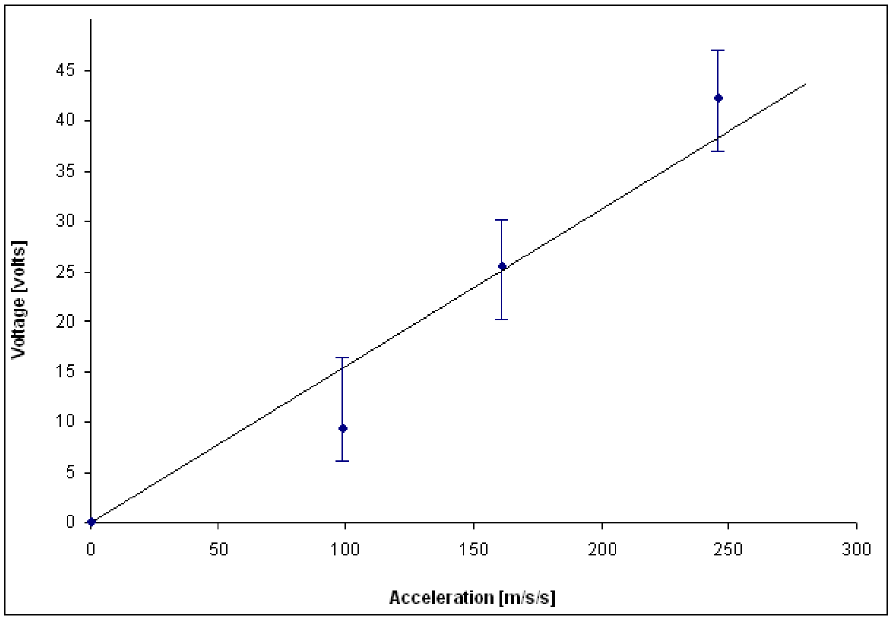

Figure 2.6 Voltage/acceleration data for the short-cantilever mount.

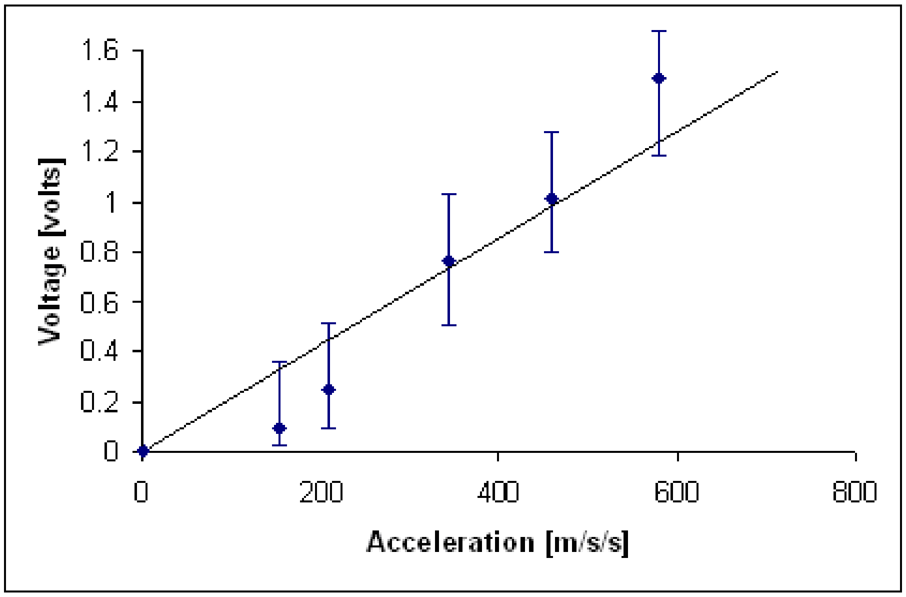

Figure 2.7 Voltage/acceleration data for the two-fixed-ends mount.

From the voltage/acceleration plots it can be seen that the short-cantilever mounting configuration (which, again, was tested with inertial mass) was the most sensitive of the four configurations tested.

In addition to being the most sensitive configuration, the short-cantilever mount was found to be more rugged than the tall-cantilever mount. Basically, in the case of the tall-cantilever mount, the back-plate was bending more than it was in the case of the short-cantilever mount. This was causing the back-plate material of the tall-cantilever configuration to fail more rapidly than that of the short-cantilever configuration.

2.1.3 Sensor Mounting: Finite Element Analysis

In addition to the experiments regarding the different mounting configurations, finite element analysis was performed on the short-cantilever and tall-cantilever configurations. This was done in an attempt to understand why these two configurations were performing the way they were.

In each of the finite element simulations, the mounting configuration being simulated was accelerated in a sinusoidal fashion, just as in the sensitivity experiments. The linear acceleration for various points on the mount at a particular instant in time was calculated, and the resulting values were plotted along the mount and sensor as color. The results of the simulations can be seen in Figures 2.8 and 2.9.

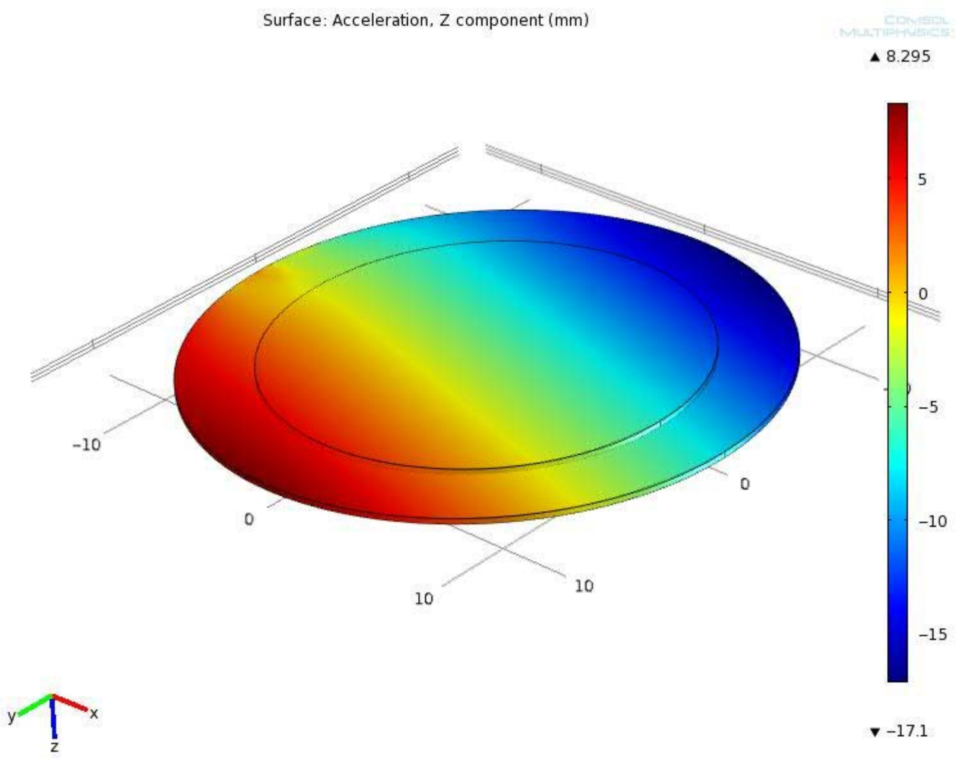

Figure 2.8 Results of finite element analysis on the tall-cantilever mount.

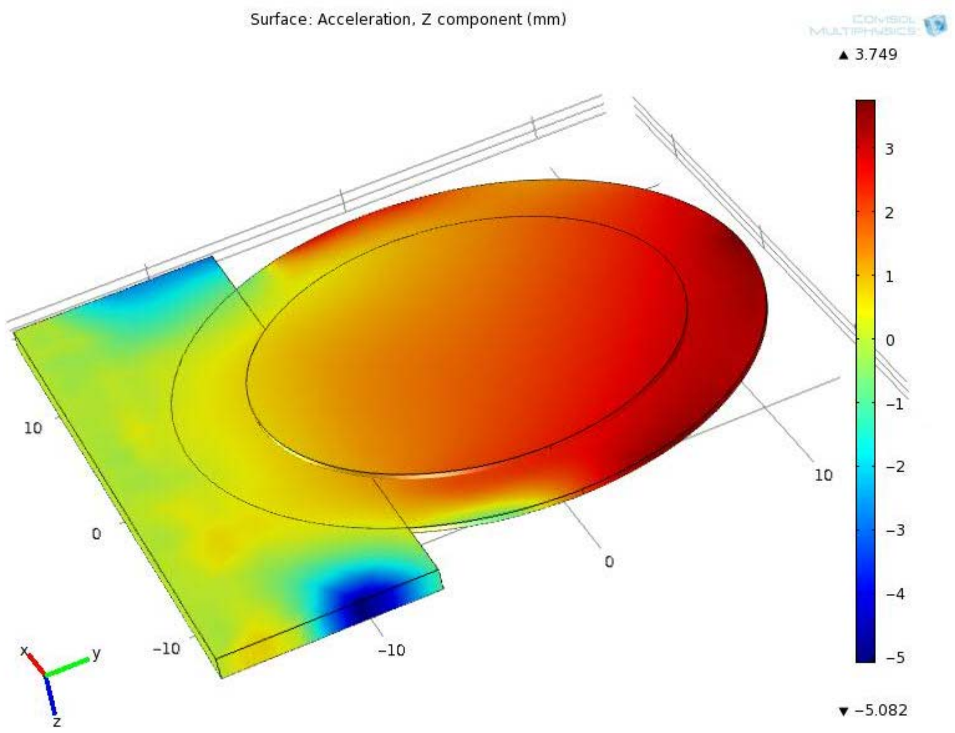

Figure 2.9 Results of finite element analysis on the short-cantilever mount.

In Figure 2.8, which concerns the tall- cantilever mount, it can be seen that the disc is accelerating positively in the area just above the mounting axis. It can also be seen that the top part of the disc is accelerating negatively. Because the output voltage of the sensor is related to the integral of the acceleration over the disc, the positive and negative accelerations negate, to some extent, each other's effect on the output voltage.

In Figure 2.9, which concerns the short-cantilever mount, it can be seen that the direction of the acceleration is uniform across the disc. As such, there is no cancellation of the type seen with the tall-cantilever mount. This is why the output voltage of the short-cantilever mount is greater than the output of the tall-cantilever mount.

Given the findings overall, it was decided that the short-cantilever mount, with inertial mass, should be used in the VPIR device.

2.1.4 Inertial Mass

In an attempt to improve the sensitivity of the sensor, the effect of inertial mass on the sensor’s output voltage was studied. Here, it was hypothesized that inertial mass would help the piezoelectric material on the sensor to bend more and thus produce a greater output voltage.

To test the hypothesis, a setup similar to that which was used to test the different mounting configurations was used. In particular, a short-cantilever device was fixed to the diaphragm of a speaker. The sensor was then oscillated in a sinusoidal fashion, at a constant frequency and amplitude. For that particular excitation the amount of mass attached to the tip of the cantilever was varied and the sensor’s peak output voltage was recorded. As before, the sensor’s peak output voltage corresponded to the peak acceleration of the sensor. As such, a relationship between output voltage and mass for a particular acceleration was obtained. The data from the experiment can be seen in Figure 2.10.

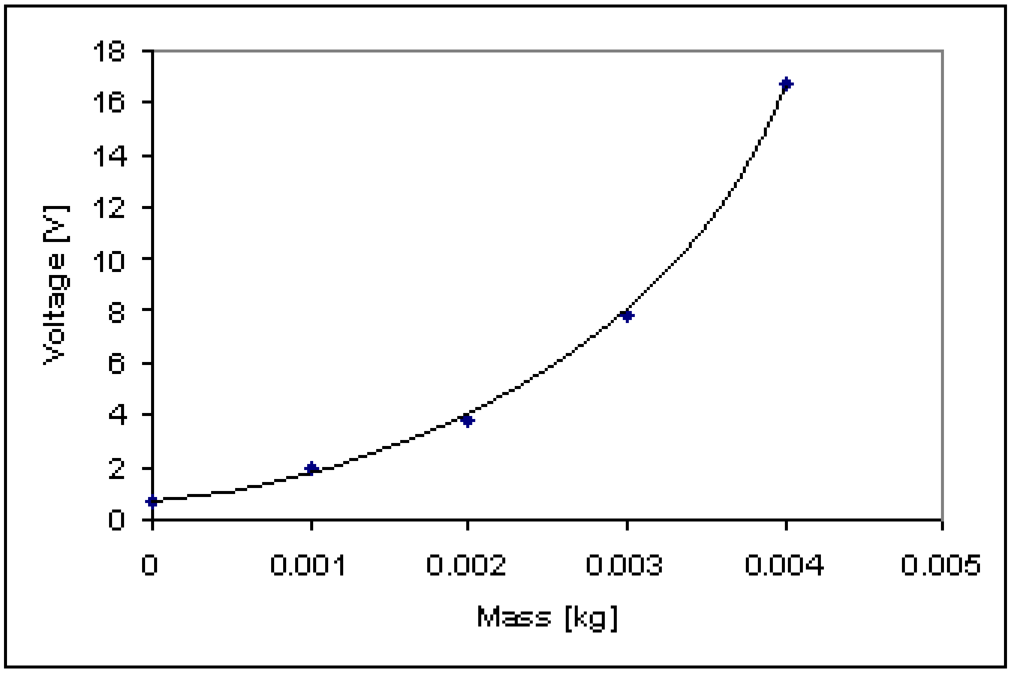

Figure 2.10 Plot of voltage/mass data.

Basically, the hypothesis proved to be correct: adding inertial mass greatly increases the sensor's output voltage. Given this, it was decided that inertial mass should be incorporated into the design of the device.

2.2 OVERVIEW OF DEVICE OPERATION

The basic purpose of the VPIR device is to count the number of times that it has passed through a vibrational threshold and to, when called upon to do so, display the count information to the user. A diagram of the VPIR device's circuitry can be seen in Figure 2.11.

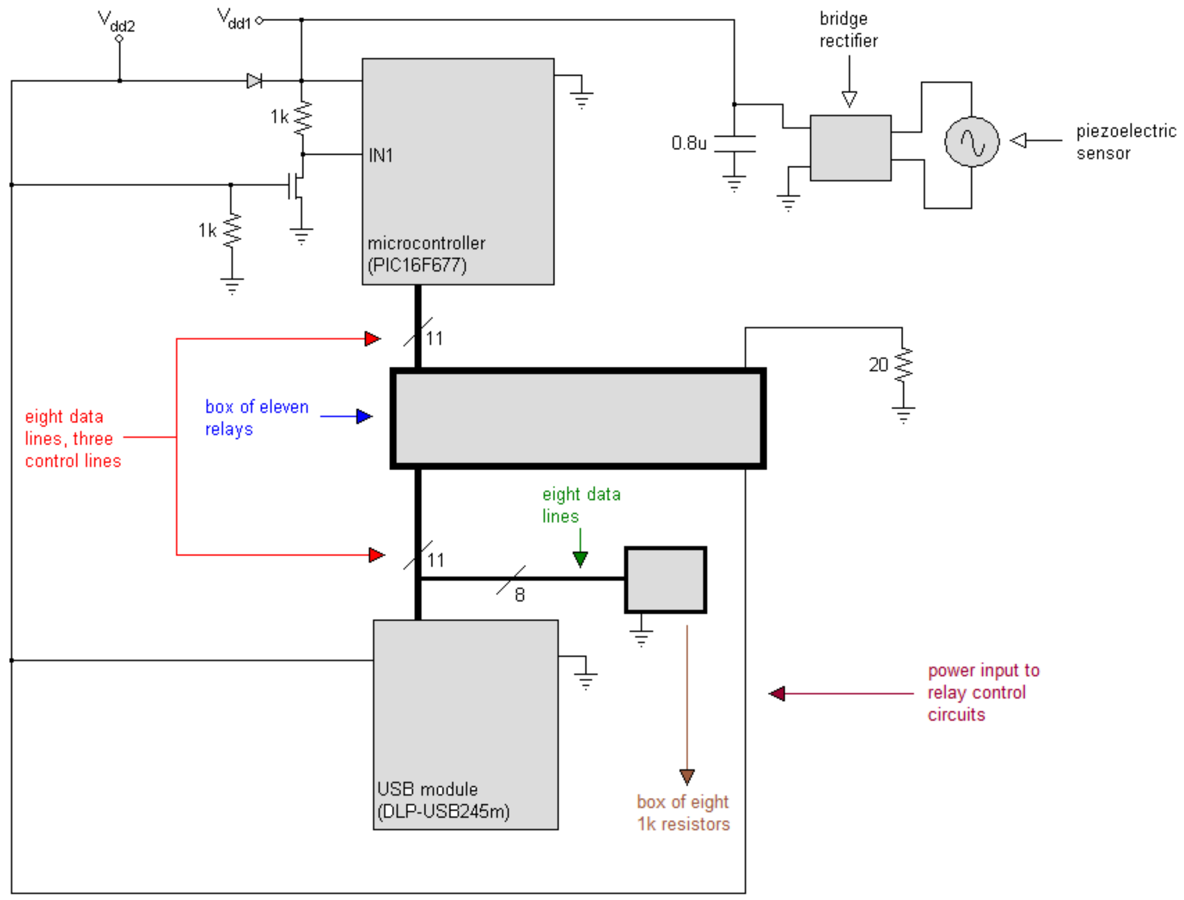
Vdd2
Vdd1
bridge rectifier
piezoelectric sensor
1k
IN1
1k
microcontroller (PIC16F677)
0.8u
11
20
eight data lines, three control lines
box of eleven relays
eight data lines
11
8
power input to relay control circuits
USB module (DLP-USB245m)
box of eight 1k resistors

Figure 2.11 Circuit diagram for VPIR device.

When the VPIR device vibrates the piezoelectric sensor produces an AC voltage. A bridge rectifier converts the AC voltage into a DC voltage. A capacitor then smooths the voltage being output from the bridge rectifier. The voltage on the capacitor is then used to power a microcontroller. As the vibrations become larger, the voltage being output from the piezoelectric sensor increases; concomitantly the voltage on the capacitor and microcontroller power pin increases. When the vibrations, and thus the voltage on the microcontroller power pin become large enough, the microcontroller turns on. When the microcontroller turns on it increments a EEPROM memory register, and thus counts the threshold crossing.

It is important to note here that the microcontroller program has two main parts: a ‘write’ part and a ‘read’ part. In order to increment the register the microcontroller must go to the ‘write’ part of its program. This manner in which this is accomplished is described in the following paragraph:

When the microcontroller is powered by vibrations, the voltage across the capacitor cannot get through the diode to the gate of the MOSFET. As such, the MOSFET gate is at zero volts. As a result of this, the impedance between the drain and the source of the MOSFET is high. Looking at the drain-to-source path of the MOSFET and the one kilo-ohm resistor connected to it as a voltage divider, it can be seen that when the drain-to-source impedance of the MOSFET is high, very little voltage drops across the one kilo-ohm resistor. As such, in this case the drain of the MOSFET is approximately equal to the voltage V_{dd1}. Likewise the microcontroller pin ‘IN1’ is approximately equal to the voltage V_{dd1}. The presence of the V_{dd1} voltage on the ‘IN1’ pin indicates to the microcontroller that it should go to the ‘write’ part of its program.

In order to read the threshold count from the device the user must first power the VPIR device up by plugging a USB cable into the device. When the USB cable is plugged in the gate of the MOSFET goes to five volts. Because of this, the impedance between the drain and the source of the MOSFET becomes very small. Again looking at the drain-to-source path and the one kilo-ohm resistor as a voltage divider, it can be seen that, in this case, very little voltage gets dropped across the drain-to-source path. As such, the drain of the MOSFET and the microcontroller pin ‘IN1’ are at approximately zero volts. The presence of zero volts on the ‘IN1’ pin indicates to the microcontroller that it should go to the ‘read’ part of its program.

In read mode, the microcontroller sends the threshold count data to the USB module using eight data lines and three control lines. The USB module then sends the data to a PC using the USB bus. On the PC end, a LabVIEW program receives the data, interacts with the user, and helps control the transfer of data to and from the VPIR device.

In the VPIR device a solid state relay had to be placed in each of the lines (except power and ground), connecting the microcontroller to the USB module. This is because it was found that when the device was being powered by vibrations, the data and control lines connecting the microcontroller to the USB module drew current and loaded the piezoelectric source down. The lines loaded the source to such an extent that the source could no longer power the microcontroller. (Here it should be noted that simple diodes could not be used to block the current because the lines are bidirectional: data is sent back and forth between the microcontroller and USB module over a single set of lines. The system whereby the microcontroller and USB module communicate is basically a half-duplex communication system.)

The relay system works as described in the following paragraph:

When the USB cable is plugged in, the relays’ positive control pins, which are connected to V_{dd2} in the circuit diagram, go to five volts. As such, when the system is powered off of the USB bus, the relays close and allow communication between the USB module and microcontroller. When the system is being powered by vibrations, V_{dd2} is at zero volts. This is because the voltage across the capacitor cannot get through the diode. As a result of this when the system is vibration powered the relays are open, which prevents the data and control lines from drawing any power.

2.3 THE USER INTERFACE

The user of the VPIR device must interact with the device to retrieve the threshold-count data and to, when necessary, set the count value in the microcontroller equal to zero. A flow chart concerning the system that is used for interfacing the user to the VPIR device can be seen in Figure 2.12. The system for interfacing works as described in the following paragraph:

The user must first connect the VPIR device to the PC via a USB cable. The user must then open the LabVIEW executable file. Once the file opens, a prompt asks the user whether he or she wants to read the count data from the device or set the count value in the device to zero. If the user selects the initialize possibility a dialog box pops up telling the user that the count value has been cleared. The LabVIEW program then terminates. If the user selects the read possibility, the program reads the count value from the microcontroller and then displays the count value to the user in a dialog box. In the same dialog box, the user is instructed to close the dialog box to continue. Once the dialog box is closed, a prompt appears asking the user whether he or she wants to set the count value in the device equal to zero or keep the current count value in memory. If the user chooses to clear the count value, the LabVIEW program instructs the microcontroller to set the count equal to zero. A dialog box then appears stating that the count has been cleared. After this, the program terminates. If the user chooses to keep the current count value in memory, the LabVIEW program immediately terminates.

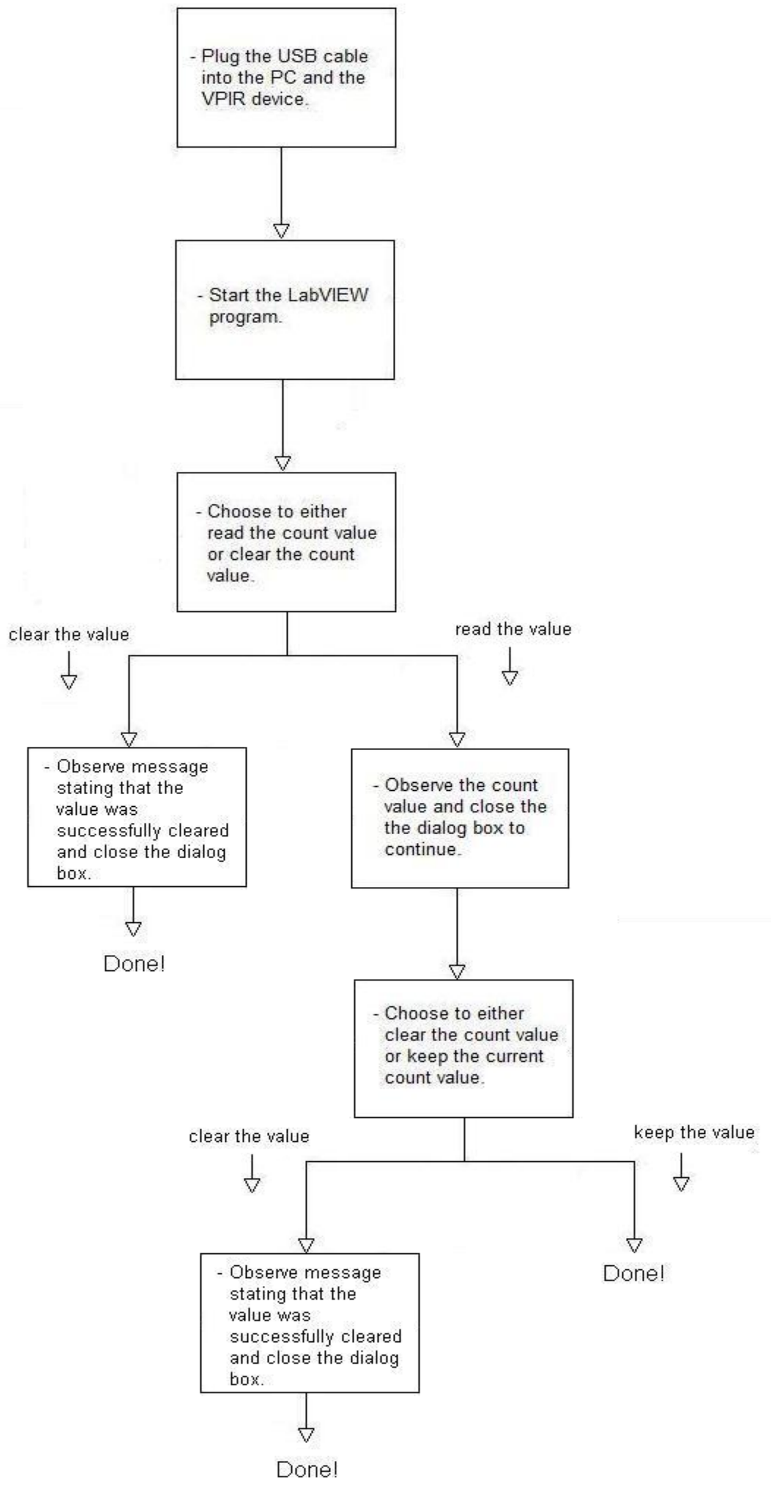
- Plug the USB cable into the PC and the VPIR device.
- Start the LabVIEW program.
- Choose to either read the count value or clear the count value.
clear the value
read the value
- Observe message stating that the value was successfully cleared and close the dialog box.
- Observe the count value and close the the dialog box to continue.
Done!
- Choose to either clear the count value or keep the current count value.
clear the value
keep the value
Done!
- Observe message stating that the value was successfully cleared and close the dialog box.
Done!

Figure 2.12 Flow chart pertaining to the user interface.

2.4 THE LABVIEW SOFTWARE

The LabVIEW program that is being used for the VPIR device can be found in the appendix, part A.1.

The LabVIEW program consists of a sequence of blocks of code. From the figures in the appendix, it can be seen that there are at total of eight blocks of code in the program. The blocks execute in sequence from block one to block eight. In block one, the LabVIEW program sets up the serial communications port. In block two, the program prompts the user as to whether he or she would like to read from or initialize the device. In block three, the program either instructs the microcontroller to send the count data to the PC or to set the count equal to zero, depending on what the user chooses to do in block two. If the program instructs the microcontroller to clear the count, it also displays a message to the user that the count has been set equal to zero.

In all of the blocks following block three, except block eight, the program's actions depend upon whether the user selected the read possibility of the initialize possibility in block two. If the user selected the initialize possibility, the program basically does nothing in blocks four through seven. If the user selected the read possibility, the program executes as described in the following paragraph:

In block four, the program runs a short delay to give the microcontroller time to output the count data. In block five, the program reads the count data sent by the microcontroller and displays it to the user in a dialog box. In block six, the program prompts the user as to whether he would like to keep the count value that is currently in memory. Here, the user can choose to either keep the current value or clear the current value. If the user decides to clear the count value, the LabVIEW program, in block seven, tells the microcontroller to set the count value equal to zero. In the same block, the program then informs the user that the count value has been cleared. If in block six the user decides to keep the current value, the program does nothing in block seven.

In the final block of the program, block eight, the program unconditionally closes the communications port and terminates.

2.5 THE MICROCONTROLLER SOFTWARE

The exact microcontroller program can be found in the appendix, in part A.2.

When the microcontroller first turns on it goes to the beginning of its program and executes the program line by line. In the first part of the program, the microcontroller basically sets itself up. Referring to the line numbers in the appendix, the first part of the microcontroller program goes from line one to line 48. The instructions in this part of the program do such things as set up the input/output pins and set the frequency of the microcontroller’s internal oscillator.

Once the microcontroller finishes setting itself up, it polls an input pin to determine whether it should go to the ‘read’ part of its program or the ‘write’ part of its program. This part of the program consists of lines 49 through 51.

In the write part of the program, the microcontroller reads the EEPROM memory location where the threshold crossing count is stored, increments the retrieved value, and writes the resulting value back to the original EEPROM location. The write part of the program consists of lines 53 through 76.

The read part of the program consists of lines 79 through 132. In this part of the program, the microcontroller first waits in a loop, polling an input pin to determine if there is any data in the USB module (from the PC) waiting to be read. When data becomes available the microcontroller goes to the ‘OBTAINDATA’ subroutine, which consists of lines 157 through 175, where it sets itself up to read data from the USB module and instructs the USB module to send it the data.

Once the microcontroller has the data, it uses the it to determine what it should do next. Here, there are two possibilities: the data either tells the microcontroller to set the count value to zero or to send the current count value to the USB module, which in turn sends the value to the PC.

If the microcontroller is to clear the count, it goes to the ‘INITIALIZE’ section of the program. In this section, the microcontroller writes the value ‘zero’ to the EEPROM memory location where the count value is stored. After clearing the count value, the microcontroller goes to the ‘BACK’ section of the program where it loops until it loses power. The INITIALIZE section of the program consists of lines 134 through 155, and the BACK section of the program consists of lines 189 through 198.

If the microcontroller is to send data to the USB module, it writes the data to the input/output pins of the USB module and tells the module to write the data into its output buffer, from which the module sends the data to the PC. To tell the module to write the data, the microcontroller toggles the ‘write’ pin on the USB module. The part of the program in which the microcontroller toggles the write pin consists of lines 107 to 123.

While toggling the write pin on the USB module, the microcontroller polls one of its input pins to determine if the PC has received the count information and thus sent new data to the USB module. When data becomes available, the microcontroller goes to the OBTAINDATA subroutine, where it sets itself up and instructs the USB module to send the data.

Once the microcontroller has the data, it uses it to determine whether the current count value should be saved or cleared.

If the count value is to be cleared, the microcontroller goes to the INITIALIZE part of the program and writes a zero to the EEPROM register holding the count value. After initializing, the microcontroller goes to the BACK section of the program where it loops until it loses power.

If the microcontroller is to hold the current count value in memory, it goes directly to the BACK section without initializing first.

2.6 THE PCB LAYOUT

After testing and debugging the VPIR circuitry on a breadboard, a custom printed circuit board was designed to hold the VPIR circuitry. This was done by using Express PCB’s proprietary layout software. ExpressPCB was used to manufacture the boards.

For the custom PCB, nearly all of the components used were surface mount. The only component that was through-hole was the USB module. To solder the surface mount components, a solder reflow process was used. Here, the Aoyue HHL3000 programmable solder reflow oven was used.

The custom printed circuit boards that were used were four-layer boards. In particular, the boards had a top layer, a bottom layer, an internal ground plane, and an internal power plane.

The layout of the VPIR circuitry on the custom printed circuit boards can be seen in Figure 2.13.

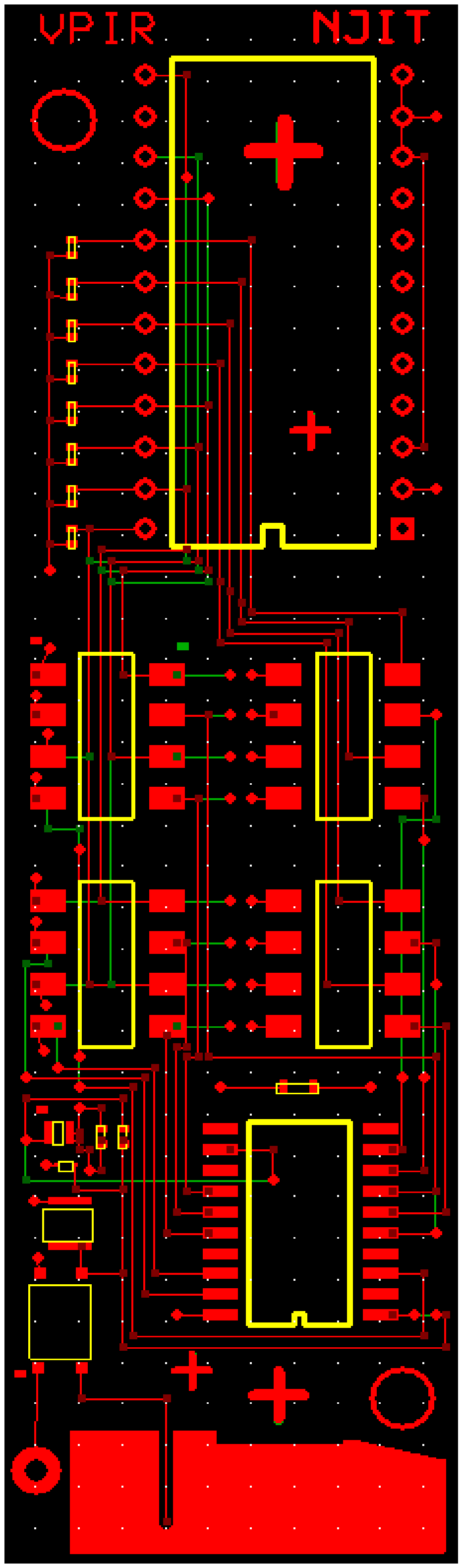

Figure 2.13 The layout for the custom printed circuit boards.

CHAPTER 3

TESTING AND DEBUGGING / RESULTS

A substantial amount of testing and debugging was necessary to get the VPIR device to operate as desired. The VPIR circuitry in particular required a great deal of debugging. The VPIR software, on the other hand, required very little debugging.

The present VPIR circuitry is shown in Figure 2.11; it is quite different from the original circuitry, which can be seen in Figure 3.1.

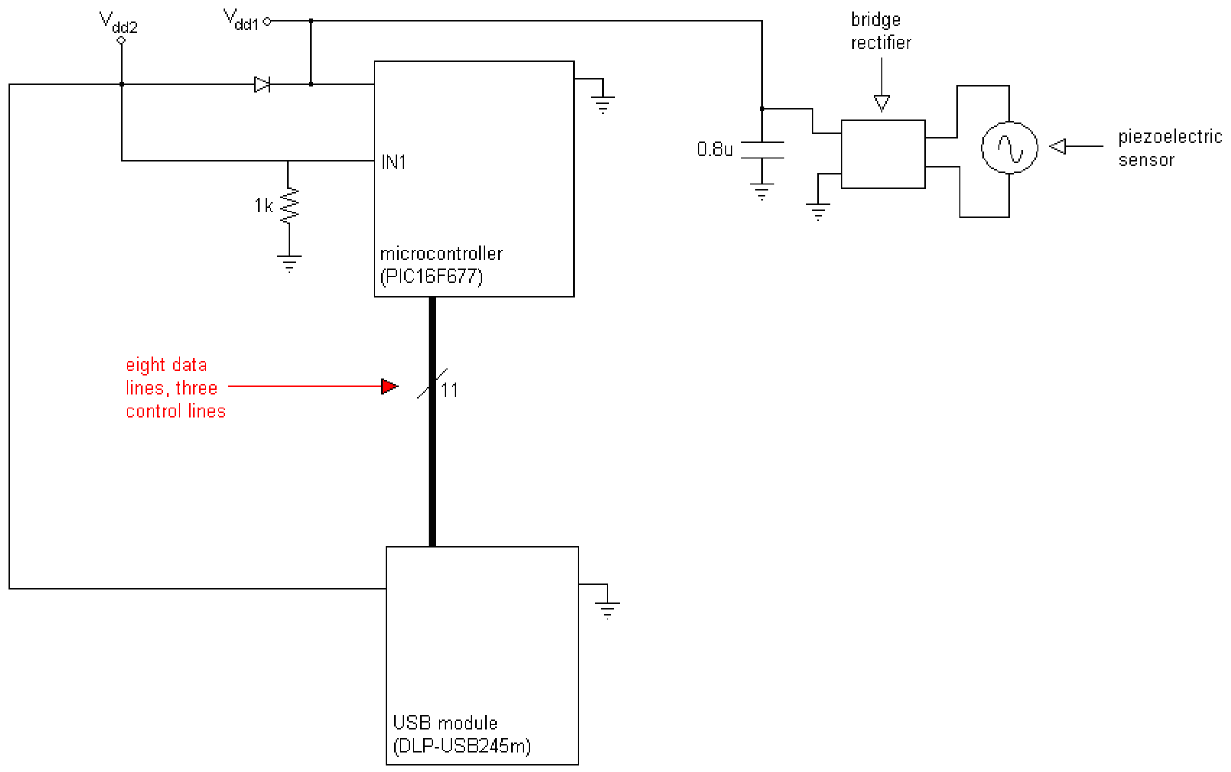

Figure 3.1 The initial design for the VPIR circuitry.

Prior to designing the custom printed circuit board for the VPIR device the circuit seen in Figure 3.1 was built and tested on a breadboard. In testing, the first problem that was encountered involved the bidirectional data pins of the USB module.

The data pins on the USB module can be configured as inputs or outputs. To write data to the module, the pins must be configured as inputs. To read data from the module, the pins should be configured as outputs. During testing, it was discovered that the data pins could not be pulled down to zero volts when configured as inputs. That is, even when the USB module's data pins were configured as inputs, the module output a weak five volts on them. As such, the microcontroller was unable to set the pins to zero volts when necessary. Essentially, the microcontroller was unable to transmit a digital zero to the USB module. To fix this, a one kilo-ohm resistor was connected between each of the data pins and ground. This dramatically reduced the voltage being asserted by the USB module, and allowed the microcontroller to assert and clear the data lines, and thus transmit digital ones and zeros as necessary.

The next problem that was discovered involved the microcontroller's 'IN1' input pin.

For vibration power, input pin IN1 was supposed to be at zero volts (this is because the voltage across the capacitor, as seen in Figure 2.11, cannot get through the diode to the IN1 pin of the microcontroller). The zero volts on the pin was then supposed to indicate to the microcontroller that the microcontroller should go to the 'write' part of its program.

For USB power, pin IN1 was supposed to be five volts (because it is connected directly to the five volt output on the USB module). The five volts was then supposed to indicate to the microcontroller that the microcontroller should go to the ‘read’ part of its program.

During testing it was found that the IN1 pin did indeed go to zero volts for vibration power and five volts for USB power. However, for vibration power, the microcontroller did not go to the ‘write’ part of its program. Instead, it went to the ‘read’ part of the program. When plugged directly into the circuit’s ground, however, the microcontroller correctly went to the ‘write’ part of the program. In essence, it seemed to be that when the input was connected as seen in Figure 3.1, the microcontroller was not seeing zero volts on the pin even though the pin, externally, was in fact at zero volts.

To fix the problem, a circuit was devised to pull the IN1 pin up to V_{dd1} for vibration power and down to zero volts for USB power. Basically, as seen in Figure 2.11, a MOSFET was used for this purpose.

For USB power, the gate of the MOSFET is at five volts and the drain-to-source impedance is low. As a result, pin one goes to zero volts. For vibration power, the gate of the MOSFET is at zero volts and the drain-to-source impedance is high. As a result, the IN1 pin gets pulled up to V_{dd1}.

The MOSFET based circuit worked. The microcontroller went to the ‘read’ part of its program for USB power and the ‘write’ part of its program for vibration power.

The final problem that was encountered involved the control and data lines connecting the USB module and microcontroller. It was found that for vibration power, even though the USB module’s power pin was at zero volts, the module’s control and data pins drew current from the microcontroller’s I/O pins. As a result of this added loading on the piezoelectric sensor, the sensor was not capable of powering the microcontroller’s process of incrementing a EEPROM register. As such, the microcontroller was unable to count threshold crossings.

To fix the problem, a relay was placed in each of the lines connecting the microcontroller to the USB module (except for the power and ground lines). Since vibrations can interfere with the operation of mechanical relays solid state relays were used. The system of relays works as described in the following paragraph:

Referring to Figure 2.11, when the device is being powered by vibrations the voltage produced by the piezoelectric sensor is not capable of getting through the diode. As such the control pins on the relays are at zero volts. Given this, and that the relays are of the ‘normally open’ type, the relays are open for vibration power. As such, for vibration power, the USB module’s data and control pins are effectively disconnected from the microcontroller, which prevents them from drawing current and loading down the sensor. For USB power, the positive control pins on the relays go to five volts. As such, for USB power, the relays are closed and allow the microcontroller and USB module to communicate with each other.

CHAPTER 4

CONCLUSION

The goal originally set forth has been wholly accomplished: a vibration-powered device capable of counting the number of times that a vibrational threshold has been crossed, and small enough to fit inside a 30mm round has been successfully built. The device is shown in Figure 4.1; it is also shown in Figure 4.2.

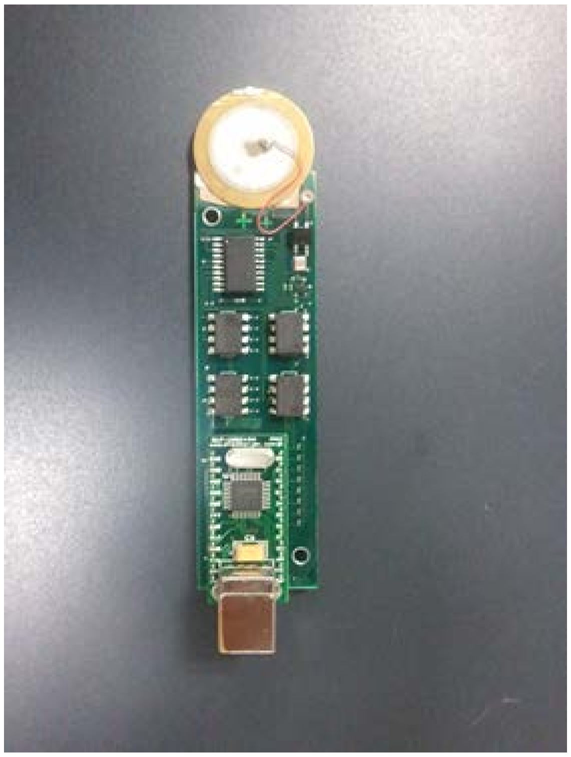

Figure 4.1 Top view of the VPIR device.

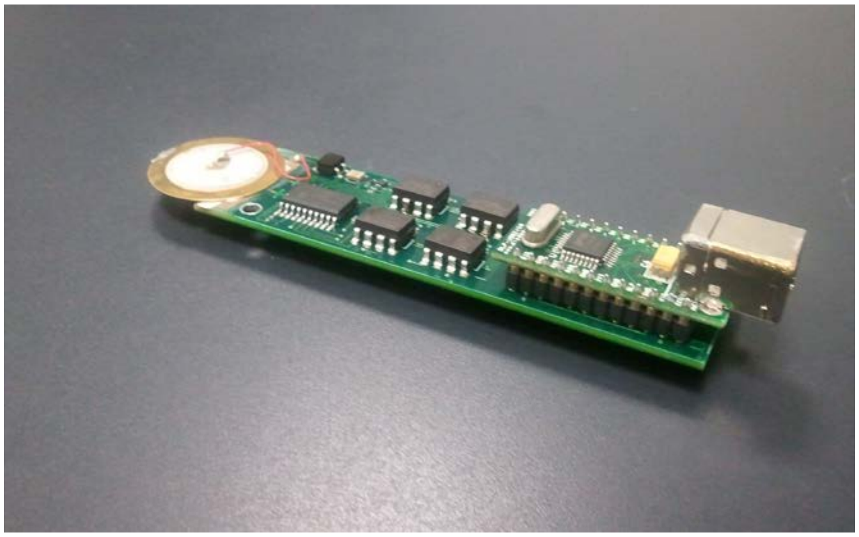

Figure 4.2 Side view of the VPIR device.

The VPIR device should be of great use to the Army: it should provide useful information as to the vibrations undergone by equipment and it will do so with very little need for maintenance.

Although the original goal has been accomplished, some work still must be done to make the VPIR device ready for military use. Primarily, the device must be tested to determine whether or not it, in its current form, is in accordance with military specifications. In addition to this, work could be done on the device to make it even smaller than it already is. Such miniaturization would likely be accomplished by doing away with the USB module and programming the microcontroller to handle all of the USB communications.

In the future, the functionality of the VPIR device could be expanded. In particular, the work done on the VPIR device could be built upon to develop a device that could record a detailed acceleration-versus-time profile of a vibration. Such information could be of great use to the Army in assessing the reliability of equipment.

A THE LABVIEW PROGRAM

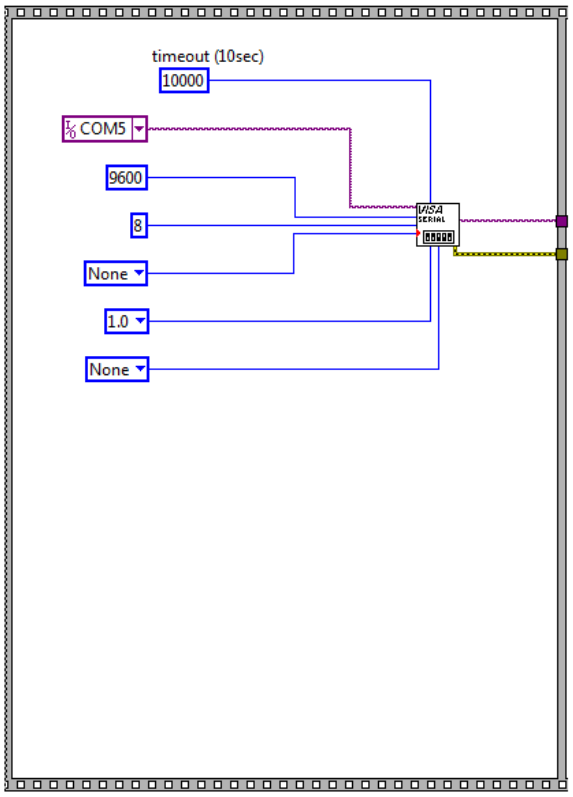

Figure A.1 Block one of the LabVIEW program.

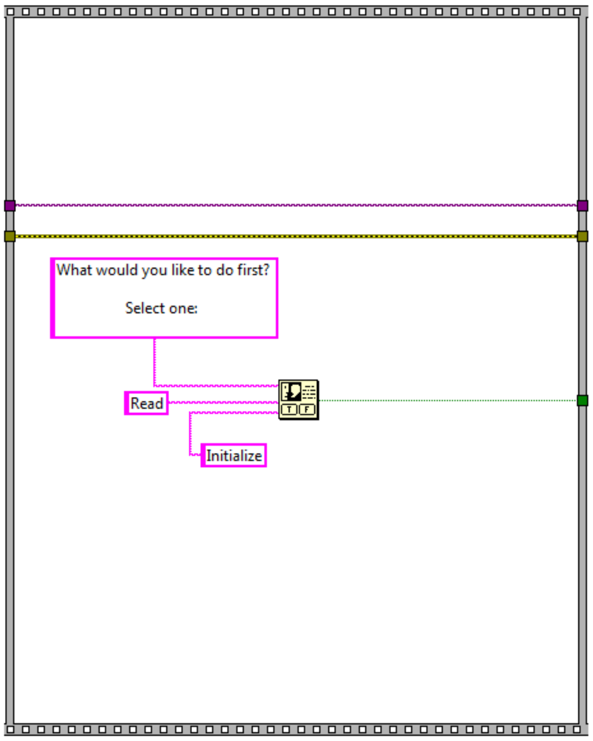

Figure A.2 Block two of the LabVIEW program.

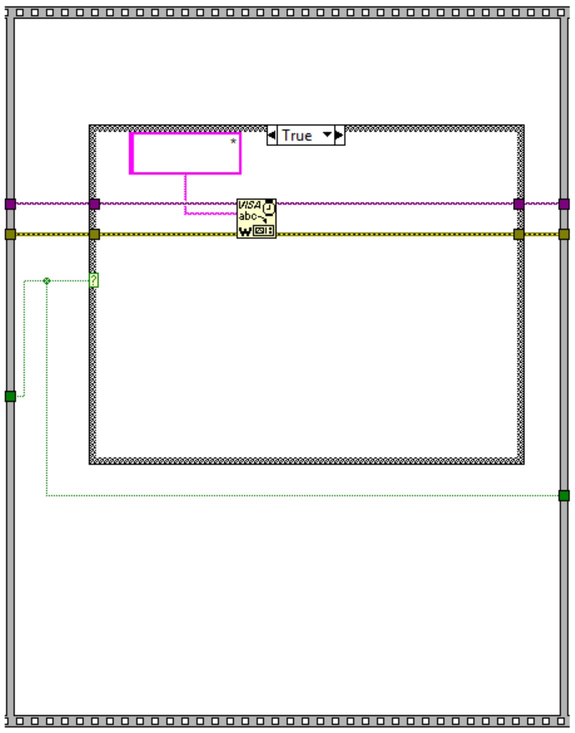

Figure A.3 Block three of the LabVIEW program (true condition).

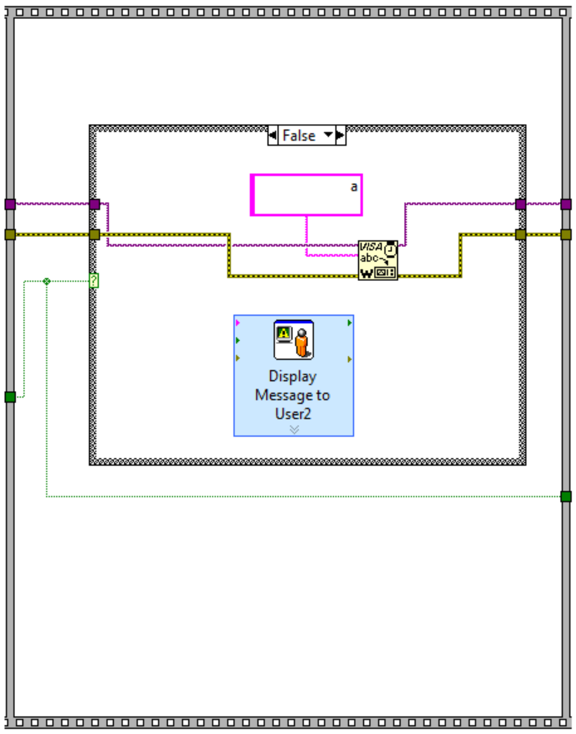

Figure A.4 Block three of the LabVIEW program (false condition).

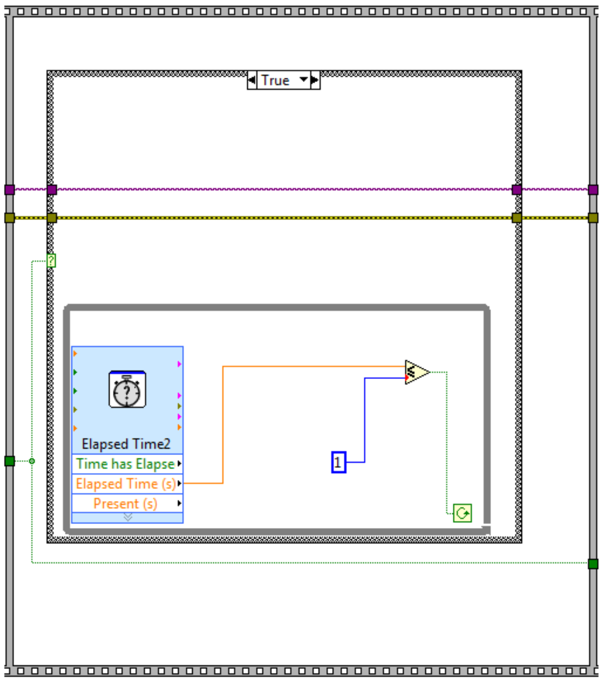

Figure A.5 Block four of the LabVIEW program (true condition).

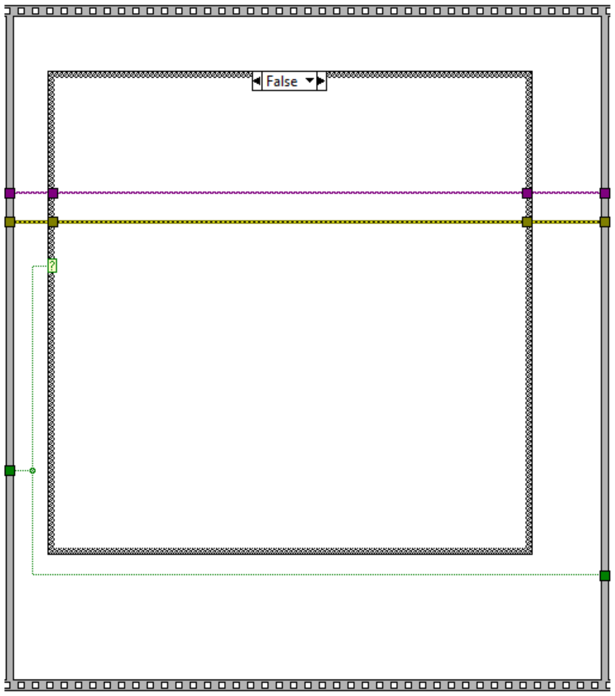

Figure A.6 Block four of the LabVIEW program (false condition).

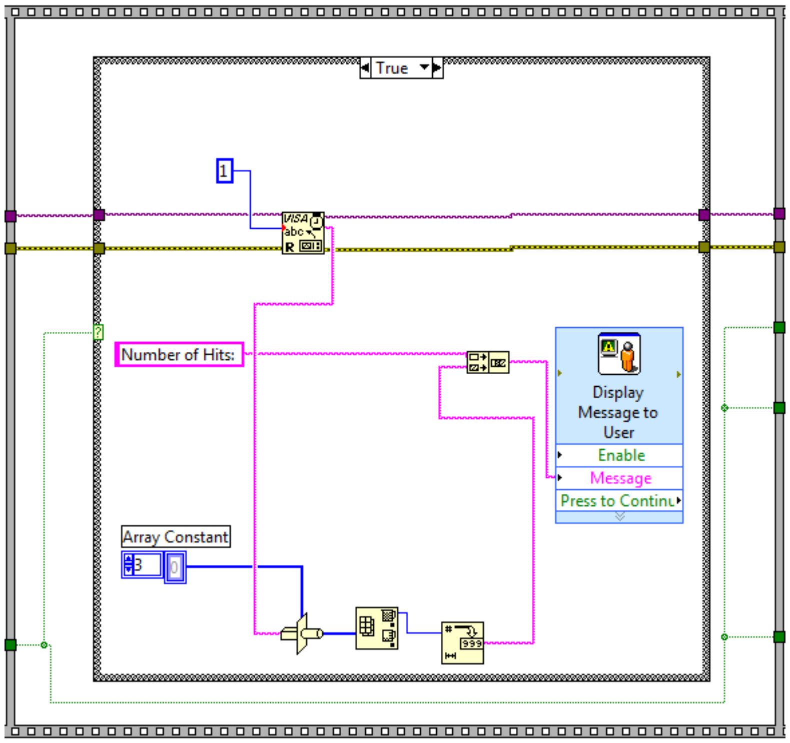

Figure A.7 Block five of the LabVIEW program (true condition).

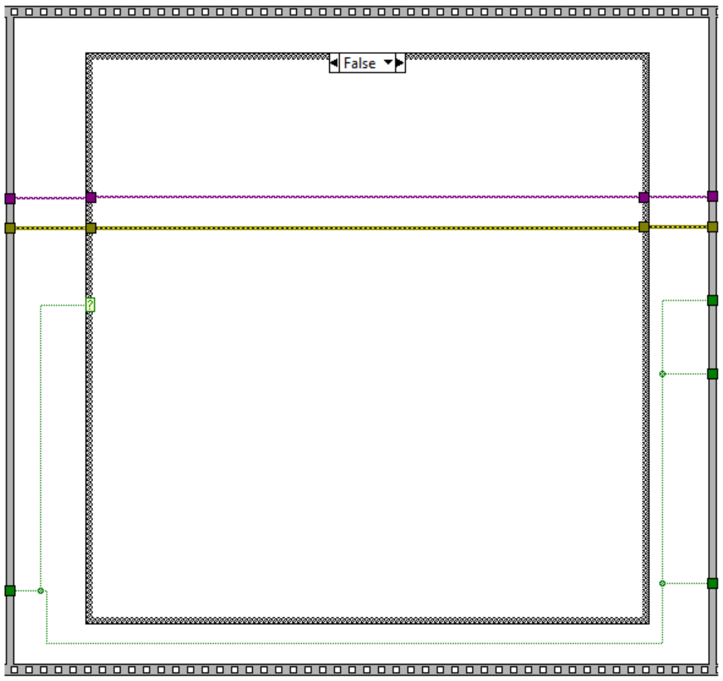

Figure A.8 Block five of the LabVIEW program (false condition).

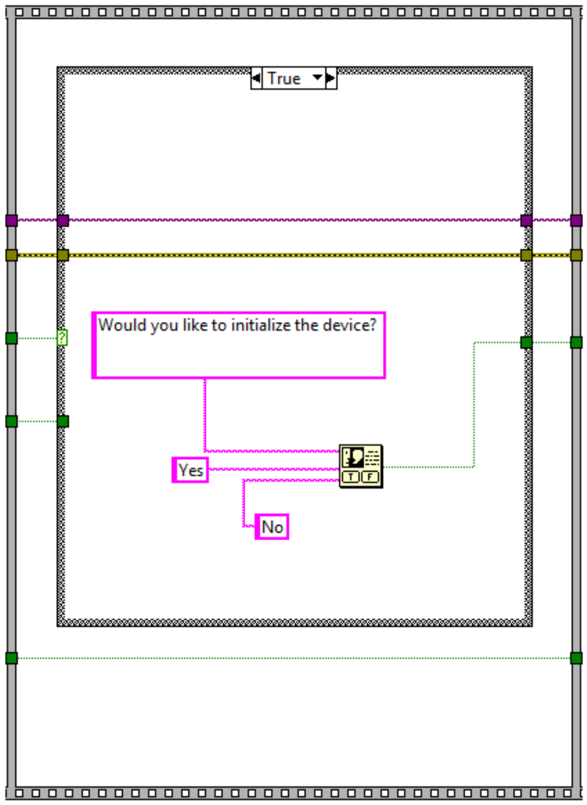

Figure A.9 Block six of the LabVIEW program (true condition).

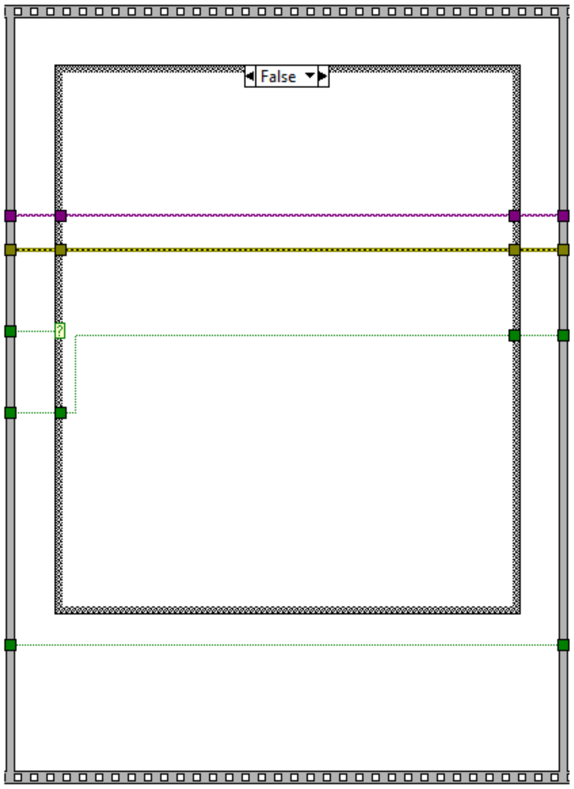

Figure A.10 Block six of the LabVIEW program (false condition).

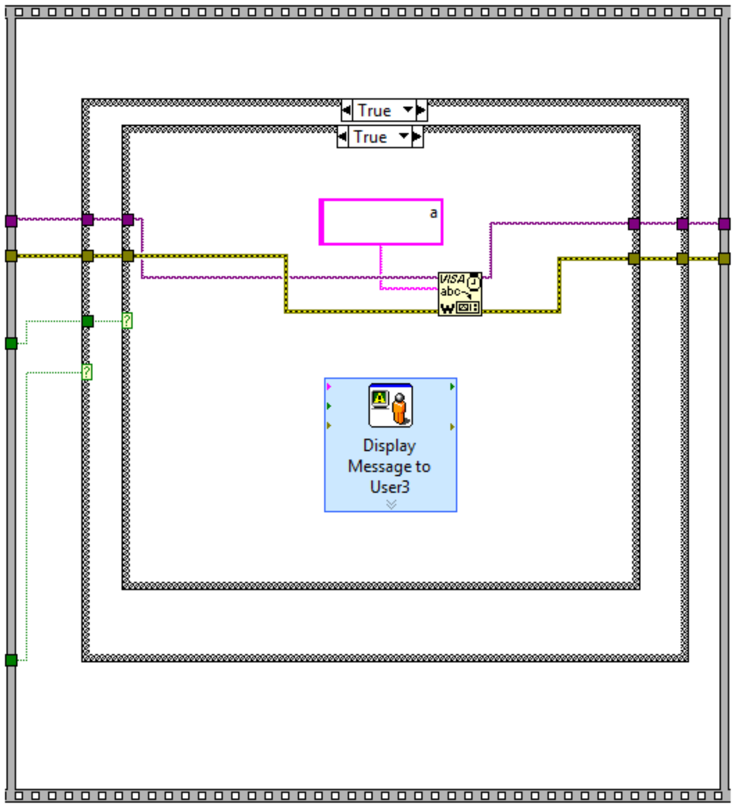

Figure A.11 Block seven of the LabVIEW program (inner condition: true, outer condition: true).

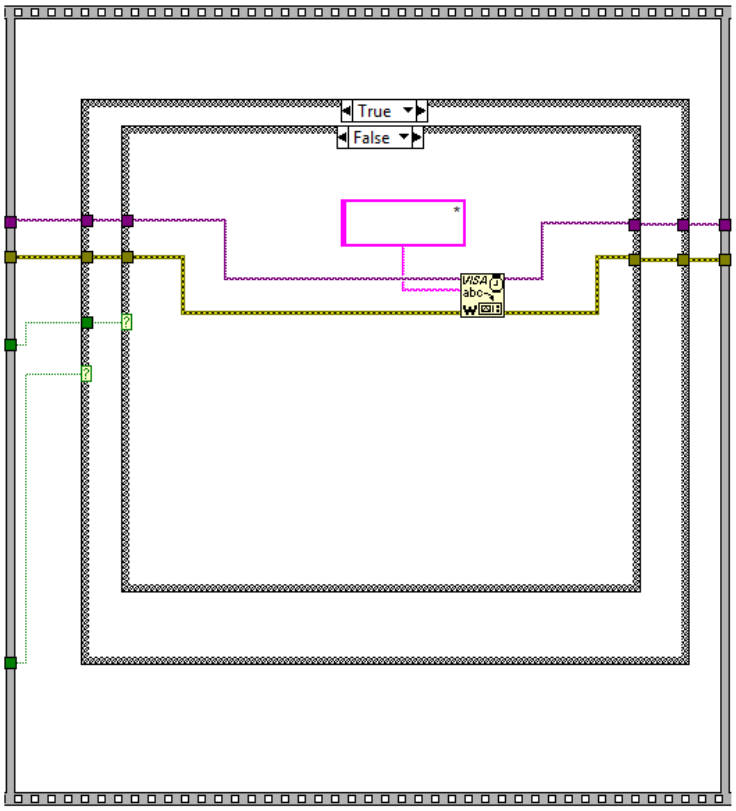

Figure A.12 Block seven of the LabVIEW program (inner condition: false, outer condition: true).

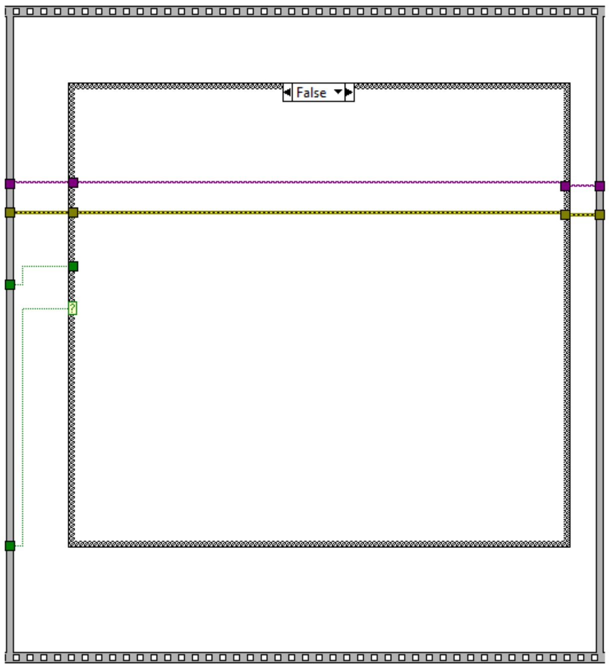

Figure A.13 Block seven of the LabVIEW program (outer condition: false, no inner condition).

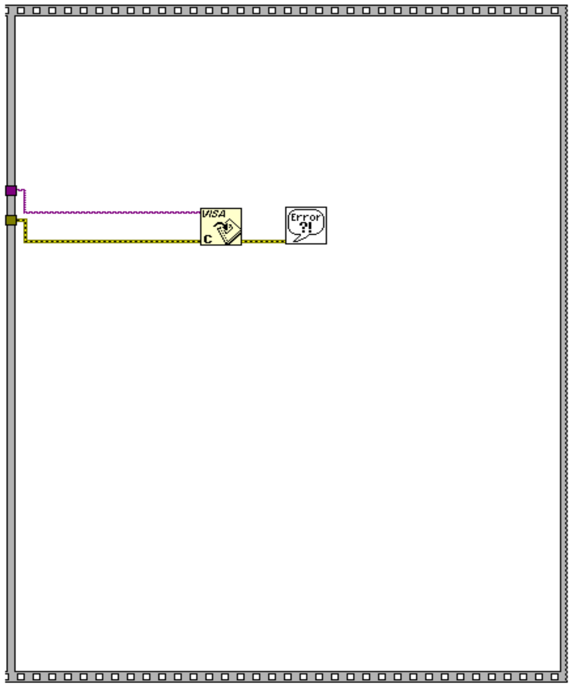

Figure A.14 Block eight of the LabVIEW program.

APPENDIX B   THE MICROCONTROLLER PROGRAM

```
list            p=16f677                        ;
#include        <p16f677.inc>                   ;


INT_VAR                 UDATA_SHR
w_temp                  RES    1                ;
status_temp     RES    1                        ;
pclath_temp     RES    1                        ;



TEMP_VAR    UDATA       0x20                    ;
temp_count  RES    1                            ;



RESET_VECTOR      CODE 0x000                    ;
nop                                             ;
goto        start                               ;

INT_VECTOR        CODE 0x004                    ;

MAIN CODE

INTERRUPT

movwf    w_temp                                 ;
movf     STATUS,w                               ;
movwf    status_temp                            ;
movf     PCLATH,w                               ;
movwf    pclath_temp                            ;

movf     pclath_temp,w                          ;
movwf    PCLATH                                 ;
movf     status_temp,w                          ;
movwf    STATUS                                 ;
swapf    w_temp,f
swapf    w_temp,w                               ;
retfie                                          ;
```

```
line 27:        START

line 28:        movlw    0x01                       ;
line 29:        banksel  OSCCON                     ;
line 30:        movwf    OSCCON                     ;

line 31:        banksel  ANSEL                      ;
line 32:        clrf     ANSEL                      ;

line 33:        banksel  ANSELH                     ;
line 34:        clrf     ANSELH                     ;

line 35:        movlw    0x10                       ;
line 36:        banksel  TRISA                      ;
line 37:        movwf    TRISA                      ;

line 38:        movlw    0x20                       ;
line 39:        banksel  TRISB                      ;
line 40:        movwf    TRISB                      ;

line 41:        banksel  TRISC                      ;
line 42:        clrf     TRISC                      ;

line 43:        banksel  PORTA                      ;
line 44:        clrf     PORTA                      ;

line 45:        banksel  PORTB                      ;
line 46:        clrf     PORTB                      ;

line 47:        banksel  PORTC                      ;
line 48:        clrf     PORTC                      ;

line 49:        banksel  PORTB                      ;
line 50:        btfss    PORTB, 5                   ;
line 51:        goto     GETDATA                    ;

line 52:        ;-----------------------------------------------
line 53:        WRITE

line 54:        banksel  EEADR                      ;  write mode begin
line 55:        clrf     EEADR                      ;

line 56:        banksel  EECON1                     ;
line 57:        bsf       EEPCON1, 0                        ;

line 58:        banksel   EEDAT                              ;
line 59:        incf      EEDAT, 1                           ;

line 60:        banksel   EECON1                             ;
line 61:        bsf       EECON1, 2                          ;

line 62:        DISABLE
line 63:        bcf       INTCON, 7                          ;
line 64:        btfsc     INTCON, 7                          ;
line 65:        goto      DISABLE                            ;

line 66:        movlw     0x55                               ;
line 67:        movwf     EECON2                             ;
line 68:        movlw     0xAA                               ;
line 69:        movwf     EECON2                             ;

line 70:        bsf       EECON1, 1                          ;

line 71:        DONEWRITE
line 72:        banksel   EECON1                             ;
line 73:        btfsc     EECON1,    1                       ;
line 74:        goto      DONEWRITE                          ;

line 75:        bcf       EECON1, 2                          ;

line 76:        goto      BACK                               ;


line 77:                                                     ;   write mode end

line 78:        ;-----------------------------------------------

line 79:        GETDATA                                       ;    read mode begin

line 80         banksel   PORTA                              ;
line 81         bsf       PORTA, 0                           ;
line 82         bsf       PORTA, 1                           ;

line 83         movlw     0x31                               ;
line 84         banksel   OSCCON                              ;
line 85         movwf     OSCCON                              ;



line 86         FIRSTREADWAIT
line 87         banksel   PORTA                               ;
line 88         btfsc     PORTA, 4                            ;
line 89         goto      FIRSTREADWAIT                       ;

line 90         call      OBTAINDATA                          ;

line 91         banksel   PORTA                               ;
line 92         movf      0x40                                ;

line 93         sublw     0x2A                                ;
line 94         btfsc     STATUS, 2                           ;
line 95         goto      SENDDATA                            ;
line 96         goto      INITIALIZE                          ;



line 97         SENDDATA

line 98         banksel   EEADR                               ;
line 99         clrf      EEADR                               ;

line 100:       banksel   EECON1                              ;
line 101:       bsf       EECON1, 0                           ;


line 102        banksel   EEDAT                               ;
line 103        movf      EEDAT, 0                            ;
line 104        banksel   PORTC                               ;
line 105        movwf     PORTC                               ;


line 106        banksel   PORTA                               ;


line 107        PINSTATE
line 108        clrf      0x20                                ;
line 109        movf      PORTA, 0                            ;
line 110        movwf     0x30                                ;
line 111:    btfss    0x30, 0                  ;
line 112:    bsf      PORTA, 0                 ;
line 113:    btfsc    0x30, 0                  ;
line 114:    bcf      PORTA, 0                 ;
line 115:    btfss    PORTA, 4                 ;
line 116:    goto     READBUFFER               ;
line 117:    LOOP
line 118:    incf     0x20, 1                  ;
line 119:    movf     0x20, 0                  ;
line 120:    sublw    0x20                     ;
line 121:    btfss    STATUS, 2                ;
line 122:    goto     LOOP                     ;
line 123:    goto     PINSTATE                 ;


line 124:    READBUFFER

line 125:    banksel  PORTA                    ;
line 126:    bsf      PORTA, 0                 ;

line 127:    call     OBTAINDATA               ;

line 128:    sublw    0x2A                     ;
line 129:    btfsc    STATUS, 2                ;
line 130:    goto     BACK                     ;
line 131:    goto     INITIALIZE               ;   read mode end

line 132:    ;-----------------------------------------

line 133:    ;============================= =INITIALIZE BEGIN

line 134:    INITIALIZE

line 135:    banksel  EEDAT                    ;
line 136:    clrf     EEDAT                    ;
line 137:    banksel  EEADR                    ;
line 138:    clrf     EEADR                    ;

line 139:    banksel  EECON1                   ;
line 140:    bsf      EECON1, 2                ;

line 141:    DISABLETWO
line 142:    bcf      INTCON, 7                ;
line 143:    btfsc    INTCON, 7                ;
line 144:     goto      DISABLETWO                    ;

line 145:     movlw     0x55                          ;
line 146:     movwf     EECON2                        ;
line 147:     movlw     0xAA                          ;
line 148:     movwf     EECON2                        ;

line 149:     bsf       EECON1, 1                     ;

line 150:     DONEWRITETWO
line 151:     banksel   EECON1                        ;
line 152:     btfsc     EECON1,    1                  ;
line 153:     goto      DONEWRITETWO                  ;

line 154:     bcf       EECON1, 2                     ;

line 155:     goto      BACK                          ;


line 156:     ;================================== =INITIALIZE END

line 157:     ;===================================OBTAINDATA BEGIN

line 158:     OBTAINDATA
line 159:     movlw     0xFF                          ;
line 160:     banksel   TRISC                         ;
line 161:     movwf     TRISC                         ;

line 162:     call      WAIT                          ;

line 163:     banksel   PORTA                         ;
line 164:     bcf       PORTA, 1                      ;

line 165:     call      WAIT                          ;

line 166:     banksel   PORTC                         ;
line 167:     movf      PORTC, 0                      ;

line 168:     banksel   PORTA                         ;
line 169:     movwf     0x40                          ;

line 170:     banksel   PORTA                         ;
line 171:     bsf       PORTA, 1                      ;
line 172:     banksel   TRISC                        ;
line 173:     clrf      TRISC                        ;

line 174:     return                                 ;

line 175:     ;============================== OBTAINDATA END



line 176:     ;================================== WAIT BEGIN


line 177:     WAIT
line 178:     banksel   PORTA                        ;
line 179:     clrf      0x20                         ;
line 180:     WAITTWO
line 181:     incf      0x20, 1                      ;
line 182:     movf      0x20, 0                      ;
line 183:     sublw     0x20                         ;
line 184:     btfss     STATUS, 2                    ;
line 185:     goto      WAITTWO                      ;

line 186:     clrf      0x20                         ;

line 187:     return                                 ;

line 188:     ;==================================WAIT END

line 189:     ;================================== BACK BEGIN

line 190:     BACK
line 191:     banksel   PORTB                        ;
line 192:     bsf       PORTB, 6                     ;
line 193:     movlw     0x01                         ;
line 194:     addlw     0x01                         ;
line 195:     movlw     0x02                         ;
line 196:     addlw     0x01                         ;
line 197:     goto      BACK                         ;

line 198:     ;================================== BACK END


line 200:     EE      code  0x2100
line 201:          DE 5,4,3,2,1
line 202:          END                          ;
```

REFERENCES

Roundy, Shad, Paul Kenneth Wright, and Jan M. Rabaey. Energy Scavenging For Wireless Sensor Networks. Boston: Kluwer Academic Publishers, 2004.

Peckol, James K. Embedded Systems. Hoboken: John Wiley and Sons, 2008.

Priya, Shashank, ed., and Daniel J. Inman, ed. Energy Harvesting Technologies. New York: Springer, 2009.