Texas A&M University at Qatar
Electrical and Computer Engineering Program

ECEN 404-502

Electrical Design Laboratory II

Semester: Spring 2014

Progress Report

Data Logger for Mechanical Systems

Team Members:
Abdulrahman Al-Malki
Faisal Al-Mutawa
Mohammed Alsooj
Yasmin Hussien

Mentor: Dr. Shehab Ahmed

Due Date: Mar 22nd, 2015

"On our honor, as Aggies, we have neither given nor received unauthorized aid on this academic work."

Abstract

Our project is an industrial project designed to work with mechanical systems in order to transform them from being merely passive to active logging tools by adding some electronic components to them. These electrical components will act as a data logger that logs different measurements such as translation, rotation and vibration and log this information against time. Afterwards it stores them into memory for easy retrieval using a LabVIEW VI that we will provide. The device might also include different measurements in the future such as temperature and pressure to get more data about the location of the system. The purpose of this project is to give clear data about the status of the mechanical system during operation. We believe that our project will be beneficial to the industry and will be very helpful in the case of operating in new locations for navigation purposes, or investigation purposes in the case of an accident.

Contents

Abstract **1**

Chapter 1: Literature Review **4**

1.1 Introduction 5

1.2 Market Analysis 6

1.2.1 Introduction *6*

1.2.2 Case under Study *6*

1.2.3 Analysis *7*

1.2.4 Impact on Design *11*

1.3 Semester Plan 12

1.3.1 Timeline *12*

1.3.2 Task Assignment *13*

Chapter 2: Project Design and Development Process **15**

2.1 Benchmarking 16

2.1.1 Introduction *16*

2.1.2 Criteria *17*

2.1.3 Comparison *18*

2.1.4 Analysis *19*

2.2 Functional Modeling 20

2.2.1 Introduction 20

2.2.2 Upper Level Model 20

2.2.3 Detailed Model 20

2.2.4 Analysis 21

2.3 Concept Generation and Selection 22

2.3.1 Introduction *22*

2.3.2 Concept Generation Matrix *22*

2.3.3 Concept Evaluation *23*

Chapter 3: Detailed System Design and Modeling **24**

3.1 System Design and Modeling 25

3.1.1 Programming Sub-system *25*

3.1.2 PCB Sub-system *26*

3.2 Simulation Results 29

3.2.1 Programming Sub-system *29*

3.2.2 PCB Sub-system *29*

3.3 Component List 30

Chapter 4: Experimental Results **31**

4.1 Testing and Troubleshooting 32

4.2 Progress 33

Chapter 5: Conclusion **34**

5.1 Discussion 35

5.2 Demo Day Plans 35

References **36**

Appendices **37**

A. Interview Script 37

B. Survey Questions 38

C. Team Agreement 40

D. Full Code 43

Chapter 1: Literature Review

1.1 Introduction 5
1.2 Market Analysis 6
1.2.1 Introduction *6*
1.2.2 Case under Study *6*
1.2.3 Analysis *7*
1.2.4 Impact on Design *11*
1.3 Semester Plan 12
1.3.1 Timeline *12*
1.3.2 Task Assignment *13*

1.1 Introduction

This is a mid-progress report of our project. It will showcase all the work we did from the beginning of the fall semester until the middle of the spring semester. This report will include five chapters explaining the different tasks that were performed throughout the both semesters and their importance in developing our project. We will first start by a literature review that includes our customer needs analysis followed by an ethnographic study to get some feedback from various customers regarding our project and to study the culture of our project. After that to further develop our data logger the following chapter will discuss how to develop our design by conducting a benchmarking analysis, functional modeling and a concept generation study. Chapters three and four will talk about the technical aspects of our project. Finally, chapter 5 will have a discussion along with future plans.

1.2 Market Analysis

1.2.1 Introduction

The first objective of this study is to conduct a thorough analysis of the customer needs and feedback regarding our senior design project "Data Logger for Mechanical Systems." The outcome of the report will help us formulate our design and optimize its features to match the needs of the customers. We have targeted people with different backgrounds to get a wider range of results and different views about the project, in order to satisfy more potential customers. This section will include the methods used in getting the data, a brief summary of the design, and data analysis.

The second objective is to do an ethnographic study to analyze the culture that is related to our project. Before diving into the details of our project, we as engineers need to analyze the people who will be using our product. We need to understand the problems and difficulties they face every day, as well as get insight from them on how we can improve our design to cater to them. Tasks were divided among team members as shown in Table 1.

Table 1: Task Division among Team Members

Team Member	Task
Abdulrahman Al-Malki	Technical Writer
Faisal Al-Mutawa	Editor/PR Person
Mohammed Alsooj	Cameraman
Yasmin Hussien	Script Writer

1.2.2 Case under Study

Since our customers are all related to the industry field, a survey was sent out to three Qatari oil and gas companies. Google Docs was used for conducting the survey and it was sent by email. A phone interview was conducted with an electrical engineer in the oil and gas department in an international company, and his responses were taken into consideration. In addition, two interviews were also conducted with two mechanical engineering professors from the faculty of Texas A&M University at Qatar. These professors were recommended by our mentor since they have a lot of experience with downhole tools. Interview script and survey are in Appendix A and B, respectively.

For the ethnographic study, so we focused our study on people with industry experience. First we talked with Dr. Shehab Ahmed to give us give us initial insight. He is an electrical engineering professor with valuable theoretical knowledge, as well as, a lot of practical experience in the oil and gas industry. Next we talked with Eng. Nasser Elfayoumi from RasGas to hear about the issues related to our project. Finally, we went to Schlumberger in Qatar to get first-hand experience with how our project can help the industry. The people we interviewed are shown in Table 2.

Table 2: People Interviewed for our Ethnographic Study

Case	Location	Person Interviewed
University Professor	Texas A&M University at Qatar	Dr. Shehab Ahmed
RasGas Engineer	RasGas HQ	Nasser Elfayoumi
Schlumberger	Industrial Area	Yassine Mhadhbi

1.2.3 Analysis

In-Person Interviews:

We conducted two interviews, both with mechanical professors who had background information about mechanical systems and data logging. These face-to-face interviews helped us gather some essential feedback regarding our project with professional engineers who have a lot of experience.

In the first interview, the professor discussed many points that support our data logging activity during the drilling process. He focused on the friction, depth, and acceleration of the mechanical tool that he suggested, such as drill string dynamic tool. In addition, he mentioned the weight on bit measurements, and he suggested we take these measurements into account when building our circuit board. The professor also pointed out that it is better for the users to get the data that are being logged instantaneously so that the users would get to know what is happening at any certain moment or time during the drilling. Finally, the professor recommended us to search more for the module of these tools such as the components, parts, and even how the tools function. The reason for this search is to help us better understand the requirements of these tools, especially when designing a circuit board with sensors that need certain constraints to be considered.

In the second interview, we also discussed drilling tools and downhole tools. The professor said that such device should have priority in accuracy more than anything. She explained that the device will have many factors affecting its accuracy and that we need to concentrate on minimizing the outside effects for a better product. She also mentioned another problem that should be considered, which is the correlation of logged data that might change for different tools; for example the dependence of speed and rotation. As for harsh environment capability, the professor thinks that temperature, friction and vibration are the most important to deal with. Finally, the size of the board itself would have to be properly selected in order to avoid vibration effects and those larger boards need to be more robust.

Phone Interview:

A phone interview was completed with an electrical engineer in the oil and gas department in an international company regarding our product. He mentioned that they don't have any constraints regarding the size of the board, however it should be within an appropriate range. He also added that their tools aren't used in harsh conditions, therefore there is no need for extreme protection. His recommendation was to focus on measuring the vibrations, temperature, flow rate and pressure, since these are the main measurements their mechanical tools use.

Survey:

For the survey portion of our analysis, we targeted oil and gas companies to get information from people who deal with mechanical systems in their everyday lives. Our survey questions weren't detailed, but rather focused on getting quick details about:

a) the type of data is mostly needed from their respective mechanical systems, and
b) the type of environments that our data logger has to withstand.

For the type of data that needs to be logged, all 80 responders chose to log vibration data. That shows us just how important it is, as it was also emphasized by our interviews. This is mostly because downhole tools were the most common mechanical systems that our responders worked with, and vibration would benefit them the most since it is the quickest way of sensing sudden changes in motion.

Other common types of data were temperature and rotation. Temperature being frequent mostly tells us that it is needed in order relate it with other data, i.e. to detect

the kind of environment the mechanical system was in when an even happened. Rotation was less frequent than vibrations, but it is till important to us because it gives more details about how the mechanical system is performing. There were also a few responses about logging linear motion pressure (part of others). They don't appear to be as important, and they were mostly chosen by people who don't work with downhole tools. Figure 1 shows a bar chart of the responses to the type of data question.

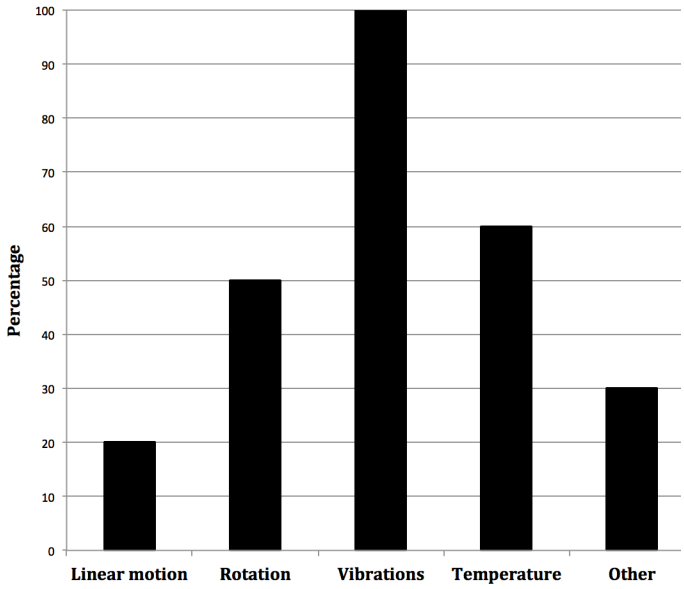

Figure 1: The responses to the question: "What type of data could our data logger collect in order to help improve your tool?"

The other set of questions were about the environment that our data logger has to withstand. Again since most responders were people who deal with downhole tools, we wanted to know what kind of harsh conditions are commonly encountered in downhole applications.

We classified the environment question into three categories:

a) Temperature
b) Depth
c) Surrounding materials

The temperature results were varied. The majority chose temperatures ranging from 0 to 50 ° C, while the rest chose more than a 100 ° C. Nonetheless, this means that we have to cater for all ranges to satisfy all customers.

The depth results were as expected, with most responders choosing more than 500 ft. This just enforces our predictions. The surrounding materials results, on the other hand, were very dispersed, which means that we have to account for every common response. This includes the "other" category, where responders suggested we account for oil and gas as surroundings. The results of the environment question are shown in three pie charts in Figure 2.

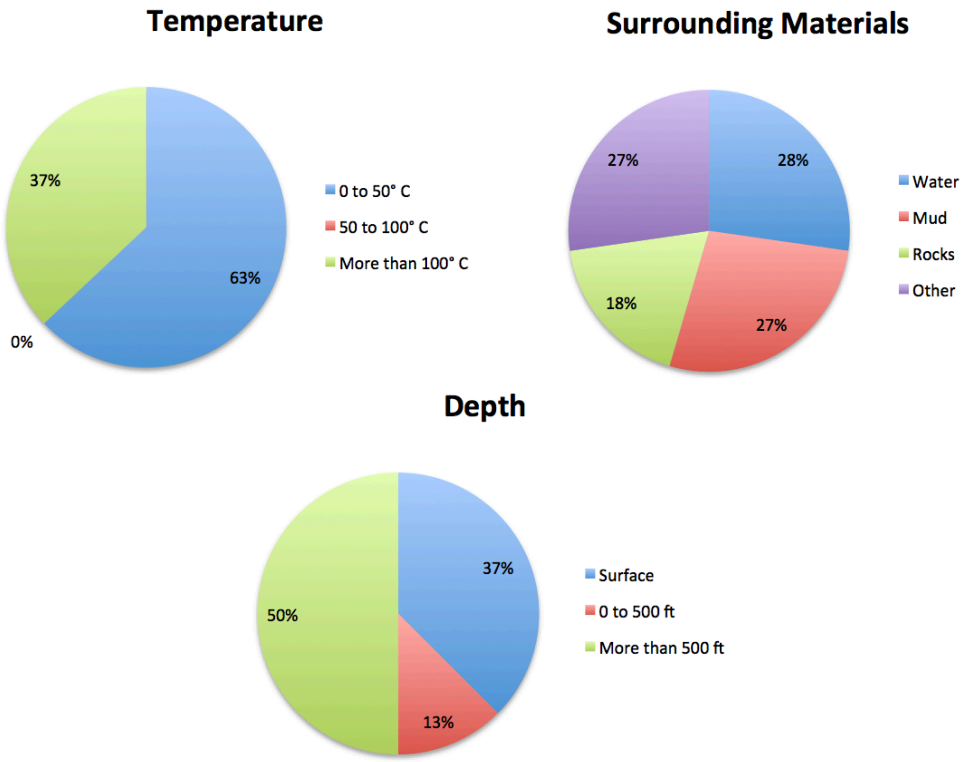

Figure 2: The responses to the question: "What kind of environment does your mechanical system operate?"

University Professor:

Dr. Shehab Ahmed gave us an analogy to help explain the idea behind our project better. He compared our data logger to fitness trackers that are currently trending around the world. He explained that these fitness trackers help record and measure our "performance" when we are exercising for example. This helps us realize if we are exercising right by checking how much we improve over time, as well as show us where we are doing something wrong so that we can improve.

He further explained that this could be said about passive mechanical systems. When the tools are being utilized in harsh environments like downhole or even outer space, it's very difficult to know whether the tool is working or not. Plus, even if it's working

we can't know how efficient it is, resulting in losing opportunities to enhance the tool. Therefore, our data logger can help to prove and improve the performance of a mechanical system.

RasGas Engineer:

Nasser Elfayoumi, a petroleum engineer, talked about examples of current systems that utilize data logging. He talked about offshore systems used in oil retrieval. These systems operate in the sea where it is significantly more difficult to control the tools than on land. This is because of thing that we can't control like the wind and the tide. This makes delicate processes like moving oil platforms very difficult and very dangerous. Because of the massive size of these platforms, any unplanned contact will result in a very powerful collision that could cost casualties and millions of dollars.

He then explained how data logging could help immensely in these situations. By having precise data logs of the motion of the oil platforms, they can be moved in a very accurate manner. This can be even further improved by having automated systems that use the data logs to move the oil platforms without human intervention. He explained this would help a lot since human workers can't possibly work in these harsh environments for prolonged periods of time. Machines, on the other hand, are capable of working in harsh environments, which is another benefit of data loggers.

Schlumberger:

We went on a trip to Schlumberger in the Industrial Area with a view to learning more about how data loggers are used currently in the industry. We went on a tour first and looked at several tools used in the field. We then took a look at how the data is looked at and interpreted by an Engineer working in Wireline. We also had an interview with Yassine Mhadhbi, who is a petroleum engineer with first hand experience in downhole tools and data logging. He talked about the safety aspect of data logging, as well as how data logging is implemented in different mechanical tools.

1.2.4 Impact on Design

From Dr. Shehab's analogy, we realized that the user experience is a very crucial part of our design project. We need to make our design as accessible and user-friendly as possible. We can do this by adding more common peripherals for when the user wants

to view the data. If time allows, we can also make the connector modular so that it can be replaced with user's preferred peripheral protocol.

For RasGas and Schlumberger, they seemed to be concerned the most with the safety aspect of data logging. In this case, we could implement in our code a "burst logging" mode where if the sensors detect movements, logging happens every millisecond instead of every second. This would help in viewing certain events in detail for troubleshooting purposes.

1.3 Semester Plan

1.3.1 Timeline

The fall semester was dedicated to doing research and planning the theoretical parts of our project. We did various assignments in order to study the market and the culture of our project, as well as doing mental exercises to gather ideas and evaluate them. Table 3 shows the detailed timeline for the fall semester.

Table 3: Project timeline for the fall semester

Task	September				October				November				Dec	
	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2
Drafting project proposal														
Designing project website														
Drafting team agreement														
Researching customer needs														
Doing ethnographic study														
Benchmarking with other products														
Modeling project functions														
Generating and evaluating concepts														
Finding and ordering components														
Drafting progress report														

The spring semester will be dedicated to the technical parts of the project. We will start with the software, move to hardware, and then finish off with optimizations and tests. Simulations will also be done throughout the semester. Table 4 shows the detailed timeline for the spring semester.

Table 4: Project timeline for the spring semester

Task	January				February				March				April			
	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
Designing real-time clock																
Establishing UART connection																
Designing PCB schematic																
Programming accelerometer																
Interfacing memory																
Ordering components																
Finalize PCB routing																
Design plastic housing																
Final testing																
Preparing for Demo Day																

1.3.2 Task Assignment

We try to assign tasks fairly and efficiently, and we also want to make sure that no team member works alone for a prolonged period of time. That is why most tasks are handled by two members, with some details handled individually. The team organization for the nontechnical parts is in the Team Agreement (Appendix C), while the technical parts are organized as follows:

Programming subsystem:

- Abdulrahman Al-Malki:
  - Main responsibilities: Designing and simulating the real-time clock and designing LabVIEW modules
  - Secondary tasks: compiling the code and helping with the UART connection and accelerometer simulation
- Faisal Al-Mutawa:
  - Main responsibilities: Establishing the UART connection, accelerometer connection/simulation, and interfacing memory
  - Secondary tasks: helping with the simulation of the real-time clock

PCB subsystem:

- Mohammed Alsooj:
  - Main responsibilities: Design of the PCB schematic and routing
  - Secondary tasks: helping with the finding the components
- Yasmin Hussien:
  - Main responsibilities: Finding and ordering the components
  - Secondary tasks: helping with PCB schematic

Chapter 2: Project Design and Development Process

2.1 Benchmarking 16

2.1.1 Introduction 16

2.1.2 Criteria 17

2.1.3 Comparison 18

2.1.4 Analysis 19

2.2 Functional Modeling 20

2.2.1 Introduction 20

2.2.2 Upper Level Model 20

2.2.3 Detailed Model 20

2.2.4 Analysis 21

2.3 Concept Generation and Selection 22

2.3.1 Introduction 22

2.3.2 Concept Generation Matrix 22

2.3.3 Concept Evaluation 23

2.1 Benchmarking

2.1.1 Introduction

The purpose of this study is to perform benchmarking on our senior design project by comparing it with existing products in the industry that have similar functionalities. By doing this, we are able to get a more in-depth look into the approach that other engineers took when with faced the problems given by their customer. Ultimately, we will be able to tell how our current design stands against the current products, and we will attempt to improve the possible inadequacies in our design.

We have found three products that have similar functionalities to our design. The first one is the H.E.A.T. Evaluation Module from Texas Instruments[1]. Like our project, it is a PCB that logs the performance of mechanical tools using various sensors. The second product is CompactDAQ from National Instruments[2], which is a stand-alone platform instead of is a bare PCB but still a rugged data logger. The third product is MSR145 from CiK Solutions[3]. It is also a stand-alone platform that records data against time. Pictures of all the products are shown in Figure 3.

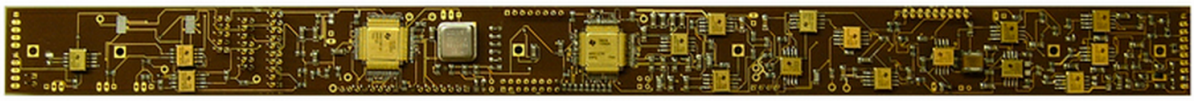

Figure 3a: The Harsh Environment Acquisition Terminal Evaluation Module

Figure 3b: CompactDAQ chassis with its various modules

Figure 3c: MSR145 inside its design housing

2.1.2 Criteria

We have decided on the following metrics that are most relevant to our design. For our design, some of the details are still undecided. But, we will still compare these details to help us decide later on.

1. **Operation time**: this assesses how efficient the product is in terms of power consumption. This is important because operations may take around 24 hours to complete while the product is inaccessible during that time.

2. **Power source:** this shows from where the product draws power. Could be onboard battery, connected to an external power supply, etc.

3. **Dimensions:** this assesses the size of the product. The smaller, the better since it might not fit in smaller mechanical tools.

4. **Sensors:** this shows what kind of sensors the product has. The more, the better assuming it does not affect the dimensions.

5. **Memory:** this assesses how much data the product can record without overwriting older data. The larger, the better assuming it does not affect the dimensions.

6. **Acquisition frequency:** this shows how often the product logs the data against time. The higher, the better assuming the memory can handle it.

7. **Temperature tolerance:** this assesses how the product handles harsh environment conditions, which in this case is temperature associated with downhole operations.

8. **Connectivity:** this shows what and how many types of connectors/buses are available in the product. The more, the better assuming it does not affect the dimensions.

9. **Price:** this assesses the reasoning behind the price tag of the product.

2.1.3 Comparison

The comparison between our design and the similar products based on the metrics we decided on are shown in Table 3.

Table 5: Benchmarking Table

Metric	Our Design Project	H.E.A.T Evaluation Module	CompactDAQ (cDAQ-9134)	MSR145 (Standard IP 60)
Operation Time	12 Hours	Infinite	Infinite	2-3 Months
Power Source	Battery	3.3 V External Supply	24 V 5A External Supply	Lithium-polymer battery (800mAh)
Dimensions (LxWxH)	10x6x1 inches*	15.6x1x0.93 inches	8.66x4.6x3.4 inches	2.8x1.5x0.9 inches
Sensors	Rotation, acceleration, vibration	Rotation, pressure, temperature	Holds 4 External Sensors from 50+ sensor modules	Holds 5 External Sensors from 8 different sensor modules or any analog signal
Memory	64 KB	Onboard 32 Mbit	SD Card (Max 16 GB)	Onboard 8 Mbit
Acquisition Frequency	1 per second	Up to 128k per second	Up to 50k per second	Up to 50 per second
Temperature Tolerance	175 ˚C	210 ˚C	70 ˚C	65 °C
Connectivity	RS232 to USB	CAN	CAN/USB	USB
Price	N/A	$ 5749	$ 7068	$323 (Excluding Sensors)

2.1.4 Analysis

After doing this study, we found several things that we could do to improve our design. They are summarized in the following points according to the metrics highlighted in bold.

1. **Acquisition frequency:** this was the most important metric where our design falls short. Currently we designed a 24-hour clock that has increments of seconds using C code. From our conclusion in our ethnographic study, we may implement a burst-logging mode that triggers when the sensors detect certain types of motions. We will have to make the frequency much higher so that it stacks up against the other products.

2. **Sensors:** right now, we focused from our customer needs assignment on the things that companies want from these data loggers. However, we may decide to implement a modular design like some of the similar products. This, of course, must not interfere with our **size** constraint since it is also requested by our customer.

3. **Power source:** we still have not decided on the type or the capacity of the battery that will power our design. The MSR145 from CiK Solutions seems to be very good in terms of **operation time**, so we might consider doing something similar to them.

4. **Memory:** this is still a vague part of our design right now. As of right now, we are thinking of storing our data logs in an onboard memory and then transferring it using CAN to a computer software. But, SD cards could be a better option since it is a prevalent standard that is available in a lot of devices right now. This must not interfere with the **temperature tolerance**, though. So if we decided to use SD cards, we will have to find a way to make it work in harsh environments.

2.2 Functional Modeling

2.2.1 Introduction

The purpose of this study is to model the various functions of our design project in order to clarify the utility of each part. We will divide each function into a block with an input and output, and relate each function to the other to form the main function of our design. This will help us gain a deeper understanding of each part, and also makes is easier to find errors later by making the design process modular.

2.2.2 Upper Level Model

First, we decided on what would be the first thing that counts as input to our product, and what is the final output that will come out of it. This will demonstrates a top-level view to our product as shown in Figure 4.

Figure 4: Top-level block diagram showing the input and the final output

2.2.2 Detailed Model

The next step is to model the various functions and energy conversions that happen inside the "product" block. This is shown in Figure 5. Note that ME and EE stand for mechanical energy and electrical energy, respectively.

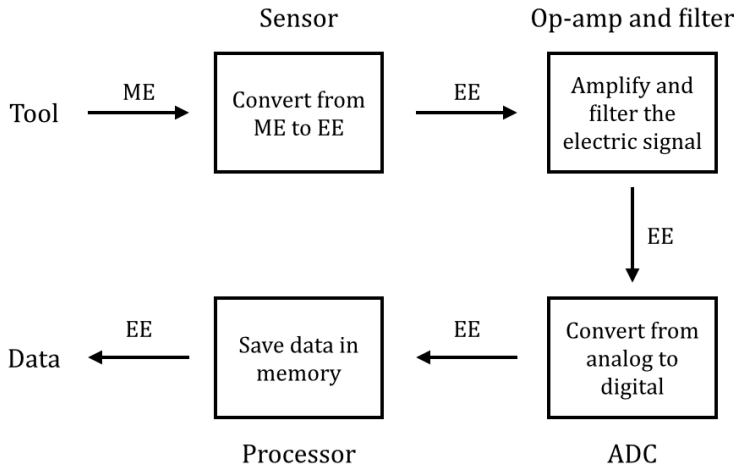

Figure 5: Detailed model showing the functions of each part in our product

The functions of each block are as follows:

1. Sensor: the sensor is used to detect the motion of the mechanical tool and convert it to electrical energy.
2. Op-amp low-pass filter: since the output of a sensor is of low voltage, an op-amp is connected to its output to make the data more discernable. The signal then is filtered by a low-pass filter to get rid of the unneeded frequencies.
3. ADC: since the motion of the mechanical tool is analog, an analog-to-digital converter is used to convert the output of the op-amp to a digital signal.
4. Processor: the digital signal from the ADC is then accesses by the processor and saved in memory against a real-time clock. The data then can be views by connecting it to the client's computer through CAN.

2.2.4 Analysis

This model helped in illustrating the details that happen between logging the data and retrieving it. We found out that what each component does and where it goes. This will benefit us greatly when implementing the design since we will be able to easily monitor each step during testing and debugging.

2.3 Concept Generation and Selection

2.3.1 Introduction

The purpose of this study is to improve our design by brainstorming ideas for the main functions of our product and evaluating them. This process is detailed in the two sections: creating a morphological matrix to come up with concepts, and creating a screening table to evaluate the concepts that were found earlier.

2.3.2 Concept Generation Matrix

This section involves breaking up our product into several functions that can be thought of as problems, and then come up with different solutions to that problem. The solutions will be either components or design elements. Table 4 shows the morphological matrix of our project that details the functions of our product along with their solutions.

Table 6: Morphological matrix for possible concepts of our projects

Function	**Solutions**		
Time Tracking	Software based	Hardware based	Crystal
Measuring Position	Hall-effect sensor	LVDT	GPS
Measuring Motion	Shock sensor	Accelerometer	Gyroscope
Retrieving Data	SD card	CAN	USB

The details of the functions and their solutions are as follows:

1. Time tracking: our data logger records data against a 24-hour clock, so we need a way to track time. Solutions are:

- Software-based clock: doesn't add anything in terms of size, but least accurate.
- Crystal-based clock: does add size to the product, but most accurate.
- Hardware-based clock: in-between the two previous solutions.

2. Measuring position: the first type of data to be logged is position. To do that we will need to incorporate sensors into our PCB. Solutions are:

- Linear variable differential transformer: most accurate but biggest in size.
- GPS: least accurate but smallest in size.
- Hall-effect sensor: in-between the two previous solutions.

3. Measuring motion: the second type is motion. This sensor will record things like speed, acceleration, rotation, and vibration. Solutions are:

- Accelerometer: most expensive but most accurate.
- Shock sensor: cheapest but least accurate.

- Gyroscope: in-between the two previous solutions.

4. Retrieving data: after logging the data, our product must provide a way to have user analyze the data. Solutions will be types of peripherals, and they are:

- USB: most widely used and available in most devices, but may not have very environmental tolerance.
- CAN: widely used in the industry and has good environmental tolerance.
- SD: easiest to use since it doesn't require software or connecting the device itself, but has very bad environmental tolerance.

After considering the solutions, four concepts were created. These concepts are shown in Table 5.

Table 7: Concepts to be evaluated

A	B	C	D
Crystal	Software	Hardware	Software
Battery	External	External	Battery
Hall effect	Hall effect	GPS	LVDT
Accelerometer	Shock	Gyroscope	Accelerometer
CAN	CAN	USB	USB

2.3.3 Concept Evaluation

A scoring table (as shown in Table 6) was used to evaluate the four concepts. Criteria were set and weighted, then each concept was rated. The ratings were summed up according to their weight, and concept A was chosen as the most suitable one.

Table 8: Scoring of the four concepts

Selection Criteria	**Weight**	**Concept Variant**							
		A		**B**		**C**		**D**	
		Rating	**Score**	**Rating**	**Score**	**Rating**	**Score**	**Rating**	**Score**
Dimensions	25%	4	0.8	5	1.0	3	0.8	3	0.8
Operation Time	20%	4	0.8	2	0.4	2	0.4	3	0.6
Acquisition Frequency	10%	4	0.4	2	0.2	2	0.2	3	0.3
Memory Capacity	10%	4	0.4	4	0.4	4	0.4	4	0.4
Connectivity	25%	4	1.0	4	1.0	5	1.3	5	1.3
Cost	10%	4	0.4	3	0.3	3	0.3	3	0.3
Total Score		4		3.55		3.3		3.8	
Rank		1		3		4		2	
Continue		Yes		No		No		No	

Chapter 3: Detailed System Design and Modeling

3.1 System Design and Modeling 25
3.1.1 Programming sub-system 25
3.1.2 PCB sub-system 26
3.2 Simulation Results 29
3.2.1 Programming sub-system 29
3.2.2 PCB sub-system 29
3.3 Component list 30

3.1 System Design and Modeling

3.1.1 Programming sub-system

The programming sub-system contains three main functions: time tracking with a real-time clock, receiving accelerometer data then saving in memory, and communicating with a PC via UART. The logic behind each part is as follows:

1. Time tracking with a real-time clock: we designed an interrupt-based clock that goes into the interrupt service routine every second and goes into an if-statement. "Second" is specified by the oscillator frequency from the processor. Below is a snippet of the interrupt service routine.

```
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt( void )
{
    if ( seconds < 59 )          // is cummulative seconds < 59?
    {
            seconds++;
                                // yes, so increment seconds
                 (LED3=!LED3);
    }
    else                         // else seconds => 59
    {
            seconds = 0x00;          // reset seconds
            if ( minutes < 59 )      // is cummulative minutes < 59?
            {
                    minutes++;
                           (LED2=!LED2); (LED3=!LED3);
            else                     // else minutes => 59
            {
                    minutes = 0x00;      // reset minutes
                    if ( hours < 23 )    // is cummulative hours < 23
                         {
                           hours++;
                                 (LED1=!LED1);(LED2=!LED2); (LED3=!LED3);
                    }
                    else
                    {
                            hours = 0x00;    // reset time
                    }
            }
    }
```

2. Receiving and saving accelerometer data:

```
int Saving(void){
while(s<DATA_BUFFER && signal==0){
            s++;
            hours_buffer[s]=hours;
            minutes_buffer[s]=minutes;
            seconds_buffer[s]=seconds;
        AD2CON1bits.SAMP = 1;   // start sampling ...
        DelayNmSec(100);                          //  for 10 mS
          AD2CON1bits.SAMP = 0;         //start Converting
      while (!IFS1bits.AD2IF);{
      for (i=0; i<2; i++){
       ADCValueZ[s] = (ADC2BUF0-1880)*1.06;
              AD1CON1bits.SAMP = 1;
              DelayNmSec(10);   //Sampling Frequency
              if (i==0){
              AD1CON1bits.SAMP = 0; //start Converting
                while (!IFS0bits.AD1IF);
               ADCValueY[s]=(ADC1BUF0-1930)*1.1425;}
                  else if(i==1){
                    AD1CON1bits.SAMP = 0; //start Converting
                  while (!IFS0bits.AD1IF);
                  ADCValueX[s]=(ADC1BUF0-0x7B1)*1.177;}}}
}}
```

3. Communicating via UART: using the RS232 port, we have functions that send the sampled data to the PC, as well as another function to receive the “Send data” command from the PC.

The full code can be found in Appendix D.

3.1.2 PCB sub-system

We have designed a PCB using the necessary components for our data logger to work. We have used CadSoft EAGLE for the schematic and routing. Figure 6 shows the schematic of the designed parts.

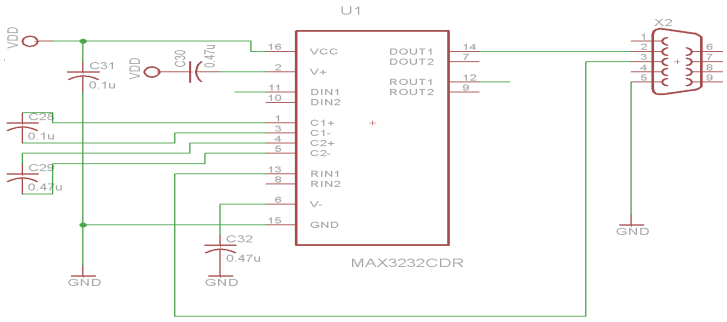

Figure 6a: serial port

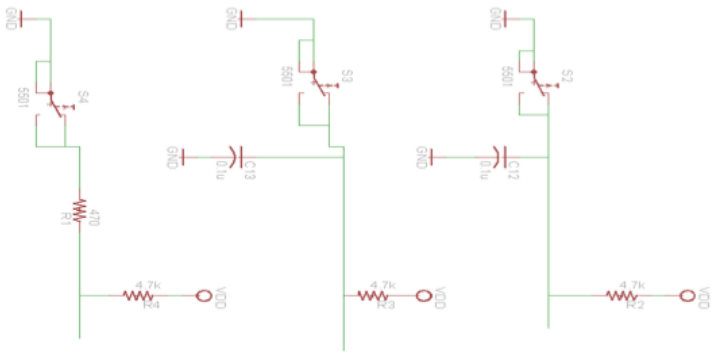

Figure 6b: Push buttons

Figure 6c: programming connector

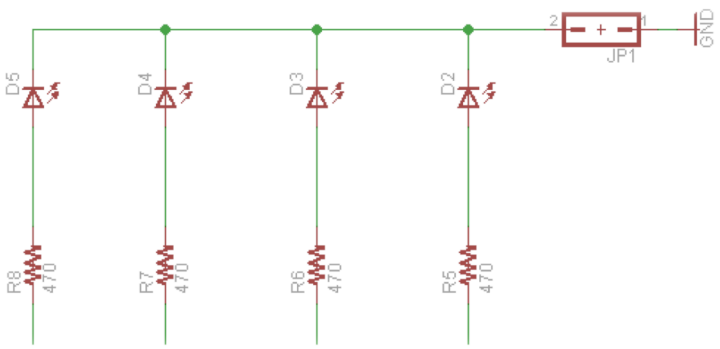

Figure 6d: LEDs

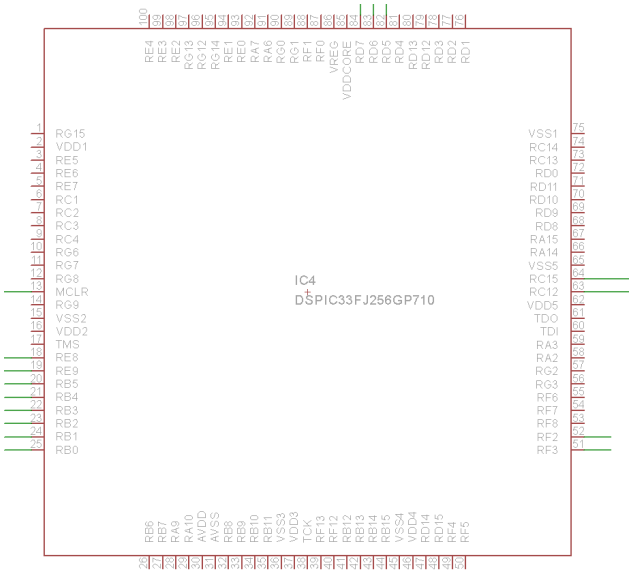

Figure 6e: the dsPIC33FJ256GP710a processor

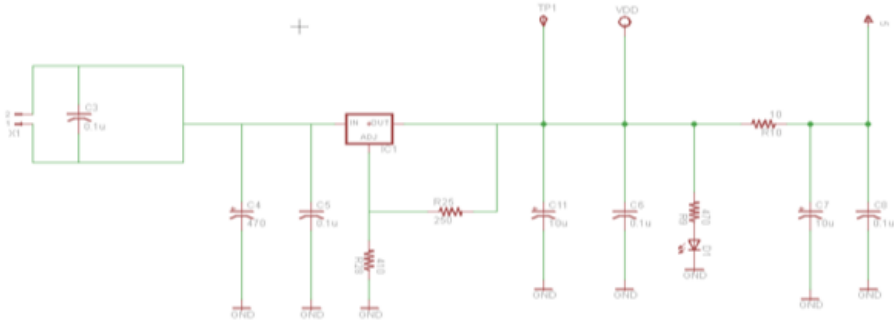

Figure 6f: power supply

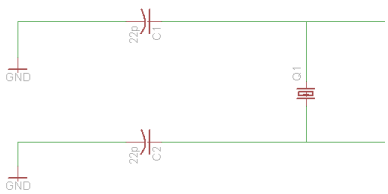

Figure 6g: Oscillator

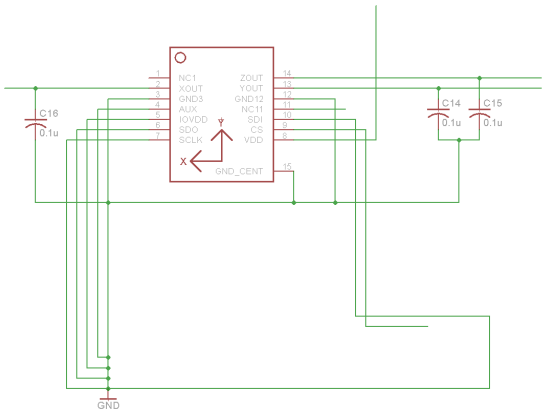

Figure 7h: accelerometer

After designing the schematic, we used CadSoft EAGLE for routing the PCB and finalizing it. Figure 8 shows the routed PCB.

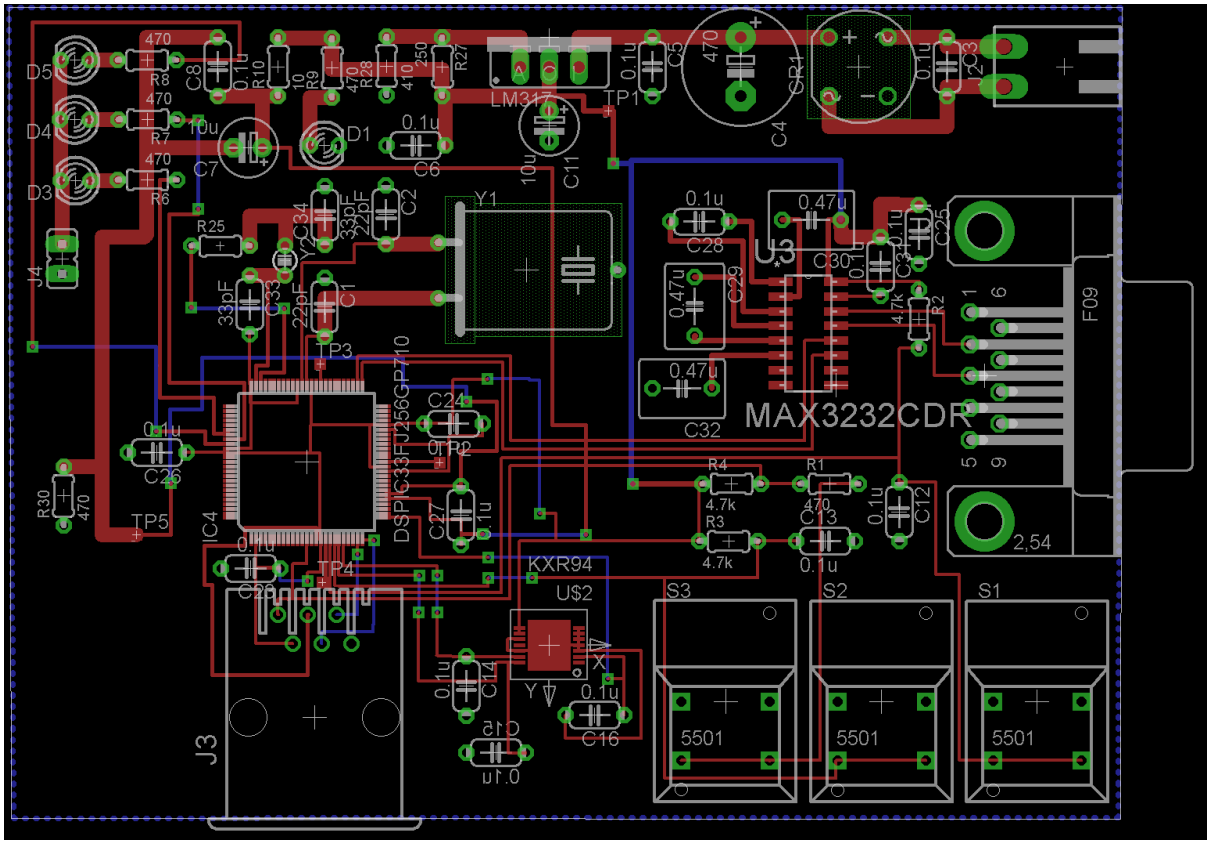

Figure 8: the PCB after routing

3.2 Simulation Results

3.2.1 Programming sub-system

The first thing we simulated was the real-time clock functionality. We used the LEDs in the development board to check if the internal variables *hours*, *minutes*, and *seconds* are changing within the function. Each variable is assigned an LED, and the LED toggles when its respective variable changes. Figure 9 shows a shot of the board right after the variables seconds and minutes changed.

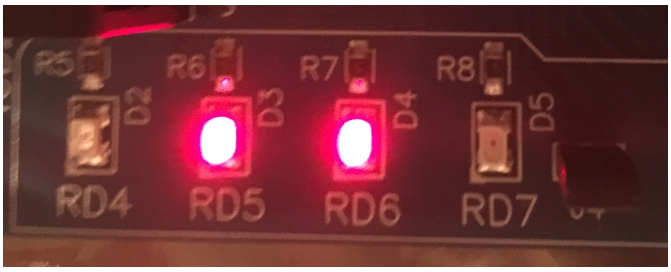

Figure 9: LEDs RD5 and RD6 flashing after the variables *minutes* and *seconds* changed.

The second simulation for the programming sub-system was trying to read the output of the UART connection. We connected the board to a PC using RS232 to serial connection, and sent out a string containing the XYZ data from the accelerometer as well as the time. The string was viewed using HyperTerminal. Figure 10 shows a screenshot form HyperTerminal.

```
X:+003 Y:+087 Z:-088 05:12:22
X:+002 Y:+087 Z:-086 05:12:23
X:+003 Y:+087 Z:-086 05:12:24
```

Figure 10: the string sent through the RS232 port

3.2.1 PCB sub-system

We used NI Multisim to simulate the power circuit in the PCB. We are using a 9 V input and then getting only 3.3 V out. The 9 V input was chosen for convenience purposes as it is easy to find 9 V batteries in any supermarket. The 3.3 V is the required voltage for our microcontroller. We are only using a voltage regulator which is very inefficient, but are looking to improve it in the future. A snapshot of the simulation is in Figure 10.

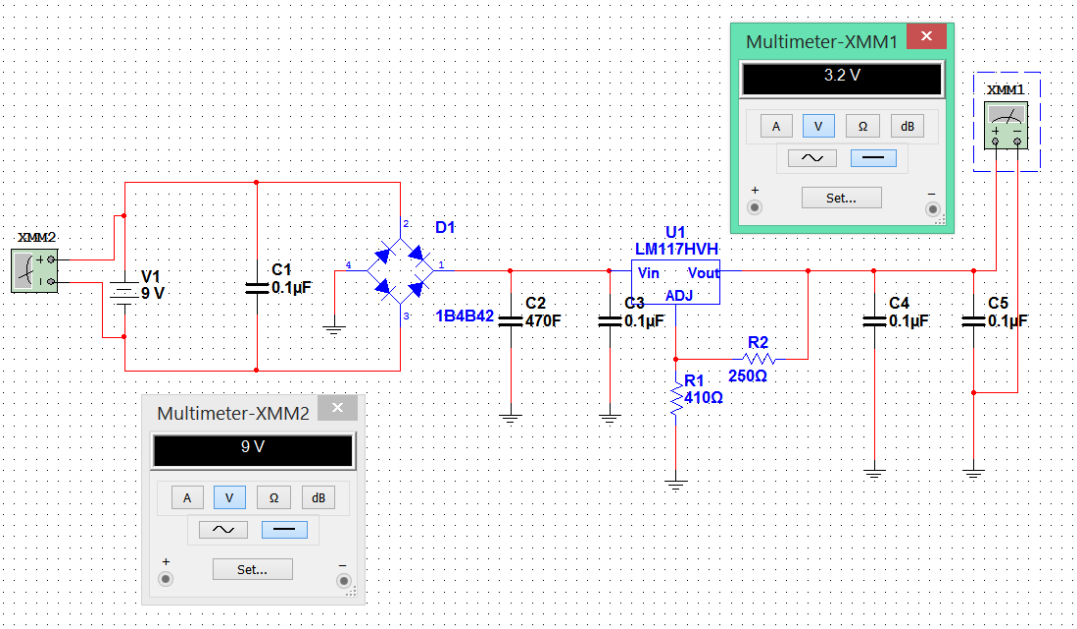

Figure 11: simulation of the power circuit

3.3 Components and Budget

Table 9 shows the component list for out project. This one was done after ordering the development board in the fall semester to test our codes.

Table 9: Components used in the project along with the budget

Parts Description[4]	Quantity	Unit Price	Total
dsPIC33FJ256GP710a processor	1	$9.37	$9.37
Terminal blocks	7	$12.93	$90.51
Crystal Oscillators	2	$0.36	$0.72
Push buttons	3	$5.11	$15.33
LEDs	6	$0.19	$1.14
RS232 chip	2	$2.34	$4.68
Serial port	2	$2.44	$4.88
Connectors	2	$0.93	$1.86
Full-bridge rectifier	2	$0.69	$1.38
Capacitors (22p, 33p,0.1u, 0.47 u, 10u, 470u)	40	N/A	$355.48
Resistors (10, 100, 250, 410, 470, 4.7k)	28	N/A	$42.00
KXD94 and KXR94 accelerometer chips	4	$9.87	$6.00
QSTB40 diode	2	$0.90	$3.00
		Total	$536.35

Chapter 4: Experimental Results

4.1 Testing and Troubleshooting.....32
4.2 Progress.....33

4.1 Testing and Troubleshooting

In order to test our code, we used LabVIEW to view and plot the output of the RS232. This is the final goal of our project, which is to show user-friendly data of our readings in LabVIEW. We are currently using to it troubleshoot the readings of the accelerometer. Figure 12 shows screenshots of the current state of our VI.

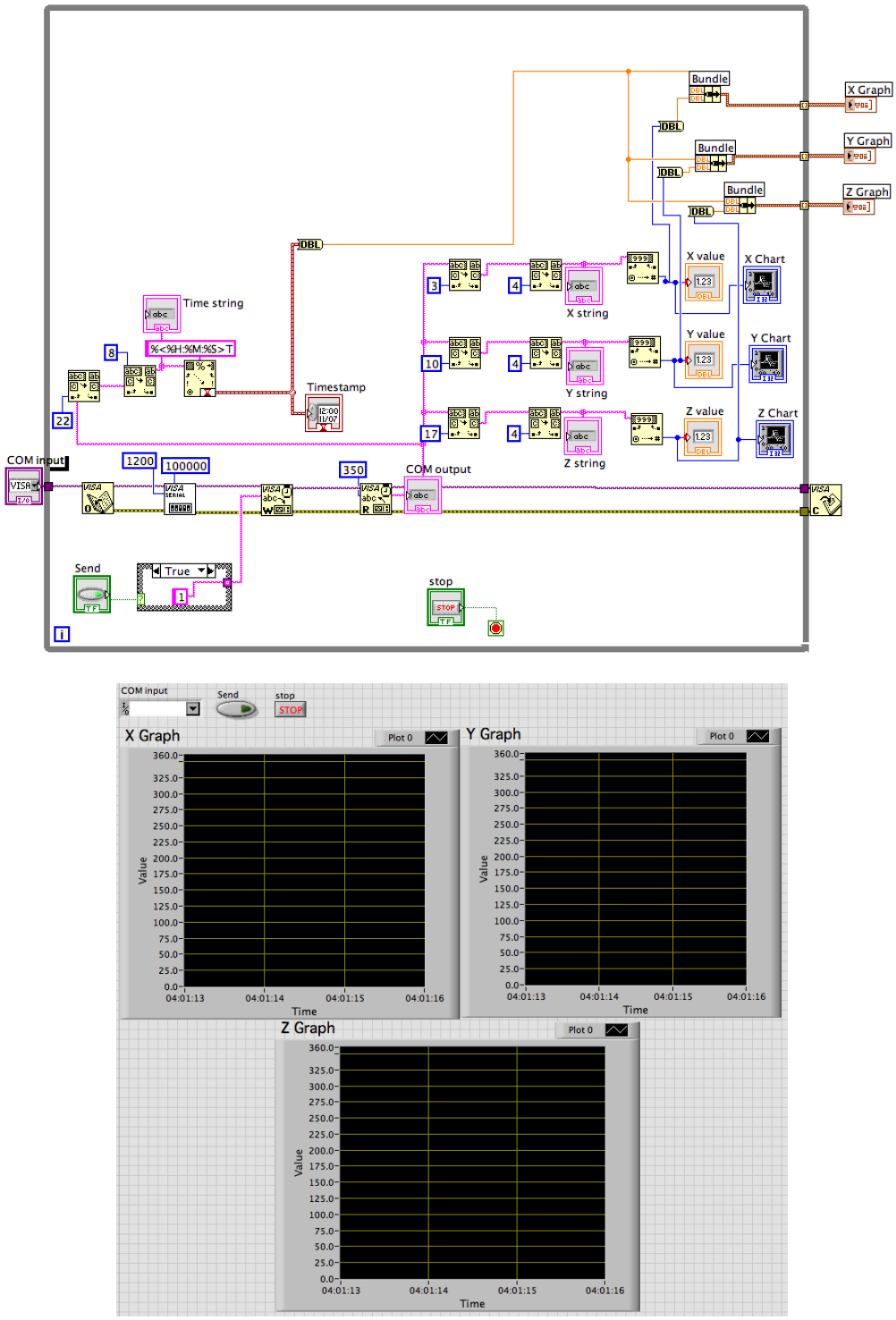

Figure 11: LabVIEW VI to show the output of our project to the user

4.2 Progress

- For the programming sub-system, the main code practically done. Right now, we are currently in process of optimizing our code to send data quickly. We are also trying to get more accurate reading of our processor.
- For the PCB sub-system, we are done with the design and now it should be printed soon. After receiving the ordered components, we wil solder them and also design the plastic housing for the PCB.

Chapter 5: Conclusion

5.1 Discussion.....35
5.2 Demo Day Plans.....35

5.1 Discussion

We have made a lot of progress during this semester since the fall semester. We now have an almost complete project that is functioning the way it should be. We will still be working on improving our code further and further, as well as trying to think of ways to improve the efficiency of the PCB. Ideas we have include programming different sensors, having more memory, designing a more efficient power supply, and increasing the operation time. We may or may not be able to finish these things during this semester, but we will still have the main functionality in our first prototype.

5.2 Demo Day Plans

For Demo Day, we plan on having a certain set-up. The PCB will be inside the plastic housing, and it will be powered by battery. We will have a visible clock so that the tester can move our product in certain ways while looking at the time. The tester then will see what he did on the LabVIEW panel and it will show his movements plotted against time.

References

[1] Retrieved from http://www.ti.com/tool/heatevm

[2] Retrieved from http://www.ni.com/data_logger/modular.htm

[3] Retrieved from http://www.cik-solutions.com/en/catalog/monitoring-datalogger/msr/products/

[4] Retrieved from http://uk.mouser.com/

Appendices

Appendix A: Interview Script

We are conducting a survey on our electrical senior design project, the data logger for mechanical systems. It is a harsh-environment electronic circuit to be added to passive mechanical systems for data logging capabilities. It will record the system's variables for later retrieval by the user. It could help in early failure detection and reduction in repair costs.

The mechanical systems that would benefit from such project are passive mechanical tools. Examples are drilling, downhole, or motorized tools.

1. After hearing our design proposal can you think of mechanical system that fits the description, please describe its functions?
2. What kind of measurements would benefit such tool, and why?
3. What kind of environment, does the tool operate at? In terms of temperature, pressure, other factors that affect the measuring accuracy?
4. Do you have any comments/suggestions/ideas to improve our design?

Appendix B: Survey Questions

Data Logger for a Mechanical System: Bringing Passive Tools to Use

This is a survey to gather customer ideas about our senior design project, the Data Logger for a Mechanical System. It is a harsh-environment electronic circuit to passive mechanical systems for data logging capabilities. It will record the system's variables for later retrieval by the user, for early failure detection and reduction in repair costs.

* Required

Do you work for the oil and gas industry? *

- ○ Yes
- ○ No

Are you familiar with/have you ever worked with mechanical systems? *

Mechanical systems are any machinery or tool that carries out various tasks such as motors, drilling tools, downhole tools etc.

- ○ Yes
- ○ No

If yes, please elaborate on the function of that/those mechanical system(s).

What type of data could our data logger collect in order to help improve your tool?

Choose all that apply.

- ☐ Linear motion
- ☐ Rotation
- ☐ Vibrations
- ☐ Temperature
- ☐ Other:

What kind of environment does your mechanical system operate in in terms of a) temperature?

- ○ Below 0° C
- ○ 0 to 50° C
- ○ 50 to 100° C
- ○ More than 100° C

What kind of environment does your mechanical system operate in in terms of b) depth?

- ○ Surface
- ○ 0 to 500 ft
- ○ More than 500 ft

What kind of environment does your mechanical system operate in in terms of c) materials around it?

Choose all that apply.

- ☐ Water
- ☐ Mud
- ☐ Rocks
- ☐ Other:

What kind of improvements would you suggest to make our data logger better for you?

Appendix C: Team Agreement

Project Title: Data-Logger for a Mechanical System

Mentor: Dr. Shehab Ahmed

Members

- Abdulrahman Al-Malki: Team Leader
- Faisal Al-Mutawa: Assistant Leader
- Mohammed Alsooj: Documents/Deadlines Coordinator
- Yasmin Hussien: Website Administrator

Team vision: Our goal is to create a professional environment for our project with a view to preparing ourselves to work in the industry. It is also crucial for us to work as hard as we can to deliver the best possible result for our project.

Roles and Responsibilities: All team members will be involved in the design and implementation of the project. In addition, below are the specialized "extra" roles for each team member.

- **Abdulrahman Al-Malki:**
  - General supervisor
  - Meeting coordinator and transcriber
  - Technical writer
- **Faisal Al-Mutawa:**
  - Public relations person
  - Simulation specialist
  - Ordering official
- **Mohammed Alsooj:**
  - Cameraman
  - Presentations organizer
  - Video editor
- **Yasmin Hussien:**
  - Website manager
  - General Designer
  - Interviewer

Commitments

- Tasks must be distributed among team members equally and fairly.
- Members must be responsible for the completion of their tasks on a timely manner.
- Members must update their partners about the progress of their work.
- Members must attend all meetings, and notify other members in case of emergencies.
- Members must be honest about their results.

Communication

- Seek first to understand, then to be understood (7 habits)
- Means of communication**:**
  - The meeting coordinator is responsible for the group managing emails, the Dropbox folder, and the WhatsApp group.
  - Every member must participate and check emails and messages regularly.
  - All emails must be cc'd to all group members.
  - All files and documents must be available to all members on Dropbox.
- Meetings
  - Meetings will be held on Saturdays from 4-6pm or as needed.
  - Location will be assigned beforehand and agreed upon by the team members.
  - Individual attendance, punctuality, and participation are expected in all meetings.
  - Tasks must be distributed equally and all members have to cooperate throughout the meeting.
  - Idea from all team members must be encouraged and included.
  - Meeting transcriber will take notes of all the meetings.

Website

- All submission must be uploaded to the website
- All team members must have access to edit/change/upload to the website.
- The site must be up to date and have all information about the project.
- Team members are responsible for uploading their respective tasks to the site.

Signatures

We hereby acknowledge that we have read and understood the team agreement, and that we agree with it in its entirety. We also acknowledge that we will abide by the Aggie Honor Code throughout this project.

"An Aggie does not lie, cheat or steal, or tolerate those who do."

Abdulrahman Al-Malki: _____

Faisal Al-Mutawa: _____

Mohammed Alsooj: _____

Yasmin Hussien: _____

Mentor – Dr. Shehab Ahmad: _____

Appendix D: Full code

```
#include "p33FJ256GP710A.h"
#include "math.h"
#include "stdio.h"
#include <xc.h>
#include <stdint.h>
#include "../../../../../../ALMALKI/Local SDP/Resources/CE113_Timer1_RTC/h/common.h"


// Configuration Routine
#pragma config FNOSC =PRIPLL
#pragma config POSCMD = XT
#pragma config FWDTEN = OFF            // Watchdog Timer disable
#pragma config FCKSM = CSECMD          // Clock Switching and Monitor
#pragma config FPWRT = PWR128
#define BAUD 9600
#define FCY  7327800
#define MILLISEC FCY/10000                    // 1 mSec delay constant
//------------------------------------------------------------------

//Assigning LEDs and Switch buttons to their PINS
#define LED1  PORTDbits.RD4   // LED connected to RD4
#define LED2  PORTDbits.RD5   // LED connected to RD5
#define LED3  PORTDbits.RD6   // LED connected to RD6
#define LED4  PORTDbits.RD7   // LED connected to RD6
#define S1    PORTAbits.RA12    // switch S1
#define S2    PORTAbits.RA13    // switch S2


// List interrupt and subroutine prototypes
void InitUART1(void);
void InitADC12(void);
void SendADC(void);
void DelayNmSec(unsigned int N);
void __attribute__((__interrupt__)) _U1TXInterrupt(void);
void __attribute__ ((interrupt, no_auto_psv)) _INT1Interrupt( void );
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt( void );
void Init_INTpin( void );
void Init_Timer1( void );
void SetupPorts(void);

// Flags assignment for LEDs interrupt (Debugging Purposes)
struct {
        unsigned                  LED1on:1;
        unsigned                  LED2on:1;
} Flags;


// ASCII buffers for serial data
#define BUFFERSIZE 28     // OutData buffer size
#define TxBOffset 2         // transmit buffer offset
#define DATA_BUFFER 2000  // Data saving buffer

// Variables and pointers assignments
unsigned char    OutData[]={"X:000 Y:000 Z:000 00:00:00\n\r"};
char c;
int decimal,j, rem;
int k;
int i;
int s;
int send;
int store;
int signal;
unsigned char TxCount;
unsigned char ADCbcd, ADCbcd1, ADCbcd2;
unsigned char ADCbcdx, ADCbcdxx, ADCbcdxxx, ADCbcdx1;
unsigned char ADCbcdy, ADCbcdyy, ADCbcdyyy;
unsigned char ADCbcdz, ADCbcdzz, ADCbcdzzz;
unsigned char hours2;
unsigned char minutes2;
unsigned char seconds2;
unsigned char seconds_unit;
unsigned char seconds_ten;
unsigned char minutes_unit;
```

```
unsigned char minutes_ten;
unsigned char hours_unit;
unsigned char hours_ten;

// List of Arrays to be used to save X,Y and Z Readings from the ADC and capture time
unsigned int __attribute__((__psv__)) ADCValueX[DATA_BUFFER];
unsigned int __attribute__((__psv__)) ADCValueY[DATA_BUFFER];
unsigned int __attribute__((__psv__)) ADCValueZ[DATA_BUFFER];
unsigned int __attribute__((__psv__)) hours_buffer[DATA_BUFFER];
unsigned int __attribute__((__psv__)) minutes_buffer[DATA_BUFFER];
unsigned int __attribute__((__psv__)) seconds_buffer[DATA_BUFFER];

// RTC  variables list
volatile unsigned char hours;
volatile unsigned char minutes;
volatile unsigned char seconds;
//---------------------------------------------------------------------

int main(void)
{

SetupPorts();
CLKDIVbits.PLLPOST=0;
CLKDIVbits.PLLPRE=0;
OSCTUN=0;
CLKDIV = 0x0000;      // Divide clock = 1
PLLFBD = 0x0002;      // PLL feedback = N+2 i.e. PLLX4

// Clock Switch to incorporate PLL
       __builtin_write_OSCCONH(0x01);               // Initiate Clock Switch to FRC with PLL
(NOSC=0b001)
       __builtin_write_OSCCONL(0x01);               // Start clock switching
       while (OSCCONbits.COSC!= 0b001);      // Wait for Clock switch to occur
       while(OSCCONbits.LOCK!=1) {};     // Wait for PLL to lock

       /* Initialize The clock */
        hours=19;
        minutes=05;
        seconds=00;

       /* Initialize Timer 1 for 32KHz real-time clock operation */
   Init_Timer1();
   /* Initialize INT1 pin used for setting Time-of-Day */
   Init_INTpin();


InitUART1();                          // Initialize UART1 For Send/Recieve operation
InitADC12();                          // Initialize ADC1 and ADC2 as 12 bit ADCs

IEC0bits.U1RXIE = 1;    //Enable UART receive Interrupt
U1MODEbits.UARTEN = 1;  //Enable UART RX

Saving();

while(1){

              if (signal==1)                                 // UART RX interrupt recieved
              {
                  for(k=0;k<=s;k++){SendADC();} //Then send ADC through UART
                  signal=0;

               }
               DelayNmSec(5000); //wait
}
}



int Saving(void){

while(s<DATA_BUFFER && signal==0){

            s++;
            hours_buffer[s]=hours;
            minutes_buffer[s]=minutes;
            seconds_buffer[s]=seconds;

        AD2CON1bits.SAMP = 1;   // start sampling ...
        DelayNmSec(100);                              //  for 10 mS
```

```
            AD2CON1bits.SAMP = 0;         //start Converting
      while (!IFS1bits.AD2IF);{
      for (i=0; i<2; i++){
       ADCValueZ[s] = (ADC2BUF0-1880)*1.06;
              AD1CON1bits.SAMP = 1;
              DelayNmSec(10);   //Sampling Frequency
              if (i==0){
              AD1CON1bits.SAMP = 0; //start Converting
                while (!IFS0bits.AD1IF);
               ADCValueY[s]=(ADC1BUF0-1930)*1.1425;
                  }
                  else if(i==1){
                    AD1CON1bits.SAMP = 0; //start Converting
                  while (!IFS0bits.AD1IF);
                  ADCValueX[s]=(ADC1BUF0-0x7B1)*1.177;
            }

            }
                              }
}}


void __attribute__((__interrupt__)) _U1TXInterrupt(void)
{
       IFS0bits.U1TXIF = 0;                 // clear TX interrupt flag

  U1TXREG = (int)OutData[TxCount++];   // Write a single reading to UART

  if (TxCount == BUFFERSIZE)
     IEC0bits.U1TXIE = 0;               // Disable Transmit Interrupts since all buffer values
are Xmitted
}
//---------------------------------------------------------------------

//UART Recieve Interrupt Routine
void __attribute__((__interrupt__)) _U1RXInterrupt(void)
{
     IFS0bits.U1RXIF = 0;        // Clear the Rx Interrupt Flag
     signal= U1RXREG;
     signal=1;
     LED4=1;
}


// initialize the UART1 for BAUD = 1200
// 8 bits, 1 start and 1 stop

void InitUART1(void)
{
U1MODE = 0x8000;
U1STA = 0x0000;
U1BRG = ((FCY/16)/BAUD) - 1;
INTCON1bits.NSTDIS = 1;              // disable nested interrupts
}

// Initialize ADC1 and ADC2 For 12bit operation
// Manual Start/Stop conversion
// Two channels for ADC1(X,Y) and a single channel for ADC2(Z)
void InitADC12(void)
{
//AD1PCFGH =0xFFFF;
AD1PCFGL = 0xFFAF;
AD2PCFGL = 0xFFDF;
AD1CON1 = 0x0400;
AD1CON1bits.ASAM=0;
AD1CON2bits.ALTS=1;
AD1CON1bits.SIMSAM=0;
AD1CON2bits.CHPS=0;
AD1CON2bits.SMPI=3;
AD2CON1 = 0x0400;     // set for 12-bit ADC operation
AD1CHS0 = 4;
AD1CHS0bits.CH0SB=6;
AD2CHS0 = 5;
AD1CSSL = 0;
AD2CSSL = 0;
AD1CON3 = 0x0002;
AD2CON3 = 0x0002; /
AD2CON2 = 0;
AD1CON1bits.ADON = 1;
```

```
AD2CON1bits.ADON = 1; // turn ADC ON
}


void SendADC(void)

{

        while (!U1STAbits.TRMT);      // wait till TSR is empty
        U1STAbits.UTXEN = 0;                  // disable tranmission

         ADCbcdxxx=ADCValueX[k]/1000;
        OutData[TxBOffset   ] = '0'+ ADCbcdxxx;
         ADCbcdxx=((ADCValueX[k]%1000)%100)/10;
        OutData[TxBOffset+ 1] = '0'+ ADCbcdxx;
         ADCbcdx=((ADCValueX[k]%1000)%100)%10;
        OutData[TxBOffset+ 2] = '0'+ ADCbcdx;


         ADCbcdyyy=(ADCValueY[k]%1000)/100;
        OutData[TxBOffset+ 6] = '0'+ ADCbcdyyy;
         ADCbcdyy=((ADCValueY[k]%1000)%100)/10;
        OutData[TxBOffset+ 7] = '0'+ ADCbcdyy;
         ADCbcdy=((ADCValueY[k]%1000)%100)%10;
        OutData[TxBOffset+ 8] = '0'+ ADCbcdy;

         //ADCValueZ[k]= (ADCValueZ[k]-0x776)*1.0588;
         ADCbcdzzz=(ADCValueZ[k]%1000)/100;
        OutData[TxBOffset+ 12] = '0'+ ADCbcdzzz;
         ADCbcdzz=((ADCValueZ[k]%1000)%100)/10;
        OutData[TxBOffset+ 13] = '0'+ ADCbcdzz;
         ADCbcdz=((ADCValueZ[k]%1000)%100)%10;
        OutData[TxBOffset+ 14] = '0'+ ADCbcdz;

       hours2 = (char)(hours_buffer[k]);
       hours_unit=hours2%10;
       hours_ten=hours2/10;
        OutData[TxBOffset +16 ] = '0'+ hours_ten;
    OutData[TxBOffset + 17] = '0'+ hours_unit;

       minutes2 = (char)(minutes_buffer[k]);
       minutes_unit=minutes2%10;
       minutes_ten=minutes2/10;
        OutData[TxBOffset + 19] = '0'+ minutes_ten;
    OutData[TxBOffset + 20] = '0'+ minutes_unit;

       seconds2 = (char)(seconds_buffer[k]);
       seconds_unit=seconds2%10;
       seconds_ten=seconds2/10;
        OutData[TxBOffset + 22] = '0'+ seconds_ten;
    OutData[TxBOffset + 23] = '0'+ seconds_unit;


  TxCount = 0;
  IFS0bits.T1IF = 0;                  // clear interrupt flag
        TxCount = 0;                                // reset Tx Buffer count
        IFS0bits.U1TXIF = 0;    // clear TX interrupt flag
        IEC0bits.U1TXIE = 1;          // enable interrupt
        U1STA = 0x0000;                                 // clear all status bits
        U1STAbits.UTXEN = 1;          // Initiate Transmission
}

//Extra Function - Not used
int binary_decimal(int n) /* Function to convert binary to decimal.*/

{
   int decimal=0, i=0, rem;
   while (n!=0)
   {
       rem = n%10;
       n/=10;
       decimal += rem*pow(2,i);
       ++i;
   }
   return decimal;
}
```

```
//Generic Function that create a 1-second Delay
void DelayNmSec(unsigned int N)
{
unsigned int j;
while(N--)
        for(j=0;j < MILLISEC;j++);
}

//RTC Interrupt routine
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt( void )
{
 //if (Flags.LED1on){
 //     send=1;}
 //if (Flags.LED2on){
 //      store=1;}

        if ( seconds < 59 )          // is cummulative seconds < 59?
        {
                seconds++;
                                // yes, so increment seconds

                 (LED3=!LED3);

        }
        else                         // else seconds => 59
        {
                seconds = 0x00;          // reset seconds
                if ( minutes < 59 )      // is cummulative minutes < 59?
                {
                        minutes++;
                           (LED2=!LED2); (LED3=!LED3);


                else                     // else minutes => 59
                {
                        minutes = 0x00;      // reset minutes
                        if ( hours < 23 )    // is cummulative hours < 23
                          {
                                hours++;
                                  (LED1=!LED1);(LED2=!LED2); (LED3=!LED3);
                        }
                        else
                        {
                                hours = 0x00;    // reset time
                        }
                }
        }

        IFS0bits.T1IF = 0;
}
```

***The following code is a modified version of example codes from Microchip.**

```
#include <p33FJ256GP710A.h>
#include "../../../../../../ALMALKI/Local SDP/Resources/CE113_Timer1_RTC/h/common.h"


#define LED1   PORTDbits.RD4          // LED connected to RD4
#define LED2   PORTDbits.RD5          // LED connected to RD5
#define LED3   PORTDbits.RD6          // LED connected to RD6
#define LED4   PORTDbits.RD7          // LED connected to RD6



/*-------------------------------------------------------------------
  Function Name: Init_INTpin
  Description:   Initialize INT1 pin used for setting RTC TOD
  Inputs:        None
  Returns:       None
-------------------------------------------------------------------*/
void Init_INTpin( void )
{
        /* set INT1 pin interrupts for negative edge */

//      /* ensure upper PORTA pins are input (for dsPICDEM 1.1 board) */
         TRISA |= 0xFFFF;
```

```
}


extern void enSecOsc();

/*---------------------------------------------------------------------
  Function Name: Init_Timer1
  Description:   Initialize Timer1 for 1 second intervals
  Inputs:        None
  Returns:       None
---------------------------------------------------------------------*/
void Init_Timer1( void )
{

    T1CON = 0;              // Timer reset
        IFS0bits.T1IF = 0;      // Reset Timer1 interrupt flag
        IPC0bits.T1IP = 4;      // Timer1 Interrupt priority level=4
        IEC0bits.T1IE = 1;      // Enable Timer1 interrupt

        PR1 = 0x8000;           // Timer1 period register = 32768
        T1CONbits.TCS = 1;      // Timer1 Clock= External

         enSecOsc();             // Enable Secondary Osc
        T1CONbits.TON = 1;      // Enable Timer1 and start the counter

}


/*---------------------------------------------------------------------
  Function Name: SetupPorts
  Description:   Configure LED
  Inputs:        None
  Returns:       None
---------------------------------------------------------------------*/

void SetupPorts(void)
{
PORTD = 0;
TRISD = 0xFF0F;                                 // Set RD7 to RD4 as outputs
AD1PCFGHbits.PCFG20 = 1;
AD1PCFGHbits.PCFG21 = 1;
TRISCbits.TRISC1=0;
PORTCbits.RC1=1;
LATC=0xFFFF;
}



void __attribute__((__interrupt__)) _OscillatorFail(void);
void __attribute__((__interrupt__)) _AddressError(void);
void __attribute__((__interrupt__)) _StackError(void);
void __attribute__((__interrupt__)) _MathError(void);
void __attribute__((__interrupt__)) _DMACError(void);

void __attribute__((__interrupt__)) _AltOscillatorFail(void);
void __attribute__((__interrupt__)) _AltAddressError(void);
void __attribute__((__interrupt__)) _AltStackError(void);
void __attribute__((__interrupt__)) _AltMathError(void);
void __attribute__((__interrupt__)) _AltDMACError(void);

/*
Primary Exception Vector handlers:
These routines are used if INTCON2bits.ALTIVT = 0.
All trap service routines in this file simply ensure that device
continuously executes code within the trap service routine. Users
may modify the basic framework provided here to suit to the needs
of their application.
*/
void __attribute__((interrupt, no_auto_psv)) _OscillatorFail(void)
{
        INTCON1bits.OSCFAIL = 0;        //Clear the trap flag
        while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AddressError(void)
{
        INTCON1bits.ADDRERR = 0;        //Clear the trap flag
        while (1);
```

```
}
void __attribute__((interrupt, no_auto_psv)) _StackError(void)
{
        INTCON1bits.STKERR = 0;         //Clear the trap flag
        while (1);
}

void __attribute__((interrupt, no_auto_psv)) _MathError(void)
{
        INTCON1bits.MATHERR = 0;        //Clear the trap flag
        while (1);
}

void __attribute__((interrupt, no_auto_psv)) _DMACError(void)
{
        INTCON1bits.DMACERR = 0;        //Clear the trap flag
        while (1);
}




/*
Alternate Exception Vector handlers:
These routines are used if INTCON2bits.ALTIVT = 1.
All trap service routines in this file simply ensure that device
continuously executes code within the trap service routine. Users
may modify the basic framework provided here to suit to the needs
of their application.
*/

void __attribute__((interrupt, no_auto_psv)) _AltOscillatorFail(void)
{
        INTCON1bits.OSCFAIL = 0;
        while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AltAddressError(void)
{
        INTCON1bits.ADDRERR = 0;
        while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AltStackError(void)
{
        INTCON1bits.STKERR = 0;
        while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AltMathError(void)
{
        INTCON1bits.MATHERR = 0;
        while (1);
}

void __attribute__((interrupt, no_auto_psv)) _AltDMACError(void)
{
        INTCON1bits.DMACERR = 0;        //Clear the trap flag
        while (1);
}
```