December 11, 2002

Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University
Burnaby, British Columbia
V5A 1S6

Subject: Post Mortem for ENSC 340: Wireless EMG Electrodes

Dear Dr. Rawicz:

The document enclosed with this letter, *Post Mortem for a Wireless EMG System*, is an outline of the technical and interpersonal experiences during ENSC 340. The goal of our project was to develop a system to sense, acquire and wirelessly transmit muscle activity data from a patient to a computer. Without wires limiting the patient's freedom of motion, this novel technology could revolutionize many aspects of rehabilitation, diagnosis and research.

In this document we present the current state of the device, deviations from the original specifications, and our future plans for the device. In addition, there is discussion on our budget, time management and personal experiences.

Wireless Medical Devices was formed in June of 2002 by four highly skilled and motivated Engineering Science students: Eric Chow, Aaron Ridinger, David Press, and Andrew Pruszynski. We look forward to hearing your comments on our proposal. Please feel free to contact us at wireless-medicaldevices@sfu.ca.

Sincerely,

Jedrzej (Andrew) Pruszynski
Chief Executive Officer
Wireless Medical Devices
Enclosure: Proposal for Wireless EMG Electrodes

Process Report for a Wireless EMG System

Project Team:	Eric Chow Dave Press Andrew Pruszynski Aaron Ridinger
Submitted To:	Dr. Andrew Rawicz Mr. Steve Whitmore
Revision Number:	1.0
Revision Date:	December 13, 2002

Table of Contents

Acronyms

CMNR Common Mode Noise Rejection
DAQ Data Acquisition
ECG Electrocardiogram
EEG Electroencephalogram
EMG Electromyograph
GUI Graphical User Interface
iEMG Independent Electromyograph
ISM Industrial, Scientific, and Medical
I/O Input/Output
MUX Multiplexer
NRZ Non-Return to Zero
PC Personal Computer
PCB Printed Circuit Board
PCI Peripheral Card Interface
RSSI Received Signal Strength Indicator
TGE Transmitter and Ground Electrode
WMD Wireless Medical Devices

1.0 - Introduction

An Electromyograph is a recording of muscle activity used for rehabilitation, injury prevention and performance enhancement. Unfortunately, current systems rely on the use of restrictive wires and equipment that result in inaccurate and inconvenient diagnosis. Applying wireless communications principles to Electromyography (EMG) will lead directly to better diagnosis, research and rehabilitation. These advances have far-reaching implications for corporations, insurance agencies, athletes and patients.

Over the past 15 weeks, our group has been dedicated to producing a wireless EMG system that is capable of eliminating the restriction associated with current EMG products. Our product, iEMG, is shown in Figure 1.1.

Figure 1.1 – Our Wireless EMG System

We have faced an enormous challenge to design, implement and document an extremely complex system within tight budgetary and time constraints. Despite many long nights with difficult trials and tribulations, we have been able to present a system that meets every major technical objective we have set forth. It has been an exciting and very rewarding project, one that makes each of our team members extremely proud.

2.0 - Current Device

Figure 2.1 presents a block diagram of the system that we have produced in fulfillment of Engineering Science 340. We have produced a system that is capable of wirelessly acquiring muscle activity from four EMG pads simultaneously. Dr. Ted Milner, an expert EMG user, has confirmed these results.

Two EMG pads are associated with each TGE, which is responsible for providing ground and power to each system component, sampling the muscle activity data from the pads and transmitting the acquired data to the RS. The RS then receives the information, formulates the data word, and passes it to the PC. The software on the PC is then capable of displaying and recording the signals.

The following sections outline the current state of each sub-component in more detail.

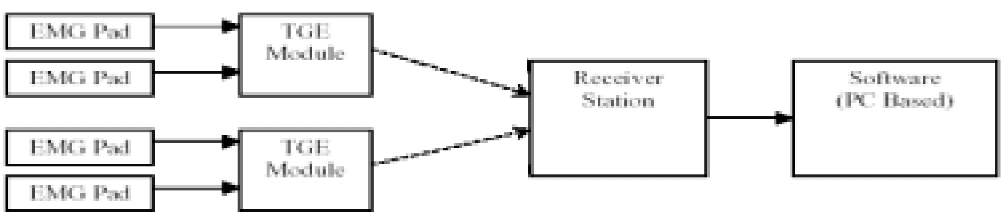

Figure 2.1 – Block Diagram of Wireless EMG System

2.1 – EMG Pad

Dr. Ted Milner of Kinesiology who has been manufacturing the devices for his research for several years provided the EMG pads. These pads are capable of acquiring the EMG data, amplifying to useful amplitudes and proving the signal at the output. We had initially stated in our Design Spec that we planned to run the pads off a +/- 3V supply, which we hoped to produce using a stack of four hearing aid batteries. We also knew that they needed a ground referenced to elsewhere on the subject’s body to yield correct operation. However, we discovered that these pads required an input of +/-5V (even though all the components in the pad should have been able to run off lower voltages, which we were never able to explain).

2.2 – TGE Module

Each TGE module is capable of sampling and transmitting the muscle activity signal from two EMG pads at a rate high enough to ensure that no data significant data is lost. A block diagram of the TGE is presented in Figure 2.2.

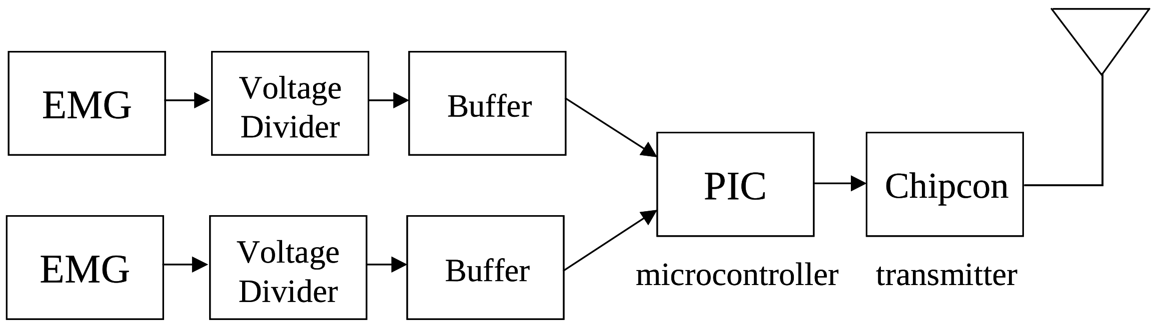

Figure 2.2 – Block Diagram of the TGE Module

TGE Hardware

As mentioned, we initially planned on using a stack of hearing aid batteries to power the TGE. However, after discovering our need a total of 10V, we decided that would simply require too many hearing aid batteries. Thus we decided to use a DC-DC voltage converter and a lower voltage battery. We decided on a 3V lithium cell, which could source enough current to run the EMG pads at over triple its own voltage, while still having sufficient voltage to run the Chipcon transmitter and PIC microcontroller (which require 2.7-3.3 Volts).

When we ran the power supply system as described above, we found that the voltage converter drew enormous current spikes from the battery, which caused a voltage swing of about 0.2 Volts in the battery's output. This proved to be too much variation for the PIC to run properly. Thus, we decided to draw the power for the PIC and Chipcon from the 10V supply through a 3.3V regulator. This was extremely power inefficient since the Chipcon draws more current than all of the EMG hardware combined, but we faced no other options short of purchasing more DC-DC converters to step the voltage down from 10V to 3.3V.

This method now allowed us to use batteries that supplied less than 3V (since the Maxim DC-DC converters we used accepted supply voltages as low as 1.8V). We then decided to produce one TGE running off the 3V lithium cells we had already bought, and another off of a pair of rechargeable AAA Ni-MH batteries. The complete TGE including two EMG pads drew 225 mA at a supply voltage of 2.8V, and this current requirement increased as the input voltage to the DC-DC converter decreased. We found that a single lithium cell could run the TGE for 6 hours continuously, while the 2 AAA's ran in for slightly over 2.5 hours.

Several hardware components were designed to adjust the EMG signal before sending it to the PIC Microcontroller for A/D conversion. The EMG signal varied between +10 and 0 volts, whereas the input to the PIC A/D converter needed to be between 3.3 and 0 volts. A simple 2-resistor voltage divider was added to vary the EMG signal between 3.3 and 0 volts, and a buffer was added before the PIC A/D input to prevent the converter from loading the circuit. The buffer was constructed using an OP-AMP in a negative feedback configuration. Initially a generic TLO72 opamp chip was used, but we found that the negative rail wasn't close enough to 0 volts, the negative power supply. To ensure the complete range down to zero volts, a rail-to-rail opamp was used.

We also faced some choices with regards to the transmitter antenna. After experimenting with various loops of wire, as well as integrated helical and loop antennas from Antenna Factor, we decided on a small helical antenna called the Antenna Factor JJB.

TGE Firmware

The firmware running on the TGE PIC is much the same as we had initially planned. Essentially, it samples the two A/D channels once per millisecond, and sends the samples to the Receiver Station.

We initially had hoped to use NRZ format data encoding, which allowed a maximum data rate of 76.8 kB/s. This allowed us to send every data sample twice, with a checksum, to reduce data errors. We also needed to send an identifier byte every millisecond before sending a set of data samples so the receiver would be able to synchronize to the incoming data stream. We found that errors in this start byte were much more catastrophic than in the data itself. Thus, the redundant data did little to help the overall sample error rate. The data rate wasn't high enough to send the samples twice in two separate packets with separate identifier bytes.

Thus, we decided to switch to Manchester encoding, which offered a lower data rate of 38.4 kB/s, but higher receiver sensitivity and thus fewer bit errors. Since each data sample was only sent once, there was no need for checksums, so we could just barely fit each packet into the allotted 1 millisecond time slot.

2.3 – Receiver Station

The RS is capable of receiving data from up two TGE modules, formulating the correct data structure and communicating this to the PC. A block diagram of the RS is presented in Figure 2.3.

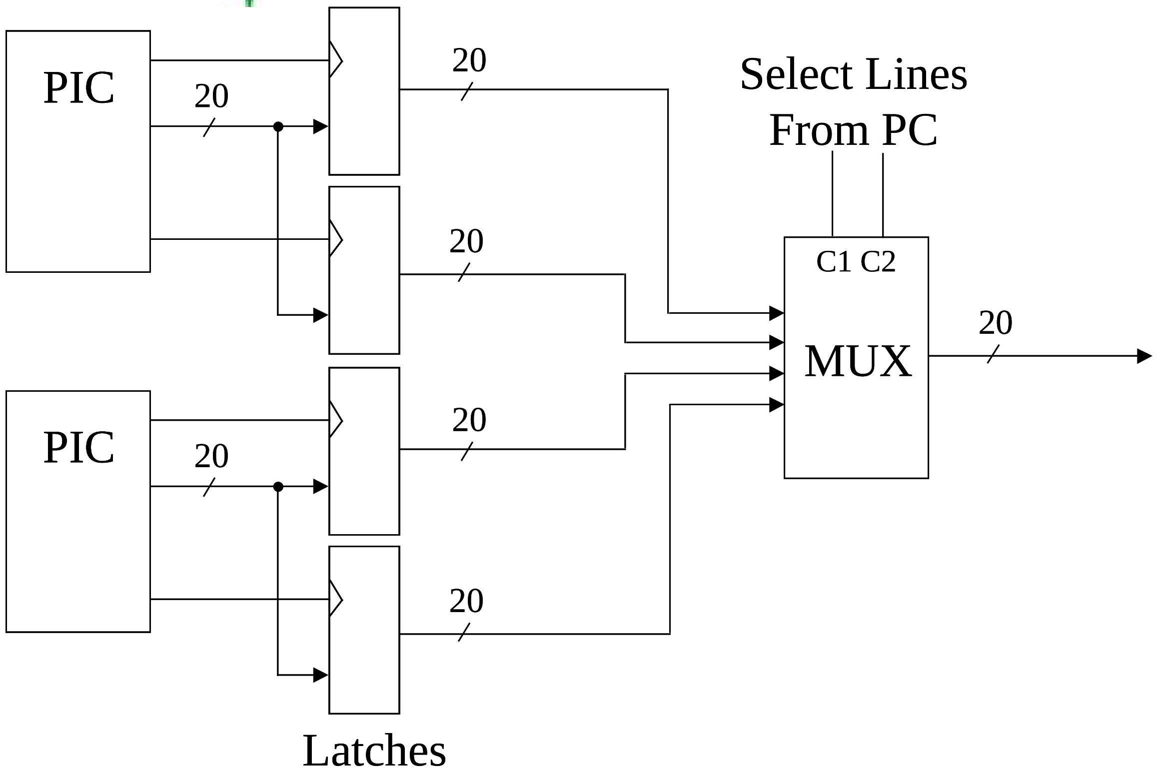

Figure 2.3 – Block Diagram of the Receiver Station

Receiver Firmware

The receiver station firmware is very similar to that described in the Design Specification. It syncronizes its USART to the incoming data by looking for the packet identifier byte. It then simply reads the incoming data, reads the A/D conversion from the Chipcon's RSSI pin, and forms the two output words (one for each sample. Each of these 20-bit data words is then latched into a set of D flip-flops. These latches ensure that the data is kept constant while the microcontroller switches between the two EMG Channels

One minor deviation from our original plans was the format of the output word, which is shown below in Figure 2.4. The incrementing ID bits are simply incremented by the PIC every time it updates the output word. This was necessary since the PC oversamples these output words, so it can simply compare the top nibble of every word it reads, and disregard the word if the incrementing ID is the same as the previous one from that particular EMG channel.

20-Bit Data		
4 Bits	**12 Bits**	**4 Bits**
Incrementing ID	EMG Data	Signal Strength

Figure 2.4 – Format of Output Word

The output of the latches is connected to multiplexing circuitry that is controlled by the PC DIO Card for use by the software.

We chose to build the RS using spaghetti board rather than design a PCB to save time and money. We felt that a multi-layer PCB would be more difficult to fix if we discovered problems, and when we change the design of the receiver station to handle more TGEs the PCB would have to be redesigned anyways.

The receive side microcontroller source code is presented in the Appendix.

2.4 – PC Software

The PC software serves three major purposes. First, it allows the user to graphically observe the incoming data in real-time. Secondly, the incoming data can be recorded for future analysis. Lastly, some Matlab code was written to allow the import and observation of the acquired data signal.

Real-time graphical observation allows the user to correctly setup the EMG electrodes by ensuring correct pad placement and amplification. In addition, the user is shown the signal strength indicator (via RSSI) from each of the transmitting TGEs. This visual representation of receive signal strength lets the operator decide if the incoming signal strength is enough to ensure data consistency.

After the system has been verified graphically and the user is satisfied with the experimental setup, the data acquisition feature can be used to save muscle activity data. When the user begins data acquisition, the system will record the data from each EMG pad into an individual binary data file. The code has been written in such a way that ensures that no data points are lost and that no data points are saved twice.

The Matlab code was written to take advantage of the pre-fabricated GUI interface. More importantly, the ability to import our data files into Matlab is extremely important for the end users who will most likely be performing their data analysis within this software environment. Since our sampling at the TGE is very synchronous, we can provide the user with an accurate time-scale for the data stream. In addition, we can guarantee that this time scale applies to all the EMG signals recorded in each trial. These are very important features for the user.

We initially planned on a windows based GUI application for our PC software but after looking into the requirements and talking with people who have done similar programs, we decided that for this project a windows based GUI would not work. One of the main reasons was to have a fast enough sampling time, we had to run the program in DOS because there was too much overhead for the API in windows. To be able to operate in windows, it is estimated that it would take an entire semester of work to write the appropriate device drivers and other software.

The source code for the C-based application software and the Matlab file for data analysis are included in the Appendix.

3.0 – Future Plans

The future for this product is extremely bright. It is ahead of the competition in terms of technology and could be produced for a much lower product price. In order to improve our market position and increase our technological advantage, there are several developments we are planning to make. The following sections outline these changes.

3.1 – More TGEs

Our most pressing issue is to expand the number of TGEs we currently have built. This does not require a redesign of our system, it simply requires manual labour and money to assemble more units. We would also have to expand the receiver station to handle the additional incoming data. The software is already designed to handle more TGE units. By making all the additional hardware, we will be capable of operating a maximum of 8 TGEs supporting up to 16 EMG pads without any significant system design changes.

3.2 – New Chipcon Transceiver

Chipcon, our supplier of wireless ISM band transceivers has just announced the release of a new transceiver. This product will be able to handle a higher data rate which would allow us to connect more EMG pads to a single TGE or send each sample twice for redundancy and error checking. The new transceiver is also designed specifically for narrow-band operation. This would allow us to fit more TGEs onto the 915MHz ISM band.

3.3 – Miniaturization

A critical body of work remains to place the design onto a PCB. We decided to avoid this for the purpose of this project to maximize our ability to debug the product and save time and money. Designing a PCB would have many significant effects on the product. Most importantly it would reduce the size and weight of the product. We estimate that a final PCB of the TGE would create a final size of 1.5” by 1.5”. A design for the RS would also create a more compact and reliable package. Product miniaturization is critical to creating an unobtrusive and high quality product.

3.4 – Acquire Other Biological Signals

There are many other biological signals that fall within the same characteristics as EMG in terms of frequency range. Our product has been designed from the very beginning to be capable of handling these other biological signals. We can transmit any signal from DC to 500 Hz, and both ECG and EEG are much lower than this limit. We have yet to test this feature and it would be extremely interesting and important to develop our ability to transmit ECG, EEG, and other patient monitoring signals via the same hardware setup.

3.5 – Increased Battery Life

Our current battery life of 2.5 hours on two rechargeable NiMH batteries and 6 hours on one non-rechargeable lithium is appropriate for EMG data acquisition applications. However, as our system is adapted to other signals, we need to investigate increasing this battery life to 24 hours on rechargeable batteries. There are several options that must be investigated. First, a different hardware solution on the TGE could eliminate waste due to heat by eliminating voltage regulators. We could use two separate DC-DC converters, one to produce 10V and the other to produce 3.3V. This would cut the current consumption of the TGE by a factor of two, more than doubling our battery lifetime. Secondly, there exist proprietary battery technologies that may be better suited in terms of power, size and weight to deliver power to the system.

3.6 – Fully Independent Wireless EMG

The initial goal of our system was to have absolutely no wires on the body. Unfortunately, the pads that were provided by Dr. Ted Milner required an external ground to work properly because of the internal operation of a differential amplifier. Therefore, one major goal is to redesign the EMG pad such that it can be used without the TGE. In this setup the transmitting would be done from each EMG pad directly, eliminating all the wires on the system. Currently, Aaron Ridinger is investigating the opportunity to develop such an EMG pad as his undergraduate thesis.

4.0 – Personal Experience

4.1 – Andrew Pruszynski

Developing this project has been one of the most rewarding, enjoyable and often frustrating experiences of my education. I have learned many technical and interpersonal skills that I will undoubtedly use for the rest of my life.

From a technical point of view, I have combined all the knowledge that I have gathered from my coursework and coop experience. We have been able to create something from only an idea. I developed the C based software which built on my experiences while working in the Biomechanics Laboratory and Ballard. I designed and manufactured the mother of all spaghetti boards that required digital design knowledge and all the soldering skills and patience that have been developed over the past four years. More important then any single discreet contribution to the project was the endless debugging to figure out what was wrong and how to fix it. This has taught me that what is wrong is often the most simple thing, so simple that hours may be spent rejecting it as a possibility.

We have completed a large and difficult task and I am extremely happy with the final outcome, the personal experience associated with getting there and the technical knowledge that I have gained.

4.2 – Eric Chow

This project has shown me that many hours of research and development must go into a product in order to ensure a high quality end result. The project required us to work on all aspects of a product design, including research, hardware/software design, assembly and testing, purchasing and finances, and documentation.

The technical skills I have learned from other courses and jobs enabled me to contribute to the technical design of our project. Notable skills included hardware design, radio frequency communications, and basic electronics laboratory skills.

My contribution to the team was primarily focused on the research of a suitable RF transmitter/receiver pair and the TGE hardware design. In addition, I worked on building the prototype and assisted in the microcontroller firmware development.

In working with Andrew, Dave, and Aaron, I have realized the importance of good communication within a group to ensure good team dynamics. Our weekly meetings helped keep the group on track, and although tasks were divided among us we made sure to that everyone was kept up to date on the status of the project.

I see our product as having great market potential as well as research potential, and I hope to continue work with this product and this group in the future.

4.3 – David Press

I found this project both incredibly frustrating and incredibly rewarding. I have gained many skills and experiences that will aid me throughout my career.

Perhaps the most valuable single thing I learned is that when working on a project of this magnitude, not everything can be perfect. There were many times near the beginning when I would spend many hours perfecting a small aspect of the project. Towards the end, I learned that sometimes ‘good enough is good enough’, which is a huge step for me as I am known as a bit of a perfectionist.

My main contribution to this project was the firmware and transmission protocols for transmitting the data wirelessly. This was the first time I had picked up the manual for a completely unfamiliar microcontroller, and needed to learn every minute detail and quirk of its operation. I feel much more comfortable that I could do the same again if need be. I

also gained a lot of experience in haggling and hounding parts suppliers for lower prices and rushed shipping. I had to communicate frequently with design engineers at Chipcon in Norway, which proved to be an interesting cultural and logistical challenge.
Overall, I am very pleased that the enormous effort we put into this project early on allowed us to accomplish our ambitious goals with time left to spare at the end of the semester.

4.4 – Aaron Ridinger

By working on this project I have further developed my technical, problem solving and interpersonal skills. I enjoyed both the technical aspects and team dynamics of this project.

New the beginning of this project I learned the many different possibilities of interfacing a PC with external devices such as USB, PCI, ISA, Firewire, and made a decision on which would work best for our project based on function and price. By working on the PC software I re-familiarized myself with programming in C, and utilized the Matlab skills I gained through past course work. Near the completion of the project I designed the TGE hardware and assembled it, which required me to recall my soldering skills I learned in high school.

Through this project and other course work this semester, I have discovered that I really don't want to design IC chips, as I initially planned on when I started studying engineering, but rather work in an area developing products to meet certain needs and solve specific solutions. After completing this project I now have an interest in bio-medical engineering. In the future I plan on doing a directed studies course in the area of EMG and mostly likely a thesis on the redesign of the EMG pad to eliminate the ground reference strap.

5.0 – Timeline

Figure 5.1 presents the original timeline as included in the project proposal.

ID	Task Name	Start	End
1	Proposal	26/08/2002	13/09/2002
2	Proposal Completed	13/09/2002	13/09/2002
3	Research	26/08/2002	30/09/2002
4	Material Collection	02/09/2002	25/10/2002
5	1st Progress Report Completed	08/10/2002	08/10/2002
6	Functional Specification	16/09/2002	14/10/2002
7	Functional Specification Completed	14/10/2002	14/10/2002
8	Initial Design Specification	20/09/2002	10/10/2002
9	Design Specifications Finalized	21/10/2002	21/10/2002
10	Initial Prototype Build	04/10/2002	14/11/2002
11	Design Specification	11/10/2002	01/11/2002
12	Design Specification Completed	01/11/2002	01/11/2002
13	Prototype Testing	06/11/2002	25/11/2002
14	2nd Progress Report Completed	29/11/2002	29/11/2002
15	Prototype Revision	21/11/2002	04/12/2002
16	Product Prototype Completed	04/12/2002	04/12/2002
17	Post Mortem Report	29/11/2002	16/12/2002
18	Final Documentation Completed	13/12/2002	13/12/2002
19	Document and Website Maintanance	26/08/2002	13/12/2002

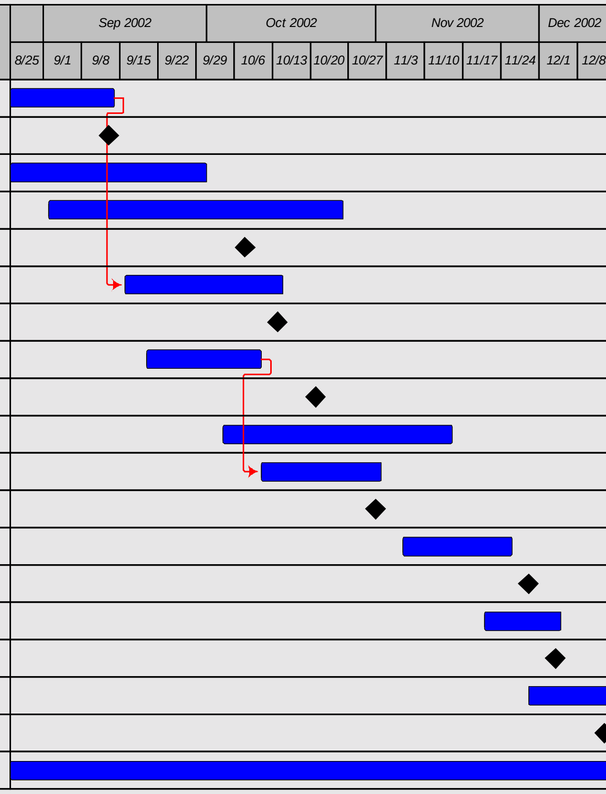

Figure 5.1 – Original Gantt Chart

It is very interesting to note that we met each of our target milestone dates. We made a significant effort to be ahead of schedule because we feared a large drain on our exam performance if too much ENSC340 work was left into the final weeks as originally planned. Thus, most of the difficult work was done very early in November, with a functioning prototype existing three weeks earlier then predicted. This allowed us several weeks to debug the system and focus our attention on assembly and testing.

6.0 – Budget

We applied and received a total of $1500 of funding from the Wighton Fund and the ESSEF. A condensed spreadsheet outlining our expenses over the course of the semester is presented in Table 6.1.

Date	Description	Cost ($CDN)
12/07/2002	Batteries	$20.13
12/07/2002	Batteries	$11.43
11/28/2002	Antennas	$50.03
11/30/2002	Op-amps, Voltage Regulators	$16.00
11/27/2002	EMG connectors	$3.37
11/19/2002	DC-DC converter, voltage step up	$128.00
10/14/2002	Transceivers	$329.60
11/06/2002	Chipcon shipping	$177.14
10/21/2002	PCI I/O card	$176.55
10/01/2002	PICs, Lynx Transmitter, Receiver	$181.47
11/06/2002	Oscillators, PICs, transmitter, receiver, UV Eraser	$266.97
10/17/2002	Chipcon initial shipping	$17.79
11/08/2002	Oscillators, multiplexers, Flip-Flops, PICs, PC board	$127.02
09/30/2002	Phone card	$5.00
11/13/2002	Phone card	$5.00
11/16/2002	PC Board, switches	$17.14
11/07/2002	Batteries	$37.02
	Total Expenses	**$1,569.66**

Table 6.1 – Simplified Table of All Expenses

It should be noted that we would be well within out budget if not for an unnecessary shipping charge due to miscommunication with the Engineering Science Office Staff. This mistake cost us $177 as our ordered product made several trips across the Atlantic Ocean. In addition, we compared competing transmitter technologies for a cost of $200. We will also be able to return approximately $300 worth of components back to the School of Engineering Science for future use. With these notes in mind, Table 6.2 presents the cost of producing the final prototype for the course.

Item	Unit Cost	Units	Total cost
Batteries	$15.77	1	$15.77
Batteries	$7.86	1	$7.86
Batteries	$9.97	1	$9.97
PC Board	$5.99	2	$11.98
Switches	$2.99	1	$2.99
PCI-I/O Card	$142.00	1	$142.00
20MHz Oscillator	$1.94	4	$7.76
Dual 4 input multiplexer	$1.19	12	$14.28
D Flip-Flop	$1.39	12	$16.68
28 pin PIC16C773	$18.99	2	$37.98
40 pin PIC16C774	$23.48	2	$46.96

PC Board	$10.53	1	$10.53
8MHz Oscillator	$1.94	4	$7.76
Transceivers CC1000PP 868MHz	$64.00	4	$256.00
DC-DC converter, voltage step up	$56.00	2	$112.00
EMG female connectors	$1.27	1	$1.27
EMG male connectors	$1.67	1	$1.67
Op-amp 7805UC??	$4.99	1	$4.99
Voltage Regulator LMC660CN???	$4.49	2	$8.98
Antenna 916MHz 1/4 wave	$10.18	2	$20.36
Antenna mini 7mm, 916MHz	$3.16	2	$6.32
Battery Holder	$2.75	2	$5.50
Batteries	$9.98	1	$9.98
Batteries	$2.59	1	$2.59
Batteries	$14.99	1	$14.99
Total Component Expenses			**$777.17**

Table 6.2 – Single Prototype Cost

We were able to produce a very complicated system without going significantly over budget. In fact, if not for an unexpected shipping cost and an extended evaluation of competing technologies, we would be well within our budgetary constraints.

7.0 – Conclusion

The past semester has been difficult, frustrating, tiring and most importantly rewarding for all the members of WMD. We have faced an enormous challenge to design, implement and document an extremely complex system within tight budgetary and time constraints. Despite many long nights with difficult trials and tribulations, we have succeeded. We now look into the future to see how to further improve our device and transform it from a project to a product.

Appendix

Transmitter Microcontroller Code (Assembly)

```
       list      p=16c773            ; list directive to define processor
       #include <p16c773.inc>        ; processor specific variable definitions

       __CONFIG   _CP_OFF & _WDT_ON & _BODEN_ON & _PWRTE_ON & _RC_OSC & _VBOR_25

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.



;***** VARIABLE DEFINITIONS
w_temp         EQU     0x70       ; variable used for context saving
status_temp    EQU     0x71       ; variable used for context saving
LAST_RES_0L            EQU     0x72
LAST_RES_0H    EQU     0x73
LAST_RES_1L    EQU     0x74
LAST_RES_1H    EQU     0x75
OUR_STAT       EQU     0x76       ; bit0 = done A/D, bit1 = last timer type, bit2 = calc
bytes out

w_prog_status  EQU     0x77
byte_to_write  EQU     0x78
num_bytes_written      EQU     0x79
next_byte_out  EQU     0x7A




dummy          EQU     0x20
dummy2         EQU     0x21
byte0          EQU     0x22
byte1          EQU     0x23
byte2          EQU     0x24
byte3          EQU     0x25
checksum       EQU     0x26

out0           EQU     0x27
out1           EQU     0x28
out2           EQU     0x29
out3           EQU     0x2A

num_bytes_written_2    EQU     0x2B

w_temp_2       EQU     0x2C
status_temp_2  EQU     0x2D



DONE_AD                EQU     0
LAST_TIMER_TYPE        EQU     1
CALC_BYTES_OUT EQU     2
PREAMBLE_MODE  EQU     3
SEND_BYTE              EQU     4
IS_ZERO                EQU     5
SEND_FILLER            EQU     6
t70us                  EQU     0
t930us                 EQU     1

;**********************************************************************
               ORG     0x000             ; processor reset vector
               clrf    PCLATH            ; ensure page bits are cleared
               goto    main              ; go to beginning of program

               ORG     0x004             ; interrupt vector location
               movwf   w_temp            ; save off current W register contents
               movf    STATUS,w          ; move status register into W register
```

```
                movwf   status_temp       ; save off contents of STATUS register


; isr code can go here or be located as a call subroutine elsewhere
;;;;;;; BEGIN ISR ;;;;;;;;;;;;;;;
isr
                bsf     STATUS, RP0                 ; page 1

                btfss   INTCON, T0IF
                goto    int_was_USART
;;; int for timer

                btfss   OUR_STAT, LAST_TIMER_TYPE
                goto    was_70us

; last timer was 930 us

                bcf     STATUS, RP0    ; page 0
                movlw   0xEE
                movwf   TMR0           ; set timer to overflow in 70 us

                bsf             PORTA, 2

                bsf     ADCON0, CHS0   ; set next A/D to ch1

                movf    ADRESH, W      ; get ch0 result
                movwf   LAST_RES_0H
                bsf     STATUS, RP0    ; page 1
                movf    ADRESL, W
                movwf   LAST_RES_0L
                bcf     OUR_STAT, LAST_TIMER_TYPE     ; last timer now was 70!


                goto    finish_timer_isr

was_70us        bcf     STATUS, RP0    ; page 0
                movlw   0x24
                movwf   TMR0           ; set timer to overflow in 930 us
                bcf     ADCON0, CHS0   ; set next A/D to ch0



                movf    ADRESH, W      ; get ch1 result
                movwf   LAST_RES_1H
                bsf     STATUS, RP0    ; page 1
                movf    ADRESL, W
                movwf   LAST_RES_1L

                bsf     OUR_STAT, LAST_TIMER_TYPE     ; last timer was 930
                bsf     OUR_STAT, CALC_BYTES_OUT



finish_timer_isr
                bcf     INTCON, T0IF   ; reset the timer0 interupt flag



                bcf     STATUS, RP0    ; page 0
                bsf     ADCON0, GO_DONE       ; start A/D conversion



                btfsc   OUR_STAT, LAST_TIMER_TYPE
                goto    end_isr
                                       ; last timer was 930 us, update the out0..out3
                movf    byte0,W
                movwf   out0
                movf    byte1,W
                movwf   out1
                movf    byte2,W
                movwf   out2


                bcf             PORTA, 2

                btfsc   OUR_STAT, PREAMBLE_MODE
                goto    end_isr
```

```
                movlw   0x01
                movwf   num_bytes_written

                goto    end_isr

int_was_USART


                bcf     STATUS, RP0    ; page 0
                btfss   OUR_STAT, PREAMBLE_MODE
                goto    send_data
send_out_preamble

                incf    num_bytes_written, F
                movf    num_bytes_written, W


                sublw   0xFF
                btfsc   STATUS, Z
                goto    end_preamble
                movlw   0x55
                movwf   TXREG
                goto    end_isr
end_preamble
                bcf             OUR_STAT, PREAMBLE_MODE
                movlw   0x55
                movwf   TXREG
                movlw   0x00
                movwf   num_bytes_written
                bsf             OUR_STAT, SEND_FILLER
                clrf    num_bytes_written
                goto    end_isr
send_data

                movlw   0x80

                btfsc   num_bytes_written, 1
                movf    out0, W
                btfsc   num_bytes_written, 2
                movf    out1, W
                btfsc   num_bytes_written, 3
                movf    out2, W
                btfsc   num_bytes_written, 4
                movlw   0x55

write_to_txreg
                movwf   TXREG

                bcf             STATUS, C
                btfss   num_bytes_written, 4
                rlf             num_bytes_written, F

                goto    end_isr


;;;;;;;;; END OF ISR CODE;;;;;;;;
end_isr

                movf     status_temp,w     ; retrieve copy of STATUS register
                movwf   STATUS            ; restore pre-isr STATUS register contents
                movf    w_temp, w
                retfie

;;;;;;;;;; END OF ISR ;;;;;;;;;;;;;;;


main
;;;;;; setup ;;;;;;;;;
                bcf     STATUS, RP1    ; access page 1
                bsf     STATUS, RP0

                clrf    num_bytes_written
                movlw   0x08
                movwf   OUR_STAT
                movlw   0x55
                movwf   next_byte_out

                movlw   0x8D           ; setup analog/digital IO     PA0,PA1 analog
                movwf   ADCON1
```

```
               movlw   0x03           ; PA7..PA2 output   PA0, PA1 input
               movwf   TRISA

               movlw   0x00
               movwf    TRISB         ; PORTB outputs

               movlw   0x50
               movwf   TRISC          ; PC 6,4 input, rest output

               movlw   0x02           ; pull ups enabled, int on rise edge, internal clock
for timer0, prescaler to timer0, 1:8 prescaler (Fosc/32)
               movwf   OPTION_REG

               movlw   0x40           ; initialize SSPSTAT
               movwf   SSPSTAT

               movlw   0x32           ; initialize USART
               movwf   TXSTA

               bsf     PIE1, TXIE     ; enable USART transmit interupt


               bcf     STATUS, RP0    ; access page 0


               movlw   0x80
               movwf   RCSTA

               movlw   0x30           ;
               movwf   SSPCON         ; enable SPI! must be after SSPSTAT has been written

               movlw   0x81           ; A/D TOsc/32, AN0, No Conversion Complete, On
               movwf   ADCON0

               movlw   0x55           ; put something in USART tx reg
               movwf   TXREG

               call    config_CC1000_TX

               movlw   0xE0           ; global interupt on, , timer0 int on, , ,
               movwf   INTCON



loop
               btfss   OUR_STAT, CALC_BYTES_OUT
               goto    loop

; just got out of 70us interupt, calculate the next message bytes
               bcf     STATUS, RP0            ; page 0



               movf    LAST_RES_0H, W
;              movlw   0x12           ; REMOVE THIS!!!
               andlw   0x0F
               movwf   byte0
               movf    LAST_RES_0L, W
;              movlw   0x34           ; REMOVE THIS!!!
               movwf   byte1


               movf    LAST_RES_1H, W
;              movlw   0x56           ; REMOVE THIS!!!
               andlw   0x0F
               movwf   byte2
               movf    LAST_RES_1L, W
;              movlw   0x78           ; REMOVE THIS!!!
               movwf   byte3

; make byte 0
               swapf   byte0, F

               swapf   byte1, W
               andlw   0x0F
               iorwf   byte0, F

; make byte 1
```

```
                swapf   byte1, F
                movlw   0xF0
                andwf   byte1, F

                movf    byte2, W
                andlw   0x0F
                iorwf   byte1, F

; make byte 2
                movf    byte3, W
                movwf   byte2


end_calc
                bcf     OUR_STAT, CALC_BYTES_OUT

                goto    loop




;;;;;;;;;;;;;;; SUBROUTINES ;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;
w_prog_addr
                movwf   byte_to_write
                movf    STATUS, W
                movwf   w_prog_status

                bcf     STATUS, RP0
                bcf     STATUS, RP1     ; page 0

                bcf     PORTC, 2        ; clear PALE

                movf    w_prog_status, W
                movwf   STATUS
                movf    byte_to_write, W

w_prog_data
                                        ;;; writes a byte to prog CC1000 thru SPI
                movwf   byte_to_write
                movf    STATUS, W
                movwf   w_prog_status   ; save the current status

                bcf     STATUS, RP0     ; page 0
                bcf     STATUS, RP1

                movf    byte_to_write, W
                movwf   SSPBUF

                bsf     STATUS, RP0     ; page 1
wait_for_SPI_0  btfss   SSPSTAT, BF     ; wait for the byte to be written to SPI
                goto    wait_for_SPI_0

                bcf     STATUS, RP0     ; page 0
                bsf     PORTC, 2        ; set PALE high

                movf    w_prog_status, W
                movwf   STATUS
                return
;;;;;;;;;;;;;;;;;;;;;;;
r_prog
                ;;;;;   reads a byte from CC1000 thru SPI
                ;;;;;   returns byte in W reg
                movf    STATUS, W
                movwf   w_prog_status   ; save the current status

                bcf     STATUS, RP1
                bsf     STATUS, RP0     ; page 1

                bsf     TRISC, 5        ; set PC5 (Data out) to input

                bcf     STATUS, RP0     ; page 0

                movwf   SSPBUF          ; write random crap from W

                bsf     STATUS, RP0     ; page 1
wait_for_SPI_1  btfss   SSPSTAT, BF     ; wait for the byte to be written to SPI
```



```
                goto    wait_for_SPI_1

                bcf     TRISC, 5        ; set PC5 to output

                movf    w_prog_status, W
                movwf   STATUS
                return
;;;;;;;;;;;;;;;;;
wait_500us
                clrf    dummy
wait1           incf    dummy
                btfss   STATUS, Z
                goto    wait1
                return
;;;;;;;;;;;;;;;;;



config_CC1000_TX
; make the transmit wake up later than the rx
                clrf    dummy2
wait3           call    wait_500us
                incf    dummy2
                btfss   STATUS, Z
                goto    wait3


                movlw   CC1000_MAIN_W
                call    w_prog_addr
                movlw   0x3A
                call    w_prog_data

                movlw   CC1000_MAIN_W
                call    w_prog_addr
                movlw   0x3B
                call    w_prog_data

                call    wait_500us
                call    wait_500us
                call    wait_500us
                call    wait_500us
                call    wait_500us


                movlw   CC1000_FREQ_2A_W
                call    w_prog_addr
                movlw   0x5B
                call    w_prog_data
                movlw   CC1000_FREQ_1A_W
                call    w_prog_addr
                movlw   0xA3
                call    w_prog_data
                movlw   CC1000_FREQ_0A_W
                call    w_prog_addr
                movlw   0x13
                call    w_prog_data

                movlw   CC1000_FREQ_2B_W
                call    w_prog_addr
                movlw   0x5B
                call    w_prog_data
                movlw   CC1000_FREQ_1B_W
                call    w_prog_addr
                movlw   0xA3
                call    w_prog_data
                movlw   CC1000_FREQ_0B_W
                call    w_prog_addr
                movlw   0x13
                call    w_prog_data

                movlw   CC1000_FSEP1_W
                call    w_prog_addr
                movlw   0x01
                call    w_prog_data
                movlw   CC1000_FSEP0_W
                call    w_prog_addr
                movlw   0xAB
                call    w_prog_data
```

```
movlw   CC1000_CURRENT_W
call    w_prog_addr
movlw   0xF3
call    w_prog_data

movlw   CC1000_FRONT_END_W
call    w_prog_addr
movlw   0x32
call    w_prog_data

movlw   CC1000_PA_POW_W              ; set PA_POW to zero for calibration
call    w_prog_addr
movlw   0x00
call    w_prog_data

movlw   CC1000_PLL_W
call    w_prog_addr
movlw   0x30
call    w_prog_data

movlw   CC1000_LOCK_W
call    w_prog_addr
movlw   0x10
call    w_prog_data

movlw   CC1000_MODEM2_W
call    w_prog_addr
movlw   0xC2
call    w_prog_data
movlw   CC1000_MODEM1_W
call    w_prog_addr
movlw   0x6F                  ; was 13, keep at 6F
call    w_prog_data
movlw   CC1000_MODEM0_W
call    w_prog_addr
movlw   0x54
call    w_prog_data

movlw   CC1000_MATCH_W
call    w_prog_addr
movlw   0x10
call    w_prog_data

movlw   CC1000_FSCTRL_W
call    w_prog_addr
movlw   0x01
call    w_prog_data

movlw   CC1000_PRESCALER_W
call    w_prog_addr
movlw   0x00
call    w_prog_data

movlw   CC1000_TEST6_W
call    w_prog_addr
movlw   0x10
call    w_prog_data

movlw   CC1000_TEST5_W
call    w_prog_addr
movlw   0x08
call    w_prog_data

movlw   CC1000_TEST4_W
call    w_prog_addr
movlw   0x3F
call    w_prog_data
movlw   CC1000_TEST3_W
call    w_prog_addr
movlw   0x04
call    w_prog_data
movlw   CC1000_TEST2_W
call    w_prog_addr
movlw   0x00
call    w_prog_data
movlw   CC1000_TEST1_W
call    w_prog_addr
movlw   0x00
call    w_prog_data
movlw   CC1000_TEST0_W
```

```
                call    w_prog_addr
                movlw   0x00
                call    w_prog_data

; begin calibration
                movlw   CC1000_CAL_W
                call    w_prog_addr
                movlw   0x66
                call    w_prog_data

                movlw   CC1000_MAIN_W
                call    w_prog_addr
                movlw   0xA1
                call    w_prog_data

                call    wait_500us
                call    wait_500us
                call    wait_500us
                call    wait_500us
                call    wait_500us

                movlw   CC1000_CURRENT_W
                call    w_prog_addr
                movlw   0xF3
                call    w_prog_data

                movlw   CC1000_CAL_W
                call    w_prog_addr
                movlw   0xE6
                call    w_prog_data

                clrf    dummy2
wait2           call    wait_500us
                incf    dummy2
                btfss   STATUS, Z
                goto    wait2


                movlw   CC1000_CAL_W
                call    w_prog_addr
                movlw   0x66
                call    w_prog_data
; end calibration



                movlw   CC1000_PA_POW_W
                call    w_prog_addr
                movlw   0xFF
                call    w_prog_data

                call    wait_500us

                return

                END                       ; directive 'end of program'
```

Receiver Microcontroller Code (Assembly)

```
        list      p=16c774            ; list directive to define processor
        #include <p16c774.inc>        ; processor specific variable definitions

        __CONFIG   _CP_OFF & _WDT_ON & _BODEN_ON & _PWRTE_ON & _RC_OSC & _VBOR_25

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.


;***** VARIABLE DEFINITIONS
```

```
w_temp          EQU     0x70        ; variable used for context saving
status_temp     EQU     0x71        ; variable used for context saving
RSSI            EQU     0x72
DATA0           EQU     0x73
OUR_STAT        EQU     0x74       ; bit0 = done A/D

w_prog_status   EQU     0x75
num_match_bytes         EQU     0x76
our_state       EQU     0x77
byte_to_write   EQU     0x78

dummy           EQU     0x20
dummy2          EQU     0x21

byte_num        EQU     0x22
byte0           EQU     0x23
byte1           EQU     0x24
byte2           EQU     0x25
byte3           EQU     0x26
byte4           EQU     0x27
byte5           EQU     0x28
byte6           EQU     0x29
byte7           EQU     0x2A

checksum1       EQU     0x2B
checksum2       EQU     0x2C
checksum3       EQU     0x2D
checksum4       EQU     0x2E

num_times_waited        EQU     0x2F

in0             EQU     0x30
in1             EQU     0x31
in2             EQU     0x32
in3             EQU     0x33
in4             EQU     0x34
in5             EQU     0x35
in6             EQU     0x36
in7             EQU     0x37

out1B           EQU     0x38
out1D           EQU     0x39
out1A           EQU     0x3A
out2B           EQU     0x3B
out2D           EQU     0x3C
out2A           EQU     0x3D
counter         EQU     0x3E
header          EQU     0x3F

; OUR_STAT bits
WAIT_FOR_SB     EQU     0
DO_CALC         EQU     1
samp1a_bad      EQU     2
samp2a_bad      EQU     3
samp1b_bad      EQU     4
samp2b_bad      EQU     5


; our_state words

WAIT_FOR_SYNC           EQU     0x04
SYNC_                   EQU     0x08

;**********************************************************************
                ORG     0x000             ; processor reset vector
                clrf    PCLATH            ; ensure page bits are cleared
                goto    main              ; go to beginning of program

                ORG     0x004             ; interrupt vector location
                movwf   w_temp            ; save off current W register contents
                movf    STATUS,w          ; move status register into W register
                movwf   status_temp       ; save off contents of STATUS register


; isr code can go here or be located as a call subroutine elsewhere
;;;;;;; BEGIN ISR ;;;;;;;;;;;;;;
isr
```

```
                bcf     STATUS, RP0     ; page 0

int_was_USART

                movf    RCREG, W

                movwf   DATA0
                btfsc   our_state, 3
                goto    get_data

wait_sync


                bcf     STATUS, RP0             ; page 0
                bcf     RCSTA, SPEN             ; disable the USART
start_count
                clrf    dummy
count_this
                incf    dummy,F
;               nop                                     ; was a nop here, but it fucks things
up!!! cant lock when initial message is all 0 or all 1 for some reason!
                btfss   PORTC,7
                goto    count_this
                btfss   dummy, 5        ; must get thru this loop 64 times to be the start
byte
                goto    start_count



                movlw   SYNC_
                movwf   our_state
                clrf    OUR_STAT


                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop

                movlw   0x00
                movwf   dummy
gohere
                incf    dummy, F
                btfss   dummy, 3
                goto    gohere

                bcf     RCSTA, CREN     ; clear an overrun if there was one
                movlw   0x90
                movwf   RCSTA           ; turn on USART

                goto    end_isr

get_data

                movf    DATA0, W
                btfss   OUR_STAT, WAIT_FOR_SB
                goto    get_data_byte

; check if its a start byte
                incf    num_times_waited, F
                btfsc   num_times_waited, 3
                goto    screwed         ; if we've looked for the start byte 8 times in a
row, we're screwed up
                sublw   0x80
                btfss   STATUS, Z
                goto    end_isr         ; nope, not a start byte

; start byte

                clrf    num_times_waited
                bcf     OUR_STAT, WAIT_FOR_SB
                movlw   0x01
                movwf   byte_num

                movf    byte0, W
                movwf   in0
```

```
                movf    byte1, W
                movwf   in1
                movf    byte2, W
                movwf   in2

                bcf             PORTA, 5
                btfsc   RSSI, 3
                bsf             PORTA, 5
                swapf   RSSI, W

                andlw   0x07
                movwf   PORTE

                movf    out1B, W
                movwf   PORTB
                movf    out1D, W
                movwf   PORTD

                bsf             PORTC, 0
                bcf             PORTC, 0

                movf    out2B, W                ; uncomment this after demo
                movwf   PORTB
                movf    out2D, W
                movwf   PORTD

                bsf             PORTC, 1
                bcf             PORTC, 1

                bsf     OUR_STAT, DO_CALC
                goto    end_isr

screwed
;               movlw   0x04            ;;;;;old
;               movwf   our_state
;               movlw   0x01
;               movwf   byte_num

                movlw   0x04            ;;;;; new
                movwf   our_state
                clrf    OUR_STAT
                clrf    num_times_waited
                movlw   0x01
                movwf   byte_num


                goto    end_isr



get_data_byte                                   ; we're not waiting for SB so its data
                                                ; W contains data
                btfsc   byte_num,0
                movwf   byte0
                btfsc   byte_num,1
                movwf   byte1
                btfsc   byte_num,2
                movwf   byte2

                bcf     STATUS, C
                rlf     byte_num, F
                btfsc   byte_num, 3
                bsf     OUR_STAT, WAIT_FOR_SB


                goto    end_isr


;;;;;;;;; END OF ISR CODE;;;;;;;;
end_isr


                movf     status_temp,w      ; retrieve copy of STATUS register
                movwf   STATUS              ; restore pre-isr STATUS register contents
                swapf    w_temp,f
                swapf    w_temp,w           ; restore pre-isr W register contents
```

```
                retfie                      ; return from interrupt
;;;;;;;;;; END OF ISR ;;;;;;;;;;;;;;;


main
;;;;;; setup ;;;;;;;;;
                bcf     STATUS, RP1     ; access page 1
                bsf     STATUS, RP0


                movlw   0x0E            ; setup analog/digital IO     PA0 analog, left
justified
                movwf   ADCON1

                movlw   0x01            ; PA7..PA1 output   PA0, input
                movwf   TRISA

                movlw   0x00
                movwf     TRISB         ; PORTB outputs
                movlw   0x00
                movwf     TRISD         ; PORTD outputs
                movlw   0x00
                movwf     TRISE         ; PORTE outputs, PE & PD general purpose I/O

                movlw   0xD0
                movwf   TRISC           ; PC 7,6,4 input, rest output

                movlw   0x02            ; pull ups enabled, int on rise edge, internal clock
for timer0, prescaler to timer0, 1:8 prescaler (Fosc/32)
                movwf   OPTION_REG

                movlw   0x40            ; initialize SPI
                movwf   SSPSTAT

                movlw   0x12            ; initialize USART
                movwf   TXSTA



                bsf     PIE1, RCIE      ; enable USART receiver interupt


                bcf     STATUS, RP0     ; access page 0


                movlw   0x04
                movwf   our_state
                clrf    OUR_STAT
                movlw   0x01
                movwf   byte_num

                movlw   0x80
                movwf   RCSTA

                movlw   0x30            ;
                movwf   SSPCON          ; enable SPI! must be after SSPSTAT has been written

                movlw   0x81            ; A/D TOsc/32, AN0, No Conversion Complete, On
                movwf   ADCON0

                call    config_CC1000_RX

                movlw   0xC0            ; global interupt on, , timer0 int off, , ,
                movwf   INTCON

                movlw   0x90            ; enable cont receive on USART
                movwf   RCSTA


loop

                bcf     STATUS,RP0              ; make sure we're on page 0
                btfss   OUR_STAT, DO_CALC
                goto    loop

analyze_data


                bcf             ADCON0, GO_DONE
                movf    ADRESH, W       ; get ch0 result
```

```
                movwf   RSSI

                incf    counter, F                    ; get the header nibble ready, 0F in
W
                movlw   0x0F
                andwf   counter, F

                movf    in0, W
                andlw   0xF0
                movwf   out1B
                swapf   out1B, F

                swapf   counter, W
                iorwf   out1B, F

                movf    in0, W
                andlw   0x0F
                movwf   out1D
                swapf   out1D

                swapf   in1, W
                andlw   0x0F
                iorwf   out1D, F

                movf    in1, W
                andlw   0x0F
                movwf   out2B

                swapf   counter, W
                iorwf   out2B, F

                movf    in2, W
                movwf   out2D


end_calc

                bsf     ADCON0, GO_DONE       ; start next A/D conversion
                bcf     OUR_STAT, DO_CALC
                goto    loop


;;;;;;;;;;;;;; SUBROUTINES ;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;
w_prog_addr
                movwf   byte_to_write
                movf    STATUS, W
                movwf   w_prog_status

                bcf     STATUS, RP0
                bcf     STATUS, RP1     ; page 0

                bcf     PORTC, 2        ; clear PALE

                movf    w_prog_status, W
                movwf   STATUS
                movf    byte_to_write, W

w_prog_data
                                        ;;; writes a byte to prog CC1000 thru SPI
                movwf   byte_to_write
                movf    STATUS, W
                movwf   w_prog_status   ; save the current status

                bcf     STATUS, RP0     ; page 0
                bcf     STATUS, RP1

                movf    byte_to_write, W
                movwf   SSPBUF

                bsf     STATUS, RP0     ; page 1
wait_for_SPI_0  btfss   SSPSTAT, BF     ; wait for the byte to be written to SPI
                goto    wait_for_SPI_0

                bcf     STATUS, RP0     ; page 0
                bsf     PORTC, 2        ; set PALE high

                movf    w_prog_status, W
```

```
                movwf   STATUS
                return
;;;;;;;;;;;;;;;;;;;;;;;
r_prog
                ;;;;;   reads a byte from CC1000 thru SPI
                ;;;;;   returns byte in W reg
                movf    STATUS, W
                movwf   w_prog_status  ; save the current status

                bcf     STATUS, RP1
                bsf     STATUS, RP0    ; page 1

                bsf     TRISC, 5       ; set PC5 (Data out) to input

                bcf     STATUS, RP0    ; page 0

                movwf   SSPBUF         ; write random crap from W

                bsf     STATUS, RP0    ; page 1
wait_for_SPI_1 btfss    SSPSTAT, BF    ; wait for the byte to be written to SPI
                goto    wait_for_SPI_1

                bcf     TRISC, 5       ; set PC5 to output

                movf    w_prog_status, W
                movwf   STATUS
                return
;;;;;;;;;;;;;;;

lock_filt
                movlw   CC1000_MODEM1_W
                call    w_prog_addr
                movlw   0x7F
                call    w_prog_data
                return

unlock_filt
                movlw   CC1000_MODEM1_W
                call    w_prog_addr
                movlw   0x6F
                call    w_prog_data
                return

;;;;;;;;;;;;;;;;;
wait_500us
                clrf    dummy
wait1           incf    dummy
                btfss   STATUS, Z
                goto    wait1
                return
;;;;;;;;;;;;;;;;;



config_CC1000_RX


                movlw   CC1000_MAIN_W
                call    w_prog_addr
                movlw   0x3A
                call    w_prog_data

                movlw   CC1000_MAIN_W
                call    w_prog_addr
                movlw   0x3B
                call    w_prog_data

                call    wait_500us
                call    wait_500us
                call    wait_500us
                call    wait_500us
                call    wait_500us


                movlw   CC1000_FREQ_2A_W
                call    w_prog_addr
                movlw   0x5B
                call    w_prog_data
                movlw   CC1000_FREQ_1A_W
```

```
        call    w_prog_addr
        movlw   0xA0
        call    w_prog_data
        movlw   CC1000_FREQ_0A_W
        call    w_prog_addr
        movlw   0x00
        call    w_prog_data

        movlw   CC1000_FREQ_2B_W
        call    w_prog_addr
        movlw   0x5B
        call    w_prog_data
        movlw   CC1000_FREQ_1B_W
        call    w_prog_addr
        movlw   0xA0
        call    w_prog_data
        movlw   CC1000_FREQ_0B_W
        call    w_prog_addr
        movlw   0x00
        call    w_prog_data

        movlw   CC1000_FSEP1_W
        call    w_prog_addr
        movlw   0x01
        call    w_prog_data
        movlw   CC1000_FSEP0_W
        call    w_prog_addr
        movlw   0xAB
        call    w_prog_data

        movlw   CC1000_CURRENT_W
        call    w_prog_addr
        movlw   0x8C
        call    w_prog_data

        movlw   CC1000_FRONT_END_W
        call    w_prog_addr
        movlw   0x32
        call    w_prog_data

        movlw   CC1000_PA_POW_W               ; set PA_POW to zero for calibration
        call    w_prog_addr
        movlw   0x00
        call    w_prog_data

        movlw   CC1000_PLL_W
        call    w_prog_addr
        movlw   0x30
        call    w_prog_data

        movlw   CC1000_LOCK_W
        call    w_prog_addr
        movlw   0x10
        call    w_prog_data

        movlw   CC1000_MODEM2_W
        call    w_prog_addr
        movlw   0xC2
        call    w_prog_data

        movlw   CC1000_MODEM1_W
        call    w_prog_addr
        movlw   0x6F
        call    w_prog_data
        movlw   CC1000_MODEM0_W
        call    w_prog_addr
        movlw   0x54
        call    w_prog_data

        movlw   CC1000_MATCH_W
        call    w_prog_addr
        movlw   0x10
        call    w_prog_data

        movlw   CC1000_FSCTRL_W
        call    w_prog_addr
        movlw   0x01
        call    w_prog_data

        movlw   CC1000_PRESCALER_W
```

```
                call    w_prog_addr
                movlw   0x00
                call    w_prog_data

                movlw   CC1000_TEST6_W
                call    w_prog_addr
                movlw   0x10
                call    w_prog_data

                movlw   CC1000_TEST5_W
                call    w_prog_addr
                movlw   0x08
                call    w_prog_data

                movlw   CC1000_TEST4_W
                call    w_prog_addr
                movlw   0x3F
                call    w_prog_data
                movlw   CC1000_TEST3_W
                call    w_prog_addr
                movlw   0x04
                call    w_prog_data
                movlw   CC1000_TEST2_W
                call    w_prog_addr
                movlw   0x00
                call    w_prog_data
                movlw   CC1000_TEST1_W
                call    w_prog_addr
                movlw   0x00
                call    w_prog_data
                movlw   CC1000_TEST0_W
                call    w_prog_addr
                movlw   0x00
                call    w_prog_data

; begin calibration
                movlw   CC1000_CAL_W
                call    w_prog_addr
                movlw   0x66
                call    w_prog_data

                movlw   CC1000_MAIN_W
                call    w_prog_addr
                movlw   0x11
                call    w_prog_data

                call    wait_500us
                call    wait_500us
                call    wait_500us
                call    wait_500us
                call    wait_500us

                movlw   CC1000_CURRENT_W
                call    w_prog_addr
                movlw   0x8C
                call    w_prog_data

                movlw   CC1000_CAL_W
                call    w_prog_addr
                movlw   0xE6
                call    w_prog_data

                clrf    dummy2
wait2           call    wait_500us
                incf    dummy2
                btfss   STATUS, Z
                goto    wait2



                movlw   CC1000_CAL_W
                call    w_prog_addr
                movlw   0x66
                call    w_prog_data
; end calibration

                return

                END                       ; directive 'end of program'
```

PC Software (C)

```
/***************************
INCLUDES
***************************/

#include      <stdio.h>
#include      <dos.h>
#include      <conio.h>
#include      <math.h>
#include      <io.h>
#include      <process.h>
#include      <alloc.h>
#include      <stdlib.h>
#include      <time.h>
#include      <fcntl.h>
#include      <sys\stat.h>
#include      <graphics.h>

/****************************
DEFINITIONS
****************************/

#define base     0xfff4
#define porta    base+0
#define portb    base+1
#define portc    base+2
#define ctrl     base+3

#define         DIAG    1
#define         ACQ     2
#define         QUIT    3

#define         Y_RES   70
#define         X_RES    1
#define         XAXSTR  140 //60
#define         XAXEND  620
#define         XSTRT    141 //61
#define         XEND     619
#define         RECLEAD 10
#define         YOFFSET 30
#define         XOFFSET 140//60

#define         XMIN    0
#define         XMAX     640
#define         YMIN     0
#define         YMAX     480

#define         PAD8    8
#define         PAD7    7
#define         PAD6    6
#define         PAD5    5
#define         PAD4    4
#define         PAD3    3
#define         PAD2    2
#define         PAD1    1

#define         YMIN8   0
#define         YMIN7    60
#define         YMIN6    120
#define         YMIN5    180
#define         YMIN4    240
#define         YMIN3    300
#define         YMIN2    360
#define         YMIN1    420

#define         YMAX8    59
#define         YMAX7    119
#define         YMAX6    179
#define         YMAX5    239
#define         YMAX4    299
#define         YMAX3    359
#define         YMAX2    419
#define         YMAX1    479

#define         XAXIS8   30
#define         XAXIS7   90
#define         XAXIS6   150
#define         XAXIS5   210
#define         XAXIS4   270
```

```
#define         XAXIS3  330
#define         XAXIS2  390
#define         XAXIS1  450

/*****************************
FUNCTION PROTOTYPES
*****************************/

void init(void);
int intro(void);
void data_diag(void);
void data_acq(void);
unsigned int truncate(unsigned long int);
void rt_graph(int *x_last,int *y_last, int ymin, int ymax, int xaxis);
void rt_sigstr(unsigned long int, int);
void draw_rectangles(void);
void draw_lines(void);

/*****************************
GLOBALS
*****************************/

FILE *fp_1, *fp_2, *fp_3, *fp_4, *fp_5, *fp_6, *fp_7, *fp_8;


unsigned long int ms_data = 0, ls_data = 0, data = 0;
unsigned long int last_data_1, last_data_2, last_data_3, last_data_4, last_data_5,
last_data_6, last_data_7, last_data_8;
int empty_1 = 1, empty_2 = 1, empty_3 = 1, empty_4 = 1, empty_5 = 1, empty_6 = 1, empty_7
= 1, empty_8 = 1;

int x_8 = XSTRT, x_7 = XSTRT, x_6 = XSTRT, x_5 = XSTRT, x_4 = XSTRT, x_3 = XSTRT, x_2 =
XSTRT, x_1 = XSTRT;
int y_8 = 30, y_7 = 90, y_6 = 150, y_5 = 210, y_4 = 270, y_3 = 330, y_2 = 390, y_1 = 450;
unsigned int short_data = 0;
int yaxis;

int mode = 0;
int input;
int mux_sel = 0x0;
int check;
int delay_sigstr = 0;

int gdriver = DETECT;
int gmode;

/************************
TEMPORARY GLOBALS
************************/

long int x = 0, y = 0;
unsigned long int test;
int tester=0;

main()
{
        init();         // initialization of IO Card and Files
        mode = intro();

        while(1)
        {
                if(mode == DIAG)
                        data_diag();

                else if(mode == ACQ)
                        data_acq();

                else if(mode == QUIT)
                        exit(0);

                mode = 0;
                mode = intro();
        }
}


void init()
{
        // Initalize the DIO Control Register, Regular Mode, A,B,CH In
        outp(ctrl, 0x9A);
```

```
        if((fp_8 = fopen("C:\\WMD\\emg1.dat", "wb+"))==NULL)
        {
                printf("Cannot open file.\n");
                exit(1);
        }
        if((fp_7 = fopen("C:\\WMD\\emg2.dat", "wb+"))==NULL)
        {
                printf("Cannot open file.\n");
                exit(1);
        }
        if((fp_6 = fopen("C:\\WMD\\emg3.dat", "wb+"))==NULL)
        {
                printf("Cannot open file.\n");
                exit(1);
        }
        if((fp_5 = fopen("C:\\WMD\\emg4.dat", "wb+"))==NULL)
        {
                printf("Cannot open file.\n");
                exit(1);
        }
        if((fp_4 = fopen("C:\\WMD\\emg5.dat", "wb+"))==NULL)
        {
                printf("Cannot open file.\n");
                exit(1);
        }
        if((fp_3 = fopen("C:\\WMD\\emg6.dat", "wb+"))==NULL)
        {
                printf("Cannot open file.\n");
                exit(1);
        }
        if((fp_2 = fopen("C:\\WMD\\emg7.dat", "wb+"))==NULL)
        {
                printf("Cannot open file.\n");
                exit(1);
        }
        if((fp_1 = fopen("C:\\WMD\\emg8.dat", "wb+"))==NULL)
        {
                printf("Cannot open file.\n");
                exit(1);
        }
}

int intro()
{
        printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n~~~~~~~~~~Welcome to 
EMGview~~~~~~~~~~\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

        printf("Please Select Your Mode: (d)iagnositc, (a)cquisition, (q)uit\n");

        while(mode == 0)
        {
                input = getch();
                if(input=='d')
                {
                        mode = DIAG;
                        printf("Entering Diagnostic Mode\n");
                }
                else if (input == 'a')
                {
                        mode = ACQ;
                        printf("Entering Acquisition Mode\n");
                }
                else if (input == 'q')
                {
                        mode = QUIT;
                        printf("Now Terminating, GoodBye");
                        delay(1000);
                }
                else
                        printf("Invald Entry, Please Try Again\n");
        }
        return mode;
}

void data_diag()
{
        ("Entering Diagnostic Mode, press any key to Exit\n");

        initgraph(&gdriver, &gmode, "\bgi");
```

```
        if(graphresult() != grOk)
        {
                printf("Error Opening Graphics Window.");
                getch();
                exit(1);
        }

        //labelling display

        outtextxy(XSTRT-42,XAXIS8, "EMG1");
        outtextxy(XSTRT-42,XAXIS7, "EMG2");
        outtextxy(XSTRT-42,XAXIS6, "EMG3");
        outtextxy(XSTRT-42,XAXIS5, "EMG4");
        outtextxy(XSTRT-42,XAXIS4, "EMG5");
        outtextxy(XSTRT-42,XAXIS3, "EMG6");
        outtextxy(XSTRT-42,XAXIS2, "EMG7");
        outtextxy(XSTRT-42,XAXIS1, "EMG8");
        outtextxy(XSTRT-115, 5, "Sig_Str");

        outtextxy(XSTRT - 79, XAXIS8+28, "TGE1");
        outtextxy(XSTRT - 79, XAXIS6+28, "TGE2");
        outtextxy(XSTRT - 79, XAXIS4+28, "TGE3");
        outtextxy(XSTRT - 79, XAXIS2+28, "TGE4");

        // draw initial y axis
        setcolor(1);
        int yaxis = 0 + YOFFSET;
        while (yaxis < 480)
        {
                line(XAXSTR,yaxis,XAXEND,yaxis);
                yaxis = yaxis+60;
        }
        // draw initial x axis
        line(XOFFSET,YMIN,XOFFSET,YMAX);

        // draw outline of signal strength boxes
        draw_rectangles();
        draw_lines();
        test = 0x0;
        while(!kbhit())
        {

                ms_data = inpw(porta);
                ls_data = (0xF0 & inp(portc)) >> 4;
                ms_data = (ms_data << 4);
                data = ((0x000FFFFF & (ms_data | ls_data)));
                outp(portc, (0x3 & (mux_sel+1)) << 2); // sets the MUX to toggle to next
pad

                if(mux_sel == 0x0)
                {
                        rt_graph(&x_8,&y_8,YMIN8,YMAX8,XAXIS8);
                        rt_sigstr(data, PAD8);
                }
                else if (mux_sel == 0x2)
                {
                        rt_graph(&x_7,&y_7,YMIN7,YMAX7,XAXIS7);
                        rt_sigstr(data, PAD7);
                }
                if (mux_sel == 0x1)
                {
                        rt_graph(&x_6,&y_6,YMIN6,YMAX6,XAXIS6);
                        rt_sigstr(data, PAD6);
                }
                else if (mux_sel == 0x3)
                {
                        rt_graph(&x_5,&y_5,YMIN5,YMAX5,XAXIS5);
                        rt_sigstr(data, PAD5);
                }

                if(mux_sel<0x3)
                        mux_sel++;
                else
                        mux_sel = 0x0;
        }
        getch();
        closegraph();
}

void data_acq()
```

```
{
// begin data acquisition
        printf("Press any Key to Begin Data Acquisition\n");
        getch();
        printf("\n~~~~~~~~~Data Acquisition Started~~~~~~~~~~\n");
        printf("\n\nPress Any Key to Stop\n");

        x = 0;
        // begin data acquistion
        while(!kbhit())
        {
                ms_data = inpw(porta);
                ls_data = (0xF0 & inp(portc)) >> 4;
                ms_data = (ms_data << 4);
                data = ((0x000FFFFF & (ms_data | ls_data)));
                check = (0xF0000 & data) >> 16;

                outp(portc, (0x3 & (mux_sel+1)) << 2); // sets the MUX to toggle to next
pad  outp(portc, (0x3 & (mux_sel+1)) << 2); // sets the MUX to toggle to next pa

                if(mux_sel == 0x0)
                {
                        if((check != last_data_8) || (empty_8 == 1))
                        {
                                short_data = truncate(data);
                                fwrite(&short_data, sizeof(short_data), 1, fp_8);
                                last_data_8 = check;
                                empty_8 = 0;
                        }
                }
                else if (mux_sel == 0x2)
                {
                        if((check != last_data_7) || (empty_7 == 1))
                        {
                                short_data = truncate(data);
                                fwrite(&short_data, sizeof(short_data), 1, fp_7);
                                last_data_7 = check;
                                empty_7 = 0;
                        }
                }
                else if (mux_sel == 0x1)
                {
                        if((check != last_data_6) || (empty_6 == 1))
                        {
                                short_data = truncate(data);
                                fwrite(&short_data, sizeof(short_data), 1, fp_6);
                                last_data_6 = check;
                                empty_6 = 0;
                        }
                }
                else if (mux_sel == 0x3)
                {
                        if((check != last_data_5) || (empty_5 == 1))
                        {
                                short_data = truncate(data);
                                fwrite(&short_data, sizeof(short_data), 1, fp_5);
                                last_data_5 = check;
                                empty_5 = 0;
                        }
                }
                if(mux_sel<0x3)
                        mux_sel++;

                else
                        mux_sel = 0x0;
        }
        getch();
        printf("Please Move the Data (emg1.dat-emg8.dat) Files Before You Attempt to
Acquire More Data\n");
        printf("\n\nPress any Key to Continue...\n");
        getch();

}


unsigned int truncate(unsigned long int in_data)
{
        int out_data;
        in_data = (0x000FFF0 & in_data) >> 4;
        out_data = in_data;
```

```
        return out_data;
}

void rt_graph(int *x_last,int *y_last, int ymin, int ymax, int xaxis)
{
        short_data = ((truncate(data))/(Y_RES));
        setcolor(0);
        rectangle(*x_last,ymin,*x_last+RECLEAD,ymax);
        setcolor(2);
        line(*x_last, *y_last,*x_last+X_RES,ymax-short_data);
        setcolor(1);
        line(XAXSTR,xaxis,XAXEND,xaxis);

        if(*x_last <= XEND)
                *x_last = *x_last + 1;
        else
                *x_last = XSTRT;

        *y_last = ymax-short_data;
}

void rt_sigstr(unsigned long int in_data, int pad)
{
        if(delay_sigstr < 500)
                delay_sigstr++;

        else
        {
                int sigstr, sigstr_last1 = 0, sigstr_last2 = 0;

                in_data = 16 - (in_data & 0xF);
                sigstr = ((4*in_data)-18)*1.53;

                if (sigstr <= 0)
                        sigstr = 64;
                else if (sigstr > 64)
                        sigstr = 0;



                if(pad == PAD8 || pad == PAD7)
                {
                        if(sigstr < 0)
                                sigstr = sigstr_last1;
                        else if (sigstr > 64)
                                sigstr = sigstr_last1;
                        else
                                sigstr_last1 = sigstr;

                        setfillstyle(0,0);
                        bar(XSTRT-89,YMAX8+34,XSTRT-86,YMIN8+29);
                        if (sigstr > 40)
                        {
                                setfillstyle(1,2);
                                bar(XSTRT-89,YMAX8+34-sigstr,XSTRT-86,(YMAX8+34));
                        }
                        else if (sigstr > 20)
                        {
                                setfillstyle(1,14);
                                bar(XSTRT-89,YMAX8+34-sigstr,XSTRT-86,(YMAX8+34));
                        }
                        else
                        {
                                setfillstyle(1,4);
                                bar(XSTRT-89,YMAX8+34-sigstr,XSTRT-86,(YMAX8+34));
                        }
                }
                else if(pad == PAD6 || pad == PAD5)
                {
                        if(sigstr < 0)
                                sigstr = sigstr_last2;
                        else if (sigstr > 64)
                                sigstr = sigstr_last2;
                        else
                                sigstr_last2 = sigstr;


                        setfillstyle(0,0);
                        bar(XSTRT-89,YMAX6+34,XSTRT-86,YMIN6+29);
                        if(sigstr > 40)
```

```
                        {
                                setfillstyle(1,2);
                                bar(XSTRT-89,YMAX6+34-sigstr,XSTRT-86,(YMAX6+34));
                        }
                        else if(sigstr > 20)
                        {
                                setfillstyle(1,14);
                                bar(XSTRT-89,YMAX6+34-sigstr,XSTRT-86,(YMAX6+34));
                        }
                        else
                        {
                                setfillstyle(1,4);
                                bar(XSTRT-89,YMAX6+34-sigstr,XSTRT-86,(YMAX6+34));
                        }

                }
                else if(pad == PAD4 || pad == PAD3)
                {
                        setfillstyle(0,0);
                        bar(XSTRT-89,YMAX4+34,XSTRT-86,YMIN4+29);
                        setfillstyle(1,1);
                        bar(XSTRT-89,YMAX4+34-sigstr,XSTRT-86,(YMAX4+34));

                }
                else if(pad == PAD2 || pad == PAD1)
                {
                        setfillstyle(0,0);
                        bar(XSTRT-89,YMAX2+34,XSTRT-86,YMIN2+29);
                        setfillstyle(1,1);
                        bar(XSTRT-89,YMAX2+34-sigstr,XSTRT-86,(YMAX2+34));
                }
                delay_sigstr = 0;
        }
}

void draw_rectangles()
{
        setcolor(7);
        rectangle(XSTRT-90, YMAX8+35, XSTRT-85, YMIN8+28);
        rectangle(XSTRT-90, YMAX6+35, XSTRT-85, YMIN6+28);
        rectangle(XSTRT-90, YMAX4+35, XSTRT-85, YMIN4+28);
        rectangle(XSTRT-90, YMAX2+35, XSTRT-85, YMIN2+28);
}

void draw_lines()
{
        setcolor(5);
        line(XSTRT - 25, XAXIS8 + 12, XSTRT - 25, XAXIS7 - 7);
        line(XSTRT - 25, XAXIS6 + 12, XSTRT - 25, XAXIS5 - 7);
        line(XSTRT - 25, XAXIS4 + 12, XSTRT - 25, XAXIS3 - 7);
        line(XSTRT - 25, XAXIS2 + 12, XSTRT - 25, XAXIS1 - 7);

        line(XSTRT - 25, YMIN7+2, XSTRT - 45, YMIN7+2);
        line(XSTRT - 25, YMIN5+2, XSTRT - 45, YMIN5+2);
        line(XSTRT - 25, YMIN3+2, XSTRT - 45, YMIN3+2);
        line(XSTRT - 25, YMIN1+2, XSTRT - 45, YMIN1+2);

        line(XSTRT-95, YMIN7+2, XSTRT - 115, YMIN7+2);
        line(XSTRT-95, YMIN5+2, XSTRT - 115, YMIN5+2);
        line(XSTRT-95, YMIN3+2, XSTRT - 115, YMIN3+2);
        line(XSTRT-95, YMIN1+2, XSTRT - 115, YMIN1+2);

        circle(XSTRT - 117, YMIN7+2, 2);
        circle(XSTRT - 117, YMIN5+2, 2);
        circle(XSTRT - 117, YMIN3+2, 2);
        circle(XSTRT - 117, YMIN1+2, 2);

        setcolor(11);
        arc(XSTRT - 124, YMIN7+2, 90, -90, 4);
        arc(XSTRT - 126, YMIN7+2, 90, -90, 6);
        arc(XSTRT - 128, YMIN7+2, 90, -90, 8);

        arc(XSTRT - 124, YMIN5+2, 90, -90, 4);
        arc(XSTRT - 126, YMIN5+2, 90, -90, 6);
        arc(XSTRT - 128, YMIN5+2, 90, -90, 8);

        arc(XSTRT - 124, YMIN3+2, 90, -90, 4);
        arc(XSTRT - 126, YMIN3+2, 90, -90, 6);
        arc(XSTRT - 128, YMIN3+2, 90, -90, 8);
```

```
        arc(XSTRT - 124, YMIN1+2, 90, -90, 4);
        arc(XSTRT - 126, YMIN1+2, 90, -90, 6);
        arc(XSTRT - 128, YMIN1+2, 90, -90, 8);
}
```

PC Software (Matlab)

```
function emgdisplay(filename);

% This gives what we want, the proper order for the data.
% open file for reading with little-endian byte order
FID2 = fopen(filename, 'r', 'l');
% read in 16 bit unsigned values into F2, and count how many values are read in
[F2, num_samples] = fread(FID2, inf, 'uint16');

% let 0FFF correspond to +1, and 0 correspond to -1;
% scale everything appropriately divide by 0FFF/2 = 2047;
scale_fact = 2047;     %change this for real code to 2047

F3 = (F2/scale_fact) - 1;

sample_rate = 1000;    % sampling rate in Hz
time_div = 1/sample_rate;
t = 0:time_div:((num_samples - 1)*time_div); % time values to plot samples against

F3T = transpose(F3);   % take transpose to match matrix dimensions for plotting
figure;        % opens new figure

plot(t, F3,'.',t,F3);

ST2 = fclose(FID2);
```