Passive Accousitc Detecion System

A Major Qualifying Project Report:

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

Boris Tolpin

Date: April 26, 2007

Professor Susan Jarvis, Project Advisor

Abstract

This report outlines a passive acoustic detection system (P.A.D.S.). This system is able to detect large objects (i.e. airplanes, tanks, motorcycles and cars) at various distances within its hearing radius despite the visual conditions. These detection signals are transmitted wirelessly to a display unit, which can potentially be located out of sight of the sensor. The P.A.D.S. is composed of two main units. The first of these units is the actual acoustic detection system, which is composed of a microphone, a microprocessor and a wireless transmitter. The second of these units is composed of a receiver unit, a microprocessor and three LED's which alert the user as to the level of detection reached, ranging from no detection to definite detection.

TABLE OF CONTENTS

Table of Figures

I. Introduction

1. Basic Design of the P.A.D.S.

The Passive Acoustic Detection System (PADS) has been designed using a microphone, two microprocessors, a wireless transmitter and receiver unit and two LED displays. The system is made of two separate modules. One of these modules detects sound levels and sends information to the other module, which receives the signal and turns on the proper LED's. The first of these two modules, the transmitter module, is the more complicated of the two to design and program. Figure one shows a top-level design of this transmitter module. The task of the transmitter on power up is to determine an ambient noise reading in the area in which it is operating. This is just one of the many functions that take place inside of the microprocessor. The rest of these functions will be discussed later.

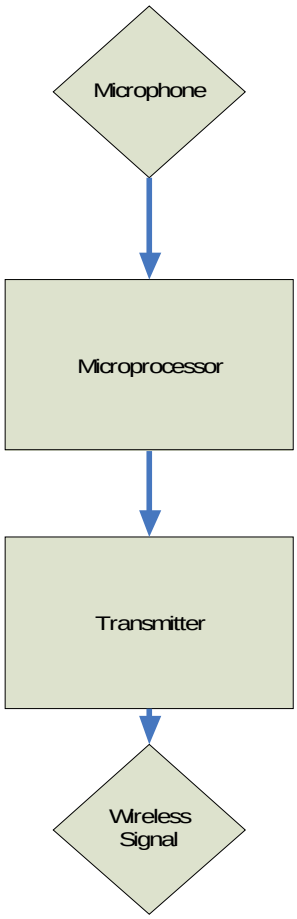

Figure 1: Top level design of the Transmitter unit

The microprocessor that is used in this design is microchip's PIC16F877a. The reason that this is used is that it has a sufficient amount of memory (8K word x 14 bits), it can be programmed in the programmer located inside of the Atwater Kent building (where the majority of the programming has been done), and that it has a large number of input/output pins (30). The programming of the PIC has been the most difficult, as well as the most time consuming aspect of the entire project. The PIC itself is responsible for

all of the “thinking” and “decision making” in the system. The rest of the components in the project all either receive data from the PIC or are intended to give data to the PIC.

Another of the components in the “pitcher unit” is the microphone, which gathers all of the noise readings in the area. These readings are then sent to an A/D converter so that they can be made into digital readings so that the PIC can compare them to the ambient noise levels. This A/D converter is an internal component of the 16F877a device. The PIC then must decide which LED it is that must be activated and send this information to the input pin of the transmitter, which wirelessly sends the data to the receiver unit. This will only work if the transmitter is properly configured, which is also the responsibility of the microprocessor.

The receiver unit of the device is considerably more basic then the transmitter unit. The job of the receiver module is to receive the data, send it to another PIC whose purpose is to turn on the proper LED. Figure 2 is a top-level block design of the receiver unit.

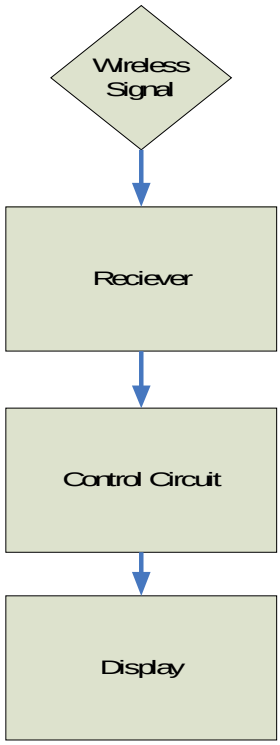

Figure 2: Top level design of the receiver unit

2. Responsibilities of the PICs

As previously mentioned the programming of the PIC inside of the transmitter has been the most difficult and time-consuming portion of the entire project. The reason that this is so complex is that the PIC is responsible for multiple processes in the overall system. As mentioned previously the PIC is initially responsible for finding the ambient noise level. This is done by first receiving a reading from the microphone and storing its numerical value to a variable when the P.A.D.S is initially turned on. Then when the next reading is received it will be offset and added to an offset version of the previous average. This is a concept called exponential averaging. This process will repeat itself until at least 255 readings have been averaged. One step in this process is shown mathematically in figure 3. This step is repeated for all 255 readings that are received. The descriptions of how these offsets are determined and more detail on this process is described in a later section.

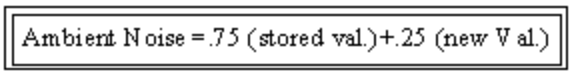

Figure 3: Mathematical form for ambient noise level where x = the new noise reading and Y = the previously stored noise reading.

Once the ambient noise level is found we can begin searching for noise readings above this reference level. The new noise level that we are now checking our readings against is called the threshold level. The PIC must continue to receive signals from the microphone block and anytime that a reading is above this threshold level the PIC must

send a message to the transmitter to send out a signal to turn on the orange LED, which represents some activity. Also when reading above the threshold is detected a counter is decremented. This is done until the counter reaches zero, and only then can the red LED be activated. This is an error checking mechanism, if the noise lasts for a specified amount of time we know that there has been a definite detection and a second signal must be sent out signaling that it is now time to turn on the red LED. The PIC then continues to sample until the noise level returns to normal, at which time the PIC sends out a signal to turn on the green LED.

Every time that a reading below the threshold level is recorded the PIC takes the reading and uses it to get a better estimate of the ambient level. This is how we ensure the most accurate and the most current ambient level is being used. Figure 4 shows the functions that will constantly go on inside of the control circuit, (the PIC). One additional responsibility of the PIC that is not shown in the figure is the smoothing of the results. The microphone, through the A/D converter will be sending more readings per second then are necessary, as a result only a certain number of readings per second will be used in any of the calculations. This will mean that it will take a specified amount of time to find the standing noise level after the P.A.D.S. is turned on, and during this time the device will obviously not output accurate data. Also it is important to note that we can not sample at just any rate. There must be at least twice the number of readings as the period of the signal that we are sampling in order to get an accurate result. Sampling at twice speed of the period of a signal is called sampling at the Nyquist rate. If we do not sample at this rate then it is possible that the largest point in a given signal will not be sampled and the results we get will be inaccurate.

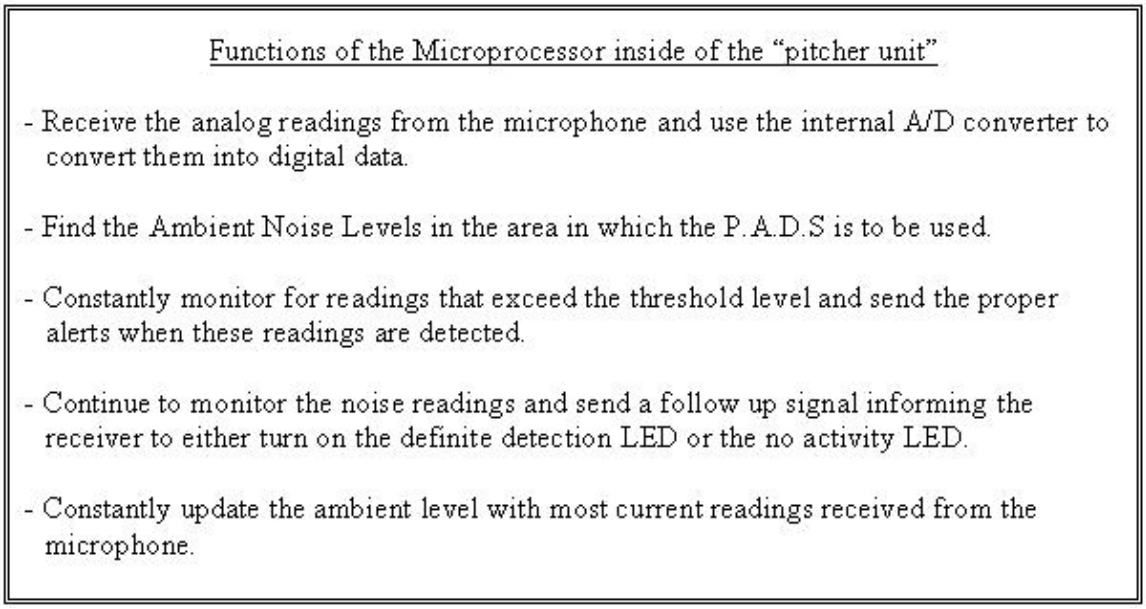

Figure 4: PIC functions from inside the transmitter

The microprocessor located inside of the receiver has considerably less to do in the system. Its job is to receive the signal from the receiver and determine what to do with it. For example, if the signal representing an orange light is received then the PIC will put the specified input pin to logic high. This will automatically turn the green LED off and turn the orange LED on. If for example the next received signal is to turn the red LED on then the PIC will turn the orange LED off and turn the red LED on. Figure 6 shows a diagram of what the final display looks like.

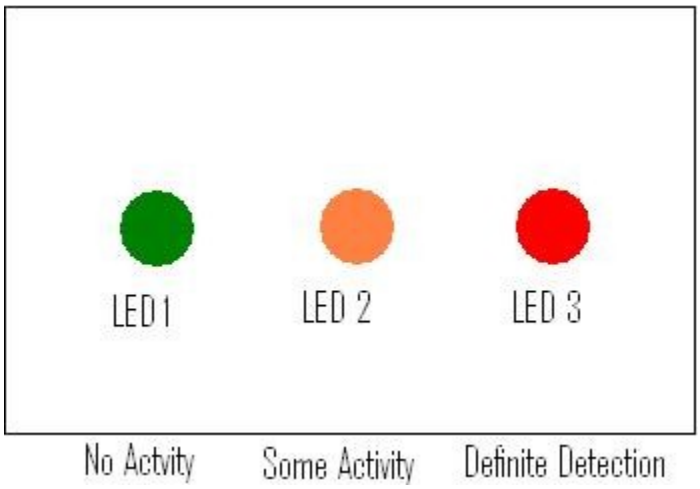

Figure 5: Diagram of Display

3. Testing of the Product

The final product, the smaller sub-blocks, and the individual components all have to be tested to make sure that the system is built correctly. If for example a sub-block is not acting as is intended, it is essential to know that the reason for this error is not a faulty part. Likewise, if the final product is not working correctly it is essential that all of the sub-blocks have been tested so that the entire design does not have to be reanalyzed. In order to optimize time during the construction and programming of the P.A.D.S. system, the test procedures have been developed prior to the actual construction of the device.

The most basic components to test are the actual parts. A basic test program can be flashed on to the PIC to make sure that the PIC is functional. This program can be as basic as setting all of the registers as output registers and then outputting a high byte. Either an oscilloscope or a function analyzer can be used to make sure that the proper data is being outputted. This test shows us that the PIC programmer that was used is working correctly, and that neither the PIC pins nor the internal circuitry has been damaged in transit. Since both the PIC in the pitcher and the PIC in the catcher are the same model both of them can be given the same test program and tested in the same manner.

The LED display was the easiest component to test. The actual LED's can be tested by placing them onto a test circuit with a resistor and a power source and seeing if they light up.

To test the microphone a noise sample has been recorded of a car driving down a quiet street. This sample has been later played for the small microphone, which is

attached to an oscilloscope. The curve from the oscilloscope is compared to the curve that is designed in Matlab of the original noise sample. These curves are virtually identical to one another so it is safe to say that the microphone is operational. These curves are not 100% identical because the microphone is recording different background noise then computer. In addition, the computer is recording the signal directly from the car while the smaller microphone is recording the signal of the car from speakers, which do not produce as clear a sound. This accounts for some of the differences between the two recordings, however that being said they are still very similar to one another.

The wireless transceiver can only be tested after the PIC has been tested. After both the transceiver and PIC are wired together, a test message is sent from the PIC to the transmitter for transmission. This test message is one byte long and alternates between high and low bits. An oscilloscope probe is attached to the transmitting pin of the transmitter. The signal being sent should be visible and should be 8 bits in duration, alternating between low and high. A second oscilloscope probe is attached to the pin on the PIC that receives the signal from receiver. This oscilloscope reading should be the same as the message that was sent. This test has not only be done when the catcher and the pitcher are right next to each other but also when the two units are on different floors or are at a distance of at least 400 feet away from one another. The data sheet on the transceiver specifies that the device should work at a range of 500 feet.

When testing the sub-blocks of the design the block that is the largest is the programming of the PIC. There are two PICs being used in the design and both must work correctly. Since it takes time to program the PICs and to place them into the completed circuit I have used the microchip simulation tools. This has sped up the

process of testing the PIC. In order to make sure that the ambient noise level is being found correctly the sample that was recorded of the car going was used. Since the PIC has an internal A/D converter, digital noise readings can be found from the sample. Once a final ambient reading was detected, it was trivial to compare it to the ambient reading that can be deduced from a limited number of samples by hand or graphically. This will be explained further in a later section.

In order to make sure that the PIC is taking the current noise readings and interpreting it correctly, the same recording that has been used to test both the microphone and the ambient noise level was used. This recording should start in the "green" area and as the car approaches should jump to the "orange" area and then after the noise has been present for approximately 2 seconds should jump to the "red" area. As the car gets further away from the microphone, the sound should diminish and the alert level should go directly back to the "green" level. Determining what alert should be activated and then transmitting this information is the job of the pitcher, however turning on these lights is the job of the catcher.

Once we know that the data is being received by the receiver, we must make sure that the PIC is getting the data and that it is interpreting it appropriately. Before we can make sure that the PIC is getting the proper data, we must first make sure that the PICs interpretation of the data is accurate. We can do this using the simulation tools. We can simulate the data being received by placing the proper signals in the registers that in our actual PIC would be connected to the receiver and making sure that the proper lights are being turned on. For example if we receive a signal to turn on a "green" LED we should see in the simulator that the pin that the green LED is attached to is pulled high. We

should also be able to see the change in signal if we change the input data to turn on an Orange LED or a red LED.

The final product test is the most basic of all of the tests. It is extremely similar to the microphone test. When a car drives past our device on a quiet night, we will be able to see the display switching from the “green” level, to the “orange” level, and then to the “red” level and then back to the “Green” level after the car is out of range. This had to be done with different size cars, which will make different amounts of noise as they drive past the device. The LED display is placed approximately 400 feet away from the Pitcher to make sure that the wireless range is as advertised.

4. Parts to be Used

The PICs that are used are the PIC16F877a, produced by microchip. Microchip offers free samples of there products, as well as free shipping and handling. The wireless transmitter and receiver pair that can be interfaced with the microprocessors can be found at sparkfun.com. This product is numbered RF-KLPA-315, and it costs approximately $15.00. At least three sets of transmitter/Receiver packages were ordered as they are extremely easy to damage. The range of this product according to its data sheet is 500 feet. This provides an adequate distance for anyone who will be using the P.A.D.S. to be completely out of sight and still be able to monitor the area. The microphone that was used is made by Knowles acoustics and is sold at Digikey.com. Digi-key offers products such as the Knowles Acoustics BL-21785, which is an extremely small high sensitivity microphone. This would be the potential high end of the microphone spectrum and it costs upwards of $75 placing it well within the specified budget of $150. It would make an excellent upgrade for the second generation of this device. The microphone that was used in this prototype however is the 9745APA-1. This is an omni directional microphone. One of the disadvantages of using this microphone is that it will not put out much voltage and for this reason there is a considerable amount of analog circuitry that amplifies the signal to a level that the PIC is better equipped to handle. This analog circuitry also peak detects the signal so that there is no fear of getting a bad reading. This will be discussed at length in a later section.

The particular microphone used for this device costs less then $2.00 and as a result multiple microphones were purchased. The rest of the parts were purchased in

Worcester Polytechnic Institute's ECE shop. Figure 6 is a picture of the wireless transmitter and receiver set that was discussed previously. As mentioned, these two parts are sold together as a set. The quarter gives an accurate approximation of the size of the product. Since one of the benefits of the P.A.D.S. system is that it is small enough to conceal easily, having a transceiver set that is this small is a major benefit.

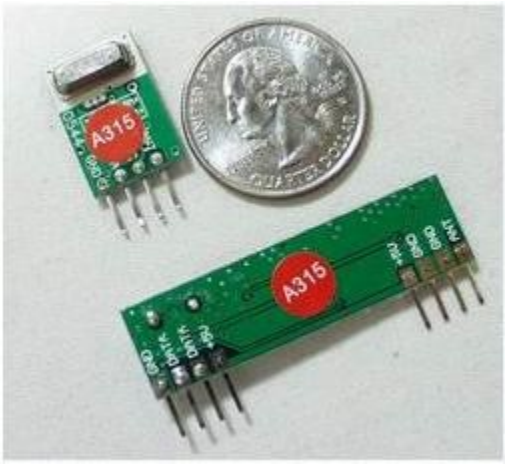

Figure 6: Picture of the wireless technology

II. Noise Level Readings & Microphones

1. Describing noise levels

A microphone can represent a constantly changing noise level by outputting a constantly changing voltage level. Noise levels however are not measured with voltage; they are measured in decibels (DB). “One decibel is one tenth of a Bel, named for Alexander Graham Bell.” A decibel describes a ratio between two levels, either a reference and a voltage level, or a reference and a power level. It is not however a linear ratio, it is an exponential ratio. This is shown in figure 7. Figure 8 shows that the decibel equation for both voltage and power are equal.

$$1.\ X(Db) = 10\log\left(\frac{P_1}{P_{REF}}\right)$$

$$2.\ IF\left(10^x = \frac{P_1}{P_{REF}}\right) Then$$

$$3.\ 10^1 = 10 = \frac{P_1}{P_{REF}}\ When\ (X = 1)$$

$$4.\ 10^2 = 100 = \frac{P_1}{P_{REF}}\ When\ (X = 2)$$

$$5.\ 10^3 = 1000 = \frac{P_1}{P_{REF}}\ When\ (X = 3)$$

Figure 7: This figure shows mathematically that the DB equation is exponential.

Line 1 in figure 7 is the DB equation in terms of power, and X is the numerical measure of the DB level. P_1 is the measured power and P_{REF} is the reference power that is necessary for determining a DB level. Line 2 simply shows line 1 manipulated slightly using basic logarithmic rules. By plugging in values for X we can see that the resulting power ratio goes up by factors of 10. We can also use the equations in lines 3 through 5 to demonstrate that because P_{REF} is set to a specific reference level we would need to raise P_i (i=1,2,3) by a factor of 10 to raise X by 1.

$$1.\ Db = 20\log\left(\frac{V_1}{V_{REF}}\right)$$

$$2.\ Db = 10\log\left(\frac{P_1}{P_{REF}}\right)$$

$$3.\ P = \frac{V^2}{R}$$

$$4.\ \frac{P_1}{P_{REF}} = \frac{\frac{V^2}{R}}{\frac{V_{REF}^2}{R}} = \left(\frac{v}{V_{REF}}\right)^2$$

$$5.\ Db = 10\log\left(\left(\frac{v}{V_{REF}}\right)^2\right)$$

$$6.\ Db = 2 \times 10\log\left(\left(\frac{v}{V_{REF}}\right)\right)$$

$$7.\ Db = 20 \times \log\left(\left(\frac{v}{V_{REF}}\right)\right)$$

$$8.\ Db = 20 \times \log\left(\left(\frac{v}{V_{REF}}\right)\right) = 10\log\left(\frac{P_1}{P_{REF}}\right)$$

Figure 8: Mathematical proof that the DB *equation in terms of voltage is equal to the* DB *equation in terms of power.*

There are myriad of reasons as to why understanding the concept of DB is important. On of these reasons is that the data sheet for the microphone being used in the P.A.D.S. system describes its output (as most microphones do) in DB. The “sensitivity level” of the microphone is -41 DB with a 3 DB error. What this means will be described in a later section but for now it is important to understand that a negative DB level simply means that the voltage output will be less then 1 Volt. We can use similar equations to the ones in line 2 in figure 7 to show this. We simply see that if $10^{-41} = \frac{V}{V_{REF}}$ and V_{REF} is set then V must be less then 1 because a base to a negative exponent is simply $\frac{1}{base^{exponent}}$.

Sounds	dB
Rocket Launching	180
Jet Engine	140
Thunderclap, Air Raid Siren 1 Meter	130
Jet takeoff (200 ft)	120
Rock Concert, Discotheque	110
Firecrackers, Subway Train	100
Heavy Truck (15 Meter), City Traffic	90
Alarm Clock (1 Meter), Hair Dryer	80
Noisy Restaurant, Business Office	70
Air Conditioning Unit, Conversational Speech	60
Light Traffic (50 Meter), Average Home	50
Living Room, Quiet Office	40
Library, Soft Whisper (5 Meter)	30
Broadcasting Studio, Rustling Leaves	20
Hearing Threshold	0

Figure 9: DB levels of certain objects

It is also important to note that a lot of objects create very specific DB levels. Figure 9 shows some noise producing objects and the DB levels that they produce. While the table is not exact as all trucks for example make slightly different noises it gives us a good idea of how much noise 90 DB represents, for example.

2. Microphone Specifications

The microphone that I have decided to use for the PADS system is the MD9745APA-1. The reason that this particular microphone is good for that job that it is intended to do is that it is omni-directional, it can receive a signal with frequency in the range of 100 HZ– 10 KHZ, and that it has a sensitivity of approximately -41 DB. What this means is that it can detect changes in noise that are much smaller then 0 DB. Also since the expected amplitude of the sound waves that I am looking for are approximately 3 KHZ the range of possible amplitudes is sufficient. The datasheet for this microphone is located in appendix A. It is also important to note that the microphone is extremely simple to wire as is shown in the datasheet.

1. Microphone circuit

It is important to talk about the circuit that has to be in place between the microphone and the PIC. The role of this circuit will be to trim the negative part of the microphone signal, as well as amplify it and peak detect. This is a process called smoothing. Figure 10 shows the circuit that was used to do smooth the output signal from the microphone, and figure 11 shows the signal after it has been clipped and low pass filtered so that the PIC can read it accurately. The input signal that was smoothed in this example is a square wave with a frequency of approximately 3 KHZ and peak to peak amplitude of approximately 10 milli volts. As is seen in the circuit the gain coming out of the amplifier is 510. This means that the signal passing through resistor U3 is 510 times larger then the signal passing through the resistor U2. We can figure out what the gain of the circuit will be by using the game equation for an operational amplifier. The equation is resistor2 divided by resistor 1. In this case resistor 2 = 510 K ohms and resistor 1 = 1 K ohm, so the gain is equal to 510K/1K which equals 510.

The diode shown is used to clip the signal so that it can not be negative and then the low pass filter shown smoothes this signal. The amplitude of the wave in the picture is approximately 4 volts. The reason that the amplitude is so high is that after the signal was amplified the amplitude on the original signal was raised to simulate a noise level being raised. We could also have done all of the work on the signal in software, but it was decided against.

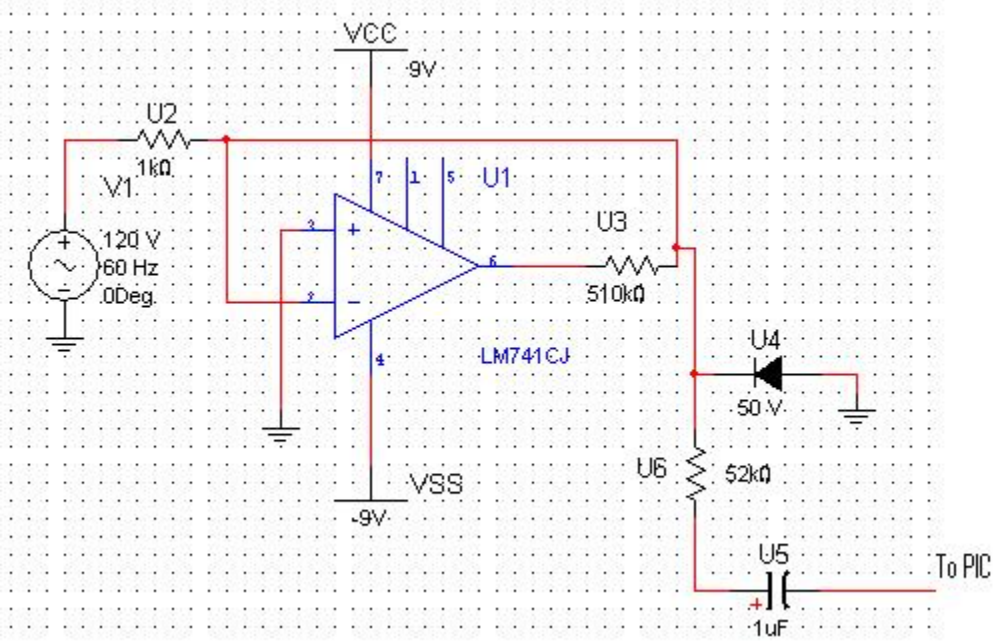

Figure 10: The circuit used to peak detect and amplify a square wave with an amplitude of approximately 10 microV

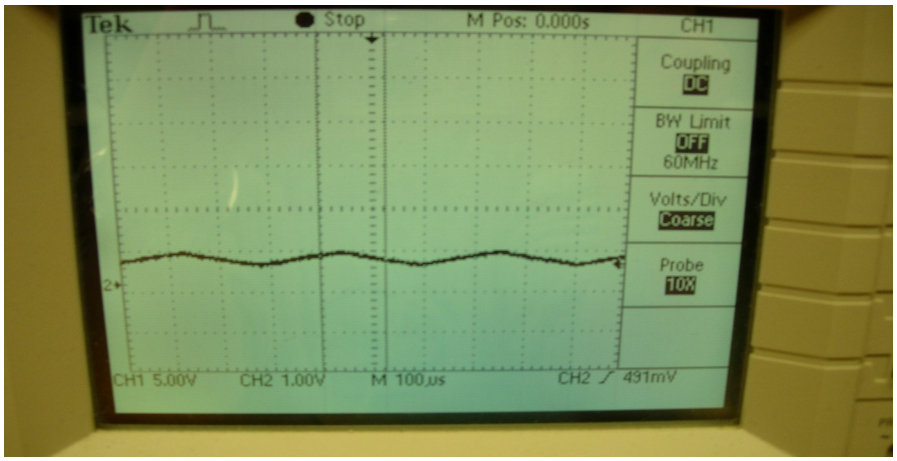

Figure 11: The result of the Peak detection and amplification and low pass filtering.

2. Microphone circuit

The microphone is one of the simpler components to test. The microphone datasheet describes how the microphone must be wired. The output of the microphone was simply attached to an oscilloscope and the top of the microphone was scratched to test whether or not it would respond to the noise. After it passed this test, music was played directly into the microphone, and again the microphone outputted showed the sound wave. This is how we can tell that the microphone is working properly.

III. A/D converter

1. Describing The Analog To Digital Converter

All of the processing that is done inside of the microcontroller requires a digital signal. As a result, an analog to digital (A/D) converter is needed to convert the analog input signal that the microphone block produces to the digital signal that is needed for all of the processing and logic. One of the biggest advantages in using the PIC16F877a microcontroller is that it contains an internal analog to digital converter, and that the datasheet does an excellent job in explaining exactly how to use it.

According to the datasheet, there are four major “registers” that are allocated to using the A/D converter. Two of these registers are used in the setup and operation of this device, and the other two are used as a place to store the output. The A/D convert zero register (ADCON0) and the A/D convert one register (ADCON1) both contain eight bits that need to be filled with the appropriate values in order for the A/D converter to operate as intended. The A/D result high register (ADRESH) and the A/D result low register (ADRESL) are used internally by the device to store a digital result when the process is complete. The reason that there are multiple “result registers” is that each register can only store eight bits and the PIC contains a ten bit A/D converter, which means that it converts an analog signal to a ten digit binary number. As a result one eight bit register can not store this output by itself and a second register is needed. One helpful option that we are provided with is the choice of which way to justify these 10 bits. This means that

we can either store the highest eight bits in one register (AHRESH) and the lowest two bits in the other register (ADRESL), or we can store the lowest eight bits in the register ADRESL and the highest 2 bits into ADRESH. This is the kind of information that we must provide to the A/D converter in the ADCON0 and ADCON1 registers. In the section entitled “Programming the Analog to Digital Converter”, the rest of the information that we must provide to the A/D converter will also be described, as well as the manner in which we must provide it.

2. Programming the Analog to Digital Converter

As mentioned earlier, one of the biggest advantages in using the internal A/D converter of the PIC microcontroller is the ease with which it can be programmed. The data sheet clearly outlines what each of the bits in the two control registers (the ADCON0 and ADCON1) is responsible for and it is simply a matter of setting these bits appropriately. Also the A/D converter has high and low voltage references that are software selectable to some combination of VDD, ground or two other potential inputs RA2 or RA3. These are also selected, as everything else is, by the ADCON0 and ADCON1 registers.

In order to perform a conversion the following process must be followed. First the ADCON0 and ADCON1 registers must be filled appropriately. Then we must delay for an “Acquisition time”. This is to give the A/D converter time to initialize. At this point the device should be on but not yet ready to be used as it should not be enabled. We are specifically warned against both turning on the device and enabling it at the same time in the data sheet. Our next step after we have implemented our software delay is to enable the device, which starts the conversion process. We enable the device by setting the “GO/DONE” bit, which is bit 2 in ADCON0. Since I have used assembly instead of C++ to program the PIC figure 12 shows the assembly language code for setting this bit. The X’s in the figure are simply holding the place of other numbers which are not important when setting the GO/DONE bit.

```
Movlw  B'XXXXX1XX' ; moves a 1 into the second bit of register W
Movwf  ADCON0;   moves the register W into ADCON0
```

Figure 12: Assembly code that sets the second bit of the register ADCON0

Once the "GO/DONE" bit is set the conversion will begin. Once the conversion is finished the "GO/DONE" bit will automatically be cleared (it will be 0). This is extremely useful as all we have to do is poll this bit to see if it is 0, and when it is we know that the conversion is completed and the results of the conversion are now in ADRESH: ADRESL.

One question that is still unanswered is how long the mandatory acquisition delay must be, and why an acquisition time is needed in the first place. We can find the answers to these questions in the PIC16F877a datasheet which tells us that "the charge holding capacitor (CHOLD) must be allowed to fully charge to the input channel voltage (in our case 5 volts) level". The datasheet also provides us with an equation to figure out the acquisition time delay. It then proceeds to show us how this is solved. This is shown in figure 13. As we can clearly see a min acquisition time of approximately 20 microseconds must be delayed for. This can be done in software by the code shown in figure 14.

EQUATION 11-1: ACQUISITION TIME

TACQ = Amplifier Settling Time + Hold Capacitor Charging Time + Temperature Coefficient

= TAMP + TC + TCOFF
= 2 µs + TC + [(Temperature – 25°C)(0.05 µs/°C)]
TC = CHOLD (RIC + RSS + RS) In(1/2047)
= - 120 pF (1 kΩ + 7 kΩ + 10 kΩ) In(0.0004885)
= 16.47 µs
TACQ = 2 µs + 16.47 µs + [(50°C – 25°C)(0.05 µs/°C)
= 19.72 µs

Note 1: The reference voltage (VREF) has no effect on the equation since it cancels itself out.

2: The charge holding capacitor (CHOLD) is not discharged after each conversion.

3: The maximum recommended impedance for analog sources is 2.5 kΩ. This is required to meet the pin leakage specification.

Figure 13: Process for finding the required acquisition time, along with notes from the Datasheet

```
;****************** Required Aquisition Delay
***********************************
                    MOVLW   H'5'
                    MOVWF       del
LOOP
                    NOP
                    NOP                                     ; 25 micro second
delay
                    DECFSZ      del, 1
                    GOTO LOOP
```

Figure 14: Code used for waiting required acquisition delay

The last part of the A/D converter that is left to discuss is how the rest of the ADCON0 and ADCON1 registers are to be set. For the most part the datasheet does a good job of describing this. As a result only few of the parameters will be discussed instead of describing how to set each and every bit. The necessary datasheet pages will however be attached to this report in appendix A so that it will be easily accessible.

Immediately after this in appendix A, the code that runs the A/D converter is shown so the reader will be able to see which bits were set and which bits were cleared.

One of the more important set of bits to understand how to set is the bits used to set the reference voltages. The reference voltages that are employed are the V_{CC} and ground voltages that are already used to power the PIC. In order to do this "1110" must be placed into the bits entitled PCFG<3:0> in the ADCON1 register. This comes directly from the chart in figure 15 in the datasheet. The reason that the row with 1110 is the correct row to use is not only because the reference voltages used are V_{DD} and V_{SS} (ground), but also because the "AN0" pin on the microcontroller is the pin that the analog input is connected to.

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O
C/R = # of analog input channels/# of A/D voltage references

Figure 15: Chart from datasheet describing how to select the reference voltages.

Another important parameter that must be set in software is the clock conversion time per bit. According to the datasheet “the A/D conversion time per bit is defined as TAD. The A/D conversion requires a minimum of 12 TAD per 10 bit conversion”. What this means is that a certain amount of time must be allotted for each bit that must be converted. The value in TAD is this amount of time. This value will vary depending on the frequency of the analog signal being examined. Clearly the higher the frequency of the analog signal the smaller TAD will need to be. There are seven possible values for TAD that the converter can assign. These values are shown in the leftmost column in figure 16. The rest of the table (which is displayed in the datasheet) shows how to fill in the values ADCS<2:0> which represent bits in the registers ADCON0 and ADCON1. Because the P.A.D.S system will be monitoring large vehicles that are known to produce a sound wave with a frequency no larger then 5 KHZ, the value 101 will be assigned to ADCS<2:0>. The rest of the values of ADCON0 and ADCON1 are easily deciphered from the datasheet and will not be discussed here for that reason. As mentioned earlier however, the necessary pages of the datasheet are located in appendix A as is the code that was used for programming and so the reader can easily examine the values that were used.

TABLE 11-1: TAD vs. MAXIMUM DEVICE OPERATING FREQUENCIES (STANDARD DEVICES (F))

AD Clock Source (TAD)		Maximum Device Frequency
Operation	ADCS2:ADCS1:ADCS0	
2 TOSC	000	1.25 MHz
4 TOSC	100	2.5 MHz
8 TOSC	001	5 MHz
16 TOSC	101	10 MHz
32 TOSC	010	20 MHz
64 TOSC	110	20 MHz
RC(1, 2, 3)	x11	(Note 1)

Note 1: The RC source has a typical TAD time of 4 µs but can vary between 2-6 µs.

2: When the device frequencies are greater than 1 MHz, the RC A/D conversion clock source is only recommended for Sleep operation.

3: For extended voltage devices (LF), please refer to **Section 17.0 "Electrical Characteristics"**.

Figure 16: Table from datasheet showing how to select values for ADCS<2:0>

It is also worth noting that the method described in this section is not the only possible method for using the internal A/D converter. The device can also be used by setting and reading certain interrupt flags but since that is not the way in which it was used in the P.A.D.S system it will not be described in this document.

3. Testing The Analog To Digital Converter

In order to test the A/D converter, that was programmed, a testing circuit was assembled. The job of this circuit is to prove beyond any reasonable doubt that the A/D converter is functioning properly. This can be accomplished by simply converting an analog signal whose digital value is previously known such as 5 volts DC. The digital signal of 5 Volts DC after conversion should be '11111111'.A basic laboratory power source can be used to provide this input signal.

The datasheet for the PIC microcontroller writes that the A/D is 10 bits. This means that there are a total of 2^{10} possible subdivisions that the digital signal can fall into. If we convert this to a voltage, each subdivision will be represented by $\frac{5}{2^{10}} = .0048\,Volts$. In addition, because the microprocessor stores these values as binary numbers each subdivision will be represented as a 10-bit binary number. Eight of these bits will be stored in one 8-bit register, and the other two will be stored in another 8-bit register. Because the volts per subdivision are so small (.0048) and our ability to filter our noise is limited, it is safe to say that we will not get an exact conversion out of the A/D converter. For this reason, we cannot trust the lower few bits of our results. Knowing this, only the highest 8 bits in the ADRESH register were stored and the lowest 2 bits are completely disregarded, as they are insignificant. Once these 8 bits are found it is possible to use a 'mov' command to output these bits using a free I/O register (PORTB).

Eight LED’s have connected to the pins that represent PORTB on the PIC. This way as the voltage coming out of the power supply is adjusted, we can see the different LED’s changing and can make sure that the A/D converter is functioning properly. When the power source is set to 5 volts all eight LED’s are lit and we know that the A/D converter is functioning properly. The A/D converter was also tested by lowering the input voltage to 2.5 volts and seeing all of the LED’s except for the one in the most significant bit light up.

IV. Setting A Threshold

After using the A/D converter to convert the analog signal from the microphone to a digital signal we must decide what to do with this signal. The first step in this process is to determine if the newly acquired signal tells us that there is a potential vehicle approaching. In order to do this we must determine how large signal's amplitude must be in order for there to be a possible detection based on the previous ambient noise level in the surrounding area. After careful research and experimentation, and after taking into account the advice of Professor Jarvis, it was decided to set the threshold at twice the ambient noise level of the surrounding environment. This means that if the reading out of the A/D converter for the ambient noise level were approximately 1.5 volts (out of a possible five) then it would take a reading that produced a level of three volts from the A/D converter for there to be detection.

The fact that the ambient noise level was simply doubled the, along with the fact that all of the programming was done in assembly made this an extremely easy segment of code to write. All that was needed was simply to shift the value of the "stored" value (ambient noise level) to the left one slot, which in binary multiplies the number by2. Figure 17 illustrates this concept. Figure 18 shows the code that sets the threshold for the PADS device. The variable "lights" is where the threshold value is stored. It is also worth noting that if doubling the stored value gives a result larger then can be attained from the A/D converter then by default "lights" is set to the highest possible value, which in binary is 11111111 which is equal to 255 in decimal or FF in hex.

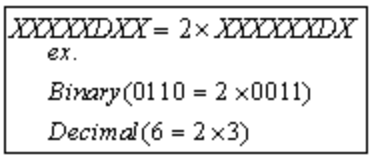

Figure 17 is an illustration of how shifting a binary number to the left affects the number

```
threshold
                    BCF    STATUS, C;    Clear carry flag
                    RLF    STORED, 0;    double stored value and save in w
                    BTFSC  STATUS, 0;    skip next instruction if C bit in status is
clear
                    MOVLW  B'11111111';
                    MOVWF  LIGHTS    ; move FF into "LIGHTS" if stored is
Greator than 10000000, else move twice STORED value into LIGHTS.
                    GOTO   compare
```

Figure 18: Code to set the Threshold

V. Finding The Average

Finding the ambient noise level is one of the most important functions of the microcontroller in the pitcher. An exponential average was used to find the ambient level. The formula that was used for this places seventy-five percent of the weight of the average onto the stored value, and twenty five percent of the average onto the most recently converted value from the A/D converter. Figure 19 shows the formula that has been described along with the inputs used to test the averaging section.

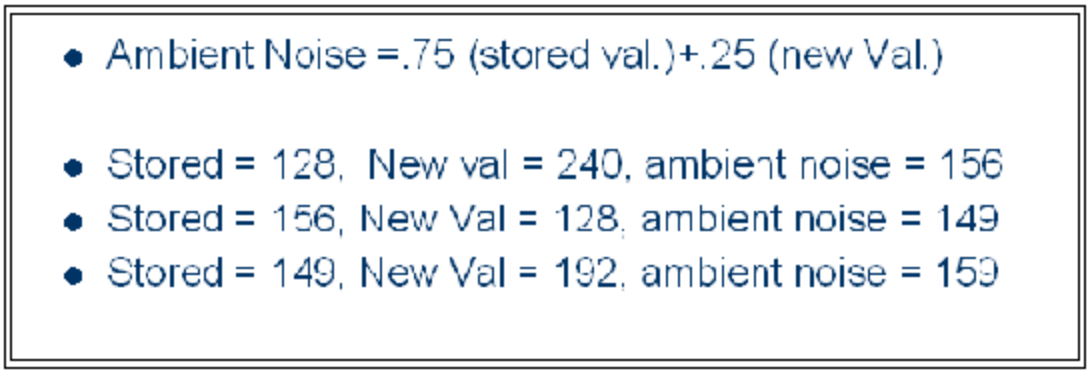

Figure 19: Averaging formula and example average calculations

Using this formula is beneficial for two main reasons. The first is that if the noise conditions change in the area in which the device is being used, the PADS system will adjust extremely quickly as the newest results are weighted significantly higher then the individual older results. In addition, this formula is extremely easy to program. The reason for this is that in order to divide a binary number by two all we need to do is shift the digits to the right one place. If we divide a number by 2 then we have 50 percent of

the original number. If we divide by 2 again then we can get 25% of the original number. If we subtract this from the original number then we get 75% of the original number. We can follow this exact process with the new number to get 25% of the new number. If we add these two results, we find the new weighted average. The variable results is the new value out of the A/D converter, and the variable stored is the previous weighted average. It is important to note that stored starts with an initial value of 65 so that it will take less repetitions to find a trustworthy ambient noise level. The code for finding the ambient noise level is located in appendix A.

VI. Comparing to the Threshold

Once we have a threshold set, we now must compare our most current results to it in order to see if this threshold point has been crossed and it is time to turn on the orange LED. What makes this difficult however is that there is no compare function in PIC assembly code. Therefore, the only way to compare two numbers is to subtract one number we are comparing from the other and see if a carry is required. If no carry is required then the first number is bigger then the other, and if a carry bit is required then the other number is bigger then the first. This way we can compare two numbers to see which one is larger. Knowing this, we simply subtract the newest result from the variable lights. If “lights” is larger then we know that the threshold has not been crossed and we use the result to update our ambient noise level. If “lights” is smaller then we go and turn on the orange LED and we do not update the average as this result does not help us find the ambient noise level in the area in which the PADS device is being used. Figure 20 is a diagram of what has just been described.

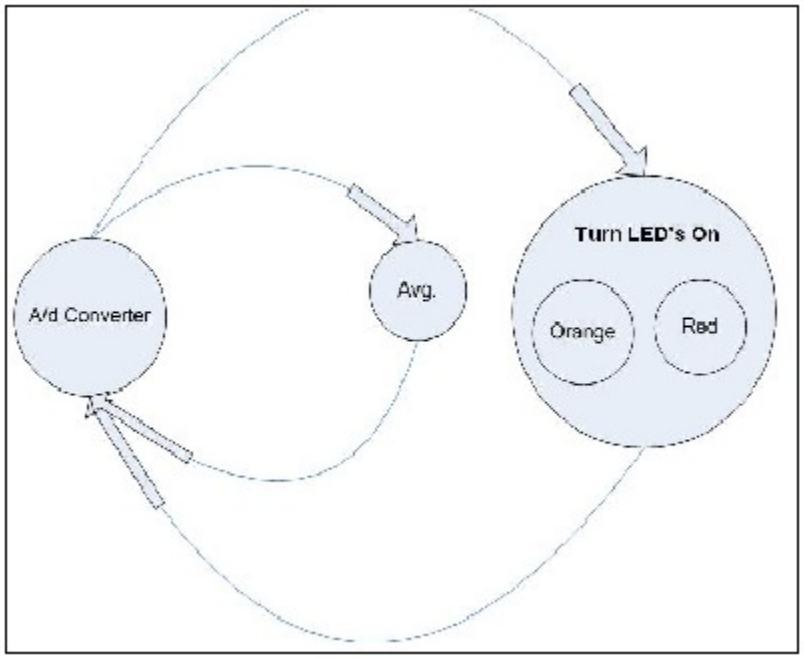

Figure 20: Diagram describing Decision making in PIC

VII. Orange LED

The orange LED is turned on immediately after a result from the A/D converter is larger then the threshold. The previously described “compare” function takes care of deciding when to turn this orange LED on. Once the orange LED is activated, some way of keeping track of how long it has been on is needed. The variable Skip_orange was used for this task. Every time the function Orange has been entered Skip_orange was decrement, the second least significant bit in port B was set and then a new value was taken from the A/D converter. Setting Port B to a value represents wireless transmission in my code. This will be elaborated on further in a later section. Once the value in Skip_orange has reached zero (it stars at 255) then it is assumed that the orange LED has been on long enough for there to be a definite detection and we go into the function Red.

VIII. Red LED

The red LED section was perhaps the smallest and easiest to code. Skip_orange was reset to the original value of 255 for the next time that orange is entered, and the third least significant bit of port B was set. The last step was to simply call a function called New_Val_Red. This function will be explained in a later section.

IX. Green LED

This function simply turns the green LED back on when the compare function decides that there is no longer a value from the A/D converter that is larger then the threshold value. It does this by simply setting the least significant bit of port B. This function also resets the variable Skip_orange just incase it got decremented in the orange section but the red section was never entered. This can happen if for example if lightening strikes. The PADS system would register the initial noise and enter the orange state, but there would be no more noise following the thunder so instead of entering the red phase it would go back to the green phase.

X. New_Val_Red

This function is intended to prevent the program from entering into the compare function when a red LED has been activated. The reason that we do not want to enter the compare function is that there are only two possible places that compare can go to. One is average, which if the result is larger then the threshold we do not want to enter, and the other is orange. If a red LED is already on then we do not want to enter orange but instead want to enter the red section. New_Val_Red simply makes sure that the call compare command is skipped after the new value from the A/D converter is acquired and instead does its own comparison and either calls the green section or the orange section.

XI. Transmitter

The only other segment left to talk about in the pitcher is the transmission segment. The way that a transmission segment is supposed to work, as described by the data sheet is that we are first supposed to set the baud rate to the proper value. Then we are supposed to set the TXSTA and RCSTA based on the descriptions of these registers provided to us in the data sheet. This description is available in appendix A. Once this is complete, we are simply supposed to be able to store our desired transmission value into an 8-bit register called TXREG. Then we should simply be able to delay until we are told that the transmission is complete. We are told that transmission is complete when bit TXIF in the PIR1 register is cleared. After doing some research it was discovered that this is not all actually must be done in order to transmit over 1 wire. There is a set of 1-wire routines, as they are called that we must also run in order to transmit data. After countless attempts however, transmission of data from one PIC to the other remains unsuccessful. Instead, the two units were connected into one for the purposes of testing the rest of the features of the PADS device. The attempted code for the transmission is located in Appendix C along with the descriptions of how to set the most important registers.

XII. Catcher

Since the transmitter was not working properly this code was never implemented. However, if the transmission code had worked properly this code would have been used to read the transmission and turn on the proper LED. The way in which it worked was that it simply tested to see if the least significant bit was set, and then if it was, turned on the proper LED. It did this for all three possible LED's. The code to show this is located in Appendix B.

XIII. Conclusion

The P.A.D.S. would be of interest to private citizens as well as to any corporation that uses surveillance equipment such as the military and security companies for example. One of the main benefits of this device is that it could eventually allow for considerably fewer people to watch a large perimeter. The first generation of the P.A.D.S. system however, does not fully have this capability because it transmits using a single RF frequency. At this point multiple transmitters can not interact with one receiver. This is only for prototyping purposes. However, the next device in the line could come with a slightly more complex transceiver package that would allow multiple transmitters.

Other additions that could potentially be added to the PADS II would be the ability for the user to communicate with the pitcher. A possible button on the catcher could be added that would allow who ever was using the device the ability to check if the pitcher was still operational. When this button on the catcher was pressed the pitcher could blink all of the LED's for example to show that it was still functional. Another addition to the PADS II could be an LCD display instead of an LED display so that the user could have some idea of just how far over the threshold level a detected signal could be.

Textron makes a device similar to the PADS II system. This device is not however available for commercial use as it is equipped with a grenade launcher, just in case someone not only wants to know if a vehicle is present but also wants to remove the vehicle from the area.

After spending a consider amount of time on this project it is fair to say that the PADS system has a majority of the functions it was intended to have. While the PADS was intended to have wireless transmission and at this point does not, it is useful in identifying noise signals that are above the threshold level. A major difference that the PADS II system would have is that it would be programmed in C code instead of in assembly, as fewer and fewer people are using assembly in industry. Also C is a considerably more user friendly language then assembly is. The only thing that made using assembly bearable was the simulator that was available. This allows the user to set variables as they please and simulate the desired program step by step. This served as an excellent way to pin point errors in code, and correct them. Figure 21 shows where the simulation tool allowed for the manipulation of variables.

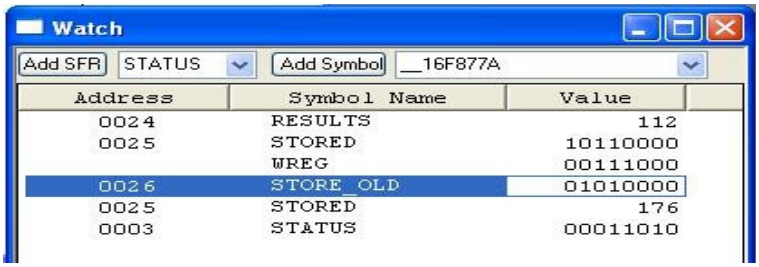

Figure 21: Tool that allows the user to manipulate variables during simulation

It is also important to note that the only references that were used for this project were the datasheets of the actual components, some of parts of these data sheets are attached while other parts are too long to attach.

Appendix A:

Datasheet pages describing how to fill the values of ADCON1 and ADCON0:

REGISTER 11-1: ADCON0 REGISTER (ADDRESS 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits

000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Channel 7 (AN7)

Note: The PIC16F873A/876A devices only implement A/D channels 0 through 4; the unimplemented selections are reserved. Do not select any unimplemented channels with these devices.

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit

When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit

1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 11-2: ADCON1 REGISTER (ADDRESS 9Fh)

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit

1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	Vss	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	Vss	7/1
0010	D	D	D	A	A	A	A	A	VDD	Vss	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	Vss	4/1
0100	D	D	D	D	A	D	A	A	VDD	Vss	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	Vss	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	Vss	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	Vss	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	Vss	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O

A/**D conversion code**:

```
 recieve_data
; ***************** Configure Device ************************************
                    CLRW

                    BCF     STATUS, RP1     ;     Select bank 1
                    BSF             STATUS, RP0

                    MOVLW   B'00001110'       ; (see datasheet for specifics on ADCON1);
                    MOVWF   ADCON1

                    BCF     STATUS, RP1     ;     Select bank 1
                    BCF             STATUS, RP0

                    MOVLW   B'01000001' ;1 in LSB signifies the device is on/inactive
                    MOVWF    ADCON0
```

```
;****************** Required Aquisition Delay *******************************
                      MOVLW   H'5'
                      MOVWF   del
LOOP
                      NOP
                      NOP                                          ; 25 micro second
delay
                      DECFSZ  del,1
                      GOTO    LOOP
;******************* Start Conversion ***************************************


                      MOVLW   B'01000101'
                      MOVWF    ADCON0
;******************Poll to see when done ************************************
LOOP1
                      NOP
                      BTFSC   ADCON0, 2         ; polling
                      GOTO    LOOP1

                      MOVLW   H'0'
                           MOVWF   SKIP
                   ; variable skip is set to 0 so that receiver data is not called again

                      CLRW
                      MOVLW    B'11111111'
                      ANDWF    ADRESH, 0;     ; copy results of ADRESH to W

                      MOVWF    RESULTS              ; store results

                      MOVLW   B'00000000'           ;0 in least sig bit turns the A/D
converter off
                      MOVWF   ADCON0

                      return
;***********************************************************************
average
                      BTFSC   SKIP_first_time,4
 ;   skip decrement if fifth digit from lsb is 0 (makes sure it runs adequette amount of
times to find avg)

                      DECF    SKIP_first_time,1
;    Decrement SKIP_first_time and save result to SKIP_first_time
                      MOVLW   B'11111111'
                      ANDWF   STORED, 0   ; copy results to W
                      MOVWF   STORE_OLD   ; copy value of stored to store_old

                      BCF     STATUS, C      ;     Clear carry flag
                      RRF         STORE_OLD, 1   ; STORE_OLD is divided by 2.
                      BCF     STATUS, C
;     Clear carry flag
                      RRF         STORE_OLD, 0
 ; STORE_OLD is divided by 2 so in total it has been divided by 4 and saved into the w
register
                      SUBWF   STORED, 1
;STORE_OLD divided by 4 (.25*stored) is subtracted from stored.
(STORED-.25*STORED=.75*STORED=STORED)

                      BCF     STATUS, C;     Clear carry flag
                      RRF         RESULTS, 1   ; RESULT is divided by 2.
                      BCF     STATUS, C      ;     Clear carry flag
                      RRF         RESULTS, 0
  ; STORE_OLD is divided by 2 again = RESULT/4=.25*RESULT and saved into the W register.


                      ADDWF   STORED,1
;  stored = .75*stored+.25*result = weighted average
            BCF       STATUS, C      ;     Clear carry flag

                      movlw B'11111111'
                      movwf SKIP_orange
                      movlw B'00000001'
                      movwf PORTD
```

```
                    MOVLW   H'1'
                    MOVWF   SKIP
; variable skip is set to 0 so that new_val is called again
                    GOTO    START
```

Appendix B:

Code for catcher

```
DET_State
                        movlw b'00000010'
                        movwf LED
green_Test
                        movlw b'00000001'      ; store green val in w
                        andwf LED,0            ; mask green slot and LED
                        movwf SAVED            ; copy contents of w reg to saved
                        btfsc SAVED,0          ; skip if green slot = 0
                        goto SET_GREEN         ; jump to turn on green led

orange_Test
                        movlw b'00000010'      ; store orange val in w
                        andwf LED,0    ; mask orange slot and LED
                        movwf SAVED    ; copy contents of w reg to saved
                        btfsc SAVED,1        ; skip if orange slot = 0
                        goto SET_ORANGE        ; jump to turn on orange led

red_Test
                        movlw b'00000100'      ; store red val in w
                        andwf LED,0            ; mask red slot and LED
                        movwf SAVED            ; copy contents of w reg to saved
                        btfsc SAVED,2          ; skip if red slot = 0
                        goto SET_RED           ; jump to turn on red led
;*********************************************************************
SET_GREEN
                        movlw B'00000001'      ; set green pin in port D as output
                        movwf PORTD                    ; active low

                        goto wait_for_next
SET_ORANGE
                        movlw B'00000010'      ; set orange pin in port D as output
                        movwf PORTD                    ; active low

                        goto wait_for_next
SET_RED
                        movlw B'00000100'       ; set orange pin in port D as output
                        movwf PORTD                    ; active low

                        goto wait_for_next

;*********************************************************************
wait_for_next
                        nop
                        nop
                        nop
loop
               goto loop
                        return
;*********************************************************************
```

Appendix C:

```
;*****************************************************************************
;

; ***************************************************
;       Dallas Semiconductor 1-Wire ROUTINES
; ***************************************************
WAIT5U:
;This takes 5uS to complete
      NOP                                       ;1µs
      NOP                                       ;1µs
      DECFSZ        TMP0,F                      ;1µs or 2µs
      GOTO          WAIT5U                      ;2µs
      RETLW 0                                   ;2µs
; -------------------------------------------------------
OW_RESET:
      OW_HIZ                                    ; Start with the line high
      CLRF PDBYTE                               ; Clear the PD byte
      OW_LO
      WAIT          .500                        ; Drive Low for 500µs
      OW_HIZ
      WAIT          .70                         ; Release line and wait 70µs for PD Pulse
      BTFSS         PORTB,DQ                    ; Read for a PD Pulse
      INCF          PDBYTE,F                    ; Set PDBYTE to 1 if get a PD Pulse
      WAIT          .400                        ; Wait 400µs after PD Pulse
      RETLW 0
; -------------------------------------------------------
DSRXBYTE: ; Byte read is stored in IOBYTE
      MOVLW         .8
      MOVWF         COUNT1                       ; Set COUNT equal to 8 to count the bits
DSRXLP:
      OW_LO
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP                                       ; Bring DQ low for 6µs
      OW_HIZ
      NOP
      NOP
      NOP
      NOP                                       ; Change to HiZ and Wait 4µs
      MOVF          PORTB,W                     ; Read DQ
      ANDLW         1<<DQ                       ; Mask off the DQ bit
      ADDLW         .255                        ; C=1 if DQ=1: C=0 if DQ=0
      RRF           IOBYTE,F                    ; Shift C into IOBYTE
      WAIT          .50                         ; Wait 50µs to end of time slot
      DECFSZ        COUNT1,F                     ; Decrement the bit counter
      GOTO          DSRXLP
      RETLW         0
; -----------------------------------------------------------------------------
DSTXBYTE:                                       ; Byte to send starts in W
      MOVWF         IOBYTE                      ; We send it from IOBYTE
      MOVLW         .8
      MOVWF         COUNT1                       ; Set COUNT equal to 8 to count the bits
DSTXLP:
      OW_LO
      NOP
      NOP
      NOP                                       ; Drive the line low for 3us
      RRF           IOBYTE,F
      BSF           STATUS,RP0                  ; Select Bank 1 of data memory
      BTFSC         STATUS,C                    ; Check the LSB of IOBYTE for 1 or 0
      BSF           TRISB,DQ                    ; HiZ the line  if LSB is 1
      BCF           STATUS,RP0                  ; Select Bank 0 of data memory
      WAIT          .60                         ; Continue driving line for 60µs
      OW_HIZ                                    ; Release the line for pullup
      NOP
      NOP                                       ; Recovery time of 2µs
      DECFSZ        COUNT1,F                     ; Decrement the bit counter
      GOTO          DSTXLP
      RETLW         0
;-----------------------------------------------------------------------------
```

```
trans_nums
               movf    RESULTS, W
               ;movlw  0xAA
               movwf   TXREG
waitTX
               btfss   PIR1, TXIF
               GOTO    waitTX

               return

;--------------------------------------------------------------------------------
setup_ds:
      CALL          OW_RESET               ; Send Reset Pulse and read for Presence
Detect Pulse
      MOVLW         SKPROM
      CALL          DSTXBYTE                      ; Send Skip ROM Command (0xCC)
      MOVLW         convert
         CALL          DSTXBYTE
        SETUPDS_here
         CALL          DSRXBYTE                  ; Read the DS2761 Current Register MSB
      MOVF          IOBYTE,W
         ADDLW               0XFF
         btfss        STATUS, C
        goto SETUPDS_here
      CALL          OW_RESET                 ; Send Reset Pulse and read for Presence
Detect Pulse
      MOVLW         SKPROM
      CALL          DSTXBYTE
         movlw               readscratchpad
      CALL          DSTXBYTE                    ; Send Read Data Command (0x69)

      CALL          DSRXBYTE                ; Read the DS2761 Current Register MSB
      MOVF          IOBYTE,W
         MOVWF NUMBER_L

         CALL          DSRXBYTE                ; Read the DS2761 Current Register LSB
      MOVF          IOBYTE,W
         MOVWF NUMBER_H
        RETURN

;*********************************************************************************
init_trans
               banksel SPBRG
               movlw   d'207'     ; set baud rate to 300
               movwf   SPBRG   ; set baud rate to 300

               movlw   0x20    ;
               movwf   TXSTA   ;

               banksel RCSTA
               movlw   0x80    ;
               movwf   RCSTA   ;

               BCF     STATUS, RP0    ; Select bank 0
               BCF     STATUS, RP1
               return
;*********************************************************************************
START
                     BTFSC SKIP, 0
                     goto recieve_data            ; turn on A/D converter and recieve
data
;_____________________________________________________________________________________
________
trans_process
                     call    init_trans
                     call    setup_ds
                     call    trans_nums
                  movlw      d'100'
                     movwf transmission_done
delay_4_trans
                     WAIT    .1000
                     decfsz transmission_done
                     goto delay_4_trans
                     goto START
                     END
```

```
;_____________________________________________________________________
_____
recieve_data
                    movlw b'01010101'
                    movwf RESULTS
                    goto trans_process
```

Txsta and Rcsta descriptions:

REGISTER 10-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit
Asynchronous mode:
Don't care.
Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled
Note: SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC**: USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit
Asynchronous mode:
1 = High speed
0 = Low speed
Synchronous mode:
Unused in this mode.

bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data, can be Parity bit

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 10-2: RCSTA: RECEIVE STATUS AND CONTROL REGISTER (ADDRESS 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit
Asynchronous mode:
Don't care.
Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode – Slave:
Don't care.

bit 4 **CREN**: Continuous Receive Enable bit
Asynchronous mode:
1 = Enables continuous receive
0 = Disables continuous receive
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit

bit 2 **FERR**: Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data (can be parity bit but must be calculated by user firmware)