Introduction

This project is a culmination of my education at Cal Poly. It involves hardware interfacing, software programming, gathering datasheet information, testing, debugging and in particular, problem solving. The idea behind this project is a wireless link between a sensor located in the shoe of an individual and a watch that displays data while the user walks or runs. The sensor detects impulses when the pedestrian contacts the ground, and calculates the total distance traveled in miles, the pace in minutes per mile for the last hundred meters elapsed, and the total time while running or walking. All of these modes can be accessed by pressing the corresponding button on the watch interface. The time can be stopped, along with the collection of data, by pressing a start/stop button. When the user has finished with their activity, they can access the data to see their overall time, distance, and average pace, so they can keep a log if they so desire.

The main purpose of this project is to produce a prototype to ensure the basic concept can be implemented in a simple design. The end goal is to have a sensor that provides useful data to a watch display. Using the successful prototype, recommendations can be made as to which components should be used for a more compact version of the device. This second device would be very similar to the desired end product, but would be used for extensive testing to make sure the product

meets required specifications as well as industry standards. The recommendations for the components of the second device take into account reliability in performance and durability for long term use. The components also need to be small in order to increase ease of use and comfort when the user has the sensor in a shoe.

The idea for this project came to me when my professor for my senior project seminar told the class that some of the best projects were those that combined an individual's passions with their engineering skills to create a project that the student could be proud of. Since I ran for Cal Poly as a member of the track and cross country teams for five years, it is easy to see that I have a passion for running. This wireless pedometer concept is one that joins my love for running and logging data about my runs, with my thirst for engineering excellence and innovation. Besides being a combination of my own pursuits, this project could be useful to any number of people who enjoy walking or running to participate in an active lifestyle. The wireless pedometer lets someone know how far they walked or ran without having to guesstimate. It also can be a motivation on a run or walk because it tells the user how fast he or she is currently going. If someone is trying to maintain a certain pace, this device would be very useful for this form of aerobic exercise.

Background

To implement my idea, I would need a sensor that somehow detected how far an individual was traveling. The watch would take care of determining the amount of time the user was walking or running and the data from these two sources would be combined to determine the individual's pace. Several routes could be taken to determine the distance traveled, but a seemingly simple solution was to use an accelerometer to determine the acceleration and force of the foot's motion in order to determine distance from that data. The data would have to be processed and then transmitted wirelessly to the receiving side of the project. The calculations would be done with a microcontroller that would synchronize processing among the different components to insure proper performance. A transceiver of some sort would provide the transmitting capabilities with the aid of an antenna.

The watch portion required time keeping capabilities as well as the manipulation of data to determine mileage, and pace. The watch portion also required a receiving system in order to communicate with the sensor and transmitter associated with it. Again a microcontroller would be the easiest solution in order to run the watch system, as it could keep track of timing and coordinate between peripherals. The microcontroller would need to connect with the receiver to get data, and an LCD would display data that the microcontroller gave to it. Finally, buttons would be

connected to the microcontroller to indicate when the user wanted to cycle through different display modes, and when to start or stop the collection of data and to start or stop the timer.

I began researching this project by seeing what products were already on the market that had a similar purpose. There are several devices that accomplish similar goals, but few had information regarding their design and hardware configuration. One device I encountered did have quite a bit of documentation breaking down the device and its components and overall functionality. This device is the Nike Plus iPod sensor system that uses a small sensor that is placed in the shoe and communicates to the user's iPod. The sensor for the Nike Plus is very similar to what my desired senior project sensor would be like. However, the receiver for the sensor's data would be a watch rather than a music device such as the iPod.

Design

The sensor requires the following parts:

- Sensing Equipment
- Microcontroller
- Transmitter
- Power Source

Originally, an accelerometer was used as the sensing equipment. The accelerometer that was used was the MMA7660FC accelerometer from Freescale Semiconductor, and was going to be used to detect acceleration due to foot falls, and send that data to the microcontroller. Due to reasons that will be addressed later in this report, the accelerometer was replaced with a simple pushbutton. The pushbutton would be pressed whenever the foot came into contact with the ground, and send a high signal to the microcontroller to alert it that a footstep had been detected.

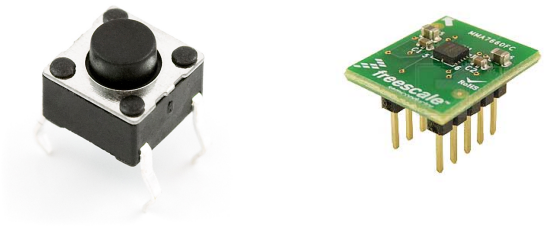

Figure 1: push button and MMA7660FC

The microcontroller that was used for the sensor portion of the project was the PIC 16F684 microcontroller produced by Microchip Technology Inc. The microcontroller is utilized to read input from the button and whenever a high signal is detected, it sends data to the wireless transmitter so that the transmitter can continue to send it along the wireless link. The microcontroller also takes care of the task of communicating with the transmitter, so that the system is set to the correct parameters for transmission. A version of PIC microcontroller with fewer pins could easily be used since there is only one input and one output from the chip. However, this chip was used in the project because size was not a huge limiting factor, and it was a part already in my possession.

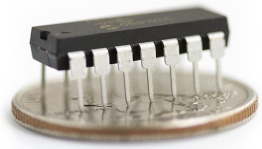

Figure 2: PICAXE 14M IC

The transmitter is an XBee OEM RF Module from the MaxStream manufacturer. It transmits in the 2.4GHz range, and utilizes the IEEE 802.15.4 protocol. It is a desirable transceiver for use in low-power, low-cost, wireless networks. Since the distance between ones wrist and foot will be less than 10 feet, a powerful transmitter was not necessary. The XBee is a reliable short range transceiver that is perfect for this scenario.

Figure 3: XBee Wireless Transceiver

The power source needs to be 3.3V. The XBee is a 3.3V device and the PIC can function at this voltage level as well. The PIC does need a higher voltage when it is being programmed, but in the final implementation, a 3.3V coin cell battery will work as the power supply for all parts on the sensor.

Figure 4: Coin Cell 3.3V Battery

The watch requires the following parts:

- Receiver
- Microcontroller
- NexysII Development Board
- LCD
- Power Source

The receiver is the same XBee module listed above. It receives data from the other XBee and the data is output to the microcontroller.

The microcontroller that was used for the sensor portion of the project was the PIC 16F88 microcontroller produced by Microchip Technology Inc. The microcontroller is responsible for communicating with the receiver, so that the watch portion is set to the correct parameters for transmission. It reads data from the receiver and transmits a high signal to the Nexys board when a step is detected. Otherwise it idles at a logic low. As with the other microcontroller previously mentioned, a microcontroller with fewer pins could easily be used, but I already had this one at my disposal.

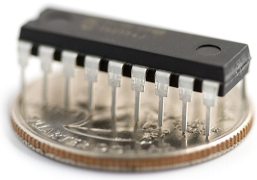

Figure 5: PICAXE 18X IC

The NexysII board is manufactured by Digilent Inc. It is a development board based around the Xilinx Spartan 3E FPGA. It has ports to connect to the LCD screen as well as the microcontroller that communicates with the receiver. The built in buttons are also used as an interface between the user and the entire system. Since I have used it before in CPE 329 to create a clock, it seemed an obvious choice to use it as the basis for the time keeping functions of my watch, as well as a good device to implement calculations for the determination of mileage traveled and the user's current pace.

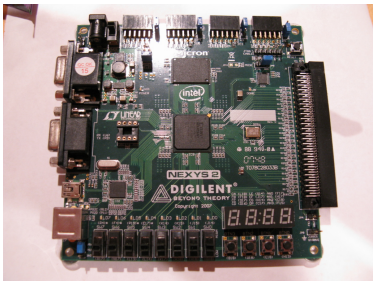

Figure 6: NEXYS 2 Development Board

The LCD screen is a module that connects to the Nexys board. It requires 5V to

operate, and it can display up to 32 characters on two different rows on the display. It is used as a means for the consumer to read the data from the system. It will display the elapsed time, mileage, and current pace.

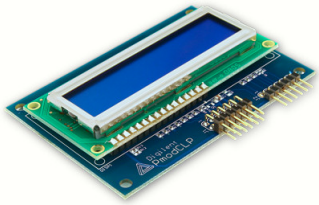

Figure 7: LCD Screen

The power source for the watch portion of the system is either through a USB connector from a computer, a power cord from a wall outlet or similar power source, or a battery. For the purposes of this project, the USB connection is the most straightforward choice.

Test Plans

In order to make sure the wireless pedometer is set up and functioning correctly, the system must be tested. The first step to a fully functioning project, is making sure the receiving portion is getting data from the transmitter. This can be seen with a scope probe to test the actual pin, but the LCD screen can also be used as a testing device. If the sensor is repeatedly activated, the mileage display on the watch should slowly increase. The buttons on the watch should also be able to cycle at will to any of the three different display modes, and the time display should keep accurate time when compared to a reliable comparison. If any of the buttons on the watch have issues, or the time is too fast or slow, the issue can be resolved in the code. Although this can be time consuming, the code must be free of bugs for it to work efficiently and correctly.

After the code has been tested for the aforementioned specifications, and the wireless link is sending data, the system should be tested as to the accuracy of the calculations for mileage and pace. Since the time keeping function has previously been tested and is assumed to be running correctly, the pace calculation will depend on the accuracy of the mileage. Therefore, it is very important to make sure the mileage calculation is setup correctly in the code. The program assumes that the stride length of the individual is three meters and then multiplies this by two to get the

distance between sensor impacts. This parameter was set at two meters because it is my average stride length while running. If there are any problems with these calculations, the code must be looked at and revised to correct the situation.

Since the purpose of this project is to set the basis for a second wireless pedometer, the test plans for that pedometer will be briefly discussed here as well. The second device will be small enough to fit within a shoe while running or walking, so the second pedometer will be used with the previously tested program and very similar hardware that the original device used. The testing for this device will let the designer know whether the calculations used in the program are accurate. A useful setting for the testing would be a 400 meter track with every 100 meters marked. The tester could start the pedometer and check it every hundred meters to ensure that the pace does update at this preset distance. Also, it would be easy to see if the pedometer is accurate by walking a mile and seeing how close the reading is to that value. Using this data, changes could be made to the calculations to make them more accurate and then use the same process to retest the new calculations.

Development and Construction

Since the project is based around the sensor, that is where the development began. Originally, an accelerometer was going to be the basis for the sensor. At first I thought the accelerometer would provide a depth of data that would lead to very accurate results in the determination of mileage and pace. However, while testing the accelerometer under different paces, the outputs from the different paces were not unique enough to accurately differentiate. Because of this, the accelerometer was going to be in its tap detection mode to detect impacts from the foot. As this was going on, I realized that the accelerometer was more or less acting as a button, and since the use of a button would require fewer resistors in its implementation, not to mention the lower cost of the button, I decided to use a simple push button as the sensing unit for the project.

The next step was to become comfortable with the PIC microcontrollers that I would be using in the project. The system I used to program these microcontrollers was PICAXE. The chips are preloaded with a bit of essential code that allows for them to understand the code written in the programming editor. The computer hooked up to the microcontroller via a serial cable and the basic circuit below was necessary for downloading the program.

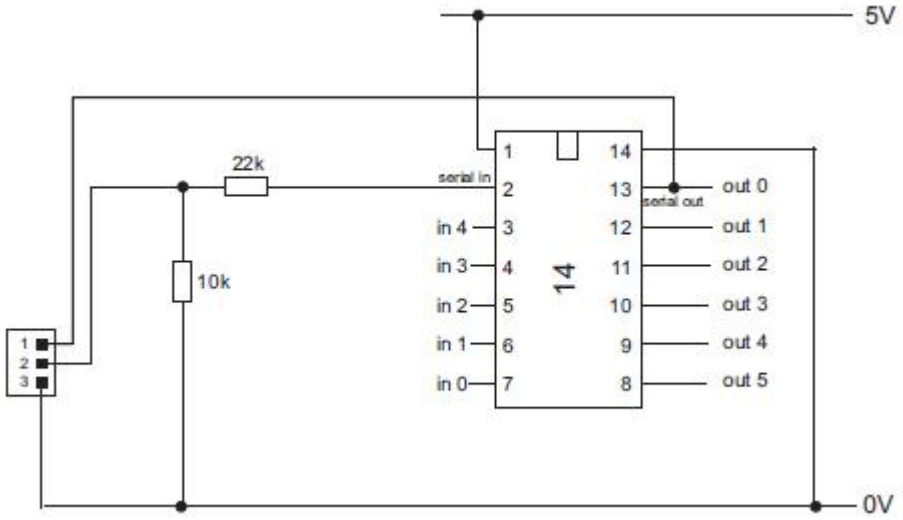

Figure 8: PICAXE Serial Download Circuit

The program was written in BASIC. To familiarize myself with programming the chip, I used some very basic situations to get a grasp of how the system downloaded the program to the chip and make sure I knew what was going on. Once I was confident with how the PICAXE downloader worked, I downloaded a program that would cause one of the outputs to go high when a corresponding input went high. Then I connected the button to the input with a pull down resistor as seen below. I then tested the setup by reading the voltage on the output while the button was pressed or not.

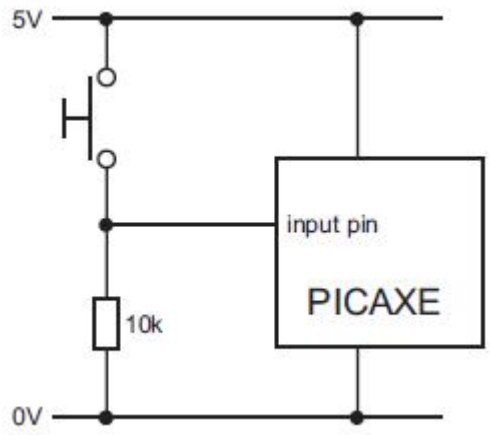

Figure 9: Push Button Connection to PICAXE

The final portion of the sensor was the wireless link. The XBee module communicates with microcontrollers via a serial UART. The PICAXE was rather simple to setup for this type of communication because there are BASIC commands that initiate the microcontroller to communicate at a certain baud rate, with or without a parity check depending on the values that are input into the command. Since the XBee is set to 9600 baud at the factory, I had to set the microcontroller to the same rate in order for them to make sense of each other's data. In order to set the microcontroller to 9600 baud it was necessary to double the frequency it operates at since the 9600 baud rate is unavailable while the PIC operates in its normal 4MHz state. With these steps taken care of, the microcontroller was ready to send data to the XBee module, and the XBee module could make sense of that data. On the other end, the second microcontroller had to undergo a similar process to provide the communication link for the received data. To test the link, the microcontroller on the

transmitting end was programmed to send a number between 0 and 255 (since the data can be 8 bits long, I randomly chose 111 as the test number). If this number was received by the receiver and its corresponding microcontroller, the PIC would turn an output pin high. Using this process, and checking and correcting the code when the link did not work, I was able to establish the working wireless link.

The construction of the watch portion of the project was a lot less hardware intensive. Almost all of the time was spent writing, testing, and correcting code within the Xilinx EDK program. The LCD screen did need to be connected to the board as did the receiving microcontroller, but after finding the pins that these peripherals would be connected to, the addresses for the pins were mapped to their correct places in the UCF file and could be dealt with within the programming environment.

The code itself consisted of the time keeping portion, the calculation of mileage and pace, and displaying these values to the LCD. The time is kept via a timer within the program that causes an interrupt every second. The interrupt increments a second counter if a condition is met that tells the program whether the start/stop button has been set to start. Whenever the second counter gets to 60, a minute counter is incremented and resets the second counter. A similar process for hours is also followed. These values are updated to the LCD if the clock display button has been pressed.

To determine mileage, the program multiplies the number of steps with a predetermined average stride length. This value is then converted to mileage. Because of memory constraints, floats and doubles were not used in the program, so integer math and some ingenuity had to be used to make sure the values were not carelessly rounded off to keep accurate results. If the mileage display button has been pressed, the mileage data will be updated to the LCD.

For the pace, the program keeps track of the number of seconds it takes for 100m to be traveled. Calculations are then used to determine the user's pace in minutes per mile. Using this formula, the pace is updated every 100m. Depending on the user's pace, this should update at least once a minute assuming the user is in motion. The pace is displayed to the LCD if the pace button has been pressed.

Integration and Testing

Putting together the separate parts of the project was very exciting, but full of anxiety as well since it represented a big step in the life of the pedometer. When I first put the entire project together, the wireless transceivers were not reading data from one another. At first I thought this was a problem with the microcontrollers, but after making sure they were putting out the correct signals with a scope probe, I had to deduce that there was something amiss with the transceivers themselves. Eventually I discovered that the transmitting XBee had to be set at a lower voltage level than in the previous testing stages in order for it to work. Looking over the data sheets and information regarding the XBee, does not do much to clarify the issue, but it does not pose a significant problem because the final device is supposed to be powered by a 3.3V battery which is the voltage that the XBee modules worked with during the final integration, even though they previously worked near 5V levels during the prior stages.

Once the signal was being detected by the receiver, the watch operations were tested. Each button's functionality was tested, and whenever there was any difference between the actual display and the desired display, the code had to be edited. The mode buttons did not need to be debounced because if they are pressed multiple times, it remains on the same display. However, the start/stop button did need

debounce code since it should only stop or start the clock once every time it is pressed. The final step for testing, and the most difficult, was testing the sensor and seeing that it produced the correct mileage and pace for the number of 'impacts' it detected. This was a hard parameter to test because there are quite a number of steps within a mile, so to get any sort of accurate reading and information, the sensor needed to be activated by hand around 800 times to get a mile on the output. Luckily though, that is the desired output, and to test whether or not these calculations are adequate in an actual pedestrian environment, the second pedometer should be made and tested as stated in the previous section of this report.

Conclusion

This Project was a very good learning experience for me. It really captured the essence of the 'Learn by Doing' philosophy that Cal Poly lives by. Being an individual project, I got to experience every facet of design and implementation, which was both a blessing and a hindrance. It was very good for me to see how everything worked out, and be part of the entire project. However, a group effort would have been much more efficient and through this project I have learned to appreciate the group dynamic more.

It is also quite apparent that costs should be limited at all times during the length of a project. It is ridiculous how much money one can save, especially if mass production is involved, by choosing a few key parts that are less expensive. This project allowed me to think about cost analysis, the pros and cons of certain devices, and the limitations that were put in place by the design and function of the pedometer. It is impressive how quickly things add up in a project; time and money are obvious currencies that can be spent like mad throughout the life of a product if one is not careful.

This project was very helpful in both allowing me to utilize my knowledge of hardware and software to make an application that is useful and relatively inexpensive as well as appreciate the depth of knowledge that there is in both of these categories since this project only skims the surface in terms of the wide breadth of engineering resources. I can easily say this project has meant a lot to my education as

an engineer. It made me realize the importance of scheduling and time management and allowing for extra time because errors and obstacles are going to occur. The hours spent in lab at the computer screen typing code made me appreciate the solidity of hardware design and development, while the time spent bent over a tiny PCB in an awkward position while soldering components made me think fondly of programming software. Overall, this wireless pedometer has given me a lot of time to think about being an engineer, and although at times it is hard work and can be very frustrating, providing a final product that does amazing things is well worth the price of long hours in the lab.

Parts List & Costs

All Parts	quantity	unit cost	price
Nexys 2 Development Board	1	129	$129.00
Character LCD w/ Parallel Interface	1	24.99	$24.99
PCB & Solder		~1.50	$1.50
Mini Push Button Switch	1	0.35	$0.35
4.7k Resistor	1	~0.00	$0.00
10k Resistor	1	~0.00	$0.00
Miscellaneous Jumper Wires		~0.00	$0.00
Picaxe - 14M (PIC 16F684)	*1*	*3.95*	*$3.95*
Picaxe - 18X (PIC 16F88)	*1*	*7.95*	*$7.95*
Xbee 1mW Chip Antenna	*2*	*22.95*	*$45.90*
Total			$213.64

Recommended Parts for 2nd Pedometer	quantity	unit cost	price
Picaxe - 08M (PIC 12F683)	*2*	*2.95*	*$5.90*
Nordic Transceiver (nRF2401A)	*2*	*4.75*	*$9.50*
16MHz Crystal	*2*	*1.5*	*$3.00*
2.4 GHz Chip Antenna	*2*	*2.95*	*$5.90*
Assorted Resistors and Capacitors	~2.50		*$2.50*
New Total			$26.80

Comparison to Original Pedometer	quantity	unit cost	price
Picaxe - 14M (PIC 16F684)	*1*	*3.95*	*$3.95*
Picaxe - 18X (PIC 16F88)	*1*	*7.95*	*$7.95*
Xbee 1mW Chip Antenna	*2*	*22.95*	*$45.90*
Original Total			$57.80

Schedule – Time Estimates

Research

Activity	Time
Picaxe Datasheets	5 hrs
Xbee Datasheets	3 hrs
Nexys Datasheets	2 hrs
Accelerometer Datasheets	2 hrs
Accelerometer Characterization	3 hrs
LCD Datasheet	1 hr
Research Total	16 hrs

Programming

Activity	Time
BASIC programming	2 hrs
EDK Hardware Configuration	3 hrs
C Timer Code	5 hrs
C Button Code	2 hrs
C LCD Code	4 hrs
C Calculation Code	5 hrs
Programming Total	21 hrs

Hardware

Activity	Time
Picaxe Serial Download Circuit	2 hrs
Xbee Interface	1 hr
Soldering Sensor	2 hrs
Nexys-LCD Interfacing	4 hrs
Hardware Total	9 hrs

Testing

Activity	Time
BASIC Program	2 hrs
Timer Code	3 hrs
Button Code	2 hrs
LCD Code/Hardware	5 hrs
Calculation Code	3 hrs
Xbee Link	5 hrs

General Picaxe	2 hrs
Sensor to Picaxe	1 hr
Picaxe to Nexys	1 hr
Testing Total	24
Total Estimated Time	70 hrs

BASIC Program Listing

```
'PICAXE 18X Receives Data from XBee
main:
        Symbol RX_PIN = 1
        SetFreq M8
        high RX_PIN
        Do
        SerIn RX_PIN, T9600_8, b0
        if b0=111 then
        high 4
        else
        low 4
        endif
        Loop

'PICAXE 14M Transmits Data to XBee
main:
        Symbol TX_PIN = 3
        SetFreq M8
        High TX_PIN
        Do
        if pin1=1 then
        SerOut TX_PIN, T9600_8, (111)
        endif
        loop
```

C Program Listing