Wearable Wireless Physiological Sensors
Final Communication

Daniel Bishop
Josh Handley
Phillip Hay
Christina Hernandez
Rosy Logioia
Gouri Shintri
Clay Smith
Adam Stevenson

May 11, 2004
CPSC 483
# Table of Contents

Implementation Notes .....................................................................................................4
Sensors ........................................................................................................................ 4
  BVP SENSOR CIRCUIT ........................................................................................4
  TEMPERATURE SENSOR CIRCUIT ....................................................................5
  GSR SENSOR CIRCUIT ........................................................................................6
Sensor Board Schematics.........................................................................................7
Sensor Board ...........................................................................................................8
Parts List: ................................................................................................................9
Inputs on Sensor Board..............................................................................................11
  BLOOD VOLUME PULSE SENSOR: ..................................................................11
  TEMPERATURE AND GSR: ...............................................................................12
  OUTPUTS:............................................................................................................12
CALIBRATION: .......................................................................................................13
  BVP SENSOR:......................................................................................................13
  TEMPERATURE SENSOR ..................................................................................14
  GSR SENSOR.......................................................................................................15
Boards .......................................................................................................................15
Data Acquisition and Transmission ...........................................................................19
  Data Acquisition ....................................................................................................19
  MCU, Timers, A/D conversion ..............................................................................19
  MCU interfacing ....................................................................................................20
  Transmission .........................................................................................................21
  Setup information ..................................................................................................21
  Code description ....................................................................................................21
  Timing Diagrams ..................................................................................................22
Graphical User Interface (GUI) .................................................................................23
Universal Serial Bus (USB) Adapter..........................................................................27
User’s Manual ............................................................................................................29
  Installing Software ...............................................................................................29
  Applying Sensors ..................................................................................................29
    Temperature ..........................................................................................................29
    Blood Volume Pulse (BVP) ..................................................................................29
    Skin Conductance ...............................................................................................29
Transmission and Display ..........................................................................................30
  GUI Menu .............................................................................................................30
    SPREADSHEET FILE OPERATIONS .................................................................30
    EXIT ...................................................................................................................30
    GRAPH START/STOP CLEAR OPERATIONS ....................................................31
    GRAPH SETTINGS OPERATIONS ....................................................................31
    MISCELLANEOUS ............................................................................................31
Course Debriefing .........................................................................................................31
  Group management style .......................................................................................31
  Safety and ethical issues ........................................................................................32
  Product testing ........................................................................................................32
  Additional steps taken if given more time .............................................................34
Wearable Sensor Timeline .........................................................................................35
Budget Outline ..........................................................................................................36
Introduction

In our proposal we had originally proposed that the wearable sensors would have the capability to monitor the emotions of the individual wearing them. Due to the sensitivity of the product, our objective altered to complete the configuration noted below:

- Temperature, galvanic skin resistance (GSR), and blood volumetric pressure (BVP) sensors are connected to a microcontroller where data is read in.
- The microcontroller sends data collected by the sensors through a wireless transmission system, where it is received on the other side and sent through the USB port of the computer.
- The GUI reads in the data from the USB port and displays it graphically on the screen in real time.

Our project was successful with the exception of the transmission system. Despite the fact that our team attempted two different wireless chip systems (chipcon and laipac), we were unable to complete a working protocol. There were various reasons for this non-completion of our revised objectives and they are as follows:

- Lack of proper testing equipment on Texas A&M campus.
- Wireless system either completely works or does not. It is very difficult to debug.
- Major delay of transmission board fabrication.

Our completed project can successfully read the skin conductance, temperature, and blood oxygen level of the individual wearing the device. It can then connect directly to a computer, where it outputs the results of the data gathered from the subject.

A basic outline of the project procedure, testing methods, and engineering design are demonstrated throughout this report. Despite the lack of wireless transmission within our product, we feel our project was very successful and would like to thank the following individuals and corporations for their support: NSF, Applied Materials, Dr. Gutierrez, Di Wu, Steve Ortiz, and Dr. Cote.
Implementation Notes

Sensors

BVP SENSOR CIRCUIT

Fig 1. Bvp circuit
TEMPERATURE SENSOR CIRCUIT

Fig 2. Temperature sensor
Fig 3. GSR circuit
Fig. 4: Sensor Board Schematics

Drawn with Eagle Layout Editor version 4.11
Sensor Board

Fig. 5: Sensor board – Actual size: 2”x2”

Fig. 6: Sensor Board – 1:3 scale
Parts List:

R1: 470 ohm resistor -- P470JCT-ND (Digikey Co.)
R2: 470 ohm resistor -- P470JCT-ND (Digikey Co.)
R3: 150 ohm resistor -- P470JCT-ND (Digikey Co.)
R4: 470 ohm resistor -- P470JCT-ND (Digikey Co.)
R5: 500 kohm potentiometer -- 72-T934-500K (Mouser Electronics)
R6: 10 Mohm resistor -- P10MJCT-ND (Digikey Co.)
R7: 10 Mohm resistor -- P10MJCT-ND (Digikey Co.)
R8: 470 ohm resistor -- P470JCT-ND (Digikey Co.)
R9: 470 ohm resistor -- P470JCT-ND (Digikey Co.)
R10: 470 ohm resistor -- P470JCT-ND (Digikey Co.)
R11: 470 ohm resistor -- P470JCT-ND (Digikey Co.)
R12: 470 ohm resistor -- P470JCT-ND (Digikey Co.)
R13: 1.2 kohm resistor -- P1.2KJCT-ND (Digikey Co.)
R14: 1.2 kohm resistor -- P1.2KJCT-ND (Digikey Co.)
R15: 1.2 kohm resistor -- P1.2KJCT-ND (Digikey Co.)
R16: 1.2 kohm resistor -- P1.2KJCT-ND (Digikey Co.)
R17: 150 ohm resistor -- P150JCT-ND (Digikey Co.)
R18: 10 kohm potentiometer -- 72-T934-10K (Mouser Electronics)
R19: 200 kohm potentiometer -- 72-T934-200K (Mouser Electronics)
R20: 1 Mohm resistor -- P1.0MJCT-ND (Digikey Co.)
R21: 12 kohm resistor -- P12KJCT-ND (Digikey Co.)
R22: 12 kohm resistor -- P12KJCT-ND (Digikey Co.)
R23: 12 kohm resistor -- P12KJCT-ND (Digikey Co.)
R24: 100 ohm resistor -- P100JCT-ND (Digikey Co.)
R25: 100 ohm resistor -- P100JCT-ND (Digikey Co.)
R26: 560 ohm resistor -- P560JCT-ND (Digikey Co.)
R27: 560 ohm resistor -- P560JCT-ND (Digikey Co.)
R28: 10 kohm potentiometer -- AAS14CT-ND (Digikey Co.)

C  9: NEV100M50VD (Midstate)
C10: NEV100M50VD (Midstate)
C11: NEV100M50VD (Midstate)
C12: NEV100M50VD (Midstate)
C13: NEV100M50VD (Midstate)
C14: NEV100M50VD (Midstate)
C15: NEV100M50VD (Midstate)
C16: NEV100M50VD (Midstate)

OPAMP A,B,C,D: OP490GS (Analog Devices) – powered by +3.5V and -3.5V
**BVP**

Disposable Pulse Oximeter

Oxisensor® II Adhesive Sensors
From Nellcor, Model D25

24 per box -- $250.00 or ask for free samples.

---

**GSR**

Disposable Skin Conductance Sensor

Oxisensor® II Adhesive Sensors
From Stoelting Co., Catalog No. 85065

Include 2 reusable snap connector leads
and 100 sensors -- $45.00

---

**THERMISTOR**

Thermistor

10Kohm thermistor
Radio Shack -- $3.00
Inputs on Sensor Board

BLOOD VOLUME PULSE SENSOR:

1. Black wire of finger sensor
2. Red wire of finger sensor
3. Silver shield of finger sensor
4. Copper shield of finger sensor
5. White wire of finger sensor
6. Green wire of finger sensor
7. Black wire of bone sensor
8. Red wire of finger sensor
9. Silver shield of bone sensor
10. Copper shield of bone sensor
11. White wire of bone sensor
12. Green wire of bone sensor
13. +3.5 volts
14. -3.5 volts
15. Ground
16. Not connected
TEMPERATURE AND GSR:

Fig. 9 IN2 on board

1. GSR red wire
3. Thermistor

2. GSR black wire
4. Thermistor

OUTPUTS:

Fig. 11 OUT on board

1. BVP final output
3. BVP output at nulling DC offset
5. Temperature output

2. GSR black wire / Ground
4. GSR red wire
6. Not connected

Fig. 10 IN2 on schematics

Fig. 12 OUT on schematics
CALIBRATION:

BVP SENSOR:

We first have to null the potential difference from the two sensors hooked up to the finger by using a 500 kohm (R5) potentiometer attached to the finger sensor so that it has the same voltage has the bone sensor. The DC is zeroed after the subtraction circuit:

![Subtraction circuit](image1)

At this point, we can put a gain on the signal through the 10 kohm (R18) potentiometer, and adjust the DC offset, through the 200 kohm (R19) potentiometer, between 0V and 3V so to satisfy the opamps and microcontroller power limitation of 3.5V.

![Gain and DC offset of final signal](image2)
This was done because the microcontroller cannot handle negative values, and we can extrapolate the pulse signal as a clean wave with no noise. Once we captured a wave, we have the test subject hold his breath for several seconds, and the wave disappears and goes off the voltage scale because no more oxygen is flowing through the blood vessels. Also, we have the subject simulate an increase in breathing, as if he was exercising, which makes the frequency value increase.

![Fig. 15 Pulse signal captured by oscilloscope.](image)

**TEMPERATURE SENSOR**

We calibrated the sensor by heating up a glass of water and measuring its temperature versus the potential difference between V1 and V2. We then plotted the temperature versus voltage difference to obtain a formula that we could use to display our results on the GUI. We also included a gain of about 5.6 to the final value so that the output voltage could be captured by the microcontroller.

**Formula:** \( T = \left( \frac{59.201}{5.6} \times \right) + 76.31 \)

In this circuit we use the 10 kohm (R28) potentiometer to make sure V1 equals V2 so that we can calibrate the thermistor to room temperature (approx. 76ºF). Once this value is set the potentiometer does not need to be recalibrated.

We take the skin temperature of the subject; we then have him remove the thermistor, touch a cold ice pack and reposition the thermistor; we observe the skin temperature decreasing accordingly. We repeat the procedure by having the subject touch a hot pack and observing the temperature increasing accordingly.
GSR SENSOR

This sensor does not need to be calibrated because it varies from person to person. We measure the voltage difference between the two head sensor on the finger (Vskin) and we apply Ohm’s Law to find out the skin resistance of the subject:

\[ 3.5 - V_{\text{skin}} = V_{\text{resistor}} \]  (resistor voltage)

\[ V_{\text{resistor}} / 1 \text{ Mohm} = i \]  (current flowing through circuit)

\[ V_{\text{skin}} / i = R_{\text{skin}} \]  (skin resistance of subject)

\[ 1 / R_{\text{skin}} = C_{\text{skin}} \]  (Conductance of skin in uSiemens)

We then test the sensor by recording the skin resistance/conductance of the subject. We then apply water on the finger to simulating sweating, and we observe the skin resistance drop and the skin conductance increase.

Boards

Our goal was to create an entire wireless system on a single board. To do this we needed a microcontroller, an analog to digital converter, and a wireless chip. The microcontroller that we decided to use was the Silicon Laboratories C8051F015. We chose this microcontroller for several reasons. One we had a C8051F015 evaluation board that a group had used the previous semester. This allowed us to immediately begin working on our microcontroller board without having to wait for another one to ship in and we did not have to purchase another one. We also chose this chip because it is a very small microcontroller, approximately 1cm square, while having all our desired features built into the chip. The microcontroller has an eight-channel analog to digital converter, flash memory, RAM, and 32 input/output pins built into the chip. The microcontroller has a 16Mhz clock as well. The cost of the microcontroller is self was about $23.

The wireless chip that we chose to go with was the Chipcon CC2400. We chose this chip because it is a low cost, low power RF transceiver that operates at approximately 2.4Ghz. The cost of this chip is approximately $45 for five chips. A professor had already purchased the development kit for this wireless chip the previous semester for a student project. The professor expressed his interest in our using the chip and gave us permission to use the development kit as well as helping pay for some of the costs of developing the board.

The boards we created were a combination of the two evaluation boards. Chipcon published one schematic while Silicon Laboratories published the other. We fabricated our board using Eagle Layout Editor version 4.09. Eagle allowed us to create a schematic, design the board from the schematic, route the wires, and make the Gerber manufacturing files. We had to explore Eagle in order to properly use the software, but after a while were able to somewhat effectively
create all the necessary files. We had help from some graduate students who knew how to use Eagle and were able to give us some design tips.

The microcontroller’s circuit was very easy for us to understand and copy into our schematic. The flash memory, RAM, analog to digital converter, and processor were all on the same chip. We had to provide power to the chip, connect the JTAG programmer correctly, and interface properly with the 32 data pins. There were quite a few capacitors and a few resistors on the evaluation board for the purpose of decoupling the power circuit. The microcontroller evaluation board has two ground planes built into the second layer. One is for the mechanical ground and the other is for the analog to digital converter, the signal ground. We did not find much reason for having two separate ground planes for the simple fact that both ground planes on the evaluation board were connected with a zero Ohm resistor. Therefore, in our design we created a single ground plane. The microcontroller used a single 3.3V power supply. This was provided on the evaluation board using a LM2937 voltage regulator. We were able to find the exact voltage regulator, although this is not needed because any 3.3V voltage regulator is sufficient as long as all design constraints are met for the microcontroller. The following schematic was used to design the microcontroller portion of the project.

Fig. 16  Board schematic
The Chipcon CC2400 is a complete wireless chip. For our design we were required to supply power, ground, the antenna circuit, and the proper inputs to control the chip. There were several capacitors, a few inductors, and a few resistors needed to complete the design that was published by Chipcon. The capacitors were used to decouple the power circuit, while the inductors were used for the antenna circuit. On the evaluation board that we obtained from the professor there were thirteen control pins, a 3.3V power supply, a 1.8V power supply, and ground pins. We connected all thirteen control pins to the microcontroller input/output pins so that we would have the maximum flexibility of the chip. The chip only requires four pins to communicate: SI, SO, CSn, and SClk. These pins can be used using a Serial Peripheral Interface (SPI). We used a slightly different antenna circuit that was on the evaluation board because of the large size of the antenna that it required. We used a simpler dipole antenna that only required two inductors. The following schematic was the basis for our design.

In our design we were unsure of whether the ground circuit on the Chipcon needed to be separate from the mechanical ground. Per the recommendations from a helping graduate student we placed six 10pF capacitors separating the two planes as well as a jumper that would allow us to bypass the capacitors if needed. After we created the boards it became evident that we needed the two grounds connected together because there was no power otherwise.

We did find two errors in our design after the board had been manufactured. We placed two voltage regulators on our board, a 1.8V regulator and a 3.3V regulator. We looked up the documentation for the 3.3V voltage regulator and designed our circuit according to that documentation. We found a 1.8V regulator in the same package and assumed that since they were in the same package that they would have the same pin layout. Unfortunately we were incorrect. We discovered this when we plugged our circuit in and the Chipcon CC2400 began getting very hot. After investigating we found the problem. We cut the incorrect wires and used wrapping wire to correct the circuit. The other slight error that we found was the BT/GR pin was connected to an input on the microcontroller instead of being tied to ground. The BT/GR pin is used for internal testing of the microcontroller and for normal operation should be tied to ground. When the microcontroller boots up all data pins are set to VCC. This was easily corrected by cutting the wire connecting the BT/GR pin to the microcontroller and using wrapping wire to connect the pin to ground.

We used the company E-Teknet to produce the boards. The company did a good job of producing the board except for two items. One the board took over twice as long to get back as had originally expected and two there was one via that was touching the ground plane when it was not supposed to be. This was discovered when we were testing to see if any two sides of a capacitor were directly connected. The via was on a wire that was connected to the VREF pin. This is a pin on the microcontroller that allows the user to externally set the
reference voltage for the analog to digital converter. This was not a problem because the microcontroller is able to internally set the reference voltage. We cut wires on both sides of the microcontroller and eliminated the problem. We were able to populate the boards ourselves. This allowed us to verify that all parts of the circuit were correct and we did not have to worry about incorrect soldering of manufactures.

Fig. 16 PCB Layout
Data Acquisition and Transmission

Data Acquisition

MCU, Timers, A/D conversion

The data acquisition portion of our project involved converting the 3 analog sensor signals to digital values. This was done using a correctly configured Silicon Laboratories C8051F015 microcontroller unit. The MCU uses a C-based IDE, and many of the “extra” functions (such as timers, UART, A/D conversion) are obtainable by setting the numerous register values correctly. The “Config v.2.05” utility is available for help in configuring these registers. The MCU is easily programmable through use of the IDE, a serial cable, the “Cygnal Serial Adapter EC2”, and the JTAG programming cable.

On the MCU, there are 32 I/O pins that can be configured for use in programming. These are set to variables that can be manipulated by using the syntax “var = Px^y” where var is the variable that will represent the pin, x is the port number (from 0 to 3), and y is the pin on the port (from 0 to 7). These pins must be set to input/output or input only by using the MCU’s crossbar. All pins on port 1 are set to I/O with the statement PRT1CF = 0xFF.

In order to make the microcontroller actually perform the A/D conversion, several things had to be done. First of all, a timer was set to throw an interrupt every X clock cycles. Since the frequency of the clock was 16 MHz (or 2MHz in some later versions of the code), the frequency of interrupt could be determined by setting the timer’s “Reload” value to (the MCU’s clock speed)/(desired frequency). For example, a sampling frequency of 500 Hz could be obtained by using the reload value of (16000000)/(500) or 32000. Some lower frequencies could not be obtained using this timer, since it can only hold a reload value of up to 16 bits or 65535. The lowest frequency obtainable at this clock speed is approximately 250 Hz.

The A/D conversion was then set up to be performed automatically at every interrupt of Timer3 (the timer used as described above). This was done by appropriately configuring the registers. The correct values of the registers can be determined by looking at the data sheet (http://www.silabs.com/products/pdf/C8051F0xxRev1_7.pdf), viewing the Comm.c code included within this report, or using the configuration utility.

In each conversion cycle, the A/D converter can only convert 1 analog signal to a digital value. The analog channel converted is determined by the value within the AMX0SL register. The analog input pin A0 corresponds to analog input channel 0, A1 corresponds to channel 1, etc. Our code implemented multiple conversions by initializing a “channel” variable to 0, and incrementing it after each conversion. When the variable became larger than 2, it would of course be set to 0 again. The channel variable was used to set which channel was currently being converted by the A/D converter by using it to specify the value in AMX0SL.
The value received from the A/D converter was a digital value between 0 and 1023. A value of 0 meant that the analog signal was 0 volts. A value of 1023 meant that the analog signal was VREF volts. VREF can either be set internally or by an external source. The gradation in values received from the A/D converter was approximately linear. For example, a value of 235 received by the A/D converter would correspond to an analog voltage of \((235)/(1024)\)\*VREF.

**MCU interfacing**

When the data from all 3 channels had been collected, a flag would be set which meant that the digital values representing the analog signals were valid. When this occurs, the data is ready for transmittal. In the current version, this involves transfer via the USB device. Ideally, however, it will involve using the Laipac wireless chips.

Before the data is ready for transfer via USB, several things must first be done to it to ensure delivery and correct interpretation. Since there are only a limited number of input pins on our transmission boards, only the lower nibble (4 bits) of the USB bus were used. This was done by wiring the upper nibble of the USB data bus to ground and wiring 4 pins on the microcontroller to data bits 0-3. The “Write” pin on the USB board was also wired to an MCU pin. With this setup, the data pins would be set to the correct values and the data would be written to the MCU when the Write pin was pulsed high.

Eight packets of 4 bits each of data were sent every time the valid data flag was set high. The packets are in the form in the following table.

<table>
<thead>
<tr>
<th>Binary Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00001111</td>
<td>Header packets</td>
</tr>
<tr>
<td>0b00001111</td>
<td></td>
</tr>
<tr>
<td>0b0000xxxx</td>
<td>High 4 bits from sensor 0</td>
</tr>
<tr>
<td>0b0000xxxx</td>
<td>Second 4 bits</td>
</tr>
<tr>
<td>0b0000xxxx</td>
<td>Data from sensor 1</td>
</tr>
<tr>
<td>0b0000xxxx</td>
<td></td>
</tr>
<tr>
<td>0b0000xxxx</td>
<td>Data from sensor 2</td>
</tr>
</tbody>
</table>

The Header packets were included to ensure that the GUI could correctly interpret which packets corresponded to which sensor data. Without this header, we encountered many problems with the visualization of the data. To ensure that there would not be any two other 0x0F packets in a row, we would set any low nibbles that were 0x0F to 0x0E. While this slightly decreased the resolution of the digital voltages, it was deemed a worthy tradeoff.

Only the upper 8 bits of data received from the A/D conversion were sent for each sensor, because of the low pin availability on the MCU.
Transmission

Setup information

The transmission was first attempted using Chipcon’s CC2400 boards. After many weeks of difficulties getting these wireless chips to work, we decided to implement wireless with the Laipac TRF-2.4G (also called TRW-2.4G). Here is the general setup that we used for testing the Laipac chips.

![Transmission Circuit Setup](image)

**Fig. 17 General Setup**

The MCUs were both connected to separate PCs, with code for wireless transmission on one and code for receiving on the other.

On both Microcontrollers, pins were set up to connect to 5 signals on the Laipac chips: CS, CE, CLK1, DATA, and DR1. ChipSelect is a MCU-controlled pin that controls when configuration data is being transferred. ChipEnable is a MCU-controlled pin that controls when the transmission packet is being transferred to the Laipac chip. CLockK1 is a MCU-controlled pin that is used to “clock in” the data. DATA is a MCU-controlled pin representing the data that is being transferred. DR1 is a pin controlled by the Laipac chip that is set to high when data is ready for reception.

Configuration is done by shifting a certain number of bits into the Laipac chip. Either 144, 120, 16, or 1 bits can be shifted into the chip. Shifting in 1 bit will allow the user to control whether the chip is set to transmit or receive. Shifting in 16 bits will allow control over this and the general configuration settings, such as data transfer rate, frequency channel, and mode of operation. Shifting in 120 bits will allow control over these and ShockBurst™ configuration, including the address packet sizes, address packet values, and CRC checking. Shifting in 144 bits will allow control over all of the previous things as well as the testing configuration, of which there is no known documentation. An in-depth description of the configuration register values can be seen in the TRF-2.4G Reference Guide, the attached code Tx.c and Rx.c, or our correspondences with Laipac technical support.

Code description

Tx.c is the code used on the transmission chip and Rx.c is the code used on the receiving chip. There are 3 main functions within the transmission and receiving
These functions are for configuration, transmission, and receiving of data. They mimic the timing diagrams described in the TRF-2.4G Reference Guide. All functions are assumed to begin with all pin values set to 0 (CS, CE, CLK1, DATA, DR1).

To enter configuration mode in the code, CS is first set to 1. Then, 120 bits of data are clocked into the Laipac chip. Several global constant definitions represent the configuration data that we wish to send. The data is clocked in on rising edges of the CLK1 pin. To send a data value of 1, for example, the sequence CLK=0, DATA=1, CLK=1, CLK=0 is performed. This is of course done with appropriate pauses between the changing of the pins. When all 120 bits of data have been clocked into the chip, CS is set back to 0.

To enter transmission mode, CE is set to 0. Then, the 8-bits of the receiving chip’s address are sent, followed by the 24-bit data payload. The data is sent in the same manner as is done with the configuration function. After this, CE is set back to 0.

The receiving code starts with a blocking wait-loop that waits until DR1 is set to 1. When this happens, the data is read off of the Laipac chip using a program-driven clock cycle. To read one bit of data from the chip, the sequence CLK=0, readData = DATA, CLK=1 is performed. readData is the bit of data that is received from the chip. In this manner, 24 bits are read from the chip.

In the main loop of Tx.c, the microcontroller configuration function is called, the Laipac configuration function is called, and an infinite loop is entered which calls the transmission function. Rx.c calls both configuration functions and enters an infinite loop which calls the receiving function.

**Timing Diagrams**

![Timing Diagram](image)
**Graphical User Interface (GUI)**

**Class Configuration**

This is a static utility class that stores graph settings and provides functionality to save and load graph settings (line color, time range, and vertical range) to a configuration file (named WeaSeL.exe.config).

**Attributes**

- TemperatureGraphSettings, ConductanceGraphSettings, PulseGraphSettings
- SecondsToGraph

**Methods**

- Static constructor: Loads data from the configuration file. If this file doesn’t exist or is corrupted, initializes the settings to reasonable default values.
- Save(): Writes the settings to the configuration file.

**Class FrequencyDisplay**

A ReadingDisplay designed for periodic signals; it displays the frequency of the signal in beats per minute.
The frequency is calculated with the use of two counters: One counts the number of maximum and minimum points on the graph, and the other keeps track of elapsed time.

**Class GraphSettings**

A simple aggregation of the configuration settings needed for a graph.

**Attributes**

- YMin, YMax: The vertical range of the graph.
- LineColor

**Class GraphSettingsControl**

A subclass of UserControl which contains controls to adjust the range and color of a graph. It is used in the implementation of the OptionsDialog class.

**Attributes**

- Settings: Returns or sets the GraphSettings of the associated graph.

**Methods**

- Constructor GraphSettingsControl(ReadingDisplay): Associates this control with a graph.
- UpdateGraph(): Changes the graph to match the values displayed in the controls.

**Class GraphTimeDialog**

A popup window that asks the user for the number of seconds of data to display on the graph.

**Attributes**

- Value: The number displayed on the form.

**Interface ISensorReader**

Abstract representation of a sensor. Implementations are provided in the SensorReaders class.

**Methods**

- Read(): Return the current reading of the sensor.

**Class MainForm**
The main form for WeaSeL, which contains the three ReadingDisplays for the sensor data, and various related controls.

**Class NumberQueue**

A sequence of numbers that expands until it reaches a maximum capacity, then behaves as a first-in-first-out queue. Used in the implementation of ScrollingLineGraph.

**Attributes**

- Capacity: The number of elements that the NumberQueue can contain.
- Count: The number of elements actually contained in the NumberQueue.

**Methods**

- Array indexing
- `operator double[]`: Converts the NumberQueue to an array.
- `Enqueue(double)`: Adds a number to the end of the NumberQueue.

**Class OptionsDialog**

A popup window containing controls for adjusting graph ranges.

**Abstract Class ReadingDisplay**

A control for displaying the output from a sensor, containing a Label and a ScrollingLineGraph. There are two concrete subclasses: ValueDisplay and FrequencyDisplay.

**Attributes**

- Settings: The GraphSettings for the graph.
- TimeSpan: The number of seconds of data to display on the graph.
- Description: Descriptive text to be displayed above the graph.
- Units: The unit of measure for the value displayed.
- `CurrentReading (abstract)`: The last value displayed on the graph.
- `AverageReading (abstract)`: The mean reading since the last call to Reset().

**Methods**

- `UpdateReading() (abstract)`: Adds a data point to the graph.
- `Reset() (virtual)`: Erases the graph and clears the data used for AverageReading.

**Class ScrollableLineGraph**
A subclass of UserControl that provides a visual display for the numbers contained in a NumberQueue.

**Attributes**

- **YMin**: The numeric value represented by the bottom of the graph
- **YMax**: The numeric value represented by the top of the graph
- **Capacity**: The maximum number of points displayable on the graph
- **Data**: a NumberQueue containing the points displayed on the graph
- **Settings**

**Methods**

- **Clear()**: Erase the graph.

**Class SensorReaders**

A class that contains three static instances of ISensorReader, one for each sensor.

**Attributes**

- **TemperatureSensor** (static)
- **ConductanceSensor** (static)
- **PulseSensor** (static)

**Class Status**

Represents the status of a USB device.

**Attributes**

- **DevicePresent**
- **DeviceOpened**
- **TXE**: “transmit buffer available”
- **RXE**: “byte available”

**Class USBDevice**

Provides communication with the USB port.

**Attributes**

- **DataAvailable**
- **AvailableBytes**
- **Status**

**Methods**
• Read(byte[] buffer, int max): Read up to max bytes from the USB device and store it in buffer.

**Class USBDeviceCouldNotBeOpenedException**

Self-explanatory. Thrown by USBDevice constructor.

**Class USBDeviceNotFoundException**

Self-explanatory. Thrown by USBDevice constructor.

**Class ValueDisplay**

A subclass of ReadingDisplay for displaying non-periodic data.

**Universal Serial Bus (USB) Adapter**

We decided to use the DLP-USB245M FIFO to connect our microcontroller to the computer. The device is was relatively cheap and easy to acquire through Mouser. The device is 1.1 compliant and is plug and play compatible given that the device drivers are already downloaded to the computer and located in the Windows system directory. One of the nicer features was that it provides a 8 Megabit per second data rate so that we would be able to send as much data as we would like to the computer from the microcontroller. We did not have worry about the microcontroller providing too much data to the USB device at one time for it USB device operates in the Mhz range, while the microcontroller has only a kHz clock.

The first step in hooking up the device is wiring it correctly on a bread board or circuit board. The device cannot be powered on and installed on the computer till this is accomplished. The first step is to open up the wiring diagram found on page 6 of the schematic located at http://www.dlpdesign.com/usb/dlp-usb245m12.pdf. First all of the pins labeled GND, are connected to ground. Second the VID and EXT pins are connected together to provide power to the USB device from the computer. If a outside power source is needed, then it can be connected to VID. Check to make sure that the voltage you hook up to the board is less than 6 V and the current is less than 500 milliamps. Next tie VCC to pin 3, the reset. Finally connect the data pins to the outputs on your microcontroller and the TXE and RD pins to your microcontroller to control the I/O operations.

The installation of the device is rather simple. First you can download the drivers from http://www.dlpdesign.com/#Drivers. You can either decide to use the Virtual com port drivers to communicate with the device using standard modem commands are you can install the drivers and the windows DLL which allows you to communicate with the device easily within a code base. Once the drivers are installed, you will need to install the device by plugging it into the computer using the USB cable. Windows will detect the device and prompt you to specify which
driver to install. You can either have windows search for the driver, or you can specify the file location where you download the drivers too. I have noticed that every now and then that the drivers will have to be installed for no apparent reason. Researching this, I found it was a known bug and a patch is available. The patch is included in the files we provide on the CD.

This particular device uses bi-directional a first in, first out buffering system that passes data to the computer and stores it within the computer in a 64k byte buffer. The computer can send data to the USB device as well, where it is stored till it is read out by the microcontroller. To determine if the device has data available on the computer, the RD pin can read from the microcontroller and the RD memory register can be read by the DLL on the computer. When sending data to the computer, the microcontroller and computer can check each other TXE bit to determine if there is room in the current packet to transmit additional data.

FIFO USB Device (www.dlpdesign.com)

To interface with the with the USB Device, a C# DLL library is provide by the website and is included in our documentation on the CD-ROM. This library allows any C# .NET application to communicate with the USB Device.

To connect the DLL, you must add the AID.dll to the project references and ensure that the FTD2XX.dll is located in the directory connecting the application. Using classes in C#, the device can communicate with a C# application.

In the case you need to just capture data off of the USB device, you can use the applications we received from DLP Design. There is version 1.0 and version 2.0. version 1.0 is free and can be found at:


The version 2.0 is able to change the flash memory that it has on board, and execute any of the USB functions that the newest version of the DLL provides. This costs 22 dollars, but we went ahead and bought it and included it on the CD.

The most useful features of the device is that it can clear the buffer on the computer quickly, if any data is cached by another program. The program can also generate text documents that can be used to record inputs.
Last, we did experience problems with getting the device to connect to the computer at times due to outside interference. It seems that the device can be easily disrupted by electrical cabling since the device and the cable. To get around this, we had to disconnect the device from the computer and plug it back in occasionally and had to raise the USB cable away from the power cables.

User’s Manual

Installing Software
In order to see meaningful output from the sensors, the Graphical User Interface software, called WeaSeL, must be installed. Run WeaSeL.exe off of the CD.

Applying Sensors

Temperature
Attach the blue end of the thermister to the tip of the middle finger by wrapping it with velcro. See Figure 1.

Blood Volume Pulse (BVP)
The BVP sensor with the green velcro should be placed around the index fingertip. Make sure the red light and the photodiode are touching the fingerpad at a 90 degree angle from each other so the light bounces off of the bone and hits the photodiode. The detector of the GSR sensor with the red velcro should be placed on the large middle joint of the index finger. In this case, the positioning of the light does not matter, but the photodiode should be right next to the bone to capture movement. Make sure both velcro straps are wrapped with the same amount of tightness. See the figure of the hand below.

Skin Conductance
Two self adhesive gel pads should be placed approximately one inch apart on the ring finger. The red GSR lead should be snapped onto the pad nearest the tip of the finger, and the black lead should be snapped onto the other pad. See the figure of the hand below.

The sensors must be powered on by setting both battery switches to the “on” position. You will know the sensor is on when you see the two red lights from the BVP sensors.
Transmission and Display

You are now ready to begin the transmitting data. Power the wireless transmitter by placing the third switch to the “on” position. Start the display by pressing the “Start” button on the WeaSeL display. The display is in real time. To refresh the screen, click the “Clear” button. To stop the display output, click the “Stop” button.

GUI Menu

SPREADSHEET FILE OPERATIONS

WeaSeL has the ability to save data to a spreadsheet (in CSV format) for future reference.

* File -> View saved data
  Opens the active spreadsheet in Excel (or whatever program is associated with .csv files), or displays an error message if there is no active spreadsheet.

* File -> Save
  Writes the average reading of all the sensors to the end of the active spreadsheet.

* File -> Save As...
  Sets the active spreadsheet to the file selected by the user, and writes a header plus the average reading for all the sensors.

EXIT

* File -> Exit
  Quits the program, and saves the graph settings (color, time range, and vertical range) to the file "WeaSeL.exe.config".
**Graph Start/Stop Clear Operations**

* Graphs -> Start/Stop  (or "start/stop" button)
  Pauses or resumes the display of the graph. (Bug: This can make the USB buffer fill up).

* Graphs -> Clear  (or "clear" button)
  Erases all graphs, and resets all the variables that are used to calculate AverageReading.

**Graph Settings Operations**

* Graphs -> Zoom -> In
  Changes the vertical range of the graph to focus on a smaller range.

* Graphs -> Zoom -> Out
  Changes the vertical range of the graph so that you can see more of the graph.

* Graphs -> Zoom -> Shorter Time Range
  Changes the horizontal scale of the graph to display fewer data points.

* Graphs -> Zoom -> Longer Time Range
  Changes the horizontal scale of the graph to display more data points.

* Graphs -> Time range...
  Opens a dialog that lets you chose how many seconds of data to display.

* Graphs -> Options...
  Opens a dialog that lets you change the color or vertical range of any of the 3 graphs.

**Miscellaneous**

* Help -> About
  Explains what "WeaSeL" stands for, and displays the names of our team members.

---

**Course Debriefing**

**Group management style**

We felt as though our team management style was successful in that we were able to effectively work together toward the same goal. There were difficulties in the group, mostly dealing with the size of our team. Having eight people working on different aspects of the same project made project implementation difficult. Each team member knew their portion on the project (i.e. sensors, transmission board, and software) but was not as familiar with the other portions of the project and the constraints needed for later implementation. This was an obstacle that needed to be overcome. If we had the project to do over again, this would be something addressed at an earlier stage of the semester.
Safety and ethical issues

There is a safety concern with directly plugging the sensors into the wall when they are connected to a human being. This obstacle was overcome by using batteries as the power system for the sensors. While this made the sensor device more bulky (which was against one of the project objectives), it was the only way to regulate the power to ensure safety of the user.

Product testing

In order to verify the design of our product, we first tested each component separately. After verifying that each component works, we put them together and tested them as a whole product. In component testing, we did the following:

- Temperature Sensor – The thermistor was immersed in hot water, and temperature changes were recorded as the water cooled. These temperatures were cross referenced with measurements made using a commercial thermometer.
- Galvanic Skin Conductor – The resistance of dry skin was measured using a multimeter. The resistance of wet skin was measured using a multimeter to verify that the resistance dropped after wetting the finger. Conductance is the inverse of resistance.
- Blood Volumetric Pulse – The blood volume pulse was tested using an oscilloscope to verify that the output wave was clean and the frequency measured to be between 1 and 3 Hz. We also used the oscilloscope to calibrate the machine for demonstration purposes.
- Transmission – no direct way to test because of no signal analyzer. Either it all worked or not at all. Fpga &xilinx to capture and process signals from chipcon
- Microcontroller – used jtag debugging to verify current register values as code ran; multimeter used to determine if pins were high/low; logic analyzer to verify timing diagrams
- GUI – function generator used to produce waves
- USB – used built-in software to verify it's working; used dll’s provided online to test buffer, determine size of buffer, and ensure captured data
- Overall Product – The following is how we demonstrated the product. Please reference the video included on the CD and final powerpoint presentation.
  - Temperature Sensor – The system was tested at body temperature (~94 degrees), room temperature (~75 degrees), cold water, and hot water temperatures. This was a broad range of temperatures that should allow for all human temperature variations.
  - Galvanic Skin Conductor – The system was tested with dry and wet skin to view the differences in skin conductance. These tests should cover all possibilities for human skin conductance.
  - Blood Volumetric Pressure – The system was tested while breathing normally, while holding your breath, and while breathing rapidly. With extreme exercise, a human could go over the range
tested with the product, which could go beyond the voltage range allowed with the product.

The entire product works as advertised except for the wireless transmission, which is no longer part of the product. We did not get the chance to conduct the tests proposed in the proposal. In the proposal, we wanted to come up with a set of exercises that would adequately test the temperature, galvanic skin conductance, and blood volume pulse sensors by increasing skin temperature, increasing sweat output, and raising heart beat. If given additional time, we would have done this and tested on people with different physical characteristics to verify our product can be used by any person.

Testing could be more thorough by increasing the range of pulse allowed within the system. We could also have tested the system for longer periods of time, since testing was usually limited within a few hours timeframe. For medical purposes, someone using the product might want to wear the device for a longer period of time. Since the sensors had to be calibrated for each individual person, we kept the calibration set for one person during testing. If more time had been available, we perhaps would have been able to set up a system where calibration for each person would not be so time consuming. A better logic analyzer for better timing diagram would also have allowed for better testing situations. A frequency analyzer to test transmission might have allowed us to isolate the issue with transmission to get that part of the project working. We would also have liked to test transmission with an actual wearable environment (such as on an exercise bike or sleeping).

Another portion of the project, which is actually going to be completed this summer by Clay and Adam, is to improve the program to actually capture chipcon control signals.
Additional steps taken if given more time

The transmission aspect of our project gave us the most difficulties throughout the design and implementation process. If we did the project again, we would spend more time doing preliminary research on the available methods of wireless transmission. This would allow us to choose a wireless device that is better suited for our needs. We would also purchase the testing and evaluation software and hardware necessary to ensure it works properly.

The sensors system was very sensitive from the beginning stages of the project. More research on the reasoning for this sensitivity may have allowed us less time attempting to get the board design completed successfully. Now knowing the obstacles of the project, our initial research could have concentrated more heavily on these issues.
# Wearable Sensor Timeline

**Beginning 03/08/04 (CDR)**

## Project: Wearable Sensors

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical Design Review</td>
<td>03/08 to 03/15</td>
</tr>
<tr>
<td>Complete Board Design / Testing</td>
<td>03/15 to 03/22</td>
</tr>
<tr>
<td>Send Schematic for Board Creation</td>
<td>03/22 to 03/29</td>
</tr>
<tr>
<td>Continue/Transmission Software</td>
<td>03/29 to 04/05</td>
</tr>
<tr>
<td>Spring Break</td>
<td>04/05 to 04/12</td>
</tr>
<tr>
<td>Specify Inputs / Outputs for Sensors</td>
<td>04/12 to 04/19</td>
</tr>
<tr>
<td>Sensors Testing / Calibration</td>
<td>04/19 to 04/26</td>
</tr>
<tr>
<td>Receive Board / Testing</td>
<td>04/26 to 05/03</td>
</tr>
<tr>
<td>Connect Sensors to Board Design</td>
<td>05/03 to 05/10</td>
</tr>
<tr>
<td>Connect Board Design to Wireless Transmission</td>
<td>05/10 to 05/17</td>
</tr>
<tr>
<td>Create GUI to View Information on Computer</td>
<td></td>
</tr>
<tr>
<td>Connect All Components</td>
<td></td>
</tr>
<tr>
<td>Debugging of Software / Hardware Components</td>
<td></td>
</tr>
<tr>
<td>Testing of Entire System</td>
<td></td>
</tr>
<tr>
<td>Completion of Final Project</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Week</td>
<td>03/08 to 03/15</td>
</tr>
<tr>
<td></td>
<td>03/15 to 03/22</td>
</tr>
<tr>
<td></td>
<td>03/22 to 03/29</td>
</tr>
<tr>
<td></td>
<td>03/29 to 04/05</td>
</tr>
<tr>
<td></td>
<td>04/05 to 04/12</td>
</tr>
<tr>
<td></td>
<td>04/12 to 04/19</td>
</tr>
<tr>
<td></td>
<td>04/19 to 04/26</td>
</tr>
<tr>
<td></td>
<td>04/26 to 05/03</td>
</tr>
<tr>
<td></td>
<td>05/03 to 05/10</td>
</tr>
<tr>
<td></td>
<td>05/10 to 05/17</td>
</tr>
</tbody>
</table>

| Current Week                               |                |
**Budget Outline**

**Sensor Boards Outline**

Boards: $75.00 x 2 boards -- they actually sent us 6
Parts: total $25.00
Shipping: $25.00

Breakdown:
- 10 - 10k pots $3.26
- 10 - 1Mohm resistor $0.84
- 10 - 150ohm resistor $0.84
- 10 - 10Mohm resistor $0.84
- 10 - 470ohm resistor $0.84
- 10 - 100uF capacitor $6.53
- 10 - 475Ohm resistor $2.75
- 10 - 100uF capacitor $1.90
- 8 - 100uF capacitor $7.65

opamps were free

**Transmission Board Outline**

USB Software- $22.50
Lapaic Transmission- $65.00
Transmission / Microcontroller Board Parts- $250
Board Fabrication- Free

**TOTAL: $537.50**