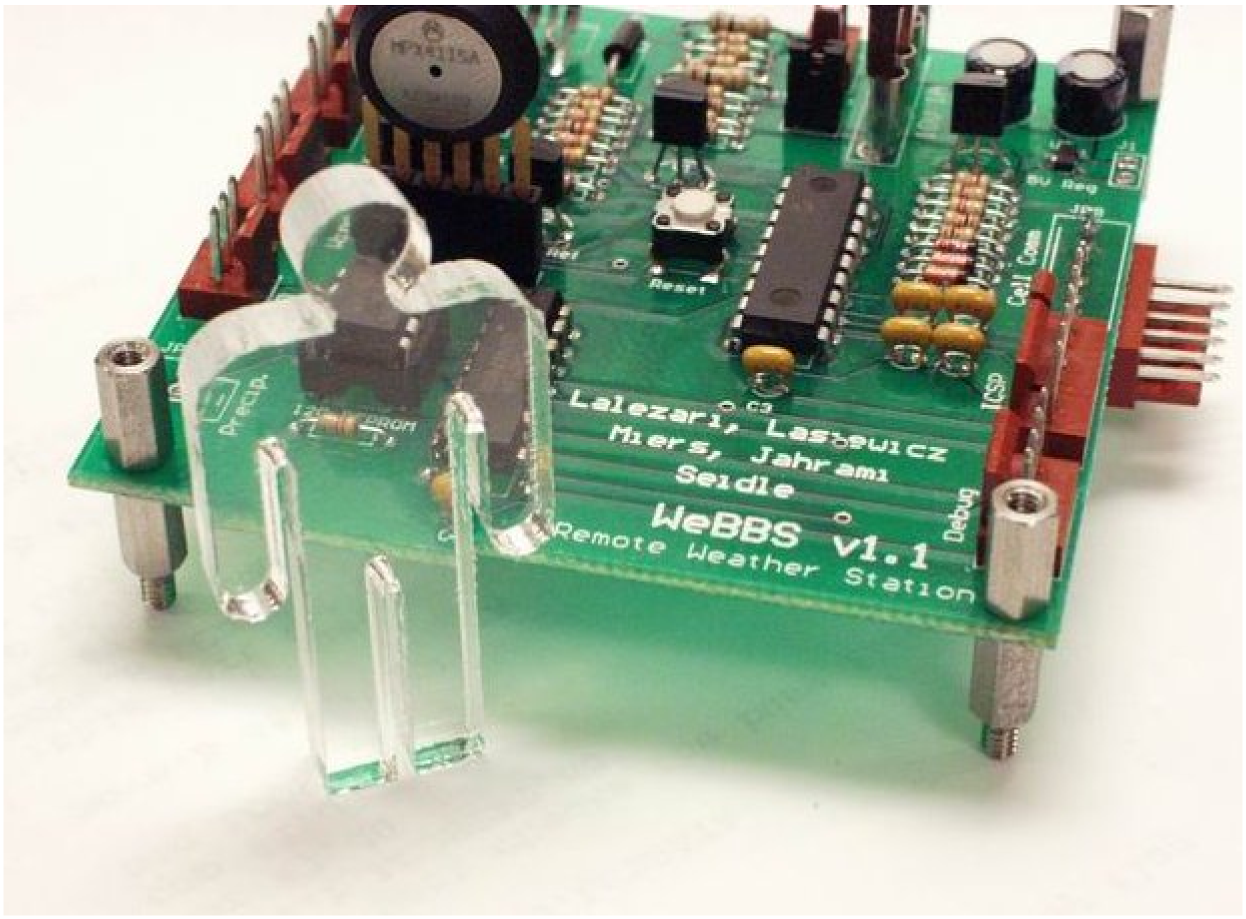
WeBBS

The Weather Bulletin Board System



Technical Manual

Fahid Jahrami, Arian Lalezari, Justin Lasiewicz,
Joshua Miers, Nathan Seidle

Revision: 27 April 2004

Table of Contents

Table of Figures

Table of Equations

Table of Tables

1 Overview

1.1 Abstract

The WeBBS team at the University of Colorado at Boulder designed the Weather Bulletin Board System for autonomous remote weather sensing. The WeBBS weather station is completely powered by a solar cell charging circuit, and communicates wirelessly using cellular telephone technology. The station is programmed to periodically record and store weather conditions from four external sensors: a temperature, pressure, and humidity sensor, as well as an ultrasonic anemometer designed by the WeBBS team to record wind speed and direction.

The weather conditions can be retrieved in three different ways. Users who are on-site may use the on-board LCD display, while off-site users will have two methods to retrieve data. Users with cellular telephones with SMS service will able to send an SMS text message or a voice call to the station, which will record their telephone number and return an SMS message with weather conditions. Users who cannot access an SMS service will be able to browse weather conditions on a website that will be regularly updated by the weather station.

1.2 Motivation

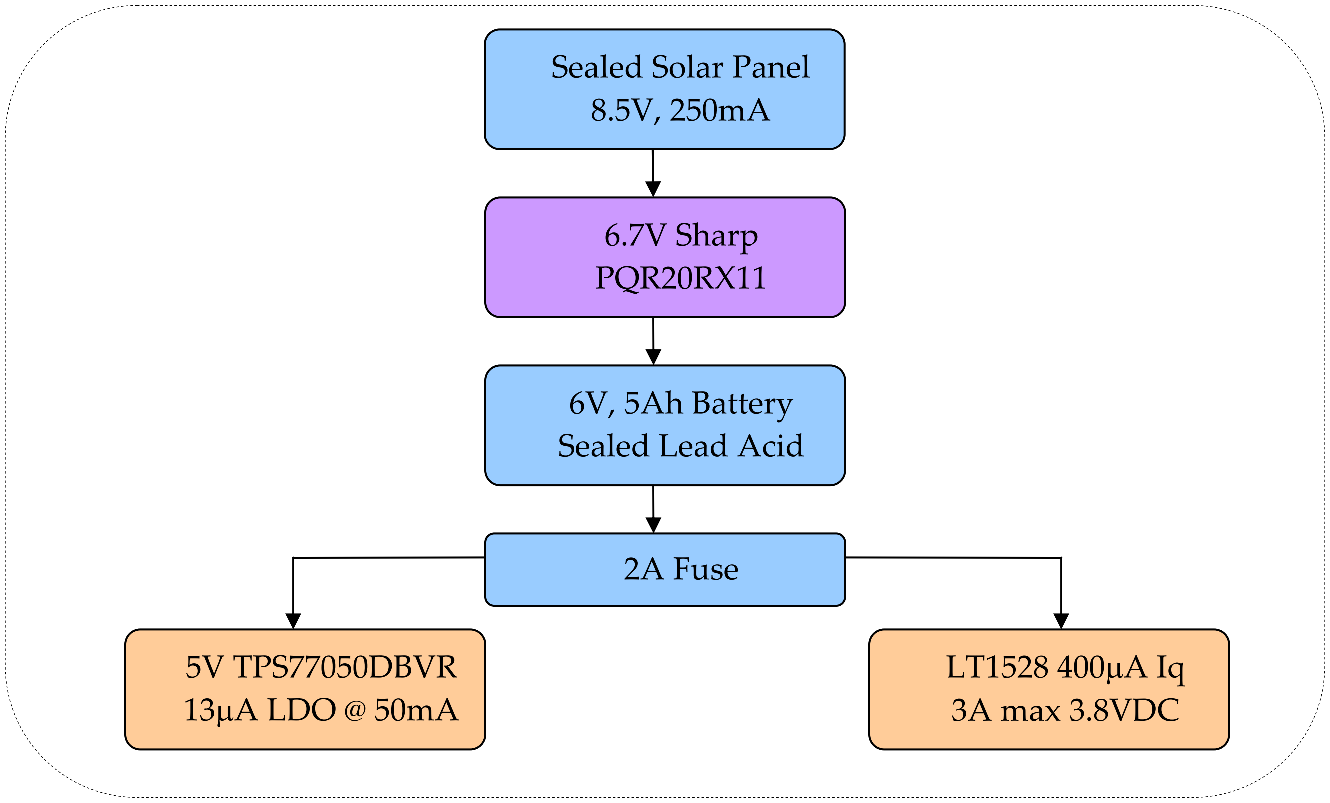
While still a young program at only twelve years old, Colorado Crew has quickly grown from one of the most "underprivileged" rowing programs in the nation to one that regularly challenges established varsity teams in national regattas.

Other than the financial and physical challenges shared by all crew team, the Colorado programs faces an obstacle not common to most established teams: the unpredictable and volatile weather conditions in Boulder. With early morning practices, there is often no way to judge the weather conditions at the Boulder Reservoir without requiring the entire team to arrive at the regular practice time. If practice is cancelled, the entire team is inconvenienced by having had come to the Reservoir, only to return home.

With this in mind, it has become the goal of the WeBBS team to provide an autonomous and financially practical solution for the Colorado Crew team. However, we feel that the applications of our product extend well beyond the scope of the Colorado Crew team. For example, ski resorts in Colorado and elsewhere share the same need for favorable weather to ensure good skiing conditions to their customers. With a weather station at each ski lift, a major resort would easily be able to monitor the conditions throughout their terrain at any time. The same goes for other large-scale outdoor facilities such as airports, amusement parks, golf courses, etc.

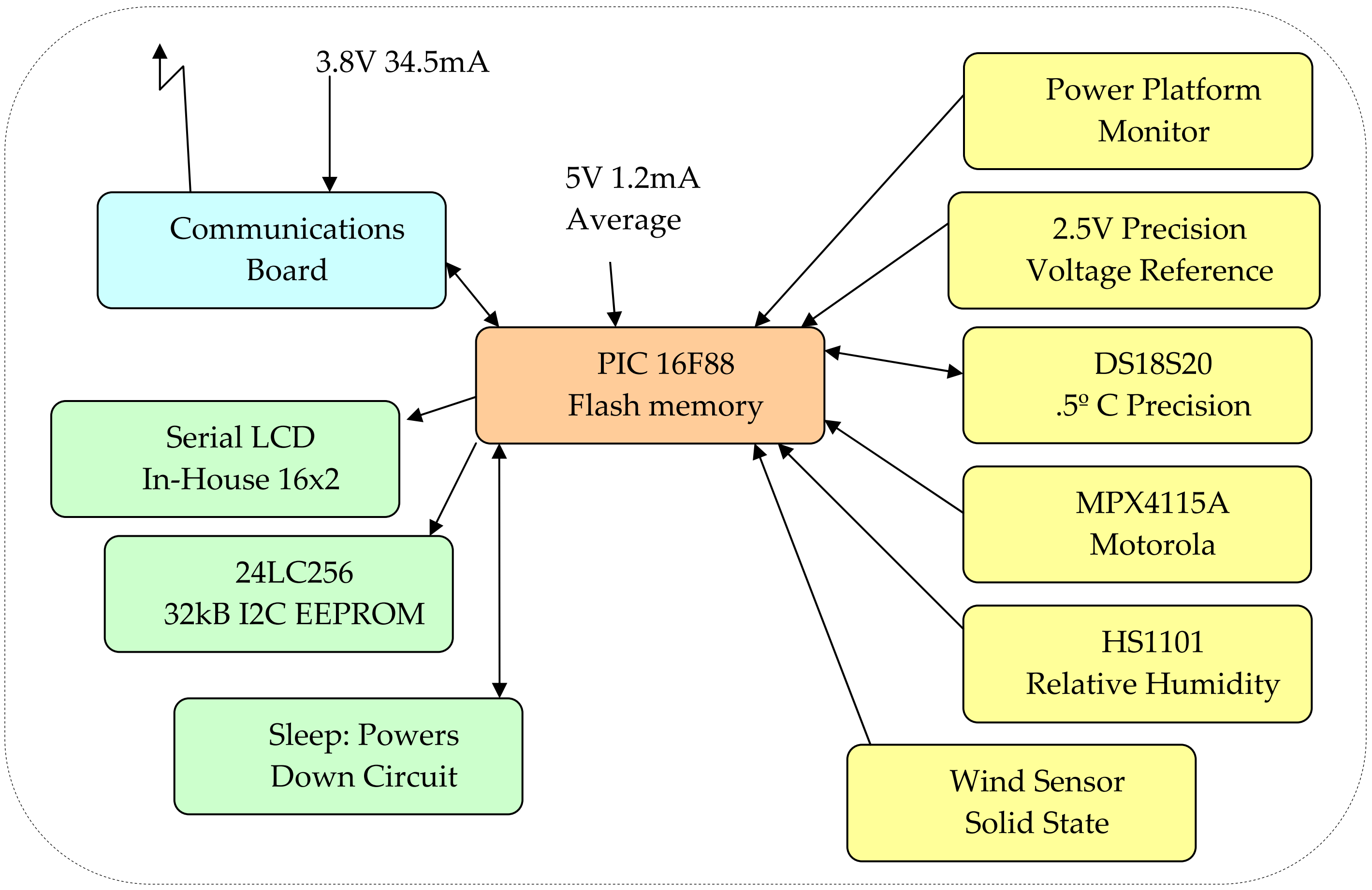
1.3 Block Diagram

1.3.1 Power Platform



Sealed Solar Panel 8.5V, 250mA
6.7V Sharp PQR20RX11
6V, 5Ah Battery Sealed Lead Acid
2A Fuse
5V TPS77050DBVR 13µA LDO @ 50mA
LT1528 400µA Iq 3A max 3.8VDC

1.3.2 Main Board



2 Enclosure

The enclosure of the WeBBS weather station consists of two main components, the main enclosure containing the primary electrical systems and the anemometer enclosure, containing the wind sensor.

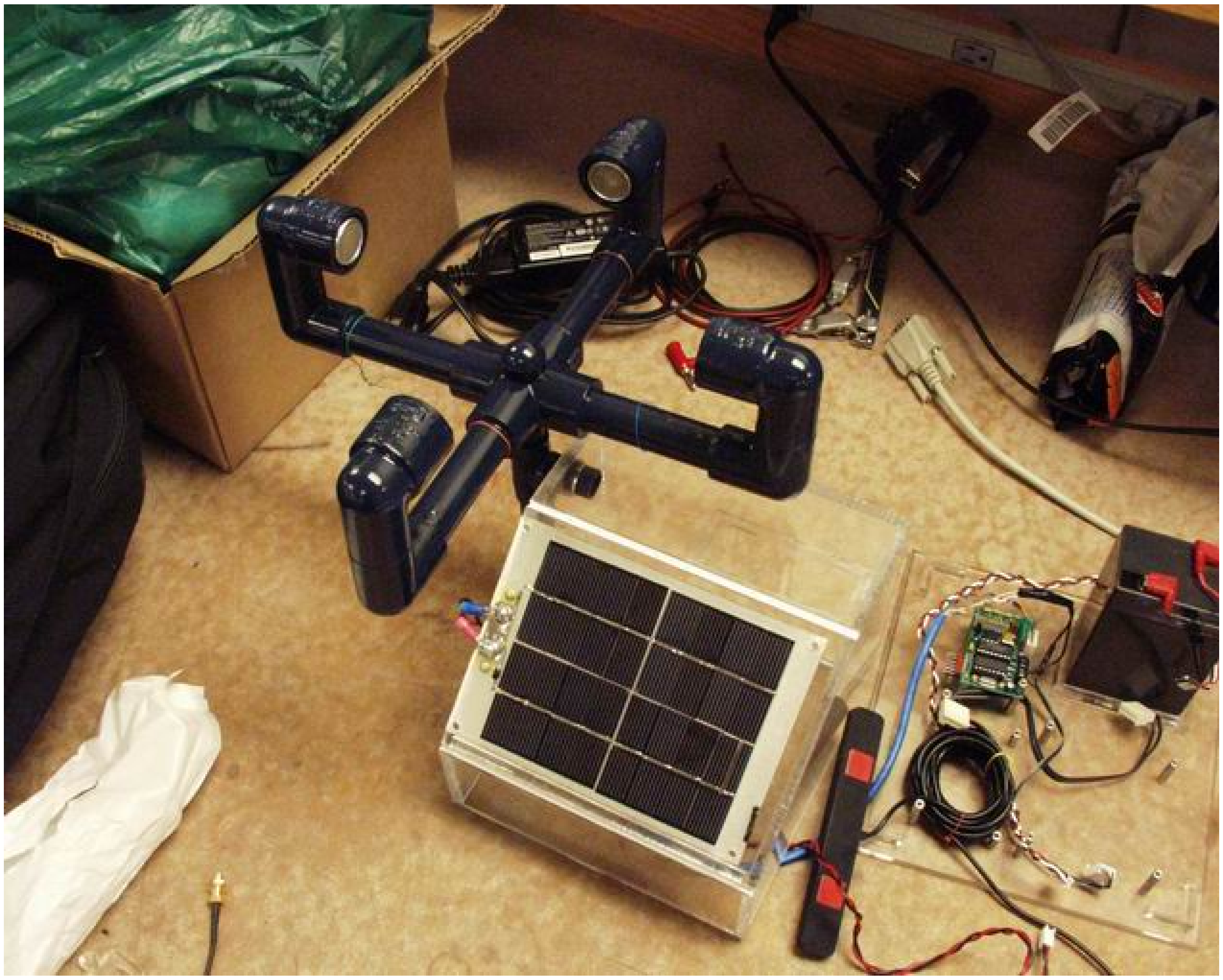


Figure 1: Main Housing

2.1 Main Enclosure

2.1.1 Design Requirement

The primary design goals of the main enclosure are:

1. To minimize the size of the enclosure without compromising the accessibility of individual internal components.

2. To minimize the vulnerability of the internal components to the potentially harmful external weather conditions.

2.1.1.1 Size

The three components presenting the primary restrictions on the size and shape of the main enclosure are the ultrasonic anemometer, the battery, and the solar panel. Because of its location on the side of the main enclosure, the ultrasonic anemometer posed a potential threat of toppling the main enclosure. Furthermore, to adequately fasten the anemometer, we provided that the two fastening locations be at least 3.5 inches apart to maintain stability. For further support, we positioned the battery on the opposite side of the enclosure as a counter-balance. The battery posed similar height requirements for the enclosure to those of the anemometer. The slanted face of the weather station is designed to provide maximal exposure to the sun for the solar panel. Since Boulder is at 40° north latitude, the slanted face is elevated by 40°. Furthermore, the width of the enclosure is determined by the longest edge of the solar panel. Again, the width requirements posed by the inclusion of the printed circuit boards inside the enclosure were similar to those of the solar panel.

2.1.1.2 Weatherproofing

The two components presenting the primary restrictions on the weatherproofing of the main enclosure are the ultrasonic anemometer and the solar panel. To affix the anemometer to the main enclosure, our design requires two holes in one side of the enclosure. Because holes present a weatherproofing challenge, the lower hole is used doubly to affix the anemometer and provide a snorkel to ventilate the interior of the enclosure and provide access to external weather conditions for the temperature, pressure, and humidity sensors. The solar panel presents two challenges to weatherproofing. First, like the anemometer, the leads of the solar panel must be connected to the internal electrical components of the weather station. In order to avoid compromising the structural integrity of the anemometer enclosure, we opt to provide a small hole uniquely for the purpose of the connectivity of the solar panel. Second, because the solar panel must be tilted, we must weatherproof two non-perpendicular seams.

2.1.2 Materials

The main enclosure is constructed entirely from 0.25-inch thick clear Plexiglas. The perpendicular planes of the enclosure are fastened and weatherproofed with Plastruct plastic solvent cement. The holes and non-perpendicular solar panel plane are weatherproofed with weather-resistant caulk.

2.1.3 Construction and Weatherproofing

All planes of the main enclosure were cut using the Radius laser cutter in the Integrated Teaching and Learning Laboratory (ITLL) machine shop at the University of Colorado at Boulder.

All planes were fastened and weatherproofed with Plastruct solvent cement and caulk.

2.1.4 Testing

The main enclosure was tested under extreme high and low temperature and humidity conditions as well as direct sunlight exposure to ensure the sustainability of its resistance to weather and internal moisture.

The enclosure was also tested against stress by applying controlled forces to each of its exposed surfaces.

2.1.5 Drawings

2.1.5.1 Conceptual Drawing

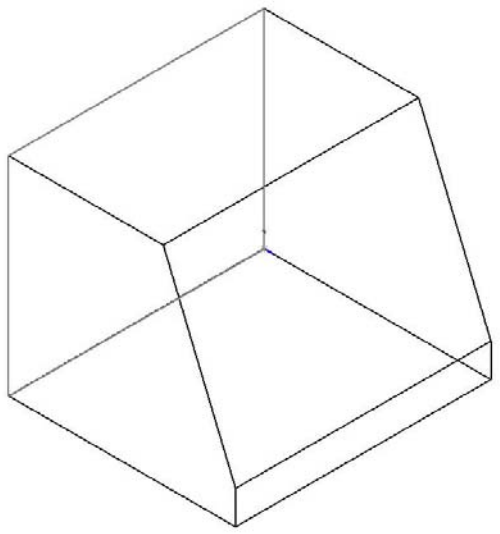


Figure 2: 3-D Conceptual Drawing

2.1.5.2 Parts Drawings

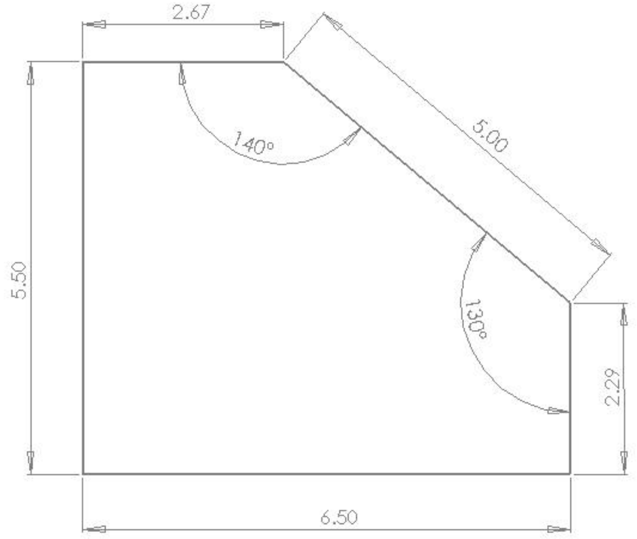


Figure 3: Main Enclosure (Left)

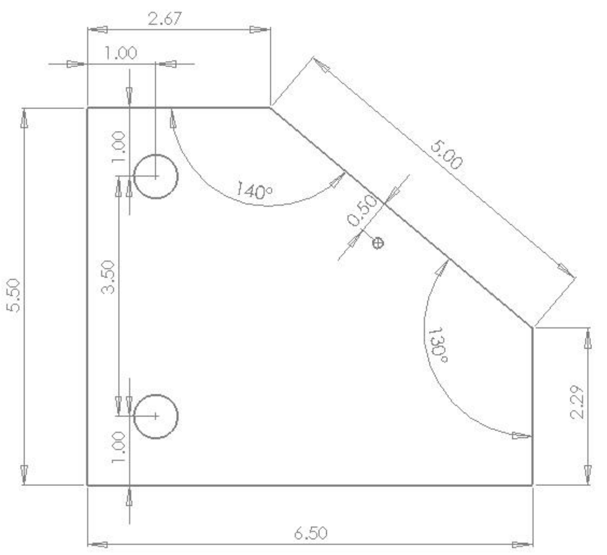


Figure 4: Main Enclosure (Right)

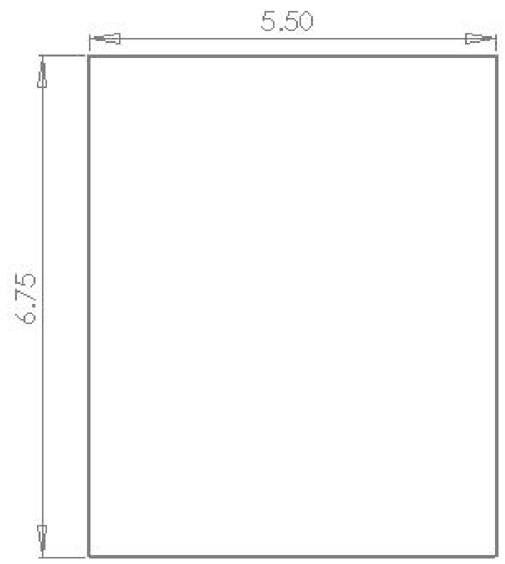


Figure 5: Main Enclosure (Back)

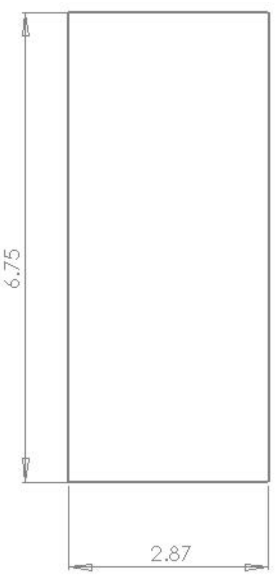


Figure 6: Main Enclosure (Top)

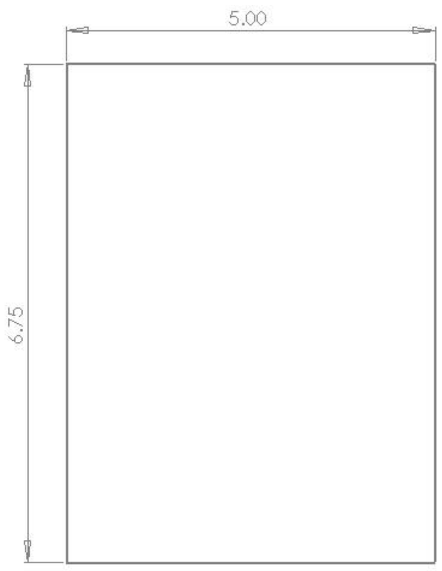


Figure 7: Main Enclosure (Solar)

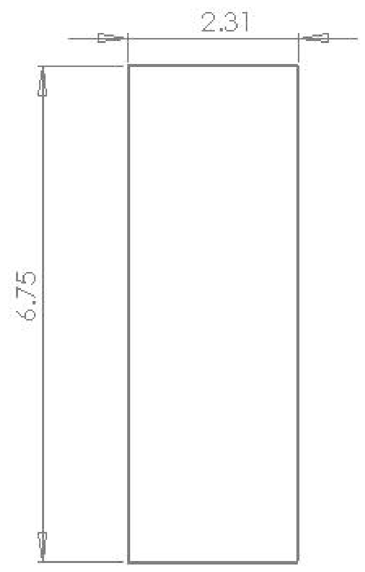


Figure 8: Main Enclosure (Front)

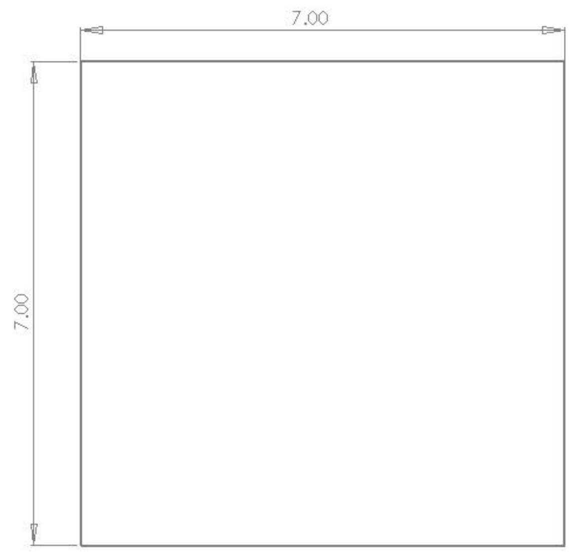


Figure 9: Main Enclosure (Base)

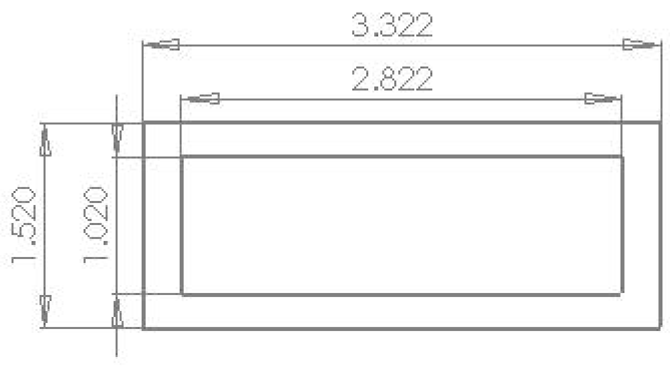


Figure 10: Main Enclosure (LCD Frame)

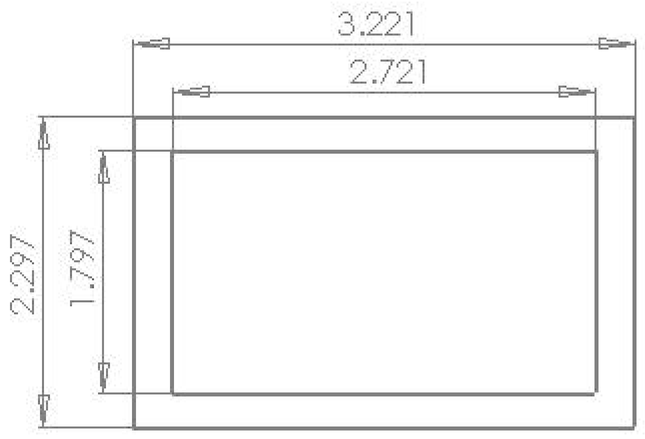


Figure 11: Main Enclosure Battery Frame

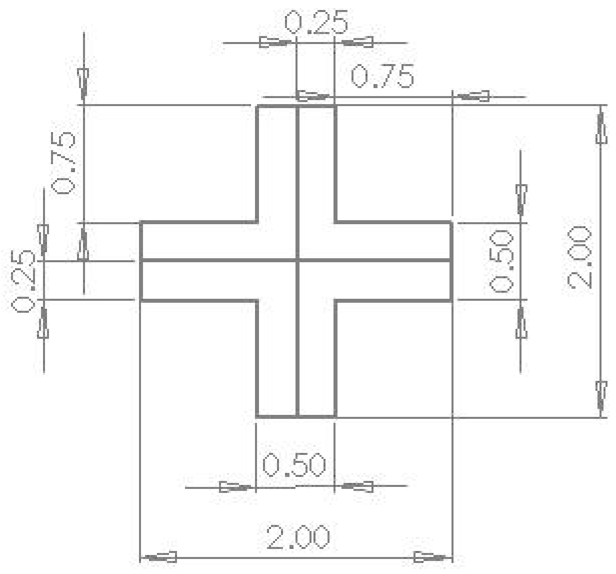


Figure 12: Main Enclosure Braces

2.2 Anemometer Enclosure

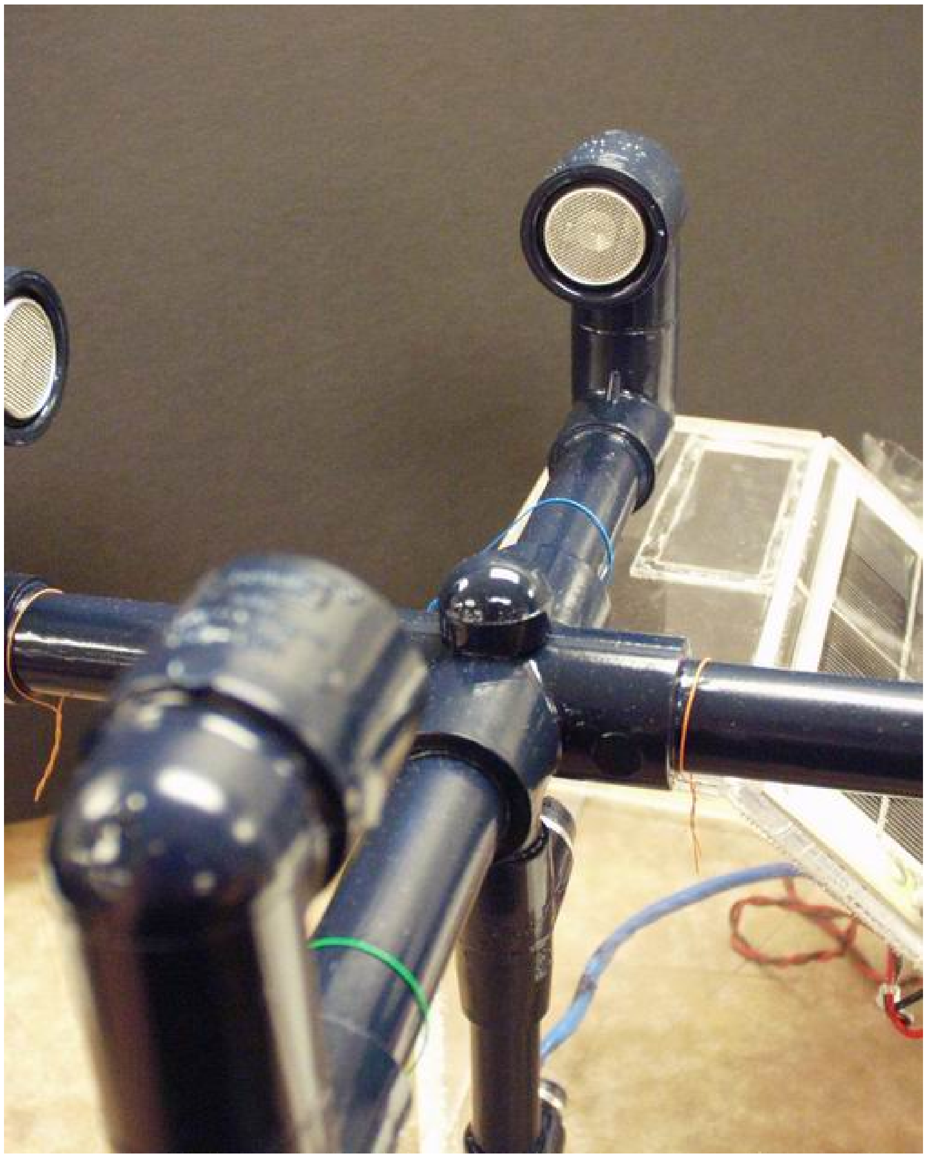


Figure 13: Anemometer Enclosure

2.2.1 Design Requirements

The primary design goals of the anemometer enclosure are:

1. To provide stability for the anemometer assembly and align the pairs of ultrasonic sensors.
2. To weatherproof the critical components of the anemometer.

2.2.1.1 Stability

Because of the required size of the anemometer, it is critical that it remains stable and that the ultrasonic sensors remain aligned. For stability, we provide two locations to anchor the anemometer enclosure to the side of the main enclosure. To maintain alignment, we provide that the wind sensor remains smaller than 12 inches in any horizontal direction, and that all connections are permanently fastened with PVC glue.

2.2.1.2 Weatherproofing

The primary concern for weatherproofing the anemometer enclosure is the protection of the internal wires tying the anemometer to the internal components of the weather station. To do this, all connections are sealed and weatherproofed with standard PVC glue. The ultrasonic sensors are weather resistant and are sealed with two o-rings to protect the inner wires of the enclosure.

2.2.2 Materials

The top assembly of the enclosure requires the use of 0.75-inch PVC pipe, in order to house the ultrasonic sensors. The vertical assembly of the enclosure is constructed with 0.5-inch PVC pipe to facilitate the connections to the main enclosure and the top assembly. The ultrasonic sensors are fastened with o-rings and all seams are sealed with standard PVC glue.

Each transducer is secured at a distance of 7.3″ from its pair.

2.2.3 Testing

The enclosure was tested for weather resistance.

3 Electrical Systems

There are six main electrical systems in the WeBBS weather station: the main system containing the main microcontroller and several peripheral sensors, the cell module system containing the communications platform, the wind module system containing the microcontroller for the anemometer, the LCD system providing easy-to-read on-site data, the power platform consisting of the solar panel, battery, and several voltage regulators, and the website receiver station, which logs recorded weather data to a Internet-available text file.

3.1 Main Board

The main board is responsible for the central processing and monitoring of the weather station. In addition to the primary controller and memory device, the main board contains three of the four weather sensors.

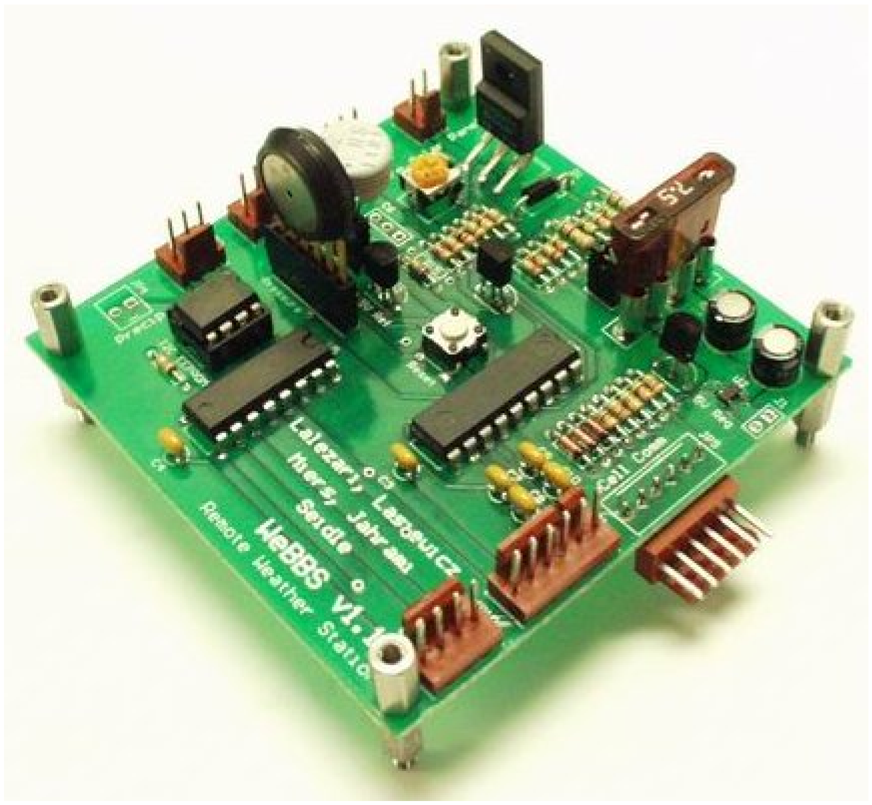


Figure 14: Main Board

3.1.1 Microcontroller

All electrical systems are coordinated by a Microchip 16F88 microcontroller.

The 16F88 features the ability to switch clock speeds in real time. To conserve power, the processor is kept at 31.25kHz except while in data recording or communications mode. In order to quicken data processing and communication, the microcontroller switches to high-speed mode at 8MHz.

The 16F88 has 4kb of on-board Flash program memory. This simplifies the system architecture by minimizing the number of external components required.

The 16F88 features the ability to interface to analog devices via seven software-configurable analog-to-digital channels.

Because the 16F88 has the ability to change its program memory space on the fly, a proprietary boot loader was developed prior to the project to aide in programming and debugging.

All communication protocols (I^2C, Dallas 1-Wire, TTL Serial, RS232) are mimicked with software routines. For debugging purposes, the onboard UART was utilized.

See Appendix A for microcontroller code.

See Data Sheet Packet for extra information on the boot loader and 16F88.

3.1.2 Memory

The main memory device on the board is the Microchip 24LC256. The primary use of the EEPROM is to log recorded weather conditions.

The 24LC256 has 256kb of available flash memory, which gives us enough memory space to record up to 2,000 complete weather station status arrays – or almost one week of weather conditions at 8 bytes per recording.

The 24LC256 interfaces on the I^2C bus protocol.

See Data Sheet Packet for extra information on the 24LC256.

3.1.3 Temperature Sensor

The temperature sensor is an off-the-shelf sensor from Dallas Semiconductor. The DS18S20 measures temperatures from –55°C to +125°C (about -67°F to +257°F) with ±0.5°C accuracy. For our purposes, we have limited the sensitivity of the device to ±1°C (about ±2°F).

The DS18S20 interfaces on the Dallas 1-Wire protocol.

See Data Sheet Packet for extra information on the DS18S20.

3.1.4 Pressure Sensor

The barometric pressure sensor is an off-the-shelf sensor from Motorola. The MPX4115A measures pressure with ±1.5% accuracy between 0°C and 85°C (about 32°F to 185°F). For our purposes, to include a broader range of temperatures, we have limited the sensitivity of the device to ±5%.

The MPX4115A interfaces through the TI 4051 8-channel analog multiplexer to the on-board analog-to-digital converter of the 16F88.

See Data Sheet Packet for extra information on the MPX4115A.

3.1.5 Humidity Sensor

The relative humidity sensor is an off-the-shelf sensor from Humirel. The HS1101 measures relative humidity with ±2% accuracy between –40°C and100°C (about –40°F to 212°F). For our purposes, we have limited the sensitivity of the device to ±5%.

The Humidity is determined through observation of the discharge RC time constant of a variable capacitor conjoined with a 983kΩ resistor. The capacitor is connected to the microcontroller on a pin that can be changed on the fly between being an input and output pin. When the pin is set as an output pin, the capacitor is charged to 5V. It is then discharged through the resistor, after the pin is changed to an input pin. The input pin is a transistor-to-transistor-logic level input. The TTL threshold for the 16F88 is 2V. Therefore, we can determine the internal capacitance of the sensor by using the equation:

Equation 1: Humidity Variable Capacitance

$$C = \frac{-t}{R \cdot \ln\left(\frac{V_f}{V_0}\right)}$$

From the measured capacitance, we can determine the relative humidity using a look-up table derived from the HS1101 data sheet.

See Data Sheet Packet for extra information on the HS1101.

3.1.6 Power Requirements

To conserve power, the main board will actively control the power to each device. Unless the weather station is in recording or communicating mode, all devices will remain powered down, and the microcontroller itself will be in low-frequency sleep mode. Weather will be recorded and logged every five minutes, and communications will occur every 15 minutes.

With only peripheral sensors active, the main board requires 14.3mA to operate. When the microcontroller is set in low-frequency sleep mode, the current consumption of the main board falls to 12.1mA with peripheral sensors active, and only 960µA with all peripheral sensors powered down. Because the main board will only take one five-second reading every five minutes, we will draw an average of 1.2mA of current from the main board.

In order to monitor the current consumption of the board, a pair of leads is available to connect an Ammeter in series with the main power supply of the board. For field operation, this pair of leads will be connected with a jumper.

3.1.7 Schematics

See Appendix B for schematics and PCB layout for the main board.

3.2 Communications Board

The communications board is responsible for operating the wireless relay between the weather station and a remote user.

The communications board houses the Telit GM862-PCS cellular telephone module, powered by the Analog Devices LT1528 voltage regulator.

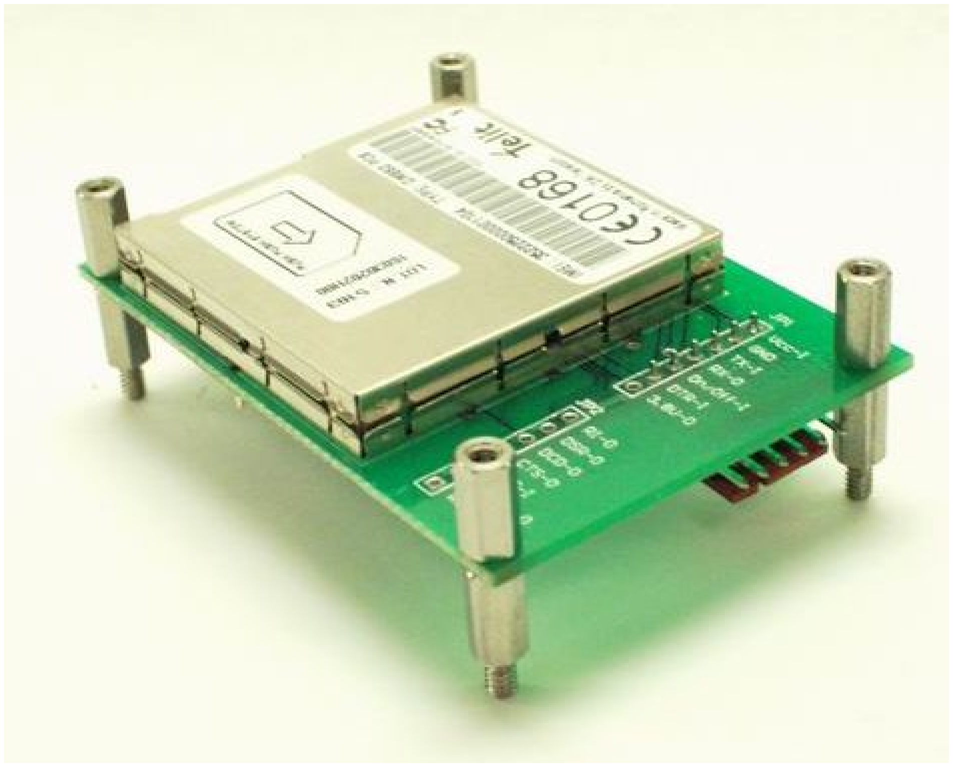


Figure 15: Communications Board

3.2.1 GSM module

The GM862-PCS module has the full suite GPRS abilities: Short Messaging Service (SMS), HyperText Markup Language (HTML), Electronic Mail (E-Mail), as well as standard modem and voice capabilities.

Users must provide a Subscriber Information Module (SIM) card that is activated under a standard GSM cellular telephone account.

Support for the GM862-PCS module can be found by creating an account at the support forum at http://www.modem-gsm.de/forum.

See Data Sheet Packet for extra information on the GM862-PCS.

3.2.2 Tri Band Antenna

The North American PCS network operates on the 1.9GHz band. To connect the GM862-PCS module to this network, we use the Laipac P1-GSM tri-band omni-directional half-wave dipole GSM antenna.

Though no data sheet is available for this device, any standard tri-band GSM antenna will do.

3.2.3 Power Requirements

When powered down, the communications board draws 1.4mA quiescent current. When the GSM module is powered up in sleep mode, the board draws 4.9mA. With full power, the board draws an average of 25mA quiescent, and an average of 250mA when communicating or polling for network registration. Assuming two minutes of call-time (at 250mA) every fifteen minutes, the average current consumption of the communications module is about 34.5mA.

Unless the module is in active communication, it will remain powered down. During periodic communications intervals (every 15 minutes) SMS messages will be replied to, and data will be sent to the website receiver station.

3.2.4 Schematics

See Appendix C for schematics and PCB layout for the communications board.

3.3 Wind Module

The design of the ultrasonic anemometer is based on an existing product available at http://electronickits.com. For our purpose, we have modified the schematic to operate at 25.2kHz, and in two directions, giving us the ability to calculate the magnitude *and* direction of the wind.

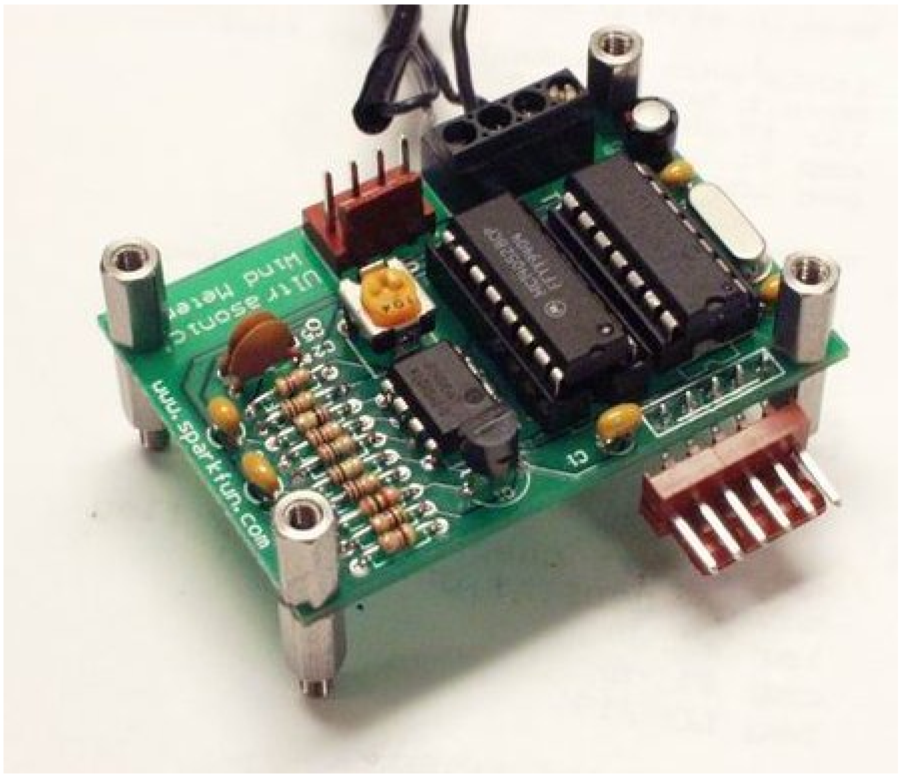


Figure 16: Wind Module

3.3.1 Ultrasonic Transducers

The 25.2kHz ultrasonic transducers were selected because of their availability as surplus parts. The 40kHz transducers are also available at http://www.digikey.com.

3.3.2 Microcontroller

The wind module utilizes the Microchip 16F688 to control each ultrasonic transceiver and calculate wind speed and direction based on recorded data, prior to communicating to the main processor.

The 16F688 uses the CC5X compiler's built-in 24-bit non-standard IEEE floating point operations for all floating point calculations.

See Data Sheet Packet for extra information on the 16F688.

3.3.3 Calculations

The ultrasonic anemometer uses the 16F688 to process recorded data from two pairs of ultrasonic transducers. The pairs are oriented at 90° from one another, at 7.3″ apart: a north-south pair, and an east-west pair.

The first step in calculating wind speed and magnitude is to determine the speed of sound in each of the four cardinal directions. To do this, the microcontroller alternately excites each transducer for 80μs, or about 2 periods, and counts the number of processor clock cycles (at 200ns per cycle) between transmission of a signal from one transducer to its reception at its pair.

For example, if it takes N clock cycles from the transmission time of the south transducer to the reception time at the north transducer, the speed of sound in the northerly direction, in inches per cycle is:

Equation 2: Speed of Sound in North Direction

$$V_N = \frac{\Delta x}{N}$$

Where the quantity Δx is the distance between the north and south transducers: 7.3″.

Similarly, the speed of sound in the south direction, in inches per cycle can be expressed as:

Equation 3: Speed of Sound in South Direction

$$V_S = \frac{\Delta x}{S}$$

To determine the *wind* speed in the northerly direction, we simply subtract the second quantity from the first and divide by two (since both sensors detect a change in the speed of sound when wind is blowing, we must divide by two so that we do not double-count for wind). Our equation for the northerly wind, still in inches per cycle becomes:

Equation 4: Northerly Wind Speed

$$W_N = \frac{V_N - V_S}{2} = \frac{\frac{\Delta x}{N} - \frac{\Delta x}{S}}{2} = \frac{S\Delta x - N\Delta x}{2NS} = \frac{\Delta x}{2}\frac{S - N}{NS}$$

To convert this speed to the more familiar units of miles per hour, some dimensional analysis is required to obtain a conversion constant:

Equation 5: Wind Speed Conversion Constant

$$\frac{1cycle}{200ns}\frac{3.6\cdot 10^{12}ns}{hr}\frac{1mi}{63360in} \cong 284091\frac{cycle\cdot mi}{in\cdot hr} = K$$

Such that the northerly wind speed in miles per hour becomes:

Equation 6: Updated Northerly Wind Speed

$$W_N = K\frac{\Delta x}{2}\frac{S-N}{NS}$$

Because the value of Dx/2 can also be known a priori, an even more robust conversion constant can be derived:

Equation 7: Updated Wind Speed Conversion Constant

$$L = K\frac{\Delta x}{2} = 284091\frac{7.3}{2} \cong 1036932\frac{cycle\cdot mi}{hr}$$

And the final value of the wind speed in the northerly direction in miles per hour becomes:

Equation 8: Final Northerly Wind Speed

$$W_N = L\frac{S-N}{NS}$$

Similar calculations are made for the easterly wind speed, which allows us to determine the overall wind magnitude by applying the Pythagorean theorem:

Equation 9: Wind Magnitude

$$|W| = \sqrt{(W_N)^2 + (W_E)^2}$$

To avoid the implementation of inverse trigonometric functions on the microcontroller, a look-up table is used to determine the trajectory of the wind. First, a preliminary angle between 0° and 85° is determined by comparing the relative magnitudes of W_N and W_E. Special cases are made for a complete lack of wind in either the northerly or easterly direction, and then the ratio of northerly wind magnitude to easterly wind magnitude is computed, and used as input to the following look-up table:

Table 1: Wind Bearing Look-up Table

$\frac{W_N}{W_E}$ min	$\frac{W_N}{W_E}$ max	Angle
-0.043660943	0.043660943	0
0.043660943	0.131652498	5
0.131652498	0.221694663	10
0.221694663	0.315298789	15
0.315298789	0.414213562	20
0.414213562	0.520567051	25
0.520567051	0.637070261	30
0.637070261	0.767326988	35
0.767326988	0.916331174	40
0.916331174	1.091308501	45
1.091308501	1.303225373	50
1.303225373	1.569685577	55
1.569685577	1.920982127	60
1.920982127	2.414213562	65
2.414213562	3.171594802	70
3.171594802	4.510708504	75
4.510708504	7.595754113	80
7.595754113	22.90376555	85

With these calculations, the 16F688 can report wind speed and direction.

3.3.4 Power Requirements

Like the temperature, pressure, and humidity sensors, the ultrasonic anemometer is considered a peripheral sensor, and may only be powered up when weather conditions are recorded.

While recorded and processing wind speed data, the wind module requires 7mA – about half of the power consumption from the main board.

3.3.5 Schematics

See Appendix D for schematics and PCB layout for anemometer board.

See Appendix E for wind sensor code.

3.4 Serial LCD Module

For demonstration purposes and for some field models, an LCD will be made available on the weather-station for on-site retrieval of weather data. The LCD used in our application is the Xiamen Ocular K21602.

See Data Sheet Packet for extra information on the K21602 LCD and the Sparkfun LCD backpack.

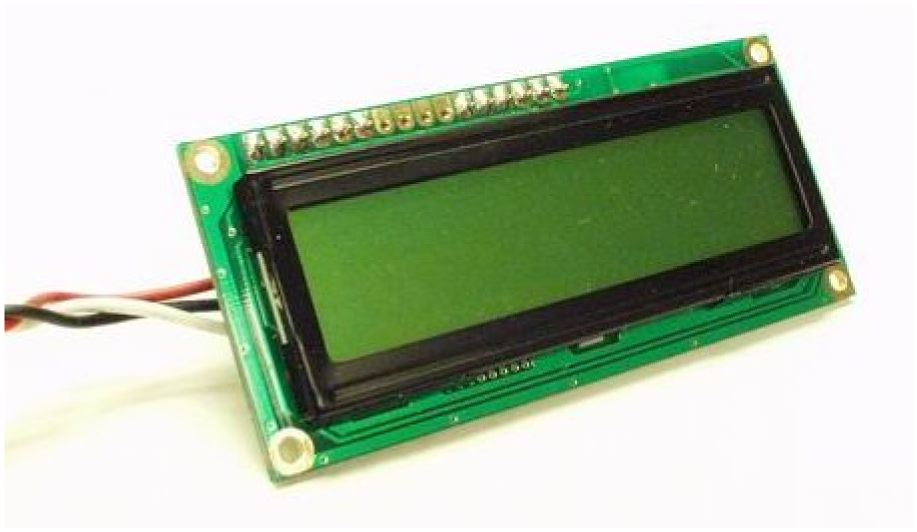


Figure 17: Serial LCD Module

3.4.1 Power Requirements

The LCD module draws 3mA without backlight, and 60mA with backlight. The backlight will be software activated, and will be illuminated for 10 seconds after each weather recording is logged. Because the LCD will only be illuminated for 10 seconds in each five-minute interval, our average current consumption will be 4.9mA.

3.4.2 Schematics

See Appendix F for code, schematics, and PCB layout.

3.5 Power Platform

On average, the WeBBS weather station will draw 1.2mA from the main board, 34.5mA from the communications board, and 4.9mA from the LCD board, for a total average consumption of 40.6mA.

To satisfy this power requirement, the weather station uses a solar panel to constant-voltage charge a high-capacity sealed lead-acid (SLA) battery. The SLA battery is used to power the weather station during poor solar visibility.

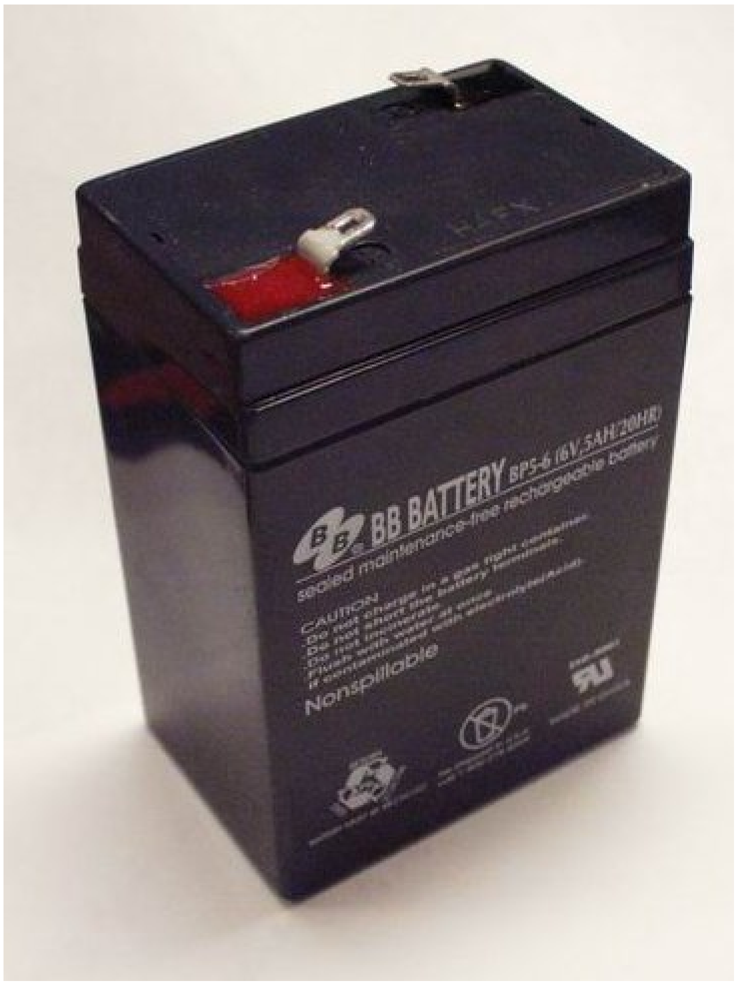

BB BATTERY BP5-6 (6V,5AH/20HR)
sealed maintenance-free rechargeable battery
CAUTION
Nonspillable

Figure 18: Battery

3.5.1 Design Requirements

To be fully autonomous, the WeBBS weather team designed for the weather station to be able to survive three days without solar charging.

Furthermore, power requirements necessitate that the weather station gain power even during the poorest solar conditions in the winter months in Colorado.

Because of variable nature of solar power, it is important that the battery charging circuit is able to regulate the voltage applied to the battery during charging.

Because of the restrictions on power from the necessary components of the WeBBS weather station, it is important that low-overhead, low current draw voltage regulators are used to minimize unnecessary quiescent current consumption.

3.5.2 Solar Panel

A solar panel will be used exclusively to recharge the power of the weather station in the field. The WeBBS weather station uses a high efficiency Solar World SPE-225-6, a 9V, 225mA solar panel rated for up to 2W of power delivery.

See Data Sheet Packet for extra information SPE-225-6.

3.5.3 Battery

The WeBBS weather station is powered with a BB BP5-6-T1 battery. The battery rated for a 5A-hr capacity. At 40.6mA average draw, this battery provides enough power to operate the weather station for 123 hours – over five days with no sun.

See Data Sheet Packet for extra information on the BB BP5-6-TI.

3.5.4 Calculations

With the main board and all peripheral weather sensors fully powered only during readings, and with the communications module powered only during two minutes of call time every 15 minutes, the WeBBS weather station draws an average of 40.6mA – or about 975mA-hr per day.

According the National Renewable Energy Labs (NREL), a typical winter day will have no more than four hours of good solar conditions. Because our solar cell generates 250mA under such conditions, we expect to recharge at least 1A-hr of battery capacity even on days with the poorest of solar conditions. As this is greater than the battery capacity required to operate the weather station each day, we expect to be able to maintain a fully charged battery throughout the winter.

3.5.5 Charging Circuit

When the sealed lead-acid (SLA) battery is near full capacity, the charging circuit on the WeBBS weather station is designed to charge the battery according to a constant-voltage charging scheme. This scheme mimics a trickle charge scheme, but the charging current falls to zero as the battery approaches full capacity. This charging scheme prevents overcharging, and improves the longevity of the battery.

When the battery's capacity falls below 80%, the charging circuit enters a constant-current charging scheme that maximizes the efficiency of the solar cell in its ability to charge the battery. This scheme mimics a flash-charge scheme, and allows the weather station to rapidly regain battery capacity when it falls below ideal levels.

See Appendix G for power platform schematics.

3.5.6 Voltage Regulators

Several voltage regulators are required to provide adequate voltages at rated currents to various components of the weather station. Again, it is critical that these components contribute as little as possible to the current consumption of the overall system.

3.5.6.1 Communications Board

The communications board uses a Linear Technology LT1528 Voltage Regulator. The Telit GM862 module has the interesting requirement of needing up to 2A during peak transmission at 3.8V. Due to the possible high current draw and need for minimum quiescent current to minimize current consumption, a suitable regulator was needed.

The LT1528 solves both these problems with features like 400µA of quiescent current and the ability to source up to 3A with adequate heat sinking.

The LT1528 is set to output 3.3V. This output voltage can be altered using two reference resistors. For our purpose, the Telit datasheet recommends 370Ω and 47Ω. Using these resistors we draw 3.8V / 417Ω = 9.11mA – nearly 32% of our actual overall current consumption. By increasing these resistors by a factor of ten (3.7kΩ and 470Ω), the wasted current consumption is reduced to less than 1mA.

3.5.6.2 Main Board

The main board utilizes a 5V LDO low-noise voltage regulator. The TI TPS77050DBVR 5V regulator has 13µA of quiescent current with very low ripple (less than 50mV). It does all this by sacrificing output current. The TPS regulator can only source up to 50mA. Luckily, our main board never approaches this level of consumption (14.3mA max avg.).

3.5.6.3 Charging Circuit

The SLA battery is charged using the Sharp PQ20RX11 variable voltage regulator. This regulator has a low voltage dropout (LDO) of 500mV maximum and was determined experimentally to be just over 300mV. This means that our solar cell output voltage can dip down to 6.9V (very cloudy) and continue to push small amounts of current into the battery. The charging regulator is set to output 6.3 volts (measured after the diode) to put the battery into constant voltage charging. This diode, on the circuit side of the regulator, is installed to prevent the battery from pushing current backwards through the charging regulator during the night. Backward biasing the charging regulator would stress the part and possibly shorten its lifespan.

3.6 Website Receiver Station

To provide weather information to customers without access to SMS devices, the WeBBS weather station is able to maintain and regularly update a website with current weather conditions.

3.6.1 Overview

Simultaneously connected to a phone line via a traditional modem and to the internet via an Ethernet connection, the website receiver station is able to automatically receive and log incoming data calls from the weather station, and store the data as a Web-available text file.

3.6.2 Code

See Appendix H for ProComm and C++ routines.

4 Parts List

4.1 Main Board

Table 2: Main Board Bill of Materials

Weather Station Bulletin Board System						25 Units		100 Units	
Manufacturer Part #	Digikey Part #	Description	Designator	Package	Qty	Price	Sum	Price	Sum
Logic									
PIC16F88-I/SS	PIC16F88-I/SS-ND	16F88 PIC microcontroller	U3	DIP-18	1	$2.66	$2.66	$2.47	$2.47
		.1uF Microcontroller Support Capacitor	C3	TH	1	$0.10	$0.10	$0.06	$0.06
		Momentary Reset SPST	S1	TH	1	$0.10	$0.10	$0.10	$0.10
		10K Pull-Up Resistor +/-5%	R12	TH	1	$0.00	$0.00	$0.00	$0.00
22-23-2061	WM4204-ND	ICSP Header	JC1	HXDR1X5	0	$0.53	$0.00	$0.00	$0.00
24LC256-I/SM	24LC256-I/SM-ND	256kBit I2C EEPROM	U4	DIP-8	1	$1.74	$1.74	$1.47	$1.47
		4.7K SDA Pull-Up Resistor +/-5%	R13	TH	1	$0.04	$0.04	$0.04	$0.04
		4-Pin Molex Connector - Debugging Comm	JP10	CM4	1	$0.25	$0.25	$0.25	$0.25
Basic Sensors									
CD74HCT4051E	296-2119-5-ND	8 Channel Analog Multiplexor	U5	DIP-16	1	$0.45	$0.45	$0.45	$0.45
		.1uF Support Capacitor	C4	TH	1	$0.10	$0.10	$0.06	$0.06
HS1101	HS1101-ND	HS1101 Humidity Sensor	C5	HDR1x3	1	$8.65	$8.65	$5.69	$5.69
		820K +/-1% Resistor	R14	TH	1	$0.05	$0.05	$0.03	$0.03
		Calibration Capacitor	C6	HDR1x3	1	$8.65	$8.65	$5.69	$5.69
		820K +/-1% Resistor	R15	TH	1	$0.05	$0.05	$0.03	$0.03
MPX4115A	MPX4115A-ND	MPX4115A Pressure Sensor	U7	HDR1x6	1	$13.78	$13.78	$12.98	$12.98
DS18S20	DS18S20-ND	DS18S20 Temperature Sensor	JP1	CM3	2	$3.80	$7.60	$2.28	$4.56
		4.7K Pull-Up Resistor +/-5%	R16	TH	1	$0.00	$0.00	$0.00	$0.00
LT1009CLP	296-9585-5-ND	Precision 2.5V Zener	D2	TO-92	1	$0.63	$0.63	$0.50	$0.50
		10K Bias Resistor +/-5%	R17	TH	1	$0.00	$0.00	$0.00	$0.00
Charging Power Supply									
SPE-225-6	---	9V 225mA Solar Cell	JP8	CM2	1	$49.00	$49.00	$49.00	$49.00
BP4-6-T1	---	6V 5AHr Lead Acid Battery	JP7	CM2	1	$6.45	$6.45	$6.45	$6.45
		Fuse	F1	TH	1	$0.25	$0.25	$0.25	$0.25
PQ20RX11	425-1722-5-ND	1A Adjustable LDO Regulator	U1	Special	1	$1.32	$1.32	$0.96	$0.96
		43K Power Control Pull-Up Resistor +/-5%	R18	TH	1	$0.00	$0.00	$0.00	$0.00
		1K Adjustment Resistor	R1	TH	1	$0.04	$0.04	$0.02	$0.02
		10K Adjustment TrimPot	RT1	TH	1	$0.04	$0.04	$0.02	$0.02
		Forward Control Diode 4001	D1	TH	1	$0.04	$0.04	$0.04	$0.04
		Peripheral Power Control GP NPN BJT	Q1	TO-92T	1	$0.05	$0.05	$0.05	$0.05
TPS77050DBVR	296-11052-1-ND	5V 150mA LDO Low IQ	U2	SOT-23A	1	$0.73	$0.73	$0.58	$0.42
		10uF/25V Filtering Capacitor - Raw	C1	TH	1	$0.18	$0.18	$0.12	$0.12
		10uF/25V Filtering Capacitor - 5V	C2	TH	1	$0.18	$0.18	$0.12	$0.12
		1Ohm Moitor Resistor - Stage 1-2	R2	TH	1	$0.04	$0.04	$0.02	$0.02
		100K Monitor Resistor - Stage 1	R3	TH	1	$0.04	$0.04	$0.02	$0.02
		43K Monitor Resistor - Stage 1	R4	TH	1	$0.04	$0.04	$0.02	$0.02
		100K Monitor Resistor - Stage 2	R5	TH	1	$0.04	$0.04	$0.02	$0.02
		43K Monitor Resistor - Stage 2	R6	TH	1	$0.04	$0.04	$0.02	$0.02
		1Ohm Moitor Resistor - Stage 2-3	R11	TH	1	$0.04	$0.04	$0.02	$0.02
		100K Monitor Resistor - Stage 3	R7	TH	1	$0.04	$0.04	$0.02	$0.02
		43K Monitor Resistor - Stage 3	R8	TH	1	$0.04	$0.04	$0.02	$0.02
		100K Monitor Resistor - Stage 4	R9	TH	1	$0.04	$0.04	$0.02	$0.02
		43K Monitor Resistor - Stage 4	R10	TH	1	$0.04	$0.04	$0.02	$0.02
Cellular Data Module									
		Cellular Data Module	---	---	1	$120.00	$120.00	$120.00	$120.00
		5-Pin Molex Connector - Cellular Connection	JP9	Special	1	$0.25	$0.25	$0.25	$0.25
		TX Diode Drop	D3, 4, 5	TH	3				
		TX Caps .1uF Universal	C7, 8	TH	2				
		DTR Diode Drop	D6, 7, 8	TH	3				
		DTR Caps .1uF Universal	C9, 10	TH	2				
		On/Off Transistor	Q2	TO-92T	1				
		47K On/Off Resistors	R19, 20	TH	2				
		10K Pull Down Resistors for DTR/TX	R21, 22	TH	2				
Ultrasonic Wind Sensor									

---	---	Ultrasonic Wind Meter Module	JP2	CM4	1	$10.00	$10.00	$10.00	$10.00
Serial LCD									
---	---	2x16 Serial LCD Module	JP3	CM3	1	$10.00	$10.00	$10.00	$10.00
Precipitation Sensor									
---	---	2 Wire Precipitation Sensor	JP4	CM2	1	$0.00	$0.00	$0.00	$0.00
PCB									
Olimex	2-3 Weeks	$26 Per Panel - 6.3"x3.3" - 12 Boards	--	--	1	$2.80	$2.80	$2.80	$2.80
Enclosure									
		$40 per 3 + $15 for Assembly			1	$18.33	$18.33	$18.33	$18.33
						Total:	$264.91	Total:	$253.45

4.2 Wind Sensor

Table 3: Wind Sensor Bill of Materials

Ultrasonic Wind Meter						25 Units		100 Units	
Manufacturer Part #	Digikey Part #	Description	Designator	Package	Qty	Price	Sum	Price	Sum
Logic									
PIC16F630	PIC16F630	16F630 PIC microcontroller	U1	DIP-14	0	$1.27	$0.00	$1.20	$0.00
PIC16F688	PIC16F688	16F688 PIC microcontroller	U1	DIP-14	1	$1.97	$1.97	$1.87	$1.87
C320C104M5U5CA	399-2155-ND	.1uF Microcontroller Support Cap	C1	TH	1	$0.09	$0.09	$0.09	$0.09
ECE-A1HKS2R2	P994-ND	2.2uF Power Decoupling Cap	C9	RC-TH	1	$0.14	$0.14	$0.11	$0.11
ECS-200-20-4	X439-ND	20Mhz Xtal - HC-49/US	Y1	Xtal-1	1	$0.52	$0.52	$0.41	$0.41
C315C270J2G5CA	399-1891-ND	27pF Xtal Capacitors	C7,C8	TH	2	$0.21	$0.42	$0.15	$0.30
		4-Pin Molex Connector - Sensors	JP2	TH	1	$0.35	$0.35	$0.28	$0.28
ED555/4DS	ED1516-ND	4 Terminal Screw Connector - Comm	JP1	TH	1	$0.72	$0.72	$0.64	$0.64
ED555/4DS	ED1516-ND	4 Terminal Screw Connector - Sensors	JP2	TH	0	$0.72	$0.00	$0.64	$0.00
Ultrasonic Wind Sensor									
---	---	Ultrasonic Sensors - 25kHz		---	4	$1.00	$4.00	$1.00	$4.00
LM358AP	296-9554-5-ND	LM358 Dual OpAmp	U3	DIP-8	1	$0.32	$0.32	$0.24	$0.24
2N3904	2N3904FS-ND	NPN GP BJT Transistor	Q1	TO-92T	1	$0.12	$0.12	$0.09	$0.09
CD74HC4052E	296-9217-5-ND	74LF4052 Analog Multiplexor	U2	DIP-16	1	$0.36	$0.36	$0.27	$0.27
EVN-D8AA03B15	D4AA15-ND	100K Potentiometer	R10	TH	1	$0.25	$0.25	$0.18	$0.18
CFR-12JB-4K7	4.7KEBK-ND	4.7K +/-5% Resistor 1/8 Watt	R9	TH	1	$0.06	$0.06	$0.02	$0.02
CFR-12JB-47K	47KEBK-ND	47K +/-5% Resistor	R1	TH	1	$0.06	$0.06	$0.02	$0.02
CFR-12JB-1K0	1.0KEBK-ND	1K +/-5% Resistor	R11	TH	1	$0.06	$0.06	$0.02	$0.02
CFR-12JB-10K	10KEBK-ND	10K +/-5% Resistor	R4,5,6,7	TH	4	$0.06	$0.22	$0.02	$0.09
CFR-12JB-100K	100KEBK-ND	100K +/-5% Resistor	R2, R3	TH	2	$0.06	$0.11	$0.02	$0.05
CFR-12JB-1M0	1.0MEBK-ND	1M +/-5% Resistor	R8	TH	1	$0.06	$0.06	$0.02	$0.02
D100D20C0GH63L6	1326PH-ND	10pF 15V Capacitor	C6	TH	1	$0.07	$0.07	$0.06	$0.06
D101K20Y5PH63L6	1371PH-ND	100pF 15V Capacitor	C2	CDR	1	$0.07	$0.07	$0.06	$0.06
D102K20Y5PH63L6	1383PH-ND	1000pF-1nF 15V Capacitor	C5	CDR	1	$0.08	$0.08	$0.06	$0.06
C320C104M5U5CA	399-2155-ND	100nF- Universal .1uF	C3, C4	TH	2	$0.09	$0.17	$0.09	$0.17
PVC Housing									
		90 Degree 3/4" to 1/2" Bushing	--	--	4	$0.59	$2.36	$0.39	$1.56
		90 Degree 1/2" L	--	--	4	$0.18	$0.72	$0.18	$0.72
		4-Way Cross 1/2"	--	--	1	$1.95	$1.95	$1.00	$1.00
		1/2" Pipe	--	--	1	$0.30	$0.30	$0.30	$0.30
PCB									
			--	--	1	$1.70	$1.70	$1.35	$1.35
Assembly									
		Manual Paid - 1hr @ 10/hr	--	--	0	$10.00	$0.00	$10.00	$0.00
		Manufactured	--	--	0	$5.00	$0.00	$5.00	$0.00
						Total:	$17.23		$13.99
						Retail:	$34.47		$27.98

4.3 Serial LCD

Table 4: Serial LCD Bill of Materials

Serial Enabled LCD Backpack						25 Units		100 Units	
Manufacturer Part #	Digikey Part #	Description	Designator	Package	Qty	Price	Sum	Price	Sum
Logic									
PIC16F630	PIC16F630	16F630 PIC microcontroller	U1	DIP-14	1	$1.46	$1.46	$1.20	$1.20
C320C104M5U5CA	399-2155-ND	.1uF Microcontroller Support Cap	C1	TH	1	$0.03	$0.03	$0.03	$0.03
ED555/3DS	ED1515-ND	3 Terminal Screw Connector	JP1	TH	1	$0.54	$0.54	$0.48	$0.48
LCD									
CFR-12JB-330R	D4AA14-ND	TrimPot - 10K	R1	TH	1	$0.25	$0.25	$0.18	$0.18
BC337	BC337OS-ND	800mA BJT	Q1	TH	1	$0.21	$0.21	$0.14	$0.14
		2x16 Character LCD w/BL - Olimex	--	--	1	$6.00	$6.00	$6.00	$6.00
		4x16 Character LCD - Timeline	--	--	0	$5.70	$0.00	$5.70	$0.00
		12 Total Connection Pins @ .021	JP2, JP3	TH	1	$0.25	$0.25	$0.25	$0.25
PCB									
		1"x1.75" - $.50/sqin	--	--	1	$0.88	$0.88	$0.88	$0.88
						Total:	$9.62		$9.16
						Retail:	$19.23		$18.32

5 Conclusions

The WeBBS project turned out to be much more complex than we initially anticipated. The design aspects were challenging and rewarding, and allowed each of us to learn new skills and apply them to the project.

Though much of the technology we used was common to at least one group member, the Telit module was something completely new for all of us. Our eventual success with the module was a $240 defining moment for our group and for our project.

Another new aspect for most of us was the development and optimization of a low-power system. This quality of the system manifested itself in almost every component in the design.

While our system is not yet completely deployable, we have reached proof-of-concept, and feel that taking the product to market is a viable next step for the project.

Overall, we feel that this has been a fun, interesting, and rewarding experience with a great group.

6 Acknowledgements

Zach "How to Eat Five Batteries" Miers

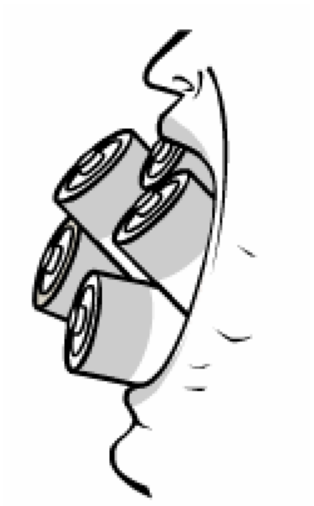


Figure 19: How to eat five batteries

Sarah "We Just Don't Have the Power" Macumber

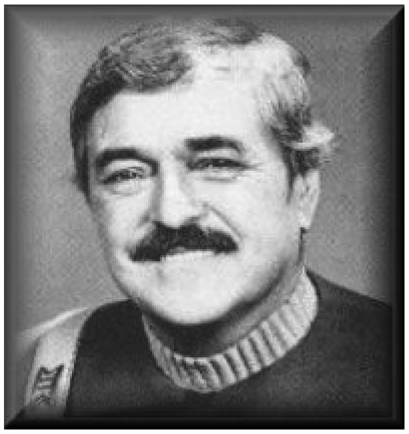


Figure 20: Scotty

Spark Fun Electronics

Spark Fun
ELECTRONICS

Figure 21: Buy our products

Mark Eaton in the machine shop in the ITLL

Appendix A: Main Microcontroller Code

This appendix contains the code from the main board microcontroller (16F88) with annotations.

A.1 Delay.c

These are the most basic routines to cause the processor to delay at various oscillator configurations.

```
/*
    7/23/02
    Nathan Seidle
    nathan.seidle@colorado.edu

    Delays for... Well, everything.

    11-11 Updated the delays - now they actually delay what they say they should.

    10-11-03 Updated delays. New CC5X compiler is muy optimized.

*/

//Really short delay
void delay_us(uns16 x)
{

#ifdef Clock_4MHz
    //Calling with 10us returns 69us
    for ( ; x > 0 ; x--);
#endif

#ifdef Clock_8MHz
    //Calling with 1us returns 11us
    //Calling with 10us returns 56us
    //for ( ; x > 0 ; x--);

    //Calling with 1us returns 7.5us
    //Calling with 10us returns 48
    //Calling with 1000us returns 4.5ms
    while(--x);

    //while(x--);
#endif

#ifdef HS_Osc
    //Calling with 10us returns 13 us
    //Calling with 1us returns 1.8us
    while(--x) nop();
#endif

}

//General short delay
void delay_ms(uns16 x)
{

#ifdef Clock_4MHz
    //Clocks out at 1002us per 1ms
    int y;
    for ( ; x > 0 ; x--)
        for ( y = 0 ; y < 108 ; y++);
#endif

#ifdef Clock_8MHz
    //Clocks out at 1006us per 1ms
    uns8 y, z;
    for ( ; x > 0 ; x--)
        for ( y = 0 ; y < 4 ; y++)
            for ( z = 0 ; z < 69 ; z++);
#endif

#ifdef HS_Osc

    uns8 y, z;
    //Clocks out to 1.00ms per 1ms
    //9.99 ms per 10ms
    for ( ; x > 0 ; x--)
        for ( y = 0 ; y < 4 ; y++)
            for ( z = 0 ; z < 176 ; z++);
#endif

}
```

```
//Delays in 31.25kHz Low Power mode using the internal 31.25kHz oscillator
void delay_s_lp(uns16 x)
{

    uns16 y;
    //Clocks out to 1.001s per 1s
    for ( ; x > 0 ; x--)
        for ( y = 0 ; y < 775 ; y++);
}
```

A.2 StdIO.c

These functions are akin to C++ printf functions. They are written to facilitate I/O on a microcontroller platform, including the hardware UART.

```
/*
    5/21/02
    Nathan Seidle
    nathan.seidle@colorado.edu

    Serial Out Started on 5-21
    rs_out Perfected on 5-24

    1Wire Serial Comm works with 4MHz Xtal
    Connect Serial_Out to Pin2 on DB9 Serial Connector
    Connect Pin5 on DB9 Connector to Signal Ground
    9600 Baud 8-N-1

    5-21 My first real C and Pic program.
    5-24 Attempting 20MHz implementation
    5-25 20MHz works
    5-25 Serial In works at 4MHz
    5-25 Passing Strings 9:20
    5-25 Option Selection 9:45

    6-9  'Stdio.c' created. Printf working with %d and %h
    7-20 Added a longer delay after rs_out
         Trying to get 20MHz on the 16F873 - I think the XTal is bad.
         20MHz also needs 5V Vdd. Something I dont have.
    2-9-03 Overhauled the 4MHz timing. Serial out works very well now.

    6-16-03 Discovered how to pass string in cc5x
        void test(const char *str);
        test("zbcdefghij"); TXREG = str[1];

        Moved to hardware UART. Old STDIO will be in goodworks.

        Works great! Even got the special print characters (\n, \r, \0) to work.


*/

//#define Serial_Out  RA2
//#define Serial_In   RA1

//Clock is defined in Delay.c!!!

//int counter;

//Setup the hardware UART TX module
void enable_uart_TX(bit want_ints)
{

#ifdef Clock_4MHz
    #ifdef Baud_9600
    SPBRG = 6; //4MHz for 9600 Baud
    #endif
#endif

#ifdef Clock_8MHz
    #ifdef Baud_4800
    SPBRG = 25; //8MHz for 4800 Baud
    #endif
    #ifdef Baud_9600
    SPBRG = 12; //8MHz for 9600 Baud
    #endif
#endif

#ifdef Crazy_Osc
    #ifdef Baud_9600
    SPBRG = 32; //20MHz for 9600 Baud
    #endif
#endif

#ifdef HS_Osc
    #ifdef Baud_9600
    SPBRG = 31; //20MHz for 9600 Baud
    #endif

    #ifdef Baud_4800
    SPBRG = 64; //20MHz for 4800 Baud
    #endif
#endif

    BRGH = 0; //Normal speed UART

    SYNC = 0;
    SPEN = 1;

    if(want_ints) //Check if we want to turn on interrupts
    {
        TXIE = 1;
        PEIE = 1;
        GIE = 1;
    }

    TXEN = 1; //Enable transmission
}

//Setup the hardware UART RX module
void enable_uart_RX(bit want_ints)
{

#ifdef HS_Osc
    #ifdef Baud_9600
    SPBRG = 31; //20MHz for 9600 Baud
    #endif

    #ifdef Baud_4800
    SPBRG = 64; //20MHz for 4800 Baud
    #endif
#endif

    BRGH = 0; //Normal speed UART

    SYNC = 0;
    SPEN = 1;

    CREN = 1;

    //WREN = 1;

    if(want_ints) //Check if we want to turn on interrupts
    {
        RCIE = 1;
        PEIE = 1;
        GIE = 1;
    }

}

//Sends nate to the Transmit Register
void putc(uns8 nate)
{
    while(TXIF == 0);
    TXREG = nate;
}

uns8 getc(void)
{
    while(RCIF == 0);
    return (RCREG);
}

//Returns ASCII Decimal and Hex values
uns8 bin2Hex(char x)
{
   skip(x);
   #pragma return[16] = "0123456789ABCDEF"
}

//Prints a string including variables
void printf(const char *nate, int16 my_byte)
{

    uns8 i, k, m, temp;
    uns8 high_byte = 0, low_byte = 0;
    uns8 y, z;

    uns8 decimal_output[5];

    for(i = 0 ; ; i++)
    {
        k = nate[i];

        if (k == '\0')
            break;

        else if (k == '%') //Print var
        {
            i++;
            k = nate[i];

            if (k == '\0')
                break;
            else if (k == '\\') //Print special characters
            {
                i++;
                k = nate[i];

                putc(k);


            } //End Special Characters
            else if (k == 'b') //Print Binary
            {
                for( m = 0 ; m < 8 ; m++ )
                {
                    if (my_byte.7 == 1) putc('1');
                    if (my_byte.7 == 0) putc('0');
                    if (m == 3) putc(' ');

                    my_byte = my_byte << 1;
                }
            } //End Binary
            else if (k == 'd') //Print Decimal
            {
                for(m = 4 ; my_byte > 0 && m < 255 ; m--)
                {
                    temp = my_byte % (uns16)10;
                    decimal_output[m] = temp;
                    my_byte = my_byte / (uns16)10;
                }

                for(m++ ; m < 5 ; m++)
                    putc(bin2Hex(decimal_output[m]));
            }
            else if (k == 'h') //Print Hex
            {
                //New trick 3-15-04
                putc('0');
                putc('x');
                putc(bin2Hex(my_byte.low8 >> 4));
                putc(bin2Hex(my_byte.low8 & 0b.0000.1111));

                /*high_byte.3 = my_byte.7;
                high_byte.2 = my_byte.6;
                high_byte.1 = my_byte.5;
                high_byte.0 = my_byte.4;

                low_byte.3 = my_byte.3;
                low_byte.2 = my_byte.2;
                low_byte.1 = my_byte.1;
                low_byte.0 = my_byte.0;

                putc('0');
                putc('x');

                putc(bin2Hex(high_byte));
                putc(bin2Hex(low_byte));*/
            } //End Hex
            else if (k == 'f') //Print Float
            {
                putc('!');
            } //End Float
            else if (k == 'u') //Print Direct Character
            {
                //All ascii characters below 20 are special and screwy characters
                //if(my_byte > 20)
                    putc(my_byte);
            } //End Direct

        } //End Special Chars

        else
            putc(k);
    }
}
```

A.3 SoftTTL.c

This is written to facilitate bit-banging TTL-level serial communications through software. Note that because these are software routines that depend on the current oscillator frequency of the PIC, multiple copies of the same function must be defined.

```
/*
    7-20-03
    Copyright Spark Fun Electronics 2003

    Software driven UART Communications - NOT RS232!
    Needed for the 16F872 because it has no onboard UART module

*/
//uns8 counter;

//Functions for 4MHz at 9600bps
#ifdef Clock_4MHz
    #ifdef Baud_9600
//===========================================================================
void soft_TTL_wait(void)
{
    int i;
    //for(i = 0 ; i < counter ; i++);
    //for(i = 0 ; i < 36 ; i++); //Works with 16F872 at 20MHz at 3.3V! - Pre registered compiler
    for(i = 0 ; i < 12 ; i++); //Checked 10-30-03 = ~104us
}

//Sends out TTL level nate @ 9600 Baud
void soft_TTL_out(uns8 nate)
{

    uns8 l;

    Soft_Serial_Out = 0;
    soft_TTL_wait();

    for(l = 0 ; l < 8 ; l++)
    {
        Soft_Serial_Out = nate.0;
        nate = rr(nate);
        soft_TTL_wait();
    }

    Soft_Serial_Out = 1;
    delay_us(100); //Hold stop bit for 100us
}
//Reads in TTL level signals @ 9600
uns8 soft_TTL_in(void)
{

    uns8 j, i, nate, counter = 0;

    while (Soft_Serial_In == 1);
    //{
    //    counter++;
    //    if (counter > 245) return(0);
    //}

    //Pause for 104us during start bit.
    //Then pause for ~52us to read middle of first data bit
    for(i = 0 ; i < 111 ; i++);

    for(j = 0 ; j < 8 ; j++)
    {
        nate = rr(nate);
        nate.7 = Soft_Serial_In; //Sample the port, should be in middle of the bit
        for(i = 0 ; i < 72 ; i++);
    }

    return(nate);

}
//===========================================================================
    #endif
#endif
//End 4MHz and 9600bps

//Functions for 8MHz at 9600bps
#ifdef Clock_8MHz
    #ifdef Baud_9600
//===========================================================================

//Sends out TTL level nate @ 9600 Baud
void soft_TTL_out(uns8 nate)
{
    uns8 l;

    Soft_Serial_Out = 0;
    delay_us(22);

    for(l = 0 ; l < 8 ; l++)
    {
        Soft_Serial_Out = nate.0;
        nate = rr(nate);
        delay_us(21);
    }

    Soft_Serial_Out = 1;
    delay_us(100); //Hold stop bit for 100us
}
//Reads in TTL level signals @ 9600
uns8 soft_TTL_in(void)
{

    uns8 j, nate, counter = 0;

    while (Soft_Serial_In == 1)
    {
        counter++;
        if (counter > 245) return('y');
    }

    //Pause for 104us during start bit.
    //Then pause for ~52us to read middle of first data bit
    delay_us(30);

    for(j = 0 ; j < 8 ; j++)
    {
        nate = rr(nate);
        nate.7 = Soft_Serial_In; //Sample the port, should be in middle of the bit
        delay_us(21);
    }

    return(nate);

}
//============================================================================
    #endif
#endif
//End 4MHz and 9600bps


//Functions for 8MHz at 57600bps
#ifdef Clock_8MHz
    #ifdef Baud_57600
//============================================================================
void soft_TTL_out(uns8 nate)
{

    uns8 l, i;

    Soft_Serial_Out = 0;
    nop();
    nop();
    //nop();//
    for(i = 0 ; i < 3 ; i++); //Pause for ~17.36us

    for(l = 0 ; l < 8 ; l++)
    {
        Soft_Serial_Out = nate.0;
        nate = rr(nate);
        nop();
        nop();
        nop();
        //nop();//
        //nop();//
        for(i = 0 ; i < 2 ; i++); //Pause for ~17.36us
    }

    Soft_Serial_Out = 1;
    delay_us(100); //Hold stop bit for 100us
    //delay_us(3); //Hold stop bit for 100us


}
//============================================================================
    #endif
#endif
//End 8MHz and 57600bps

//Functions for 8MHz at 115200bps
#ifdef Clock_8MHz
    #ifdef Baud_115200
//============================================================================

//Sends out TTL levels - nate out at 8MHz 115200 Baud
void soft_TTL_out(uns8 nate)
{
    uns8 l, i;
    Serial_Out_BB = 0;
    for(i = 0 ; i < 1 ; i++); //Pause for ~8.68us

    for(l = 0 ; l < 8 ; l++)
    {
        Serial_Out_BB = nate.0;
        nate = rr(nate);
        nop(); nop(); nop(); nop(); nop(); //Checked 11-24-03
        //nop();//added 1-22
    }

    Serial_Out_BB = 1;
    //delay_us(30); //Hold stop bit for 100us

    for(i = 0 ; i < 15 ; i++); //Pause for ~100us
    //delay_us(3); //Hold stop bit for 100us

}

//Reads in TTL level signals @ 8MHz 115200
uns8 soft_TTL_in(void)
{

    uns8 j, i = 0, nate;

    //Wait ~500us for start bit, return 0 if none is found
    while (Serial_In_BB == 1)
    {
        i++;
        if (i > 61) return(0);
    }

    //Pause for 8.68us during start bit.
    //Then pause for ~4.34us to read middle of first data bit
    //Total 13.02us
    for(i = 0 ; i < 1 ; i++);
    //nop(); nop(); nop(); nop(); nop(); nop(); nop();

    for(j = 0 ; j < 8 ; j++)
    {
        nate = rr(nate);
        nate.7 = Serial_In_BB; //Sample the port, should be in middle of the bit ~8.68us
        nop(); nop(); nop(); nop(); nop(); nop(); //Checked 11-24-03
    }

    /*The PIC is now sitting in the middle of the stop bit. It has
    ~52us before the next start bit of the next transmission is allowed.
    If you do too much during this time, you may miss the next start bit
    and corrupt the next incoming byte.*/

    return(nate);
}
//===========================================================================
    #endif
#endif
//End 8MHz and 115200bps

//Functions for 20MHz at 9600bps
#ifdef HS_Osc
    #ifdef Baud_9600
//===========================================================================
void soft_TTL_wait(void)
{
    int i;
    for(i = 0 ; i < 56 ; i++); // One of many (53-61) that work fine with new CC5x
}

//Sends out TTL levels - nate out at 9600 Baud
void soft_TTL_out(uns8 nate)
{
    uns8 l;

    Serial_Out_BB = 0;
    soft_TTL_wait();

    for(l = 0 ; l < 8 ; l++)
    {
        Serial_Out_BB = nate.0;
        nate = rr(nate);
        soft_TTL_wait();
    }

    Serial_Out_BB = 1;
    delay_us(100); //Hold stop bit for 100us

}

//Reads in TTL level signals @ 9600
uns8 soft_TTL_in(void)
{

    uns8 j, i, nate, counter = 0;

    //Wait ~500us for start bit, return 0 if none is found
    while (Serial_In_BB == 1)
    {
        counter++;
        if (counter > 245) return('t');
    }

    //Pause for 104us during start bit.
    //Then pause for ~52us to read middle of first data bit
    for(i = 0 ; i < 111 ; i++);

    for(j = 0 ; j < 8 ; j++)
    {
        nate = rr(nate);
        nate.7 = Serial_In_BB; //Sample the port, should be in middle of the bit
        for(i = 0 ; i < 72 ; i++);
    }

    /*The PIC is now sitting in the middle of the stop bit. It has
    ~52us before the next start bit of the next transmission is allowed.
    If you do too much during this time, you may miss the next start bit
    and corrupt the next incoming byte.*/

    return(nate);
}
//=========================================================================
    #endif
#endif
//End 20MHz and 9600bps

//Returns ASCII Decimal and Hex values
uns8 bin2HexTTL(char x)
{
   skip(x);
   #pragma return[16] = "0123456789ABCDEF"
}

//Prints a string including variables
void printf_bb_ttl(const char *nate, int16 my_byte)
{
    //Display byte

    uns8 i, k, m, temp;
    uns8 high_byte = 0, low_byte = 0;
    uns8 y, z;

    uns8 decimal_output[5];

    for(i = 0 ; ; i++)
    {
        k = nate[i];

        if (k == '\0')
            break;

        else if (k == '%') //Print var
        {
            i++;
            k = nate[i];

            if (k == '\0')
                break;
            else if (k == 'b') //Print Binary
            {
                for( m = 0 ; m < 8 ; m++ )
                {
                    if (my_byte.7 == 1) soft_TTL_out('1');
                    if (my_byte.7 == 0) soft_TTL_out('0');

                    if (m == 3) soft_TTL_out(' ');

                    my_byte = my_byte << 1;
                }
            } //End Binary
            else if (k == 'd') //Print Decimal
            {
                for(m = 4 ; my_byte > 0 && m < 255 ; m--)
                {
                    temp = my_byte % (uns16)10;
                    decimal_output[m] = temp;
                    my_byte = my_byte / (uns16)10;
                }

                for(m++ ; m < 5 ; m++)
                    soft_TTL_out(bin2HexTTL(decimal_output[m]));
            }
            else if (k == 'h') //Print Hex
            {
                high_byte.3 = my_byte.7;
                high_byte.2 = my_byte.6;
                high_byte.1 = my_byte.5;
                high_byte.0 = my_byte.4;

                low_byte.3 = my_byte.3;
                low_byte.2 = my_byte.2;
                low_byte.1 = my_byte.1;
                low_byte.0 = my_byte.0;

                soft_TTL_out('0');
                soft_TTL_out('x');

                soft_TTL_out(bin2HexTTL(high_byte));
                soft_TTL_out(bin2HexTTL(low_byte));
            } //End Hex
            else if (k == 'f') //Print Float
            {
                soft_TTL_out('!');
            } //End Float

            else if (k == 'u') //Print Direct Character
            {
                //if(my_byte > 20)
                    soft_TTL_out(my_byte);
            } //End Direct

        } //End Special Chars

        else
            soft_TTL_out(k);
    }
}
```

A.4 Main.c

This is the main control code that operates the power regulation, cell phone communications and data acquisition and processing.

```
/*
    1-23-04
    Copyright Spark Fun Electronics 2003

    Nathan Seidle
    nathan.seidle@colorado.edu

    Remote Weather Station using the PIC 16F877A

    1-23-04 See if we can catch RC time constants to measure an unknown cap
            We can accurately measure ~164pF to 1pF using a multimeter as a control
            and against current known weather conditions.

            Vfin = Vint * exp(-t/RC)
            Vfin = 1V - (Microchip 16F88 ST threshold, datasheet page 173-D040
            Vint = 5.00V - This is actual power supply voltage - charges cap plate to 5V
            R = 983,000 Ohms -> Was 814,000 Ohms
            C = Unknown capacitance on humidity sensor. Should be 160-205pF (HS1100 datasheet page 2)
            t = Returned by function in machine cycles (200ns per machine cycle)

            C = -t * 500E-9 / ( ln(1/5) * 983000 )
            C = t * 3.16434E-13

            Adding the LM335A precision temp sensor. Requires calibration and external 5k resistor.

            Moving to DS18S20 temp sensor. No calibration, better precision, more complex interface.

    1-24-04 Hmm, the 1820 is now working. After 4-5 hours of testing and datasheet reading, the 1-wire
            protocol makes perfect sense, but I'm not sure what the 'trick' was, in my case...
            It works! 70.7F in my room.

            Now polling the 1820 for T conversion completion. Timer1 reports
            ~0x26A1EC instructions = 2,531,820 * .2us = 506.4ms
            pass before the temp conversion is complete.

            Received Data cable for Sprint TouchPoint 1100 cell phone today. On the cell phone, under
            setup: incoming calls is set to 'Data'. When the cell phone is called from another line,
            hyperterminal at 9600bps says 'RING' just like the old modems did! The 'ATA' command for
            Answer initiates the standard noisy modem connect tones. It seems we now have a fully
            fuction modem link over cell technology.

    3-13-04 Migration to 16F88 on WeBBS v1.0 PCB. Using 4051 multiplexer. Bloader seems to work really
            well with the 8mhz internal osc and RS232 shift circuit - amazing!

            Multiplexor - LSB is Select1, MSB is Select3

            The MUX should be left on a channel that does not allow current flow to minimize
            power usage. All power_level channels are a bad idea. >30mA

    4-10-04 All the timing for the 1-wire protocol had to be tweaked by hand to adjust for the
            8MHz interal operation. Timing worked great at 20MHz but at 8MHz, everything had to be
            increased carefully. It now works.

    4-18-04 Trying out the diode drop to 3.3V for the GPRS module interface


*/
#define Baud_9600
#define Clock_8MHz //8MHz internal operations

#include "c:\Pics\c\16F88.h"  // device dependent interrupt definitions

#pragma config |= 0x3F10 //Internal RC Oscillator

#pragma origin 4

//Pin definitions
#pragma bit DQ @ PORTA.2

#define FALSE   0
#define TRUE    1

#define INPUT   1
#define OUTPUT  0

#define Select1                     PORTB.7
#define Select2                     PORTB.6
#define Select3                     PORTB.4

#define Cell_TX                     PORTB.3
#define Cell_RX                     PORTA.3

#define Analog_In                   PORTA.0
#define Peripheral_Power            PORTA.1
#define Charge_Control              PORTA.4
#define Analog_In_Direction         TRISA.0
#define Charge_Control_Direction    TRISA.4


#define Soft_Serial_Out   Analog_In
#define Soft_Serial_In   PORTA.7
#define Serial_Out_BB   Analog_In
#define Serial_In_BB   PORTA.7
//End Pin Definitions

//Function definitions
#include "c:\Pics\code\Delay.c"   // Delays
#include "c:\Pics\code\Stdio.c"   // Basic Serial IO
#include "c:\Pics\code\softttl.c"   // Software based Basic Serial IO
#include "c:\Pics\code\soft232.c"   // Software based Basic Serial IO
//#include "c:\Pics\code\i2c.c"   // I2C routines for the eeprom interface
void boot_up(void);

void check_temperature(void);
void check_humidity(void);
void check_pressure(void);
void check_power(void);
void check_wind(void);
void check_signal(void);

uns16 read_analog(void);

void dazed_confused(void);

void send_1wire_command(uns8 cmd);
uns8 read_1wire_byte(void);
void reset_1wire_bus(void);

void Cell_out(uns8 nate);
uns8 Cell_in(void);
void Cell_power(void);
void printf_Cell(const char *nate, int16 my_byte);

//Global Variables
int16 current_temperature_C;
int16 current_temperature_F;
int16 current_humidity;
int16 current_pressure;
int16 current_battery_level;
int16 current_solar_level;

uns8 current_wind_speed[5];
uns8 current_wind_direction[3];
uns8 current_wind_rose[2];

uns8 current_signal_strength[2];

bit Incoming_Call;

uns8 b;
//End Global Variables

void main()
{
    uns8 choice, nate, i;

    nate = 0;

    boot_up();

    while(1)
    {
        nate++;

        printf("\n\r\n\rWelcome to the Weather Station BBS v1.1\n\r", 0);
        printf("==========================================\n\r", 0);
        printf("Main Menu:\n\r", 0);
        printf(" 1) Start Monitor\n\r", 0);
        printf(" 2) Current Status\n\r", 0);
        printf(" 6) Power down peripherals\n\r", 0);
        printf(" 7) Power up peripherals\n\r", 0);
        printf(" a) Sleep for 5 sec\n\r", 0);
        //printf(" d) Power Cycle Cell Phone\n\r", 0);
        printf(" Nate=%h\n\r", nate);
        printf("\n\r : ", 0);

        //Set the monitor to start on boot
        if(nate != 1)
            choice = getc();
        else
            choice = '1';

        if (choice == '1')
        {
            printf("Starting Monitor\n\r", 0);

            while(1)
                dazed_confused();
        }
```

```
else if (choice == '2')
{
    printf("Current status...\n\r", 0);

    check_humidity(); //Read current humidity
    printf("Humidity = %d", current_humidity);
    putc('%');
    printf("\n\r", 0);

    check_pressure(); //Read current barometric pressure
    printf("Barometric Pressure in mili bars = %d\n\r", current_pressure);

    check_temperature(); //Read one-wire bus to DS18S20 sensors
    printf("Temp in C = %d\n\r",current_temperature_C);

    check_wind();//Check wind

    check_power(); //Output current power levels
    printf("Battery Level = %dV\n\r", current_battery_level);
    printf("Solar Level = %dV\n\r", current_solar_level);

    check_signal();
    printf("Signal strength:", 0);
    putc(current_signal_strength[0]);
    putc(current_signal_strength[1]);

    printf("\n\rWind speed is: ", 0);
    for(i = 0 ; i < 5 ; i++)
        putc(current_wind_speed[i]);

    printf("\n\rWind direction is: ", 0);
    for(i = 0 ; i < 3 ; i++)
        putc(current_wind_direction[i]);

    printf("\n\rWind rose is: ", 0);
    for(i = 0 ; i < 2 ; i++)
        putc(current_wind_rose[i]);

    printf("\n\rPress return...", 0);
    while(getc() != '\r');

}
else if (choice == '6')
{
    printf("P down periphs\n\r", 0);

    //Change the Mux connect back to an input
    Analog_In_Direction = 1; //0 = Output

    //Select pressure sensor on the multiplexer
    Select3 = 0; Select2 = 0; Select1 = 0;

    OPTION.7 = 1; //Disable weak pull-ups

    Peripheral_Power = 0;
}
else if (choice == '7')
{
    printf("P up periphs\n\r", 0);

    //Change the Mux connect back to an input
    Analog_In_Direction = 1; //0 = Output

    //Select pressure sensor on the multiplexer
    Select3 = 1; Select2 = 0; Select1 = 1;

    OPTION.7 = 0; //Enable weak pull-ups

    Peripheral_Power = 1;
}
else if (choice == 'a')
{
    printf("Sleeping for 5 seconds\n\r", 0);
    //Tune the internal oscillator
    OSCCON = 0b.0000.0000; //Setup internal oscillator for 31.25kHz
    //while(OSCCON.2 == 0); //Wait for frequency to stabilize

    delay_s_lp(5);

    //Tune the internal oscillator
    OSCCON = 0b.0111.0000; //Setup internal oscillator for 8MHz
    while(OSCCON.2 == 0); //Wait for frequency to stabilize

    printf("Awake\n\r", 0);
}
else if (choice == 'd')
{
    printf("Sending hard power signal\n\r", 0);

    Cell_power();
}
else if (choice == 'e')
{
    uns8 i;
```

```
            printf_Cell("AT\n\r", 0);

            for(i = 0 ; i < 20 ; i++)
                putc(Cell_in());
        }

        else
        {
            printf("choice = %h", choice);
        }

    }

    while(1);

}//End Main

//Initializes the various ports and interrupts
void boot_up(void)
{
    //Tune oscillator to try to get better A/D readings
    OSCTUNE = 0b.0000.0000;

    //Tune the internal oscillator
    OSCCON = 0b.0111.0000; //Setup internal oscillator for 8MHz
    while(OSCCON.2 == 0); //Wait for frequency to stabilize

    //Setup Ports
    CMCON = 0b.0000.0111; //Turn off comparator on RA port
    ANSEL = 0b.0000.0000; //Turn Port A to digital IO

    //Pins:
    //RA0 - Analog_In : May be input or output
    //RA1 - Power Control : Always output, start with everything off
    //RA2 - One-Wire Temp : I/O
    //RA3 - Cell_RX : Always input
    //RA4 - Charge Control : Always Output, Start charging on boot!
    //RA5 - VPP NC
    //RA6 - Humidity Capactitor : I/O
    //RA7 - Wind - Receiver : Always input
    PORTA = 0b.1111.1101;
    TRISA = 0b.1110.1101; //0 = Output 1 = Input

    //RB0 - SCL : Always output
    //RB1 - SDA : I/O
    //RB2 - RX : Input but handled with hardware
    //RB3 - Cell_TX : Always output - Held high until transmission begins
    //RB4 - Select 3 : Always output
    //RB5 - TX : Output but handled with hardware
    //RB6 - Select 2 : Always Output
    //RB7 - Select 1 : Always output
    PORTB = 0b.0000.1110;
    TRISB = 0b.0000.0110; //0 = Output 1 = Input

    OPTION.7 = 0; //Enable weak pull-ups

    enable_uart_TX(0);
    enable_uart_RX(0);

    Peripheral_Power = 1;

}

//This function power cycles (either on or off) the cell module
void Cell_power(void)
{
    printf("Sending hard power signal\n\r", 0);

    Peripheral_Power = 1;
    delay_ms(100);

    //Select the cell on/off on the MUX - Channel 7
    Select3 = 1; Select2 = 1; Select1 = 1;

    //Make the Mux connect an output
    Analog_In_Direction = 0; //0 = Output

    //Send a 1 second high pulse
    Analog_In = 1;
    delay_ms(1500);
    Analog_In = 0;
    delay_ms(1500);

    //Return to Channel 0
    Select3 = 0; Select2 = 0; Select1 = 0;

    //Change the Mux connect back to an input
    Analog_In_Direction = 1; //0 = Output
}

//This function checks the signal strength returned by the cell module
void check_signal(void)
{
    uns8 i;
    current_signal_strength[0] = 'X';
    current_signal_strength[1] = 'X';

    printf_Cell("AT+CSQ\n\r", 0);

    //Poll for special returned character
    for(i = 0 ; i < 20 ; i++)
    {
        if (Cell_in() == ':')
        {
            current_signal_strength[0] = Cell_in(); //Catch space
            current_signal_strength[0] = Cell_in(); //Catch signal strength 10s or 1s digit
            current_signal_strength[1] = Cell_in(); //Catch signal strength 1s digit or ','

            if (current_signal_strength[1] == ',') current_signal_strength[1] = ' '; //Erase the comma if it is
there
        }
    }

    //Output to terminal
    //putc(current_signal_strength[0]);
    //putc(current_signal_strength[1]);

}

//Sends out TTL level nate @ 9600 Baud
void Cell_out(uns8 nate)
{

    uns8 l;

    Cell_TX = 0;
    delay_us(20);

    for(l = 0 ; l < 8 ; l++)
    {
        Cell_TX = nate.0;
        nate = rr(nate);
        delay_us(20);
    }

    Cell_TX = 1;
    delay_us(100); //Hold stop bit for 100us
}
//Reads in TTL level signals @ 9600
uns8 Cell_in(void)
{

    uns8 j, nate;
    uns16 counter = 0;

    while (Cell_RX == 1)
    {
        counter++;
        if (counter > 1245) return('z');
    }

    //Pause for 104us during start bit.
    //Then pause for ~52us to read middle of first data bit
    //for(i = 0 ; i < 111 ; i++);
    delay_us(30);

    for(j = 0 ; j < 8 ; j++)
    {
        nate = rr(nate);
        nate.7 = Cell_RX; //Sample the port, should be in middle of the bit
        delay_us(20);
    }

    return(nate);

}

//Checks the current voltage level on the MPX4115A Pressure sensor
void check_pressure(void)
{
    uns16 pressure_value;
    uns24 mb_pressure;

    //Select pressure sensor on the multiplexer
    Select3 = 0; Select2 = 0; Select1 = 0;

    pressure_value = read_analog();

    //Reported value is a number, 0-1023 with 1023 being Vdd.
    //Pressure = p_v / 9.207 + 10.56 - from MPX4115A datasheet
    //Or Pressure * 100 = p_v * 10000 / 9216 + 106
    //mili bars is kpa * 10

    //Update 4-16-04 : Our Vcc is 4.41V after power control transistor
    //Pressure = p_v / 9.207 + 10.56
    //p_v = analog_reading * Vcc / 1023
    //p_v = 574 * 5.1 / 1023
    mb_pressure = pressure_value * (uns24)10000;
    mb_pressure /= 7961;
    mb_pressure += 106;

    current_pressure = mb_pressure;

    //printf("Barometric Pressure in mili bars = %d\n\r", current_pressure);

}

#pragma codepage 1

//Prints a string including variables
void printf_Cell(const char *nate, int16 my_byte)
{
    //Display byte

    uns8 i, k, m, temp;

    uns8 decimal_output[5];

    for(i = 0 ; ; i++)
    {
        k = nate[i];

        if (k == '\0')
            break;

        else if (k == '%') //Print var
        {
            i++;
            k = nate[i];

            if (k == '\0')
                break;
            else if (k == 'u') //Print Direct Character
                Cell_out(my_byte);
            else if (k == 'd') //Print Decimal
            {
                //Print negative sign and take 2's compliment
                if(my_byte < 0)
                {
                    putc('-');
                    my_byte ^= 0xFFFF;
                    my_byte++;
                }

                //Divide number by a series of 10s
                for(m = 4 ; my_byte > 0 && m < 255 ; m--)
                {
                    temp = my_byte % (uns16)10;
                    decimal_output[m] = temp;
                    my_byte = my_byte / (uns16)10;
                }

                for(m++ ; m < 5 ; m++)
                    Cell_out(bin2Hex(decimal_output[m]));

            } //End Decimal

        } //End Special Chars

        else
            Cell_out(k);
    }
}


//This function does the main processing
void dazed_confused(void)
{
    uns8 i, j;
    uns16 seconds;


    //printf("Checking current conditions\n\r", 0);

    //Power up peripherals
    Peripheral_Power = 1;
    OPTION.7 = 0; //Enable weak pull-ups
    delay_ms(500);

    check_humidity(); //Read current humidity
    check_pressure(); //Read current barometric pressure
    check_temperature(); //Read one-wire bus to DS18S20 sensors
    check_wind();//Check wind
    check_power(); //Output current power levels
    check_signal(); //Check cell signal

    //Update the LCD display
    //Select the LCD on the MUX - Channel 6
    Select3 = 1; Select2 = 1; Select1 = 0;

    //Make the Mux connect an output
    Analog_In_Direction = 0; //0 = Output
```

```
delay_ms(100);
soft_TTL_out(' ');
delay_ms(100);
soft_TTL_out(' ');
delay_ms(100);
soft_TTL_out(124);
soft_TTL_out(8); //Re-init LCD
delay_ms(100);
soft_TTL_out(254);
soft_TTL_out(1); //Clear screen
delay_ms(4000);

for(seconds = 10 ; seconds > 0 ; )
{

    for(j = 0 ; j < 5 ; j++)
    {
        soft_TTL_out(254);
        soft_TTL_out(1); //Clear screen
        delay_ms(1);

        printf_bb_ttl("T:%d ", current_temperature_C);
        printf_bb_ttl("H:%d", current_humidity);
        soft_TTL_out('%');
        printf_bb_ttl(" P:%d", current_pressure);
        printf_bb_ttl("B:%dmV ", current_battery_level);

        //Pause for 5 seconds
        printf_bb_ttl("     %d", --seconds);
        delay_ms(1000);
    }

    for(j = 0 ; j < 5 ; j++)
    {
        //Erase screen
        soft_TTL_out(254);
        soft_TTL_out(1); //Clear screen
        delay_ms(1);

        printf_bb_ttl("WS:", 0);
        for(i = 0 ; i < 5 ; i++)
            soft_TTL_out(current_wind_speed[i]);

        printf_bb_ttl(" WD:", 0);
        for(i = 0 ; i < 3 ; i++)
            soft_TTL_out(current_wind_direction[i]);

        //printf_bb_ttl(" WR:", 0);
        //for(i = 0 ; i < 2 ; i++)
        //    soft_TTL_out(current_wind_rose[i]);

        check_signal(); //Check cell signal

        printf_bb_ttl(" CSS:", 0);
        for(i = 0 ; i < 2 ; i++)
            soft_TTL_out(current_signal_strength[i]);

        printf_bb_ttl("       %d", --seconds);
        delay_ms(1000);
    }
}

//Power on cell module
soft_TTL_out(254);
soft_TTL_out(1); //Clear screen
delay_ms(10);
printf_bb_ttl("Powering cell   module         ", 0);

while(Cell_RX == 0)
{
    Cell_power(); //Power up cell module
    delay_ms(5000);
}

soft_TTL_out(254);
soft_TTL_out(1); //Clear screen
delay_ms(1);
printf_bb_ttl("Cell is ON", 0);
delay_ms(5000);

//Now poll until signal is available
for(j = 0 ; j < 15 ; j++)
{
    check_signal();

    soft_TTL_out(254);
    soft_TTL_out(1); //Clear screen
    delay_ms(1);
    printf_bb_ttl("Signal Level: %u", current_signal_strength[0]);
    printf_bb_ttl("%u", current_signal_strength[1]);
    printf("Waiting for signal\n\r", 0);

    delay_ms(5000);
```

```
        if(current_signal_strength[1] == '9' || current_signal_strength[1] == 'X')
        {
            //if(current_signal_strength[0] != '9') break;
            //No signal, keep going
        }
        else
            break;
    }

    if(j == 15)
    {
        soft_TTL_out(254);
        soft_TTL_out(1); //Clear screen
        delay_ms(1);
        printf_bb_ttl("Call failed", 0);
        delay_ms(3000);
        //return;
    }

    soft_TTL_out(254);
    soft_TTL_out(1); //Clear screen
    delay_ms(1);
    printf_bb_ttl("Placing call to base", 0);
    delay_ms(2000);

    //Call josh's house

//    printf_Cell("ATD 3034445579\n\r", 0);

    delay_ms(50000); //20 Seconds for connection
/*
    printf_Cell("weather ============================\n\r", 0);
    printf_Cell("weather Temperature = %dC\n\r", current_temperature_C);
    printf_Cell("weather Humidity = %d\n\r", current_humidity);
    printf_Cell("weather Pressure = %dmiliBar\n\r", current_pressure);

    printf_Cell("weather Wind Speed = ", 0);
    for(i = 0 ; i < 5 ; i++)
        Cell_out(current_wind_speed[i]);
    printf_Cell("\n\r", 0);

    printf_Cell("weather Wind Direction = ", 0);
    for(i = 0 ; i < 3 ; i++)
        Cell_out(current_wind_direction[i]);
    printf_Cell("\n\r", 0);

    printf_Cell("weather Signal Strength = %d Bars\n\r", current_signal_strength);
    printf_Cell("weather Battery Voltage = %dmV\n\r", current_battery_level);
    printf_Cell("weather Solar Voltage = %dmV\n\r", current_solar_level);
    printf_Cell("weather ============================\n\r", 0);
*/

    printf("weather ============================\n\r", 0);
    printf("weather Temperature = %dC\n\r", current_temperature_C);
    printf("weather Humidity = %d\n\r", current_humidity);
    printf("weather Pressure = %dmiliBar\n\r", current_pressure);

    printf("weather Wind Speed = ", 0);
    for(i = 0 ; i < 5 ; i++)
        putc(current_wind_speed[i]);
    printf("\n\r", 0);

    printf("weather Wind Direction = ", 0);
    for(i = 0 ; i < 3 ; i++)
        putc(current_wind_direction[i]);
    printf("\n\r", 0);

    printf("weather Signal Strength = %d Bars\n\r", current_signal_strength);
    printf("weather Battery Voltage = %dmV\n\r", current_battery_level);
    printf("weather Solar Voltage = %dmV\n\r", current_solar_level);
    printf("weather ============================\n\r", 0);
/*

    printf("weather ============================\n\r", 0);
    printf("weather Temperature = %dC\n\r", current_temperature_C);
    printf("weather Humidity = %d\n\r", current_humidity);
    printf("weather Pressure = %dmiliBar\n\r", current_pressure);
    printf("weather Battery Voltage = %dV\n\r", current_battery_level);
    printf("weather ============================\n\r", 0);

    delay_ms(3000);
*/

    //Hangup phone by turning off cell module
    soft_TTL_out(254);
    soft_TTL_out(1); //Clear screen
    delay_ms(1);
    printf_bb_ttl("Powering down   cell", 0);
    printf("Powering down cell\n\r", 0);
    while(Cell_RX == 1)
    {
        Cell_power();
        delay_ms(10000);
    }
    soft_TTL_out(254);
    soft_TTL_out(1); //Clear screen
    delay_ms(1);
    printf_bb_ttl("Update complete! Going to low power mode", 0);
    delay_ms(2000);

    //Change the Mux connect back to an input
    Analog_In_Direction = 1; //0 = Output

    //Select pressure sensor on the multiplexer
    Select3 = 0; Select2 = 0; Select1 = 0;

    OPTION.7 = 1; //Disable weak pull-ups

    Peripheral_Power = 0;

    printf("Low power mode\n\r", 0);
    //Tune the internal oscillator
    OSCCON = 0b.0000.0000; //Setup internal oscillator for 31.25kHz
    //while(OSCCON.2 == 0); //Wait for frequency to stabilize

    delay_s_lp(350);

    //Tune the internal oscillator
    OSCCON = 0b.0111.0000; //Setup internal oscillator for 8MHz
    while(OSCCON.2 == 0); //Wait for frequency to stabilize

    printf("8MHz mode\n\r", 0);

}


//Powers the wind sensor and listens for the incoming wind speed
//and direction
//looks like:
//$$$SSS.SS00DDDaa$$
//Wind sensor TX in located on RA7
void check_wind(void)
{
    uns8 i;

    current_wind_speed[0] = '0';
    current_wind_speed[1] = '0';
    current_wind_speed[2] = '0';
    current_wind_speed[3] = '0';
    current_wind_speed[4] = '0';
    current_wind_direction[0] = '0';
    current_wind_direction[1] = '0'; //10s digit of direction angle
    current_wind_direction[2] = '0'; //1s digit of direction angle
    current_wind_rose[0] = '0'; //Rose direction 'N' or '-', etc
    current_wind_rose[1] = '0'; //Rose direction 'S' or 'E', etc

    //Wait for three incoming $'s
    for(i = 0 ; i < 200 ; i++)
        if(soft_TTL_in() == '$') break;
    for(i = 0 ; i < 200 ; i++)
        if(soft_TTL_in() == '$') break;
    for(i = 0 ; i < 200 ; i++)
        if(soft_TTL_in() == '$') break;

    //Begin recording
    current_wind_speed[0] = soft_TTL_in(); //This is fake, first number is always 0
    current_wind_speed[0] = soft_TTL_in(); //10s digit
    current_wind_speed[1] = soft_TTL_in(); //1s digit
    current_wind_speed[2] = soft_TTL_in(); //Should be '.'
    current_wind_speed[3] = soft_TTL_in(); //.1's digit
    current_wind_speed[4] = soft_TTL_in(); //.01's digit

    current_wind_direction[0] = soft_TTL_in(); //Filler 0
    current_wind_direction[0] = soft_TTL_in(); //Filler 0
    current_wind_direction[0] = soft_TTL_in(); //100s digit of direction angle
    current_wind_direction[1] = soft_TTL_in(); //10s digit of direction angle
    current_wind_direction[2] = soft_TTL_in(); //1s digit of direction angle

    current_wind_rose[0] = soft_TTL_in(); //Rose direction 'N' or '-', etc
    current_wind_rose[1] = soft_TTL_in(); //Rose direction 'S' or 'E', etc

}

//Charges the variable capacitor humidity sensor
//Returns machine cycles, 200us each.
void check_humidity(void)
{
    uns24 cap_size;

    //Setup port to charge cap
    PORTA.6 = 1;
    TRISA.6 = 0; //0 = Output, 1 = Input

    delay_ms(3); //Charge cap

    TMR1H = 0; TMR1L = 0; //Clear timer1

    TRISA.6 = 1; //0 = Output, 1 = Input
    TMR1ON = 1; //Start timing
    while(PORTA.6 == 1);
    TMR1ON = 0; //Stop timing

    cap_size.high8 = 0;
    cap_size.mid8 = TMR1H;
    cap_size.low8 = TMR1L;

    //printf("Humidity Timer = %d\n\r", cap_size);

    //The capsize was measured and calibrated against the timer value
    cap_size *= (uns24)316;
    cap_size /= 1000;
    cap_size += 50; //Adjust for 50pF of pin capacitance

    //printf("Humidity Sensor = %dpF\n\r", cap_size);

    current_humidity = 5;
    if(cap_size > 166) current_humidity = 10;
    if(cap_size > 168) current_humidity = 15;
    if(cap_size > 170) current_humidity = 20;
    if(cap_size > 172) current_humidity = 25;
    if(cap_size > 173) current_humidity = 30;
    if(cap_size > 174) current_humidity = 35;
    if(cap_size > 176) current_humidity = 40;
    if(cap_size > 178) current_humidity = 45;
    if(cap_size > 179) current_humidity = 50;
    if(cap_size > 182) current_humidity = 55;
    if(cap_size > 183) current_humidity = 60;
    if(cap_size > 184) current_humidity = 65;
    if(cap_size > 186) current_humidity = 70;
    if(cap_size > 188) current_humidity = 75;
    if(cap_size > 191) current_humidity = 80;
    if(cap_size > 193) current_humidity = 85;
    if(cap_size > 195) current_humidity = 90;
    if(cap_size > 198) current_humidity = 95;
    if(cap_size > 200) current_humidity = 100;

    //printf("Humidity = %d", current_humidity);
    //putc('%');
    //printf("\n\r", 0);
}

//Checks the current voltage on the 2.5V Reference
void check_power(void)
{

    uns16 power_status[5];
    uns8 i;
    uns24 power_level;
    uns16 vref;

    //The MUX is power controlled - checking anything attached, we must power MUX
    //Peripheral_Power = 1;
    //delay_ms(100);

    //Select 001 for 2.5V Reference
    Select3 = 0; Select2 = 0; Select1 = 1;
    power_status[0] = read_analog();

    //Select 010 for Power Level 1 - Solar current 1
    Select3 = 0; Select2 = 1; Select1 = 0;
    power_status[1] = read_analog();

    //Select 011 for Power Level 2 - Battery Level
    Select3 = 0; Select2 = 1; Select1 = 1;
    power_status[2] = read_analog();

    //Select 100 for Power Level 3 - Circuit current usage 1
    Select3 = 1; Select2 = 0; Select1 = 0;
    power_status[3] = read_analog();

    //Select 101 for Power Level 4 - Solar Voltage
    //We have to stop driving Charge_Control if we want to see the voltage!

    //Revision 2 PCB
    Charge_Control_Direction = INPUT;
    Select3 = 1; Select2 = 0; Select1 = 1;
    power_status[4] = read_analog();
    Charge_Control_Direction = OUTPUT;

    vref = power_status[0];

    power_level = power_status[2];
    power_level *= (uns16)250;
    power_level /= (uns16)vref;
    power_level *= (uns16)314;
    current_battery_level = power_level / 10;

    power_level = power_status[4];
    power_level *= (uns16)250;
    power_level /= (uns16)vref;
    power_level *= (uns16)314;
    current_solar_level = power_level / 10;
    printf("\n\r", 0);
    printf("Power Array = Vref       Pre-Batt   Batt Lvl   Circ Draw Solar Lvl", 0);

    printf("\n\r", 0);
    printf("Power Array = ", 0);
    for(i = 0 ; i < 5 ; i++)
    {
        power_level = power_status[i];

        printf("%h-", power_level.mid8);
        printf("%h  ", power_level.low8);
    }

    //Calculate out the various voltages based on the Vref(2.5V) reading
    vref = power_status[0];
    //Voltage = p_status[x] * 250 / vref
    //Example:
    //Voltage = 394 * 250 / 514 = 191.6 = 1.91V

    printf("\n\r", 0);
    printf("Power Array = ", 0);
    for(i = 0 ; i < 5 ; i++)
    {
        power_level = power_status[i];
        power_level *= 250;
        power_level /= vref;
        //print_decimal(power_level, 2);
        printf("V    ", 0);
    }



}

//Communicates with DS18S20 1-wire digital temp sensor
void check_temperature(void)
{
    uns8 i, temp;
    uns8 temperature_array[10];

    reset_1wire_bus(); //Reset bus
    send_1wire_command(0xCC); //Send out Skip ROM command
    send_1wire_command(0x44); //Tell sensor to get temperature
    //This can take as much as 750ms - nominally 506ms
    //Wait for a 1 response
    while(read_1wire_byte() == 0);

    reset_1wire_bus(); //Reset bus
    send_1wire_command(0xCC); //Send out Skip ROM command
    send_1wire_command(0xBE); //Read the scratch pad - ie the temperature bytes

    for(i = 0 ; i < 9 ; i++)
    {
        temp = read_1wire_byte();
        temperature_array[i] = temp;
    }

    current_temperature_C = temperature_array[0];
    current_temperature_C >>= 1;

    //printf("Temp in C = %d\n\r",current_temperature_C);

    //for(i = 0 ; i < 9 ; i++)
    //    printf("%h-",temperature_array[i]);
    //printf("\n\r",0);


}

//Resets all slaves on the 1wire bus
void reset_1wire_bus(void)
{
    //Low for min 480us
    TRISA.2 = 0; //Output a zero reset pulse length
    DQ = 0;
    delay_ms(1);

    //Read in response
    TRISA.2 = 1;
    delay_us(13); //Actually originally delay_us(60);
    while(DQ == 0);

}

//Sends out a command serially across 1 wire bus
void send_1wire_command(uns8 cmd)
{
    //Data is sent out LSB first
    uns8 i;

    for(i = 0 ; i < 8 ; i++)
    {
        TRISA.2 = 0; //Output 0
        DQ = 0;
        delay_us(1); //Actually originally delay_us(5);
        if(cmd.0 == 0) TRISA.2 = 0; //Continue 0 output
        if(cmd.0 == 1) TRISA.2 = 1; //Release and let line rise
        delay_us(13); //Actually originally delay_us(60);

        TRISA.2 = 1; //Release port before next slot
        delay_us(1); //Actually originally delay_us(5);

        cmd = cmd >> 1; //Rotate to next data bit
    }

}

//Serially reads the current byte from 11 wire bus
uns8 read_1wire_byte(void)
{
    //Data is received LSB first
    uns8 i, data_in;

    for(i = 0 ; i < 8 ; i++)
    {
        TRISA.2 = 0; //Output a zero to signal slot
        DQ = 0;
        //delay_us(3); //Hold a minimum of 1us
        nop2(); nop2();

        TRISA.2 = 1; //Release line for DS18S20 take over
        delay_us(1); //Actually originally delay_us(5);

        data_in = data_in >> 1;
        data_in.7 = DQ; //Read the data bus

        delay_us(10);  //Originally delay_us(55); //Wait for minimum of 45us for next slot
    }

    return(data_in);
}

//Reads the Analog level off port_num 0-7 - returns the raw voltage level
uns16 read_analog(void)
{
    uns16 temp;

    //Read Channel port_num - for Analog voltage
    TRISA = TRISA | 0b.0000.0001; //0 = Output, 1 = Input

    //Select analog port # - Only on 16F88
    ANSEL = 0b.0000.0001;

    //ADCON0 = 0b.0100.0001; //Select Fosc/16 for 8mhz, and channel RA0/AN0, A/D on
    //ADCON1 = 0b.1100.0000; //Setup ADC for right justified with all AN ports as Analog ADCS2 = 1
    ADCON0 = 0b.1000.0001; //Select Fosc/16 for 8mhz, and channel RA0/AN0, A/D on
    ADCON1 = 0b.1000.0000; //Setup ADC for right justified with all AN ports as Analog ADCS2 = 1

    delay_ms(5);

    GO = 1; //Convert RA0 to digital

    while(GO == 1);

    temp.high8 = ADRESH;
    temp.low8 = ADRESL;

    return(temp);
}
```

Appendix B: Main Board Schematics and PCB

This appendix has circuit layout information for the main board of the WeBBS weather station.

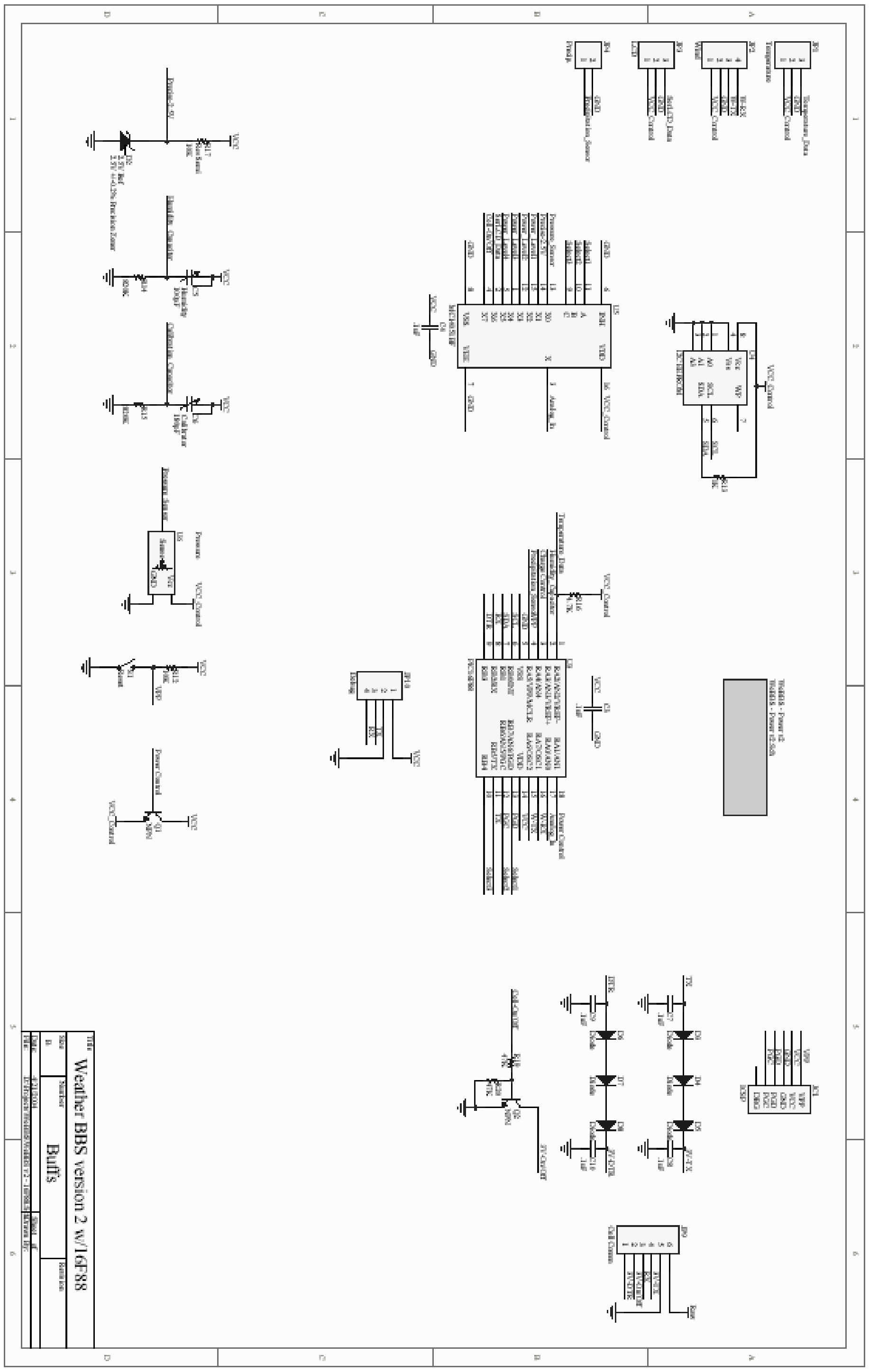


Figure 22: Main Board Schematic

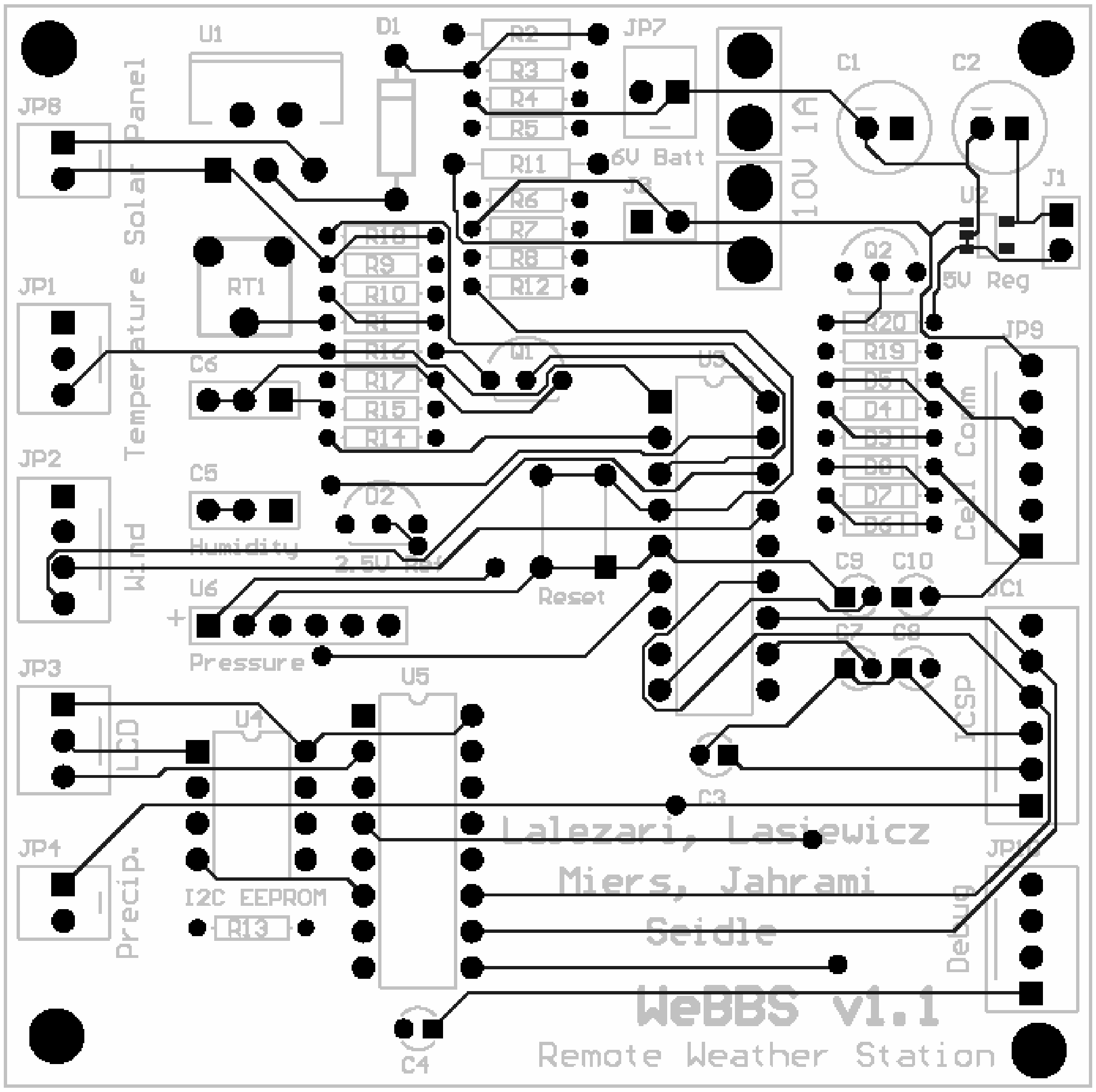


Figure 23: Main Board PCB

Appendix C: Communications Schematics and PCB

This appendix has circuit layout information for the communications board of the WeBBS weather station.

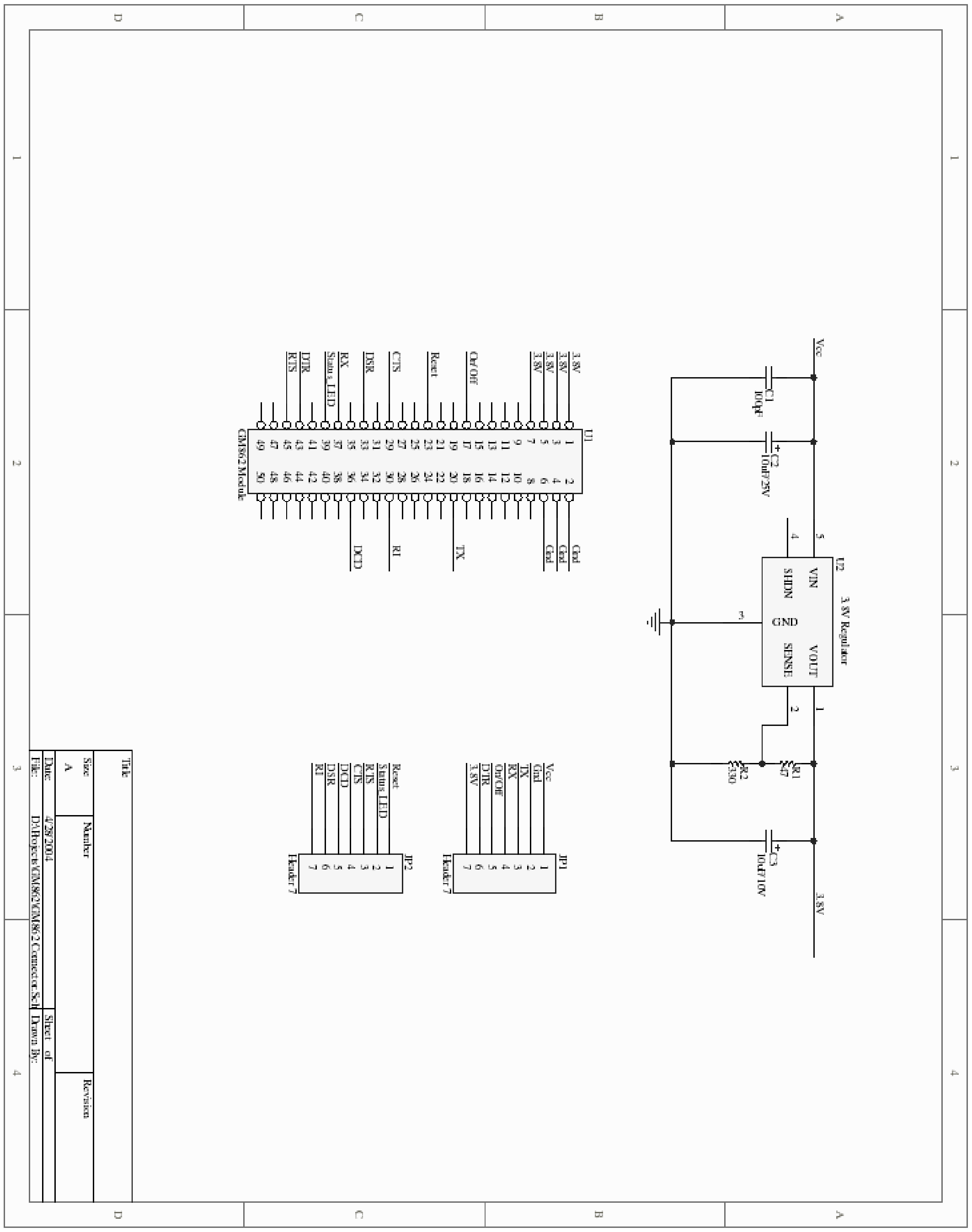


Figure 24: Communications Board Schematic

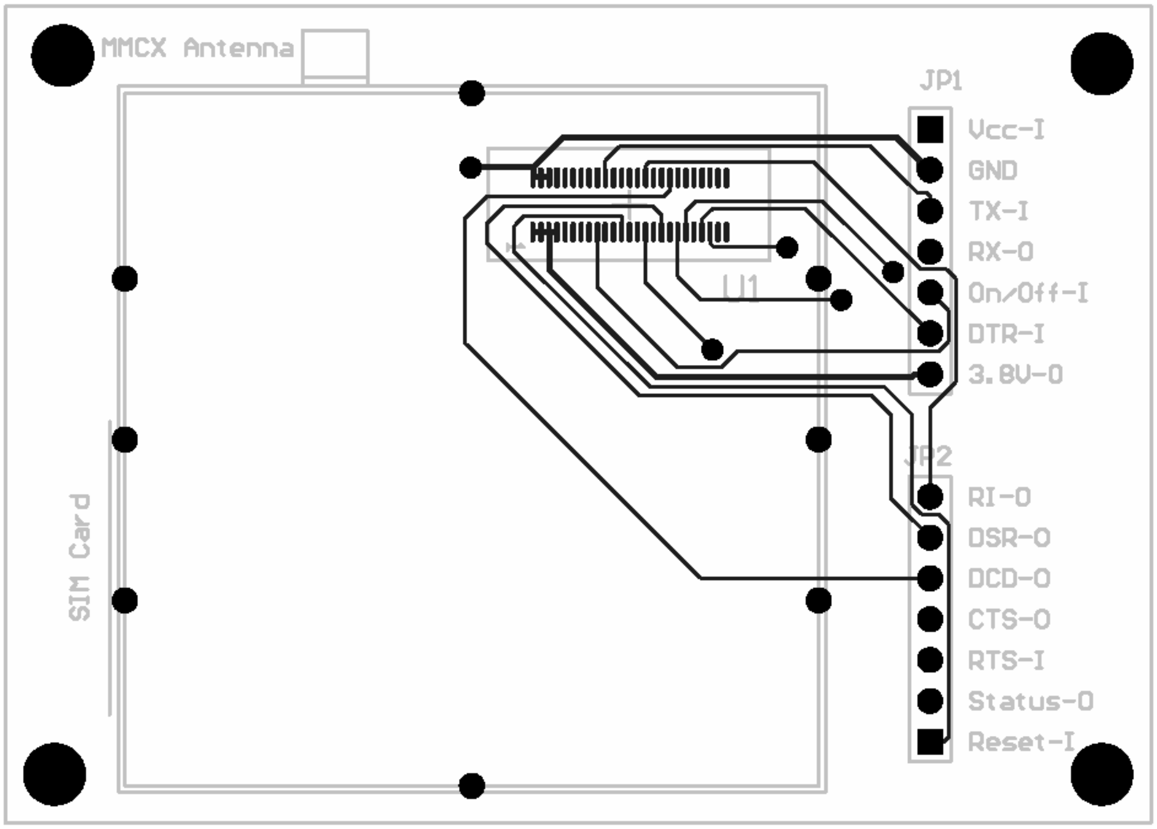


Figure 25: Communications Board PCB

Appendix D: Wind Board Schematics and PCB

This appendix has circuit layout information for the anemometer board of the WeBBS weather station.

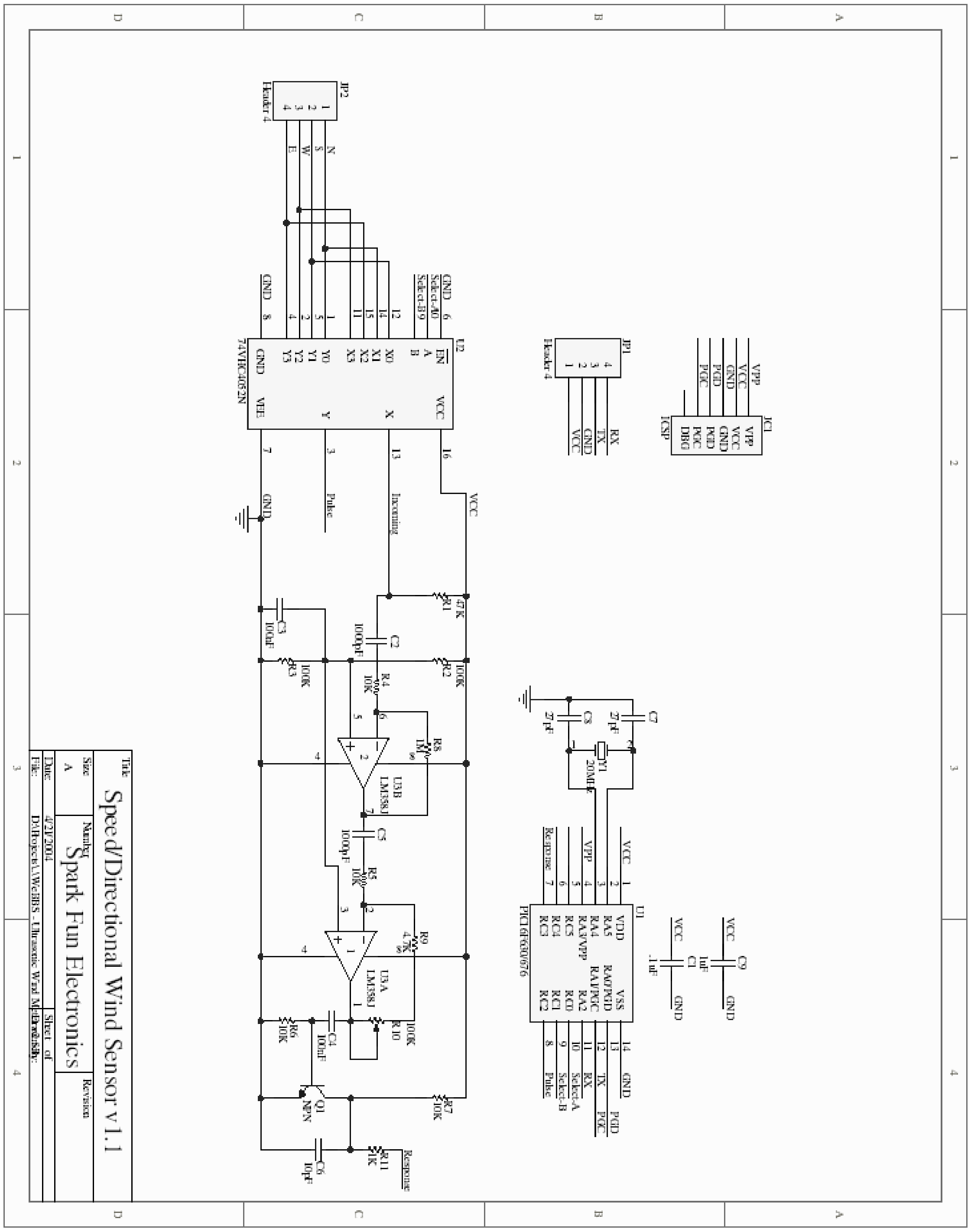


Figure 26: Wind Sensor Schematic

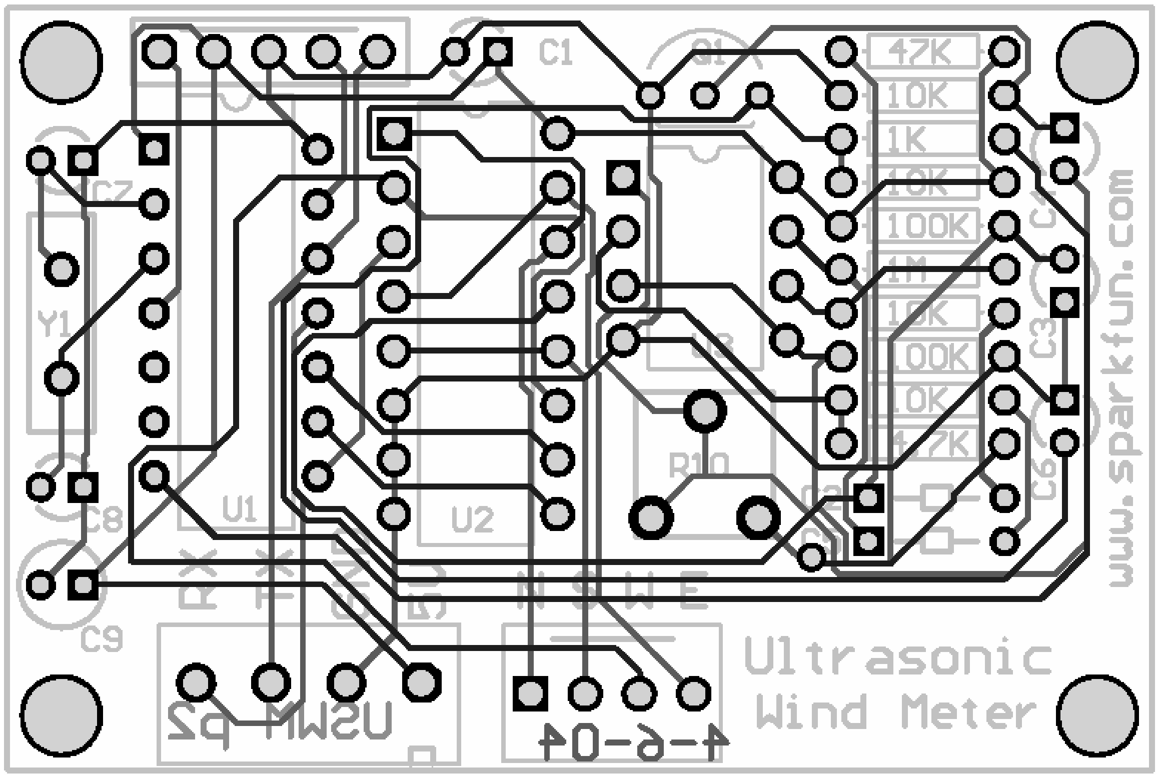


Figure 27: Wind Sensor PCB

Appendix E: Wind Microcontroller Code

This appendix contains annotated code used to process information from the ultrasonic anemometer on the WeBBS weather station.

E.1 Soft232.c

This is written to facilitate bit-banging serial RS232-level serial communications through software. Note that because these are software routines that depend on the current oscillator frequency of the PIC, multiple copies of the same function must be defined.

```
/*
    7-20-03
    Copyright Spark Fun Electronics 2003

    Software driven RS232
    Needed for the 16F872 because it has no onboard UART module

    10-12-03 Overhauled. Everything has been gone over with a fine toothed comb. Timing
             and logic should all check out. Functions preform very well.

    11-6-03 First time at 8MHz with 57600 - no problems!
    11-24-03 115200 @ 8MHz - Works.

*/
//uns8 counter;

//Functions for 20MHz at 9600bps
#ifdef HS_Osc
    #ifdef Baud_9600
//============================================================================
void soft_rs232_wait(void)
{
    int i;
    for(i = 0 ; i < 56 ; i++); // One of many (53-61) that work fine with new CC5x
}

//Sends out RS232 levels - nate out at 9600 Baud
void soft_rs232_out(uns8 nate)
{
    uns8 l;

    Serial_Out_BB = 1;
    soft_rs232_wait();

    for(l = 0 ; l < 8 ; l++)
    {
        Serial_Out_BB = !nate.0;
        nate = rr(nate);
        soft_rs232_wait();
    }

    Serial_Out_BB = 0;
    delay_us(100); //Hold stop bit for 100us

}

//Reads in RS232 level signals @ 9600
uns8 soft_rs232_in(void)
{

    uns8 j, i, nate, counter = 0;

    //Wait ~500us for start bit, return 0 if none is found
    while (Serial_In_BB == 0)
    {
        counter++;
        if (counter > 245) return(0);
    }

    //Pause for 104us during start bit.
    //Then pause for ~52us to read middle of first data bit
    for(i = 0 ; i < 111 ; i++);
    //for(i = 0 ; i < 101 ; i++); //Worked with 16F630 @ 20MHz

    for(j = 0 ; j < 8 ; j++)
    {
        nate = rr(nate);
        nate.7 = !Serial_In_BB; //Sample the port, should be in middle of the bit
        for(i = 0 ; i < 72 ; i++);
    }

    /*The PIC is now sitting in the middle of the stop bit. It has
    ~52us before the next start bit of the next transmission is allowed.
    If you do too much during this time, you may miss the next start bit
    and corrupt the next incoming byte.*/

    return(nate);
}
//=============================================================================
    #endif
#endif
//End 20MHz and 9600bps


//Functions for 4MHz at 9600bps
#ifdef Clock_4MHz
    #ifdef Baud_9600
//=============================================================================
void soft_rs232_wait(void)
{
    int i;
    for(i = 0 ; i < 9 ; i++); //Checked 10-30-03 = ~104us
    nop();
}

//Sends out RS232 levels - nate out at 4MHz 9600 Baud
void soft_rs232_out(uns8 nate)
{
    uns8 l;

    Serial_Out_BB = 1;
    soft_rs232_wait();

    for(l = 0 ; l < 8 ; l++)
    {
        Serial_Out_BB = !nate.0;
        nate = rr(nate);
        soft_rs232_wait();
    }

    Serial_Out_BB = 0;
    delay_us(100); //Hold stop bit for 100us

}

//Reads in RS232 level signals @ 4MHz 9600
//Timing set on 10-31-03
uns8 soft_rs232_in(void)
{

    uns8 j, i = 0, nate;

    //Wait ~500us for start bit, return 0 if none is found
    //while (Serial_In_BB == 0);
    //{
    //    i++;
        //if (i > 61) return('q');
    //    if (i == 254) return('q');
    //}

    //Pause for 104us during start bit.
    //Then pause for ~52us to read middle of first data bit
    for(i = 0 ; i < 20 ; i++);

    for(j = 0 ; j < 8 ; j++)
    {
        nate = rr(nate);
        nate.7 = !Serial_In_BB; //Sample the port, should be in middle of the bit
        for(i = 0 ; i < 13 ; i++);
        //nop(); nop(); nop(); //Clocks out to 104us between each bit
    }

    /*The PIC is now sitting in the middle of the stop bit. It has
    ~52us before the next start bit of the next transmission is allowed.
    If you do too much during this time, you may miss the next start bit
    and corrupt the next incoming byte.*/

    return(nate);
}
//=============================================================================
    #endif
#endif
//End 4MHz and 9600bps


//Functions for 4MHz at 57600bps
#ifdef Clock_4MHz
    #ifdef Baud_57600
//=============================================================================
void soft_rs232_wait(void)
{
    int i;
    for(i = 0 ; i < 3 ; i++); //Checked 11-6-03 = ~17.36us
```

```
}

//Sends out RS232 levels - nate out at 4MHz 57600 Baud
void soft_rs232_out(uns8 nate)
{
    uns8 l, i;

    Serial_Out_BB = 1;
    nop(); nop(); //nop();//
    for(i = 0 ; i < 3 ; i++); //Pause for ~17.36us

    for(l = 0 ; l < 8 ; l++)
    {
        Serial_Out_BB = !nate.0;
        nate = rr(nate);
        nop(); nop(); nop(); //nop();//nop();
        for(i = 0 ; i < 2 ; i++); //Pause for ~17.36us
    }

    Serial_Out_BB = 0;
    delay_us(100); //Hold stop bit for 100us

}

//Reads in RS232 level signals @ 4MHz 57600
uns8 soft_rs232_in(void)
{

    uns8 j, i, nate;
    uns16 counter = 0;

    //Wait ? for start bit, return 0 if none is found
    while (Serial_In_BB == 0);
    //{
    //    counter++;
    //    if (counter > 245) return(0);
    //}

    //Pause for 17.36us during start bit.
    //Then pause for ~8.68us to read middle of first data bit
    //Total 26.08us
    for(i = 0 ; i < 5 ; i++);
    nop(); nop(); nop(); nop(); nop();

    for(j = 0 ; j < 7 ; j++)
    {
        nate = rr(nate);
        nate.7 = !Serial_In_BB; //Sample the port, should be in middle of the bit ~17.5us
        nop(); nop(); nop(); nop();
        for(i = 0 ; i < 2 ; i++); //Pause for ~17.36us
    }

    nate = rr(nate);
    nate.7 = !Serial_In_BB; //Sample the port, should be in middle of the bit ~17.5us

    /*The PIC is now sitting in the middle of the stop bit. It has
    ~52us before the next start bit of the next transmission is allowed.
    If you do too much during this time, you may miss the next start bit
    and corrupt the next incoming byte.*/

    return(nate);
}
//===========================================================================
    #endif
#endif
//End 4MHz and 57600bps

//Functions for 8MHz at 9600bps
#ifdef Clock_8MHz
    #ifdef Baud_9600
//===========================================================================

//Sends out RS232 levels - nate out at 8MHz 9600 Baud
void soft_rs232_out(uns8 nate)
{
    //Fully working 12-26-03
    uns8 l, i;

    Serial_Out_BB = 1;
    for(i = 0 ; i < 27 ; i++); //Pause for ~104us

    for(l = 0 ; l < 8 ; l++)
    {
        Serial_Out_BB = !nate.0;
        nate = rr(nate);
        for(i = 0 ; i < 28 ; i++); //Pause for ~104us
    }

    Serial_Out_BB = 0;
    for(i = 0 ; i < 35 ; i++); //Hold stop bit for ~108us

}

//Reads in RS232 level signals @ 8MHz 9600
uns8 soft_rs232_in(void)
```

```
{

    uns8 j, i, nate;
    //uns16 counter = 0;

    //Wait ? for start bit, return 0 if none is found
    while (Serial_In_BB == 0);
    //{
    //    counter++;
    //    if (counter > 245) return(0);
    //}

    //Pause for 104us during start bit.
    //Then pause for 52us to read middle of first data bit
    //Total 156us
    for(i = 0 ; i < 43 ; i++);

    for(j = 0 ; j < 8 ; j++)
    {
        nate = rr(nate);
        nate.7 = !Serial_In_BB; //Sample the port, should be in middle of the bit ~104us
        for(i = 0 ; i < 28 ; i++); //Pause for ~104us
    }


    return(nate);
}
//===========================================================================
    #endif
#endif
//End 8MHz and 9600bps

//Functions for 8MHz at 19200bps
#ifdef Clock_8MHz
    #ifdef Baud_19200
//===========================================================================

//Sends out RS232 levels - nate out at 8MHz 19200 Baud
void soft_rs232_out(uns8 nate)
{
    //Fully working 12-26-03
    uns8 l, i;

    Serial_Out_BB = 1;
    for(i = 0 ; i < 13 ; i++); //Pause for ~52us

    for(l = 0 ; l < 8 ; l++)
    {
        Serial_Out_BB = !nate.0;
        nate = rr(nate);
        for(i = 0 ; i < 12 ; i++); //Pause for ~52us
    }

    Serial_Out_BB = 0;
    for(i = 0 ; i < 170 ; i++); //Hold stop bit for ~108us

}

//Reads in RS232 level signals @ 8MHz 19200
uns8 soft_rs232_in(void)
{

    uns8 j, i, nate;
    //uns16 counter = 0;

    //Wait ? for start bit, return 0 if none is found
    while (Serial_In_BB == 0);
    //{
    //    counter++;
    //    if (counter > 245) return(0);
    //}

    //Pause for 104us during start bit.
    //Then pause for 52us to read middle of first data bit
    //Total 156us
    for(i = 0 ; i < 43 ; i++);

    for(j = 0 ; j < 8 ; j++)
    {
        nate = rr(nate);
        nate.7 = !Serial_In_BB; //Sample the port, should be in middle of the bit ~104us
        for(i = 0 ; i < 28 ; i++); //Pause for ~104us
    }


    return(nate);
}
//===========================================================================
    #endif
#endif
//End 8MHz and 19200bps

//Functions for 8MHz at 57600bps
#ifdef Clock_8MHz
    #ifdef Baud_57600
```

```
//=========================================================================

//Sends out RS232 levels - nate out at 8MHz 57600 Baud
void soft_rs232_out(uns8 nate)
{
    uns8 l, i;

    Serial_Out_BB = 1;
    nop(); nop(); nop();//
    for(i = 0 ; i < 3 ; i++); //Pause for ~17.36us

    for(l = 0 ; l < 8 ; l++)
    {
        Serial_Out_BB = !nate.0;
        nate = rr(nate);
        nop(); nop(); nop(); nop();//nop();
        for(i = 0 ; i < 2 ; i++); //Pause for ~17.36us
    }

    Serial_Out_BB = 0;
    delay_us(100); //Hold stop bit for 100us

}

//Reads in RS232 level signals @ 8MHz 57600
uns8 soft_rs232_in(void)
{

    uns8 j, i, nate;
    //uns16 counter = 0;

    //Wait ? for start bit, return 0 if none is found
    while (Serial_In_BB == 0);
    //{
    //    counter++;
    //    if (counter > 245) return(0);
    //}

    //Pause for 17.36us during start bit.
    //Then pause for ~8.68us to read middle of first data bit
    //Total 26.08us
    for(i = 0 ; i < 5 ; i++);
    nop(); nop(); nop(); nop(); //nop();

    for(j = 0 ; j < 7 ; j++)
    {
        nate = rr(nate);
        nate.7 = !Serial_In_BB; //Sample the port, should be in middle of the bit ~17.5us
        nop(); nop(); nop(); //nop();
        for(i = 0 ; i < 2 ; i++); //Pause for ~17.36us
    }

    nate = rr(nate);
    nate.7 = !Serial_In_BB; //Sample the port, should be in middle of the bit ~17.5us

    /*The PIC is now sitting in the middle of the stop bit. It has
    ~52us before the next start bit of the next transmission is allowed.
    If you do too much during this time, you may miss the next start bit
    and corrupt the next incoming byte.*/

    return(nate);
}
//=========================================================================
    #endif
#endif
//End 8MHz and 57600bps

//Functions for 8MHz at 115200bps
#ifdef Clock_8MHz
    #ifdef Baud_115200
//=========================================================================
void soft_rs232_wait(void)
{
    int i;
    for(i = 0 ; i < 3 ; i++); //Checked ? = ~8.68us

}

//Sends out RS232 levels - nate out at 8MHz 115200 Baud
void soft_rs232_out(uns8 nate)
{
    uns8 l, i;

    Serial_Out_BB = 1;

    //Original 2-3-04
    //for(i = 0 ; i < 1 ; i++); //Pause for ~8.68us
    //Testing
    nop(); nop(); nop(); nop(); nop();
    nop(); nop(); nop(); nop(); nop();
    nop(); nop(); nop(); nop(); nop();


    for(l = 0 ; l < 8 ; l++)
    {
        Serial_Out_BB = !nate.0;
        nate = rr(nate);
        nop(); nop(); nop(); nop(); nop(); //Checked 11-24-03
        //nop();//added 1-22
    }

    Serial_Out_BB = 0;
    delay_us(30); //Hold stop bit for 100us

    //delay_us(3)1-22; //Hold stop bit for 100us

}

//Reads in RS232 level signals @ 8MHz 115200
uns8 soft_rs232_in(void)
{

    uns8 j, i = 0, nate;

    //Wait ~500us for start bit, return 0 if none is found
    while (Serial_In_BB == 0)
    {
        i++;
        if (i > 61) return(0);
    }

    //Pause for 8.68us during start bit.
    //Then pause for ~4.34us to read middle of first data bit
    //Total 13.02us
    for(i = 0 ; i < 1 ; i++);
    //nop(); nop(); nop(); nop(); nop(); nop(); nop();

    for(j = 0 ; j < 8 ; j++)
    {
        nate = rr(nate);
        nate.7 = !Serial_In_BB; //Sample the port, should be in middle of the bit ~8.68us
        nop(); nop(); nop(); nop(); nop(); nop(); //Checked 11-24-03
    }

    /*The PIC is now sitting in the middle of the stop bit. It has
    ~52us before the next start bit of the next transmission is allowed.
    If you do too much during this time, you may miss the next start bit
    and corrupt the next incoming byte.*/

    return(nate);
}
//============================================================================
    #endif
#endif
//End 8MHz and 115200bps



//Returns ASCII Decimal and Hex values
uns8 bin2Hex_2(char x)
{
   skip(x);
   #pragma return[16] = "0123456789ABCDEF"
}

//Prints a string including variables
void printf_bb(const char *nate, uns8 my_byte)
{
    //Display byte

    uns8 i, k, m;
    uns8 high_byte = 0, low_byte = 0;
    uns8 y, z;

    for(i = 0 ; ; i++)
    {
        k = nate[i];

        if (k == '\0')
            break;

        else if (k == '%') //Print var
        {
            i++;
            k = nate[i];

            if (k == '\0')
                break;
            else if (k == 'b') //Print Binary
            {
                for( m = 0 ; m < 8 ; m++ )
                {
                    if (my_byte.7 == 1) soft_rs232_out('1');
                    if (my_byte.7 == 0) soft_rs232_out('0');

                    if (m == 3) soft_rs232_out(' ');

                    my_byte = my_byte << 1;
                }
            } //End Binary
            else if (k == 'h') //Print Hex
            {
                //New trick 3-15-04
                soft_rs232_out(bin2Hex_2(my_byte >> 4));
                soft_rs232_out(bin2Hex_2(my_byte & 0b.0000.1111));

                /*high_byte.3 = my_byte.7;
                high_byte.2 = my_byte.6;
                high_byte.1 = my_byte.5;
                high_byte.0 = my_byte.4;

                low_byte.3 = my_byte.3;
                low_byte.2 = my_byte.2;
                low_byte.1 = my_byte.1;
                low_byte.0 = my_byte.0;

                soft_rs232_out('0');
                soft_rs232_out('x');

                soft_rs232_out(bin2Hex(high_byte));
                soft_rs232_out(bin2Hex(low_byte));*/
            } //End Hex
            else if (k == 'f') //Print Float
            {
                soft_rs232_out('!');
            } //End Float

            else if (k == 'u') //Print Direct Character
            {
                if(my_byte > 20) soft_rs232_out(my_byte);
            } //End Direct

        } //End Special Chars

        else
            soft_rs232_out(k);
    }
}
```

E.2 UltrasonicWindMeter.c

This code is responsible for the controlling, recording, and processing of the anemometer data.

```
/*
    3-12-04
    Copyright Spark Fun Electronics© 2004

    Ultra Sonic Wind Meter

    3-12-04 New PCBs v1 are in. Let's see if we can get some ultrasonic sensors to talk!

            Select_A = 0; Select_B = 0; //North->South
            Select_A = 1; Select_B = 0; //South->North
            Select_A = 0; Select_B = 1; //West->East
            Select_A = 1; Select_B = 1; //East->West

    3-15-04 Adding high-level math routines


            How to decrypt 24-bit PSEUDO floating point

            0x73 0x2B 0x00 = 0.0032616

            First find 2^(0x73-0x7F) = 2^(115-127) = 0.0024414
            Mkay, now the tough one

            The second two bytes (0x2B and 0x00) correspond to bits
            Bit #   7 6 5 4  3 2 1 0          7 6 5 4 3 2 1 0
            0x2B =  0 0 1 0  1 0 1 1   0x00 = 0 0 0 0 0 0 0 0 (Let's ignore the second byte since its all 0s)

            Now bit 7 is the sign bit, 0 for positive, 1 for negative. Our number is positive (0)

            The rest of the bits start counting in halves:
            Bit 6 = .5
            Bit 5 = .25
            Bit 4 = .125
            Bit 3 = .0625
            Bit 2 = .03125
            Bit 1 = .015625
            Bit 0 = .0078125

            and continue for second byte from bit 7 if needed

            Now take '1' and add to it the partials:
            1 + .25 + .0625 + .015625 + .0078125

            Total = 1.3359375

            Now multiply by the first 2^? number thing:

            1.3359375 * 0.0024414 = 0.0032615578125 ~= .0032616 Done!

    3-29-04 Migrated to the 16F688 for more room. 1K in the 16F630 vs 4K in the 16F688 (plus a UART!).
            No software support for the 688, only by using the MCP under MPLAB can we program the
            688.

*/
#define HS_Osc
#define Baud_9600

/*#include "c:\WeBBS\c\16F688.h"  // device dependent definitions
#include "c:\WeBBS\c\int16CXX.H"*/
#include "c:\Pics\c\16F688.h"  // device dependent interrupt definitions
#include "c:\Pics\c\int16CXX.H"

#pragma config|= 0x3182 //HS Oscilator

uns8 data_byte_in;

//Pin declarations
#define Serial_In_BB    PORTA.2
#define Serial_Out_BB   PORTA.1

#define Pulse_Pin       PORTC.2
//#define Response_Pin    PORTC.3
#define Response_Pin    PORTA.0

#define Select_A        PORTC.0
#define Select_B        PORTC.1

#define Test_Pin        PORTC.4
//Sensor type selection
//#define SensorType_40kHz
#define SensorType_25kHz

#define AVG_AMT     4

#define POSITIVE 1
```

```
#define NEGATIVE 0

#define NORTH 0
#define NORTHEAST 1
#define EAST 2
#define SOUTHEAST 3
#define SOUTH 4
#define SOUTHWEST 5
#define WEST 6
#define NORTHWEST 7

uns16 Temp_Timer;

//Interrupt Vectors
#pragma origin 4
interrupt serverX( void)
{
    int_save_registers
    char sv_FSR = FSR;  // save FSR if required

    if(RAIF) //RA0 Change Interrupt
    {
        Temp_Timer.high8 = TMR1H;
        Temp_Timer.low8 = TMR1L;

        //Turn off any future interrupts
        RAIE = 0;

        //Clear RAIF Flag
        RAIF = 0;
    }

    FSR = sv_FSR;               // restore FSR if saved
    int_restore_registers
}

//Global Variables
bit Direction_N;
bit Direction_E;
uns8 rose;

uns16 Timer_N;
uns16 Timer_S;
uns16 Timer_W;
uns16 Timer_E;

uns24 Timer_NAvg;
uns24 Timer_SAvg;
uns24 Timer_WAvg;
uns24 Timer_EAvg;

float24 Magnitude;
uns16 Wind_Speed;
uns24 Wind_SpeedAvg;
uns16 Wind_Angle;

uns16 Timer_DifferenceNS;
uns16 Timer_DifferenceEW;

uns8 b;
//End Global Variables

/*
#include "c:\WeBBS\code\Delay.c"   // Delays. What do you expect?
#include "c:\WeBBS\c\math24f.h"   // Basic Serial IO
#include "c:\WeBBS\c\math24lb.h"   // Basic Serial IO
#include "c:\WeBBS\code\soft232.c"   // Software based Basic Serial IO
*/
#include "c:\Pics\code\Delay.c"   // Delays
#include "c:\Pics\code\soft232.c"   // Software based Basic Serial IO
#include "c:\Pics\code\softTTL.c"   // Software based Basic Serial IO
#include "c:\Pics\c\math24f.h"   // Basic Serial IO
#include "c:\Pics\c\math24lb.h"   // Basic Serial IO

//Function declarations
void boot_up(void);
uns16 sonicboom(void);
void get_readings(void);
void find_magnitude(void);
void find_readings(void);
void angle_lookup(void);

void print_decimal(uns16 number, uns8 decimal_spot);
//End Function Definitions

void main(void)
{
    //Initialize PIC
    boot_up();

    uns8 i;

    /*
    while(1)
    {
        find_readings();

        printf_bb("N=%h-", Timer_N.high8);
        printf_bb("%h ", Timer_N.low8);

        printf_bb("S=%h-", Timer_S.high8);
        printf_bb("%h ", Timer_S.low8);

        printf_bb("E=%h-", Timer_E.high8);
        printf_bb("%h ", Timer_E.low8);

        printf_bb("W=%h-", Timer_W.high8);
        printf_bb("%h ", Timer_W.low8);

        printf_bb("\n\r", 0);
    }
    */

    while(1)
    {
        //Get current timer values
        //find_readings();


        Timer_NAvg = 0;
        Timer_SAvg = 0;
        Timer_EAvg = 0;
        Timer_WAvg = 0;

        //Get timer values for all four directions
        for(i = 0 ; i < AVG_AMT ; i++)
        {
            find_readings();

            Timer_NAvg += Timer_N;
            Timer_SAvg += Timer_S;
            Timer_EAvg += Timer_E;
            Timer_WAvg += Timer_W;
        }

        Timer_N = Timer_NAvg / AVG_AMT;
        Timer_S = Timer_SAvg / AVG_AMT;
        Timer_E = Timer_EAvg / AVG_AMT;
        Timer_W = Timer_WAvg / AVG_AMT;


        //Find wind speed
        find_magnitude();

        //Find wind direction
        angle_lookup();

        /*
        printf_bb("N=%h-", Timer_N.high8);
        printf_bb("%h ", Timer_N.low8);

        printf_bb("S=%h-", Timer_S.high8);
        printf_bb("%h ", Timer_S.low8);

        printf_bb("E=%h-", Timer_E.high8);
        printf_bb("%h ", Timer_E.low8);

        printf_bb("W=%h-", Timer_W.high8);
        printf_bb("%h ", Timer_W.low8);
        */

        printf_bb_ttl("$$$", 0);
        print_decimal(Wind_Speed, 2);

        print_decimal(Wind_Angle, 6);

        if (rose == NORTH) printf_bb_ttl("-N", 0);
        if (rose == NORTHEAST) printf_bb_ttl("NE", 0);
        if (rose == EAST) printf_bb_ttl("-E", 0);
        if (rose == SOUTHEAST) printf_bb_ttl("SE", 0);
        if (rose == SOUTH) printf_bb_ttl("-S", 0);
        if (rose == SOUTHWEST) printf_bb_ttl("SW", 0);
        if (rose == WEST) printf_bb_ttl("-W", 0);
        if (rose == NORTHWEST) printf_bb_ttl("NW", 0);

        printf_bb_ttl("$$$", 0);

        /*
        printf_bb("dTN=%h-", Timer_DifferenceNS.high8);
        printf_bb("%h ", Timer_DifferenceNS.low8);

        printf_bb("dTW=%h-", Timer_DifferenceEW.high8);
        printf_bb("%h ", Timer_DifferenceEW.low8);
        */

        //if(Direction_N == POSITIVE) printf_bb(" DN=POS ", 0);
        //if(Direction_N == NEGATIVE) printf_bb(" DN=NEG ", 0);
        //if(Direction_E == POSITIVE) printf_bb("DE=POS ", 0);
        //if(Direction_E == NEGATIVE) printf_bb("DE=NEG ", 0);
        printf_bb_ttl("\n\r", 0);

    }


}//End Main

#pragma codepage 1

//Initializes the various ports and interrupts
void boot_up(void)
{
    //Setup Ports
    ANSEL = 0b.0000.0000; //Turn PortA to digital I/O
    CMCON0 = 0b.0000.0111; //Turn off comparator on RA port

    PORTA = 0b.0000.0000;
    TRISA = 0b.0000.0101; //RX is PORTA2

    PORTC = 0b.0000.0000;
    TRISC = 0b.0000.1000;   //0 = Output, 1 = Input

    OPTION.7 = 0; //Enable weak pull-ups

    printf_bb("Ultra Sonic Wind Meter v1.3\n\r\n\r", 0);

    rose = NORTH;

    //Setup interrupts for incoming response pulses
    IOCA.0 = 1;
    RAIE = 1;
    PEIE = 0;
    GIE = 0;

}

void find_readings(void)
{
    uns8 i;

    //G,Or,Br,Bl
    //N,S,W,E
    Select_B = 0; Select_A = 0; //North TX->South RX
    Timer_S = sonicboom();

    Select_B = 0; Select_A = 1; //South TX->North RX
    Timer_N = sonicboom();

    Select_B = 1; Select_A = 0; //
    Timer_E = sonicboom();

    Select_B = 1; Select_A = 1; //
    Timer_W = sonicboom();

    /*Timer_EAvg = 0;
    Timer_WAvg = 0;

    for(i = 0 ; i < AVG_AMT ; i++)
    {
        Select_B = 1; Select_A = 0; //West TX->East RX
        Timer_EAvg += sonicboom();
    }
    Timer_E = Timer_EAvg / AVG_AMT;

    for(i = 0 ; i < AVG_AMT ; i++)
    {
        Select_B = 1; Select_A = 1; //East TX->West RX
        Timer_WAvg += sonicboom();
    }
    Timer_W = Timer_WAvg / AVG_AMT;
    */
}

//Transmits a 40kHz pulse to the transducers.
//Then monitors the time to reach the other side.
uns16 sonicboom(void)
{
    uns8 i;
    uns16 reading;

    //Allow min of 1ms for MUX to switch
    //Allow min of 1ms between readings
    delay_ms(10);

    Temp_Timer = 0;
    Test_Pin = 1;

    TMR1H = 0; TMR1L = 0; //Clear timer value
    TMR1IF = 0; //Reset the timer interrupt flag

    TMR1ON = 1; //Start timer

    //Pulses for 25.2kHz Sensors
    Pulse_Pin = 1; for(i = 0 ; i < 13 ; i++); nop(); //19.8us
    Pulse_Pin = 0; for(i = 0 ; i < 13 ; i++); nop();
    Pulse_Pin = 1; for(i = 0 ; i < 13 ; i++); nop();
    Pulse_Pin = 0; for(i = 0 ; i < 13 ; i++); nop();

    delay_us(100);

    //Enable interrupts for PORTA change
    RAIF = 0;
    RAIE = 1;
    GIE = 1;

    //Wait for response to cause ISR
    while(TMR1IF == 0);

    //while(Response_Pin == 1)
    //    if(TMR1IF == 1) break;

    TMR1ON = 0; //Stop timer
    GIE = 0;

    Test_Pin = 0;

    reading.high8 = TMR1H;
    reading.low8 = TMR1L;

    //return(reading);

    return(Temp_Timer);
}

//Prints a 16 bit decimal
void print_decimal(uns16 number, uns8 decimal_spot)
{
    uns8 ascii_out[5];

    uns16 a = number;
    uns8 i;
    uns8 b;

    b = a / 10000; //10000s
    a = a % 10000;
    ascii_out[0] = b + '0';

    b = a / 1000; //1000s
    a = a % 1000;
    ascii_out[1] = b + '0';

    b = a / 100; //100s
    a = a % 100;
    ascii_out[2] = b + '0';

    b = a / 10; //10s
    a = a % 10;
    ascii_out[3] = b + '0';

    ascii_out[4] = a + '0'; //1s

    //Print the final answer with decimal
    for(i = 0 ; i < 5 ; i++)
    {
        //putc(ascii_out[i]);
        //if(i == decimal_spot) putc('.');

        soft_TTL_out(ascii_out[i]);
        if(i == decimal_spot) soft_TTL_out('.');
    }

}

void find_magnitude(void)
{
    //Begin magnitude calculation
    float24 Wn, We, Temp;

    //This is a constant based on the speed of sound
    //uns24 l = 1034944;
    uns24 l = 3034944;

    //Wn = 0;
    //We = 0;
    //Timer_DifferenceNS = 0;
    //Timer_DifferenceEW = 0

    //Figure out parts and directions
    //========================================
    if (Timer_N == Timer_S)
    {
        Wn = 0; //There is no wind in this direction
        Timer_DifferenceNS = 0;
    }
    else
    {
        if (Timer_N > Timer_S)
        {
            Timer_DifferenceNS = Timer_N - Timer_S;
            Direction_N = POSITIVE;
        }
        else if (Timer_N < Timer_S)
        {
            Timer_DifferenceNS = Timer_S - Timer_N;
            Direction_N = NEGATIVE;
        }

        //Correct for small pertubations of a delta of 10
        if(Timer_DifferenceNS > 200) Timer_DifferenceNS -= 200;

        if(Timer_DifferenceNS < 8) Timer_DifferenceNS = 0;

        //North/South Vector = (N-S)/(N*S) * L
        //L = ?Constant based on speed of sound and??
        Temp = Timer_N;
        Temp = Temp * Timer_S;
        Wn = Timer_DifferenceNS;
        Wn = Wn * l;
        Wn = Wn / Temp;
    }
    //===========================================

    //Same for east west
    if (Timer_E == Timer_W)
    {
        We = 0; //There is no wind in this direction
        Timer_DifferenceEW = 0;
    }
    else
    {
        if (Timer_E > Timer_W)
        {
            Timer_DifferenceEW = Timer_E - Timer_W;
            Direction_E = POSITIVE;
        }
        else if (Timer_E < Timer_W)
        {
            Timer_DifferenceEW = Timer_W - Timer_E;
            Direction_E = NEGATIVE;
        }

        if(Timer_DifferenceEW > 200) Timer_DifferenceEW -= 200;

        if(Timer_DifferenceEW < 8) Timer_DifferenceEW = 0;

        Temp = Timer_E;
        Temp = Temp * Timer_W;
        We = Timer_DifferenceEW;
        We = We * l;
        We = We / Temp;
    }
    //===========================================

    //c = sqrt ( a^2 + b^2 )
    Wn = Wn * Wn;
    We = We * We;
    Wn = Wn + We;
    Magnitude = sqrt(Wn);

    //We now have the wind magnitude, now we convert it to a 16bit int
    Magnitude = Magnitude * 100;
    Wind_Speed = Magnitude;
}

void angle_lookup(void)
{
    //Bottom half and top half are calculated during magnitude function

    uns24 ratio;
    uns24 ratio2;

    //Check for no wind
    if (Timer_DifferenceEW == 0 && Timer_DifferenceNS == 0)
    {
        Wind_Angle = 0;
        return;
    }
    //Check to see if we have ONLY a E/W wind
    if (Timer_DifferenceEW == 0)
    {
        if(Direction_N == POSITIVE) Wind_Angle = 0;
        if(Direction_N == NEGATIVE) Wind_Angle = 180;
        return;
    }
    //Check for ABSOLUTE N/S Wind
    if (Timer_DifferenceNS == 0)
    {
        if(Direction_E == POSITIVE) Wind_Angle = 90;
        if(Direction_E == NEGATIVE) Wind_Angle = 270;
        return;
    }

    ratio = Timer_DifferenceEW * (uns32)1000;
    ratio2 = ratio / Timer_DifferenceNS;
    Wind_Angle = 0;
    if (ratio2 > 66) Wind_Angle = 5;
    if (ratio2 > 154) Wind_Angle = 10;
    if (ratio2 > 245) Wind_Angle = 15;
    if (ratio2 > 339) Wind_Angle = 20;
    if (ratio2 > 440) Wind_Angle = 25;
    if (ratio2 > 549) Wind_Angle = 30;
    if (ratio2 > 668) Wind_Angle = 35;
    if (ratio2 > 803) Wind_Angle = 40;
    if (ratio2 > 957) Wind_Angle = 45;
    if (ratio2 > 1140) Wind_Angle = 50;
    if (ratio2 > 1364) Wind_Angle = 55;
    if (ratio2 > 1648) Wind_Angle = 60;
    if (ratio2 > 2028) Wind_Angle = 65;
    if (ratio2 > 2571) Wind_Angle = 70;
    if (ratio2 > 3431) Wind_Angle = 75;
    if (ratio2 > 5027) Wind_Angle = 80;
    if (ratio2 > 9131) Wind_Angle = 85;
    if (ratio2 > 45829) Wind_Angle = 90;

    //Now adjust angle to appropriate quadrant
    //Quadrand calculation this is to find the absolute direction of the wind
    if(Direction_E == POSITIVE)
    {
        if (Direction_N == POSITIVE) //Quadrant 1
        {
            //Do nothing angle = angle;
        }

        if (Direction_N == NEGATIVE) Wind_Angle = 180 - Wind_Angle; //Quadrant 4
    }
    else if(Direction_E == NEGATIVE)
    {
        if (Direction_N == NEGATIVE) Wind_Angle = 180 + Wind_Angle; //Quadrant 3

        if (Direction_N == POSITIVE) Wind_Angle = 360 - Wind_Angle; //Quadrant 4
    }

    //now we also want to output the direction of the wind in the form of
    //n, nw, ne and so on
    rose = NORTH;

    if(Wind_Angle < (uns16)22 || Wind_Angle > (uns16)326) rose = NORTH;

    if( Wind_Angle > (uns16)22 )
        if( Wind_Angle < (uns16)67 )
            rose = NORTHEAST;

    if( Wind_Angle > (uns16)67 )
        if( Wind_Angle < (uns16)113 )
            rose = EAST;

    if(Wind_Angle > (uns16)113 && Wind_Angle < (uns16)156) rose = SOUTHEAST;
    if(Wind_Angle > (uns16)156 && Wind_Angle < (uns16)206) rose = SOUTH;
    if(Wind_Angle > (uns16)206 && Wind_Angle < (uns16)246) rose = SOUTHWEST;
    if(Wind_Angle > (uns16)246 && Wind_Angle < (uns16)296) rose = WEST;
    if(Wind_Angle > (uns16)296 && Wind_Angle < (uns16)336) rose = NORTHWEST;

    //this makes all the cardinal directions a range of 50 degrees and makes all the ones on
    //the 45's a range of 40 degrees.

}
```

Appendix F: Serial LCD Board Schematics and PCB

This appendix has circuit layout information for the LCD fanny pack board of the WeBBS weather station.

F.1 Serial Backpack LCD

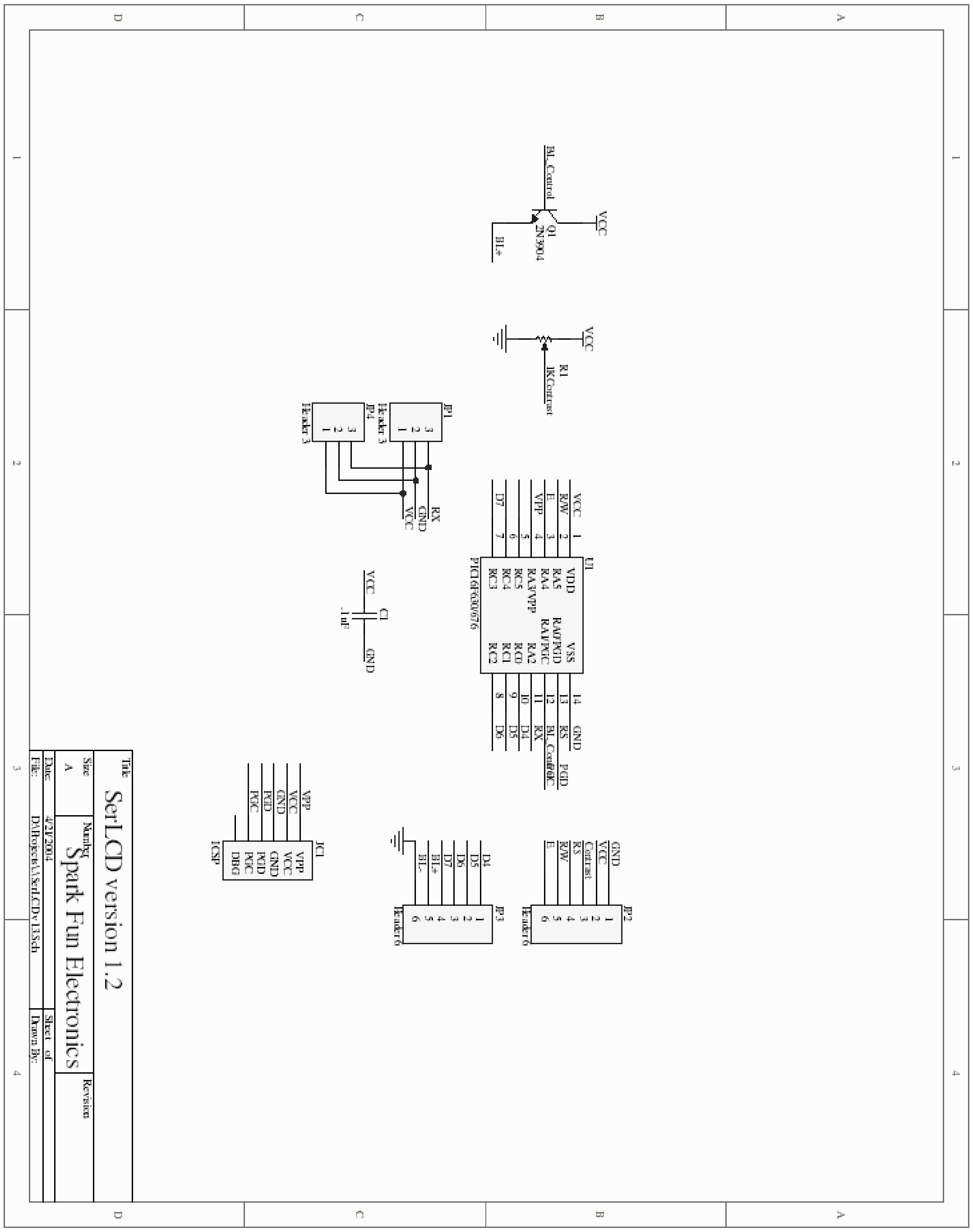


Figure 28: Serial Backpack Schematic

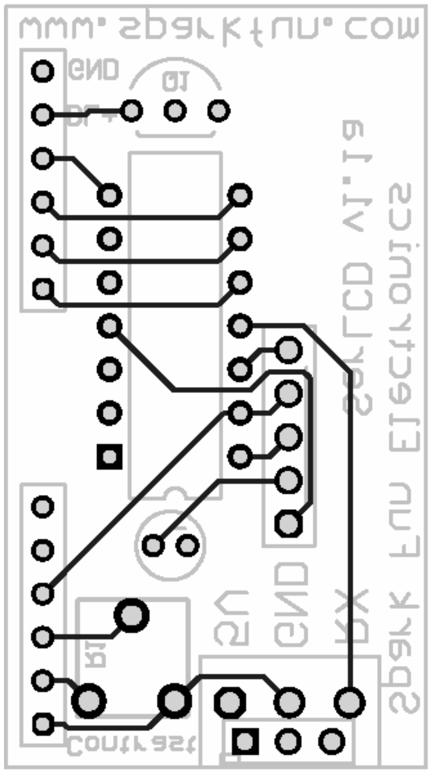


Figure 29: Serial Backpack LCD

F.2 SerLCD.c

The following code interfaces the HD44780 Hitachi parallel chip set to a simple one-wire 9600bps serial interface.

```
/*
    1-16-04
    Copyright Spark Fun Electronics© 2004

    Uses 16F630 to drive Parallel LCD with Serial Commands.

    Thanks to Olimex for providing the original C test code for the PIC-MT-C w/ 16F873

    1-17-04 Started migrating 16F873A code to 16F630

    1-17-04pm Works well from user keyboard input. No busy flag checking yet.
              Ok, busy flag works.

    1-19-04 Got commands to pass through after decimal 254 is seen. The special 8-bit data
            mode command is also correctly ignored.

    1-30-04 v1.1 Added 16x4 line support

    2-10-04 v1.2 Added BL Control. Moved to direct RC0-RC3 = D4-D7 mapping. This significantly
            improved timing between characters.

    2-12-04 Added EEPROM recording of Backlight State. Also added Super Special
            Command input. Pass the '|' character to the LCD, then pass '1' to turn
            on/off backlight.

            Added universal support for any LCD upto 4 lines, 40 characters wide.

    2-16-04 New PCBs v1.1 - Replaced some if statements with EL1, EL2, etc, in the 4 line wrapping
            routine to try to speed things up.

    2-17-04 v1.3 Timing has now been nearly reduced to within one stop bit. In the past,
            wrapping characters were some times lost because of the time required to relocate
            the cursor. Now almost all characters are seen, every time, every location.
            There are stuff certain combinations that cause a character miss.

    3-3-04 New PCB - 1.1a. BL Control has been routed to PGC - bad idea. Causes problems with programming.
           Unit must be powered +5V while programming to get it to work. Will move this line on next revision.
    4-5-04 Customer reported problems with interface. Baud rate was too low. The test bed was running
           at much too low of a speed. The test bed has been corrected, the code must now be tweaked.

    4-14-04 Added a special command 8 that will re-initialize the LCD. There are certain circumstances (low
            power) where the PIC will keep running, but the LCD will shut down.


*/
#define Clock_4MHz
#define Baud_9600

#include "d:\Pics\c\16F630.h"  // device dependent interrupt definitions
//#include "d:\Pics\c\16F676.h"  // device dependent interrupt definitions
#include "d:\Pics\c\int16CXX.H"

#pragma config|= 0x3194 //Internal 4MHz Oscilator

//LCD Type
#define Display_16x2    0
#define Display_16x4    1

//Hardware port definitions
#define     RS              PORTA.0
#define     BL_Control      PORTA.1
#define     Serial_In_BB    PORTA.2
#define     E               PORTA.4
#define     R_W             PORTA.5
#define     Serial_Out_BB   PORTA.1

#define     D4              PORTC.0
#define     D5              PORTC.1
#define     D6              PORTC.2
#define     D7              PORTC.3

//Constant definitions
#define     FALSE       0
#define     TRUE        1

#define     CLR_DISP        0b.0000.0001 //Clear display

#define     CUR_HOME        0b.0000.0010    //Move cursor home and clear screen memory
#define     CUR_RIGHT       0b.0001.0100    //Move cursor one to right
#define     CUR_LEFT        0b.0001.0000    //Move cursor one to left

#define     SCROLL_RIGHT    0b.0001.1100    //Scroll entire screen right one space
#define     SCROLL_LEFT     0b.0001.1000    //Scroll entire screen left one space

#define     DISP_ON         0b.0000.1100    //Turn visible LCD on
#define     DISP_OFF        0b.0000.1000    //Turn visible LCD off

#define     UNDERLINE_ON    0b.0000.1110    //Turn on underlined cursor
#define     UNDERLINE_OFF   0b.0000.1100    //Turn off underlined cursor

#define     BLINKCUR_ON     0b.0000.1101    //Turn on blinking box cursor
#define     BLINKCUR_OFF    0b.0000.1100    //Turn off blinking box cursor

#define     DUALCUR_ON      0b.0000.1111    //Turn on blinking box and underline cursor
#define     DUALCUR_OFF     0b.0000.1100    //Turn off blinking box and underine cursor

#define     SET_CURSOR      0b.1000.0000    //SET_CURSOR + X : Sets cursor position to X

#define     ENTRY_INC       0b.0000.0110 //
#define     DD_RAM_ADDR     0b.1000.0000 //
#define     DD_RAM_ADDR2    0b.1100.0000 //
#define     CG_RAM_ADDR     0b.0100.0000 //

//Global variable declarations
uns8 data_byte_in;
uns8 cursor_position;
uns8 Backlight_Setting;
uns8 LCD_Type;
uns8 LCD_Width;
uns8 LCD_Lines;
uns8 EL1, EL2, EL3, EL4;
uns8 CP1, CP2, CP3, CP4;
//End Global variable declarations

//Interrupt Vectors
#pragma origin 4
interrupt serverX( void)
{
    int_save_registers
    char sv_FSR = FSR;  // save FSR if required

    if(INTF) //RA2 Change Interrupt
    {
        uns8 j, i;

        for(i = 0 ; i < 20 ; i++);

        for(j = 0 ; j < 7 ; j++)
        {
            data_byte_in.7 = Serial_In_BB; //Sample the port, should be in middle of the bit
            data_byte_in = rr(data_byte_in);
            for(i = 0 ; i < 12 ; i++);
            nop(); nop(); nop(); //Clocks out to 104us between each bit
        }

        data_byte_in.7 = Serial_In_BB; //Sample the port, should be in middle of the bit

        //Clear INT Flag
        INTF = 0;
    }

    FSR = sv_FSR;               // restore FSR if saved
    int_restore_registers
}

//Function declarations
void boot_up(void);
void send_char(uns8);
void send_cmd(uns8);
void special_commands(void);
void delay_ms(uns16);
void send_string(const char* incoming_string);
void delay_ms(uns16);
void LCD_wait(void);
uns8 onboard_eeread(uns8 e_address);
void onboard_eewrite(uns8 e_data, uns8 e_address);
void init_lcd(void);

//End Function declarations

void main(void)
{
    //Initialize LCD and PIC
    boot_up();

    while(1)
    {
        //Check to see if we have new data available
        if (data_byte_in == 0)
        {
            sleep(); //Wait for incoming command to wake us up...
            nop(); //Executes after wake-up and before INTF runs
        }

        //Check for special LCD command
        if (data_byte_in == 254)
        {
            //Wait for incoming command to wake us up...
            sleep();
            nop(); //Executes after wake-up and before INTF runs

            //Ignore the one command that will send the LCD into 8-bit mode
            if ( (data_byte_in >> 4) != 3 ) //If not 0b.0000.0011, then send it to LCD
                send_cmd(data_byte_in);

            //Correct cursor position variable if certain commands are received
            if (data_byte_in == CLR_DISP)
                cursor_position = 0;
            else if (data_byte_in.7 == 1) //Move cursor command
                cursor_position = data_byte_in & 0b.0111.1111; //Ignore first bit - obtain address

            data_byte_in = 0;
        }
        //Super Special LCD Commands
        else if (data_byte_in == 124) //This is the '|' character above '\'
        {
            //Wait for incoming command to wake us up...
            sleep();
            nop(); //Executes after wake-up and before INTF runs

            special_commands();

            data_byte_in = 0;
        }
        else
        {
            send_char(data_byte_in);
            data_byte_in = 0;

            cursor_position++;

            //When the cursor gets to the end of one line, it must
            //advance to the next visual line
            //==============================================================
            if (cursor_position == EL1) //End of line one
            {
                if(LCD_Lines == 1)
                {
                    cursor_position = 0; //Return to beginning of line 1
                    send_cmd(CP1);
                }
                else
                {
                    cursor_position = 64; //Beginning of line 2
                    send_cmd(CP2);
                }
            }

            else if (cursor_position == EL2) //End of line 2
            {
                if(LCD_Lines == 2)
                {
                    cursor_position = 0; //Return to line 1
                    send_cmd(CP1);
                }
                else
                {
                    cursor_position = LCD_Width; //Beginning of line 3
                    send_cmd(CP3);
                }
            }

            else if (cursor_position == EL3) //End of line 3
            {
                cursor_position = EL2; //Beginning of line 4 is the end of line 2
                send_cmd(CP4);
            }

            else if (cursor_position == EL4) //End of line 4
            {
                cursor_position = 0; //Beginning of line 1
                send_cmd(CP1);
            }

            //===============================================================

        }//End regular character printing

   }//End infinite loop


}//End Main

//This is called when the control character is received.
//The control character is 124 or '|' (above the '\' on standard US layout keyboards)
void special_commands(void)
{
    //Backlight control
    if (data_byte_in == 1)
    {
        Backlight_Setting = 1; //Turn on backlight setting
        BL_Control = Backlight_Setting; //Turn on backlight
        onboard_eewrite(Backlight_Setting, 0); //Record to spot 0
    }
    else if(data_byte_in == 2)
    {
        Backlight_Setting = 0; //Turn off backlight setting
        BL_Control = Backlight_Setting; //Turn off backlight
        onboard_eewrite(Backlight_Setting, 0); //Record to spot 0
    }

    //LCD type commands
    else if(data_byte_in == 3)
    {
        LCD_Width = 20;
        onboard_eewrite(LCD_Width, 1);
    }
    else if(data_byte_in == 4)
    {
        LCD_Width = 16;
        onboard_eewrite(LCD_Width, 1);
    }
    else if(data_byte_in == 5)
    {
        LCD_Lines = 4;
        onboard_eewrite(LCD_Lines, 2);
    }
    else if(data_byte_in == 6)
    {
        LCD_Lines = 2;
        onboard_eewrite(LCD_Lines, 2);
    }
    else if(data_byte_in == 7)
    {
        LCD_Lines = 1;
        onboard_eewrite(LCD_Lines, 2);
    }
    else if(data_byte_in == 8) //Re-init command
    {
        init_lcd();
    }


    //LCD_Width is the number of seen characters
    //EL1 is End Line 1 - Normally LCD_Width
    //EL2 is End Line 2 - Normally 64 + LCD_Width
    //EL3 is End Line 3 - LCD_Width + LCD_Width
    //EL4 is End Line 4 - 64 + LCD_Width + LCD_Width

    EL1 = LCD_Width;
    EL2 = 64 + LCD_Width;
    EL3 = LCD_Width + LCD_Width;
    EL4 = 64 + LCD_Width + LCD_Width;

    //CP1 = Cursor Position 1 - 0 Always
    //CP1 = CP1 & 0b.1000.000 - this is to set the DRAM command
    //CP2 = 64 Always = 0b.1000.0000 | 64
    //CP3 = LCD_width = 0b.1000.0000 | LCD_Width
    //CP4 = 64 + LCD_Width = 0b.1000.0000 | (64 + LCD_Width)

    CP1 = 0b.1000.0000 | 0;
    CP2 = 0b.1000.0000 | 64;
    CP3 = 0b.1000.0000 | LCD_Width;
    CP4 = 0b.1000.0000 | (LCD_Width + 64);

}

//Initializes the various ports and interrupts
//Also inits the LCD
void boot_up(void)
{
    //Tune the internal oscillator
    //==========================================================
    #define ADDRESS_LAST_CODE_LOCATION 0x3FF

    #asm
     DW /*CALL*/ 0x2000 + ADDRESS_LAST_CODE_LOCATION
    #endasm

    OSCCAL = W;
    //==========================================================

    //Setup Ports
    //ANSEL = 0b.0000.0000; //Disable ADC on all pins - Only on the 16F676
    CMCON = 0b.0000.0111; //Turn off comparator on RA port

    PORTA = 0b.0000.0000;
    TRISA = 0b.0000.0100;

    PORTC = 0b.0000.0000;
    TRISC = 0b.0000.0000;   //0 = Output, 1 = Input

    OPTION.7 = 0; //Enable weak pull-ups

    //Init LCD
    init_lcd();

    //Setup interrupts for incoming RX
    data_byte_in = 0;
    INTF = 0;
    //INTEDG = 1; //Int on rising edge change - Used for direct RS232 interfaces
    INTEDG = 0; //Int on falling edge change - Used for TTL embedded systems
    INTE = 1; //Enable the RA2 Int, for incoming serial commands/characters
    GIE = 1;
}

//Initializes the 4-bit parallel interface to the HD44780
void init_lcd(void)
{

    RS = 0;
    R_W = 0;

    //Tell the LCD we are using 4bit data communication
    delay_ms(750);
    PORTC = 0b.0000.0011;
    E = 1; E = 0;

    delay_ms(100);
    PORTC = 0b.0000.0011;
    E = 1; E = 0;

    delay_ms(100);
    PORTC = 0b.0000.0011;
    E = 1; E = 0;

    delay_ms(10);
    PORTC = 0b.0000.0010;
    E = 1; E = 0;

    send_cmd(DISP_ON);
    send_cmd(CLR_DISP);
    //LCD Init Complete

    send_string("  SparkFun.com");
    send_cmd(SET_CURSOR + 64);
    send_string("  SerLCD v1.1a");

    //Retrieve last Backlight state
    Backlight_Setting = onboard_eeread(0);
    if (Backlight_Setting > 1) Backlight_Setting = 1; //Default
    onboard_eewrite(Backlight_Setting, 0); //Record to spot 0
    BL_Control = Backlight_Setting; //Turn on/off the backlight

    //Retrieve last LCD Type
    LCD_Width = onboard_eeread(1);
    if (LCD_Width > 80) LCD_Width = 16; //Default
    onboard_eewrite(LCD_Width, 1); //Record to spot 1

    //Set end lines
    EL1 = LCD_Width;
    EL2 = 64 + LCD_Width;
    EL3 = LCD_Width + LCD_Width;
    EL4 = 64 + LCD_Width + LCD_Width;

    //Set cursor positions
    CP1 = 0b.1000.0000 | 0;
    CP2 = 0b.1000.0000 | 64;
    CP3 = 0b.1000.0000 | LCD_Width;
    CP4 = 0b.1000.0000 | (LCD_Width + 64);

    LCD_Lines = onboard_eeread(2);
    if (LCD_Lines > 8) LCD_Lines = 2; //Default
    onboard_eewrite(LCD_Lines, 2); //Record to spot 2

    delay_ms(750);
    send_cmd(CLR_DISP);

    cursor_position = 0;
}

//Checks the busy flag and waits until LCD is ready for next command
void LCD_wait(void)
{
    bit i = 1;

    TRISC = 0b.0000.1111;   //0 = Output, 1 = Input

    R_W = 1; //Tell LCD to output status
    RS = 0;

    while(i == 1)
    {
        E = 1;
        i = D7; //Read data bit 7 - Busy Flag
        E = 0;

        E = 1; E = 0; //Toggle E to get the second four bits of the status byte - but we don't care
    }

    TRISC = 0b.0000.0000;   //0 = Output, 1 = Input
}

//Sends an ASCII character to the LCD
void send_char(uns8 c)
{
    LCD_wait();

    R_W = 0; //set LCD to write
    RS = 1; //set LCD to data mode

    PORTC = c >> 4;
    E = 1; E = 0; //Toggle the Enable Pin

    PORTC = c;
    E = 1; E = 0;
}

//Sends an LCD command
void send_cmd(uns8 d)
{
   LCD_wait();

    //TRISC = 0b.0000.0000;   //0 = Output, 1 = Input

    R_W = 0; //set LCD to write

    PORTC = d >> 4;
    E = 1; E = 0;

    PORTC = d;
    E = 1; E = 0;
}

//Sends a given string to the LCD. Will start printing from
//current cursor position.
void send_string(const char *incoming_string)
{
    uns8 x;

    for(x = 0 ; incoming_string[x] != '\0' ; x++)
        send_char(incoming_string[x]);
}

//General short delay
void delay_ms(uns16 x)
{
    //Clocks out at 1002us per 1ms
    int y;
    for ( ; x > 0 ; x--)
        for ( y = 0 ; y < 108 ; y++);
```

```
}

//Reads e_data from the onboard eeprom at e_address
uns8 onboard_eeread(uns8 e_address)
{
    EEADR = e_address; //Set the address to read
    RD = 1; //Read it
    return(EEDAT); //Read that EEPROM value
}

//Write e_data to the onboard eeprom at e_address
void onboard_eewrite(uns8 e_data, uns8 e_address)
{
    bit temp_GIE = GIE;

    EEIF = 0; //Clear the write completion intr flag
    EEADR = e_address; //Set the address
    EEDAT = e_data; //Give it the data
    WREN = 1; //Enable EE Writes
    GIE = 0; //Disable Intrs

    //Specific EEPROM write steps
    EECON2 = 0x55;
    EECON2 = 0xAA;
    WR = 1;
    //Specific EEPROM write steps

    while(EEIF == 0); //Wait for write to complete
    WREN = 0; //Disable EEPROM Writes

    GIE = temp_GIE; //Set GIE to its original state
}
```

Appendix G: Power Platform Schematics

This appendix has circuit schematics for the power platform of the WeBBS weather station.

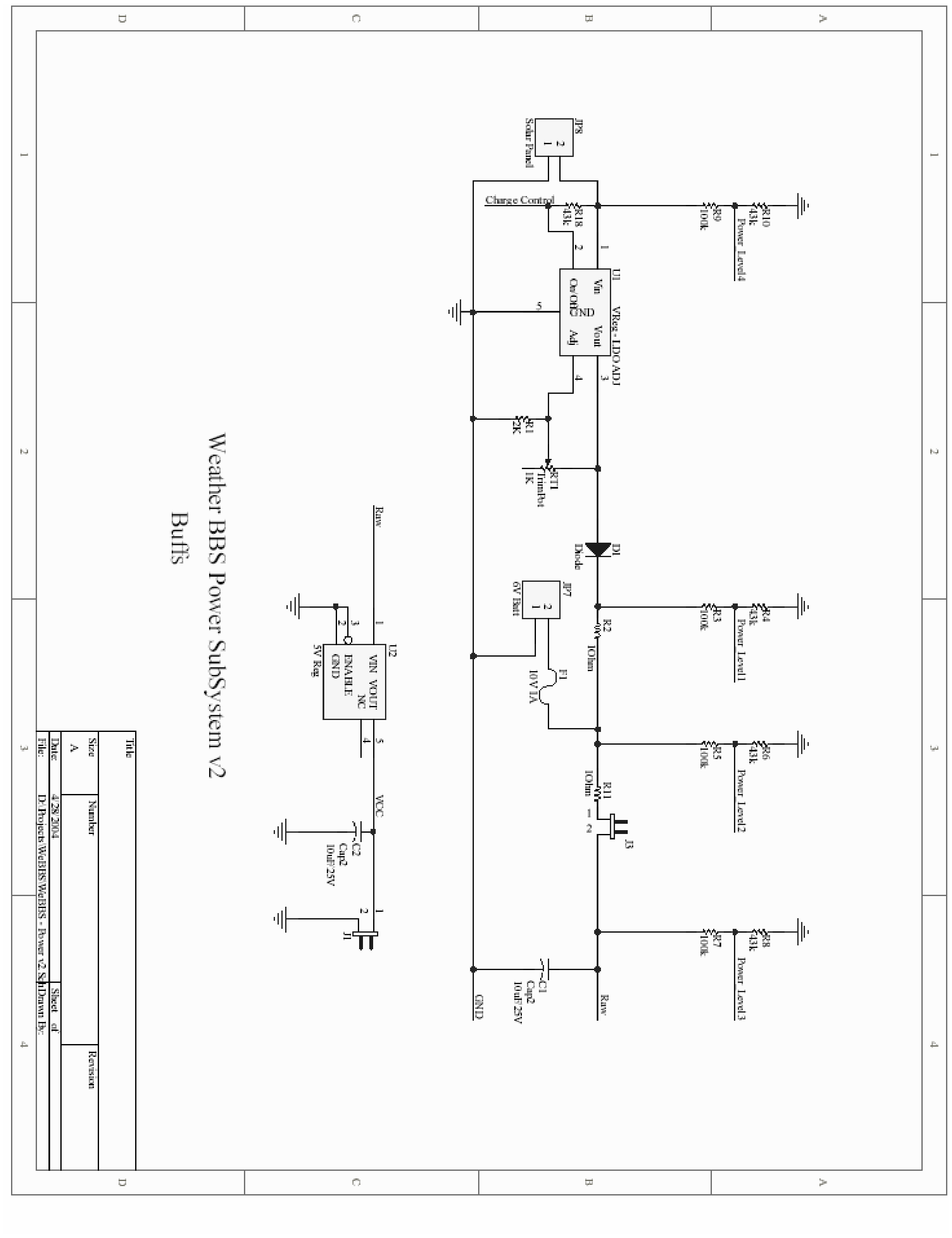


Figure 30: Power Platform Schematic

Appendix H: Website Receiver Station Code

This appendix contains annotated code used at the remote website receiver station.

H.1 Procomm Terminal Script

With Procomm set to automatically answer phone calls, the following script allows us to automatically capture incoming data and save it to a text file.

```
;Continually captures the terminal screen to a file.  v1.00

;*****************************************************************************
; This file will update our webpage.
; This file is meant to capture the entire contents of the terminal window
; and save it into a text file.
;
; script.bat then parses pertinent weather information and saves it to a
; new text file that is posted on a webpage.
;*****************************************************************************
proc main

   string Fname = "file.cap"                            ;File that holds the screen shots.
   string pathname                                      ;Path to the Capture directory.
   clear
   while 1                                              ; Run fo'eva'

   set capture file Fname                               ; Set name of capture file.
   set capture overwrite ON
   fetch capture path pathname                          ; Command to find the current path to the Procomm\Aspect
directory.
     while $CARRIER == 0
        yield                                           ; yield some processor time and wait
        yield
        yield
        yield
        yield
        yield
        yield
        yield
        yield
        yield
        yield                                           ; yield a bit more processor time and wait
        yield
        yield
        yield
        yield
        yield
        yield
        yield
        yield
        yield
        yield                                           ; yield just a bit more processor time and wait
        yield
        yield
        yield
        yield
        yield
        yield
        yield
        yield
        yield                                                                       ; yield one last time
      endwhile
   capture on                                           ; Open up the capture file.
     while $CARRIER                                     ; Loop while connected to remote weather station.
      yield                                           ; Yield processing time
     endwhile
   capture off                                        ; Close the capture file.
   run ("G:\webpage\script.bat")                                ; Run script to capture and store weather data
   clear                                                        ;clears the screen


   endwhile
endproc
```

H.2 Windows Batch File

Each time Procomm records data from an incoming call, the following batch file is executed to call two C++ scripts, and moves the text file with weather data from the Procomm directory to a temporary directory, where the data will be parsed before a corrected file is moved to the web page directory.

```
cls
G:
cd webpage
del inbox.txt
C:
cd \Program Files\Symantec\Procomm Plus\Capture
copy file.cap G:\webpage\file.cap
G:
cd webpage
rename file.cap inbox.txt
firstrun.exe
cd\
cd webpage\secondpart
del weather.txt
cd..
copy weather.txt G:\webpage\secondpart
cd secondpart
secondrun.exe
C:
cd Inetpub\wwwroot
del weather.txt
G:
cd webpage\secondpart
copy stats.txt C:\Inetpub\wwwroot\weather.txt
```

H.3 Firstrun.exe

When called by the batch file, this code takes the text file containing the raw call data received by the Procomm script, and looks for the keywords “weather”, “temp”, “wind”, and “time”. All lines not beginning with one of these code words is ignored. If any line begins with one of these code words, the line is stored into a new weather text file. Finally, this code prepends a time stamp to the weather information for display on the web site.

```
// Zachary Thomas Miers
//
// compression file for capstone 2004
//
// need inbox.txt
//
// This file in a file takes specific lines out of the file
//      and inputs those lines into a new file

#include <winbgim.h>
#include <iostream>
#include <iomanip>
#include <conio.h>
#include <fstream>
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <time.h>

using namespace std;

int main ()//START OF MAIN PROGRAM
{

    delay (500);                    // gives up time to processor -- also waits for file to be written to disk

    ofstream outfile;               // sets outfile
    ifstream infile;                // sets infile

    string weather;                 //decleration of what you want to output in a string

  time_t rawtime;
  struct tm * timeinfo;

  time ( &rawtime );                                   // gets the raw time from system clock
  timeinfo = localtime ( &rawtime );                   // sets raw time to time info
  printf ( "time taken: %s", asctime (timeinfo) );     // displays time to dos window


infile.open ("inbox.txt");          // opens the input file

outfile.open ("weather.txt");       // opens the output file

int count = 0;                      // error checking

cerr << count;                      // error checking
outfile << "Following Weather Conditons Taken:  ";      //displays that line
outfile << asctime (timeinfo) << endl;                          // shows the system clock time after prvous line

while (! infile.eof())              // runs while it is not the end of the file
{

getline (infile,weather);           // gets a entire line at a time

if((weather[0] == 'w' && weather[1] == 'e' && weather[2] == 'a' && weather[3] == 't' && weather[4] == 'h' &&
weather[5] == 'e' && weather[6] == 'r') || (weather[0] == 't' && weather[1] == 'e' && weather[2] == 'm' &&
weather[3] == 'p') || (weather[0] == 'w' && weather[1] == 'i' && weather[2] == 'n' && weather[3] == 'd') ||
(weather[0] == 't' && weather[1] == 'i' && weather[2] == 'm' && weather[3] == 'e'))
{
 outfile << weather << endl;        // the lines that you want to output
}

}

    outfile.close();                //close the output file the one you are writing to
    infile.close();                 //close the input file the one you are writing from

delay (500);                        // gives up time to processor -- also waits for file to be written to disk

return 0;                          // ends the program
}
```

H.4 Secondrun.exe

This code takes the output file from firstrun.exe and removes the codeword weather. After this, it takes the existing weather condition text file, and appends a new reading, before exchanging the old weather log text file with a new version containing the most recent update.

```
// Zachary Thomas Miers
//
// Capstone 2004
//
// clean up weather after file
//
// second file of 2

#include <fstream>
#include <iostream>
#include <cstdlib>
#include <string>

//using std::ifstream;
//using std::ofstream;
//using std::cout;
using namespace std;

void noWeather(ifstream& inStream, ofstream& outStream);

int main()
{
    ifstream fin;
    ofstream fout;

    cout << "Begin editing files.\n";

    fin.open("weather.txt");

    fout.open("stats.txt", ios::app);

    noWeather(fin, fout);

    fin.close();
    fout.close();

    cout << "end of editing files.\n";

    return 0;

}

void noWeather(ifstream& inStream, ofstream& outStream)
{


    string next;

    inStream >> next;
    while (! inStream.eof())
    {


        if ((next == "weather"))
        {
            outStream << " ";
        }
        else
        {
            outStream << next;
        }
   if (inStream.peek()=='\n')
     outStream <<  endl;

   if (inStream.peek()==' ')
     outStream <<  ' ';

    inStream >> next;

    }
                        outStream << " " << endl;
}
```