

Smart Charger

Senior Project Final Report

Design Team #1

Claudiu Bouruc

Eric Boyer

Mike Petrak

Steven Savage

Dr. Tom T. Hartley

May 4, 2005

Table of Contents

Table of Contents.....	i
List of Figures.....	ii
List of Tables.....	iii
List of Equations.....	iv
Table of Listings.....	v
Abstract.....	1
Introduction.....	2
Problem Definition: Goals, Objectives, Constraints.....	2
Design Requirements.....	3
Accepted Technical Design.....	5
DC Power Supply.....	7
Microcontroller Power Supply.....	10
Voltage and Current Sensing.....	11
System Control: Microcontroller-Based Approach.....	12
PWM Control Algorithm.....	14
Microcontroller Hardware Design Interface.....	14
External A/D Converter.....	15
RS-232 Serial Interface.....	16
Pulse Width Modulation.....	17
Liquid Crystal Display Interface.....	17
Precision Temperature Sensing.....	19
Infrared Remote Input.....	19
PC Software Application.....	21
Battery Connector.....	38
Overall Printed Circuit Board Design.....	38
Final Product Design.....	39
Parts List.....	42
Testing Procedures: Validation and Verification.....	43
Financial Budget.....	54
Project Schedule.....	56
Design Team Information.....	61
Conclusions and Recommendations.....	61
References.....	62
Appendix.....	63
Implementation Listings: Microcontroller Code.....	70
Implementation Listings: Software Application Code.....	95

List of Figures

Figure 1 - Smart Charger Overall Block Diagram.	6
Figure 2 - Switch-mode DC Power Supply Overview.	7
Figure 3 - Switch-mode DC Power Supply Circuit Schematic.	7
Figure 4 - Smoothed DC voltage from the rectifier.....	8
Figure 5 - Microcontroller Power Supply Overview.....	10
Figure 6 - 5V Voltage Regulator.....	11
Figure 7 - Voltage Sensing Network Schematic.	11
Figure 8 - Microchip MCP 607 Precision Operational Amplifier.	12
Figure 9 - Current Sensing Network Schematic.	12
Figure 10 - High Level Dataflow Diagram.	13
Figure 11 – System output as a function of the duty cycle..	14
Figure 12 - 40-pin DIL PIC18F452 Package and Schematic.....	15
Figure 13 - AD7705 Interface Schematic.	16
Figure 14 - Maxim RS-232 Serial Interface.	17
Figure 15 - CrystalFontz 40x4 LCD with Backlighting in Operation.	18
Figure 16 - LCD with Hitachi Controller Interface.	18
Figure 17 - LCD Screen During Battery Charging Mode of Operation.	18
Figure 18 - LM34 Precision Temperature Sensor Package.	19
Figure 19 - Sharp GP1UM28YK Operational Block Diagram.	20
Figure 20 - System Use Case Diagram.	22
Figure 21 - Overall Class Diagram.	23
Figure 22 - Charging sequence diagram.	28
Figure 23 - Charging process general state diagram.	30
Figure 24 - Server Application Class Diagram.....	35
Figure 25 - Example Battery Holders.	38
Figure 26 - Finished Product Design.	39
Figure 27 - Finished Product – Front View.	40
Figure 28 - Finished Product – Top View.	40
Figure 29 - Finished Product in Overall System.....	41
Figure 30 - Voltage Output Verification.....	44
Figure 31 - Current Output Verification.....	44
Figure 32 - Sealed lead acid battery.....	47
Figure 33 - Sealed lead acid overall charge plot.....	48
Figure 34 - Voltage and current for sealed lead acid charge.....	49
Figure 35 – 9.6V NiCd Battery.....	49
Figure 36 – NiCd fast charge.....	50
Figure 37 – NiCd Charge voltage and current.....	51
Figure 38 – NiMh AA cells in charging holder.....	51
Figure 39 - Sealed lead acid overall charge plot.....	48
Figure 40 - Implementation Gantt Chart Timeline View.	58
Figure 41 - Actual Gantt Chart Timeline View.	60
Figure A1 - Logic Control Board Schematic with Headers to Accessory Devices.	63
Figure A2 - Preliminary System Control PCB Artwork.	64
Figure A3 - Smart Charger Main Form.....	65

Figure A4 - Smart Charger Charging Form	66
Figure A5 - Smart Charger Help About Form.....	66
Figure A6 - Smart Charger Options Form.....	67
Figure A7 - Smart Charger Server Main Form.....	68
Figure A8 - Smart Charger Server Options Form.....	69

List of Tables

Table 1 - Application-Server data packet structure.	25
Table 2 - PC-Device Communication Packet Protocol.	36
Table 3 - Implementation Parts List.....	42
Table 4 - Efficiency Test.....	53
Table 5 - Labor Cost.	54
Table 6 - Material Cost.	54
Table 7 – Design and Implementation Gantt Chart Tasks and Description.....	56
Table 8 –Actual Gantt Chart Tasks and Description.....	59
Table A1 - Communication Protocol.	65

List of Equations

Equation 1 – DC Voltage.....	8
Equation 2 – Ripple Factor for a Full-Wave Rectifier.....	8
Equation 3 – Smoothing Capacitor Value.....	8
Equation 4 – Duty Ratio or Duty Cycle.....	9
Equation 5 – Minimum Inductance Value for Switch-Mode Power Supply.....	9
Equation 6 – Parameters to Construct the Inductor.....	9
Equation 7 – Minimum Filtering Capacitor Value.....	10
Equation 8 – Voltage Conversion.....	34
Equation 9 – Efficiency.....	53

Table of Listings

Listing 1 - Pseudo code to Communicate with the External ADC.	16
Listing 2 - PWM Signal Pseudo code.....	17
Listing 3 - LCD Pseudo code.	18
Listing 4 - Pseudo code to return Desired Temperature Reading.	19
Listing 5 - Infrared Receiving Routines.	20
Listing 6 - Building a Sample object from packet data.	26
Listing 7 - Pseudo code for data analysis voltage calculation.	27
Listing 8 - Pseudo code for data analysis battery temperature first derivative.....	28
Listing 9 - Lead acid fast charging algorithm.	30
Listing 10 - Nickel Cadmium fast charging algorithm.....	31
Listing 11 - Nickel metal hydride fast charging algorithm.....	32
Listing 12 - Lithium Ion fast charging algorithm.....	34

Microcontroller Code

Listing A1. Main routine in scharger.c.....	70
Listing A2. pwmcontrol.h.....	72
Listing A3. pwmcontrol.c.....	72
Listing A4. pwm.h.....	74
Listing A5. pwm.c.....	74
Listing A6. voltages.h.....	74
Listing A7. voltages.c.....	75
Listing A8. currents.h.....	75
Listing A9. currents.c.....	76
Listing A10. temperatures.h.....	76
Listing A11. temperatures.c.....	77
Listing A12. lcd440s.h.....	78
Listing A13. lcd440s.c.....	79
Listing A14. serial.h.....	84
Listing A15. serial.c.....	84
Listing A16. ad7705spi.h.....	87
Listing A17. ad7705spi.c.....	88
Listing A18. analog.c.....	90
Listing A19. led.c.....	91
Listing A20. freetimer.h.....	91
Listing A21. freetimer.c.....	91
Listing A22. ir.h.....	92
Listing A23. ir.c.....	92

PC Software Application Code

Listing A24. Main Form Code.....	95
Listing A25. Charge Form Code.....	99
Listing A26. Options Form Code.....	102
Listing A27. Form About Code.....	103
Listing A28. Charger Class.....	105

Listing A29. Configuration Class.....	108
Listing A30. Data Analyzer Class.....	113
Listing A31. Event Logger Class.....	117
Listing A32. Graph Class.....	119
Listing A33. IBattery Interface Class.....	121
Listing A34. Lead Acid Interface Class.....	122
Listing A35. Lead Acid Class.....	123
Listing A36. LiIon Class.....	131
Listing A37. ModSmartCharger Standard Module.....	138
Listing A38. NiCad Class.....	138
Listing A39. NiMh Class.....	145
Listing A40. Power Supply Class.....	152
Listing A41. Power Supply Device Driver Class.....	155
Listing A42. Sample Class.....	160
Listing A43. Data Collection Class.....	161
Listing A44. Sealed Lead Acid Class.....	163

Server Application Code

Listing A45. Main Form Code.....	170
Listing A46. Options Form Code.....	172
Listing A47. BytePair Class.....	173
Listing A48. CheckSum Calculator Class.....	174
Listing A49. CommManager Class.....	175
Listing A50. Configuration Class.....	179
Listing A51. ModMain Module.....	184
Listing A52. Power Supply Emulator Class.....	184
Listing A53. SocketManager Class.....	190

Abstract

With the increase of portable electrical devices, the amount of different types of batteries a person owns has greatly increased. Along with different types, the need for longer battery life has also increased. These increases have led to the need for a universal charger. The purpose of this project is to design and build a true universal battery charger. This battery charger will charge various battery chemistries, including lead acid, nickel cadmium, nickel metal hydride, and lithium ion battery types. The charger will charge a variety of different battery sizes, with various amp-hour capacities. With a PC, the project will give the user the control and feedback necessary to completely charge a battery without damaging the battery from overcharging. The project will also act as a variable switching DC power supply using either voltage or current regulation capable of powering a variety of DC loads.

Features of the Smart Charger:

- A universal battery charger that can charge different sizes of batteries
- An intelligent and fast battery charging solution for multiple battery chemistries
- Report all feedback data of the charging process in an easy to interpret fashion
- Easy to use PC software
- Variable DC switching power supply
- Infrared remote control interface to the variable DC switching power supply

Introduction

Statement of Need The future of many portable electrical devices—cellular telephones, notebook computers, personal digital assistants, and still others to be developed—will heavily depend upon the use of advanced battery technology. As battery technology improves, there is a growing need to give the user more control and feedback during the battery charging process while preventing battery damage due to overcharging. Current battery chargers offer little to no control over the charging process. Furthermore, overcharging is a common problem with the simple, inexpensive chargers found in many consumer electronics products.

Problem Definition

Goals:

- This project will address this need by developing an intelligent and fast battery charging solution for multiple battery chemistries.
- This solution will improve the charging efficiency and battery life while preventing battery damage due to overcharging.
- Implement a DC power supply for the battery charging system that will work independently of a PC.
- When interfaced to a PC, a software application will take control of the power supply to implement an intelligent battery charging solution.
- Report all feedback data of the charging process in an easy to interpret fashion – verifiable by a survey of potential battery charging solution users.

Objectives:

- Output voltage in the range of 0 – 20 V with a 20 mV resolution.
- Output current in the range of 0 – 10 A with a 10 mA resolution.
- Provide a high level of feedback during the charging process within ± 20 mV and ± 10 mA tolerances for error.
- Have a liquid crystal display (LCD) to provide user feedback – this will allow the unit to function as a standalone DC power supply when disconnected from a computer.
- Have a user friendly graphical interface compatible with Microsoft Windows to display the real-time visual feedback.

Constraints:

- Charge the following types of modern batteries:
Lithium-Ion (Li-On), Nickel Cadmium (NiCd), Nickel Metal Hydride (NiMH), and Sealed Lead Acid (SLA).
- Support rechargeable batteries in the range of 0 – 20 V.
- Require no modifications to the hardware of existing batteries (i.e. provide a universal interface).
- Must not overcharge any of the four battery types.
- Monitor the temperature of the battery and the charging environment.

- Stay within the current budget of the Electrical and Computer Engineering Department - \$300.

Design Requirements

- Output voltage in the range of 0 – 20 V with a 20 mV resolution.
 - After performing research on the commonly used rechargeable Li-On, NiCd, NiMH, and SLA battery chemistries, it was determined the 0 – 20 V range would support nearly all rechargeable batteries used to power domestic and light industrial electronic equipment.
 - The 20 mV resolution was selected as a reasonable objective mainly due to the 10 bit analog-to-digital (A/D) hardware built into commonly available microcontrollers ($\frac{20Volts}{2^{10}} \approx 20mV$). An accurate and efficient charging process and high level feedback depend upon the highest resolution attainable. This resolution must be fine enough to provide support for charging algorithms that detect subtle voltage changes during the charging cycle, yet not so fine that the voltage feedback level of precision becomes irrelevant. In addition, a much higher resolution will unnecessarily overburden the hardware and software resources required to implement the design and will introduce much electrical noise into the system. This will be further explored in the Alternative Design section.
- Output current in the range of 0 – 10 A with a 10 mA resolution.
 - Research on commonly used rechargeable batteries reveals the currents required to ‘fast’ charge the battery at the 1 C rate¹ lie within this range (fast charging is a charging rate of greater than 0.5C). Too low of a current range will not allow fast charging of high capacity batteries, while too high a current range will introduce unnecessary design complexity that would not be commensurate with the size and type of batteries intended for this project.
 - The 10 mA resolution was also determined by considering the 10 bit A/D hardware embedded in commonly available microcontrollers ($\frac{10Amps}{2^{10}} \approx 10mA$). This is a reasonable estimate to produce a high level feedback of the charging current.
- Provide a high level of feedback during the charging process within ± 20 mV and ± 10 mA tolerances for error.
 - The intelligent charging system will report the real-time feedback of percentage charge, time to charge, elapsed time while charging, voltage,

¹ A charge rate of 1 C means that the battery is charged at the same rate as its nominal capacity. For example, a 4.2 Ah (amp-hour) battery would require a 4.2 A charging rate for a period of approximately one hour, with the absolute time dependent on initial state of charge and energy conversion efficiency during the charging process.

current, and power during the charging process². This feedback will be within the specified resolution for voltage and current.

- Have a user friendly graphical interface compatible with Microsoft Windows to display the real-time visual feedback.
 - This custom PC application allows the user to select the desired battery chemistry and control the charging process. In addition, the user will receive graphical feedback of the charging process, as mentioned in the previous objective.
- Have a liquid crystal display (LCD) to provide user feedback – this will allow the unit to function as a standalone DC power supply when disconnected from a computer.
 - The LCD will display a text-only user menu selection-type interface.
 - The user will be allowed to select the constant voltage- or constant-current mode of operation. The voltage selection must be within the specified limits of 0-20 V and the current selection within the specified limits of 0 – 10 A.
 - When functioning as a DC power supply, the LCD will display the constant voltage or constant current value.
 - This display will also alert the user of any error conditions detected during the charging process or when functioning solely as a DC power supply.
- Implement an infrared remote control interface to allow user interaction with the power supply while in “stand alone” mode.
 - An infrared transmitter, combined with the LCD display form a comprehensive user interface.
 - Remote operation allows more freedom in the physical placement of the power supply.

² Not all battery chemistries may support certain feedback information, such as time to charge, without a priori knowledge of the battery condition, and as such, certain feedback information may not be supported.

Accepted Technical Design: High-Level

After performing a detailed alternative design analysis, the SmartCharger group has decided to design the battery charging system using a microcontroller, an external analog-to-digital converter, a switch-mode power supply, and an LCD and custom PC application for high-level user feedback. In the process, the group extracted a clearer understanding of the problem definition and how to meet this definition in an actual technical design. As the core of the SmartCharger design, this section will discuss all hardware and software components of the accepted technical design and is a major step towards successfully implementing the design in the spring semester.

A good initial understanding of the design can be obtained by carefully examining the overall high-level block diagram shown in Figure 1 on the following page. In order to prevent the propagation of electrical noise from the switch-mode power supply to the integrated logic control components, the system is divided into two boards: a power electronics board and a system control board. As the core source of power to the battery, the design will begin with the switch-mode power supply, proceed to the logic control circuitry, and finally progress to the custom PC application design. The resulting detailed hardware design will combine the system components into a full schematic illustrating the interfaces between the components

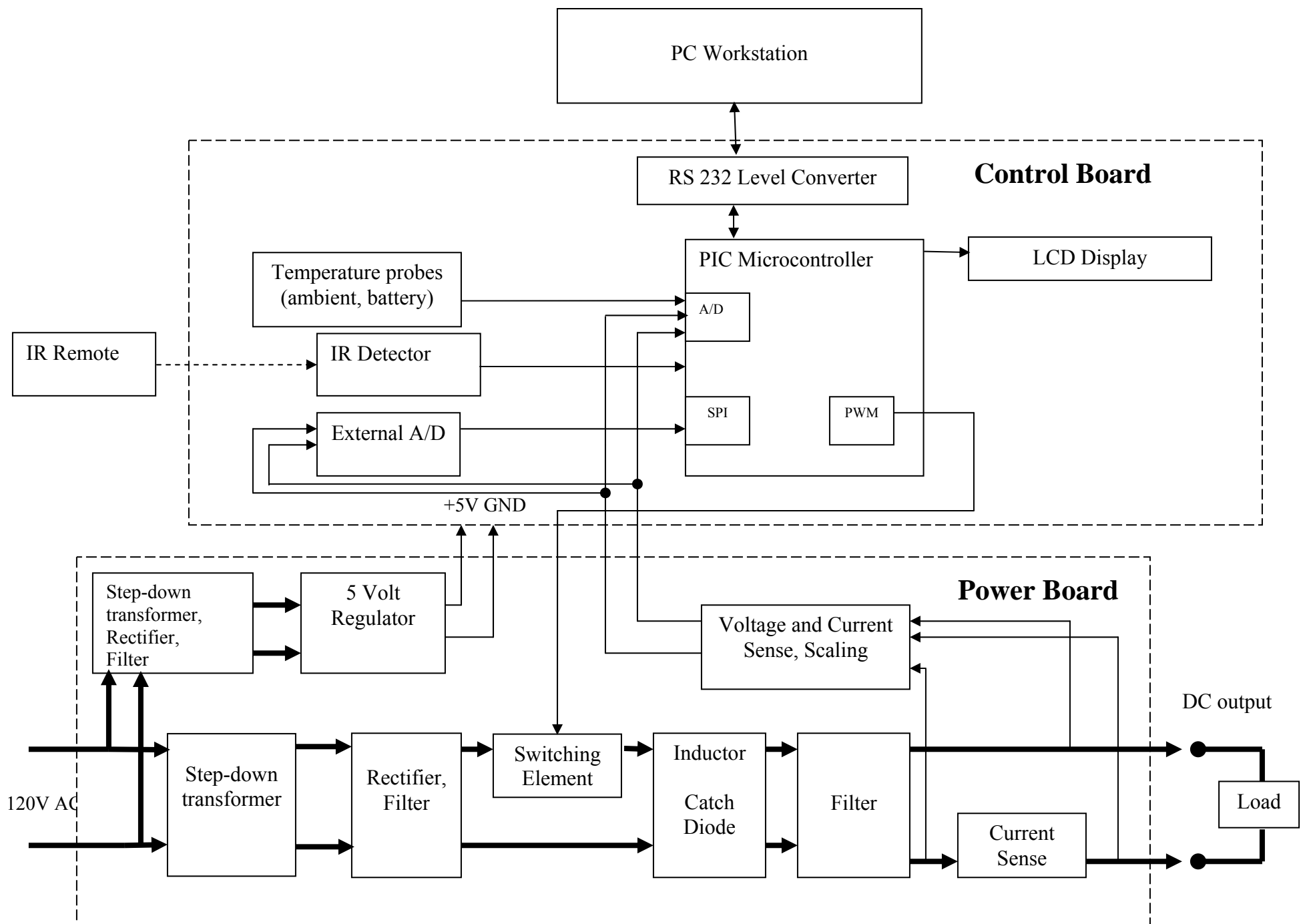


Figure 1. SmartCharger Overall Block Diagram.

DC Power Supply

The core of the Smart Charger design is the DC power supply. This power supply must operate in a constant voltage or constant current mode. With these two modes, the power supply can recharge most common types of batteries using the correct algorithms. From the knowledge gained through the Alternative Design Analysis process, the power supply will be a switching type. A switching power supply will allow for high efficiency, about 80 – 90 %, and high reliability. A switching power supply will also be smaller and lighter than a linear type. The switching power supply will have two converters: an uncontrolled full-wave bridge rectifier to obtain an unregulated DC voltage that will be fed into a buck converter to get the desired output, and a buck converter to step down the voltage to the desired level. With the two converters, a range of 0 to 20V could be obtained at the output, which should be enough to charge most batteries. An overview of this supply is shown in Figure 2.

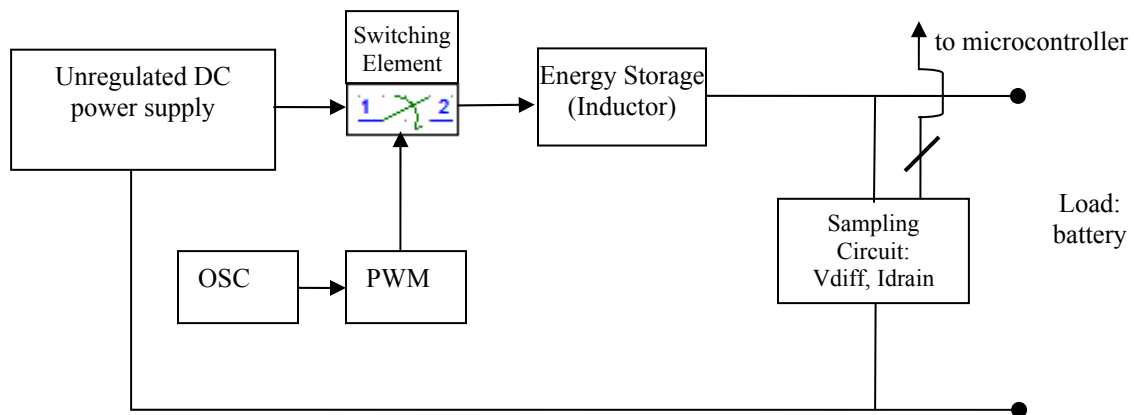


Figure 2. Switch-mode DC Power Supply Overview.

A detailed view of the power supply is shown in Figure 3. The unregulated DC comes from a full wave bridge converter. The switching element is a power MOSFET and the energy storage element is an inductor. All these parts are explained below.

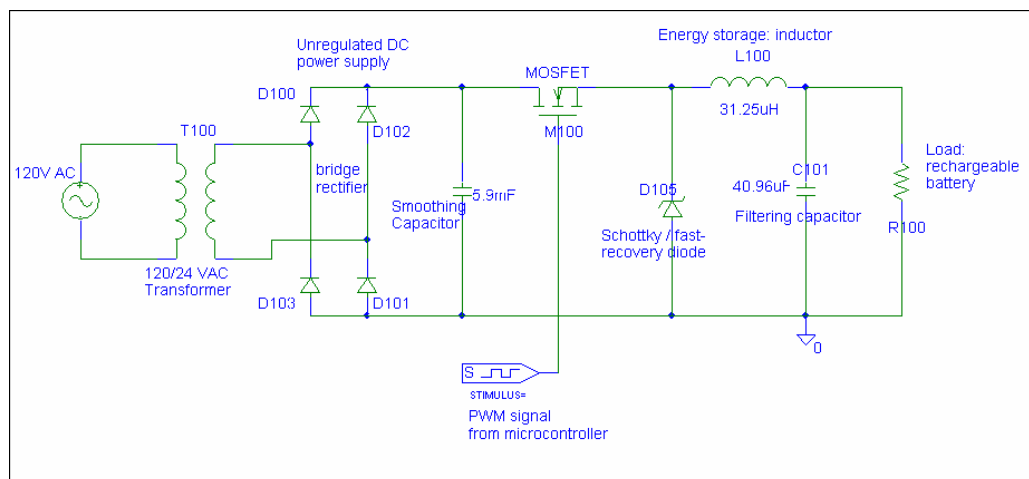


Figure 3. Switch-mode DC Power Supply Circuit Schematic

The AC line voltage signal will be stepped down to 24VAC (rms value) with the aid of a transformer. The AC transformer output voltage will go through a full-wave uncontrollable bridge rectifier. The AC input voltage is applied to the diagonally opposite ends of the bridge and the DC voltage is available on the other side. The bridge rectifier is a circuit which converts an AC voltage to a DC voltage using both half cycles of the AC input voltage.

The value of the DC voltage obtained out of the bridge can be determined by Equation 1, where V_m is the peak AC voltage:

$$V_{DC} = \frac{1}{\pi} \int_0^{\pi} V_m \sin \omega t dt = \frac{V_m}{\pi} \left[-\cos \omega t \right]_0^{\pi} = \frac{2V_m}{\pi} \quad \text{Equation 1}$$

For our power supply, V_m is $24 * \sqrt{2} = 34V$, thus yielding a V_{DC} or an average voltage equal to 22V. The ripple factor for a full-wave rectifier is given by Equation 2, which yields a 0.5V ripple at the bridge output.

$$\gamma = \sqrt{\left(\frac{V_{rms}}{V_{DC}} \right)^2 - 1} \quad \text{Equation 2}$$

Since there is no average current through a capacitor, the smoothing capacitor will have no DC current going through it, thus all the DC current will go into the MOSFET, while the AC components will be filtered out. Also, the capacitor will smooth out the unregulated voltage coming from the full bridge rectifier. Figure 4 shows the rectified varying DC (dotted line) and the smoothed DC (solid line). The capacitor charges quickly near the peak of the varying DC, and then discharges as it supplies current to the output. The smoothing capacitor value can be computed with Equation 3.

$$V_{dc} = V_m - \frac{V_m}{4fRC} \quad \text{Equation 3}$$

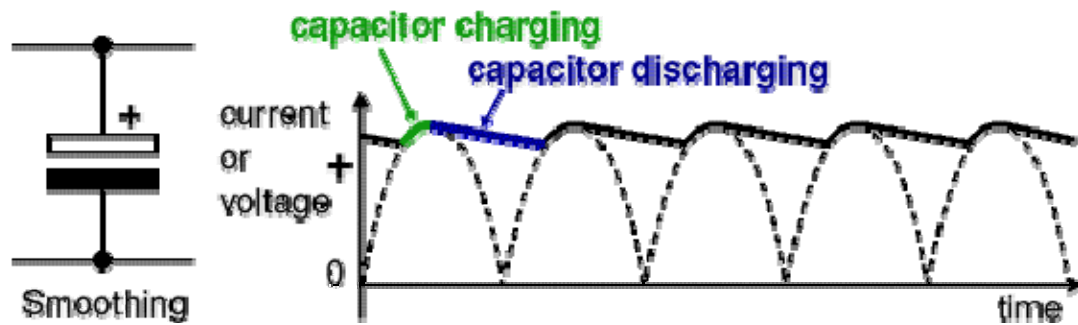


Figure 4. Smoothed DC voltage from the rectifier.

For an AC frequency of 60 Hz, an input peak voltage of 34V, a 22V DC voltage and a 2Ω maximum output resistance, the capacitor value required is 5.9mF. The larger the

capacitor, the better, since an increased level of smoothing of the input voltage will result in a nearly constant DC input voltage. Our team chose to use a 13mF capacitor.

The buck converter will decrease the input voltage to the desired value by turning a switch on and off. In order to control the supply, the microcontroller will dynamically send a PWM signal to a high side driver used to turn on and off the signal to the gate of an n-channel power MOSFET. When the MOSFET is turned on the diode will not work since it is reversed bias, and the inductor will begin to store energy. When the transistor switches off, the diode becomes forward biased, and the inductor releases energy to the load, and then the cycle repeats. The purpose of this diode is to direct current flow in the circuit and to ensure that there is always a path for the current to flow into the inductor. The Schottky diode was chosen because of the very fast switching times that the diode is capable of, and also the low forward-voltage drop given by its construction (a metal-semiconductor barrier, rather than a semiconductor-semiconductor barrier found in most of the other diodes). To determine the voltage, the PWM will have a duty ratio or duty cycle given by Equation 4.

$$\frac{V_O}{V_{IN}} = D \quad \text{Equation 4}$$

The duty cycle D will change depending on the desired output voltage V_O . The MOSFET driver has its own power supply that floats above the source of the MOSFET, and it is optically isolated from the rest of the circuit by the use of an opto-isolated chip. The chip then sends the signal to a regular high side driver, which passes it along to the gate of the MOSFET, thus turning it on and off, as desired. A boot-strap topology was attempted using a 2110 driver, with limited success, that is why the team chose the above described method to drive the MOSFET (after burning few 2110's).

An inductor is also used in the buck converter to store energy when the switch is on and to supply energy when the switch is turned off. Equation 5 was used to determine the required minimum inductance value:

$$L_{crit} = \frac{1-D}{2} * TR \quad \text{Equation 5}$$

In this equation D is the duty cycle, T is the period and R is the maximum resistance. For a PWM switching at a frequency of 32 kHz, the period T is 31.25μs and the maximum resistance R is 2Ω. For the minimum duty cycle, the largest critical inductor value is calculated to be 31.25μH. Since the value of the needed inductor is known, the following formula can be used to determine how to properly build it:

$$L = \frac{N^2 \mu A}{l} \quad \text{Equation 6}$$

Here, N is the number of turns, μ is the permeability of the core material, A is the area of the coil, and l is the average length of the coil. Given a regular size core with an area of

around 2 square inches, a permeability μ of around $25\mu_0$ for a high flux core, and an average length of the coil of 10 inches, the desired number of turns would be around 5 turns. A 25mH inductor was used in the building of the project.

A capacitor is used at the output as a filter to reduce ripple in the voltage. Since switched power regulators are commonly used with high current, high-performance power supplies, the capacitor should be chosen for minimum loss. Equation 7 will determine the capacitor value.

$$C = \frac{1 - D}{(8Lf^2)\Delta I} \quad \text{Equation 7}$$

L is the inductance and f is the frequency at which the PWM operates. Given the desired ripple ΔI as 0.1, the minimum filtering capacitor value is $40.96 \mu F$. In building the final product, a 13mF capacitor was used since it was readily available to the team.

Microcontroller Power Supply

The microcontroller is the brain of the design. It requires a constant 5V input that has to be as “clean” as possible, meaning that it is desired to have as small ripple voltage as possible. Since the current requirements are rather small (less than 1A), it was proposed that a commercial transformer will be used to power up the microcontroller. This will ensure the proper voltage will be supplied at all times, making the overall project more reliable. Figure 5 shows a general schematic of the power supplies:

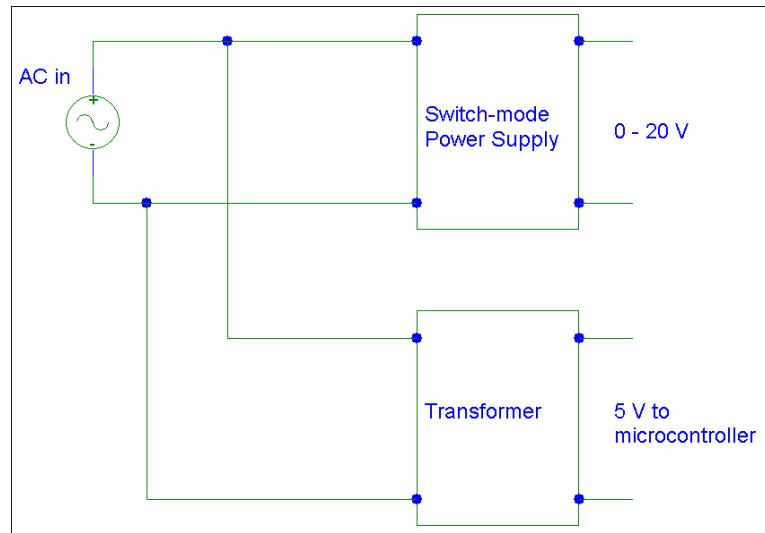


Figure 5. Microcontroller Power Supply Overview

The schematic in Figure 6 shows the basic construction of the 5V supply. It is based on the inexpensive 7805 voltage regulator.

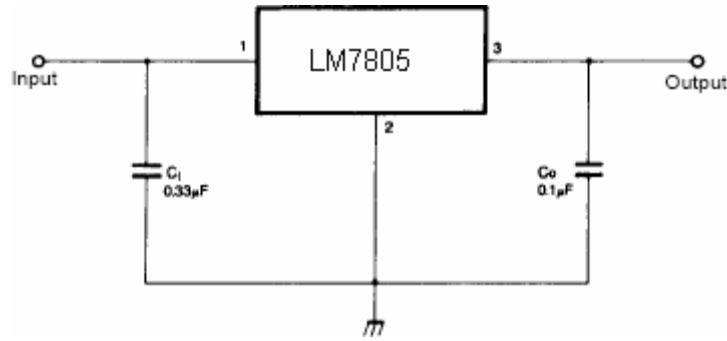


Figure 6. 5V Voltage Regulator

Voltage and Current Sensing

In order to accurately and efficiently charge a battery and deliver feedback to the user, voltage and current sensing networks must be built into the switch-mode power supply circuitry. To monitor the voltage across the battery, the 0-20 V output will be scaled using two precision resistors. The voltage output will be sent to Channel 0 on the internal PIC ADC to control the PWM for the battery charging algorithm and the standalone power supply modes of operation. This signal will also be sent to Channel 1 on the external ADC. This signal will be for the high-precision feedback to the microcontroller and PC for user feedback purposes. The voltage sensing network in the context of the power supply circuitry is shown in Figure 7. As can be seen from this schematic, a simple voltage-divider branch will scale the outputs to achieve compatibility with the PIC ADC and external ADC input voltage requirements.

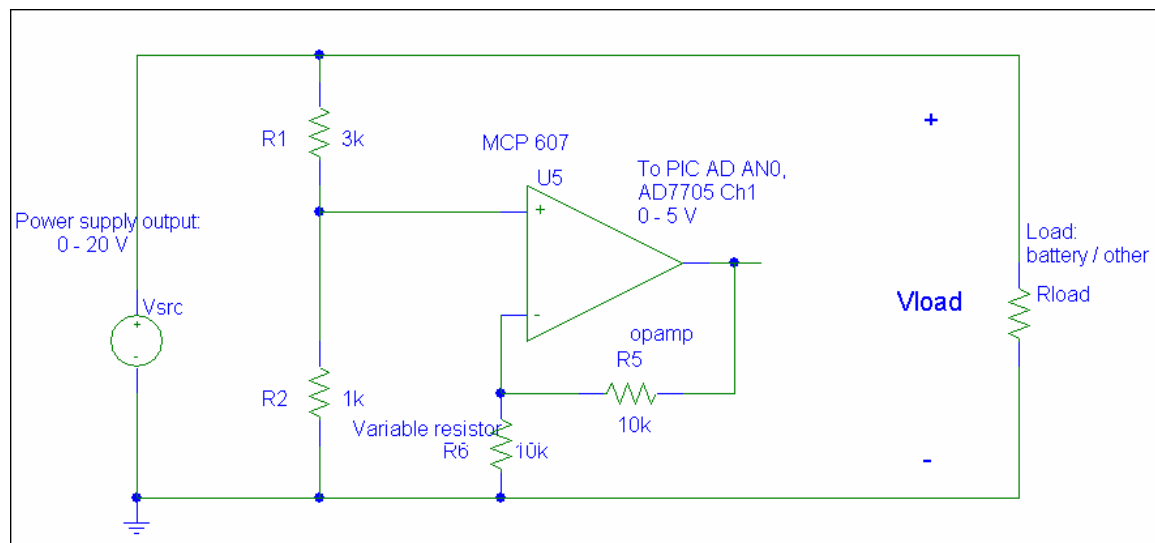


Figure 7. Voltage Sensing Network Schematic.

These signals will be isolated from the power supply circuitry using a voltage follower operational amplifier stage. The purpose of this is to prevent an over voltage condition on the ADC pins and to isolate the ADC input impedance requirements from the rest of the circuit. For this application, the Microchip MCP 607 op-amps were chosen because of their precision low power operation and rail-to-rail output swing capability. These

devices, which are shown in Figure 8, are available in dual packages, which will reserve an additional op-amps for the current sensing network.

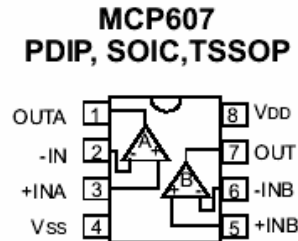


Figure 8. Microchip MCP 607 Precision Operational Amplifier.

In order to sense the current flowing through the battery, a low-side current sensing network will be used. Low-side current monitoring consists of measuring the voltage across a small current sense resistor connected in series with the ground path and dividing by the known value of the sense resistor. This technique is illustrated in Figure 9. Since the voltage measured across this $0.01\ \Omega$ resistor is small, it is amplified before being sent to the two A/D converters.

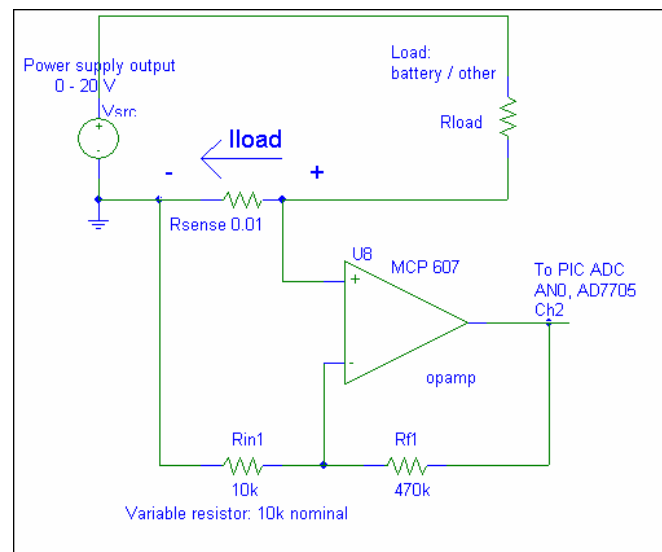


Figure 9. Current Sensing Network Schematic.

System Control: Microcontroller-Based Approach

After evaluating the options for system control, the embedded microcontroller-based approach was determined to be the most suitable alternative for this “smart” design. As the core controller, the integrated microcontroller will serve many purposes: implementing the PWM control loop to change the voltage or current output, processing the serial digital voltage and current sensing signals from the external ADC, processing the temperature sensor feedback signals, transmitting these sensing signals to the PC application via the RS-232 serial interface, processing the infrared-remote input, and driving the LCD for user feedback. This overview is illustrated in the high-level block diagram of Figure 10.

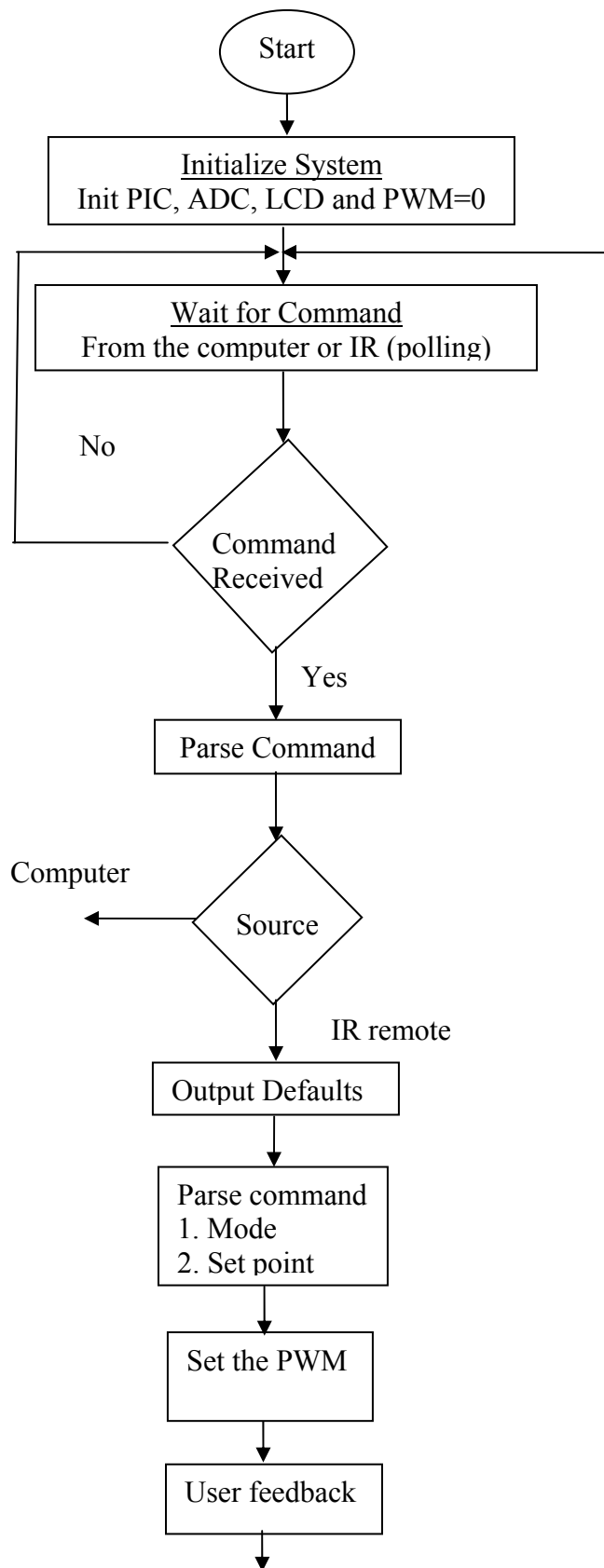


Figure 10. High Level Dataflow Diagram.

PWM Control Algorithm

To implement the “Set the PWM” block in the dataflow diagram, a simple algorithm was used. First an initial guess for the duty cycle, proportional to the present setpoint, is calculated. Next, a sample of the present voltage and current is taken from the external A/D converter. A default “delta” increment/decrement of 5 is set for the duty cycle. In voltage mode, if the current sample is within 5 % of the maximum current, this increment is limited to 1. The rate of change is determined based upon the present error and added to the present duty cycle. As long as the current is not over the limit, or overloaded, this calculated duty cycle is set. The procedure is similar in current mode, except that the checks against maximum current and overloaded current are changed to checks against the voltage.

The pwm control algorithm described was modeled in Matlab. The plot in Figure 11 shows the simultaneous voltage and current outputs along with the resistance as a function of the duty cycle.

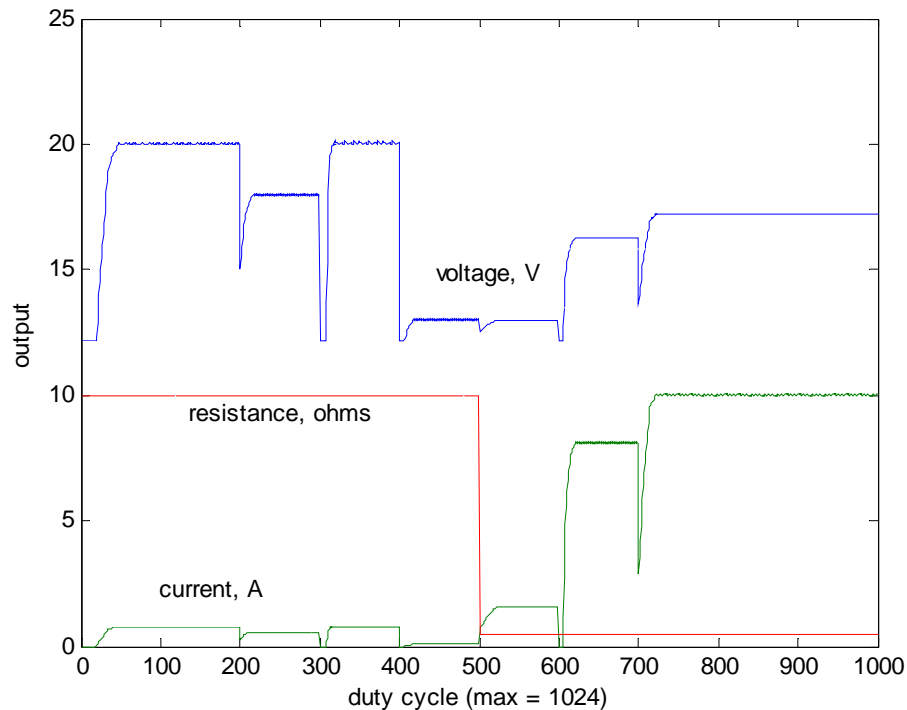


Figure 11. System output as a function of the duty cycle.

The detailed pwm control algorithm is included in the Appendix in the file “pwmcontrol.c”.

Microcontroller Hardware Design Interface

The 40-pin Dual In Line PIC18F452 and 8 MHz clock oscillator that will be used as input to pins 13 and 14 is shown in Figure 12. The microcontroller will actually operate at 32 kHz by setting a 4x phase lock loop multiplier.

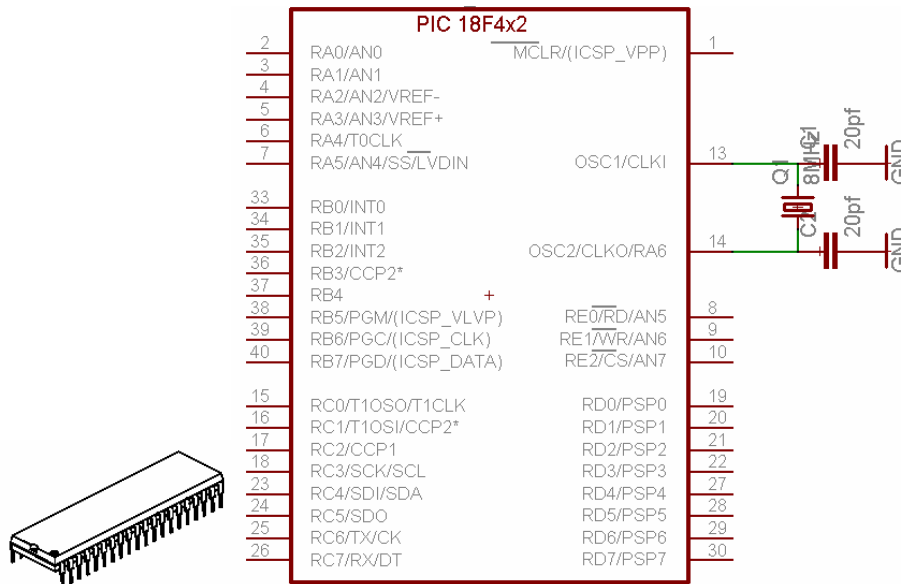


Figure 12. 40-pin DIL PIC18F452 Package and Schematic (including the 8 Mhz clock oscillator).

Pin 17 will be used as output for the PWM signal (through a logic board to power supply board header), input / output (I/O) pins 18, 23, and 24 for the SPI communication to the external ADC, I/O pins 25 and 26 for the Maxim RS-232 interface, and output pins 19-21 and 27-30 of PORTD for the liquid crystal display controller command and data bus. The final design will also include an LED for testing and verification purposes (although not shown to the user). Each of these subcomponents will be considered separately.

External Analog-to-Digital Converter

The Alternative Design Analysis clearly revealed that the high-quality feedback required to control the battery charging algorithm is a vital part of this design. As a result, a high-resolution external A/D converter (ADC) was selected over the built-in converter. The function of this device is to convert the differential analog voltage and current sensing signals into digital signals to be processed by the microcontroller. The microcontroller will report these digital signals to the computer application that will display the signals and adjust the charging algorithm if required.

The driving criteria behind selecting a suitable external ADC was high-resolution; the sampling frequency need not be very fast for data acquisition of battery systems. As a result, the Analog Devices AD7705 16-bit Sigma-Delta ADC was chosen. This device yields a resolution of $20V/2^{16} = 0.305 \text{ mV}$ over a 0-20 V design constraint. The AD7705 features no missing codes, two fully-differential analog input channels, a differential reference input, 4.75 V to 5.25 V operation, a three-wire Serial Peripheral Interface (SPI), and many more features. Particularly attractive is the high degree of compatibility with the microcontroller: the flexible serial interface and SPI protocols are fully compatible with the PIC18F452 and the supply voltage range allows for the microcontroller and ADC to be powered on the same system logic board

For this battery monitoring application, one input channel will be used to differentially measure the voltage across the battery. The other input channel will be used to monitor the current to the battery. The AD7705 device and input / output schematic is

shown in Figure 13. The external ADC will be placed on the system logic board which is powered by the 5 V supply (VDD). This particular device requires a 2.4576 Mhz oscillator as clock for MCLK, and communication is done on the (active low) data ready, or DRDY' signal. There will be a logic header attaching the ADC to the other integrated board components.

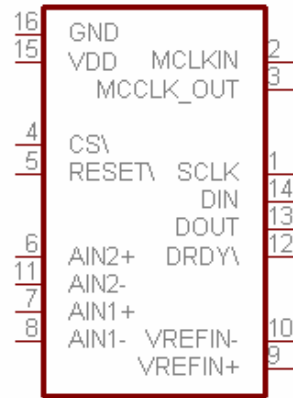


Figure 13. AD7705 Interface Schematic.

To communicate with the external ADC from the microcontroller, the pseudo code driver shown in Listing 1 will be implemented. The embedded C pseudo code displayed for all listings uses the actual functions and variables provided in the CCS C compiler.

```
adc_init();           // Initialization routine: configures clock, chip
                      // select, reset pins, delay; calls
                      // setup_adc_device() to setup the
                      // device parameters

// Start A/D conversion
if (required_time_elapsed) // Endless synchronous reading
    if (ADC_DRDY){
        globalVoltage = read_adc_value(ch 1);
        globalCurrent = read_adc_value(ch 2);
    }
```

Listing 1. Pseudo code to Communicate with the External ADC.

RS-232 Serial Interface

The bi-directional feedback between the microcontroller and PC application will be made possible via the Maxim RS-232 serial interface. This device performs a level conversion from the TTL level signals of the microcontroller to the RS-232 levels of the computer. This hardware interface is shown in Figure 14.

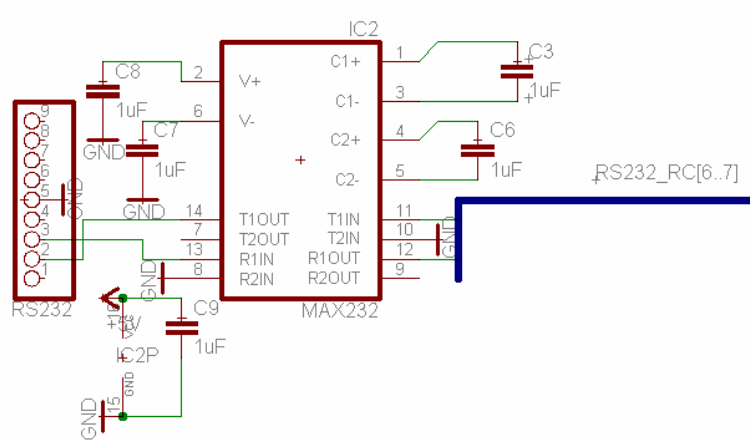


Figure 14. Maxim RS-232 Serial Interface.

The communication protocol is shown in Table A1 in the Appendix.

Pulse Width Modulation

The Pulse Width Modulation (PWM) signal will be generated by the microcontroller to adjust the voltage or current output of the DC power supply. A change in pulse width is initiated by either a change in the desired voltage or current from the infrared remote or from the PC charging application. This signal will be sent to the base of the power MOSFET in order to perform this function. In order to isolate the system control board from the power electronics board, the signal will be passed through an isolation buffer (TC1141N). The PWM signal will be sent on pin CCP1 (pin 17) and generated with the pseudo code shown in Listing 2:

```
void pwm_init(){
    setup capture compare module for PWM;
    configure PWM frequency;
}

// In the endless loop, use:
pwm_init();           // Initialize PWM to zero width
value = read_adc();    // Feedback on both channels
set_pwm1_duty(value); // Sets pulse high time for each cycle
```

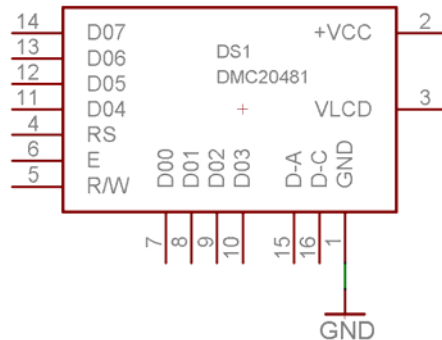
Listing 2. PWM Signal Pseudo code

Liquid Crystal Display Interface

A central component to the user feedback of this system is the liquid crystal display (LCD). The LCD will be used to display the load voltage, load current, and battery temperature (ambient, etc.) to the user during both the battery charging and the standalone programmable power supply modes of operation. For this design, the 40x4 (40 columns x 4 rows) CrystalFontz alphanumeric LCD with the built-in Hitachi 44780 controller and LED backlighting will be implemented for this important purpose. The manufacturer's sample picture of this LCD is shown in Figure 15 below.



The LCD display and controller hardware interface is shown in Figure 16. The microcontroller will communicate with the LCD using a four-bit bus consisting of D00-D03—a “nibble mode” operation. In addition, the RS, E, and R/W lines shown in the figure are used for command / data, enable, and read / write signals, respectively.



A sample screen during operation of the SmartCharger system is illustrated in Figure 17.

[illegible]

Figure 17. LCD Screen During Battery Charging Mode of Operation.

The pseudo code to communicate with the LCD is included in Listing 3 below.

```

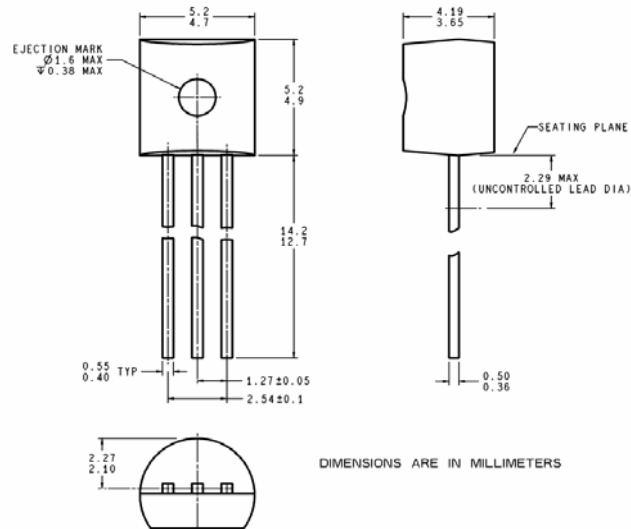
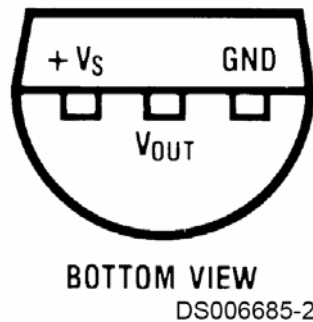
lcd_init(); // Initialize the display.
lcd_gotoxy(left-most column, top row); // position (1,1)
// Default screen - specify desired putc function
printf(lcd_putc, "Mode: V ... Temp. ...", temp);
if (display to be updated){
    lcd_gotoxy(x loc., y loc.); // Go to location on LCD.
    printf(lcd_putc, "Message to Display...", temp);
    // Calls lcd_putc() to place each character on display
}

```

Precision Temperature Sensing

The LM34 precision temperature sensor is a 3 terminal device that provides a voltage output in linear proportion to temperature. The output of the device is 10mv per degree Fahrenheit, referenced from 0 degrees Fahrenheit. The device has a guaranteed accuracy of 1° F at 70° F . For example, 0°F outputs 0mV, 70° F outputs 700mV. The device can be powered from a 5-30 volt source. The device comes in multiple packages, including a TO-92 plastic package that will be used in this design. The LM34 package is shown in Figure 18. The function to read the temperature is displayed in Listing 4 below.

TO-92 Plastic Package



Order Number LM34CZ, LM34CAZ or LM34DZ
NS Package Z03A

Figure 18. LM34 Precision Temperature Sensor Package.

```
double getTemperature(int adcValue){  
    //10 bit adc, max adc input at 5000mV  
    return adcValue/((2^10)-1)*(5/10);    // Temperature calculation  
}
```

Listing 4. Pseudo code to return Desired Temperature Reading.

Infrared Remote Input

The Infrared detection is implemented with the Sharp GP1UM28YK infrared detecting unit. This unit combines a number of signal processing functions in a compact 3 terminal device. The operational block diagram of the device can be seen in Figure 19.

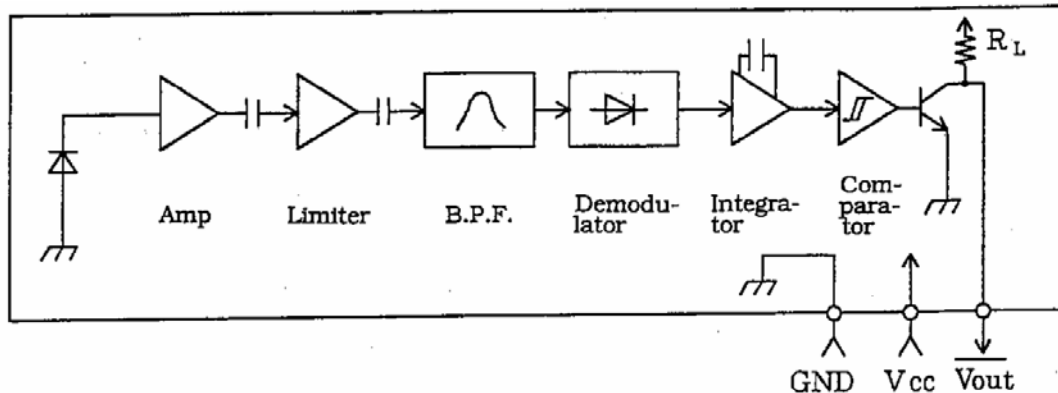


Figure 19. Sharp GP1UM28YK Operational Block Diagram.

The device discriminates between a plethora of extraneous infrared noise sources and locks in on the modulated 38 kHz source produced by an infrared remote control. The device implements negative logic, and outputs approximately 5 volts during inactive periods, and 0 volts when in the presence of the 38 kHz source. The device is connected to the microcontroller via an external interrupt pin that is configured to interrupt on falling edge signals. The microcontroller interprets the command from the infrared remote control by timing the interrupts, or falling edges of the signal, in the interrupt service routine. The C code to implement this function is shown in Listing 5.

```
Initialization;
void IR_isr(){
    isr_initialization;
    if (IRhit)
        return;          // Don't process another command until
                          // last command is read
    deltaTime = timer() - lastTime;
    lastTime = timer();
    if (deltaTime_too_small || deltaTime_too_large) {
        receiveBit = 0;
        return;
    }
    set_databit_based_on_deltaTime;
    shift_left(IRFrameAddress, shift_amount, databit);
    if(entireFrameReceived && IRFrameValid)
        IRhit = TRUE;}

int IRhit() { return IRh; }

int getIRCommand() {
    IRh = FALSE;
    Return IRFrame.command;
}
```

Listing 5. Infrared Receiving Routines.

PC Software Application

The PC application was written using Visual Basic 6 development environment. This is an object oriented language very popular with Microsoft Windows developers. The application will be engineered using an object oriented style by using the notion of class modules and objects. These class modules implement data encapsulation by providing a public interface while hiding the internal data and implementation. To further segment the functionality of the application, and to allow for enhanced testing and features, the application was written as a client-server application.

Because the application is object oriented, it makes sense to model much of the application using UML (unified modeling language). This is an open standard modeling language managed by the Object Management Group (OMG), and used extensively in industry. One benefit of using a graphical modeling language is that the computing system can be understood by the computer programmer as well as client users of that software. The use of UML in documenting Object Oriented (OO) software design has become the de-facto standard, and in fact “the UML’s importance comes from its wide use and standardization within the OO development community. The UML has become not only the dominant graphical notation within the OO world but also a popular technique in non-OO circles.”³

This document will primarily utilize UML diagrams to provide a general, overall software system description with supplemental descriptions for those not familiar with UML syntax. The use of pseudo code will also be utilized to provide more focus on the underlying details. The actual code used in the application can be seen in the Appendix.

Figure 20 represents the general system use case with respect to the user. The user may interact with the Power supply hardware directly to power a general DC load via the local hardware user interface. The user may also interact with a personal computer (PC) via 2 different client applications. These include a text based interface via windows HyperTerminal, and a graphical interface via the custom visual basic application. The server application can service a single instance of both of these applications simultaneously. The client and server can run on the same machine by configuring the client to connect to the local machine name or TCP/IP loop back address 127.0.0.1. The text based interface via HyperTerminal connects through port 23, and the application connects through port 107. By using these ports the client can connect via the university wireless network and not get blocked by the universities firewall that blocks many in not all of the user configurable ports above 1024. A connection can even be established by configuring a virtual private network (VPN) connection, that would allow access to the server application running in the senior design lab, and therefore the smart charger hardware in the lab, from anywhere in the world. The practical advantage of the client server arrangement is that a single application could access multiple charger hardware, and that hardware may be located in harsh environments such as in a factory. The server application not only acts as a bridge between the RS232 interface of the device and a TCP/IP connection, but also decodes the proprietary packet structure that is native to the device, formats this data for both types of client applications, and also acts as an emulator for the hardware that allows for testing of the software, such as battery charging algorithms, without actually being connected to the hardware or an actual

³ *UML Distilled*, Martin Fowler, Addison-Wesley publishing 2004

battery. The emulator can be configured to emulate a variety of different loads by calling a particular load model. Models that were implemented included a fixed resistor model, time variant variable resistance model, constant dVdt model as well as a sealed lead acid battery model and a NiCd battery model. All of the charging algorithms were tested on these two battery models to verify that the algorithms transition as expected.

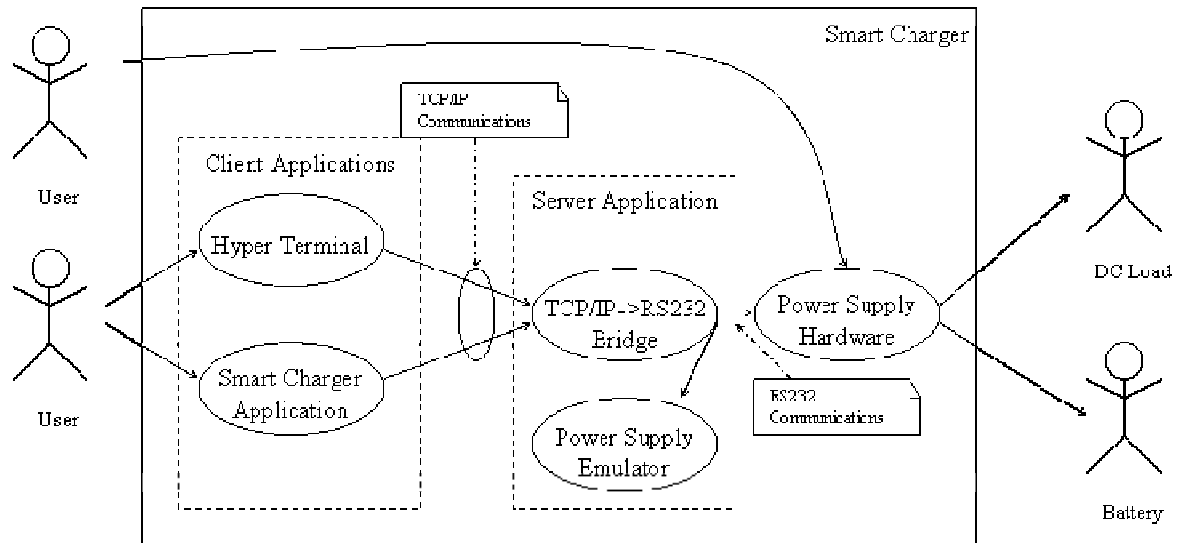


Figure 20. System Use Case Diagram.

The operation of object oriented programs is determined by the class objects used to create the application and the interaction of these class objects. Figure 21 shows the overall class diagram of the smart charger application. Each class object is further described in the discussion.

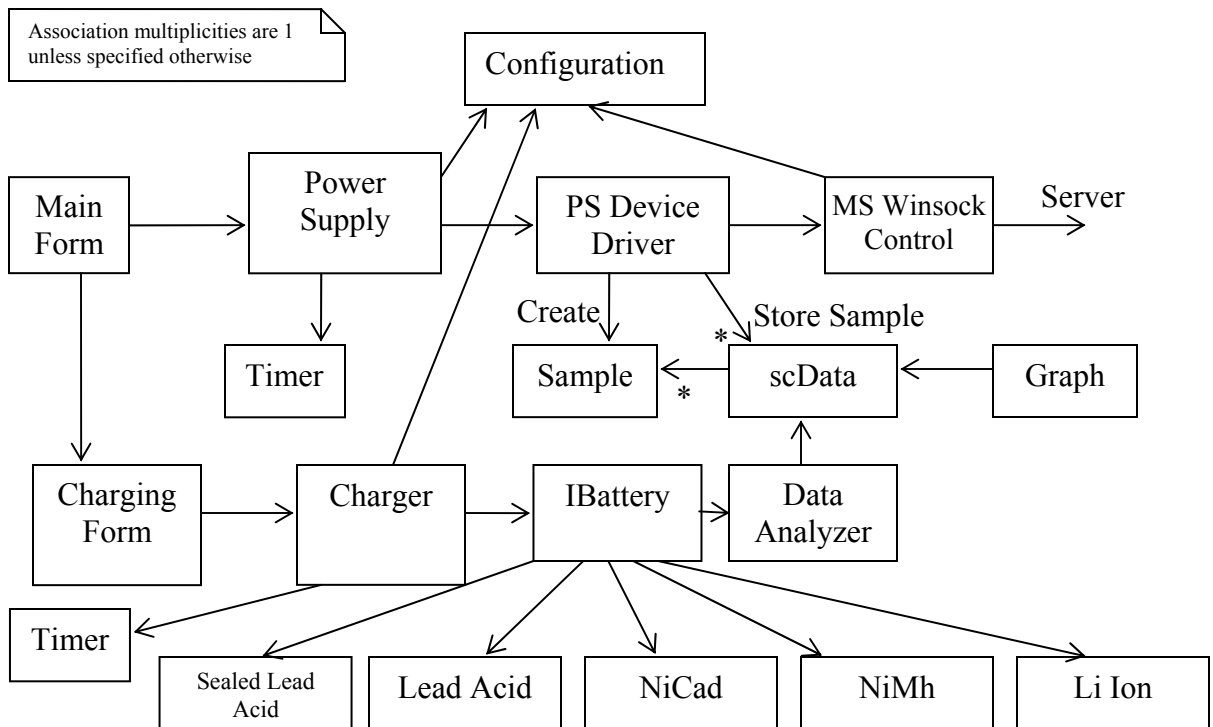


Figure 21. Overall Smart Charger Application Class Diagram.

Main Form

The main form implements the main user interface when the application starts. This form provides all the control and feedback necessary to operate the power supply manually. This form interacts with the Power Supply class to allow manual operation of the power supply. The main form is the starting point in opening other application forms, such as the options form and the charging form via a menu bar control.

Configuration Class Summary		
Primary Attributes ComSettings:String PSSettings ChargerSettings	Primary Methods Load() Save() getComSettings() getPSSettings() getChargerSettings()	Primary Responsibilities <ul style="list-style-type: none"> • Loads configuration data from registry • Dispatch settings via public interface • Save edits to registry

The configuration class contains initialization information used by various component classes of the application. This includes communication port and baud rate setting, update rates, and sampling frequency settings. The Configuration class stores the

configuration data in the operating system registry. The information contained in the configuration class can be edited via the options form.

PowerSupply Class Summary		
Primary Attributes Mode:MODETYPE VoltSet:double CurrentSet:double	Primary Methods setVoltage(double) setCurrent(double) getSample():Sample timerEvent()	Primary Responsibilities <ul style="list-style-type: none"> • Direct power supply hardware in response to user interactions

The PowerSupply class is a high level representation of the power supply. Public methods of this class include those that a power supply user would typically invoke, including setVoltage and setCurrent. These parameters are stored as data members of the class. The PowerSupply class sends messages to PSDeviceDriver, which is a class in the chain that implements communication between the PowerSupply class and the physical hardware. PowerSupply periodically queries the hardware as a response to an event trigger (typically once a second) from the Timer object. The device then sends a status message indicating the Voltage, Current, and Temperatures. This information is stored as an instance of a Sample object, with a time stamp appended to the data, and then stored in the scData container class.

PSDeviceDriver Class Summary		
Primary Attributes	Primary Methods setVoltage(double) setCurrent(double) getSample():Sample recEvent():Event	Primary Responsibilities <ul style="list-style-type: none"> • Data conversion between higher level PowerSupply class and server • Package monitored parameters (voltage, current, temperature, time) into Sample object

The PSDeviceDriver receives messages from server, decodes the message data, and distributes the extracted data and control information to the rest of the application. PSDeviceDriver also encodes messages that are then routed to the server application that ultimately gets forwarded to the smart charger power supply hardware to control its operation. PSDeviceDriver uses a similar packet structure to that used by the server to communicate with the device, with the major difference being the data is formatted in space delimited ASCII strings and not byte oriented raw data, and no checksum is used between server and client because the TCP/IP protocols enforce data integrity. Commands sent to the server begin with a upper case letter that indicates the command

type, and responses from the server begin with a lower case letter. This packet structure is given in Table 1.

Command	Data	Description
“a”	Na	Acknowledged
“n”	Na	Not Acknowledged
“p”	Value (0-1023)	PWM value
“s”	Mode,control,error	Status
“r”	voltage, current, ambient temperature, battery temperature	Sample Data
“O”	0=off, 1=on	Set output on/off
“C”	0=local control 1=pc control	Set control
“R”	Na	Get sample
“S”	Na	Get status
“I”	Current	Set current
“V”	Voltage	Set voltage
“P”	Na	Get pwm value

Table 1. Application-Server data packet structure.

Sample Class Summary		
Primary Attributes Voltage:double Current:double AmbientTemp:double BatteryTemp:double TimeStamp:double	Primary Methods getVoltage():double getCurrent():double getATemp():double getBTemp():double getTime():double Sample(V,C,AT,BT,TS):Sample	Primary Responsibilities <ul style="list-style-type: none"> • Container class for monitored parameters

The Sample class is a container class for monitored voltage, current, temperature, and time. A sample represents a snapshot of these monitored parameters at a particular instant of time. This data is converted to a Sample class type, and ultimately stored in the Data container class. Listing 6 shows the pseudo code function for building a Sample object from a communications packet received from the device.


```

function getSample(byte[] m) as Sample
    S=new Sample
    // int make16(byte HiByte, byte LoByte)
    //ret var = fraction of max x full scale value
    double V = make16(message[1],message[2])/(2^16-1)*20
    double I = make16(message[3],message[4])/(2^16-1)*10
    double TA = make16(message[5],message[6])/(2^10-1)*500
    double TB = make16(message[7],message[8])/(2^10-1)*500
    double time = SystemTimer()

    return Sample(V,I,TA,TB,time)
end function

```

Listing 6. Building a Sample object from packet data.

scData Collection Class Summary		
Primary Attributes DataSet.Sample[]	Primary Methods addSample() getSample(int index)	Primary Responsibilities <ul style="list-style-type: none"> Provides “History” of charging process Container class for Sample data Provides raw data to DataAnalyzer class

The scData class is a storage class that holds an array of samples. It is used by the Graph class to display data and trends over time. The scData class is also used by the DataAnalyzer class which processes queries from the Charger class during charging operations.

DataAnalyzer Class Summary		
Data Members Data DataSet (byRef)	Methods getVoltage(int n=60) getCurrent(int n=60) getTemp(int n=60) getDVdt(int n=60) getD2Vdt2(int n=60) getDTdt(int n=60) getDeltaT(int n=60)	Responsibilities <ul style="list-style-type: none"> Data Analysis

The DataAnalyzer class is a utility class that is used by the Charger class to query the Data class. Recall that the Data class is an ordered collection of Sample objects, and these Sample objects represent a data set of voltage, current, temperature and time. The Data class provides the raw history data used by the DataAnalyzer class. The DataAnalyzer exposes a number of public methods that is necessary information to implement a charging algorithm. Most of the methods accept an optional integer

argument of time in seconds, indicating a return value of the average data for the last t seconds. For example `getVoltage()` returns the last voltage sample, whereas `getVoltage(60)` returns the average voltage of the sample data for the past minute. This averaging is advantageous because the charging process is slow and the averaging will minimize the effect of an inconsistent sample.

Other methods include the first derivative of voltage, temperature, or current with respect to time. To get an accurate sense of rate of change for a parameter, the default behavior is to look at the last minute of samples to determine an average rate of change for the previous minute. To determine the average rate of change among many samples, we apply a linear curve fit to the data using a linear least square regression algorithm and determine the slope of this best fit line. The regression algorithm requires vector-scalar subtraction, and vector multiplication which are coded as private member functions of the class. Listing 7 is an example pseudo code implementation for `getVoltage()` and Listing 8 implements the first derivative of temperature with respect to time.

```
function getVoltage(optional t=0) as double
    //DC is a reference to the Data collection
    //index 0 is always the latest data (FIFO type data structure)
    v=0
    n=getIndex(t)    //function that returns the index of the last data
                    //from within t seconds ago
    for i=0 to n-1
        v=v+DC[0].getVoltage()
    next i
    return v/n
end function
```

Listing 7. Pseudo code for data analysis voltage calculation.

```
function getDTdt(optional t=60) as double
    //returns the first derivative of battery temperature with
    //respect to time
    //DC is a reference to the Data collection
    //index 0 is always the latest data (FIFO type data structure)
    T=0;
    n=getIndex(t); //function that returns the index of the last data
                  //from within t seconds ago

For loop1 = 0 To n
    bt(loop1) = mvarData(loop1).batTemp
    ts(loop1) = mvarData(loop1).timeStamp
Next loop1
getdTdt = leastSquaresSlope(ts, degT) * 60 'scale to F/min

xmean = mean(ts)
ymean = mean(bt)
xmxmean = vectorMinusScaler(ts, xmean)
ymymean = vectorMinusScaler(bt, ymean)
sumx2 = vectorMultiply(xmxmean, xmxmean)
sumxy = vectorMultiply(ymymean, xmxmean)
If sumx2 = 0 Then
    return 0
Else
    return = sumxy / sumx2
```

```
End If
end function
```

Listing 8. Pseudo code for data analysis battery temperature first derivative

Charging Form

The charging form is called by the main form when the user selects charging from the tools menu. This form accepts user input regarding a particular battery a user wishes to charge. This information is passed to the Charger class. The primary information necessary to implement a charging algorithm include battery chemistry, Amp Hour capacity of the battery, and number of cells. A sequence diagram for a typical charging scenario can be seen as Figure 22.

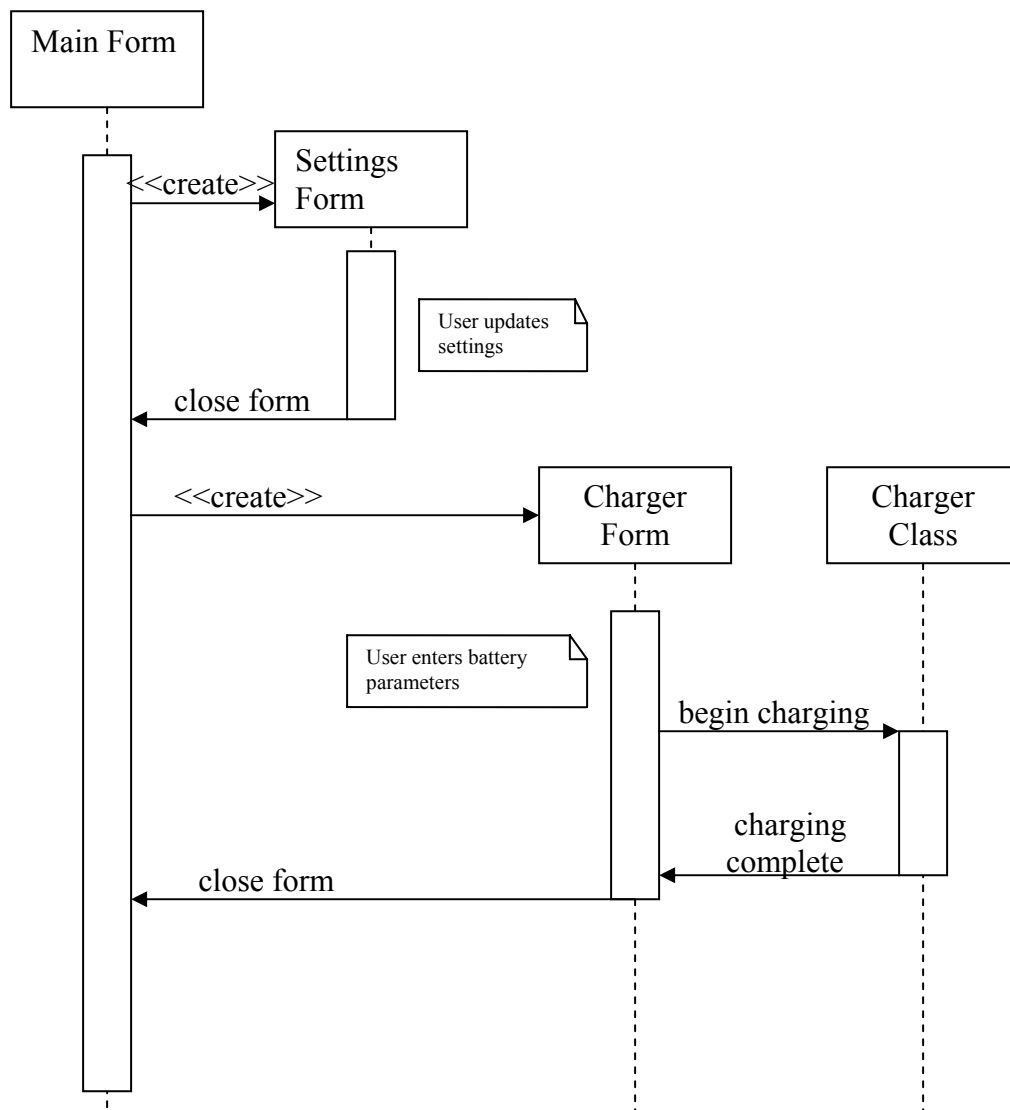


Figure 22. Charging sequence diagram.

Charger Class Summary		
Data Members Chemistry:BATYPE AhCapacity NumberOfCells:Int TerminalVoltage	Methods timer()	Responsibilities <ul style="list-style-type: none"> • Process data from charger form • Determine appropriate charging algorithm based on user feedback • Dispatch charging algorithm via periodic Timer event

The Charger class implements the functionality of a battery charger by directing PowerSupply to apply voltage or current to the battery as directed by a charging algorithm. The Charger class receives information about the battery to charge from the Charging Form. From this information a battery chemistry class (Lead Acid, NiCad, NiMh, Li-Ion) is referenced via the IBattery abstract interface class. This class contains the method definitions that a particular chemistry is required to implement, and all chemistry classes implement the IBattery interface, which implements the specific charging algorithm for the particular battery chemistry. The Charger class enables a Timer object that acts as a free running periodic event trigger. The Timer object calls a public method of Charger that periodically dispatches the charge method of a referenced chemistry class.

The following battery chemistry classes implement the charging algorithm for that particular chemistry. The battery charging process is modeled as a state machine, where the entire charging process may contain many states.

As a general rule, a fast charging algorithm requires a more stringent, more complex algorithm than a slow charging algorithm. For example, if the charging process is not terminated when a battery is fully charged, excess energy can cause gas to be produced in the cell, as well as heat. At slow charge rates the gas can be safely vented and the heat can safely be dissipated. During a fast charge, the cell may not be capable of dissipating the excessive energy, and a cell could rupture. Figure 23 represents the state diagram for the charging process in a general sense.

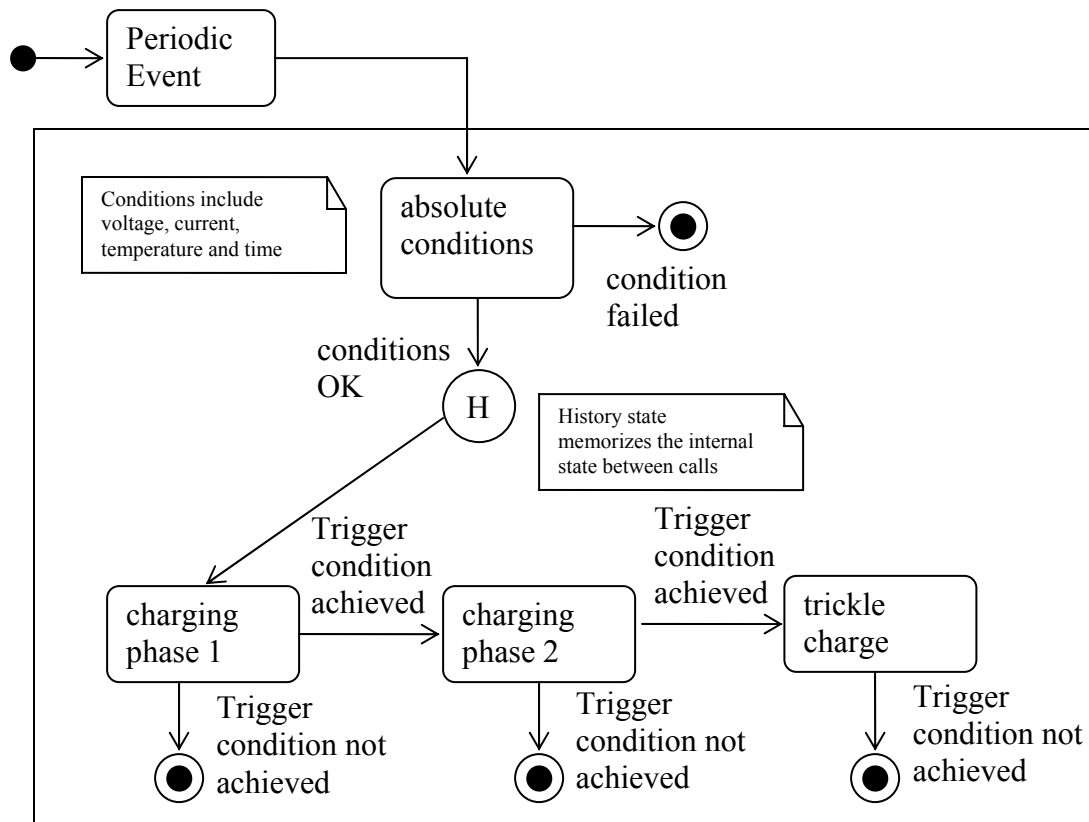


Figure 23. Charging process general state diagram.

Lead Acid Class Summary		
Data Members C_rate:Double Cells:Int TerminalVoltage	Methods fastCharge()	Responsibilities <ul style="list-style-type: none"> Implement charging algorithm for lead acid battery types Implements the IBattery interface

The lead acid battery is charged using voltage regulation with current triggers. Listing 9 is an example of a fast charge algorithm for lead acid batteries.

```

function charge()
    static state=0 //store state information between calls
    //DA is the data analysis object
    //PS is the PowerSupply object

    //process general conditions that are state independent
    if DA.getTemperature() < 0F call terminateCharge()
    if DA.getTemperature() > 113F call terminateCharge()
  
```

```

if DA.getDeltaTemp() > 10F call terminateCharge()
if DA.getDeltaTime() > 60 call terminateCharge()

select case (state)
case (0)
  PS.setCurrent(C_rate)
  if DA.getVoltage() > (cells * 2.45 volts)
    state=1
  end if
case (1)
  PS.setVoltage(Cells * 2.45 volts)
  state=2
case (2)
  if DA.getCurrent() < (.2 * C_rate)
    PS.setVoltage(Cells * 2.25 volts)
    state=3
  end if
case (4)
  //trickle charge until time expires
end select
end function

```

Listing 9. Lead acid fast charging algorithm.

NiCad Class Summary		
Data Members C_rate:Double Cells:Int TerminalVoltage	Methods fastCharge() slowCharge()	Responsibilities <ul style="list-style-type: none"> Implement charging algorithm for Nickel Cadmium (NiCad) battery types

NiCad batteries are charged at a constant current. Many end of charge termination conditions exist based upon voltage and temperature. Inflection point detection calculation of the second derivative of voltage is an advanced method of charge termination that has superior performance over other termination methods.

Temperature sensing also produces valuable indications of the charging process. The charging process for NiCad cells is endothermic, and the battery temperature can actually drop below ambient temperature until the battery reaches a fully charged level, when the process quickly turns exothermic and battery temperatures rise. Other voltage termination triggers, and temperature triggers can be used as a backup to the primary trigger. Listing 10 is an example of a fast charge algorithm for NiCad batteries.

```

function fastCharge()
  static state=0 //store state information between calls
  //DA is the data analysis object
  //PS is the PowerSupply object
  //process general conditions that are state independent
  if DA.getTemperature() < 41F call terminateCharge()

```

```

if DA.getTemperature() > 104F call terminateCharge()
if DA.getDeltaT(>15F call terminateCharge()
if DA.getDeltaTime() > 90 minutes call terminateCharge()
if DA.getVoltage() >= (1.7V * Cells) call terminateCharge()
select case (state)
  case (0)
    PS.setCurrent(C_rate amps)
    state=1
  case (1)
    if DA.getD2Vdt2()=0 eoc=true          //inflection point or
    if DA.getDvdt()<=-20 mv/min eoc=true  //-dv/dt (backup) or
    if DA.getDtdt()>=1.8F/min eoc=true    //dT/dt (backup) then end
    if eoc                                // fast charge
      PS.setCurrent(C_rate*.025) //setup trickle charge
      state=2
    end if
  case (2)
    //trickle charge until time expires
end select
end function

```

Listing 10. Nickel Cadmium fast charging algorithm

NiMh Class Summary		
Data Members C_rate:Double Cells:Int TerminalVoltage	Methods fastCharge() slowCharge()	Responsibilities <ul style="list-style-type: none"> Implement charging algorithm for Nickel Metal Hydride (NiMh) battery types

Nickel Metal Hydride batteries are charged at a constant current in a similar fashion as the Nickel Cadmium battery. The voltage termination characteristics are typically not as pronounced as the NiCad battery. Temperature indicators for the NiMh battery are also more difficult to determine because the charging reaction is exothermic throughout the charging cycle, although heat is more rapidly generated when the cell is being overcharged. As with the NiCad cell, a comprehensive charge termination strategy includes multiple trigger conditions, or backup conditions, and many of these conditions are identical to those used with NiCad batteries. Listing 11 is an example of a fast charge algorithm for NiMh batteries.

```

function fastCharge()
  static state=0 //store state information between calls
  //DA is the data analysis object
  //PS is the PowerSupply object
  //process general conditions that are state independent
  if DA.getTemperature() < 41F call terminateCharge()
  if DA.getTemperature() > 104F call terminateCharge()
  if DA.getDeltaT(>15F call terminateCharge()

```

```

if DA.getDeltaTime() > 90 minutes call terminateCharge()
if DA.getVoltage() >= (1.8V * Cells) call terminateCharge()
select case (state)
  case (0)
    PS.setCurrent(C_rate amps)
    state=1
  case (1)
    if DA.getDVdt() <= -10 mv/min eoc=true  //-dv/dt  or
    if DA.getDTdt() >= .9F/min eoc=true    //dT/dt (backup) then end
    if eoc                                // fast charge
      PS.setCurrent(C_rate*.025)          //setup trickle charge
      state=2
    end if
  case (2)
    //trickle charge until time expires
end select
end function

```

Listing 11. Nickel metal hydride fast charging algorithm

LiIon Class Summary		
Data Members C_rate:Double Cells:Int TerminalVoltage CellVoltage	Methods fastCharge() slowCharge()	Responsibilities <ul style="list-style-type: none"> Implement charging algorithm for Lithium Ion (LiIon) battery types

Lithium Ion (Li Ion) batteries are the most advanced types in this group. These batteries are charged at a constant current at a particular cell terminal voltage. This voltage level varies among manufacturers, typically 4.1 or 4.2 volts/cell. A unique aspect to proper charging of Li Ion batteries is monitoring battery terminal voltage, and not charging voltage, while charging in current regulation mode. To perform this measurement, the supply must be effectively turned off during the measurement.

Some manufacturers do not recommend a float or trickle charge after charging. If trickle charging is desired, it should be done in a similar manner to the fast charge, but at a greatly reduced rate or .025C or lower, and terminated after a fixed time period. Listing 12 is an example of a fast charge algorithm for Li Ion batteries.

```

function fastCharge()
  static state=0 //store state information between calls
  //DA is the data analysis object
  //PS is the PowerSupply object
  //process general conditions that are state independent
  if DA.getTemperature() < 41F call terminateCharge()
  if DA.getTemperature() > 104F call terminateCharge()
  if DA.getDeltaT()>15F call terminateCharge()
  if DA.getDeltaTime() > 90 minutes call terminateCharge()

```



```

if DA.getVoltage() >= (CellVoltage * Cells) call terminateCharge()
select case (state)
case (0)
    PS.setCurrent(C_rate amps)
    state=1
case (1)
    PS.setCurrent(0) //turn off supply
    PS.getSample() //get a sample with supply off
    if DA.getVoltage(1)=Cells*(CellVoltage-50mv)
        PS.setVoltage(CellVoltage*Cells)
        state=2
    end if
case (2)
    if DA.getCurrent()<(50ma*Cells)
        terminateCharge()
    end if
end select
end function

```

Listing 12. Lithium Ion fast charging algorithm

Smart Charger Server

The smart charger server was also written in Visual Basic. The primary function of the server is to convert the TCP/IP packet data sent and received by a client application to a serial port data packet that can be interpreted or sent by the device. The server application also handles a number of other details that is relate to this primary function, and isolates these complexities from the client application. For example a client user simply types a command “setv 10 <enter>” in HyperTerminal to set the hardware to voltage regulation mode at 10 volts. This message is sent to the server at 1 or more characters at a time (determined by HyperTerminal) and is buffered by the application. After the message is fully entered the server decodes the message and determines the device command and parameters to be sent to the device. The 10 volt argument is converted into a 2 byte value determined by Equation 8.

$$y = desired_v \frac{0xFFFF}{20} \quad \text{Equation 8}$$

For this example the return value is 0x8000. This argument is converted into two bytes that are transmitted separately, along with an ASCII command prefix and a checksum byte suffix. This new packet is added to a que to wait for transmittal to the device. When prior operations have completed (possibly a previous command by the client, or another client that is accessing the device simultaneously), the packet is transmitted to the device via RS232 communication. The server waits for a response from the device, acknowledging the message or returning a request for data. If a response is not received in a configurable timeout period, the message is resent in a configurable number of retry attempts. When the acknowledge message or data message is received from the device, it is decoded and the device data is verified by checking a calculated checksum with the transmitted checksum. The raw data is extracted from the packet and reformatted as a numeric text string that the client application can read. For the HyperTerminal application, a response string is constructed to be displayed in the terminal window, such as “Voltage = nn.ddd volts”. The message to the smart charger application is a condensed version of the same message. If both client connections are active, then a

reply message is sent to both clients simultaneously. A reply message to the client is constructed with the type of reply message appended to the data, or a “not acknowledged” is sent to the client if there is no response from the device, if the checksum is bad, or if “not acknowledged” is sent directly from the device. Figure 24 shows the class diagram for the server application.

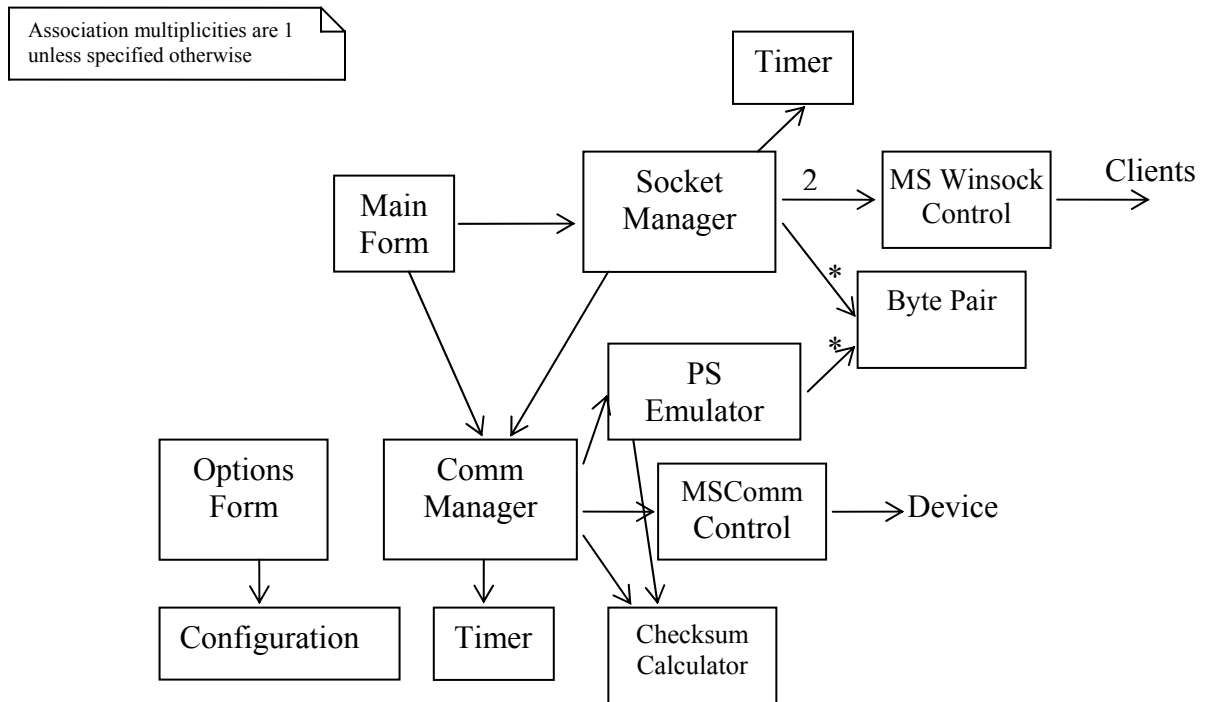


Figure 24. Server Application Class Diagram

Socket Manager Class Summary		
Data Members SetV:BytePair SetI:BytePair GetV:BytePair GetI:BytePair	Methods decodeRecDataApp() decodeRecDataTerm() ParseCommandApp() ParseCommandTerm()	Responsibilities <ul style="list-style-type: none"> • Decodes/Encodes client messages • Encodes/Decodes device messages • Drives MS Winsock control

The Socket manager is primarily responsible for encode and decoding of messages between the device and the client applications. Messages destined for the device are

passed to the CommManager class which maintains a queue of messages awaiting transmission. The encoding of device messages is given in Table 2. The Socket Manager class uses the custom data type BytePair to facilitate the conversion of raw data formatted in 16 bit integers and floating point strings

The Socket Manager class manipulates the Winsock Control events, including connection request, data arrival and close events. This class is also associated with a timer control that times out when there is no client activity within a certain time period, causing the TCP/IP port resource to be close and allow other connections. Under normal conditions the client notifies the server to close the connection, but if the application closes without sending a close notification, or if communication is lost, the server will eventually close the connection so the client may reconnect or another client may connect. Only one smart charger client, and one HyperTerminal client connection are allowed at the same time.

Packet Type	PC sends			Device Replies									notes
	Byte 0	1	2	Byte 0	1	2	3	4	5	6	7	8	
set Voltage	'V'	hb	lb	'V'	Hb	lb							Vmax=0xFFFF, 0V=0x0000
set Current	'I'	hb	lb	'I'	Hb	lb							Imax=0xFFFF, 0A=0x0000
set Control	'C'	0,1		'c'	0,1								0=local control, 1=pc control
set OutputOnOff	'O'	0,1		'd'	0,1								0=output off, 1=output on
get Data	'R'			'r'	Vhb	Vlb	Ihb	Ilb	T0hb	T0lb	T1hb	T1lb	device to send actual voltage, current, temperatures (V,I 16bit, T 10 bit)
get PWM	'P'			'p'	hb	lb							device replies with current 10 bit PWM value
get Status	'S'			's'	Mode	Control	errorCode						get mode, control, last error status
improptu error				'e'	errorCode								device detects error or warring condition and transmits error code without prompting

General Notes:

multi byte data types expressed in big endian form

Voltage and Current input/output are 16 bit values

10 bit values (such as temperature and pwm status) are right justified in 16 bit field

Table 2. PC-Device Communication Packet Protocol.

Comm Manager Class Summary		
Data Members packets:String Collection waitForDevice:Bool retryCount:integer checksum:ChecksumC alculator	Methods addPacket() extractPacket() sendPacket() responseVerification()	Responsibilities <ul style="list-style-type: none"> • Manages data transmission to/from device/emulator • Checksum verification • Drives MS Comm control

The Comm Manager class controls data flow to and from the device. The class maintains a queue of messages to be transmitted and enforces synchronization between transmitted and received messages. The Comm Manager directly manages the MS Comm control and its associated OnComm event. The Comm manager also provides a

link to the PSEmulator class that emulates not only the device, but also an assumed load on the device, such as a battery.

MS Comm Control

This is a control built into Visual Basic. The control has an input() method that returns a byte array of buffered data received on a serial port. The control also has an output() method that transmits data in a byte array to a device on the serial port. The control is set up from configuration data, such as port number and baud rate, from the Configuration class.

BytePair Class Summary		
Data Members HighByte:Byte LowByte:Byte	Methods floatValue() toString() RawValue() setScaleFactors()	Responsibilities <ul style="list-style-type: none"> • Implements data conversion

The byte pair class is an abstract data type that handles the bulk of the data conversion requirements of the server. The device accepts most arguments as two bytes, or 16 bit values that are a scaling of some maximum value (such as voltage scaled to 20 volts as in Equation 8). The class accepts arguments as two separate bytes, a 16 bit value, or a floating point value. This value is stored internally as two bytes, which are private data members of the class. The data can be accessed by a number of member calls, such as `y=bp_object.floatValue` that would calculate the floating point value based on the internal data and some scaling factor.

Checksum Calculator Class Summary		
Data Members none	Methods csValid() Value()	Responsibilities <ul style="list-style-type: none"> • Tests received packet for correct checksum. • Calculates checksum for packets to send

The checksum calculator contains methods for evaluating a received packet for proper checksum value, or to calculate a checksum value to append on a packet to be sent to the device. The checksum is a single byte that represents the 8 least significant bits of the sum of byte values for a particular packet. By comparing the transmitted value to one calculated at the receiver (both the server and the device calculate and decode checksum)

the data received can be verified as accurate, to some extent. The integrity of the data is important in the smart charger design because a load, such as a battery, may be damaged or explode if voltage or current commands are improperly interpreted by the device.

Battery connector

Perhaps the simplest part of the project may prove to be the hardest to achieve. Given the wide variety of batteries today, it is difficult to have a universal connector that will attach the battery to the charger. Some of the most common types of batteries, like AA type and D type, have standard battery holders types that can be used to charge a number of these batteries simultaneously as seen in Figure 25. Others may have two prongs sticking out, or even flat metal contacts as it is often seen in cellular telephone batteries. For those reasons, the team decided to fit the charging system with the most common types of connectors. Thus, some standard AAA, AA, C and D type connectors along with some alligator clips for the prongs, and a clamp system for the metal contacts, will be provided with the charger. This should accommodate most types of batteries.

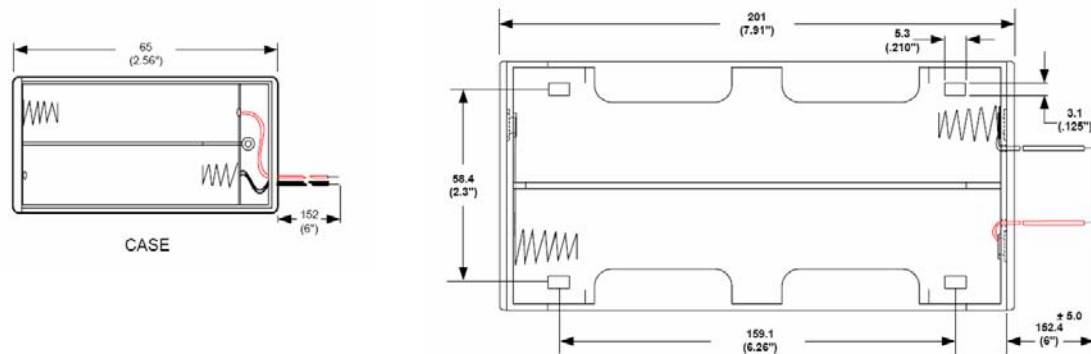


Figure 25. Example Battery Holders.

Overall Printed Circuit Board (PCB) Design:

The design is based on a two-board design. The control board contains the microcontroller, accessories, and various headers to peripheral components such as the LCD display and IR remote detector. Only low power analog and digital signals (5 volts or less, with the exception of low power RS-232 level shift circuit voltages) are transmitted and received on this board. The comprehensive circuit schematic for this board is shown in the Appendix as Figure A1. The preliminary PCB artwork for the system control board is shown in Figure A2.

The power board contains the circuits that implement the main switching power supply and microcontroller linear supply, along with the voltage and current sense circuits. The boards will be connected by an interconnecting cable that supplies power to the control board and transmits PWM signals and voltage/current sense signals between the boards.

Final Product Design

After integrating all of the components described above, the finished design product is envisioned to appear as shown in Figure 26.

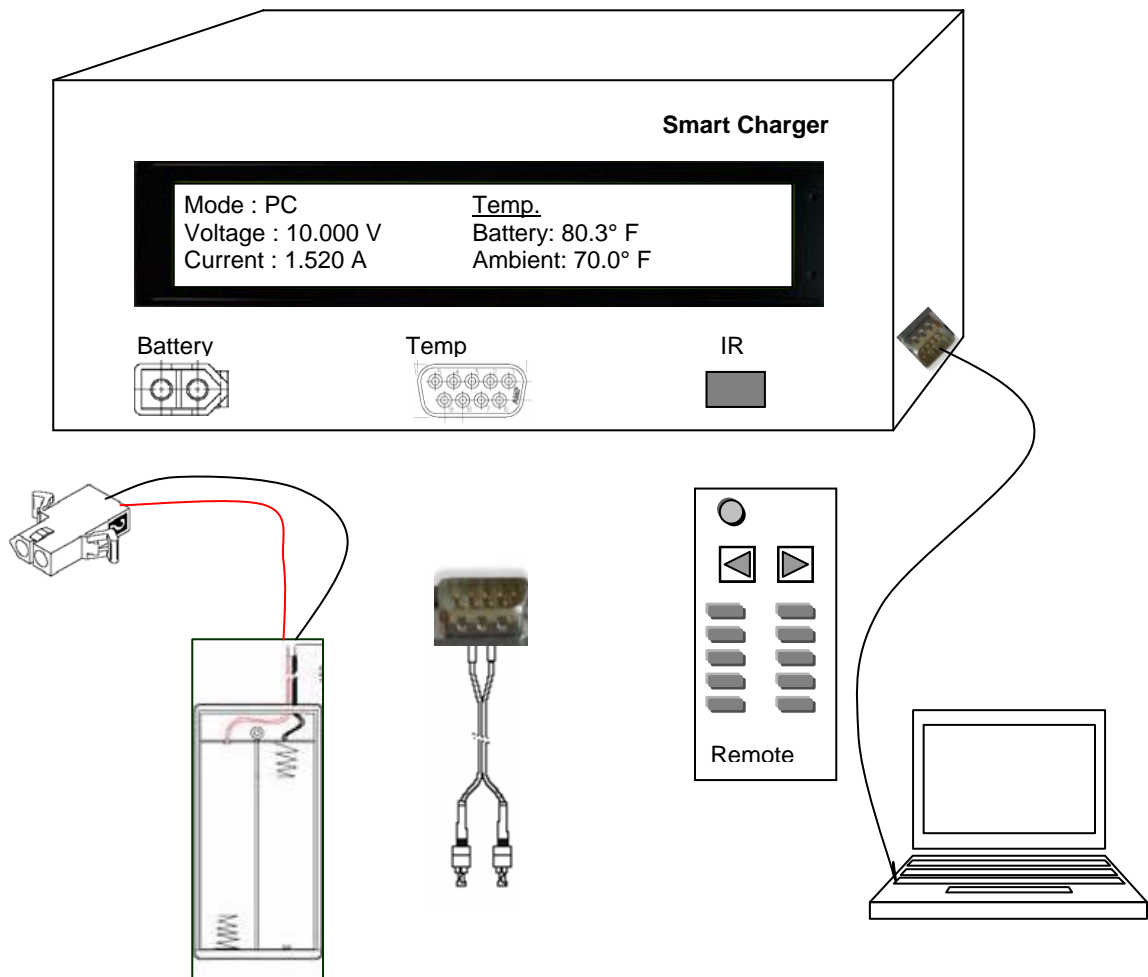


Figure 26. Finished Product Design Diagram.



Figure 27. Finished Product – Front View.

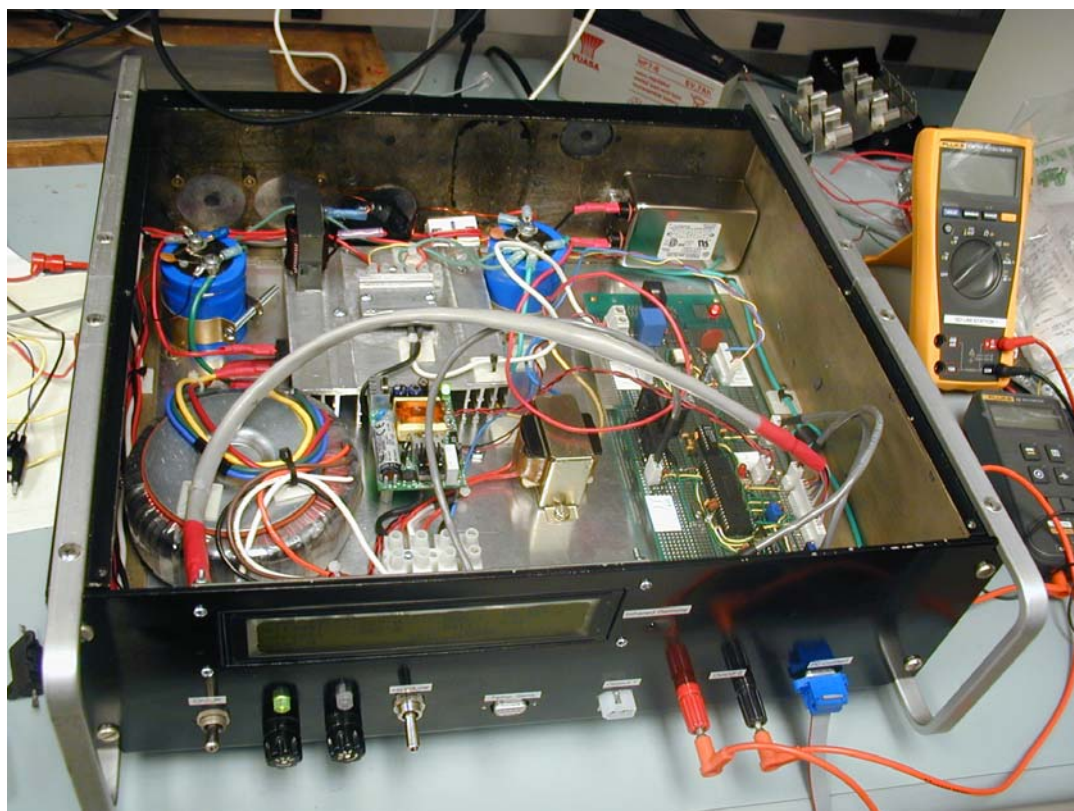


Figure 28. Finished Product – Top View.



Figure 29. Finished Product in Overall System.

Parts List

The finalized parts list shown in Table 3 includes all parts necessary to implement the Technical Design discussed above.

Table 3. Implementation Parts List.

Qty.	Refdes	Part Num.	Description
1	IC1	PIC18F452-I/P	PIC Microcontroller
1	IC2	MAX 232	RS 232 level converter
1	IC3	AD7705BN	Analog to digital converter 16 pin dip
1	IC4	MCP607-I/P	Microchip precision rail-to-rail op amp 8pin dip
1	IC5	TC1411NCPA	Microchip power mosfet driver 8pin dip
1	Q1	ECS-80-20-4	CRYSTAL 8.00MHZ 20PF HC-49/US
1	Q2	MP024S	CRYSTAL 2.4576MHZ HC-49, 32PF
2	R1	3296Y-1-103	Bourns 3296Y 10K pot .5W top adjust 3/8" SQ CERM SL MT
1	R2	3296Y-1-200	Bourns 3296Y 20 ohm pot .5W top adjust 3/8" SQ CERM SL MT
1	R3	ERD-S2TJ222V	2.2K 1/4W 5%
1	R4	ERO-S2PHF4703	470K 1/4 W 1% metal film
2	R5,R7	3296Y-1-203	Bourns 3296Y 20K pot .5W top adjust
1	R6	MFR-25FBF-10K0	10K 1/4W 1%
2	R8	CFR-25JB-1K0	1K 1/4W 5%
1		ERD-S2TJ470V	47ohm 1/4W 5%
1	Z1	LM4040AIZ-2.5	LM4040AIZ-2.5 precision 2.5V reference TO92 package
1		2-640379-4	CONN IC SOCKET 40POS DIP 15AU
2		2-640358-4	CONN IC SOCKET 16POS DIP 15AU
2		2-640463-4	CONN IC SOCKET 8POS DIP 15AU
2	C1,C2	1206CG200J9B200	20pf disk cap
1	C3	ECA-1EM100	10uF Electrolytic 25VDC
5	C4,C5,C6,C7,C8	ECA-1HM010	1uf Electrolytic
2	C9,C10	GRM0335C1E300JD01D	30pf disk cap
1		EEU-FC1E470	47uF
1	D1	1N4148	1N4148
3		LM34DZ-ND	IC SENSOR TEMP PREC FAHR TO-92
1		CFAH4004A-YYB-JP	CrystalFontz 40x4 LCD
1		GP1UM27XK	Sharp GP1UM28XK IR remote control receiver
1		351-43-0201	molex RC car battery connector plug
2		357-28-0201	molex RC car battery connector socket pin
4	rs232,temp,pb	22-23-2091	molex 9 pin .100 pin header
4		22-01-3097	molex 9 pin .100 connector
3	ICSP,LCD1	22-23-2061	molex 6 pin .100 pin header
2		22-01-3067	molex 6 pin .100 pin connector
2	LCD2	22-23-2041	molex 4 pin .100 pin header
2		22-01-3047	molex 4 pin .100 connector
2	IR	22-23-2031	molex 3 pin .100 pin header
2		22-01-3037	molex 3 pin .100 pin connector
100		08-50-0114	molex Crimp terminals for .100 housings
2		747905-2	9 pin D-sub Female

1		747904-2	9 pin D-sub Male
1		03-09-2022	molex .093 2 cir Plug 12A
5		03-09-1022	molex .093 2 cir Receptacle 12A
1		02-09-2103	molex .093 male terminals 14-20 crimp
1		02-09-1104	molex .093 female terminals 14-20 crimp
1		02-09-2118	molex .093 male terminals 18-22 crimp
1		BH48AAL	AA 8 cell battery holder
1		BH26CW	C 6 cell battery holder
1		BH24DL	D 4 cell battery holder
1		364-1042-ND	Power Line EMI Filter, 10A
1		IRFP244	HEX/MOS N-CH 250V 15A TO-247AC
3		10CTQ150-1-ND	DIODE SCHOTTKY 150V 10A TO-262
2		LM7805CT-ND	Voltage Regulator, 5V,1A
4		493-1079-ND	CAP 0.33uF, 35V Elec Radial
3		P11214-ND	CAP 27UF 25V ELECT FC RADIAL
1		688CKS035M	CAP 6800uF 35V Elect Radial
2		P11855-ND	CAP 2700UF 200V ELECT TS-UQ
1		182S12	Power Toroid Transformer 225W 24V@9.38A
1		235-1151-ND	Thermistor PTF 12 100HM@110C
1		TLP351-ND	IC IRED PHOTOCOUPLER 8-DIP
1		102-1052-ND	SENSOR CURRENT 10A 5V
1			Case
1			Miscellaneous

Testing Procedures: Verification and Validation

A vital part to a successful implementation of the entire SmartCharger design is the ability to test individual components and interfaces between components during the alternative design analysis, technical design, and especially the implementation phases. Beginning with the conception and detailed definition of this design, the team has agreed to modularize the design and test each component separately. After it has been verified that each component functions alone, the component will be interfaced with local subsystem components and eventually emerge as a functioning part of the entire design. The entire design must verify that the listed requirements specifications are met. This section will discuss the testing procedures that have already been performed and the more formal testing procedures that will be used when the hardware and software has been prototyped.

Hardware Testing:

Power Supply: Since the switch-mode power supply is the core of this design, the electrical engineers have built a prototype power supply in order to gain experience with these supplies, to determine the voltage and current outputs under varying parameters and PWM signals, and determine what is necessary to proceed with the power supply design. This prototype was first tested with digital multimeters and an oscilloscope. The pulse width modulation signal was driven by a pulse generator and varied with respect to frequency and duty ratio.

The final power supply design was tested to verify that it can output a voltage in the range of 0-20 V and a current in the range of 0-10 A. After taking samples across the entire voltage range, the results were plotted in Figure 30.

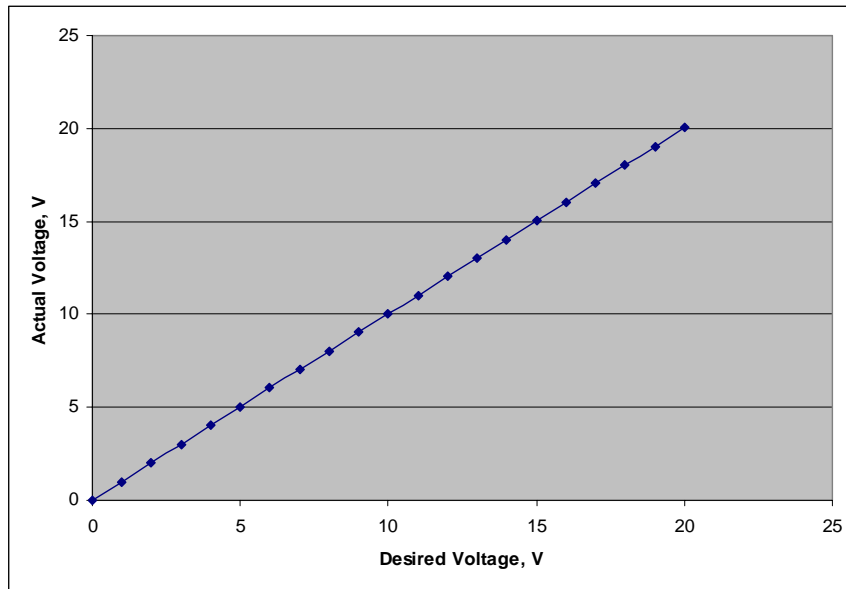


Figure 30. Voltage Output Verification.

Similar samples were taken across the entire current range and the results were plotted in Figure 31.

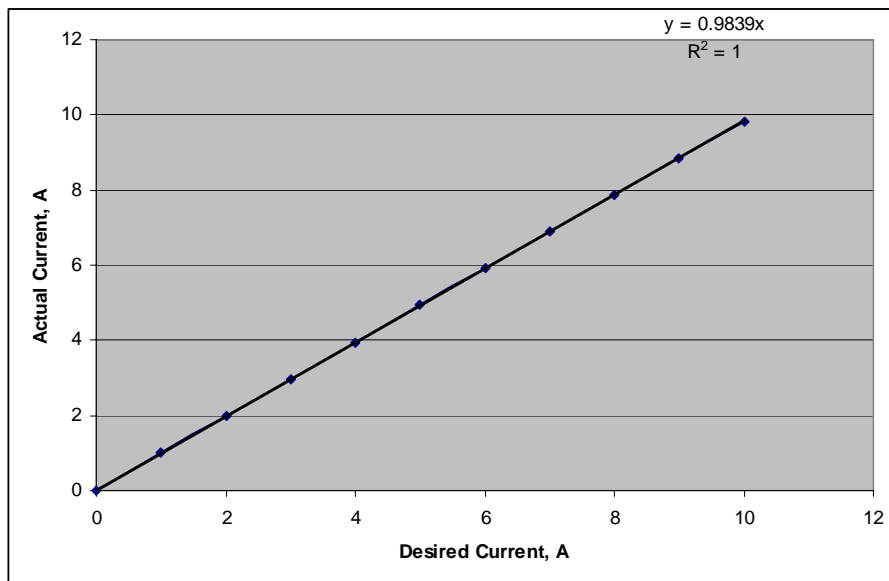


Figure 31. Current Output Verification

Microcontroller and Accessories: The microcontroller and control board accessories necessary for high-level feedback have been tested during the actual design phase to decide what can be feasibly implemented given the time, budget, and experience constraints. In addition, the group individually tested these logic components in a

breadboard circuit to compare the compatibility of interconnected devices and decide which devices should be purchased for the actual implementation.

Software Testing:

Microcontroller software: The microcontroller software will be tested with the Microchip MPLAB IDE and In-Circuit Debugger-2 (ICD-2). The microcontroller and various parts of the logic system can be easily tested with a development board, such as the LAB-X1 board, before it is implemented in the actual design.

The final logic board design will also be tested with the ICD-2 module to make sure that the voltage and current feedback signals are accurate to within ± 20 mV and ± 10 mA tolerances for error.

Visual Basic Custom PC application: The PC application will be tested and debugged with the Microsoft Visual Basic.NET environment. It will be tested on the common Microsoft Windows XP and Windows 2000 operating systems to ensure compatibility with Microsoft Windows users.

System Testing:

As each component passes the preliminary testing procedures, it will then be interfaced with the rest of the system on its board (power board and control / logic board). The hardware will be tested and troubleshot by hand with digital multimeters and an oscilloscope. This step is to ensure that good soldering methods were used and that the system is relatively free of excessive amounts of electrical noise.

After following the testing procedures discussed and validating that the system meets the LCD and infrared remote requirements, the entire design will meet the stated design requirements specifications.

Battery Charging:

Because the primary purpose of smart charger is charging batteries, testing and verification of the charging algorithms were an important aspect of the design. The team was able to obtain and charge three of the four major battery types listed in our design specifications. Because the LiIon battery types are so new, and somewhat rarer in use, we were unable to acquire a LiIon battery in the time we had for testing.

The charging algorithms are summarized as follow. For more detail on the charging algorithms, please refer to the chemistry class descriptions in the application design section of the document, and the code listing in the appendix. The algorithms were derived from various sources, with the major source being "Handbook of Batteries" (Linden). Note that one major advantage that smart charger has over most other battery chargers is that the charge algorithms are completely defined in software as a visual basic class module. Support for new chemistries or better algorithms for existing chemistries simply require coding the algorithm, which is relatively simple by using the existing charge methods as a template.

- Lead Acid 3 stage modified constant voltage fast charge algorithm
 - Stage 1. Constant current charge at 'Charge Rate' until voltage drops to 2.39V/cell

- Stage 2. Constant voltage charge at 2.39V/cell level until current drops below .2C
 - Stage 3. follow by a trickle charge at 2.21V/cell
- Sealed Lead Acid 3 stage modified constant voltage fast charge algorithm
 - Stage 1. Constant current charge at 'Charge Rate' until voltage drops to 2.45V/cell
 - Stage 2. Constant voltage charge at 2.45V/cell level until current drops below .2C
 - Stage 3. follow by a trickle charge at 2.25V/cell
- NiCd 3 stage -dVdt fast charge algorithm
 - Stage 1. 1C charge for one minute, allow data to stabilize
 - Stage 2. continue 1C charge terminated with -dVdt (primary) or 1.8F (secondary) dTdt
 - Stage 3. follow by a maintenance (trickle) charge at .025C rate
- NiMh 4 stage fast charge algorithm.
 - Stage 1. 1C charge for one minute, allow data to stabilize
 - Stage 2. continue 1C charge terminated with 1.8F dTdt (primary) or -dVdt (secondary) sensing
 - Stage 3. follow by a .1C topping charge for 45 minutes
 - Stage 4. follow by a maintenance charge at .025C rate
- LiIon 3 stage fast charge algorithm
 - Stage 1. 1C charge terminated when open circuit cell voltage = 4.1V or 4.2V (depending on type)
 - Stage 2. follow by constant voltage charge at the above cell voltage until I drops to < 50ma/cell
 - Stage 3. follow by a maintenance (trickle) charge at .025C rate

Notes:

- Sealed lead acid and lead acid algorithms are basically the same, but with different set points. Sealed lead acid can also be charged at a faster rate.
- Linden describes a -20 mV/cell deltaV charge termination, but dVdt=0 is considered better by other sources and was implemented instead. That's basically what I'm doing (as soon as dVdt is a negative value). Implemented by taking the last minute of samples and performing a linear least squares regression on the data. The slope of that line is used as dVdt, and scaled to mV/min. dTdt is the temperature derivative implemented in a similar manner.
- Linden says the best termination for NiMh is dTdt (temperature) so that is the primary termination method, although a negative -dVdt will also trigger next stage.
- Linden says little about LiIon as far as charging procedures and methods. Algorithm taken from an ATMEL microcontroller application note. Requires turning off the output of the PS during voltage measurements during stage 1.
- Algorithms terminate if fault conditions are detected, including max temp, min temp, delta temp, max cell voltage, max current, excessive charge time (except while in trickle stage)

- Currently all algorithms are evaluated every second (adjustable)

The first battery we attempted to charge was a 4AH sealed lead acid (SLA) 6 cell battery shown in Figure 32. This battery has a nominal 12 volt output.



Figure 32. Sealed lead acid battery

The battery was first discharged by a 2 Ω resistive load until the battery failed to deliver an appreciable power to the load. The charging application was configured by selecting sealed lead acid in the chemistry selection drop down list, and updating the mAh capacity to 4000mAh and number of cells to 6. We chose to charge the battery at a fairly high rate of .5C which limits the charge current to 2A during the charge cycle. The measurements are automatically logged in a comma separated value (.csv) file, and upon completion of charging the file was imported into excel. The data was plotted in excel, and this plot can be seen in Figure 33. The charge cycle was completed over approximately a 3 hour period, and a data set of 12376 samples points (with a sample composed of time stamp, voltage, current, battery temperature and battery temperature) were logged. The main charge completed in approximately 70 minutes, and then a constant voltage trickle charge was applied. The plots show a slight elevation in temperature during the rapid charge phase, with a subsequent cool down during the trickle charge phase.

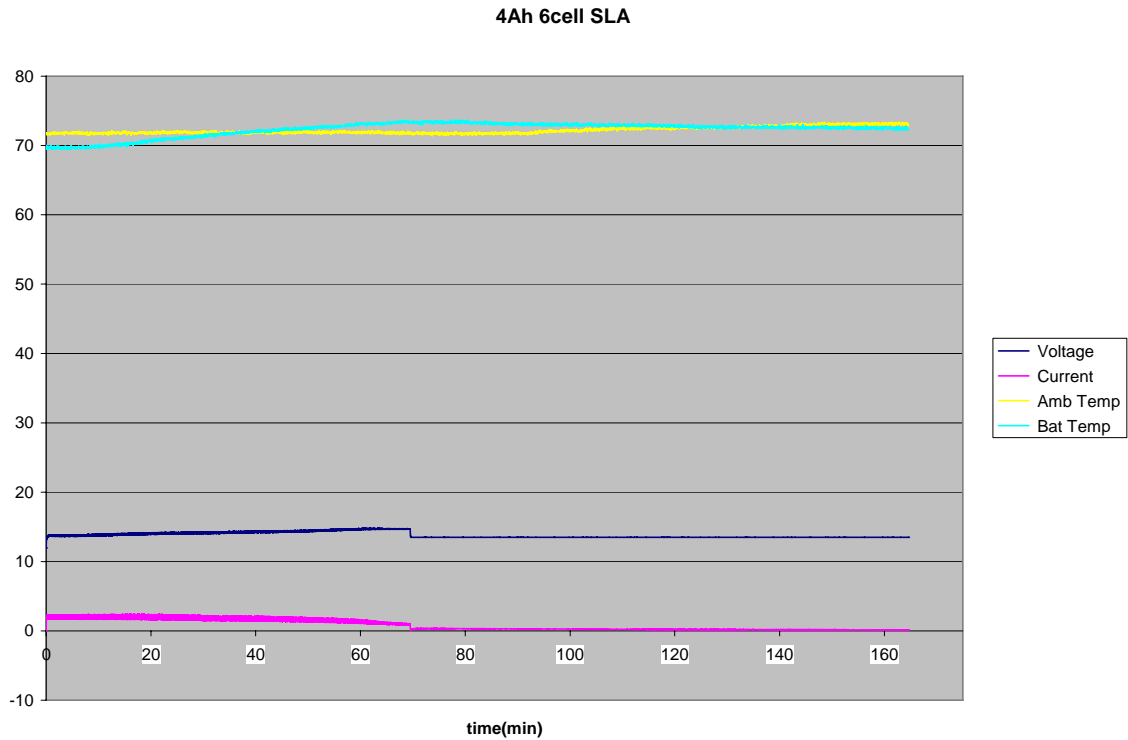


Figure 33. Sealed lead acid overall charge plot

The voltage and current were plotted separately in Figure 34. The graph indicated fairly significant steady state voltage and current fluctuation during the constant current 2 amp charge. We also discovered that the transition to constant voltage trickle charge at 13.5 volts occurred prematurely in the charge cycle. This was due to an inconsistent sample that indicated a current less than .4 amps, causing a transition to trickle charge phase. Upon some investigation some time latter, we discovered poor filtering response, which was traced to a loose connection on the output filter capacitor. This would explain the poor regulation we observed during this charge, as well as a possible cause for the inconsistent sample. An improvement to the algorithm to prevent a false transitions in future charges due to a single inconsistent sample was to transition on an average current value (over the past 60 seconds), instead of exclusively based on the last sample.

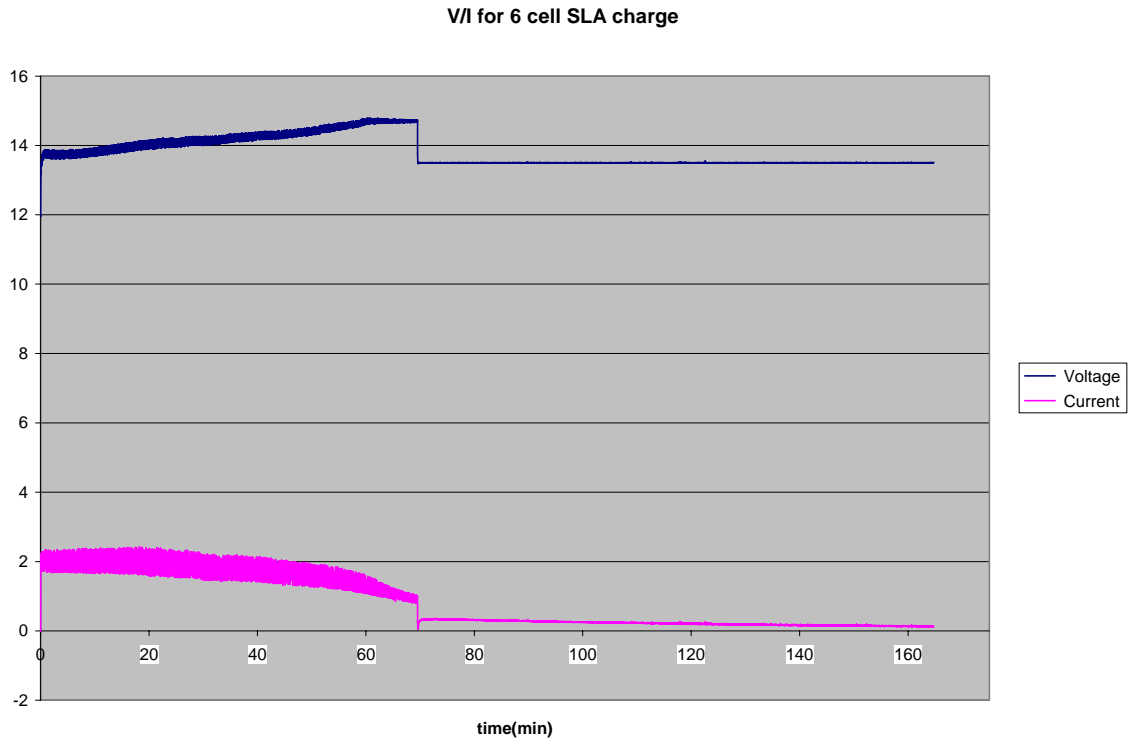


Figure 34. Voltage and current for sealed lead acid charge

Our next battery is an 8 cell Nickel Cadmium (NiCd) battery pack shown in Figure 35.



Figure 35. 9.6V NiCd Battery

The pack is not rated, but with some internet research we learned that the packs are rated at approximately 750 maH. This battery pack is designed for an RC toy, and has a standard “Tamiya” connector common to many RC toys. We connected the pack to the smart charger by an interface connector harness with one end that plugs into the battery pack and the other end that plugs into the smart charger front panel connector. The battery temperature sensor was taped to the center of the battery pack, and the ambient temperature sensor was placed in close proximity to the battery. To test the performance of smart charger and demonstrate its advance abilities, we chose to charge at

a 1.33C rate, and therefore charge the battery in less than 1 hour. The application was configured for the NiCd battery, and the battery parameters were entered, and charging commenced. The charge was completed in approximately 42 minutes. During the charging process the application collected and stored 2581 samples, which are plotted in Figure 36. The algorithm performed flawlessly, and the battery was charged quickly without overcharging or excessive heating.

The primary charge termination is $-dV/dt$ which occurs as the battery voltage peaks and begins to drop. As seen in Figure 37, the charge was terminated at the optimal time. Most commercially available fast chargers use a $-\Delta V/dt$ that requires the battery voltage to drop below a maximum value by some value, although by that point the battery is partially overcharged. The smart charger $-dV/dt$ algorithm minimizes the overcharge by terminating the charge before a $-\Delta V/dt$ termination would, and could therefore be considered superior to such chargers in that respect.

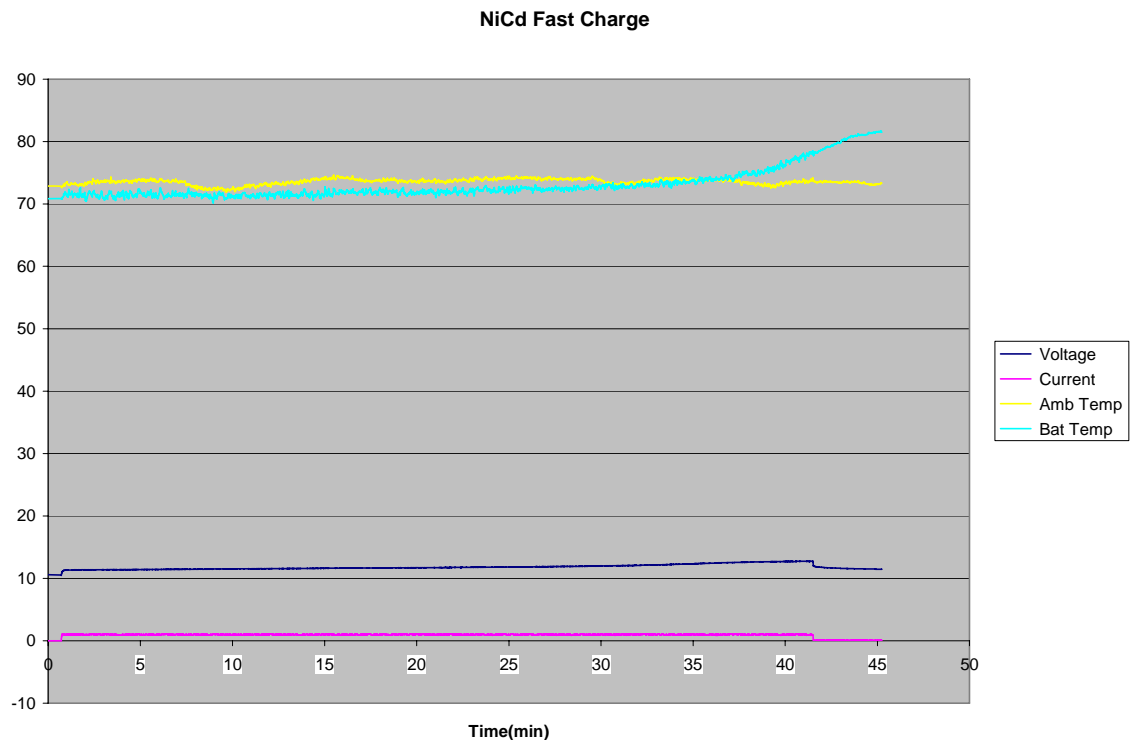


Figure 36. NiCd fast charge

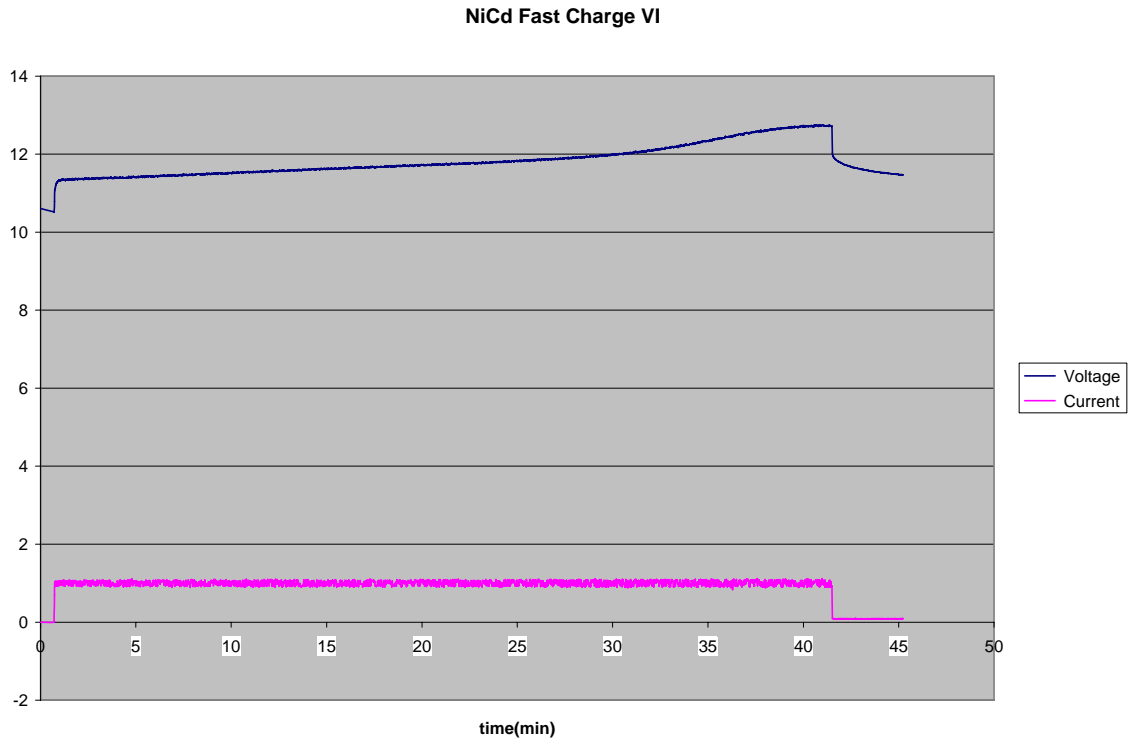


Figure 37. NiCd Charge voltage and current

Our third battery to charge was 4 1200mAH Nickel Metal Hydride (NiMh) AA size cells shown in Figure 38. The cells were inserted into the AA size battery holder that we built for charging individual cells. The battery temperature was taped to the side junction of two cells in the charge holder, and the ambient temperature probe was placed close to the battery. This holder has a harness that plugs directly into the front panel socket connection.

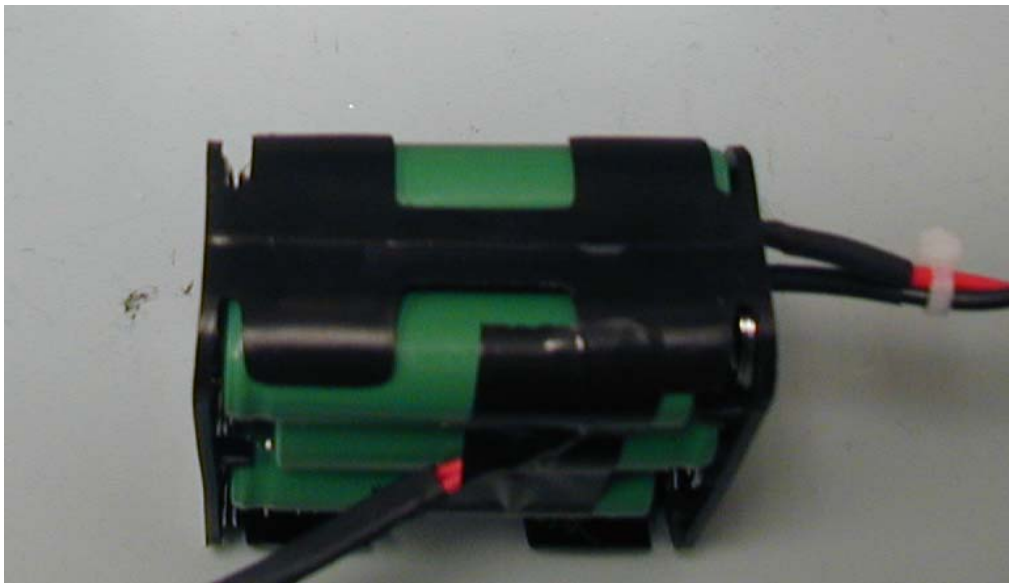


Figure 38. 4 NiMh AA cells in charging holder

The application was configured for the NiMh cells by entering the configuration data into the charging application form. The application commenced charging, and logged 1633 data samples over a 28 minute period that corresponds to the plot of Figure 39.

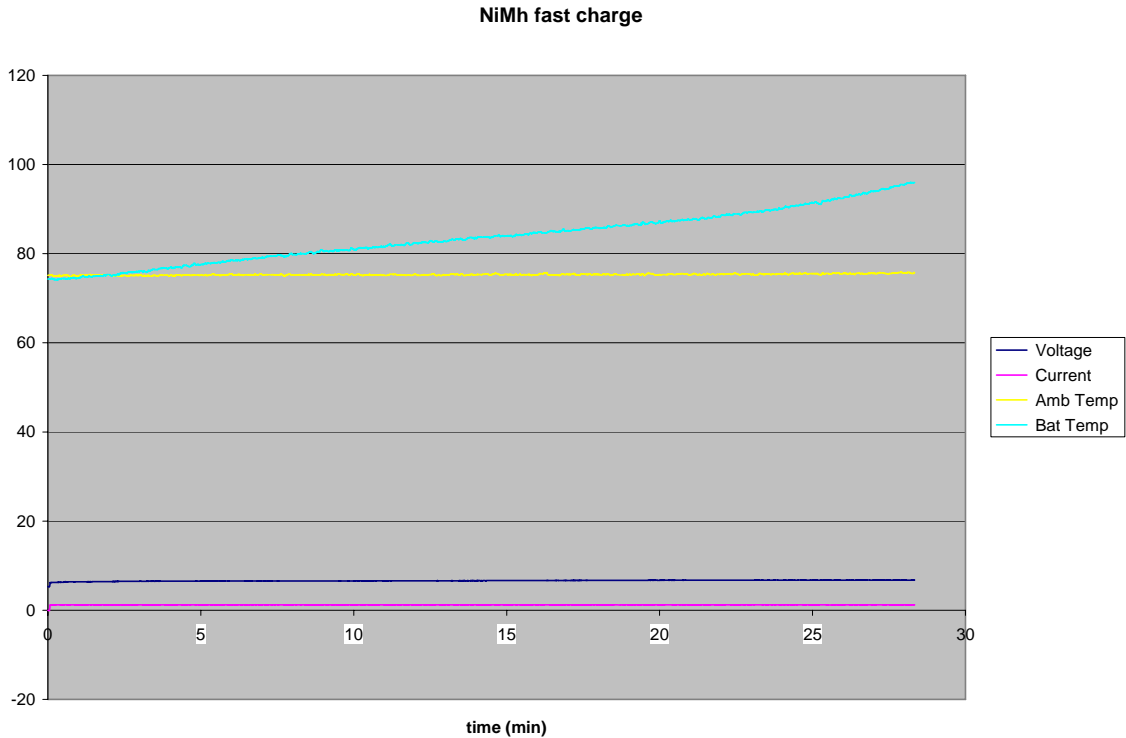


Figure 39. NiMh fast charge

The primary charge termination for the NiMh is dT/dt , or rate of change in temperature per unit time. The set point rate of change was $1.8^{\circ}\text{F}/\text{min}$, which was never achieved during the charge. The charge was abruptly terminated by tripping a global difference in temperature check set at 20°F average differential temperature averaged over a 1 minute period. It was not clear if this was a normal response to charging, but it did demonstrate that charging would terminate on other conditions in addition to the expected triggers given in the charging algorithm.

Power Supply Efficiency

The power supply was connected to a 2 Ω load, and AC voltage and current meters were attached to the input. The RMS power input to the supply is $V \cdot I$ using the true RMS fluke meters. The output power is the DC voltage multiplied by the DC current. The efficiency is calculated by Equation 9.

$$E = \frac{DCpower}{ACpower - Standbypower} \quad \text{Equation 9}$$

The results of the efficiency test is given in Table 4.

Table 4. Efficiency Test.

	AC Volts	AC Amps	AC Power (Watts)	DC Volts	DC Amps	DC Power (Watts)	Efficiency
Standby	122.5	.106	12.98	0	0	0	
50% load	122.3	.873	106.77	10	5.1	51	54.4%
82 % load	122.5	2.51	307.6	18.17	9.05	164.43	55.8%

Financial Budget

The team's labor budget has been set by the Electrical and Computer Engineering Department at \$10.00 per hour for each student for 15 weeks. Table 5 shows the initial estimated time/cost for the project and the actual time/cost for the project. The initial time turned out to be too low and twice amount of time was needed to complete the project. When working on the project, many unknown problems appeared like noise problem and getting the gate driver to work properly. The estimated material cost is shown in Table 6. Most of the items did not change during the building of the project. Only the current sensor resistor prove to be inadequate and a chip was need to do the current sensing. However, more parts were needed to get the project to work, like a gate driver and case. During the building of the project, parts were found at home that was use in the project. Since these parts were not bought or have full labels on them, they were grouped into a miscellaneous group and assigned a approximated value.

Table 5. Labor Cost.

Design Team Member	Initial Estimated Cost	Revised Estimated Cost	Actual Cost
Claudiu Bouruc	\$1,500.00	\$2,500.00	\$2,500.00
Eric Boyer	\$1,500.00	\$2,500.00	\$2,500.00
Mike Petrak	\$1,500.00	\$2,500.00	\$2,500.00
Steven Savage	\$1,500.00	\$2,500.00	\$2,500.00
Total Cost	\$6,000.00	\$10,000.00	\$10,000.00

Table 6. Material Cost.

Qty.	Part Num.	Description	Initial Cost	Revised Cost	Actual Cost
1	PIC18F452-I/P	PIC Microcontroller	\$9.38	\$9.38	\$9.38
1	MAX 232	RS 232 level converter	\$3.31	\$3.31	\$3.31
1	AD7705BN	Analog to digital converter 16 pin dip	\$9.70	\$9.70	\$9.70
1	MCP607-I/P	Microchip precision rail-to-rail op amp 8pin dip	\$1.33	\$1.33	\$1.33
1	TC1411NCPA	Microchip power mosfet driver 8pin dip	\$0.95	\$0.95	\$0.95
1	ECS-80-20-4	CRYSTAL 8.00MHZ 20PF HC-49/US	\$0.58	\$0.58	\$0.58
1	MP024S	CRYSTAL 2.4576MHZ HC-49, 32PF	\$2.08	\$2.08	\$2.08
2	3296Y-1-103	Bourns 3296Y 10K pot .5W top adjust 3/8" SQ CERM SL MT	\$5.00	\$5.00	\$5.00
1	3296Y-1-200	Bourns 3296Y 20 ohm pot .5W top adjust 3/8" SQ CERM SL MT	\$2.50	\$2.50	\$2.50
1	ERD-S2TJ222V	2.2K 1/4W 5%	\$0.07	\$0.07	\$0.07
1	ERO-S2PHF4703	470K 1/4 W 1% metal film	\$0.17	\$0.17	\$0.17
2	3296Y-1-203	Bourns 3296Y 20K pot .5W top adjust	\$5.00	\$5.00	\$5.00
1	MFR-25FBB-10K0	10K 1/4W 1%	\$0.11	\$0.11	\$0.11
2	CFR-25JB-1K0	1K 1/4W 5%	\$0.13	\$0.13	\$0.13

1	ERD-S2TJ470V	47ohm 1/4W 5%	\$0.07	\$0.07	\$0.07
1	LM4040AIZ-2.5	LM4040AIZ-2.5 precision 2.5V reference TO92 package	\$2.39	\$2.39	\$2.39
1	2-640379-4	CONN IC SOCKET 40POS DIP 15AU	\$1.80	\$1.80	\$1.80
2	2-640358-4	CONN IC SOCKET 16POS DIP 15AU	\$1.74	\$1.74	\$1.74
2	2-640463-4	CONN IC SOCKET 8POS DIP 15AU	\$1.14	\$1.14	\$1.14
2	1206CG200J9B200	20pf disk cap	\$0.19	\$0.19	\$0.19
1	ECA-1EM100	10uF Electrolytic 25VDC	\$0.21	\$0.21	\$0.21
5	ECA-1HM010	1uf Electrolytic	\$1.05	\$1.05	\$1.05
2	GRM0335C1E300JD01D	30pf disk cap	\$0.11	\$0.11	\$0.11
1	EEU-FC1E470	47uF	\$0.28	\$0.28	\$0.28
1	1N4148	1N4148	\$0.07	\$0.07	\$0.07
3	LM34DZ-ND	IC SENSOR TEMP PREC FAHR TO-92	\$6.99	\$6.99	\$6.99
1	CFAH4004A-YYB-JP	CrystalFontz 40x4 LCD	\$48.70	\$48.70	\$48.70
1	GP1UM27XK	Sharp GP1UM28XK IR remote control receiver	\$1.33	\$1.33	\$1.33
1	351-43-0201	molex RC car battery connector plug	\$0.44	\$0.44	\$0.44
2	357-28-0201	molex RC car battery connector socket pin	\$0.20	\$0.20	\$0.20
4	22-23-2091	molex 9 pin .100 pin header	\$3.48	\$3.48	\$3.48
4	22-01-3097	molex 9 pin .100 connector	\$2.68	\$2.68	\$2.68
3	22-23-2061	molex 6 pin .100 pin header	\$1.80	\$1.80	\$1.80
2	22-01-3067	molex 6 pin .100 pin connector	\$0.96	\$0.96	\$0.96
2	22-23-2041	molex 4 pin .100 pin header	\$0.94	\$0.94	\$0.94
2	22-01-3047	molex 4 pin .100 connector	\$0.70	\$0.70	\$0.70
2	22-23-2031	molex 3 pin .100 pin header	\$0.78	\$0.78	\$0.78
2	22-01-3037	molex 3 pin .100 pin connector	\$0.58	\$0.58	\$0.58
100	08-50-0114	molex Crimp terminals for .100 housings	\$5.34	\$5.34	\$5.34
2	747905-2	9 pin D-sub Female	\$4.94	\$4.94	\$4.94
1	747904-2	9 pin D-sub Male	\$1.82	\$1.82	\$1.82
1	03-09-2022	molex .093 2 cir Plug 12A	\$0.33	\$0.33	\$0.33
5	03-09-1022	molex .093 2 cir Receptacle 12A	\$1.80	\$1.80	\$1.80
1	02-09-2103	molex .093 male terminals 14-20 crimp	\$0.79	\$0.79	\$0.79
1	02-09-1104	molex .093 female terminals 14-20 crimp	\$0.78	\$0.78	\$0.78
1	02-09-2118	molex .093 male terminals 18-22 crimp	\$0.81	\$0.81	\$0.81
1	BH48AAL	AA 8 cell battery holder	\$1.46	\$1.46	\$1.46
1	BH26CW	C 6 cell battery holder	\$1.60	\$1.60	\$1.60
1	BH24DL	D 4 cell battery holder	\$1.71	\$1.71	\$1.71
1	364-1042-ND	Power Line EMI Filter, 10A	\$18.53	\$18.53	\$18.53
1	12FR010	RES CURRENT SENSE .010 OHM 2W	\$1.56	\$0.00	\$0.00
1	IRFP244	HEX/MOS N-CH 250V 15A TO-247AC	\$3.16	\$3.16	\$3.16
3	10CTQ150-1-ND	DIODE SCHOTTKY 150V 10A TO-262	\$2.64	\$2.64	\$2.64
2	LM7805CT-ND	Voltage Regulator, 5V,1A	\$1.06	\$1.06	\$1.06
4	493-1079-ND	CAP 0.33uF, 35V Elec Radial	\$0.92	\$0.92	\$0.92
3	P11214-ND	CAP 27UF 25V ELECT FC RADIAL	\$0.84	\$0.84	\$0.84
1	688CKS035M	CAP 6800uF 35V Elect Radial	\$10.00	\$10.00	\$10.00
2	P11855-ND	CAP 2700UF 200V ELECT TS-UQ	\$16.24	\$16.24	\$16.24
1	182S12	Power Torois Transormer 225W 24V@9.38A	\$54.66	\$54.66	\$54.66
1	235-1151-ND	Thermistor PTF 12 100HM@110C	\$0.00	\$2.78	\$2.78
1	TLP351-ND	IC IRED PHOTOCOUPLER 8-DIP	\$0.00	\$1.50	\$1.50
1	102-1052-ND	SENSOR CURRENT 10A 5V	\$0.00	\$20.75	\$20.75

1		Case	\$0.00	\$20.00	\$20.00
1		Miscellaneous	\$0.00	\$30.00	\$30.00
			\$249.92	\$323.40	\$323.40

Team Funding

The team's material funding has been set at \$75 per design team member. With four design team members, this yields a total material funding of \$300 for this design project.

Project Schedule

The Smart Charger team has developed a strategy for completing a successful design implementation during the Spring 2005 semester. This strategy is shown in the form of an Implementation Gantt Chart. Table 7 shows the detailed tasks and description and Figure 40 shows the implementation schedule in a timeline format. Since the design and implementation gantt chart were very similar, the implementation gantt chart turned out to be the same as our design chart. The actual was different due to the first demonstration being moved up to April 9. This movement changed the schedule for the System Integration section. The days of assembly and testing for the final project were combined into one week in order to meet the April 9 deadline. All other schedules stayed the same due to their complexity and needed the whole time assigned to them. Overall, the project went as planned with each task being completed in the given time.

Table 7. Design and Implementation Gantt Chart Tasks and Description.

Task Name	Duration	Start	Finish	Predecessors	Resource Names
Revise Gantt Chart	15 days	Tue 1/18/05	Mon 2/7/05		
<input checked="" type="checkbox"/> Implement Project Design	76 days	Tue 1/18/05	Sat 4/30/05		
<input checked="" type="checkbox"/> Hardware Implementation	44 days	Tue 1/18/05	Fri 3/18/05		
Breadboard Components	2 wks	Tue 1/18/05	Mon 1/31/05		
Layout and Generate Control Board	2 wks	Tue 2/1/05	Mon 2/14/05		Steven Savage, Mike Petrak
Layout and Generate Power Electronics Board	2 wks	Tue 2/1/05	Mon 2/14/05		Eric Boyer, Claudiu Bouruc
Assemble Hardware	1 wk	Mon 2/21/05	Fri 2/25/05	6	
Test Hardware	2 wks	Mon 2/28/05	Fri 3/11/05	7	
Revise Hardware	2 wks	Mon 2/28/05	Fri 3/11/05	7	
Demonstrate Hardware	1 wk	Mon 3/14/05	Fri 3/18/05	8,9	
SDC & FA Hardware Approval	0 days	Fri 3/18/05	Fri 3/18/05	10	
<input checked="" type="checkbox"/> Software Implementation	55 days	Tue 1/18/05	Mon 4/4/05		
Code class modules to implement power supply	1 wk	Tue 1/18/05	Mon 1/24/05		Steven Savage
Create main power supply form	1 wk	Tue 1/25/05	Mon 1/31/05	13	Steven Savage
Code charger class modules to implement charger	1 wk	Tue 2/1/05	Mon 2/7/05	14	Steven Savage
Create charger form	1 wk	Tue 2/8/05	Mon 2/14/05	15	Steven Savage
Code PWM function, PID loop, external and internal ADC interf	1 wk	Tue 2/15/05	Mon 2/21/05	16	Mike Petrak
Code infrared remote, temperature sensing, RS-232 interface	1 wk	Tue 2/22/05	Mon 2/28/05	17	Mike Petrak
Code LCD interface	1 wk	Tue 3/1/05	Mon 3/7/05	18	Mike Petrak
Code mainline function	1 wk	Tue 3/8/05	Mon 3/14/05	19	Mike Petrak
Test Software	2 wks	Tue 3/15/05	Mon 3/28/05	20	Steven Savage, Eric Boyer, Mike Petrak
Revise Software	2 wks	Tue 1/25/05	Mon 2/7/05	13	Steven Savage
Demonstrate Software	1 wk	Tue 3/29/05	Mon 4/4/05	21,22	
SDC & FA Software Approval	0 days	Mon 4/4/05	Mon 4/4/05	23	
<input checked="" type="checkbox"/> System Integration	21 days	Tue 4/5/05	Sat 4/30/05	11,24	
Assemble Complete System	1 wk	Tue 4/5/05	Sat 4/9/05		
Test Complete System	1 wk	Mon 4/11/05	Fri 4/15/05	26	Claudiu Bouruc, Eric Boyer, Mike Petrak, Steven Savage
Revise Complete System	1 wk	Mon 4/11/05	Fri 4/15/05	26	Claudiu Bouruc, Eric Boyer, Mike Petrak, Steven Savage
Demonstration of Complete System	0 days	Fri 4/15/05	Fri 4/15/05	27,28	Claudiu Bouruc, Eric Boyer, Mike Petrak, Steven Savage
Advisory Council Oral Presentation	0 days	Fri 4/15/05	Fri 4/15/05	29	
Demonstration w/deduction	0 days	Sat 4/30/05	Sat 4/30/05	29	
Develop Final Report	1 day	Sun 4/17/05	Sun 4/17/05	29	

Gantt Chart

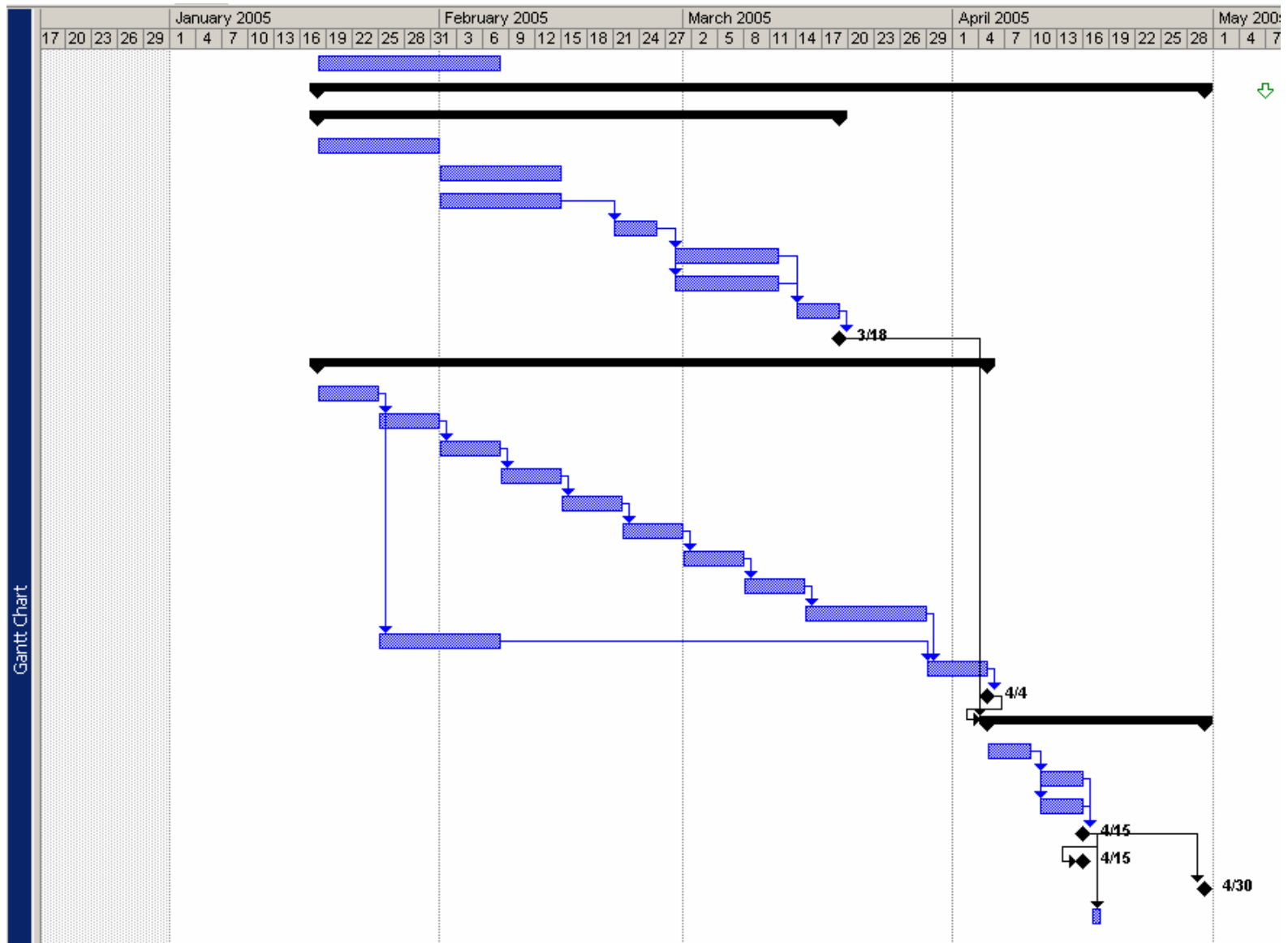


Figure 40. Implementation Gantt Chart Timeline View.

Table 8. Actual Gantt Chart Tasks and Description

Gantt Chart	Task Name	Duration	Start	Finish	Predecessors	Resource Names
	Revise Gantt Chart	15 days	Tue 1/18/05	Mon 2/7/05		
	<input checked="" type="checkbox"/> Implement Project Design	76 days	Tue 1/18/05	Sat 4/30/05		
	<input checked="" type="checkbox"/> Hardware Implementation	44 days	Tue 1/18/05	Fri 3/18/05		
	Breadboard Components	2 wks	Tue 1/18/05	Mon 1/31/05		
	Layout and Generate Control Board	2 wks	Tue 2/1/05	Mon 2/14/05		Steven Savage, Mike Petrak
	Layout and Generate Power Electronics Board	2 wks	Tue 2/1/05	Mon 2/14/05		Eric Boyer, Claudiu Bouruc
	Assemble Hardware	1 wk	Mon 2/21/05	Fri 2/25/05	6	
	Test Hardware	2 wks	Mon 2/28/05	Fri 3/11/05	7	
	Revise Hardware	2 wks	Mon 2/28/05	Fri 3/11/05	7	
	Demonstrate Hardware	1 wk	Mon 3/14/05	Fri 3/18/05	8,9	
	SDC & FA Hardware Approval	0 days	Fri 3/18/05	Fri 3/18/05	10	
	<input checked="" type="checkbox"/> Software Implementation	55 days	Tue 1/18/05	Mon 4/4/05		
	Code class modules to implement power supply	1 wk	Tue 1/18/05	Mon 1/24/05		Steven Savage
	Create main power supply form	1 wk	Tue 1/25/05	Mon 1/31/05	13	Steven Savage
	Code charger class modules to implement charger	1 wk	Tue 2/1/05	Mon 2/7/05	14	Steven Savage
	Create charger form	1 wk	Tue 2/8/05	Mon 2/14/05	15	Steven Savage
	Code PWM function, PI loop, external and internal ADC interfaces	1 wk	Tue 2/15/05	Mon 2/21/05	16	Mike Petrak
	Code user interface, infrared remote, temperature sensing, RS-232 interface	1 wk	Tue 2/22/05	Mon 2/28/05	17	Mike Petrak
	Code LCD interface	1 wk	Tue 3/1/05	Mon 3/7/05	18	Mike Petrak
	Code mainline function	1 wk	Tue 3/8/05	Mon 3/14/05	19	Mike Petrak
	Test Software	2 wks	Tue 3/15/05	Mon 3/28/05	20	Steven Savage, Eric Boyer, Mike Petrak
	Revise Software	2 wks	Tue 1/25/05	Mon 2/7/05	13	Steven Savage
	Demonstrate Software	1 wk	Tue 3/29/05	Mon 4/4/05	21,22	
	SDC & FA Software Approval	0 days	Mon 4/4/05	Mon 4/4/05	23	
	<input checked="" type="checkbox"/> System Integration	27 days	Mon 3/28/05	Sat 4/30/05		
	Assemble Complete System	1 wk	Mon 3/28/05	Fri 4/1/05		
	Test Complete System	1 wk	Mon 4/4/05	Fri 4/8/05	26	Claudiu Bouruc, Eric Boyer, Mike Petrak, Steven Savage
	Revise Complete System	1 wk	Mon 4/4/05	Fri 4/8/05	26	Claudiu Bouruc, Eric Boyer, Mike Petrak, Steven Savage
	Demonstration of Complete System	0 days	Sat 4/9/05	Sat 4/9/05	27,28	Claudiu Bouruc, Eric Boyer, Mike Petrak, Steven Savage
	Advisory Council Oral Presentation	0 days	Mon 4/11/05	Mon 4/11/05	29	
	Demonstration w/deduction	0 days	Sat 4/30/05	Sat 4/30/05	29	
	Develop Final Report	1 day	Mon 5/2/05	Mon 5/2/05	29	

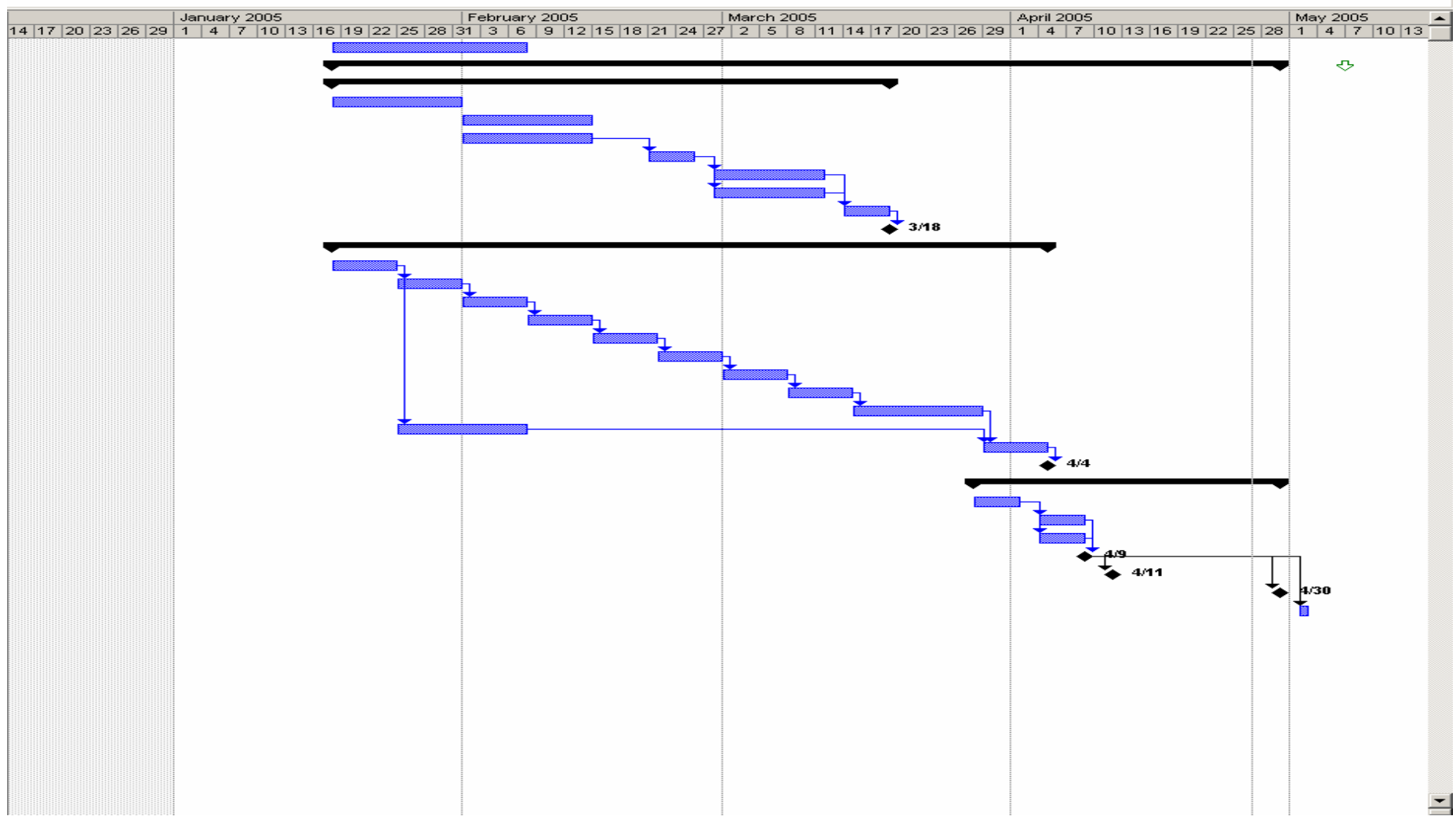


Figure 41. Actual Gantt Chart Timeline View

Design Team Information

Claudiu Bouruc

Electrical Engineer

Eric Boyer

Electrical Engineer

Mike Petrak

Computer Engineer

Steven Savage

Computer Engineer

Team Leader

Conclusions and Recommendations

The design presented above was inspired by today's need of a universal battery charger. The Smart Charger is intended to provide charging for the most common type of rechargeable batteries. It is designed to be a very powerful tool for the knowledgeable engineer or technician.

The ability of the charger to output real time graphs with the current and voltage levels during battery charging makes it a very useful tool for engineers. Depending on the chemistry of the battery that is being charged, some parameters are closely monitored, thus giving the user an insight of the charging process.

Since the heart of the Smart Charger system is a switching power supply with variable output, the system can easily convert from a battery charger to just a plain power supply. This versatility would be appealing in the industry.

One particular part of the project that was hard to predict and correct was a high frequency noise issue that cluttered the output signal. The noise spike was visible whenever the MOSFET would turn on or off, with a higher amplitude when it was turning on. The noise was in the 20 MHz range and it proved to be very hard to filter out. Our team tried to eliminate the noise from the output, with limited success, using a low pass filter, choke inductor, and ferrite cores. Even though we have managed to reduce the levels of the high frequency to acceptable levels, it was still present, and it is getting worse with higher current output. At this point the source of the noise is believed to be the free-wheeling diode which needs a certain amount of time to switch between reverse bias to conducting. One solution that may fix this problem is to use another MOSFET in the place of the diode, and to switch in such that it will overlap with the main switching MOSFET, thus eliminating the transient time.

Another difficult area in the development of the project was the temperature sensing. Due to high levels of noise at high currents, the temperature sensors would lose precision, which led to incorrect readings, or high jumps between two displayed readings. The team feels that this situation would improve once the high frequency noise disappears from the system. So taking care of the noise should correct the temperature problem, as well.

Another constraint that our team was confronted with was to come up with an elegant solution for a connector that would hook up to any battery topology. As trivial as this may sound, this is a very complicated issue due to the fact that batteries come in countless shapes and forms. One approach that our team took was to provide two banana jacks which would allow for alligator clips and other cables to be used to connect to the battery. Additionally, there is a universal connector that can be used for a variety of battery packs by adding the matching side to the battery needed to be charged either by mechanical means or by soldering it together. Overall, our system is a pioneer in the hybrid world of battery chargers and power supplies, and could provide a solid base for future teams to build upon.

References

1. Microchip MCP 607 Precision Operational Amplifier Data Sheet.
2. Microchip PIC18F452 Data Sheet.
3. Analog Devices AD7705 External Analog-to-Digital Converter Data Sheet.
4. Maxim RS-232 Interface Data Sheet.
5. CrystalFontz 40x4 Liquid Crystal Display Data Sheet.
6. LM34 Precision Temperature Sensor Package Data Sheet.
7. Sharp GP1UM28YK Infrared Detecting Unit Data Sheet.

Appendix

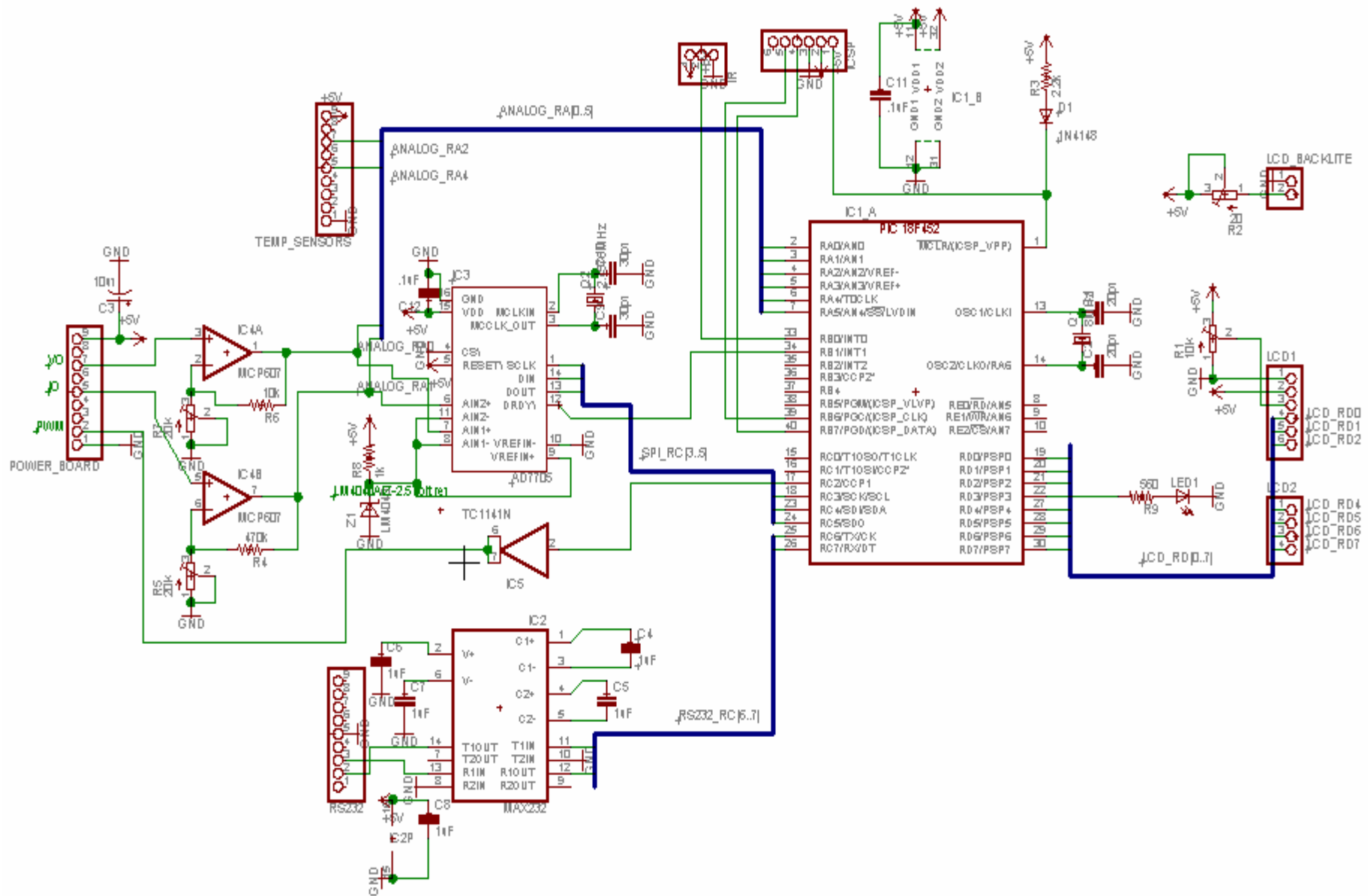


Figure A1. Logic Control Board Schematic with Headers to Accessory Devices.

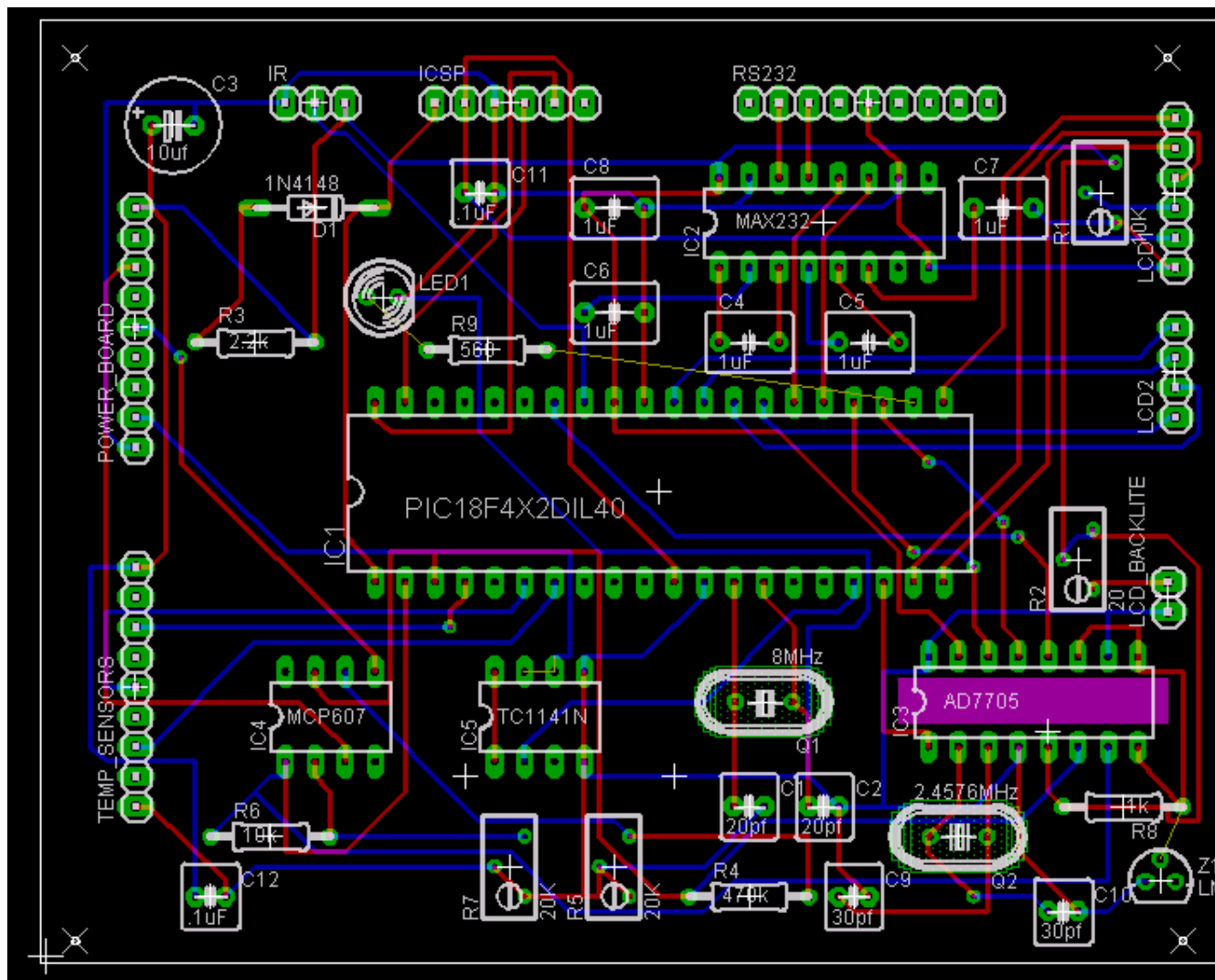


Figure A2. Preliminary System Control PCB Artwork.

Table A1. Communication Protocol.

	PC sends			Device Replies									
Packet Type	Byte 0	1	2	Byte 0	1	2	3	4	5	6	7	8	notes
set Voltage	'V'	hb	lb	'v'	hb	lb							Vmax=0xFFFF, 0V=0x0000
set Current	'I'	hb	lb	'i'	hb	lb							Imax=0xFFFF, 0A=0x0000
set Control	'C'	0,1		'c'	0,1								0=local control, 1=pc control
set OutputOnOff	'O'	0,1		'o'	0,1								0=output off, 1=output on
get Data	'R'			'r'	Vhb	Vlb	Ihb	Ilb	T0hb	T0lb	T1hb	T1lb	device to send actual voltage, current, temperatures (V,I 16bit, T 10 bit)
get PWM	'P'			'p'	hb	lb							device replies with current 10 bit PWM value
get Status	'S'			's'	Mode	Control	errorCode						get mode, control, last error status
impromptu error				'e'	errorCode								device detects error or warning condition and transmits error code without prompting

General Notes:

multi byte data types expressed in big endian form

Voltage and Current input/output are 16 bit values

10 bit values (such as temperature and pwm status) are right justified in 16 bit field

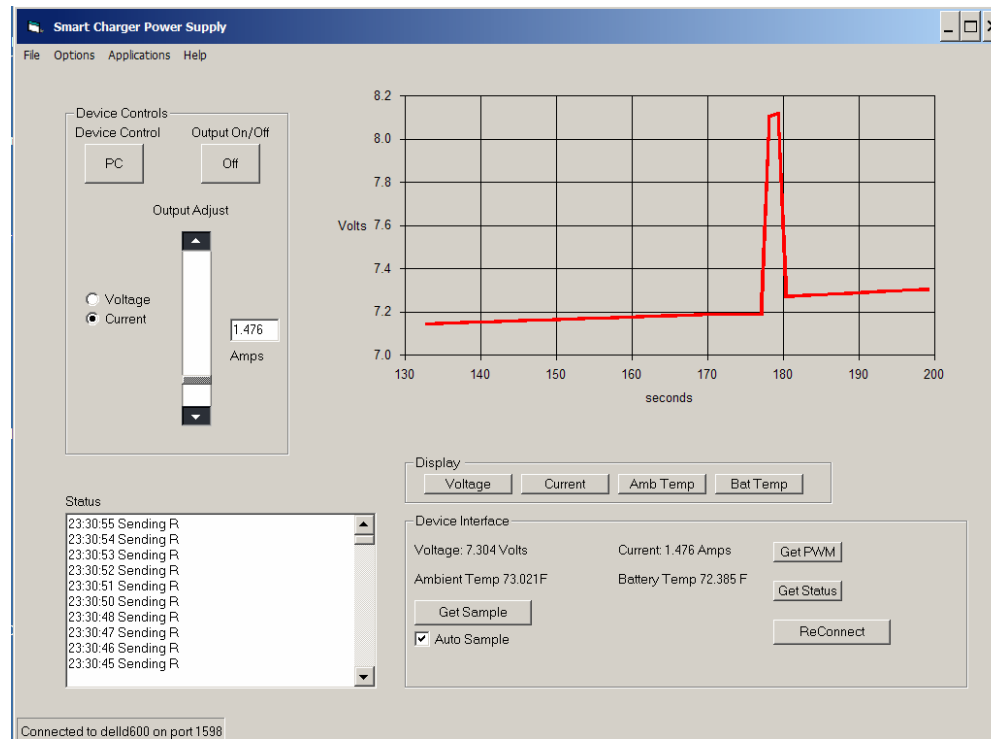
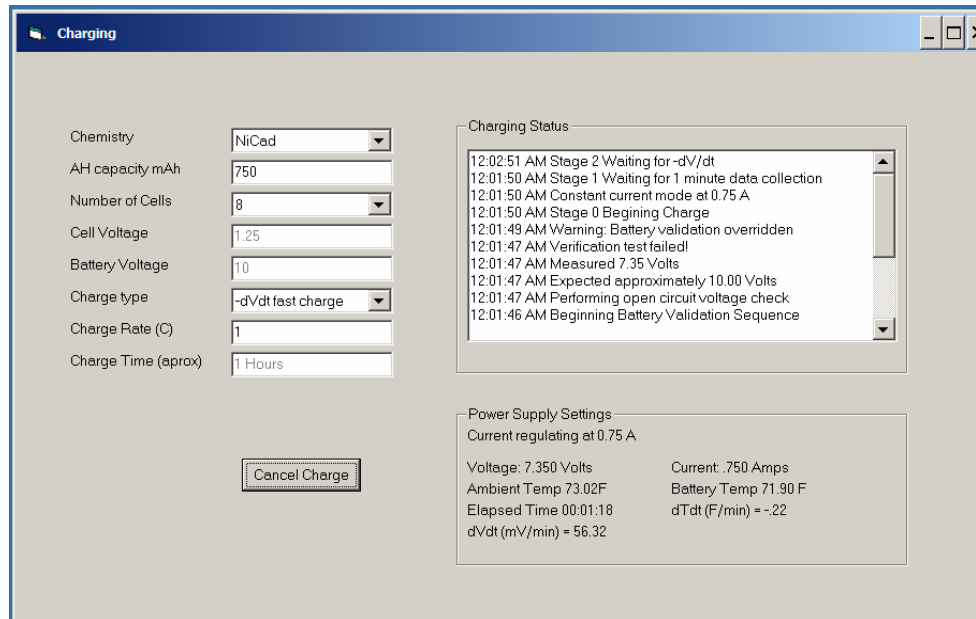



Figure A3. Smart Charger Main Form



The "Charging" window contains the following controls and information:

- Chemistry:** NiCad (dropdown)
- AH capacity mAh:** 750 (text input)
- Number of Cells:** 8 (dropdown)
- Cell Voltage:** 1.25 (text input)
- Battery Voltage:** 10 (text input)
- Charge type:** -dVdt fast charge (dropdown)
- Charge Rate (C):** 1 (text input)
- Charge Time (aprox):** 1 Hours (text input)
- Cancel Charge:** Button
- Charging Status:**
 - 12:02:51 AM Stage 2 Waiting for -dV/dt
 - 12:01:50 AM Stage 1 Waiting for 1 minute data collection
 - 12:01:50 AM Constant current mode at 0.75 A
 - 12:01:50 AM Stage 0 Beginning Charge
 - 12:01:49 AM Warning: Battery validation overridden
 - 12:01:47 AM Verification test failed!
 - 12:01:47 AM Measured 7.35 Volts
 - 12:01:47 AM Expected approximately 10.00 Volts
 - 12:01:47 AM Performing open circuit voltage check
 - 12:01:46 AM Beginning Battery Validation Sequence
- Power Supply Settings:**
 - Current regulating at 0.75 A
 - Voltage: 7.350 Volts
 - Ambient Temp 73.02F
 - Elapsed Time 00:01:18
 - dVdt (mV/min) = 56.32
 - Current: .750 Amps
 - Battery Temp 71.90 F
 - dTdt (F/min) = -.22

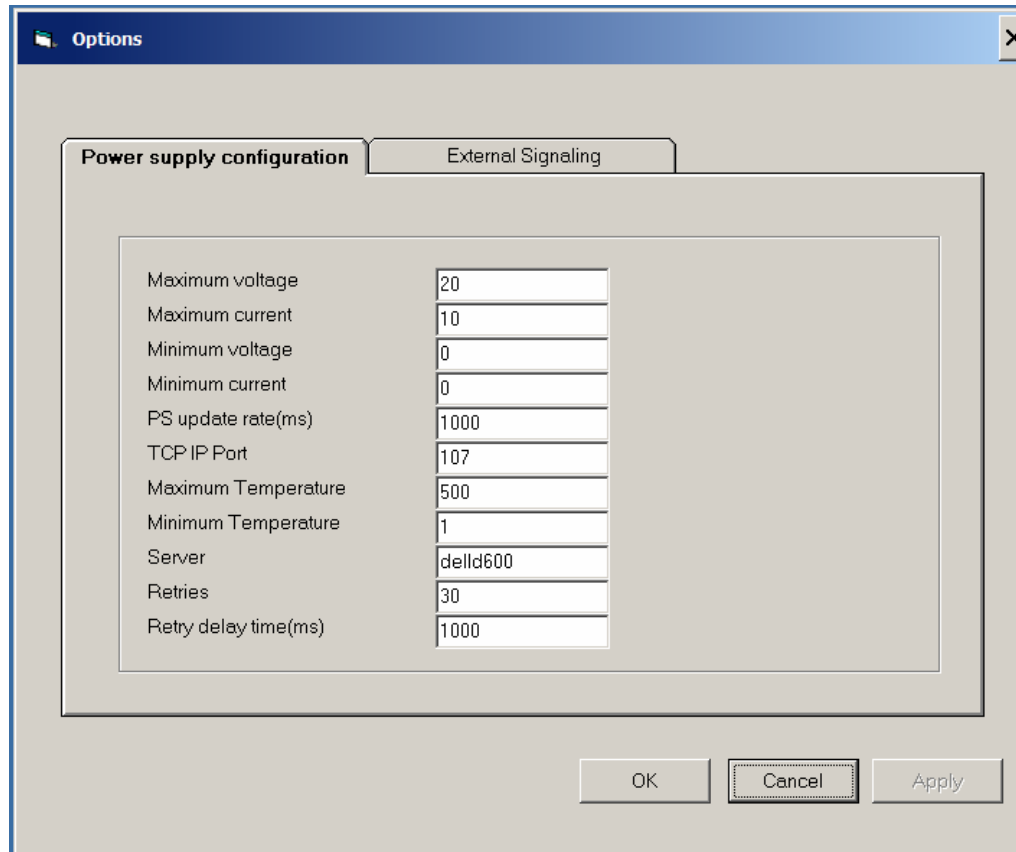
Figure A4. Smart Charger Charging Form



The "About SmartCharger" window displays the following information:

- SmartCharger**
- Version 1.0.0**
- By Steven Savage 12/20/04**
- Description:** This application allows control of the smart charger power supply and charges batteries if desired.
- Legal Notice:** This software cannot be copied or modified without permission of the author!
- Buttons:** OK, System Info...

Figure A5. Smart Charger Help About Form



The image shows a software window titled "Options" with a close button (X) in the top right corner. Inside the window, there are two tabs: "Power supply configuration" (which is selected) and "External Signaling". The "Power supply configuration" tab contains a list of settings, each with a text input field to its right. The settings and their values are: Maximum voltage (20), Maximum current (10), Minimum voltage (0), Minimum current (0), PS update rate(ms) (1000), TCP IP Port (107), Maximum Temperature (500), Minimum Temperature (1), Server (delld600), Retries (30), and Retry delay time(ms) (1000). At the bottom right of the window, there are three buttons: "OK", "Cancel", and "Apply".

Setting	Value
Maximum voltage	20
Maximum current	10
Minimum voltage	0
Minimum current	0
PS update rate(ms)	1000
TCP IP Port	107
Maximum Temperature	500
Minimum Temperature	1
Server	delld600
Retries	30
Retry delay time(ms)	1000

Figure A6. Smart Charger Options Form

Smart Charger TCP-RS232 bridge server and PS emulator.

File Options

Smart Charger TCP-RS232 bridge server and PS emulator.
 By Steven Savage
 5/1/2005 11:27:18 PM Accepted application connection from 192.168.1.101 on port 1598
 5/2/2005 12:07:13 AM Client inactivity timeout
 5/2/2005 12:07:13 AM Application connection closed
 5/2/2005 12:10:39 AM Accepted application connection from 192.168.1.101 on port 1602

Close Terminal Connection Close Application Connection

Mode
☐ Device
☒ Emulate

Data Monitor

To Comm
 82

From Comm
 114 85 112 17 116 5 214 5 192 92

Name: DelID600 IP: 192.168.1.101 Terminal Interface port: 23 Application Interface port: 107 Comm Port: 1 19200,n,8,1

Figure A7. Smart Charger Server Main Form

The image shows a software dialog box titled "Options" with a close button (X) in the top right corner. It contains two tabs: "Server configuration" (selected) and "Spare". The "Server configuration" tab contains a group box labeled "defined in form load" which lists various parameters and their values in input fields:

Parameter	Value
Maximum voltage	20
Maximum current	10
Minimum voltage	0
Minimum current	0
TCP IP Port1	23
TCP IP Port2	107
Maximum Temperature	500
Minimum Temperature	1
Retries	3
Retry delay time(ms)	100
Comm port	1

At the bottom of the dialog box are three buttons: "OK", "Cancel" (which is highlighted with a dashed border), and "Apply".

Figure A8. Smart Charger Server Options Form

Implementation Listings:

Microcontroller Code:

Listing A1. Main routine in scharger.c .

```
// Smart Charger main code
// scharger.c
// Written by Steven Savage and Michael Petrak

#include <18F452.h>                // specifies all settings for the Microchip
PIC18F452
#fuses H4, NOWDT, NOPROTECT, NOLVP
#use delay( clock = 32000000 )    // 32 MHz clock
#use rs232( baud = 19200, XMIT = PIN_C6, RCV = PIN_C7 )// , ERRORS )
// errors causes the compiler to keep receive errors in the variable RS232_ERRORS and to
// reset errors when they occur

//#define SIM 1                  // used in MPLAB simulation mode
#define COMBUFFSIZE 10

static int1 packet_ready = false; // packet ready flag
static int1 onOff = 1; // output is on when the system starts
static int8 buff[ COMBUFFSIZE ] = { 0 }; // circular buffer for communication
//between computer and PIC
static int8 commNotActive = 0; // this counter keeps a track of approximately how many
//seconds the communication between the PIC and the PC has not been active during PC
//control mode
static int16 duty = 0; // duty cycle begins at 0
static int1 ExtADReady = false; // run PWM control algorithm in mainline routine
static float setpoint=0;

static int8 Echan; //external adc channel 0=voltage, 1=current
enum STATE { NOREG, VOLTAGE, CURRENT }; // present state of the Smart Charger
enum STATE mode = NOREG; // begin in voltage mode

enum LED { NONE, RED, GREEN, AMBER };

enum CONTROLLER { LOCAL, PC }; // present control of the Smart Charger
enum CONTROLLER control = LOCAL; // start in local control

#include "pwmcontrol.c"
#include "pwm.c" // pulse width modulation
#include "voltages.c" //voltage functions
#include "currents.c" //current functions
#include "temperatures.c" //temperature functions
#include "LCD440s.c" // LCD
#include "serial.c" // processes the RS232 serial transfers
#include "AD7705spi.c" // external ADC
#include "analog.c" //direct access to internal analog channels and analog
//configuration
#include "led.c" // front-panel LED
#include "freetimer.c" // free-running timer
#include "ir.c" // IR remote

// Main routine initializes all devices and performs system control
void main()
{
    int32 t; // for the timer

    setLED( RED );
```

```

// Initialize modules
freetimer_init(); // Setup free running timer with Timer0

#ifndef SIM

    lcd_init();          // Init. the LCD
    adc_init();           // Initialize 10-bit PIC adc; defined in lm34.c
    pwm_init();           // Init. pulse width modulation
    ext_adc_init();       // Init external 16-bit adc AD7705 on hardware SPI

    lcd_putc1("\fSmart Charger\n");
    delay_ms( 10 );
    lcd_putc2("\fSteven Savage, Michael Petrak,\n");
    lcd_putc2("Claudiu Bouruc, Eric Boyer\n");
    delay_ms( 3000 );

#endif

// Setup external interupt for ir detector
EXT_INT_EDGE( 0, H_TO_L );          // external interupt 0, high to low transition
enable_interrupts( INT_RDA );       // interrupt on RS-232
enable_interrupts( INT_RTCC );      // interrupt on Timer0 overflow
enable_interrupts( INT_EXT );       // external interupt #0
EXT_INT_EDGE( 1, H_TO_L );
enable_interrupts( INT_EXT1 );
enable_interrupts( GLOBAL );        // enable global interrupts

setLED( NONE );

// Endless loop
for(;;){

    // sample the temperatures to fill the appropriate buffers
    if ( BIT_TEST( pulse(), 1 ) ) {
        sampleTemps();
    }

    //extADready is set by ext ad converter data ready interrupt, save new voltage and
    //current samples
    //and calc new pwm
    if (extADready){ //external adc interupt flag bit, new data ready
        if (Echan){
            UpdateExtCurrent();          //update the current variables
            if ( mode == CURRENT ){ //if current mode then calc new pwm
                if (onOff)
                    duty=getPWM(); //determine pwm
                else
                    duty=0;
                set_pwm1_duty( duty ); // set the duty cycle
            }
            Echan=0;                      //toggle channel for next sample
        }
        else{
            UpdateExtVoltage();
            if ( mode == VOLTAGE ){
                if (onOff)
                    duty=getPWM();
                else
                    duty=0;
                set_pwm1_duty( duty ); // set the duty cycle
            }
        }
    }
}

```

```

        Echan=1;
    }
    eadc_select_channel(Echan); //toggle channel select
    extADready=false;
} //end of if

if ((mode == NOREG) && (duty > 0)){
    setLED( GREEN );
    set_pwm1_duty(duty); //allow manual control of duty cycle
} //end of if

```

Listing A2. pwmcontrol.h

```

// pwmcontrol.h

#ifndef PWMCONTROL_H
#define PWMCONTROL_H

void setSetpoint(int16 sp);
int16 getPWM( void );

#endif

```

Listing A3. pwmcontrol.c

```

// pwmcontrol.c
//implements PWM controller

#include "pwmcontrol.h"
//#include "serial.h"
#include "voltages.h"
#include "currents.h"

#define MAXI 32000
#define MINI -32000
#define CERRDELAY 500

#define maxCurrent 10
#define maxV 20
#define maxPWM 1023
#define minPWM 0

static signed int16 actPWM = 0;

void setVsetpoint(int16 sp)
{
    if ( setpoint == 0 ) {
        setpoint=(float)sp/0xffff*20; //convert integer setpoint into float
        actPWM = setpoint * 1023 / 40;
    }
    else {
        setpoint = (float)sp/0xffff*20;
    }
    mode=VOLTAGE;
}

void setIsetpoint(int16 sp)
{
    if ( setpoint == 0 ) {
        setpoint=(float)sp/0xffff*10; //convert integer setpoint into float

```

```

        actPWM = 10;
    }
    else
        setpoint=(float)sp/0xffff*10;    //convert integer setpoint into float

    mode=CURRENT;
}

int16 getPWM( void )
{

    float actVoltage;
    float actCurrent;
    float delta;
    signed int16 dpwm;
    int16 lastPWM;

    int8 dVf;
    int8 dIf;

    if ( setpoint == 0 ){
        setLED( NONE );
        return 0;
    }//end if

    dVf=5;
    dIf=10;

    lastPWM=actPWM;

    actVoltage=getExtVoltage();
    actCurrent=getExtCurrent();
    if ( mode == VOLTAGE ) {
        delta=setpoint-actVoltage;
        if ((actCurrent+.5)>maxI) //current close to rail, limit delta
            dVf=1;
        if (delta>0)                //determine increment or decrement
            dpwm=delta * dVf + 1;
        else
            dpwm=delta * dVf -1;

        if (actCurrent<maxI){        //normal calculation
            actPWM=actPWM + dpwm;
            setLED( GREEN );
        }
        else{ //current over limit
            setLED( RED );
            delta=maxI-actCurrent;
            if (delta>0)
                dpwm=delta * dIf + 1;
            else
                dpwm=delta * dIf -1;
            actPWM=actPWM+dpwm;    //overloaded
        }
    }
    else{ //current mode
        delta=setpoint-actCurrent;
        if ((actVoltage+1)>maxV) //voltage close to rail, limit delta
            dIf=1;
        if (delta>0)
            dpwm=delta * dIf + 1;
        else

```



```

        dpwm=delta * dIf -1;

        if (actVoltage<maxV){           //normal calculation
            actPWM=actPWM + dpwm;
            setLED( GREEN );
        }
        else{ //voltage over limit
            setLED( RED );
            delta=maxV-actVoltage;
            if (delta>0)
                dpwm=delta * dVf + 1;
            else
                dpwm=delta * dVf -1;
            actPWM=actPWM+dpwm; //overloaded
        }
    }

    //slew rate limits
    //if (actPWM > (lastPWM+10))
    //  actPWM=lastPWM+10;

    //if (actPWM < (lastPWM-10))
    //  actPWM=lastPWM-10;

    //absolute limits
    if ( actPWM > maxPWM )
        actPWM = maxPWM;

    if (actPWM < minPWM)
        actPWM = minPWM;

    return actPWM;

}

```

Listing A4. pwm.h

```

// pwm.h

#ifndef PWM_H
#define PWM_H

void pwm_init();

#endif

```

Listing A5. pwm.c

```

#include "pwm.h"

void pwm_init(){
    setup_ccp1(CCP_PWM);
    setup_timer_2(T2_DIV_BY_1, 255, 1);
    set_pwm1_duty(0); // 0% duty cycle at init.
}

```

Listing A6. voltages.h

```

#ifndef VOLTAGES_H
#define VOLTAGES_H

int16 getRawVoltage();
float getVoltage();
int16 getRawExtVoltage();
float getExtVoltage();

```

```

void UpdateExtVoltage();

float convertToVoltage(int16 arg, int16 max);
//for 16 bit conversion pass 0xFFFF in max
//for 10 bit conversion pass 0x03FF in max
#endif

```

Listing A7. voltages.c

```

//voltage acquisition routines
#include "ad7705spi.h"
#include "analog.h"
static int16 ExtRawVoltage;
static float ExtVoltage;

int16 getRawVoltage(){
    return getAnalog(0); //voltage on analog channel 0
}

float getVoltage(){
    int16 data;
    data=getRawVoltage();
    return ((float)data*22/0x3ff); //scaled voltage 10bit resolution
}

//external volatge from ad7705
int16 getRawExtVoltage(){
    return ExtRawVoltage; //voltage on ext adc chan 0
}

float getExtVoltage(){
    return ExtVoltage; //return floating point voltage
}

void UpdateExtVoltage(){
    ExtRawVoltage=read_adc_value(0); //update raw 16bit integer value
    ExtVoltage=(float)ExtRawVoltage*22/0xffff; //update floating point value
}

float convertToVoltage(int16 arg, int16 max){
    //for 16 bit conversion pass 0xFFFF in max
    //for 10 bit conversion pass 0x03FF in max
    return ((float)arg*22/max); //scaled voltage
}

```

Listing A8. currents.h

```

#ifndef CURRENTS_H
#define CURRENTS_H

int16 getRawCurrent();
float getCurrent();
int16 getRawExtCurrent();
float getExtCurrent();
void UpdateExtCurrent();

float convertToCurrent(int16 arg, int16 max);
//for 16 bit conversion pass 0xFFFF in max
//for 10 bit conversion pass 0x03FF in max
#endif

```

Listing A9. currents.c

```
//current acquisition routines
#include "ad7705spi.h"
#include "analog.h"
static int16 RawExtCurrent;
static float ExtCurrent;

int16 getRawCurrent(){
    return getAnalog(1); //Current on analog channel 1
}

float getCurrent(){
    int16 data;
    data=getRawCurrent();
    return ((float)data*11/0x3ff); //scaled current 10bit resolution
}

//external volatge from ad7705
int16 getRawExtCurrent(){
    return RawExtCurrent; //current on ext adc chan 1
}

float getExtCurrent(){
    return ExtCurrent; //floating point current value
}

void UpdateExtCurrent(){
    signed int16 y;
    int16 value;

    value=read_adc_value(1);

    if (value < 1340){ //1340 coresponds to 225ma
        y=value+((float)value - 1340)*0.654321;
        if (y<0)
            RawExtCurrent=0;
        else
            RawExtCurrent=y; //non linear conversion factor for current < 200ma
    }
    else
        RawExtCurrent=value;

    ExtCurrent=(float)RawExtCurrent*11/0xffff;
}

float convertToCurrent(int16 arg, int16 max){
    //for 16 bit conversion pass 0xFFFF in max
    //for 10 bit conversion pass 0x03FF in max
    return ((float)arg*11/max); //scaled voltage
}
```

Listing A10. temperatures.h

```
#ifndef TEMPERATURES_H
#define TEMPERATURES_H

void sampleTemps();
int16 getRawAmbTemp();
int16 getRawBatTemp();
float getAmbTemp();
float getBatTemp();
```

```
float convertToTemp(int16 arg);

#endif
```

Listing A11. temperatures.c

```
//functions to retrieve ambient and battery temperatures

#include "analog.h"
#define TEMPBUFFSIZE 12

static int16 tempAmbBuff[ TEMPBUFFSIZE ] = { 0 };
static int16 tempBatBuff[ TEMPBUFFSIZE ] = { 0 };

//call this function periodically to update temperature array buffer
void sampleTemps(){
    static int8 tIndex=0;

    int16 data;
    data = getAnalog( 2 );

    // store the value in the buffer
    tempAmbBuff[ tIndex ] = data;

    data = getAnalog( 3 );

    // store the value in the buffer
    tempBatBuff[ tIndex ] = data;

    tIndex++;

    // if the index just went beyond the buffer, wrap around
    if ( tIndex == TEMPBUFFSIZE )
        tIndex = 0;
}

int16 getRawAmbTemp(){
    int8 count;
    int16 sum;
    int16 max;
    int16 min;
    int16 data;
    sum = 0;
    max=0;
    min=1024;

    for ( count = 0; count < TEMPBUFFSIZE; count++ ){
        data=tempAmbBuff[count];
        if (data>max)
            max=data;
        if (data<min)
            min=data;
        sum += data;
    }
    sum=sum-max-min;

    return sum;
}

int16 getRawBatTemp(){
    int8 count;
```

```

    int16 sum;
    int16 max;
    int16 min;
    int16 data;
    sum = 0;
    max=0;
    min=1024;

    for ( count = 0; count < TEMPBUFFSIZE; count++ ){
        data=tempBatBuff[count];
        if (data>max)
            max=data;
        if (data<min)
            min=data;
        sum += data;
    }
    sum=sum-max-min;

    return sum;
}

float getAmbTemp(){
    float temp;
    int16 data;
    data=getRawAmbTemp();
    temp = ((float)data*125)/2560;
    return temp;
}

float getBatTemp(){
    float temp;
    int16 data;
    data=getRawBatTemp();
    temp = ((float)data*125)/2560;
    return temp;
}

// utility conversion function
float convertToTemp(int16 arg){
    return ((float)arg*125)/256;
}

```

Listing A12. lcd440s.h

```

// LCD440s.h
#ifndef LCD440S_H
#define LCD440S_H

int lcd_read_byte1();
int lcd_read_byte2();
void lcd_send_nibble1( int n );
void lcd_send_nibble2( int n );
void lcd_send_byte1( int address, int n );
void lcd_send_byte2( int address, int n );
void lcd_init(); // call this to initialize the LCD
void lcd_init1();
void lcd_init2();
void lcd_gotoxy1( int x, int y );
void lcd_gotoxy2( int x, int y );
void lcd_putc1( char c );
void lcd_putc2( char c );

```

```

void updateDisplay();
void displayLCDError( int8 error );

#endif

```

Listing A13. lcd440s.c

```

/////////////////////////////////////////////////////////////////
////                      LCD440s.C                      ////
////          Driver for common 4x40 LCD modules          ////
////                                                     ////
////  lcd_init()    Must be called before any other function.  ////
////                                                     ////
////  lcd_putc(c)   Will display c on the next position of the LCD.  ////
////               The following have special meaning:         ////
////               \f  Clear display                          ////
////               \n  next line                               ////
////               \b  Move back one position                 ////
////                                                     ////
////  lcd_gotoxy(x,y) Set write position on LCD (upper left is 1,1)  ////
////                  column x, row y                          ////
////  lcd_getc(x,y)   Returns character at position x,y on LCD      ////
////                                                     ////
/////////////////////////////////////////////////////////////////
////          (C) Copyright 1996,1997 Custom Computer Services  ////
////  This source code may only be used by licensed users of the CCS C  ////
////  compiler.  This source code may only be distributed to other  ////
////  licensed users of the CCS C compiler.  No other use, reproduction  ////
////  or distribution is permitted without written permission.      ////
////  Derivative programs created using this software in object code  ////
////  form are not restricted in any way.                          ////
/////////////////////////////////////////////////////////////////
#include "LCD440s.h"
#include "AD7705spi.h"
#include "temperatures.h"

//modified Steve Savage 10/7/04
// As defined in the following structure the pin connection is as follows:
//      pic      lcdName      LCD pin number pin header
//      gnd       13           1
//      Vcc       14           2
//      Vee(Vlc)  12           3
//
//      D0        rs           11          4
//      D1        rw           10          5
//      D2        enable1      9           6
//      D3        enable2     15          7
//      D4        D4           4           8
//      D5        D5           3           9
//      D6        D6           2          10
//      D7        D7           1          11
//
//      LCD pins D0-D3 are not used.

struct lcd_pin_map {
    BOOLEAN rs;
    BOOLEAN rw;
    BOOLEAN enable1;
    BOOLEAN enable2;

    int      data : 4;
}

// This structure is overlaid
// on to an I/O port to gain
// access to the LCD pins.
// The bits are allocated from
// low order up.  rs will
// be pin D0.

```

```

    } lcd;

#byte lcd = 0x0F83 // This puts the entire structure
                  // on to port D (at address 8)
#define lcd_type 2 // 0=5x7, 1=5x10, 2=2 lines
#define lcd_line_two 0x40

//int lcdline = 0;
int const LCD_INIT_STRING[4] = {0x20 | (lcd_type << 2), 0xc, 1, 6};
// These bytes need to be sent to the LCD
// to start it up.

// The following are used for setting
// the I/O port direction register.

struct lcd_pin_map const LCD_WRITE = {0,0,0,0,0}; // For write mode all pins are out
struct lcd_pin_map const LCD_READ = {0,0,0,0,15}; // For read mode data pins are in

int lcd_read_byte1()
{
    int low, high;
    set_tris_d(LCD_READ);
    lcd.rw = 1;
    delay_cycles(2);
    lcd.enable1 = 1;
    delay_cycles(2);
    high = lcd.data;
    lcd.enable1 = 0;
    delay_cycles(2);
    lcd.enable1 = 1;
    delay_us(1);
    low = lcd.data;
    lcd.enable1 = 0;
    set_tris_d(LCD_WRITE);
    return ((high << 4) | low);
}

int lcd_read_byte2()
{
    int low, high;
    set_tris_d(LCD_READ);
    lcd.rw = 1;
    delay_cycles(2);
    lcd.enable2 = 1;
    delay_cycles(2);
    high = lcd.data;
    lcd.enable2 = 0;
    delay_cycles(2);
    lcd.enable2 = 1;
    delay_us(1);
    low = lcd.data;
    lcd.enable2 = 0;
    set_tris_d(LCD_WRITE);
    return ((high << 4) | low);
}

void lcd_send_nibble1(int n)
{
    lcd.data = n;
    delay_cycles(2);
    lcd.enable1 = 1;

```

```

    delay_us(2);
    lcd.enable1 = 0;
}

void lcd_send_nibble2(int n)
{
    lcd.data = n;
    delay_cycles(2);
    lcd.enable2 = 1;
    delay_us(2);
    lcd.enable2 = 0;
}

void lcd_send_byte1(int address, int n)
{
    lcd.rs = 0;
    while ( bit_test(lcd_read_byte1(), 7) );
    lcd.rs = address;
    delay_cycles(2);
    lcd.rw = 0;
    delay_cycles(2);
    lcd.enable1 = 0;
    lcd_send_nibble1(n >> 4);
    lcd_send_nibble1(n & 0xf);
}

void lcd_send_byte2(int address, int n)
{
    lcd.rs = 0;
    while ( bit_test(lcd_read_byte2(), 7) );
    lcd.rs = address;
    delay_cycles(2);
    lcd.rw = 0;
    delay_cycles(2);
    lcd.enable2 = 0;
    lcd_send_nibble2(n >> 4);
    lcd_send_nibble2(n & 0xf);
}

void lcd_init()
{
    int8 lcdInitCount;
    for ( lcdInitCount = 0; lcdInitCount <= 9; lcdInitCount++ ) {
        lcd_init1();
        delay_ms(2);    // extended delay between initializations
        lcd_init2();
    }
}

void lcd_init1()
{
    int i;
    set_tris_d(LCD_WRITE);
    lcd.rs = 0;
    lcd.rw = 0;
    lcd.enable1 = 0;
    delay_ms(20);
    for (i = 1; i <= 3; ++i)
    {
        lcd_send_nibble1(3);
        delay_ms(5);
    }
}

```



```

    lcd_send_nibble1(2);
    for (i = 0; i <= 3; ++i)
        lcd_send_byte1(0, LCD_INIT_STRING[i]);
}

```

```

void lcd_init2()
{
    int i;
    //set_tris_d(LCD_WRITE);
    lcd.rs = 0;
    lcd.rw = 0;
    lcd.enable2 = 0;
    //delay_ms(20);
    for (i = 1; i <= 3; ++i)
    {
        lcd_send_nibble2(3);
        delay_ms(5);
    }
    lcd_send_nibble2(2);
    for (i = 0; i <= 3; ++i)
        lcd_send_byte2(0, LCD_INIT_STRING[i]);
}

```

```

// x = column (1 to 40)
// y = row (1 to 2)
void lcd_gotoxy1(int x, int y)
{
    int address;
    if (y != 1)
        address = lcd_line_two;
    else
        address = 0;
    address += x - 1;
    lcd_send_byte1(0, 0x80 | address);
}

```

```

void lcd_gotoxy2(int x, int y)
{
    int address;
    if (y != 1)
        address = lcd_line_two;
    else
        address = 0;
    address += x - 1;
    lcd_send_byte2(0, 0x80 | address);
}

```

```

void lcd_putc1(char c)
{
    switch (c)
    {
        case '\f':
            lcd_send_byte1(0, 1);
            //delay_ms(2);
            break;
        case '\n':
            lcd_gotoxy1(1, 2);
            break;
        case '\b':
            lcd_send_byte1(0, 0x10);
            break;
    }
}

```

```

        default:
            lcd_send_byte1(1, c);
            break;
    }
    delay_us(20);
}

void lcd_putc2(char c)
{
    switch (c)
    {
        case '\f':
            lcd_send_byte2(0, 1);
            //delay_ms(2);
            break;
        case '\n':
            lcd_gotoxy2(1, 2);
            break;
        case '\b':
            lcd_send_byte2(0, 0x10);
            break;
        default:
            lcd_send_byte2(1, c);
            break;
    }
    delay_us(20);
}

void updateDisplay()
{
    // float Volt, Cur;
    float AmbTemp, BattTemp; // used for display of voltage and current
    // Volt = getExtVoltage();
    // Cur = getExtCurrent();
    AmbTemp = getAmbTemp();

    // Row 1: PS / Battery Charger; voltage / current mode; Temp.
    if ( control == LOCAL )
        printf( lcd_putc1, "\fPower Supply: " );
    else
        printf( lcd_putc1, "\fBattery Charger: " );

    if ( mode == VOLTAGE )
        printf( lcd_putc1, "volt. mode      " );
    else if ( mode == CURRENT )
        printf( lcd_putc1, "curr. mode      " );
    else if ( mode == NOREG )
        printf( lcd_putc1, "noreg. mode      " );

    //lcd_gotoxy1( 30, 0 );
    printf( lcd_putc1, "Temp.\n" );

    // Row 2: Voltage, Ambient temp.
    printf( lcd_putc1, "Voltage: %5.3f V      ", getExtVoltage() ); // to LCD
    //lcd_gotoxy2( 23, 1 );
    printf( lcd_putc1, "Amb.: %5.2f\xdf F\n", AmbTemp );

    // add a delay between the switching of enables
    delay_ms( 10 );

    // Row 3: Current, Battery temp.
    printf( lcd_putc2, "\fCurrent: %5.3f A      ", getExtCurrent() );

```

```

//lcd_gotoxy2( 23, 0 );
if ( control == PC ) {    // only check battery temperature in PC controlled mode
    BattTemp = getBatTemp();
    printf( lcd_putc2, "Batt.: %5.2f\xdf F\n", BattTemp ) ;
}
else                      // display "battery not available"
    printf( lcd_putc2, "Batt.: N/A\n" );

// Row 4
if ( commNotActive == 10 && control == PC ) {
    displayLCDError( 1 );
    delay_ms( 4000 );
    control = LOCAL; // change to voltage controlled mode
    setpoint = 0;
    commNotActive = 0;
}
else {
    printf( lcd_putc2, "Duty: %ld          ", duty ); // display the duty cycle
    printf( lcd_putc2, "SP: %5.3f \n", setpoint ); // display the setpoint
}
}

void displayLCDError( int8 error )
{
    switch( error ) {
        case 1:
            printf( lcd_putc2, "Communications timeout with the PC.\n" );
            break;
        default:
            break;
    }
}

```

Listing A14. serial.h

```

// serial.h

#ifndef SERIAL_H
#define SERIAL_H

void nb_putc(int8 ch);
#int_rda
void processSerial();
void processCommand();

#endif

```

Listing A15. serial.c

```

//serial.c

#include "serial.h"
#include <stdlib.h>          // for ASCII to int conversions
//#include "LCD440s.h"      // LCD function prototypes
#include "temperatures.h"   // Temp. sensor header file
#include "AD7705spi.h"     // external ADC header file
#include "voltages.h"
#include "currents.h"
#include "picontrol.h"

#define OUTBUFFSIZE 20

static int8 outbuff[ OUTBUFFSIZE ];

```

```

static int8 inIdx = 0;
static int8 outIdx = 0;

static int8 inptr = 0;          // input pointer for the circular buffer

static int8 cSum = 0; // checksum to verify packets when transmitting

#int_rda
// processSerial executes on an interrupt of the RS-232
void processSerial()
{
    // keep a static int8 to count the number of bytes in the command;
    // when the correct number of bytes has been received, set the packet_ready flag
    // also, set the inptr back to the beginning of the array and reset the byte counter
    int8 ch; // to store the byte that has just been received
    static int8 byteCnt = 0; // counts the number of bytes in the present command packet

    if ( kbhit() ) {          // if a byte has been received
        setLED( AMBER );
        commNotActive = 0;    // reset the inactive communications counter
        ch = getc();           // get and store the byte
        buff[inptr++] = ch;    // store that byte in the buffer
        byteCnt++;            // increment the number of bytes
        if ( ( byteCnt == 1 && // 1-byte packet
            ( buff[0] == 'R' || // check for the case of a complete getData command;
              buff[0] == 'P' || // a complete getPWM command,
              buff[0] == 'S' ) ) || // a complete getStatus command,
            ( byteCnt == 3 && // 3-byte packets
            ( buff[0] == 'C' || // a complete setControl command,
              buff[0] == 'O' ) ) || // a complete setOutputOnOff command,
            ( byteCnt == 4 && // 4-byte packets
            ( buff[0] == 'V' || // a complete setVoltage command,
              buff[0] == 'I' ) ) ) { // or a complete setCurrent command
            inptr = 0;        // reset the buffer insertion pointer
            byteCnt = 0;      // reset the byte counter
            packet_ready = true; // now the voltage command can be processed
            setLED( NONE );
        } // end complete packet check
    } // end kbhit check
}

void processCommand()
{
    // Initialize variables
    int8 Cmdtype; // packet type or byte 0
    static int8 Error; // Error code; initialized to 0 by default
    int16 AmbTemp = 0; // stores temperatures
    int16 BattTemp = 0;
    int16 Vlt = 0;
    int16 Cur = 0;

    Cmdtype = buff[0]; // type of command - the first character

    switch(Cmdtype){
        // check errors at this stage: check Error 1, 2??, and 10 here
        case 'V': // set Voltage
            if ( ( buff[0] + buff[1] + buff[2] ) == buff[3] ) { // verify the checksum;
// checksums force the sum to 0
                control = PC; // automatically switch to pc control
                Vlt = make16( buff[1], buff[2] ); // the 16-bit voltage is the
                                                    //concatenation of the hb and lb
            }
        }
    }
}

```

```

        setVSetpoint( Vlt );
        nb_putc( 'a' );          // respond with acknowledge
    }
    else
        nb_putc( 'n' );        // respond with negative acknowledge
    break;
case 'I': // set Current
    if ( ( buff[0] + buff[1] + buff[2] ) == buff[3] ) { // verify the checksum
        control = PC;        // automatically switch to pc control
        Cur = makel6( buff[1], buff[2] ); // the 16-bit current is the
                                         // concatenation of the hb and lb

        setISetpoint( Cur );
        nb_putc( 'a' );      // respond with acknowledge
    }
    else
        nb_putc( 'n' );      // respond with negative acknowledge
    break;
case 'C': // set Control
    if ( ( buff[0] + buff[1] ) == buff[2] ) { // verify checksum
        if ( buff[1] == 1 ) // pc control
            control = PC;
        else
            control = LOCAL;
        nb_putc( 'a' ); // respond with acknowledge
    }
    else
        nb_putc( 'a' ); // respond with negative acknowledge
    break;
case 'O': // set OutputOnOff -
    if ( ( buff[0] + buff[1] ) == buff[2] ) { // verify checksum
        if ( buff[1] == 1 ) // output on
            onOff = 1;
        else
            onOff = 0; // clear for LiOn batteries
        nb_putc( 'a' ); // respond with acknowledge
    }
    else
        nb_putc( 'n' ); // respond with negative acknowledge
    break;
case 'R': // get Data
    Vlt = getRawExtVoltage();
    Cur = getRawExtCurrent();
    AmbTemp = getRawAmbTemp();
    BattTemp = getRawBatTemp();
    send8( 'r' );
    send16( Vlt );
    send16( Cur );
    send16( AmbTemp );
    send16( BattTemp );
    sendcSum();
    break;
case 'P': // get PWM
    send8( 'p' );
    send16( Duty );
    sendcSum();
    break;
case 'S': // get Status
    send8( 's' );
    send8( Mode );
    send8( control );
    sendcSum(); // return the checksum for verification
    break;

```

```

        default:    // impromptu error
            send8( 'e' );
            send8( Error );
            sendcSum();
            break;
    }
}

//non blocking putc stores characters in an output buffer and drives
//interrupt routine to transmit characters
void nb_putc( int8 ch )
{
    int8 nextIdx;
    nextIdx = ( inIdx + 1 ) % OUTBUFFSIZE;
    while( nextIdx == outIdx ); //block while buffer is full
    outBuff[nextIdx] = ch;
    inIdx = nextIdx;
    enable_interrupts( INT_TBE );
}

//transmit buffer empty interrupt routine

//transmit buffer empty
#INT_TBE
void transmit_rs232()
{
    putc( outBuff[outIdx] );
    outIdx = ( outIdx + 1 ) % OUTBUFFSIZE;
    if (outIdx == inIdx)
        disable_interrupts( INT_TBE );
}

void send8( int8 arg )
{
    cSum += arg;
    putc( arg ); // nb_putc( arg );
}

void send16( int16 arg )
{
    int8 b;
    b = make8( arg, 1 ); // get the MSB
    cSum += b;
    putc( b ); // nb_putc( b );
    b = make8( arg, 0 ); // get the LSB
    cSum += b;
    putc( b ); // nb_putc( b );
}

void sendcSum()
{
    putc( cSum ); // nb_putc( cSum );
    cSum = 0;
}

```

Listing A16. ad7705spi.h

```

// AD7705spi.h
#ifndef AD7705SPI_H
#define AD7705SPI_H

// function prototypes
void ext_adc_init();

```

```

int1 adc_read_not_ready();
int16 read_adc_value(int1 ch);
void eadc_select_channel(int8 channel);
int16 read_adc_word();
void adc_disable();
void send8( int8 arg );
void send16( int16 arg );
void sendcSum();
#int_ext1
void adexternal_isr();

#endif

```

Listing A17. ad7705spi.c

```

/////////////////////////////////////////////////////////////////
////                                AD7705.C                                ////
////                                Driver for analog device AD7705          ////
////                                                                ////
//// ext_adc_init()                Call after power up                ////
////                                                                ////
//// read_adc_value(channel)Read adc value from the specified channel    ////
////                                                                ////
//// adc_disable()    Disables the adc conversion                    ////
/////////////////////////////////////////////////////////////////
////                                (C) Copyright 1996,2003 Custom Computer Services        ////
//// This source code may only be used by licensed users of the CCS C    ////
//// compiler.  This source code may only be distributed to other        ////
//// licensed users of the CCS C compiler.  No other use, reproduction    ////
//// or distribution is permitted without written permission.            ////
//// Derivative programs created using this software in object code      ////
//// form are not restricted in any way.                                  ////
/////////////////////////////////////////////////////////////////
//// Driver routines for the AD7705 chip
//Assuming a 2.4576 crystal ocsillator is used between MCLK IN and MCLK OUT

#include "AD7705spi.h"    // header file includes function prototypes

//Operation modes (setup register)
#define ADC_NORMAL 0x00
#define ADC_SELF 0x40      //self calibration
#define ADC_ZERO_SCALE 0x80    //zero scale calibration
#define ADC_FULL_SCALE 0xc0    //full scale calibration

//Gain settings    (setup register)
#define ADC_GAIN_1 0x00
#define ADC_GAIN_2 0x08
#define ADC_GAIN_4 0x10
#define ADC_GAIN_8 0x18
#define ADC_GAIN_16 0x20
#define ADC_GAIN_32 0x28
#define ADC_GAIN_64 0x30
#define ADC_GAIN_128 0x38

//Polar operations (setup register)
#define ADC_BIPOLAR 0x00
#define ADC_UNIPOLAR 0x04

//update rates (defined in clock register)
#define ADC_50 0x04
#define ADC_60 0x05
#define ADC_250 0x06

```

```

#define ADC_500 0x07

//channel selection (communications register)
#define ADC_CH_0 0x00
#define ADC_CH_1 0x01

#define ADC_REG_COMM 0x00
#define ADC_REG_SETUP 0x10
#define ADC_REG_CLOCK 0x20
#define ADC_REG_DATA 0x30
#define ADC_REG_TEST 0x40
#define ADC_REG_OFFSET 0x60
#define ADC_REG_GAIN 0x70

// Initialization routine: AD7705 used in SPI mode
void ext_adc_init()
{
    int8 reg, i;

    // configure and initialize microcontroller serial port
    setup_spi(SPI_XMIT_L_TO_H|SPI_MASTER|SPI_H_TO_L|SPI_CLK_DIV_64);

    for(i=0 ; i<4 ; i++)
        spi_write(0xFF); //initialize spi interface with write of 32 ones

    reg = ADC_REG_CLOCK; //select clock register; 0x20 hex
    spi_write(reg);

    // updated to 500 Hz update rate - MP
    reg = ADC_500; //sample rate
    spi_write(reg);

    reg = ADC_REG_SETUP; //select setup register, with ch0; 0x10 hex
    spi_write(reg);

    reg = ADC_SELF|ADC_GAIN_1|ADC_BIPOLAR; //setup information ch0, calibrate
    spi_write(reg);

    while(adc_read_not_ready()); //poll data ready pin; wait until calibration is complete

    reg = ADC_REG_SETUP|ADC_CH_1; //select setup register, with ch1
    spi_write(reg);

    reg = ADC_SELF|ADC_GAIN_1|ADC_BIPOLAR; //setup information ch1, calibrate
    spi_write(reg);

    while(adc_read_not_ready()); //wait until calibration is complete

    delay_ms(3);
}

int1 adc_read_not_ready(){
    return input(PIN_B1);
}

//read an adc value from the specified channel
int16 read_adc_value(int1 ch)
{
    while(adc_read_not_ready()); //wait for
    if(ch)
        //communications register set to read
        // of data register of channel 1

```



```

        spi_write(0x39);
    else
        //communications register set to read
        // of data register of channel 0
        spi_write(0x38);

    while(adc_read_not_ready());
    return read_adc_word();
}

void eadc_select_channel(int8 channel){
    spi_write(channel);
}

int16 read_adc_word()
{
    int16 data;
    int8 hByte;
    int8 lByte;
    hByte = spi_read(0xFF);
    lByte = spi_read(0xFF);
    data = makel6(hByte, lByte);
    return data;
}

//disable the a/d conversion
void adc_disable()
{
    spi_write( 0x20 );    //Communications Register set to
                        //write of clock register
    spi_write( 0x10 );    //Clock Register info here
}

// external interrupt routine to use the AD7705
#int_ext1
void adexternal_isr()
{
    extADready = true;
}

```

Listing A18. analog.c

```

//analog.c
#include <stdlib.h> // for read_adc

#define MAXCHARGETIME 100

void adc_init()
{
    setup_port_a( ALL_ANALOG ); //all analog inputs
    setup_adc( ADC_CLOCK_DIV_64 );
}

int16 getAnalog( int chan ){
    static int8 chanMem = 8; //an invalid channel to force selection
    if( chan != chanMem ){    //test if channel already selected
        set_adc_channel( chan );
        delay_us( MAXCHARGETIME );
        chanMem = chan;
    }
    return read_adc();
}

```

Listing A19. led.c .

```
// led.c
void setLED(LED desiredMode)
{
    if (desiredMode == NONE) {
        OUTPUT_BIT(PIN_B2, 0);
        OUTPUT_BIT(PIN_B3, 0);
    }
    // initialization
    else if (desiredMode == RED) { // turn on pin1
        OUTPUT_BIT(PIN_B2, 1);
        OUTPUT_BIT(PIN_B3, 0);
    }
    // duty > 0
    else if (desiredMode == GREEN ) { // turn on pin2
        OUTPUT_BIT(PIN_B2, 0);
        OUTPUT_BIT(PIN_B3, 1);
    }
    else if (desiredMode == AMBER) { // turn on both pins
        OUTPUT_BIT(PIN_B2, 1);
        OUTPUT_BIT(PIN_B3, 1);
    }
}
```

Listing A20. freetimer.h

```
// freetimer.h
#ifndef FREETIMER_H
#define FREETIMER_H

void freetimer_init();
#int_rtcc
void timer0_isr();
int32 timer();
int16 pulse();

#endif
```

Listing A21. freetimer.c

```
//#include "freetimer.h"

static int16 tmr_high;

void freetimer_init(){
    setup_timer_0(RTCC_DIV_8|RTCC_INTERNAL);    //setup free running timer, timer 0 16 bit
    mode
}

#int_rtcc
void timer0_isr(){
    tmr_high++;    //increment high byte
}

int32 timer(){
    return make32(tmr_high, get_timer0());
}

int16 pulse(){
    static int16 last_read;
    int16 this_read;
    int16 pulse;
    this_read = tmr_high;
```

```
// this_read=((int16)(timer()<<8));
pulse = last_read^this_read;
last_read = this_read;
return pulse;
}
```

Listing A22. ir.h

```
// ir.h

#ifndef IR_H
#define IR_H

#include <int_ext>
void ir_isr();
int1 IRhit();
int8 getIRCommand();
int16 getIRaddress();
void processIRCommand();

#endif
```

Listing A23. ir.c

```
#include "ir.h"
#include "LCD440s.h" // LCD function prototypes

// define IR remote button commands
// #define resetb 0x3f // start / stop
#define off 0x3f // start / stop
#define fastinc 0x47 // tele increment
#define fastdec 0xc7 // wide decrement
#define preciseinc 0x4f // counter reset
#define precisedec 0x77 // f start / stop
#define vmodeb 0x9f // tape return
#define imodeb 0x27 // on screen
#define incduty 0xbf // rewind
#define decduty 0xff // fast-forward
#define noregmode 0x17 // stop

#define MAXVSP 20
#define MAXISP 10
#define MINSP 0
#define FVINT 1 // about 1 V increment / decrement
#define FCINT 1 // about 1 A increment / decrement
const float PVINT = 0.1;
const float PCINT = 0.1;

struct IRFrameType
{
    int8 commandc; //command complement
    int8 command; //ir command
    int16 address; //ir address word
} IRFrame;

static int1 IRh; //tracks if a command has been received

#include <int_ext>
void ir_isr()
{
    //connect to ext interrupt 0
    static int32 lastTime;
    int32 deltaTime;
```

```

int1 databit;
static int8 recBit;

if ( IRh )
    return; //don't process another receive until last command is read

deltaTime = timer() - lastTime; //calc delta time (us)
lastTime = timer();             //update lasttime for next pass
// 1050 1675 2300
if ( ( deltaTime < 1050 ) || ( deltaTime > 2300 ) ) { //pulse not valid
    recBit = 0;
    return;
}

if ( deltaTime < 1675 )
    databit = 1;
else
    databit = 0;

shift_left( &IRFrame, 4, databit ); //shift bit into frame

if ( ( ++recBit ) == 32 ) { //entire frame received
    if ( ( IRFrame.command ^ IRFrame.commandc ) == 0xFF )
        IRh = TRUE;
}
}

int1 IRhit()
{
    return IRh;
}

int8 getIRCommand()
{
    IRh = FALSE; //clear received flag
    return IRFrame.command;
}

int16 getIRAddress()
{
    return IRFrame.address;
}

// The system must be in local control for this function to be executed.
// This prevents the user from changing settings when in PC / battery-charging mode.
// Note: the checks for voltage affecting current and current affecting voltage have not
//       yet been implemented.

void processIRCommand()
{
    // If a button on the IR remote is hit...take action
    int8 cmd;
    cmd = getIRCommand();

    //printf( lcd_putc2, "Cmd: %X\n", cmd );

    switch ( cmd ) {
        case off:
            setpoint = MINSP;
            duty = 0;
            setLED( NONE );
            break;
    }
}

```

```

case vmodeb: // select voltage mode button
    // check present state of the system
    if ( mode != VOLTAGE ) { // the system must be in CURRENT mode
        mode = VOLTAGE; // change to voltage mode
        setpoint = MINSP; // I think this would be most safe
    }
    break;
case imodeb: // select current mode button
    // check present state of the system
    if ( mode != CURRENT ) { // the system must be in VOLTAGE mode
        mode = CURRENT; // change to current mode
        setpoint = MINSP; // I think this would be most safe
    }
    break;
case fastinc: // select fast increment button
    // store voltage
    if ( mode == VOLTAGE )
        if ( ( setpoint + FVINT ) >= MAXVSP ) // check for overvoltage request
            setpoint = MAXVSP; // saturates at 20 V
        else
            setpoint += FVINT; // about 1 V increment
    else if ( mode == CURRENT )
        if ( ( setpoint + FCINT ) >= MAXISP ) // check for overcurrent request
            setpoint = MAXISP; // saturates at 10 A
        else
            setpoint += FCINT; // about 1 A increment
    break;
case fastdec: // select fast decrement button
    if ( mode == VOLTAGE )
        if ( setpoint <= FVINT ) // check for undervoltage request
            setpoint = MINSP;
        else
            setpoint -= FVINT; // about 1 V decrement
    else if ( mode == CURRENT )
        if ( setpoint <= FCINT ) // check for undercurrent request
            setpoint = MINSP;
        else
            setpoint -= FCINT; // about 1 A decrement
    break;
case preciseinc: // select precise increment button
    // store voltage
    if ( mode == VOLTAGE )
        if ( ( setpoint + PVINT ) >= MAXVSP ) // check for overvoltage request
            setpoint = MAXVSP; // saturates at 20 V
        else
            setpoint += PVINT; // about 0.1 V increment
    else if ( mode == CURRENT )
        if ( ( setpoint + PCINT ) >= MAXISP ) // check for overcurrent request
            setpoint = MAXISP; // saturates at 10 A
        else
            setpoint += PCINT; // about 0.1 A increment
    break;
case precisec: // select precise decrement button
    if ( mode == VOLTAGE )
        if ( setpoint <= PVINT ) // check for undervoltage request
            setpoint = MINSP;
        else
            setpoint -= PVINT; // about 1 V decrement
    else if ( mode == CURRENT )
        if ( setpoint <= PCINT ) // check for undercurrent request
            setpoint = MINSP;
        else

```

```

        setpoint -= PCINT; // about 1 A decrement
    break;
case incduty:
    if ( duty >= 0 && duty <= 975 )
        duty += 25;
    break;
case decduty:
    if ( duty >= 25 && duty <= 1023 )
        duty -= 25;
    break;
case noregmode:
    mode = NOREG;
    break;
default:
    break;
} // end switch
} //end of irHit

```

PC Software Application Code:

Listing A24. Main Form Code

```

Option Explicit
'form scope variable declarations
Dim ps As PowerSupply
Dim WithEvents PSdrv As PSDeviceDriver
Dim graphobj As Graph

Private Sub chkAutoSample_Click()
If chkAutoSample.value = 1 Then
    PSdrv.PeriodicSampling (True)
Else
    PSdrv.PeriodicSampling (False)
End If
End Sub

Private Sub cmdDispAmbTemp_Click()
    graphobj.GraphParamater = sAmbientTemperature
    graphobj.DispChart
End Sub

Private Sub cmdDispBatTemp_Click()
    graphobj.GraphParamater = sBatteryTemperature
    graphobj.DispChart
End Sub

Private Sub cmdDispCurrent_Click()
    graphobj.GraphParamater = sCurrent
    graphobj.DispChart
End Sub

Private Sub cmdDispVoltage_Click()
    graphobj.GraphParamater = sVoltage
    graphobj.DispChart
End Sub

Private Sub cmdGetPWM_Click()
    lblPWMvalue = PSdrv.getPWMvalue

```

```

End Sub

Private Sub cmdGetStatus_Click()
    lblPWMvalue = PSdrv.getPWMvalue

End Sub

Private Sub cmdDeviceControl_Click()
ps.SetPCControl (Not ps.Control)
updateControl
End Sub

Private Sub cmdOnOff_Click()
ps.SetOutputOn (Not ps.OutputOn)    'toggle output state
updateOnOff
End Sub

Private Sub updateOnOff()
    If ps.OutputOn Then
        cmdOnOff.Caption = "On"
    Else
        cmdOnOff.Caption = "Off"
    End If
End Sub

End Sub

Private Sub updateControl()
    If ps.Control Then
        cmdDeviceControl.Caption = "PC"
    Else
        cmdDeviceControl.Caption = "Local User"
    End If
End Sub

End Sub

Private Sub cmdReconnect_Click()
PSdrv.connect
End Sub

Private Sub cmdUpdate_Click()
'get a single sample
PSdrv.getSample
End Sub

Private Sub Form_Load()
Set ps = New PowerSupply
Set PSdrv = ps.PSDeviceDriver    'reference needed for event trig
Set graphobj = New Graph
Set graphobj.chart = Me.MSChart1    'pass reference to chart obj
Set graphobj.SampleData = ps.PSDeviceDriver.DataCol

graphobj.Chart_Initialize    'initialize chart
statbar.Panels(1).Text = "Not connected"
Call UpdateScrollBar
Call updateOnOff
Call updateControl
lblPWMvalue = ""    'clear label
End Sub

Private Sub UpdateScrollBar()

vsbOutput.Visible = False
If ps.Mode = VoltageRegulation Then
    vsbOutput.value = ps.voltage * 1000
    vsbOutput.Max = config.MinVoltage * 1000
    vsbOutput.Min = config.MaxVoltage * 1000

```

```

        txtOutput.Text = Format(ps.voltage, "##.000")
        lblOutText.Caption = "Volts"

Else    'current regulation
    vsbOutput.value = ps.current * 1000
    vsbOutput.Min = config.MaxCurrent * 1000
    vsbOutput.Max = config.MinCurrent * 1000
    txtOutput.Text = Format(ps.current, "##.000")
    lblOutText.Caption = "Amps"
End If
vsbOutput.Visible = True

End Sub

Private Sub imgOnOff_Click()
End Sub

Private Sub mnuAbout_Click()
frmAbout.Show

End Sub

Private Sub mnuCharging_Click()
frmCharge.Show
Set frmCharge.PSreference = ps

End Sub

Private Sub mnuConfigure_Click()
frmOptions.Show

End Sub

Private Sub mnuExit_Click()
    Unload Me
End Sub

Private Sub optMode_Click(Index As Integer)

If Index = 0 Then
    ps.Mode = VoltageRegulation
    '    ps.voltage = 0 'initialize voltage to 0 when changing modes
Else
    ps.Mode = CurrentRegulation
    '    ps.current = 0 'initialize current to 0 when changing modes
End If
Call UpdateScrollBar

End Sub

Private Sub Output_Click()
End Sub

Private Sub PSdrv_RecPacket(pacType As String)
'Dim dCol As scDataCol
Static SAMP As Sample
Select Case pacType
Case "r"
    Set SAMP = PSdrv.DataCol.LastItem 'set object reference to last data collected
    lblVoltage.Caption = "Voltage: " & Format(SAMP.voltage, "##.000") & " Volts"

```



```

    lblCurrent.Caption = "Current: " & Format(SAMP.current, "##.000") & " Amps"
    lblAmbTemp.Caption = "Ambient Temp " & Format(SAMP.ambTemp, "##.000") & " F"
    lblBatTemp.Caption = "Battery Temp " & Format(SAMP.batTemp, "##.000") & " F"
    graphobj.DispChart
Case "e"
    MsgBox ("Error code received")
Case "n"

End Select
End Sub

Private Sub txtOutput_KeyPress(KeyAscii As Integer)
If KeyAscii = Asc(vbCr) Then
    On Error GoTo handle 'if text box conversion to numeric fails, exit loop
    If ps.Mode = VoltageRegulation Then
        ps.voltage = txtOutput.Text
        'txtOutput.Text = Format(ps.voltage, "##.000")

    Else
        ps.current = txtOutput.Text
        'txtOutput.Text = Format(ps.current, "##.000")
    End If

handle:
    On Error GoTo 0 'reset error trap
    If ps.Mode = VoltageRegulation Then
        txtOutput.Text = Format(ps.voltage, "##.000")
    Else
        txtOutput.Text = Format(ps.current, "##.000")
    End If

    '    Call updateControl 'process device control mode change indication
    Call UpdateScrollBar

End If
End Sub

Private Sub vsbOutput_Change()

If ps.Mode = VoltageRegulation Then
    ps.voltage = vsbOutput.value / 1000
    txtOutput.Text = Format(ps.voltage, "##.000")

Else
    ps.current = vsbOutput.value / 1000
    txtOutput.Text = Format(ps.current, "##.000")
End If

Call updateControl 'process device control mode change indication
End Sub

Private Sub vsbOutput_Scroll()
    ' If ps.Mode = VoltageRegulation Then
    '     txtOutput.Text = vsbOutput.value / 1000

    ' Else
    '     txtOutput.Text = vsbOutput.value / 1000

End Sub

Private Sub Winsock1_Close()
statbar.Panels(1).Text = "Not connected"

```

```
End Sub
```

```
Private Sub Winsock1_Connect()  
statbar.Panels(1).Text = "Connected to " & Winsock1.RemoteHost & _  
" on port " & Winsock1.LocalPort
```

```
End Sub
```

Listing A25. Charge Form Code

```
Dim Charge As Charger  
Dim bat As IBattery
```

```
Private Sub cboChargeType_Click()  
'    Select Case cboChargeType.Text  
'    Case bat.ProfileName(  
    bat.ProfileNum = cboChargeType.ListIndex  
    txtStatus.Text = bat.ProfileDiscription(bat.ProfileNum)
```

```
End Sub
```

```
Private Sub cboChemistry_Click()  
If cboChemistry.enabled Then  
    Select Case cboChemistry.Text  
    Case "Lead Acid"  
        Charge.BatType = LA  
    Case "Sealed Lead Acid"  
        Charge.BatType = SLA  
    Case "NiCad"  
        Charge.BatType = NiCd  
    Case "NiMh"  
        Charge.BatType = NiMHyd  
    Case "Li Ion"  
        Charge.BatType = LithIon  
    Case Else  
        Call MsgBox("Must select chemistry", vbExclamation)  
        Exit Sub  
    End Select  
    Set bat = Charge.Battery 'get the battery reference  
    cmdBeginCharge.enabled = True  
    cboChargeType.enabled = True
```

```
    Call UpdateFields  
End If  
End Sub
```

```
Private Sub cboNumCells_Click()  
If cboNumCells.enabled = True And cboNumCells.ListIndex > 0 Then  
    bat.cells = CInt(cboNumCells.Text)  
    Call UpdateFields  
End If  
End Sub
```

```
Private Sub cmdGetProfiles_Click()  
Dim errMessage As String  
Dim cells As Integer  
Dim ah As Integer  
On Error GoTo handle  
'setup possible error message, if type conversion fails handle will print message  
errMessage = "Cells must be an integer value greater than 0"
```

```

cells = txtCells.Text
If cells > 0 Then bat.cells = cells

errMessage = "Amp hour capacity must be a number greater than 0"
ah = ahcap.Text
If cells > 0 Then bat.cells = cells

On Error GoTo 0
Exit Sub      'dont execute error handle routine

handle:
    Call MsgBox(errMessage, vbExclamation)
On Error GoTo 0 'reset error handling
End Sub

Private Sub cmdBeginCharge_Click()
Static start As Boolean
'Set bat.PowerSupplyRef = Charge.PowerSupplyObj
start = Not start

If Charge.IsCharging Then
    Charge.StatusMsg ("Charge canceled by user")
    Charge.CancelCharge
Else
    Charge.StatusMsg ("Beginning Battery Validation Sequence")
    Charge.BeginCharge
End If

If Charge.IsCharging Then
    cmdBeginCharge.Caption = "Cancel Charge"
Else
    cmdBeginCharge.Caption = "Begin Charge"
End If

End Sub

Private Sub Form_Load()
Set Charge = New Charger
'Set bat = charge.Battery      'get battery object reference
cboChemistry.enabled = False
cboChemistry.AddItem ("Select Chemistry")
cboChemistry.AddItem ("Lead Acid")
cboChemistry.AddItem ("Sealed Lead Acid")
cboChemistry.AddItem ("NiCad")
cboChemistry.AddItem ("NiMh")
cboChemistry.AddItem ("Li Ion")
cboChemistry.ListIndex = 0
cboChemistry.enabled = True
txtAHcap.Text = ""
txtCellV.Text = ""
txtBatteryV.Text = ""
txtStatus.Text = ""
txtChargeRate = ""
txtChargeTime = ""

lblOutput = ""
lblVoltage = ""
lblCurrent = ""
lblAmbTemp = ""
lblBatTemp = ""
lblElapTime = ""

```

```

lbldVdt = ""
lbldTdt = ""
lbldIdt = ""

End Sub

Property Set PSreference(psref As PowerSupply)
    Set Charge.PowerSupply = psref
End Property

Private Sub UpdateFields()
txtAHcap.Text = bat.mAhCapacity
txtCellV.Text = bat.CellVoltage
txtBatteryV.Text = bat.BatteryVoltage
txtChargeRate.Text = bat.chargeRate
txtChargeTime.Text = bat.chargeTime

cboNumCells.enabled = False 'disable control events
cboNumCells.Clear
Call cboNumCells.AddItem("Cells", 0)
For loop1 = 1 To config.MaxVoltage / bat.CellVoltage
    Call cboNumCells.AddItem(CStr(loop1), loop1)
Next loop1
cboNumCells.ListIndex = bat.cells
cboNumCells.enabled = True

cboChargeType.enabled = False
cboChargeType.Clear
Call cboChargeType.AddItem("Select Charge Type")
For loop1 = 1 To bat.MaxProfiles
    Call cboChargeType.AddItem(bat.ProfileName(loop1), loop1)
Next loop1
cboChargeType.ListIndex = bat.ProfileNum
cboChargeType.enabled = True

End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
'Set bat.DataAnalyzerRef = Nothing
'Set bat.PowerSupplyRef = Nothing
'Set bat.ChargerRef = Nothing
Set bat = Nothing
'Set Charge.DataAnalyzer = Nothing
'Set Charge.PowerSupply = Nothing
Set Charge.Battery = Nothing
Charge.CancelCharge

Set Charge = Nothing
End Sub

Private Sub txtAHcap_Change()
On Error GoTo handle
    bat.mAhCapacity = CSng(txtAHcap)

handle: 'skip update
On Error GoTo 0 'reset error trap
End Sub

Private Sub txtChargeRate_Change()
On Error GoTo handle
    bat.chargeRate = CSng(txtChargeRate)

```

```

handle: 'skip update
On Error GoTo 0 'reset error trap

End Sub

Private Sub txtChargeTime_Change()
On Error GoTo handle
    bat.chargeTime = CSng(txtChargeTime)

handle: 'skip update
On Error GoTo 0 'reset error trap

End Sub

```

Listing A26. Options Form Code

```

Option Explicit
Public listind As Integer

Private Sub cmdApply_Click()
    Call updateSettings
End Sub

Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub cmdOK_Click()
    If cmdApply.enabled = True Then
        Call updateSettings
    End If
    Unload Me
End Sub

Private Sub lblcboDestDefault_Click()

End Sub

Private Sub Form_Load()
Dim loop1 As Integer
For loop1 = 0 To config.NumberOfSettings - 1
    If loop1 > 0 Then
        Load lblPar(loop1)
        Load txtPar(loop1)
        txtPar(loop1).Top = txtPar(loop1 - 1).Top + txtPar(loop1 - 1).Height
        lblPar(loop1).Top = txtPar(loop1).Top
    End If
    lblPar(loop1).Caption = config.configName(loop1)
    txtPar(loop1).Text = config.configValue(loop1)
    lblPar(loop1).Visible = True
    txtPar(loop1).Visible = True
Next loop1
cmdApply.enabled = False
cboSelectCom.AddItem ("Com port 1")
cboSelectCom.AddItem ("Com port 2")
cboSelectCom.AddItem ("TCP/IP bridge")
fraUserDef.Caption = ""
End Sub

```

```

Private Sub updateSettings()
Dim loop1 As Integer

For loop1 = 0 To (txtPar.Count - 1)
    config.configValue(loop1) = txtPar.Item(loop1).Text
    txtPar.Item(loop1).Text = config.configValue(loop1) 'display current values after
filtering
Next loop1
config.SaveSettings

cmdApply.enabled = False

End Sub

Private Sub txtPar_Change(Index As Integer)
    cmdApply.enabled = True

End Sub

```

Listing A27. Form About Code

```

Option Explicit

' Reg Key Security Options...
Const READ_CONTROL = &H20000
Const KEY_QUERY_VALUE = &H1
Const KEY_SET_VALUE = &H2
Const KEY_CREATE_SUB_KEY = &H4
Const KEY_ENUMERATE_SUB_KEYS = &H8
Const KEY_NOTIFY = &H10
Const KEY_CREATE_LINK = &H20
Const KEY_ALL_ACCESS = KEY_QUERY_VALUE + KEY_SET_VALUE + _
    KEY_CREATE_SUB_KEY + KEY_ENUMERATE_SUB_KEYS + _
    KEY_NOTIFY + KEY_CREATE_LINK + READ_CONTROL

' Reg Key ROOT Types...
Const HKEY_LOCAL_MACHINE = &H80000002
Const ERROR_SUCCESS = 0
Const REG_SZ = 1 ' Unicode nul terminated string
Const REG_DWORD = 4 ' 32-bit number

Const gREGKEYSYSINFOLOC = "SOFTWARE\Microsoft\Shared Tools Location"
Const gREGVALSYSINFOLOC = "MSINFO"
Const gREGKEYSYSINFO = "SOFTWARE\Microsoft\Shared Tools\MSINFO"
Const gREGVALSYSINFO = "PATH"

Private Declare Function RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey As
Long, ByVal lpSubKey As String, ByVal ulOptions As Long, ByVal samDesired As Long, ByRef
phkResult As Long) As Long
Private Declare Function RegQueryValueEx Lib "advapi32" Alias "RegQueryValueExA" (ByVal
hKey As Long, ByVal lpValueName As String, ByVal lpReserved As Long, ByRef lpType As
Long, ByVal lpData As String, ByRef lpcbData As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32" (ByVal hKey As Long) As Long

Private Sub cmdSysInfo_Click()
    Call StartSysInfo
End Sub

Private Sub cmdOK_Click()
    Unload Me

```

End Sub

Private Sub Form_Load()

```
Me.Caption = "About " & App.Title
lblVersion.Caption = "Version " & App.Major & "." & App.Minor & "." & App.Revision
lblTitle.Caption = App.Title
lblDescription.Caption = "By Steven Savage 12/20/04" & vbCrLf & vbCrLf & _
"This application allows control of the smart charger power supply" & _
" and charges batteries if desired."
lblDisclaimer.Caption = "This software cannot be copied or" & vbCrLf & _
"modified without permission of the author!"
picIcon.Picture = frmMain.Icon 'display the application icon
```

End Sub

Public Sub StartSysInfo()

On Error GoTo SysInfoErr

Dim rc As Long

Dim SysInfoPath As String

' Try To Get System Info Program Path\Name From Registry...

If GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFO, gREGVALSYSINFO, SysInfoPath) Then

' Try To Get System Info Program Path Only From Registry...

ElseIf GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFOLOC, gREGVALSYSINFOLOC, SysInfoPath) Then

' Validate Existence Of Known 32 Bit File Version

If (Dir(SysInfoPath & "\MSINFO32.EXE") <> "") Then

SysInfoPath = SysInfoPath & "\MSINFO32.EXE"

' Error - File Can Not Be Found...

Else

GoTo SysInfoErr

End If

' Error - Registry Entry Can Not Be Found...

Else

GoTo SysInfoErr

End If

Call Shell(SysInfoPath, vbNormalFocus)

Exit Sub

SysInfoErr:

MsgBox "System Information Is Unavailable At This Time", vbOKOnly

End Sub

Public Function GetKeyValue(KeyRoot As Long, KeyName As String, SubKeyRef As String, ByRef KeyVal As String) As Boolean

Dim i As Long

' Loop Counter

Dim rc As Long

' Return Code

Dim hKey As Long

' Handle To An Open Registry

Key

Dim hDepth As Long

'

Dim KeyValType As Long

' Data Type Of A Registry Key

Dim tmpVal As String

' Tempory Storage For A

Registry Key Value

Dim KeyValSize As Long

' Size Of Registry Key

Variable

```
' -----
' Open RegKey Under KeyRoot {HKEY_LOCAL_MACHINE...}
' -----
```

rc = RegOpenKeyEx(KeyRoot, KeyName, 0, KEY_ALL_ACCESS, hKey) ' Open Registry Key

```

If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError          ' Handle Error...

tmpVal = String$(1024, 0)                               ' Allocate Variable Space
KeyValSize = 1024                                       ' Mark Variable Size

'-----
' Retrieve Registry Key Value...
'-----
rc = RegQueryValueEx(hKey, SubKeyRef, 0, _
                    KeyValType, tmpVal, KeyValSize)      ' Get/Create Key Value

If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError          ' Handle Errors

If (Asc(Mid(tmpVal, KeyValSize, 1)) = 0) Then           ' Win95 Adds Null Terminated
String...
    tmpVal = Left(tmpVal, KeyValSize - 1)              ' Null Found, Extract From
String
    Else                                               ' WinNT Does NOT Null
Terminate String...
    tmpVal = Left(tmpVal, KeyValSize)                  ' Null Not Found, Extract
String Only
End If
'-----
' Determine Key Value Type For Conversion...
'-----
Select Case KeyValType                                ' Search Data Types...
Case REG_SZ                                           ' String Registry Key Data
Type
    KeyVal = tmpVal                                    ' Copy String Value
Case REG_DWORD                                        ' Double Word Registry Key
Data Type
    For i = Len(tmpVal) To 1 Step -1                  ' Convert Each Bit
        KeyVal = KeyVal + Hex(Asc(Mid(tmpVal, i, 1))) ' Build Value Char. By Char.
    Next
    KeyVal = Format$("&h" + KeyVal)                    ' Convert Double Word To
String
End Select

GetKeyValue = True                                     ' Return Success
rc = RegCloseKey(hKey)                                ' Close Registry Key
Exit Function                                          ' Exit

GetKeyError:                                          ' Cleanup After An Error Has Occured...
    KeyVal = ""                                       ' Set Return Val To Empty
String
    GetKeyValue = False                               ' Return Failure
    rc = RegCloseKey(hKey)                            ' Close Registry Key
End Function

```

Listing A28. Charger Class

```

Option Explicit
'A charging algorithm must implement the Ibattery interface, see Ibattery for
implementation details
'To initialize Ibattery
'1. create Ibattery object
'2. set Ibattery ps to power supply object
'3. create dataAnalyzer object
'4. set dataAnalyzer.datacollection reference to current data collection
'5. set Ibattery.dataAnalyzer reference to dataAnalyzer created in 4

'Charger maintains a high level control of charging, controls if charging alg is run

```



```

'or not
Private mvarDataAnalyzer As DataAnalyzer
Public Enum BatteryType
    LA = 1
    SLA = 2
    NiCd = 3
    NiMHyd = 4
    LithIon = 5
End Enum

Public Enum CBMSGTYPE
    ChargeAborted = 1
    MajorFault = 2
    Warning = 3
End Enum

'local variable(s) to hold property value(s)
Private mvarPowerSupply As PowerSupply 'local copy
Private WithEvents PSDdrv As PSDeviceDriver
Private mvarBatType As BatteryType 'local copy
Private mvarBattery As IBattery
Private startTime As Date
Private charging As Boolean
Private mvarSealedLeadAcid As SealedLeadAcid

Public Property Get SealedLeadAcid() As SealedLeadAcid
    If mvarSealedLeadAcid Is Nothing Then
        Set mvarSealedLeadAcid = New SealedLeadAcid
    End If

    Set SealedLeadAcid = mvarSealedLeadAcid
End Property

Public Property Set SealedLeadAcid(vData As SealedLeadAcid)
    Set mvarSealedLeadAcid = vData
End Property

Public Property Get Battery() As IBattery
    Set Battery = mvarBattery
End Property

Public Property Set Battery(vData As IBattery)
    Set mvarBattery = vData
End Property

Public Property Let BatType(ByVal vData As BatteryType)
'used when assigning an Object to the property, on the left side of a Set statement.
'Syntax: Set x.BatType = Form1
Dim da As DataAnalyzer
Select Case vData 'set the battery object to a specific implementation
    Case Is = LA
        Set mvarBattery = New LeadAcid
    Case Is = SLA
        Set mvarBattery = New SealedLeadAcid
    Case Is = NiCd
        Set mvarBattery = New NiCad

```

```

    Case Is = NiMHyd
        Set mvarBattery = New NiMh
    Case Is = LithIon
        Set mvarBattery = New LiIon
End Select
    Set da = New DataAnalyzer
    Set da.DataSet = mvarPowerSupply.PSDeviceDriver.DataCol
    Set mvarBattery.DataAnalyzerRef = da
    Set mvarBattery.PowerSupplyRef = mvarPowerSupply
    Set mvarBattery.ChargerRef = Me
    Set PSDdrv = mvarPowerSupply.PSDeviceDriver
End Property

Public Property Get BatType() As BatteryType
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.BatType
    BatType = mvarBatType
End Property

Public Property Set PowerSupply(ByVal vData As PowerSupply)
'used when assigning an Object to the property, on the left side of a Set statement.
'Syntax: Set x.PowerSupplyObj = Form1
    Set mvarPowerSupply = vData
End Property

Public Property Get DataAnalyzer() As DataAnalyzer
    If mvarDataAnalyzer Is Nothing Then
        Set mvarDataAnalyzer = New DataAnalyzer
    End If

    Set DataAnalyzer = mvarDataAnalyzer
End Property

Public Sub BeginCharge()
    If (mvarBattery.Validate) Then 'validate battery and settings
        PSDdrv.PeriodicSampling (True) 'start sampling
        startTime = Now
        charging = True
    End If
End Sub

Public Function IsCharging() As Boolean
    IsCharging = charging
End Function

Public Sub CancelCharge()
    mvarPowerSupply.SetOutputOn (False)
    PSDdrv.PeriodicSampling (False) 'stop sampling
    charging = False
End Sub

Private Sub Class_Terminate()
    Set mvarSealedLeadAcid = Nothing

    Set mvarBattery = Nothing
    Set mvarDataAnalyzer = Nothing
End Sub

```

```

Public Sub CBfromBattery(reason As CBMSGTYPE)
'call back from Ibattery
If reason = ChargeAborted Then
    frmCharge.cmdBeginCharge.Caption = "Begin Charge"    'reset label
    CancelCharge
End If

End Sub

Private Sub PSDdrv_RecPacket(pacType As String)
Static SAMP As Sample

If pacType = "r" Then
    If charging Then mvarBattery.Charge
    Set SAMP = PSDdrv.DataCol.LastItem    'set object refrence to last data collected
    With frmCharge
        .lblVoltage.Caption = "Voltage: " & Format(SAMP.voltage, "##.000") & " Volts"
        .lblCurrent.Caption = "Current: " & Format(SAMP.current, "##.000") & " Amps"
        .lblAmbTemp = "Ambient Temp " & Format(SAMP.ambTemp, "##.00") & " F"
        .lblBatTemp = "Battery Temp " & Format(SAMP.batTemp, "##.00") & " F"
        .lblElapTime = "Elapsed Time " & Format(Now - startTime, "hh:mm:ss")
    End With
End If
End Sub

Public Sub StatusMsg(msg As String, Optional ByVal arg As Double = -1, Optional ByVal
msgtrailer As String = "")
With frmCharge.txtStatus
    If arg = -1 Then
        .Text = Time & " " & msg & vbCrLf & .Text
    Else
        .Text = Time & " " & msg & " " & Format(arg, "#0.00") & " " & msgtrailer & vbCrLf
    End If
End With
End Sub

```

Listing A29. Configuration Class

```

Option Explicit
Const APPNAME As String = "Smart Charger" 'used with registry set/get
Const SECTION As String = "Settings"
Private Type ConfigType
    name As String
    value As String
End Type

Private mvarconfig() As ConfigType 'dynamic array of config objects
Private mvarPort As Integer 'local copy
Private mvarPSupdate As Integer 'local copy
Private mvarMaxCurrent As Single 'local copy
Private mvarMinCurrent As Single 'local copy
Private mvarMaxVoltage As Single 'local copy
Private mvarMinVoltage As Single 'local copy
Private mvarMaxTemperature As Single 'local copy
Private mvarMinTemperature As Single 'local copy
Private mvarServer As String 'local copy
'local variable(s) to hold property value(s)
Private mvarRetries As Integer 'local copy
Private mvarRetryDelayTime As Integer 'local copy

```

```

Public Property Get RetryDelayTime() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.RetryDelayTime
    RetryDelayTime = mvarRetryDelayTime
End Property

Public Property Get Retries() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Retries
    Retries = mvarRetries
End Property

Public Property Get Server() As String
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Server
    Server = mvarServer
End Property

Public Property Get MinTemperature() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.MinTemperature
    MinTemperature = mvarMinTemperature
End Property

Public Property Get MaxTemperature() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.MaxTemperature
    MaxTemperature = mvarMaxTemperature
End Property

Public Property Get Port() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.ComPort
    Port = mvarPort
End Property

Public Property Get PSupdate() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.PSupdate
    PSupdate = mvarPSupdate
End Property

Public Property Get MaxCurrent() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.MaxCurrent
    MaxCurrent = mvarMaxCurrent
End Property

Public Property Get MinCurrent() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.PSupdate
    MinCurrent = mvarMinCurrent
End Property

Public Property Get MaxVoltage() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.MaxVoltage
    MaxVoltage = mvarMaxVoltage
End Property

```

```

Public Property Get MinVoltage() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.PSupdate
    MinVoltage = mvarMinVoltage

End Property

Public Sub SaveSettings()
    Dim name As String
    Dim value As String
    Dim i As Integer

    For i = 0 To NumberOfSettings - 1
        name = mvarconfig(i).name
        value = mvarconfig(i).value
        SaveSetting APPNAME, SECTION, name, value
    Next i

End Sub

Public Sub LoadSettings()
End Sub

Public Property Get NumberOfSettings() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.NumberOfSettings
    NumberOfSettings = UBound(mvarconfig) + 1
End Property

Private Sub Class_Initialize()
'create the mconfig object when the Configuration class is created

    Dim defaultValue As String    'default parameter value
    Dim name As String            'default parameter name
    Dim i As Integer              'index variable

    i = 0
    ReDim Preserve mvarconfig(i)
    name = "Maximum voltage"
    defaultValue = "20"
    mvarconfig(i).name = name
    mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
    mvarMaxVoltage = CSng(mvarconfig(i).value)

    i = 1
    ReDim Preserve mvarconfig(i)
    name = "Maximum current"
    defaultValue = "10"
    mvarconfig(i).name = name
    mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
    mvarMaxCurrent = CSng(mvarconfig(i).value)

    i = 2
    ReDim Preserve mvarconfig(i)
    name = "Minimum voltage"
    defaultValue = "0"
    mvarconfig(i).name = name
    mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
    mvarMinVoltage = CSng(mvarconfig(i).value)

    i = 3
    ReDim Preserve mvarconfig(i)

```

```

name = "Minimum current"
defaultValue = "0"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarMinCurrent = CSng(mvarconfig(i).value)

i = 4
ReDim Preserve mvarconfig(i)
name = "PS update rate(ms)"
defaultValue = "1000"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarPSupdate = CInt(mvarconfig(i).value)

i = 5
ReDim Preserve mvarconfig(i)
name = "TCP IP Port"
defaultValue = "107"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarPort = CInt(mvarconfig(i).value)

i = 6
ReDim Preserve mvarconfig(i)
name = "Maximum Temperature"
defaultValue = "500"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarMaxTemperature = CSng(mvarconfig(i).value)

i = 7
ReDim Preserve mvarconfig(i)
name = "Minimum Temperature"
defaultValue = "1"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarMinTemperature = CSng(mvarconfig(i).value)

i = 8
ReDim Preserve mvarconfig(i)
name = "Server"
'defaultValue = "130.101.12.42"
defaultValue = "127.0.0.1"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarServer = mvarconfig(i).value

i = 9
ReDim Preserve mvarconfig(i)
name = "Retries"
defaultValue = "3"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarRetries = mvarconfig(i).value

i = 10
ReDim Preserve mvarconfig(i)
name = "Retry delay time(ms)"
defaultValue = "1000"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarRetryDelayTime = mvarconfig(i).value

```

```

End Sub
Public Property Let configValue(Index As Integer, arg As String)
'this procedure allow update of paramaters with error checking
'must synchronize this procedure with the initialize procedure
On Error GoTo handle
    If Index < 0 Or Index > NumberOfSettings Then GoTo handle
    Dim cname As String
    cname = configName(Index)
    Dim test As Variant
    Select Case cname
    Case "Maximum voltage"
        test = CSng(arg) 'test if argument will cast to single
        'vsb limited to +-32757, scroll bar set at test*1000
        If test < 0 Or test > 32 Then GoTo handle 'drop out if less than 0
        mvarMaxVoltage = test
        mvarconfig(Index).value = arg
    Case "Maximum current"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Or test > 32 Then GoTo handle 'drop out if less than 0
        mvarMaxCurrent = test
        mvarconfig(Index).value = arg
    Case "Minimum voltage"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Or test > 32 Then GoTo handle 'drop out if less than 0
        mvarMinVoltage = test
        mvarconfig(Index).value = arg
    Case "Minimum current"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Or test > 32 Then GoTo handle 'drop out if less than 0
        mvarMinCurrent = test
        mvarconfig(Index).value = arg
    Case "PS update rate(ms)"
        test = CInt(arg) 'test if argument will cast to int
        If test < 0 Or test > 10000 Then GoTo handle
        mvarPSupdate = test
        mvarconfig(Index).value = arg
    Case "TCP IP Port"
        test = CInt(arg) 'test if argument will cast to int
        If test < 0 Or test > 60000 Then GoTo handle 'what is the high limit on a port?
        mvarPort = test
        mvarconfig(Index).value = arg
    Case "Maximum Temperature"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Then GoTo handle
        mvarMaxTemperature = test
        mvarconfig(Index).value = arg
    Case "Minimum Temperature"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Then GoTo handle
        mvarMinTemperature = test
        mvarconfig(Index).value = arg
    Case "Server"
        ' **** how do you validate a server argument?
        ' test = CInt(arg) 'test if argument will cast to integer
        ' If test < 0 Then GoTo handle
        mvarconfig(Index).value = arg
        mvarServer = arg
    Case "Retries"
        test = CInt(arg) 'test if argument will cast to integer
        If test < 0 Then GoTo handle
        mvarconfig(Index).value = arg

```

```

        mvarRetries = test

    Case "Retry delay time(ms)"
        test = CInt(arg) 'test if argument will cast to integer
        If test < 0 Then GoTo handle
        mvarconfig(Index).value = arg
        mvarRetryDelayTime = test

    End Select
handle:

End Property

Public Property Get configName(Index As Integer) As String

    Let configName = mvarconfig(Index).name
End Property
Public Property Get configValue(Index As Integer) As String

    Let configValue = mvarconfig(Index).value
End Property

```

Listing A30. Data Analyzer Class

Data Analyzer Class

```

Option Explicit
Private mvarData As scDataCol 'local copy

Public Property Set DataSet(ByRef vData As scDataCol)
'used when assigning an Object to the property, on the left side of a Set statement.
'Syntax: Set x.DataSet = Form1
    Set mvarData = vData
End Property

Public Property Get DataSet() As Data
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.DataSet
    Set DataSet = mvarData
End Property

Public Function getTemperature() As Single
'return the latest battery temperature

    getTemperature = mvarData.LastItem.batTemp
End Function

Public Function getDeltaTemperature() As Single
'return the latest battery temperature
    getDeltaTemperature = mvarData.LastItem.batTemp - mvarData.LastItem.ambTemp
End Function
Public Function getAvgDeltaTemperature(t As Integer) As Single
'return the latest battery temperature

    getAvgDeltaTemperature = getAvgBatTemp(t) - getAvgAmbTemp(t)
End Function

Public Function getAmbientTemperature() As Single
'return the latest ambient temperture
    getAmbientTemperature = mvarData.LastItem.ambTemp
End Function

```



```

Public Function getVoltage() As Single
    getVoltage = mvarData.LastItem.voltage
End Function

Public Function getAvgAmbTemp(t As Integer) As Single
Dim startIndex As Integer
Dim endIndex As Integer
Dim temp() As Single
Dim loop1 As Integer

startIndex = getSampleIndex(t) 'get index of sample within t seconds ago
endIndex = mvarData.Count

ReDim temp(startIndex To endIndex)

For loop1 = startIndex To endIndex
    temp(loop1) = mvarData(loop1).ambTemp
Next loop1

getAvgAmbTemp = mean(temp)
End Function

Public Function getAvgBatTemp(t As Integer) As Single
Dim startIndex As Integer
Dim endIndex As Integer
Dim temp() As Single
Dim loop1 As Integer

startIndex = getSampleIndex(t) 'get index of sample within t seconds ago
endIndex = mvarData.Count

ReDim temp(startIndex To endIndex)

For loop1 = startIndex To endIndex
    temp(loop1) = mvarData(loop1).batTemp
Next loop1

getAvgBatTemp = mean(temp)
End Function

Public Function getAvgVoltage(t As Integer) As Single
Dim startIndex As Integer
Dim endIndex As Integer
Dim v() As Single
Dim loop1 As Integer

startIndex = getSampleIndex(t) 'get index of sample within t seconds ago
endIndex = mvarData.Count

ReDim v(startIndex To endIndex)

For loop1 = startIndex To endIndex
    v(loop1) = mvarData(loop1).voltage
Next loop1

getAvgVoltage = mean(v)
End Function

Public Function getCurrent() As Single
    getCurrent = mvarData.LastItem.current
End Function

```

```

Public Function getdVdt() As Single
'this function returns dV/dt in units mV/min
Dim loop1 As Integer
Dim lastIndex As Integer
Dim v() As Single
Dim t() As Single

Dim startIndex As Integer
Dim endIndex As Integer
startIndex = getSampleIndex(60)
endIndex = mvarData.Count

ReDim v(startIndex To endIndex)
ReDim t(startIndex To endIndex)

For loop1 = startIndex To endIndex
    v(loop1) = mvarData(loop1).voltage
    t(loop1) = mvarData(loop1).timeStamp
Next loop1
getdVdt = leastSquaresSlope(t, v) * 60000 'scale to mv/min
End Function

Public Function getdIdt() As Single
'this function returns dI/dt in units mA/min
Dim loop1 As Integer
Dim lastIndex As Integer
Dim i() As Single
Dim t() As Single

Dim startIndex As Integer
Dim endIndex As Integer

startIndex = getSampleIndex(60)
endIndex = mvarData.Count

ReDim i(startIndex To endIndex)
ReDim t(startIndex To endIndex)

For loop1 = startIndex To endIndex
    i(loop1) = mvarData(loop1).current
    t(loop1) = mvarData(loop1).timeStamp
Next loop1
getdIdt = leastSquaresSlope(t, i) * 60000 'scale to mA/min
End Function

Public Function getdTdt() As Single
'this function returns dT/dt of the battery in units degrees F/min
Dim loop1 As Integer
Dim lastIndex As Integer
Dim degT() As Single
Dim t() As Single

Dim startIndex As Integer
Dim endIndex As Integer

startIndex = getSampleIndex(60)
endIndex = mvarData.Count

ReDim degT(startIndex To endIndex)

```

```

ReDim t(startIndex To endIndex)

For loop1 = startIndex To endIndex
    degT(loop1) = mvarData(loop1).batTemp
    t(loop1) = mvarData(loop1).timeStamp
Next loop1
getdTdt = leastSquaresSlope(t, degT) * 60 'scale to F/min
End Function

Private Function leastSquaresSlope(x As Variant, y As Variant) As Single

Dim xmean As Single
Dim ymean As Single
Dim xmxmean As Variant
Dim ymymean As Variant
Dim sumx2 As Single
Dim sumxy As Single

xmean = mean(x)
ymean = mean(y)
xmxmean = vectorMinusScaler(x, xmean)
ymymean = vectorMinusScaler(y, ymean)
sumx2 = vectorMultiply(xmxmean, xmxmean)
sumxy = vectorMultiply(ymymean, xmxmean)
If sumx2 = 0 Then
    leastSquaresSlope = 0
Else
    leastSquaresSlope = sumxy / sumx2
End If
End Function

Private Function getSampleIndex(t As Integer) As Integer
'returns the index of the last sample that occurred within t seconds from the last sample

'determine a threshold time
Dim loop1 As Integer
Dim threstime As Single

threstime = mvarData.LastItem.timeStamp - t
'determine the last sample index that satisfies the time criteria
For loop1 = mvarData.Count To 1 Step -1
    If mvarData(loop1).timeStamp < threstime Then Exit For
Next loop1
If loop1 < 1 Then loop1 = 1
getSampleIndex = loop1
End Function

Private Function mean(arg As Variant) As Single
'returns the mean of arg array
Dim tot As Double
Dim loop1 As Integer

tot = 0
For loop1 = LBound(arg) To UBound(arg)
    tot = tot + arg(loop1)
Next loop1
mean = tot / (UBound(arg) - LBound(arg) + 1)
End Function

Private Function vectorMinusScaler(varg As Variant, sarg As Single) As Variant
'this function subtracts a scalar from a vector and returns a vector
Dim loop1 As Integer

```

```

ReDim temp(LBound(varg) To UBound(varg)) As Single

For loop1 = LBound(varg) To UBound(varg)
    temp(loop1) = varg(loop1) - sarg
Next loop1
vectorMinusScaler = temp
End Function

Private Function vectorMultiply(arg1 As Variant, arg2 As Variant) As Single
'this function multiplies two vectors (vectors must be same length)
Dim temp As Single
Dim loop1 As Integer
temp = 0
    For loop1 = LBound(arg1) To UBound(arg1)
        temp = temp + arg1(loop1) * arg2(loop1)
    Next loop1
vectorMultiply = temp
End Function

```

Listing A31. Event Logger Class

```

'local variable(s) to hold property value(s)
Private mvarEventFileName As String 'local copy
Private mvarWriteToFileThresh As Integer 'local copy
Private mvarListStorageSize As Integer 'local copy
Private fileBufCount As Integer
Private msgListCol As New Collection 'private storage of list items

Public Function GetMessage(Index As Integer) As String
    If Index > 0 And Index <= msgListCol.Count Then
        GetMessage = msgListCol.Item(Index)
    Else
        GetMessage = "Eventlog access error"
    End If
End Function

Public Property Let ListStorageSize(ByVal vData As Integer)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.ListStorageSize = 5
    If vData >= mvarWriteToFileThresh Then
        mvarListStorageSize = vData
    End If
End Property

Public Property Get ListStorageSize() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.ListStorageSize
    ListStorageSize = mvarListStorageSize
End Property

Public Property Get MaxIndex() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.MaxIndex
    MaxIndex = msgListCol.Count
End Property

Public Property Let WriteToFileThresh(ByVal vData As Integer)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.WriteToFileThresh = 5
    mvarWriteToFileThresh = vData
End Property

```

```

Public Property Get WriteToFileThresh() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.WriteToFileThresh
    WriteToFileThresh = mvarWriteToFileThresh
End Property

Public Property Let EventFileName(ByVal vData As String)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.EventFileName = 5
    mvarEventFileName = vData
End Property

Public Property Get EventFileName() As String
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.EventFileName
    EventFileName = mvarEventFileName
End Property

Public Sub AddMessage(msg As String)
Dim message As String
message = Date & " " & Time & " " & msg
msgListCol.Add (message)

'write to disk after mvarwritetofilethresh number of messages
fileBufCount = fileBufCount + 1
If fileBufCount = mvarWriteToFileThresh Then
    Call WriteFileBuf
End If

'delete from begining of list to maintain a list size of mvarliststoragesize
If msgListCol.Count > mvarListStorageSize Then
    msgListCol.Remove (1)
End If

End Sub

Private Sub WriteFileBuf() 'write file buffer items to disk
'build file name and path
'EventFile = App.Path & "\database\" & Format(Date, "mmm yy") & mvarEventFileName '
Returns month and year
EventFile = App.Path & Format(Date, "mmm yy") & mvarEventFileName ' Returns month and
year

Open EventFile For Append As #1

For loop1 = msgListCol.Count - fileBufCount + 1 To msgListCol.Count
    Print #1, msgListCol.Item(loop1)
Next loop1

fileBufCount = 0 'reset file buffer count
Close #1

End Sub

Private Sub Class_Initialize()
mvarEventFileName = "Events.Log"
mvarWriteToFileThresh = 100
mvarListStorageSize = 500
End Sub

Private Sub Class_Terminate()
    Call WriteFileBuf 'write any remaining file data to disk

```

End Sub

Listing A32. Graph Class

Option Explicit

```
'local variable(s) to hold property value(s)
Private mvarchart As MSChart 'local copy
'local variable(s) to hold property value(s)
Private mvarSampleData As scDataCol 'local copy
Private initTime As Single 'initial time
'local variable(s) to hold property value(s)
Private mvarGraphParamater As SampleFieldType 'local copy
Const MAXDATAPOINTS = 60

Public Property Let GraphParamater(ByVal vData As SampleFieldType)
'used when assigning an Object to the property, on the left side of a Set statement.
'Syntax: Set x.GraphParamater = Form1
    Dim label As String

    mvarGraphParamater = vData

    Select Case vData
        Case sVoltage
            label = "Volts"
        Case sCurrent
            label = "Amps"
        Case sAmbientTemperature
            label = "DegF Amb"
        Case Else
            label = "DegF Bat"
    End Select

    mvarchart.Plot.Axis(VtChAxisIdY).AxisTitle.Text = label
End Property

Public Property Get GraphParamater() As SampleFieldType
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.GraphParamater
    GraphParamater = mvarGraphParamater
End Property

Public Property Set SampleData(ByVal vData As scDataCol)
'used when assigning an Object to the property, on the left side of a Set statement.
'Syntax: Set x.SampleData = Form1
    Set mvarSampleData = vData
End Property

Public Property Get SampleData() As scDataCol
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.SampleData
    Set SampleData = mvarSampleData
End Property

Public Property Set chart(ByRef vData As MSChart)
'used when assigning an Object to the property, on the left side of a Set statement.
'Syntax: Set x.chart = Form1
    Set mvarchart = vData
End Property
```

```

Public Property Get chart() As MSChart
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.chart
    Set chart = mvarchart
End Property

Public Sub DispChart()

Dim loop1 As Integer
Dim col As Integer
Dim row As Integer
Dim dsize As Integer
Dim minRow As Integer
Dim maxRow As Integer

'col = 2                'add 1 col 0(1) is time stamp

maxRow = mvarSampleData.Count
minRow = maxRow - MAXDATAPOINTS
If minRow < 1 Then minRow = 1

row = maxRow - minRow + 1
With mvarchart

If initTime = -1 And mvarSampleData.Count > 0 Then
    initTime = mvarSampleData.Item(1).timeStamp
End If
dsize = mvarSampleData.Count

For loop1 = 0 To maxRow - minRow
    .row = loop1 + 1
    .Column = 1
    .Data = mvarSampleData.Item(loop1 + minRow).timeStamp - initTime
    .Column = 2
Select Case mvarGraphParamater
    Case sAmbientTemperature
        .Data = mvarSampleData.Item(loop1 + minRow).ambTemp
    Case sBatteryTemperature
        .Data = mvarSampleData.Item(loop1 + minRow).batTemp
    Case sVoltage
        .Data = mvarSampleData.Item(loop1 + minRow).voltage
    Case sCurrent
        .Data = mvarSampleData.Item(loop1 + minRow).current
End Select
Next loop1
End With
End Sub

Public Sub Chart_Initialize()
'initialize chart object

Dim loop1 As Integer

initTime = -1
mvarGraphParamater = sVoltage

With mvarchart
.ToDefaults
.chartType = VtChChartType2dXY
.Plot.UniformAxis = False

```

```

.SeriesColumn = 1
.ColumnCount = 2
.RowCount = MAXDATAPOINTS + 1
.ShowLegend = False
.RandomFill = False
'.Plot.SeriesCollection(1).SeriesMarker.Show = True

    With .Plot.Axis(VtChAxisIdY).AxisTitle
        .VtFont.Size = 8
        .Visible = True
        .Text = "Volts"
    End With

    With .Plot.Axis(VtChAxisIdX).AxisTitle
        .VtFont.Size = 8
        .Visible = True
        .Text = "seconds"
    End With

    '.Title.VtFont.Size = 10
    '.Title = "Voltage Plot"
    ' .Legend.Location.LocationType = VtChLocationTypeBottom

    .Plot.Axis(VtChAxisIdY).AxisScale.Type = VtChScaleTypeLinear

    .Plot.Axis(VtChAxisIdX).AxisScale.Type = VtChScaleTypeLinear
End With
End Sub

```

Listing A33. IBattery Interface Class

```

'battery interface abstract class

Public Property Let mAhCapacity(ByVal vData As Single)
End Property

Public Property Get mAhCapacity() As Single
End Property

Public Property Let cells(ByVal vData As Integer)
End Property

Public Property Get cells() As Integer
End Property

Public Property Let BatteryVoltage(ByVal vData As Single)
End Property

Public Property Get BatteryVoltage() As Single
End Property

Public Property Let CellVoltage(ByVal vData As Single)
End Property

Public Property Get CellVoltage() As Single
End Property

Public Property Let ProfileNum(ByVal vData As Integer)
End Property

Public Property Get ProfileNum() As Integer
End Property

```



```

Public Property Let chargeRate(ByVal vData As Single)
End Property

Public Property Get chargeRate() As Single
End Property

Public Property Let chargeTime(ByVal vData As Single)
End Property

Public Property Get chargeTime() As Single
End Property

Public Property Get MaxProfiles() As Integer
End Property

Public Property Get ProfileName(ByVal idx As Integer) As String
End Property

Public Property Get ProfileDiscription(ByVal idx As Integer) As String
End Property

Public Function Charge() As Boolean
End Function

Public Function Validate() As Boolean
End Function

Public Property Set PowerSupplyRef(ByRef vData As PowerSupply)
End Property

Public Property Set DataAnalyzerRef(ByRef vData As DataAnalyzer)
End Property

Public Property Set ChargerRef(ByRef vData As Charger)
End Property

```

Listing A34. Lead Acid Interface Class

```

'battery interface abstract class

Public Property Let mAhCapacity(ByVal vData As Single)
End Property

Public Property Get mAhCapacity() As Single
End Property

Public Property Let cells(ByVal vData As Integer)
End Property

Public Property Get cells() As Integer
End Property

Public Property Let BatteryVoltage(ByVal vData As Single)
End Property

Public Property Get BatteryVoltage() As Single
End Property

Public Property Let CellVoltage(ByVal vData As Single)
End Property

```

```

Public Property Get CellVoltage() As Single
End Property

Public Property Let ProfileNum(ByVal vData As Integer)
End Property

Public Property Get ProfileNum() As Integer
End Property

Public Property Let chargeRate(ByVal vData As Single)
End Property

Public Property Get chargeRate() As Single
End Property

Public Property Let chargeTime(ByVal vData As Single)
End Property

Public Property Get chargeTime() As Single
End Property

Public Property Get MaxProfiles() As Integer
End Property

Public Property Get ProfileName(ByVal idx As Integer) As String
End Property

Public Property Get ProfileDiscription(ByVal idx As Integer) As String
End Property

Public Function Charge() As Boolean
End Function

Public Function Validate() As Boolean
End Function

Public Property Set PowerSupplyRef(ByRef vData As PowerSupply)
End Property

Public Property Set DataAnalyzerRef(ByRef vData As DataAnalyzer)
End Property

Public Property Set ChargerRef(ByRef vData As Charger)
End Property

```

Listing A35. Lead Acid Class

```

Implements IBattery
Option Explicit
'Implements charge control of lead acid batteries (non sealed)
'local variable(s) to hold property value(s)
Private mvarCells As Integer 'local copy
Private mvarmAhCapacity As Double 'local copy
'local variable(s) to hold property value(s)
Private mvarCellVoltage As Single 'local copy
Const MAXPROFILE = 2
Private ProfileNum As Integer
Private Type PROFILETYPE
    name As String
    discription As String
End Type
Private Profile(MAXPROFILE) As PROFILETYPE

```

```

Private ps As PowerSupply
Private da As DataAnalyzer
Private ch As Charger
Private mvarChargeTime As Single 'local copy
Private mvarChargeRate As Single 'local copy
Public Property Let IBattery_chargeRate(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBATTERY_chargeRate = 5
    If vData <= 1 And vData > 0 Then
        mvarChargeRate = vData
        updateChargeTimeEst
    End If
End Property

Public Property Get IBattery_chargeRate() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBATTERY_chargeRate
    IBattery_chargeRate = mvarChargeRate

End Property

Public Property Let IBattery_chargeTime(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBATTERY_chargeTime = 5
    mvarChargeTime = vData
End Property

Public Property Get IBattery_chargeTime() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBATTERY_chargeTime
    IBattery_chargeTime = mvarChargeTime
End Property

Private Sub updateChargeTimeEst()
If ProfileNum = 1 And mvarChargeRate > 0 Then
    mvarChargeTime = 1 / mvarChargeRate
End If
frmCharge.txtChargeTime = mvarChargeTime & " Hours"
End Sub

Private Sub Class_Initialize()
'lead acid battery parameters
Dim txt As String
mvarCellVoltage = 2.2
mvarCells = 1
ProfileNum = 1
mvarChargeRate = 1 'C rate of charge
Profile(1).name = "Modified CV Fast Charge"
Profile(2).name = "Timed Charge"
txt = "3 stage modified constant voltage fast charge algorithm. " & vbCrLf
txt = txt & "Stage 1. Constant current charge at 'Charge Rate' until voltage drops to 2.39V/cell" & vbCrLf
txt = txt & "Stage 2. Constant voltage charge at 2.39V/cell level until current drops below .2C " & vbCrLf
txt = txt & "Stage 3. follow by a trickle charge at 2.21V/cell"

Profile(1).discription = txt
frmCharge.txtChargeTime.enabled = False
frmCharge.txtCellV.enabled = False 'no change to cell voltage allowed
End Sub

```

```

Private Sub Class_Terminate()
Set ps = Nothing
Set da = Nothing
Set ch = Nothing

End Sub

Public Property Let IBattery_BatteryVoltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_BatteryVoltage = 5
    mvarCells = vData / mvarCellVoltage
End Property

Public Property Get IBattery_BatteryVoltage() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_BatteryVoltage
    IBattery_BatteryVoltage = mvarCells * mvarCellVoltage
End Property

Public Property Let IBattery_CellVoltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_CellVoltage = 5
    If vData > 0 Then 'cell voltage must be greater than zero
        mvarCellVoltage = vData
    End If
End Property

Public Property Get IBattery_CellVoltage() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_CellVoltage
    IBattery_CellVoltage = mvarCellVoltage
End Property

Public Property Let IBattery_mAhCapacity(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.mAhCapacity = 5

    mvarmAhCapacity = vData
End Property

Public Property Get IBattery_mAhCapacity() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.mAhCapacity
    IBattery_mAhCapacity = mvarmAhCapacity
End Property

Public Property Let IBattery_cells(ByVal vData As Integer)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.Cells = 5
    If vData > 0 Then
        mvarCells = vData
    End If
End Property

Public Property Get IBattery_cells() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Cells
    IBattery_cells = mvarCells
End Property

Public Property Let IBattery_ProfileNum(ByVal vData As Integer)
    If vData > 0 Then

```

```

        ProfileNum = vData
        If ProfileNum = 2 Then
            frmCharge.txtChargeTime.enabled = True
        Else
            frmCharge.txtChargeTime.enabled = False
        End If
        updateChargeTimeEst
    End If
End Property

Public Property Get IBattery_ProfileNum() As Integer
    IBattery_ProfileNum = ProfileNum
End Property

Public Property Get IBattery_MaxProfiles() As Integer
    IBattery_MaxProfiles = MAXPROFILE
End Property

Public Property Get IBattery_ProfileName(ByVal idx As Integer) As String
    If idx > 0 And idx <= MAXPROFILE Then
        IBattery_ProfileName = Profile(idx).name
    End If
End Property

Public Property Get IBattery_ProfileDiscription(ByVal idx As Integer) As String
    If idx > 0 And idx <= MAXPROFILE Then
        IBattery_ProfileDiscription = Profile(idx).discription
    End If
End Property

Public Function IBattery_Charge() As Boolean
    Select Case ProfileNum
        Case 1
            Call fastCharge
        Case 2
            Call timedCharge
    End Select
'Else
'    IBattery_Validate
'End If
End Function

Public Property Set IBattery_PowerSupplyRef(ByRef vData As PowerSupply)
    Set ps = vData
End Property

Public Property Set IBattery_DataAnalyzerRef(ByRef vData As DataAnalyzer)
    Set da = vData
End Property

Public Property Set IBattery_ChargerRef(ByRef vData As Charger)
    Set ch = vData
End Property

Public Function IBattery_Validate() As Boolean
'addition battery checks before charging
Dim v As Double
Dim bv As Double, cv As Double
Dim result As VbMsgBoxResult
Dim BatValid As Boolean
Dim MBtitle As String
Dim MBmsg As String

```

```

Const minAHcap = 100
BatValid = False

'check battery parameters
MBtitle = "Battery parameter error"
If mvarmAhCapacity <= minAHcap Then
    MBmsg = "mAh capacity must be greater than " & minAHcap
    Call MsgBox(MBmsg, vbInformation, MBtitle)
    ch.CBfromBattery (ChargeAborted)
    ch.StatusMsg ("Battery validation failed")

    GoTo exitfn
End If

cv = mvarCells * mvarCellVoltage
result = MsgBox("Connect battery to power supply now", vbOKCancel, "Battery Validation
Testing")
If result = vbOK Then
    ch.StatusMsg ("Performing open circuit voltage check")
    If (ps.PSDeviceDriver.getSample) Then
        v = da.getVoltage
        If v < cv * 0.8 Or v > cv Then
            Call ch.StatusMsg("Expected approximately", cv, "Volts")
            Call ch.StatusMsg("Measured", v, "Volts")
            ch.StatusMsg ("Verification test failed!")
            MBtitle = "Verification Test Failed"
            MBmsg = "Do you want to override the verification and attempt to charge
anyway?" _
                & vbCrLf & "This is NOT recommended! Accepting incorrect charging parameters"
            _
                & vbCrLf & "may cause a battery explosion and personal injury!"
            result = MsgBox(MBmsg, vbYesNo + vbExclamation, MBtitle)
            If result = vbYes Then
                BatValid = True
                ch.StatusMsg ("Warning: Battery validation overridden")
            Else
                Call ch.CBfromBattery(ChargeAborted)
                ch.StatusMsg ("Charging aborted")
                GoTo exitfn
            End If
        End If
    Else
        'private data member, validation result
        BatValid = True
    End If
Else
    ch.StatusMsg ("Error obtaining data from charger")
End If
Else
    'connect battery canceled
    Call ch.CBfromBattery(ChargeAborted)
    ch.StatusMsg ("Charging aborted")
    GoTo exitfn

End If

If (mvarChargeRate * mvarmAhCapacity / 1000) > config.MaxCurrent Then
    MBmsg = "Charging paramaters exceed PS current limits, do you want to continue with
reduced current settings?"
    MBtitle = "Battery Validation Testing"
    result = MsgBox(MBmsg, vbOKCancel, MBtitle)
    If result = vbOK Then

```

```

        Me.IBattery_chargeRate = config.MaxCurrent / (mvarmAhCapacity / 1000)    'also
updates charge time
        frmCharge.txtChargeRate = mvarChargeRate
    Else
        BatValid = False
        ch.CBfromBattery (ChargeAborted)
        ch.StatusMsg ("Battery validation failed")
        GoTo exitfn
    End If
End If
exitfn:

IBattery_Validate = BatValid
End Function

Private Sub fastCharge()
Static stage As Integer

Dim batTemp As Single
Dim deltaTemp As Single
Dim msg As String
Dim delta_t
Dim C As Single
Dim CRate As Single
Dim ChargeVoltage As Single
Dim InitChargeCurrent As Single
Dim TrickleVoltage As Single
Dim stage2trig As Single
Dim stage3trig As Single
Dim mV As Single    'measured voltage
Dim mC As Single    'measured current

batTemp = da.getTemperature
deltaTemp = da.getDeltaTemperature
delta_t = deltaTime()
mV = da.getVoltage    'current voltage
mC = da.getCurrent    'current current

CRate = mvarmAhCapacity / 1000
InitChargeCurrent = mvarChargeRate * CRate
ChargeVoltage = 2.39 * mvarCells
TrickleVoltage = 2.21 * mvarCells
stage2trig = ChargeVoltage
stage3trig = 0.2 * CRate

absolute checks
If batTemp > 113 Then
    msg = "Battery high temperature fault, battery temperature = " & batTemp & " F"
    Call MajorFault(msg)
    Exit Sub
End If

If batTemp < 0 Then
    msg = "Battery low temperature fault, battery temperature = " & batTemp & " F"
    Call MajorFault(msg)
    Exit Sub
End If

If deltaTemp > 10 Then
    msg = "Battery high differential temperature fault, Battery temperature = " _
    & batTemp & " F" & " ambient temperature= " & da.getAmbientTemperature

```

```

    Call MajorFault(msg)
    Exit Sub
End If

If delta_t > mvarChargeTime * 3600 Then
    msg = "Battery charge overtime fault, charging for longer than 1 hour"
    Call MajorFault(msg)
    Exit Sub
End If

If mC > (2 * CRate) Then
    msg = "Battery current exceeded 2C rate"
    Call MajorFault(msg)
    Exit Sub
End If

'rate of change of temperature F/min
frmCharge.lbldTdt.Caption = "dTdt (F/min) = " & Format(da.getdTdt, "##.00")

'state machine implementation
Select Case stage
Case 0 'set up a constant current charge at 1C until voltage threshold is achieved
    ch.StatusMsg ("Stage 0 Begining Charge")
    SetCurrent (InitChargeCurrent)
    Call ch.StatusMsg("Constant current mode at", InitChargeCurrent, "A")
    ps.SetOutputOn (True)
    stage = 1
    Call ch.StatusMsg("Stage " & stage & " Waiting for output voltage >", stage2trig,
"V")

Case 1 'current regulation at 1C
    frmCharge.lblDVdt = "dVdt(mV/min) = " & Format(da.getdVdt, "##.00") 'update dVdt
label
    If mV > ChargeVoltage Then 'set up voltage regulation upon threshold
        setVoltage (ChargeVoltage)
        stage = 2
        Call ch.StatusMsg("Stage " & stage & " Waiting for output current <", stage3trig,
"A")
        frmCharge.lblDVdt = "" 'clear label

    End If
Case 2 'voltage regulating at charge voltage
    frmCharge.lblIdt = "dIdt(mA/min) = " & Format(da.getdIdt, "##.00") 'update dIdt
label

    If mC > (CRate * 1.05) Then 'check if current draw is exceeding c rate (with
hysteresis)
        'revert back to CC mode again, stage 1
        SetCurrent (CRate)
        Call ch.StatusMsg("Reverting back to constant current mode at", CRate, "A")
        stage = 1
    End If
    If mC < stage3trig Then
        setVoltage (TrickleVoltage)
        stage = 3
        ch.StatusMsg ("Stage " & stage & " Charge complete ")
        Call ch.StatusMsg(" Trikle charging at", TrickleVoltage, "V")
    End If
Case 3 'voltage regulating at trickle voltage
    frmCharge.lblIdt = "dIdt(mA/min) = " & Format(da.getdIdt, "##.00") 'update dIdt
label

```



```

End Select
End Sub

Private Sub MajorFault(msg As String)
ps.PSDeviceDriver.PeriodicSampling (False)
ps.SetOutputOn (False) 'turn off output
ps.voltage = 0 'turn off command
ch.StatusMsg ("Major Fault: " & msg)
Call ch.CBfromBattery(ChargeAborted)
Call MsgBox(msg, vbCritical, "Major Charging Fault")
End Sub

Private Sub MinorFault()
End Sub

Private Sub timedCharge()
End Sub

Private Sub setVoltage(v As Double)
Dim msg As String

ps.Mode = VoltageRegulation
ps.voltage = v
If ps.voltage = v Then 'only updates if successfull transmit to device
    frmCharge.lblOutput = "Voltage regulating at " & ps.voltage & " V"
Else
    msg = "Power supply did not acknowledge last set voltage command"
    Call MajorFault(msg)
End If
End Sub

Private Sub SetCurrent(i As Double)
Dim msg As String
ps.Mode = CurrentRegulation
ps.current = i
If ps.current = i Then 'only updates if successfull transmit to device
    frmCharge.lblOutput = "Current regulating at " & ps.current & " A"
Else
    msg = "Power supply did not acknowledge last set current command"
End If
End Sub

Private Function deltaTime() As Integer
'calculates delta t
Static init As Boolean
Static initTime As Date
Dim thisTime As Date

If Not init Then
    initTime = Now
    init = True
End If

thisTime = Now
deltaTime = (thisTime - initTime) * 86400#

Debug.Print deltaTime
End Function

```

Listing A36. LiIon Class

```
Implements IBattery
Option Explicit

'local variable(s) to hold property value(s)
Private mvarCells As Integer 'local copy
Private mvarmAhCapacity As Double 'local copy
'local variable(s) to hold property value(s)
Private mvarCellVoltage As Single 'local copy
Const MAXPROFILE = 2
Private ProfileNum As Integer
Private Type PROFILETYPE
    name As String
    discription As String
End Type
Private Profile(MAXPROFILE) As PROFILETYPE
Private ps As PowerSupply
Private da As DataAnalyzer
Private ch As Charger
Private mvarChargeRate As Single 'local copy
Private mvarChargeTime As Single 'local copy (in hours)

Public Property Let IBattery_chargeTime(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBATTERY_chargeTime = 5
    mvarChargeTime = vData
End Property

Public Property Get IBattery_chargeTime() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBATTERY_chargeTime
    IBattery_chargeTime = mvarChargeTime
End Property

Public Property Let IBattery_chargeRate(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBATTERY_chargeRate = 5
    mvarChargeRate = vData
    updateChargeTimeEst
End Property

Public Property Get IBattery_chargeRate() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBATTERY_chargeRate
    IBattery_chargeRate = mvarChargeRate
End Property

Private Sub updateChargeTimeEst()
If ProfileNum = 1 And mvarChargeRate > 0 Then
    mvarChargeTime = 1 / mvarChargeRate
End If
frmCharge.txtChargeTime = mvarChargeTime & " Hours"
End Sub

Private Sub Class_Initialize()
'lead acid battery parameters
Dim txt As String
mvarCellVoltage = 4.1
mvarCells = 1
mvarChargeRate = 1 'C rate of charge
```

```

ProfileNum = 1
Profile(1).name = "Fast Charge"
Profile(2).name = "Timed Charge"
txt = "3 stage fast charge algorithm. " & vbCrLf
txt = txt & "Stage 1. 1C charge terminated when open circuit cell voltage = 4.1V or 4.2V  
(depending on type)" & vbCrLf
txt = txt & "Stage 2. follow by constant voltage charge at the above cell voltage until I  
drops to < 50ma/cell" & vbCrLf
txt = txt & "Stage 3. follow by a maintenance (trickle) charge at .025C rate"

Profile(1).discription = txt

End Sub

Private Sub Class_Terminate()
Set ps = Nothing
Set da = Nothing
Set ch = Nothing

End Sub

Public Property Let IBattery_BatteryVoltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_BatteryVoltage = 5
    mvarCells = vData / mvarCellVoltage
End Property

Public Property Get IBattery_BatteryVoltage() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_BatteryVoltage
    IBattery_BatteryVoltage = mvarCells * mvarCellVoltage
End Property

Public Property Let IBattery_CellVoltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_CellVoltage = 5
    If vData > 0 Then 'cell voltage must be greater than zero
        mvarCellVoltage = vData
    End If
End Property

Public Property Get IBattery_CellVoltage() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_CellVoltage
    IBattery_CellVoltage = mvarCellVoltage
End Property

Public Property Let IBattery_mAhCapacity(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.mAhCapacity = 5
    mvarmAhCapacity = vData
End Property

Public Property Get IBattery_mAhCapacity() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.mAhCapacity
    IBattery_mAhCapacity = mvarmAhCapacity
End Property

Public Property Let IBattery_cells(ByVal vData As Integer)
'used when assigning a value to the property, on the left side of an assignment.

```

```

'Syntax: X.Cells = 5
    If vData > 0 Then
        mvarCells = vData
    End If
End Property

Public Property Get IBattery_cells() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Cells
    IBattery_cells = mvarCells
End Property

Public Property Let IBattery_ProfileNum(ByVal vData As Integer)
    If vData > 0 Then
        ProfileNum = vData
        If ProfileNum = 2 Then
            frmCharge.txtChargeTime.enabled = True
        Else
            frmCharge.txtChargeTime.enabled = False
        End If
        updateChargeTimeEst
    End If
End Property

Public Property Get IBattery_ProfileNum() As Integer
    IBattery_ProfileNum = ProfileNum
End Property

Public Property Get IBattery_MaxProfiles() As Integer
    IBattery_MaxProfiles = MAXPROFILE
End Property

Public Property Get IBattery_ProfileName(ByVal idx As Integer) As String
    If idx > 0 And idx <= MAXPROFILE Then
        IBattery_ProfileName = Profile(idx).name
    End If
End Property

Public Property Get IBattery_ProfileDiscription(ByVal idx As Integer) As String
    If idx > 0 And idx <= MAXPROFILE Then
        IBattery_ProfileDiscription = Profile(idx).discription
    End If
End Property

Public Function IBattery_Charge() As Boolean
    Select Case ProfileNum
        Case 1
            Call fastCharge
        Case 2
            Call timedCharge
    End Select
'Else
'    IBattery_Validate
'End If
End Function

Public Property Set IBattery_PowerSupplyRef(ByRef vData As PowerSupply)
    Set ps = vData
End Property

Public Property Set IBattery_DataAnalyzerRef(ByRef vData As DataAnalyzer)
    Set da = vData

```

```

End Property

Public Property Set IBattery_ChargerRef(ByRef vData As Charger)
    Set ch = vData
End Property

Public Function IBattery_Validate() As Boolean
'addition battery checks before charging
Dim v As Double
Dim bv As Double, cv As Double
Dim result As VbMsgBoxResult
Dim BatValid As Boolean
Dim MBtitle As String
Dim MBmsg As String

Const minAHcap = 100
BatValid = False

'check battery parameters
MBtitle = "Battery parameter error"
If mvarmAhCapacity <= minAHcap Then
    MBmsg = "mAhour capacity must be greater than " & minAHcap
    Call MsgBox(MBmsg, vbInformation, MBtitle)
    ch.CBfromBattery (ChargeAborted)
    ch.StatusMsg ("Battery validation failed")

    GoTo exitfn
End If

MBtitle = "Battery parameter verification"
MBmsg = "Li-Ion cell voltages are manufacture dependent and normally 4.1 or 4.2V/cell." &
vbCrLf
MBmsg = MBmsg & "Please verify the proper cell voltage before continuing. Is the cell
voltage correct?"
    result = MsgBox(MBmsg, vbYesNo + vbExclamation, MBtitle)
    If result = vbYes Then
        ch.StatusMsg ("Cell voltage verified by user OK")
    Else
        Call ch.CBfromBattery(ChargeAborted)
        ch.StatusMsg ("Charging aborted")
        GoTo exitfn
    End If

cv = mvarCells * mvarCellVoltage
result = MsgBox("Connect battery to power supply now", vbOKCancel, "Battery Validation
Testing")
If result = vbOK Then
    ch.StatusMsg ("Performing open circuit voltage check")
    If (ps.PSDeviceDriver.getSample) Then
        v = da.getVoltage
        If v < cv * 0.8 Or v > cv Then
            Call ch.StatusMsg("Expected approximately", cv, "Volts")
            Call ch.StatusMsg("Measured", v, "Volts")
            ch.StatusMsg ("Verification test failed!")
            MBtitle = "Verification Test Failed"
            MBmsg = "Do you want to override the verification and attempt to charge
anyway?" _
                & vbCrLf & "This is NOT recommended! Accepting incorrect charging parameters"
            _
                & vbCrLf & "may cause a battery explosion and personal injury!"

```

```

        result = MsgBox(MBmsg, vbYesNo + vbExclamation, MBtitle)
        If result = vbYes Then
            BatValid = True
            ch.StatusMsg ("Warning: Battery validation overridden")
        Else
            Call ch.CBfromBattery(ChargeAborted)
            ch.StatusMsg ("Charging aborted")
            GoTo exitfn
        End If
    Else
        'private data member, validation result
        BatValid = True
    End If
Else
    ch.StatusMsg ("Error obtaining data from charger")
End If
End If

If (mvarChargeRate * mvarmAhCapacity / 1000) > config.MaxCurrent Then
    MBmsg = "Charging paramaters exceed PS current limits, do you want to continue with
reduced current settings?"
    MBtitle = "Battery Validation Testing"
    result = MsgBox(MBmsg, vbOKCancel, MBtitle)
    If result = vbOK Then
        Me.IBattery_chargeRate = config.MaxCurrent / (mvarmAhCapacity / 1000)    'also
updates charge time
        frmCharge.txtChargeRate = mvarChargeRate
    Else
        BatValid = False
        ch.CBfromBattery (ChargeAborted)
        ch.StatusMsg ("Battery validation failed")
        GoTo exitfn
    End If
End If

exitfn:
IBattery_Validate = BatValid
End Function

Private Sub fastCharge()
Static stage As Integer

Dim batTemp As Single
Dim deltaTemp As Single
Dim msg As String
Dim delta_t As Integer
Static ocvc_t As Integer
Dim C As Single
Dim CRate As Single
Dim InitChargeCurrent As Single
Dim ChargeCurrent As Single
Dim TrickleCurrent As Single
Dim stage3trig As Single
Dim stage4trig As Single
Dim mV As Single    'measured voltage
Dim mC As Single    'measured current
Dim dVdt As Single
Dim dTdt As Single
Dim ChargeVoltage As Single

batTemp = da.getTemperature
deltaTemp = da.getDeltaTemperature

```

```

delta_t = deltaTime()
mV = da.getVoltage 'current voltage
mC = da.getCurrent 'current current
CRate = mvarmAhCapacity / 1000
InitChargeCurrent = mvarChargeRate * CRate
ChargeVoltage = mvarCells * mvarCellVoltage
TrickleCurrent = CRate * 0.025

'stage3trig = ChargeVoltage
stage4trig = 0.05 * mvarCells 'triger at 50ma/cell

'absolute checks
If batTemp > 104 Then
    msg = "Battery high temperature fault (>104F), battery temperature = " & batTemp & "
    F"
    Call MajorFault(msg)
    Exit Sub
End If

If batTemp < 41 Then
    msg = "Battery low temperature fault (<41F), battery temperature = " & batTemp & " F"
    Call MajorFault(msg)
    Exit Sub
End If

If deltaTemp > 10 Then
    msg = "Battery high differential temperature fault, Battery temperature = " _
    & batTemp & " F" & " ambient temperature= " & da.getAmbientTemperature
    Call MajorFault(msg)
    Exit Sub
End If

If delta_t > mvarChargeTime * 5400 Then
    msg = "Battery charge overtime fault"
    Call MajorFault(msg)
    Exit Sub
End If

frmCharge.lbldTdt.Caption = "dTdt (F/min) = " & Format(da.getdTdt, "##.00")

'state machine implementation
Select Case stage
Case 0 'set up a constant current charge at 1C until voltage threshold is achieved
    ch.StatusMsg ("Stage 0 Begining Charge")
    SetCurrent (InitChargeCurrent)
    Call ch.StatusMsg("Constant current mode at", InitChargeCurrent, "A")
    ps.SetOutputOn (True)
    stage = 1
    ocvc_t = delta_t + 10 'open circuit voltage check memory timeout (10 sec initially)
    Call ch.StatusMsg("Stage " & stage & " Waiting for threshold voltage of",
    ChargeVoltage, "V")

Case 1 'regulating in current mode, waiting for periodic open circuit voltage checks
    If delta_t > ocvc_t Then 'sample frequency is equal to est charge time / 60
        setVoltage (0) 'setup for a oc voltage measurement
        stage = 2
    End If
    frmCharge.lbldVdt.Caption = "dVdt (mV/min) = " & Format(da.getdVdt, "##.00")

Case 2 'ps output is off, this sample is the open circuit battery voltage
    Call ch.StatusMsg("Evaluating battery voltage, V=", da.getVoltage, "V")
    If da.getVoltage > ChargeVoltage Then

```

```

        stage = 3
        setVoltage (ChargeVoltage)
        Call ch.StatusMsg("Stage " & stage & " Waiting for ", stage4trig, " mA")
    Else
        stage = 1          'continue in current mode
        ocvc_t = delta_t + (60 / mvarChargeTime) 'open circuit voltage check memory
    timeout
        SetCurrent (InitChargeCurrent)
    End If
Case 3 'voltage regulating at charge voltage, waiting for current decrease
    frmCharge.lblIdt.Caption = "dIdt (mA/min) = " & Format(da.getdIdt, "##.00")
    If da.getCurrent < stage4trig Then
        stage = 4
        SetCurrent (TrickleCurrent)
        Call ch.StatusMsg("Stage " & stage & " Trickle charging at ", TrickleCurrent, "
mA")
    End If
Case 4 'current regulating at trickle current
    frmCharge.lblVdt.Caption = "dVdt (mV/min) = " & Format(da.getdVdt, "##.00")
End Select
End Sub

Private Sub MajorFault(msg As String)
ps.PSDeviceDriver.PeriodicSampling (False)
ps.SetOutputOn (False) 'turn off output
ps.voltage = 0 'turn off command
ch.StatusMsg ("Major Fault: " & msg)
Call ch.CBfromBattery(ChargeAborted)
Call MsgBox(msg, vbCritical, "Major Charging Fault")
End Sub

Private Sub MinorFault()

End Sub

Private Sub timedCharge()

End Sub

Private Sub setVoltage(v As Double)
Dim msg As String
ps.Mode = VoltageRegulation
ps.voltage = v
If ps.voltage = v Then 'only updates if successfull transmit to device
    frmCharge.lblOutput = "Voltage regulating at " & ps.voltage & " V"
Else
    msg = "Power supply did not acknowledge last set voltage command"
    Call MajorFault(msg)
End If
End Sub

Private Sub SetCurrent(i As Double)
Dim msg As String
ps.Mode = CurrentRegulation
ps.current = i
If ps.current = i Then 'only updates if successfull transmit to device
    frmCharge.lblOutput = "Current regulating at " & ps.current & " A"
Else
    msg = "Power supply did not acknowledge last set current command"
End If
End Sub

Private Function deltaTime() As Integer

```



```

'calculates delta t
Static init As Boolean
Static initTime As Date
Dim thisTime As Date

If Not init Then
    initTime = Now
    init = True
End If

thisTime = Now
deltaTime = (thisTime - initTime) * 86400#

Debug.Print deltaTime
End Function

```

Listing A37. ModSmartCharger Standard Module

```

Option Explicit
Public config As Configuration
Public EventLog As EventLogger

Public Sub main()
'program entry point, make global reference to configuration and show main form
Set config = New Configuration
Set EventLog = New EventLogger
frmMain.Show
End Sub

```

Listing A38. NiCad Class

```

Implements IBattery
Option Explicit

'local variable(s) to hold property value(s)
Private mvarCells As Integer 'local copy
Private mvarmAhCapacity As Double 'local copy
'local variable(s) to hold property value(s)
Private mvarCellVoltage As Single 'local copy
Const MAXPROFILE = 2
Private ProfileNum As Integer
Private Type PROFILETYPE
    name As String
    discription As String
End Type
Private Profile(MAXPROFILE) As PROFILETYPE
Private ps As PowerSupply
Private da As DataAnalyzer
Private ch As Charger
Private mvarChargeRate As Single 'local copy
Private mvarChargeTime As Single 'local copy

Public Property Let IBattery_chargeTime(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_chargeTime = 5
    mvarChargeTime = vData
End Property

Public Property Get IBattery_chargeTime() As Single
'used when retrieving value of a property, on the right side of an assignment.

```

```

'Syntax: Debug.Print X.IBattery_chargeTime
    IBattery_chargeTime = mvarChargeTime
End Property

Public Property Let IBattery_chargeRate(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_chargeRate = 5
    If vData <= 3 And vData > 0 Then
        mvarChargeRate = vData
        updateChargeTimeEst
    End If
End Property

Public Property Get IBattery_chargeRate() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_chargeRate
    IBattery_chargeRate = mvarChargeRate
End Property

Private Sub updateChargeTimeEst()
If ProfileNum = 1 And mvarChargeRate > 0 Then
    mvarChargeTime = 1 / mvarChargeRate
End If
frmCharge.txtChargeTime = mvarChargeTime & " Hours"
End Sub

Private Sub Class_Initialize()
'lead acid battery parameters
Dim txt As String
mvarCellVoltage = 1.25
mvarCells = 1
mvarChargeRate = 1 'C rate of charge

ProfileNum = 1
Profile(1).name = "-dVdt fast charge"
Profile(2).name = "Timed Charge"
txt = "3 stage -dVdt fast charge algorithm. " & vbCrLf
txt = txt & "Stage 1. 1C charge for one minute, allow data to stabilize" & vbCrLf
txt = txt & "Stage 2. continue 1C charge terminated with -dVdt (primary) or 1.8F (secondary) dTdt" & vbCrLf
txt = txt & "Stage 3. follow by a maintenance (trickle) charge at .025C rate"

Profile(1).discription = txt

End Sub

Private Sub Class_Terminate()
Set ps = Nothing
Set da = Nothing
Set ch = Nothing
End Sub

Public Property Let IBattery_BatteryVoltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_BatteryVoltage = 5
    mvarCells = vData / mvarCellVoltage
End Property

Public Property Get IBattery_BatteryVoltage() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_BatteryVoltage

```

```

    IBattery_BatteryVoltage = mvarCells * mvarCellVoltage
End Property

Public Property Let IBattery_CellVoltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_CellVoltage = 5
    If vData > 0 Then 'cell voltage must be greater than zero
        mvarCellVoltage = vData
    End If
End Property

Public Property Get IBattery_CellVoltage() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_CellVoltage
    IBattery_CellVoltage = mvarCellVoltage
End Property

Public Property Let IBattery_mAhCapacity(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.mAhCapacity = 5
    mvarmAhCapacity = vData
End Property

Public Property Get IBattery_mAhCapacity() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.mAhCapacity
    IBattery_mAhCapacity = mvarmAhCapacity
End Property

Public Property Let IBattery_cells(ByVal vData As Integer)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.Cells = 5
    If vData > 0 Then
        mvarCells = vData
    End If
End Property

Public Property Get IBattery_cells() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Cells
    IBattery_cells = mvarCells
End Property

Public Property Let IBattery_ProfileNum(ByVal vData As Integer)
    If vData > 0 Then
        ProfileNum = vData
        If ProfileNum = 2 Then
            frmCharge.txtChargeTime.enabled = True
        Else
            frmCharge.txtChargeTime.enabled = False
        End If
        updateChargeTimeEst
    End If
End Property

Public Property Get IBattery_ProfileNum() As Integer
    IBattery_ProfileNum = ProfileNum
End Property

Public Property Get IBattery_MaxProfiles() As Integer
    IBattery_MaxProfiles = MAXPROFILE
End Property

```

```

Public Property Get IBattery_ProfileName(ByVal idx As Integer) As String
    If idx > 0 And idx <= MAXPROFILE Then
        IBattery_ProfileName = Profile(idx).name
    End If
End Property

Public Property Get IBattery_ProfileDiscription(ByVal idx As Integer) As String
    If idx > 0 And idx <= MAXPROFILE Then
        IBattery_ProfileDiscription = Profile(idx).discription
    End If
End Property

Public Function IBattery_Charge() As Boolean
    Select Case ProfileNum
        Case 1
            Call fastCharge
        Case 2
            Call timedCharge
    End Select
'Else
'    IBattery_Validate
'End If
End Function

Public Property Set IBattery_PowerSupplyRef(ByRef vData As PowerSupply)
    Set ps = vData
End Property

Public Property Set IBattery_DataAnalyzerRef(ByRef vData As DataAnalyzer)
    Set da = vData
End Property

Public Property Set IBattery_ChargerRef(ByRef vData As Charger)
    Set ch = vData
End Property

Public Function IBattery_Validate() As Boolean
'addition battery checks before charging
Dim v As Double
Dim bv As Double, cv As Double
Dim result As VbMsgBoxResult
Dim BatValid As Boolean
Dim MBtitle As String
Dim MBmsg As String

Const minAHcap = 100
BatValid = False

'check battery parameters
MBtitle = "Battery parameter error"
If mvarmAhCapacity <= minAHcap Then
    MBmsg = "mAhour capacity must be greater than " & minAHcap
    Call MsgBox(MBmsg, vbInformation, MBtitle)
    ch.CBfromBattery (ChargeAborted)
    ch.StatusMsg ("Battery validation failed")

    GoTo exitfn
End If

cv = mvarCells * mvarCellVoltage

```

```

result = MsgBox("Connect battery to power supply now", vbOKCancel, "Battery Validation
Testing")
If result = vbOK Then
    ch.StatusMsg ("Performing open circuit voltage check")
    If (ps.PSDeviceDriver.getSample) Then
        v = da.getVoltage
        If v < cv * 0.8 Or v > cv * 1.1 Then
            Call ch.StatusMsg("Expected approximately", cv, "Volts")
            Call ch.StatusMsg("Measured", v, "Volts")
            ch.StatusMsg ("Verification test failed!")
            MBtitle = "Verification Test Failed"
            MBmsg = "Do you want to override the verification and attempt to charge
anyway?" _
                & vbCrLf & "This is NOT recommended! Accepting incorrect charging parameters"
            _
                & vbCrLf & "may cause a battery explosion and personal injury!"
            result = MsgBox(MBmsg, vbYesNo + vbExclamation, MBtitle)
            If result = vbYes Then
                BatValid = True
                ch.StatusMsg ("Warning: Battery validation overridden")
            Else
                Call ch.CBfromBattery(ChargeAborted)
                ch.StatusMsg ("Charging aborted")
                GoTo exitfn
            End If
        Else
            'private data member, validation result
            BatValid = True
        End If
    Else
        ch.StatusMsg ("Error obtaining data from charger")
    End If
End If

If (mvarChargeRate * mvarmAhCapacity / 1000) > config.MaxCurrent Then
    MBmsg = "Charging paramaters exceed PS current limits, do you want to continue with
reduced current settings?"
    MBtitle = "Battery Validation Testing"
    result = MsgBox(MBmsg, vbOKCancel, MBtitle)
    If result = vbOK Then
        Me.IBattery_chargeRate = config.MaxCurrent / (mvarmAhCapacity / 1000) 'also
updates charge time
        frmCharge.txtChargeRate = mvarChargeRate
    Else
        BatValid = False
        ch.CBfromBattery (ChargeAborted)
        ch.StatusMsg ("Battery validation failed")
        GoTo exitfn
    End If
End If
exitfn:
IBattery_Validate = BatValid
End Function

Private Sub fastCharge()
Static stage As Integer

Dim batTemp As Single
Dim deltaTemp As Single
Dim msg As String
Dim delta_t
Dim C As Single

```

```

Dim CRate As Single
Dim InitChargeCurrent As Single
Dim ChargeCurrent As Single
Dim TrickleCurrent As Single
Dim stage2trig As Single
Dim stage3trig As Single
Dim mV As Single      'measured voltage
Dim mC As Single      'measured current
Dim dVdt As Single
Dim dTdt As Single

batTemp = da.getTemperature
deltaTemp = da.getDeltaTemperature
delta_t = deltaTime()
mV = da.getVoltage      'current voltage
mC = da.getCurrent      'current current
CRate = mvarmAhCapacity / 1000
InitChargeCurrent = mvarChargeRate * CRate
ChargeCurrent = CRate
TrickleCurrent = CRate * 0.025
'stage2trig = ChargeVoltage
'stage3trig = 0.2 * CRate

'absolute checks
If batTemp > 140 Then '60C (linden)
    msg = "Battery high temperature fault (>104F), battery temperature = " & batTemp & "
    F"
    Call MajorFault(msg)
    Exit Sub
End If

If batTemp < 41 Then
    msg = "Battery low temperature fault (<41F), battery temperature = " & batTemp & " F"
    Call MajorFault(msg)
    Exit Sub
End If

If deltaTemp > 15 Then
    msg = "Battery high differential temperature fault, Battery temperature = " _
    & batTemp & " F" & " ambient temperature= " & da.getAmbientTemperature
    Call MajorFault(msg)
    Exit Sub
End If

If delta_t > mvarChargeTime * 5400 Then
    msg = "Battery charge overtime fault"
    Call MajorFault(msg)
    Exit Sub
End If

If mV > 1.6 * mvarCells Then
    msg = "Battery overvoltage fault, voltage > " & 1.5 * mvarCells
    Call MajorFault(msg)
    Exit Sub
End If

dVdt = da.getdVdt      'rate of change of voltage mv/min
dTdt = da.getdTdt      'rate of change of temperature F/min
frmCharge.lblVdt.Caption = "dVdt (mV/min) = " & Format(dVdt, "##.00")
frmCharge.lblTdt.Caption = "dTdt (F/min) = " & Format(dTdt, "##.00")

```

```

'state machine implementation
Select Case stage
Case 0 'set up a constant current charge at 1C until voltage threshold is achieved
    ch.StatusMsg ("Stage 0 Begining Charge")
    SetCurrent (InitChargeCurrent)
    Call ch.StatusMsg("Constant current mode at", InitChargeCurrent, "A")
    ps.SetOutputOn (True)
    stage = 1
    Call ch.StatusMsg("Stage " & stage & " Waiting for 1 minute data collection")

Case 1 '1 minute wait time
    If delta_t > 60 Then 'set up for -dv/dt termination
        stage = 2
        Call ch.StatusMsg("Stage " & stage & " Waiting for -dV/dt ")
    End If
Case 2 'current regulating at charge current, waiting for EOC triggers dV/dt or dT/dt
    If dVdt < 0 * mvarCells Then 'check for - slope
        ch.StatusMsg ("-dv/dt detected, Fast charge complete, reverting to trickle
charge")
        SetCurrent (TrickleCurrent)
        stage = 3
    End If
    If dTdt > 2.8 Then
        ch.StatusMsg ("2.8 deg F/min dT/dt detected, Fast charge complete, reverting to
trickle charge")
        SetCurrent (TrickleCurrent)
        stage = 3
    End If
Case 3 'current regulating at trickle current

End Select

End Sub

Private Sub MajorFault(msg As String)
ps.PSDeviceDriver.PeriodicSampling (False)
ps.SetOutputOn (False) 'turn off output
ps.voltage = 0 'turn off command
ch.StatusMsg ("Major Fault: " & msg)
Call ch.CBfromBattery(ChargeAborted)
Call MsgBox(msg, vbCritical, "Major Charging Fault")
End Sub

Private Sub MinorFault()
End Sub

Private Sub timedCharge()
End Sub

Private Sub setVoltage(v As Double)
Dim msg As String
ps.Mode = VoltageRegulation
ps.voltage = v
If ps.voltage = v Then 'only updates if successfull transmit to device
    frmCharge.lblOutput = "Voltage regulating at " & ps.voltage & " V"
Else
    msg = "Power supply did not acknowledge last set voltage command"
    Call MajorFault(msg)
End If
End Sub

Private Sub SetCurrent(i As Double)

```

```

Dim msg As String
ps.Mode = CurrentRegulation
ps.current = i
If ps.current = i Then 'only updates if successfull transmit to device
    frmCharge.lblOutput = "Current regulating at " & ps.current & " A"
Else
    msg = "Power supply did not acknowledge last set current command"
End If
End Sub

Private Function deltaTime() As Integer
'calculates delta t
Static init As Boolean
Static initTime As Date
Dim thisTime As Date

If Not init Then
    initTime = Now
    init = True
End If

thisTime = Now
deltaTime = (thisTime - initTime) * 86400#

Debug.Print deltaTime
End Function

```

Listing A39. NiMh Class

```

Implements IBattery
Option Explicit

'local variable(s) to hold property value(s)
Private mvarCells As Integer 'local copy
Private mvarmAhCapacity As Double 'local copy
'local variable(s) to hold property value(s)
Private mvarCellVoltage As Single 'local copy
Const MAXPROFILE = 2
Private ProfileNum As Integer
Private Type PROFILETYPE
    name As String
    discription As String
End Type
Private Profile(MAXPROFILE) As PROFILETYPE
Private ps As PowerSupply
Private da As DataAnalyzer
Private ch As Charger
Private mvarChargeRate As Single 'local copy
Private mvarChargeTime As Single 'local copy

Public Property Let IBattery_chargeTime(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBATTERY_chargeTime = 5
    mvarChargeTime = vData
End Property

Public Property Get IBattery_chargeTime() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBATTERY_chargeTime
    IBattery_chargeTime = mvarChargeTime
End Property

```



```

Public Property Let IBattery_chargeRate(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_chargeRate = 5
    If vData <= 3 And vData > 0 Then
        mvarChargeRate = vData
        updateChargeTimeEst
    End If
End Property

Public Property Get IBattery_chargeRate() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_chargeRate
    IBattery_chargeRate = mvarChargeRate
End Property

Private Sub updateChargeTimeEst()
If ProfileNum = 1 And mvarChargeRate > 0 Then
    mvarChargeTime = 1 / mvarChargeRate
End If
frmCharge.txtChargeTime = mvarChargeTime & " Hours"
End Sub

Private Sub Class_Initialize()
'lead acid battery parameters
Dim txt As String
mvarCellVoltage = 1.25
mvarCells = 1
mvarChargeRate = 1 'C rate of charge

ProfileNum = 1
Profile(1).name = "dTdt fast charge"
txt = "4 stage fast charge algorithm. " & vbCrLf
txt = txt & "Stage 1. 1C charge for 2 minutes, allow data to stabilize" & vbCrLf
txt = txt & "Stage 2. continue 1C charge terminated with 1.8F dTdt (primary) or -dVdt
(secondary) sensing " & vbCrLf
txt = txt & "Stage 3. follow by a .1C topping charge for 45 minutes " & vbCrLf
txt = txt & "Stage 4. follow by a maintenance (trickle) charge at .025C rate"

Profile(1).discription = txt
Profile(2).name = "Timed Charge"
End Sub

Private Sub Class_Terminate()
Set ps = Nothing
Set da = Nothing
Set ch = Nothing

End Sub

Public Property Let IBattery_BatteryVoltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_BatteryVoltage = 5
    mvarCells = vData / mvarCellVoltage
End Property

Public Property Get IBattery_BatteryVoltage() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_BatteryVoltage
    IBattery_BatteryVoltage = mvarCells * mvarCellVoltage
End Property

```

```

Public Property Let IBattery_CellVoltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_CellVoltage = 5
    If vData > 0 Then 'cell voltage must be greater than zero
        mvarCellVoltage = vData
    End If
End Property

Public Property Get IBattery_CellVoltage() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_CellVoltage
    IBattery_CellVoltage = mvarCellVoltage
End Property

Public Property Let IBattery_mAhCapacity(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.mAhCapacity = 5
    mvarmAhCapacity = vData
End Property

Public Property Get IBattery_mAhCapacity() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.mAhCapacity
    IBattery_mAhCapacity = mvarmAhCapacity
End Property

Public Property Let IBattery_cells(ByVal vData As Integer)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.Cells = 5
    If vData > 0 Then
        mvarCells = vData
    End If
End Property

Public Property Get IBattery_cells() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Cells
    IBattery_cells = mvarCells
End Property

Public Property Let IBattery_ProfileNum(ByVal vData As Integer)
    If vData > 0 Then
        ProfileNum = vData
        If ProfileNum = 2 Then
            frmCharge.txtChargeTime.enabled = True
        Else
            frmCharge.txtChargeTime.enabled = False
        End If
        updateChargeTimeEst
    End If
End Property

Public Property Get IBattery_ProfileNum() As Integer
    IBattery_ProfileNum = ProfileNum
End Property

Public Property Get IBattery_MaxProfiles() As Integer
    IBattery_MaxProfiles = MAXPROFILE
End Property

Public Property Get IBattery_ProfileName(ByVal idx As Integer) As String
    If idx > 0 And idx <= MAXPROFILE Then

```

```

        IBattery_ProfileName = Profile(idx).name
    End If
End Property

Public Property Get IBattery_ProfileDiscription(ByVal idx As Integer) As String
    If idx > 0 And idx <= MAXPROFILE Then
        IBattery_ProfileDiscription = Profile(idx).discription
    End If
End Property

Public Function IBattery_Charge() As Boolean
    Select Case ProfileNum
        Case 1
            Call fastCharge
        Case 2
            Call timedCharge
    End Select
'Else
'    IBattery_Validate
'End If
End Function

Public Property Set IBattery_PowerSupplyRef(ByRef vData As PowerSupply)
    Set ps = vData
End Property

Public Property Set IBattery_DataAnalyzerRef(ByRef vData As DataAnalyzer)
    Set da = vData
End Property

Public Property Set IBattery_ChargerRef(ByRef vData As Charger)
    Set ch = vData
End Property

Public Function IBattery_Validate() As Boolean
'addition battery checks before charging
Dim v As Double
Dim bv As Double, cv As Double
Dim result As VbMsgBoxResult
Dim BatValid As Boolean
Dim MBtitle As String
Dim MBmsg As String

Const minAHcap = 100
BatValid = False

'check battery parameters
MBtitle = "Battery parameter error"
If mvarmAhCapacity <= minAHcap Then
    MBmsg = "mAhour capacity must be greater than " & minAHcap
    Call MsgBox(MBmsg, vbInformation, MBtitle)
    ch.CBfromBattery (ChargeAborted)
    ch.StatusMsg ("Battery validation failed")

    GoTo exitfn
End If

cv = mvarCells * mvarCellVoltage
result = MsgBox("Connect battery to power supply now", vbOKCancel, "Battery Validation
Testing")
If result = vbOK Then
    ch.StatusMsg ("Performing open circuit voltage check")

```

```

    If (ps.PSDeviceDriver.getSample) Then
        v = da.getVoltage
        If v < cv * 0.8 Or v > cv Then
            Call ch.StatusMsg("Expected approximately", cv, "Volts")
            Call ch.StatusMsg("Measured", v, "Volts")
            ch.StatusMsg ("Verification test failed!")
            MBtitle = "Verification Test Failed"
            MBmsg = "Do you want to override the verification and attempt to charge
anyway?" _
            & vbCrLf & "This is NOT recommended! Accepting incorrect charging parameters"
            _
            & vbCrLf & "may cause a battery explosion and personal injury!"
            result = MsgBox(MBmsg, vbYesNo + vbExclamation, MBtitle)
            If result = vbYes Then
                BatValid = True
                ch.StatusMsg ("Warning: Battery validation overridden")
            Else
                Call ch.CBfromBattery(ChargeAborted)
                ch.StatusMsg ("Charging aborted")
                GoTo exitfn
            End If
        Else
            'private data member, validation result
            BatValid = True
        End If
    Else
        ch.StatusMsg ("Error obtaining data from charger")
    End If
End If

If (mvarChargeRate * mvarmAhCapacity / 1000) > config.MaxCurrent Then
    MBmsg = "Charging paramaters exceed PS current limits, do you want to continue with
reduced current settings?"
    MBtitle = "Battery Validation Testing"
    result = MsgBox(MBmsg, vbOKCancel, MBtitle)
    If result = vbOK Then
        Me.IBattery_chargeRate = config.MaxCurrent / (mvarmAhCapacity / 1000)    'also
updates charge time
        frmCharge.txtChargeRate = mvarChargeRate
    Else
        BatValid = False
        ch.CBfromBattery (ChargeAborted)
        ch.StatusMsg ("Battery validation failed")
        GoTo exitfn
    End If
End If

exitfn:
IBattery_Validate = BatValid
End Function

Private Sub fastCharge()
Static stage As Integer
Static startTop As Single

Dim batTemp As Single
Dim deltaTemp As Single
Dim msg As String
Dim delta_t
Dim C As Single
Dim CRate As Single
Dim InitChargeCurrent As Single

```

```

Dim ChargeCurrent As Single
Dim toppingCurrent As Single
Dim TrickleCurrent As Single
Dim Stage2aTrig As Single
Dim stage2bTrig As Single
Dim mV As Single      'measured voltage
Dim mC As Single      'measured current
Dim dVdt As Single
Dim dTdt As Single
Dim maxBatV As Single

batTemp = da.getTemperature
deltaTemp = da.getAvgDeltaTemperature(30)
Debug.Print "NiMh delta temp = " & deltaTemp

delta_t = deltaTime()
mV = da.getVoltage      'current voltage
mC = da.getCurrent      'current current
CRate = mvarmAhCapacity / 1000
InitChargeCurrent = mvarChargeRate * CRate

ChargeCurrent = CRate
toppingCurrent = CRate * 0.1
TrickleCurrent = CRate * 0.025
Stage2aTrig = -5 * mvarCells 'in mv/minute
stage2bTrig = 3

'absolute checks
If batTemp > 140 Then '60C (Linden)
    msg = "Battery high temperature fault (>104F), battery temperature = " & batTemp & " F"
    Call MajorFault(msg)
    Exit Sub
End If

If batTemp < 41 Then
    msg = "Battery low temperature fault (<41F), battery temperature = " & batTemp & " F"
    Call MajorFault(msg)
    Exit Sub
End If

If deltaTemp > 10 Then
    msg = "Battery high differential temperature fault, Battery temperature = " _
    & batTemp & " F" & " ambient temperature= " & da.getAmbientTemperature
    Call MajorFault(msg)
    Exit Sub
End If

If stage <> 4 And delta_t > mvarChargeTime * 5400 Then
    msg = "Battery charge overtime fault"
    Call MajorFault(msg)
    Exit Sub
End If

maxBatV = mvarCells * mvarCellVoltage * (1 + CRate / 5)
If mV > maxBatV Then
    msg = "Battery overvoltage fault, voltage > " & maxBatV
    Call MajorFault(msg)
    Exit Sub
End If

```

```

dVdt = da.getdVdt    'rate of change of voltage mv/min
dTdt = da.getdTdt    'rate of change of temperature F/min
frmCharge.lblVdt.Caption = "dVdt (mV/min) = " & Format(dVdt, "##.00")
frmCharge.lblTdt.Caption = "dTdt (F/min) = " & Format(dTdt, "##.00")

'state machine implementation
Select Case stage
Case 0 'set up a constant current charge at 1C until voltage threshold is achieved
    ch.StatusMsg ("Stage 0 Beginning Charge")
    SetCurrent (InitChargeCurrent)
    Call ch.StatusMsg("Constant current mode at", InitChargeCurrent, "A")
    ps.SetOutputOn (True)
    stage = 1
    Call ch.StatusMsg("Stage " & stage & " Waiting for 1 minute data collection")

Case 1 '1 minute wait time
    If delta_t > 120 Then 'set up for -dv/dt termination
        stage = 2
        Call ch.StatusMsg("Stage " & stage & " Waiting for " & Stage2aTrig & "mV/min -
dV/dt ")
    End If

Case 2 'current regulating at charge current, waiting for EOC triggers dV/dt or dT/dt

    If dVdt < Stage2aTrig Then 'check for -dVdt cell
        stage = 3
        ch.StatusMsg ("- dv/dt detected, Fast charge complete.")
        Call ch.StatusMsg("Stage " & stage & " beginning topping charge")
        SetCurrent (toppingCurrent)
        startTop = delta_t
    End If
    If dTdt > stage2bTrig Then
        stage = 3
        ch.StatusMsg (stage2bTrig & " deg F/min dT/dt detected, Fast charge complete,
beginning topping charge")
        Call ch.StatusMsg("Stage " & stage & " beginning 45 minute topping charge")
        SetCurrent (toppingCurrent)
        startTop = delta_t
    End If

Case 3 'topping charge
    If delta_t > startTop + (60 * 45) Then 'topping charge for 45 minutes
        stage = 4
        ch.StatusMsg ("Topping charge complete. Cancel charge and remove battery when
desired")
        Call ch.StatusMsg("Stage " & stage & " beginning maintenance (trickle) charge")
        SetCurrent (TrickleCurrent)
    End If

Case 4 'current regulating at trickle current

End Select
End Sub

Private Sub MajorFault(msg As String)
ps.PSDeviceDriver.PeriodicSampling (False)
ps.SetOutputOn (False) 'turn off output
ps.voltage = 0 'turn off command
ch.StatusMsg ("Major Fault: " & msg)
Call ch.CBfromBattery(ChargeAborted)
Call MsgBox(msg, vbCritical, "Major Charging Fault")
End Sub

Private Sub MinorFault()
End Sub

```

```

Private Sub timedCharge()
End Sub

Private Sub setVoltage(v As Double)
Dim msg As String
ps.Mode = VoltageRegulation
ps.voltage = v
If ps.voltage = v Then 'only updates if successfull transmit to device
    frmCharge.lblOutput = "Voltage regulating at " & ps.voltage & " V"
Else
    msg = "Power supply did not acknowledge last set voltage command"
    Call MajorFault(msg)
End If
End Sub

Private Sub SetCurrent(i As Double)
Dim msg As String
ps.Mode = CurrentRegulation
ps.current = i
If ps.current = i Then 'only updates if successfull transmit to device
    frmCharge.lblOutput = "Current regulating at " & ps.current & " A"
Else
    msg = "Power supply did not acknowledge last set current command"
End If
End Sub

Private Function deltaTime() As Integer
'calculates delta t
Static init As Boolean
Static initTime As Date
Dim thisTime As Date

If Not init Then
    initTime = Now
    init = True
End If

thisTime = Now
deltaTime = (thisTime - initTime) * 86400#

End Function

```

Listing A40. Power Supply Class

```

Option Explicit
'this class controls the high level power supply operations
'inclusing seting of voltage and currents, setting software limits
'turn on and off the output, sets the controller PC or
Private PSdd As PSDeviceDriver

'local variable(s) to hold property value(s)
'Private mvarconfig As Configuration 'local copy
Private mvarGrapher As Graph 'local copy
Private mvarVoltage As Double 'local copy
Private mvarCurrent As Double 'local copy
Private mvarVoltageLowLimit As Double 'local copy
Private mvarVoltageHighLimit As Double 'local copy
Private mvarCurrentLowLimit As Double 'local copy
Private mvarCurrentHighLimit As Double 'local copy
Public Enum RegulationModeType
    VoltageRegulation = 1

```

```

    CurrentRegulation = 2
End Enum

'local variable(s) to hold property value(s)
Private mvarmode As RegulationModeType 'local copy
Private mvarControl As Boolean 'local copy
'local variable(s) to hold property value(s)
Private mvarOutputOn As Boolean 'local copy
Public Event RecSample()

Public Property Get OutputOn() As Boolean
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.OutputOn
    OutputOn = mvarOutputOn
End Property

Public Property Get Control() As Boolean
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Control
    Control = mvarControl
End Property

Public Function SetOutputOn(out As Boolean) As Boolean
    SetOutputOn = PSdd.SetOutputOn(out)
    If SetOutputOn Then
        mvarOutputOn = out
    End If
End Function

Public Function SetPCCControl(ctl As Boolean) As Boolean
    SetPCCControl = PSdd.SetPCCControl(ctl)
    If SetPCCControl Then 'if update was sucessfull
        mvarControl = ctl 'update local variable
    End If
End Function

Public Property Let Mode(ByVal vData As RegulationModeType)
'used when assigning an Object to the property, on the left side of a Set statement.
'Syntax: Set x.mode = Form1
    mvarmode = vData
    'if changing modes, set setpoint to 0
    If vData = CurrentRegulation Then
        mvarCurrent = 0
    Else
        mvarVoltage = 0
    End If
End Property

Public Property Get Mode() As RegulationModeType
'used when retrieving value of a property, on the right side of an assignment.
'Syntax : Debug.Print X.mode
    Mode = mvarmode
End Property

Public Property Let CurrentHighLimit(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.setCurrentHighLimit = 5
    mvarCurrentHighLimit = vData
End Property

Public Property Get CurrentHighLimit() As Single
'used when retrieving value of a property, on the right side of an assignment.

```



```

'Syntax: Debug.Print X.setCurrentHighLimit
    CurrentHighLimit = mvarCurrentHighLimit
End Property

Public Property Let CurrentLowLimit(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.setCurrentLowLimit = 5
    mvarCurrentLowLimit = vData
End Property

Public Property Get CurrentLowLimit() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.setCurrentLowLimit
    CurrentLowLimit = mvarCurrentLowLimit
End Property

Public Property Let VoltageHighLimit(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.setVoltageHighLimit = 5
    mvarVoltageHighLimit = vData
End Property

Public Property Get VoltageHighLimit() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.setVoltageHighLimit
    VoltageHighLimit = mvarVoltageHighLimit
End Property

Public Property Let VoltageLowLimit(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.setVoltageLowLimit = 5
    mvarVoltageLowLimit = vData
End Property

Public Property Get VoltageLowLimit() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.setVoltageLowLimit
    VoltageLowLimit = mvarVoltageLowLimit
End Property

Public Property Let current(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.setCurrent = 5
    If Mode = CurrentRegulation Then
        If PSdd.SetCurrent(vData) Then
            mvarCurrent = vData
            mvarControl = True 'pc in control
        End If
    End If
End Property

Public Property Get current() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax : Debug.Print X.setCurrent
    current = mvarCurrent
End Property

Public Property Let voltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.setVoltage = 5
    If Mode = VoltageRegulation Then

```

```

        If PSdd.setVoltage(vData) Then
            mvarVoltage = vData
            mvarControl = True
        End If
    End If

End Property

Public Property Get voltage() As Single
    'used when retrieving value of a property, on the right side of an assignment.
    'Syntax: Debug.Print X.setVoltage
    voltage = mvarVoltage
End Property

Public Property Set Grapher(ByVal vData As Graph)
    'used when assigning an Object to the property, on the left side of a Set statement.
    'Syntax: Set x.Grapher = Form1
    Set mvarGrapher = vData
End Property

Public Property Get Grapher() As Graph
    'used when retrieving value of a property, on the right side of an assignment.
    'Syntax: Debug.Print X.Grapher
    Set Grapher = mvarGrapher
End Property

Private Sub Class_Initialize()
    'create the mPSDeviceDriver object when the PowerSupply class is created
    Set PSdd = New PSDeviceDriver

    mvarmode = VoltageRegulation
    mvarVoltage = 0
    mvarCurrent = 0

End Sub

Public Property Get PSDeviceDriver() As PSDeviceDriver
    Set PSDeviceDriver = PSdd
End Property

Public Property Set PSDeviceDriver(vData As PSDeviceDriver)
    Set PSdd = vData
End Property

Private Sub Class_Terminate()
    Set PSdd = Nothing
End Sub

```

Listing A41. Power Supply Device Driver Class

```

Option Explicit
'this class handles translation to device dependent high level packets, adds data to data
collection.
Private mvarDataCol As scDataCol
'To fire this event, use RaiseEvent with the following syntax:
'RaiseEvent RecPacket[(arg1, arg2, ... , argn)]
Public Event RecPacket(pacType As String)
Private PwmValue As Integer
Private Mode As Integer      '1=voltage, 0=current
Private tempSamp As New Sample
Private WithEvents soc As Winsock
Private WithEvents tmr As Timer

```

```

Private pwmUpdate As Boolean      'these boolean are used a semiphores
                                  'to coordinate send and receive events

Private errorCode As Integer
Private responded As Boolean
Private flags As Integer
Const ACK = 1    'if any of the following are set, set resp as well
Const NAK = 2
Const SAMP = 4
Const PWM = 8
Const ERR = 16
Const STAT = 32

Public Function SetOutputOn(OnOff As Boolean) As Boolean
    Dim Timeout As Single
    Dim resp As Boolean
    '    Dim packet As String
    Timeout = Timer + config.RetryDelayTime / 1000    'retry delay is in ms
    If OnOff Then
        sendPacket ("O 1")
    Else
        sendPacket ("O 0")
    End If
    Do While Timer < Timeout And Not responded
        DoEvents
    Loop
    If flags And ACK Then
        SetOutputOn = True
    Else
        SetOutputOn = False    'invalid data, no response before timeout
        processNotAck
    End If
End Function

Public Function SetPCControl(Control As Boolean) As Boolean
    Dim Timeout As Single
    Timeout = Timer + config.RetryDelayTime / 1000    'retry delay is in ms
    If Control Then
        sendPacket ("C 1")
    Else
        sendPacket ("C 0")
    End If
    Do While Timer < Timeout And Not responded
        DoEvents
    Loop
    If ACK And flags Then
        SetPCControl = True
    Else
        SetPCControl = False    'invalid data, no response before timeout
        processNotAck
    End If
End Function

Public Function getSample() As Boolean
    Dim Timeout As Single
    Timeout = Timer + 1
    pwmUpdate = False    'gets set to true after update
    sendPacket ("R")
    Do While Timer < Timeout And Not responded
        DoEvents
    Loop
    If SAMP And flags Then

```

```

        getSample = True
    Else
        getSample = False      'invalid data, no response before timeout
        processNotAck
    End If
End Function

Public Function getStatus() As Boolean
    Dim Timeout As Single
    Timeout = Timer + config.RetryDelayTime / 1000
    ' StatusUpdate = False      'gets set to true after update
    sendPacket ("S")
    Do While Timer < Timeout And Not responded
        DoEvents
    Loop
    If STAT And flags Then
        getStatus = True
    Else
        getStatus = False      'invalid data, no response before timeout
        processNotAck
    End If
End Function

Public Function SetCurrent(current As Single) As Boolean
    Dim Timeout As Single
    Dim packet As String
    Timeout = Timer + config.RetryDelayTime / 1000      'retry delay is in ms
    packet = "I " & current
    sendPacket (packet)
    Do While Timer < Timeout And Not responded
        DoEvents
    Loop
    If ACK And flags Then
        SetCurrent = True
    Else
        SetCurrent = False      'invalid data, no response before timeout
        processNotAck
    End If
End Function

Public Function setVoltage(voltage As Single) As Boolean
    Dim packet As String
    Dim Timeout As Single
    Timeout = Timer + config.RetryDelayTime / 1000      'retry delay is in ms
    packet = "V " & voltage
    sendPacket (packet)
    Do While Timer < Timeout And Not responded
        DoEvents
    Loop
    If ACK And flags Then
        setVoltage = True
    Else
        setVoltage = False      'invalid data, no response before timeout
        processNotAck
    End If
End Function

Public Function getPWMvalue() As Integer
    Dim Timeout As Single
    Timeout = Timer + config.RetryDelayTime / 1000      'retry delay is in ms
    pwmUpdate = False      'gets set to true after update
    sendPacket ("P")

```

```

    Do While Timer < Timeout And Not responded
        DoEvents
    Loop
    If PWM And flags Then
        getPWMvalue = PwmValue
    Else
        getPWMvalue = -1    'invalid data, no response before timeout
        processNotAck
    End If
End Function

Private Sub processNotAck()
    If Not responded Then
        dispMessage ("No reply from server")
    End If
    If flags And NAK Then
        dispMessage ("Not acknowledged")
    End If
End Sub

Private Sub sendPacket(pac As String)
    '    pacMan.addPacket (pac)
    responded = False
    flags = 0    'clear flags
    If soc.State = sckConnected Then
        dispMessage ("Sending " & pac)
        soc.SendData (pac)
    Else
        dispMessage ("Not connected to server")
    End If
End Sub

End Sub

Private Sub dispMessage(msg As String)
    frmMain.txtStatus = Format(Now, "hh:mm:ss") & " " & msg & vbCrLf & frmMain.txtStatus
End Sub

Public Property Get DataCol() As scDataCol
    Set DataCol = mvarDataCol
End Property

Private Sub Class_Initialize()
    Set tmr = frmMain.TimerPS
    tmr.enabled = False

    Set soc = frmMain.Winsock1
    Set mvarDataCol = New scDataCol
    connect
End Sub

Public Sub connect()
    If soc.State <> sckClosed Then soc.Close
    soc.RemoteHost = config.Server
    soc.RemotePort = config.Port
    soc.Protocol = sckTCPProtocol
    soc.connect
End Sub

Private Sub Class_Terminate()
    Set mvarDataCol = Nothing
    Set tmr = Nothing

```

```

End Sub

Private Sub soc_DataArrival(ByVal bytesTotal As Long)
Dim arg As Variant
Dim msg As String
Dim command As String
Dim thisTimer As Single
Static lastTimer As Single
Static offSetTime As Single

soc.GetData msg, vbString

arg = Split(msg, " ", 5)
If UBound(arg) >= 0 Then
    command = arg(0)
    responded = True
    Select Case command
        Case "a"
            flags = ACK
        Case "n"
            flags = NAK
        Case "e"
            flags = ERR
            errorCode = arg(1)
        Case "p"
            flags = PWM
            PwmValue = arg(1)
            pwmUpdate = True
        Case "r"
            flags = SAMP
            thisTimer = Timer()
            If thisTimer < lastTimer Then
                'rollover at midnight detection, add on offset to timestamp
                offSetTime = offSetTime + 86400 '(24*60*60) num seconds in day
            End If
            lastTimer = thisTimer 'update sample memory
            tempSamp.timeStamp = thisTimer + offSetTime
            tempSamp.voltage = arg(1)
            tempSamp.current = arg(2)
            tempSamp.ambTemp = arg(3)
            tempSamp.batTemp = arg(4)
            Call mvarDataCol.Add(tempSamp)
        Case "s"
            flags = STAT
        Case "i"
    End Select
    RaiseEvent RecPacket(command)
End If
End Sub

Public Sub PeriodicSampling(enabled As Boolean)
If enabled Then
    tmr.Interval = config.PSupdate
    tmr.enabled = True
Else
    tmr.enabled = False
End If
End Sub

Private Sub tmr_Timer()
    Call getSample
End Sub

```

Listing A42. Sample Class

```
Option Explicit
'local variable(s) to hold property value(s)
Private mvarVoltage As Single 'local copy
Private mvarCurrent As Single 'local copy
Private mvarAmbTemp As Single 'local copy
Private mvarBatTemp As Single 'local copy
Private mvartimeStamp As Single 'local copy

Public Enum SampleFieldType
    sVoltage = 1
    sCurrent = 2
    sAmbientTemperature = 3
    sBatteryTemperature = 4
End Enum

Const MAX10 = &H3FF&
Const MAX16 = &HFFFF&

Public Function toString() As String
'display a sample in string format
    toString = "Time= " & mvartimeStamp & _
               "Voltage= " & mvarVoltage & _
               "Current= " & mvarCurrent & _
               "Amb Temp= " & mvarAmbTemp & _
               "Bat Temp= " & mvarBatTemp

End Function

Public Function getTime() As Date
End Function

Public Property Let timeStamp(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.timeStamp = 5
    mvartimeStamp = vData
End Property

Public Property Get timeStamp() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.timeStamp
    timeStamp = mvartimeStamp
End Property

Public Property Let batTemp(ByVal vData As Double)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.BatTemp = 5
    mvarBatTemp = vData
End Property

Public Property Get batTemp() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.BatTemp
    batTemp = mvarBatTemp
End Property

Public Property Let ambTemp(ByVal vData As Double)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.AmbTemp = 5
    mvarAmbTemp = vData
End Property
```

```

Public Property Get ambTemp() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.AmbTemp
    ambTemp = mvarAmbTemp
End Property

Public Property Let current(ByVal vData As Double)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.Current = 5
    mvarCurrent = vData
End Property

Public Property Get current() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Current
    current = mvarCurrent
End Property

Public Property Let voltage(ByVal vData As Double)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.Voltage = 5
    mvarVoltage = vData
End Property

Public Property Get voltage() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Voltage
    voltage = mvarVoltage
End Property

```

Listing A43. Data Collection Class

```

Option Explicit
'local variable to hold collection
Private mCol As Collection
Const MAXCOLLECTIONSIZE = 500 'maximum number of stored samples
Const WRITEBUFFSIZE = 100
Private writeBuffCount As Integer

Public Function Add(ByRef arg As Sample, Optional sKey As String) As Sample
'create a new object
Dim objNewMember As Sample
Set objNewMember = New Sample

'copy all fields to new object
objNewMember.timeStamp = arg.timeStamp
objNewMember.voltage = arg.voltage
objNewMember.current = arg.current
objNewMember.ambTemp = arg.ambTemp
objNewMember.batTemp = arg.batTemp

'set the properties passed into the method
If Len(sKey) = 0 Then
    mCol.Add objNewMember
Else
    mCol.Add objNewMember, sKey
End If
'return the object created
Set Add = objNewMember
Set objNewMember = Nothing

```



```

    If mCol.Count = MAXCOLLECTIONSIZE Then mCol.Remove (1)

    writeBuffCount = writeBuffCount + 1
    If writeBuffCount = WRITEBUFFSIZE Then
        writeToDisk
    End If
End Function

Public Property Get Item(vntIndexKey As Variant) As Sample
    'used when referencing an element in the collection
    'vntIndexKey contains either the Index or Key to the collection,
    'this is why it is declared as a Variant
    'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
    Set Item = mCol(vntIndexKey)
End Property

Public Property Get LastItem() As Sample
    'used when referencing an element in the collection
    'vntIndexKey contains either the Index or Key to the collection,
    'this is why it is declared as a Variant
    'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
    Set LastItem = mCol(mCol.Count)
End Property

Public Property Get Count() As Long
    'used when retrieving the number of elements in the
    'collection. Syntax: Debug.Print x.Count
    Count = mCol.Count
End Property

Public Sub Remove(vntIndexKey As Variant)
    'used when removing an element from the collection
    'vntIndexKey contains either the Index or Key, which is why
    'it is declared as a Variant
    'Syntax: x.Remove(xyz)
    mCol.Remove vntIndexKey
End Sub

Public Property Get NewEnum() As IUnknown
    'this property allows you to enumerate
    'this collection with the For...Each syntax
    Set NewEnum = mCol.[_NewEnum]
End Property

Private Sub Class_Initialize()
    'creates the collection when this class is created
    Set mCol = New Collection
End Sub

Private Sub Class_Terminate()
    'destroys collection when this class is terminated
    writeToDisk 'write buffer contents to disk before exiting
    Set mCol = Nothing
End Sub

Private Sub writeToDisk()
    Dim filename As String
    Dim filenum As Integer
    Dim loop1 As Integer
    Static initialized As Boolean

    Dim SAMP As Sample

```

```

On Error GoTo handle

filename = Format(Now, "d-mmm") & " dat.csv"
filenum = FreeFile

Open filename For Input As filenum
Close filenum

Open filename For Append As filenum

start:
If Not initialized Then
    initialized = True
    Write #filenum,          'blank line
End If

For loop1 = mCol.Count - writeBuffCount + 1 To mCol.Count

    Set SAMP = mCol.Item(loop1)
    Write #filenum, SAMP.timeStamp, SAMP.voltage, SAMP.current, SAMP.ambTemp,
SAMP.batTemp
Next loop1

Close filenum
writeBuffCount = 0 'reset write buffer count
Exit Sub

handle:
Select Case ERR.Number
Case Is = 53
    'file doesn't exist, write write header
    Open filename For Append As filenum
    Print #filenum, "Time, Voltage, Current, Amb Temp, Bat Temp"    'print# does not use
    ""
    GoTo start
Case Else
    MsgBox ("Problem with file write operation: Number " & ERR.Number & ERR.Description)
End Select
End Sub

```

Listing A44. Sealed Lead Acid Class

```

Implements IBattery
Option Explicit
'Implements sealed lead acid battery charging algorithms
'local variable(s) to hold property value(s)
Private mvarCells As Integer 'local copy
Private mvarmAhCapacity As Double 'local copy
'local variable(s) to hold property value(s)
Private mvarCellVoltage As Single 'local copy
Const MAXPROFILE = 2
Private ProfileNum As Integer
Private Type PROFILETYPE
    name As String
    discription As String
End Type
Private Profile(MAXPROFILE) As PROFILETYPE
Private ps As PowerSupply
Private da As DataAnalyzer
Private ch As Charger
Private mvarChargeTime As Single 'local copy
Private mvarChargeRate As Single 'local copy

```

```

Public Property Let IBattery_chargeRate(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_chargeRate = 5
    If vData <= 3 And vData > 0 Then
        mvarChargeRate = vData
        updateChargeTimeEst
    End If
End Property

Public Property Get IBattery_chargeRate() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_chargeRate
    IBattery_chargeRate = mvarChargeRate

End Property

Public Property Let IBattery_chargeTime(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_chargeTime = 5
    mvarChargeTime = vData
    updateChargeTimeEst
End Property

Public Property Get IBattery_chargeTime() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_chargeTime
    IBattery_chargeTime = mvarChargeTime
End Property

Private Sub updateChargeTimeEst()
If ProfileNum = 1 And mvarChargeRate > 0 Then
    mvarChargeTime = 1 / mvarChargeRate
End If
frmCharge.txtChargeTime = mvarChargeTime & " Hours"
End Sub

Private Sub Class_Initialize()
'lead acid battery parameters
Dim txt As String
mvarCellVoltage = 2.2
mvarCells = 1
mvarChargeRate = 1 'C rate of charge

ProfileNum = 1
Profile(1).name = "Fast Charge"
Profile(2).name = "Timed Charge"
txt = "3 stage modified constant voltage fast charge algorithm. " & vbCrLf
txt = txt & "Stage 1. Constant current charge at 'Charge Rate' until voltage drops to 2.45V/cell" & vbCrLf
txt = txt & "Stage 2. Constant voltage charge at 2.45V/cell level until current drops below .2C " & vbCrLf
txt = txt & "Stage 3. follow by a trickle charge at 2.25V/cell"

Profile(1).discription = txt
frmCharge.txtChargeTime.enabled = False
frmCharge.txtCellV.enabled = False 'no change to cell voltage allowed
End Sub

Private Sub Class_Terminate()
Set ps = Nothing
Set da = Nothing

```

```

Set ch = Nothing

End Sub

Public Property Let IBattery_BatteryVoltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_BatteryVoltage = 5
    mvarCells = vData / mvarCellVoltage
End Property

Public Property Get IBattery_BatteryVoltage() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_BatteryVoltage
    IBattery_BatteryVoltage = mvarCells * mvarCellVoltage
End Property

Public Property Let IBattery_CellVoltage(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.IBattery_CellVoltage = 5
    If vData > 0 Then 'cell voltage must be greater than zero
        mvarCellVoltage = vData
    End If
End Property

Public Property Get IBattery_CellVoltage() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.IBattery_CellVoltage
    IBattery_CellVoltage = mvarCellVoltage
End Property

Public Property Let IBattery_mAhCapacity(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.mAhCapacity = 5
    mvarmAhCapacity = vData
End Property

Public Property Get IBattery_mAhCapacity() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.mAhCapacity
    IBattery_mAhCapacity = mvarmAhCapacity
End Property

Public Property Let IBattery_cells(ByVal vData As Integer)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.Cells = 5
    If vData > 0 Then
        mvarCells = vData
    End If
End Property

Public Property Get IBattery_cells() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Cells
    IBattery_cells = mvarCells
End Property

Public Property Let IBattery_ProfileNum(ByVal vData As Integer)
    If vData > 0 Then
        ProfileNum = vData
        If ProfileNum = 2 Then
            frmCharge.txtChargeTime.enabled = True
        Else

```

```

        frmCharge.txtChargeTime.enabled = False
    End If
    updateChargeTimeEst
End If
End Property

Public Property Get IBattery_ProfileNum() As Integer
    IBattery_ProfileNum = ProfileNum
End Property

Public Property Get IBattery_MaxProfiles() As Integer
    IBattery_MaxProfiles = MAXPROFILE
End Property

Public Property Get IBattery_ProfileName(ByVal idx As Integer) As String
    If idx > 0 And idx <= MAXPROFILE Then
        IBattery_ProfileName = Profile(idx).name
    End If
End Property

Public Property Get IBattery_ProfileDiscription(ByVal idx As Integer) As String
    If idx > 0 And idx <= MAXPROFILE Then
        IBattery_ProfileDiscription = Profile(idx).discription
    End If
End Property

Public Function IBattery_Charge() As Boolean
    Select Case ProfileNum
        Case 1
            Call fastCharge
        Case 2
            Call timedCharge
    End Select
'Else
'    IBattery_Validate
'End If
End Function

Public Property Set IBattery_PowerSupplyRef(ByRef vData As PowerSupply)
    Set ps = vData
End Property

Public Property Set IBattery_DataAnalyzerRef(ByRef vData As DataAnalyzer)
    Set da = vData
End Property

Public Property Set IBattery_ChargerRef(ByRef vData As Charger)
    Set ch = vData
End Property

Public Function IBattery_Validate() As Boolean
'addition battery checks before charging
Dim v As Double
Dim bv As Double, cv As Double
Dim result As VbMsgBoxResult
Dim BatValid As Boolean
Dim MBtitle As String
Dim MBmsg As String

Const minAHcap = 100
BatValid = False

```

```

'check battery parameters
MBtitle = "Battery parameter error"
If mvarmAhCapacity <= minAHcap Then
    MBmsg = "mAhour capacity must be greater than " & minAHcap
    Call MsgBox(MBmsg, vbInformation, MBtitle)
    ch.CBfromBattery (ChargeAborted)
    ch.StatusMsg ("Battery validation failed")

    GoTo exitfn
End If

cv = mvarCells * mvarCellVoltage
result = MsgBox("Connect battery to power supply now", vbOKCancel, "Battery Validation
Testing")
If result = vbOK Then
    ch.StatusMsg ("Performing open circuit voltage check")
    If (ps.PSDeviceDriver.getSample) Then
        v = da.getVoltage
        If v < cv * 0.8 Or v > cv * 1.1 Then
            Call ch.StatusMsg("Expected approximately", cv, "Volts")
            Call ch.StatusMsg("Measured", v, "Volts")
            ch.StatusMsg ("Verification test failed!")
            MBtitle = "Verification Test Failed"
            MBmsg = "Do you want to override the verification and attempt to charge
anyway?" _
_
            & vbCrLf & "This is NOT recommended! Accepting incorrect charging parameters"
_
            & vbCrLf & "may cause a battery explosion and personal injury!"
            result = MsgBox(MBmsg, vbYesNo + vbExclamation, MBtitle)
            If result = vbYes Then
                BatValid = True
                ch.StatusMsg ("Warning: Battery validation overridden")
            Else
                Call ch.CBfromBattery(ChargeAborted)
                ch.StatusMsg ("Charging aborted")
                GoTo exitfn
            End If
        Else
            'private data member, validation result
            BatValid = True
        End If
    Else
        ch.StatusMsg ("Error obtaining data from charger")
    End If
End If

If (mvarChargeRate * mvarmAhCapacity / 1000) > config.MaxCurrent Then
    MBmsg = "Charging paramaters exceed PS current limits, do you want to continue with
reduced current settings?"
    MBtitle = "Battery Validation Testing"
    result = MsgBox(MBmsg, vbOKCancel, MBtitle)
    If result = vbOK Then
        Me.IBattery_chargeRate = config.MaxCurrent / (mvarmAhCapacity / 1000) 'also
updates charge time
        frmCharge.txtChargeRate = mvarChargeRate
    Else
        BatValid = False
        ch.CBfromBattery (ChargeAborted)
        ch.StatusMsg ("Battery validation failed")
        GoTo exitfn
    End If
End If

```

```

exitfn:
IBattery_Validate = BatValid
End Function

Private Sub fastCharge()
Static stage As Integer
Dim batTemp As Single
Dim deltaTemp As Single
Dim msg As String
Dim delta_t
Dim C As Single
Dim CRate As Single
Dim InitChargeCurrent As Single
Dim ChargeVoltage As Single
Dim TrickleVoltage As Single
Dim stage2trig As Single
Dim stage3trig As Single
Dim mV As Single      'measured voltage
Dim mC As Single      'measured current

batTemp = da.getTemperature
deltaTemp = da.getDeltaTemperature
delta_t = deltaTime()
mV = da.getVoltage      'current voltage
mC = da.getCurrent      'current current
CRate = mvarmAhCapacity / 1000
InitChargeCurrent = mvarChargeRate * CRate

ChargeVoltage = 2.45 * mvarCells
TrickleVoltage = 2.25 * mvarCells
stage2trig = ChargeVoltage
stage3trig = 0.2 * CRate

'absolute checks
If batTemp > 113 Then
    msg = "Battery high temperature fault, battery temperature = " & batTemp & " F"
    Call MajorFault(msg)
    Exit Sub
End If

If batTemp < 10 Then
    msg = "Battery low temperature fault, battery temperature = " & batTemp & " F"
    Call MajorFault(msg)
    Exit Sub
End If

If deltaTemp > 15 Then
    msg = "Battery high differential temperature fault, Battery temperature = " _
    & batTemp & " F" & " ambient temperature= " & da.getAmbientTemperature
    Call MajorFault(msg)
    Exit Sub
End If

If delta_t > mvarChargeTime * 3600 And stage <> 3 Then
    msg = "Battery charge overtime fault, charging for longer than 1 hour"
    Call MajorFault(msg)
    Exit Sub
End If

If mC > (2 * CRate) Then
    msg = "Battery current exceeded 2C rate"
    Call MajorFault(msg)

```

```

Exit Sub
End If

'state machine implementation
Select Case stage
Case 0 'set up a constant current charge at 1C until voltage threshold is achieved
    ch.StatusMsg ("Stage 0 Begining Charge")
    SetCurrent (InitChargeCurrent)
    Call ch.StatusMsg("Constant current mode at", InitChargeCurrent, "A")
    ps.SetOutputOn (True)
    stage = 1
    Call ch.StatusMsg("Stage " & stage & " Waiting for output voltage >", stage2trig,
"V")

Case 1 'current regulation at 1C
    frmCharge.lblVdt = "dVdt(mV/min) = " & Format(da.getdVdt, "##.00") 'update dVdt
label

    If mV > ChargeVoltage Then 'set up voltage regulation upon threshold
        setVoltage (ChargeVoltage)
        stage = 2
        Call ch.StatusMsg("Stage " & stage & " Waiting for output current <", stage3trig,
"A")
        frmCharge.lblVdt = "" 'clear label
    End If
Case 2 'voltage regulating at charge voltage
    frmCharge.lblIdt = "dIdt(mA/min) = " & Format(da.getdIdt, "##.00") 'update dIdt
label

    If mC > InitChargeCurrent * 1.05 Then 'check if current draw is exceeding c rate
(with hysteresis)
        'revert back to CC mode again, stage 1
        SetCurrent (InitChargeCurrent)
        Call ch.StatusMsg("Reverting back to constant current mode at",
InitChargeCurrent, "A")
        stage = 1
    End If
    If mC < stage3trig Then
        setVoltage (TrickleVoltage)
        stage = 3
        ch.StatusMsg ("Stage " & stage & " Charge complete ")
        Call ch.StatusMsg(" Trikle charging at", TrickleVoltage, "V")
        frmCharge.lblIdt = "" 'clear label

    End If
Case 3 'voltage regulating at trickle voltage
    frmCharge.lblVdt = "dVdt(mV/min) = " & Format(da.getdVdt, "##.00") 'update dVdt
label

End Select
End Sub

Private Sub MajorFault(msg As String)
ps.PSDeviceDriver.PeriodicSampling (False)
ps.voltage = 0 'turn off command
ps.SetOutputOn (False) 'turn off output
ch.StatusMsg ("Major Fault: " & msg)
Call ch.CBfromBattery(ChargeAborted)
Call MsgBox(msg, vbCritical, "Major Charging Fault")
End Sub

Private Sub MinorFault()

```



```

End Sub

Private Sub timedCharge()
End Sub

Private Sub setVoltage(v As Double)
Dim msg As String
ps.Mode = VoltageRegulation
ps.voltage = v
If ps.voltage = v Then 'only updates if successfull transmit to device
    frmCharge.lblOutput = "Voltage regulating at " & ps.voltage & " V"
Else
    msg = "Power supply did not acknowledge last set voltage command"
    Call MajorFault(msg)
End If
End Sub

Private Sub SetCurrent(i As Double)
Dim msg As String
ps.Mode = CurrentRegulation
ps.current = i
If ps.current = i Then 'only updates if successfull transmit to device
    frmCharge.lblOutput = "Current regulating at " & ps.current & " A"
Else
    msg = "Power supply did not acknowledge last set current command"
End If
End Sub

Private Function deltaTime() As Integer
'calculates delta t
Static init As Boolean
Static initTime As Date
Dim thisTime As Date

If Not init Then
    initTime = Now
    init = True
End If

thisTime = Now
deltaTime = (thisTime - initTime) * 86400#

Debug.Print deltaTime
End Function

```

Server Application Code

Listing A45. Main Form Code

```

Option Explicit
Dim WithEvents Server As SocketManager
Dim WithEvents com As CommManager
Dim Emulate As Boolean
Const SETTING = "19200,n,8,1"

Private Sub cmdCloseApp_Click()
Server.socApp_Close
End Sub

Private Sub cmdCloseTerm_Click()

```

```

Server.socTerm_Close
End Sub

Private Sub com_recFromCom(packet As String)
    txtFromComm.Text = packet
End Sub

Private Sub com_sentToCom(packet As String)
    txtToComm.Text = packet
End Sub

Private Sub Form_Load()
    txtBox.Text = "" 'clear box

    Set Server = New SocketManager
    Set com = Server.getCommManager
    com.MSCommSettings = SETTING
    com.InitComm 'initialize port
    Server.BeginListen

    frmMain.Caption = "Smart Charger TCP-RS232 bridge server and PS emulator."

    dispText ("Smart Charger TCP-RS232 bridge server and PS emulator.")
    dispText ("By Steven Savage ")

    statBar.Panels(1).Text = "Name: " & Server.ServerName
    statBar.Panels(2).Text = "IP: " & Server.ServerIPAddress
    statBar.Panels(3).Text = "Terminal Interface port: " & config.Port1
    statBar.Panels(4).Text = "Application Interface port: " & config.Port2

    statBar.Panels(5).Text = "Comm Port: " & config.CommPort
    statBar.Panels(6).Text = SETTING
    optMode(0).value = True 'initially set for device
    txtToComm.Text = "" 'clear text box
    txtFromComm.Text = "" 'clear text box
End Sub

Public Sub dispText(msg As String)
    txtBox = txtBox & msg & vbCrLf
End Sub

Private Sub mnuExit_Click()
    Unload Me
End Sub

Private Sub mnuOptions_Click()
    frmOptions.Show
End Sub

Private Sub optMode_Click(Index As Integer)
    If Index = 0 Then
        com.Emulate = False
    Else
        com.Emulate = True 'emulator mode
    End If
End Sub

Private Sub server_ClientServiceRequest(service As SERVICETYPE, ClientIP As String,
ClientPort As Integer)
If service = Application Then
    dispText (Now & " Accepted application connection from " & ClientIP & _
" on port " & ClientPort)

```

```

        cmdCloseApp.Enabled = True
Else
    dispText (Now & " Accepted terminal connection from " & ClientIP & _
        " on port " & ClientPort)
    cmdCloseTerm.Enabled = True
End If
End Sub

Private Sub server_ConexionClosed(service As SERVICETYPE)
If service = Application Then
    dispText (Now & " Application connection closed")
    cmdCloseApp.Enabled = False
Else
    dispText (Now & " Terminal connection closed")
    cmdCloseTerm.Enabled = False
End If
End Sub

```

Listing A46. Options Form Code

```

Option Explicit
Public listind As Integer

Private Sub cmdApply_Click()
    Call updateSettings
End Sub

Private Sub cmdCancel_Click()
    Unload Me
End Sub

Private Sub cmdOK_Click()
    If cmdApply.Enabled = True Then
        Call updateSettings
    End If
    Unload Me
End Sub

Private Sub Form_Load()
Dim loop1 As Integer
For loop1 = 0 To config.NumberOfSettings - 1
    If loop1 > 0 Then
        Load lblPar(loop1)
        Load txtPar(loop1)
        txtPar(loop1).Top = txtPar(loop1 - 1).Top + txtPar(loop1 - 1).Height
        lblPar(loop1).Top = txtPar(loop1).Top
    End If
    lblPar(loop1).Caption = config.configName(loop1)
    txtPar(loop1).Text = config.configValue(loop1)
    lblPar(loop1).Visible = True
    txtPar(loop1).Visible = True
Next loop1

cmdApply.Enabled = False
End Sub

Private Sub updateSettings()
Dim loop1 As Integer

For loop1 = 0 To (txtPar.Count - 1)
    config.configValue(loop1) = txtPar.Item(loop1).Text

```

```

        txtPar.Item(loop1).Text = config.configValue(loop1) 'display current values after
filtering
Next loop1
config.SaveSettings
cmdApply.Enabled = False
End Sub

Private Sub txtPar_Change(Index As Integer)
    cmdApply.Enabled = True
End Sub

```

Listing A47. BytePair class

```

Option Explicit
'local variable(s) to hold property value(s)
Private mvarHighByte As Byte 'local copy
Private mvarLowByte As Byte 'local copy
'local variable(s) to hold property value(s)
Private MAXI As Long
Private maxF As Double

Public Sub setScaleFactors(maxFloat As Variant, Optional maxInt As Long = &HFFFF&)
'used for float value calculations, scalling to float values
MAXI = maxInt
maxF = maxFloat
End Sub

Public Property Let floatValue(ByVal vData As Single)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.floatValue = 5
Dim value As Long
    If vData > maxF Then vData = maxF    'check and adjust bounds
    If vData < 0 Then vData = 0          'check and adjust bounds
    value = (vData / maxF) * MAXI
    fromLong (value)
End Property

Public Property Get floatValue() As Single
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.floatValue
    floatValue = toLong() / MAXI * maxF
End Property

Public Property Let StringValue(ByVal vData As String)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.StringValue = 5
    If Len(vData) > 1 Then
        mvarHighByte = Asc(Mid(vData, 1, 1))
        mvarLowByte = Asc(Mid(vData, 2, 1))
    End If
End Property

Public Property Get StringValue() As String
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.StringValue
    StringValue = Chr(mvarHighByte) & Chr(mvarLowByte)

End Property

Public Property Let RawValue(ByVal arg As Long)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.RawValue = 5

```

```

        fromLong (arg)
End Property

Public Property Get RawValue() As Long
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.RawValue
        RawValue = toLong()
End Property

Public Property Let LowByte(ByVal vData As Byte)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.LowByte = 5
        mvarLowByte = vData
End Property

Public Property Get LowByte() As Byte
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.LowByte
        LowByte = mvarLowByte
End Property

Public Property Let HighByte(ByVal vData As Byte)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.HighByte = 5
        mvarHighByte = vData
End Property

Public Property Get HighByte() As Byte
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.HighByte
        HighByte = mvarHighByte
End Property

Private Function toLong() As Long
        toLong = mvarHighByte * &H100& + mvarLowByte
End Function

Private Sub fromLong(arg As Long)
        mvarLowByte = arg And &HFF
        mvarHighByte = (arg \ &H100) And &HFF      'integer division and mask
End Sub

```

Listing A48. Checksum Calculator Class

```

Option Explicit

Public Function value(arg As String) As Integer
'returns the checksum value
Dim cSum As Integer
Dim loop1 As Integer
Dim strlen As Integer

cSum = 0
strlen = Len(arg)
For loop1 = 1 To strlen
        cSum = (cSum + Asc(Mid(arg, loop1, 1))) Mod &H100
Next loop1
value = cSum
End Function

Public Function char(arg As String) As String
'returns the checksum value cast as an ancii character
char = Chr(value(arg))

```

```

End Function

Public Function CSValid(arg As String) As Boolean
'this function returns true if the packet contains a valid checksum
Dim temp As String
Dim calculated As Integer
Dim received As Integer

temp = Left(arg, Len(arg) - 1) 'packet minus last character which is checksum
calculated = value(temp)
received = Asc(Right(arg, 1)) 'convert last character to char code
If calculated = received Then
    CSValid = True
Else
    CSValid = False
End If
End Function

```

Listing A49. CommManager Class

```

Option Explicit
'this class manages the com control and low level packet encode/decode
'local variable(s) to hold property value(s)
Private WithEvents tmr As Timer
Private WithEvents tmrPreventLOC As Timer
Private WithEvents com As MSComm
Private packets As New Collection
Private waitingForDevice As Boolean
Private retryCount As Integer
Private checkSum As New CheckSumCalculator

'Private mvarCommPort As Integer 'local copy
Private mvarMSCommSettings As String 'local copy
Private mvarEmulate As Boolean 'local copy
'Private mvarRetries As Integer
Private WithEvents emulator As PSEmulator

Public Enum COMFAULTTYPE
    MaxRetries = 3
End Enum

'To fire this event, use RaiseEvent with the following syntax:
'RaiseEvent comError[(arg1, arg2, ... , argn)]
Public Event receivedDataPacket(pac As String)
Public Event comError(pac As String, comError As COMFAULTTYPE)
Public Event sentToCom(packet As String)
Public Event recFromCom(packet As String)

Public Property Get getEmulator() As PSEmulator
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.getEmulator
    Set getEmulator = emulator
End Property

'local variable(s) to hold property value(s)
Public Property Let MSCommSettings(ByVal vData As String)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.MSCommSettings = 5
    mvarMSCommSettings = vData
End Property

```

```

Public Property Get MSCommSettings() As String
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.MSCommSettings
    MSCommSettings = mvarMSCommSettings
End Property

Public Property Let Emulate(ByVal vData As Boolean)
'used when assigning a value to the property, on the left side of an assignment.
'Syntax: X.Emulate = 5
    mvarEmulate = vData
End Property

Public Property Get Emulate() As Boolean
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Emulate
    Emulate = mvarEmulate
End Property

Public Sub InitComm()
On Error GoTo handle
com.Settings = mvarMSCommSettings
com.CommPort = config.CommPort
com.RThreshold = 1
com.PortOpen = True
Exit Sub

handle:
Call MsgBox("Problem initializing com port", vbCritical)
On Error GoTo 0 'reset error handle
End Sub

Public Sub addPacket(pac As String)
Dim firstchar As String
firstchar = Left(pac, 1)
If firstchar = "R" Or firstchar = "P" Or firstchar = "S" Then
    packets.Add (pac)
Else
    packets.Add (pac & checksum.char(pac))
End If

If Not waitingForDevice Then
    sendPacket
End If

End Sub

Private Sub sendPacket()
'index 1 is always the oldest packet

If packets.Count > 0 Then
    retryCount = retryCount + 1
    If retryCount > config.Retries Then
        'give up on packet and contine
        RaiseEvent comError(packets(1), MaxRetries)
        packets.Remove (1)
        waitingForDevice = False
        If packets.Count > 0 Then
            retryCount = 1
        Else
            retryCount = 0
        End If
        Exit Sub 'nothing more to transmit
    End If
End If

```

```

        End If
        tmr.Enabled = True
        RaiseEvent sentToCom(PacketToString(packets(1)))      'display sent packet on main
form
        If mvarEmulate Then
            emulator.ProcessPCrequest (packets(1))
        Else
            com.output = packets(1)
        End If
        waitingForDevice = True
    End If
End Sub

Private Function packetType(pac As String) As String
'return first character of packet
packetType = Left(pac, 1)
End Function

Private Sub Class_Initialize()
    Dim TaskID As Long
    Set emulator = New PSEmulator
    Set tmr = frmMain.Timeout
    Set com = frmMain.MSComm1
    tmr.Enabled = False
    tmr.Interval = config.RetryDelayTime
End Sub

Private Sub Class_Terminate()
    If com.PortOpen = True Then com.PortOpen = False
End Sub

Private Sub com_OnComm()
    Dim data As String
    If com.InBufferCount > 0 Then
        data = com.Input
        extractPacket (data)
    End If
End Sub

Private Sub extractPacket(pacData As String)
    Static data As String
    Dim dataLength As Integer
    Dim firstchar As String
    Dim packet As String

    packet = ""
    data = data & pacData
    dataLength = Len(data)
    firstchar = Left(data, 1)
    Select Case firstchar
    Case "a", "n"
        packet = Left(data, 1)
    Case "e"
        If dataLength >= 3 Then
            packet = Left(data, 3)
        End If
    Case "p", "s"
        If dataLength >= 4 Then
            packet = Left(data, 4)
        End If
    Case "r"
        If dataLength >= 10 Then

```



```

        packet = Left(data, 10)
    End If
Case Else
    data = ""    'delete invalid data

End Select

If packet <> "" Then    'if packet is not empty then packet is extracted from the
stream
    RaiseEvent recFromCom(PacketToString(packet))    'display on main form
    responseVerification (packet)    'check response compared with transmitted packet
    data = Right(data, Len(data) - Len(packet)) 'strip off part that's processed
End If
End Sub

Private Sub responseVerification(data As String)
'this sub compares received packet with last sent packet and test's for valid response
and valid checksum
'if invalid do nothing, timeout will handle
Dim dataLength As Integer
Dim firstchar As String
Dim transPacketType As String
Dim recPacketType As String
Dim valid As Boolean
    valid = False
    dataLength = Len(data)
    recPacketType = Left(data, 1)
    If packets.Count > 0 Then    'data could arrive with no transmit packet ie 'e'
        transPacketType = Left(packets(1), 1)
    End If

    Select Case recPacketType
    Case "a"
        If transPacketType = "V" Or transPacketType = "I" Or _
            transPacketType = "O" Or transPacketType = "C" Then
            valid = True
        End If

    Case "n"
        If transPacketType = "V" Or transPacketType = "I" Or _
            transPacketType = "O" Or transPacketType = "C" Then
            'not acknowledged result
            'should resend before timeout, needs work
            valid = False
            tmr_Timer    'immediatly call timer
        End If

    Case "e"
        If checkSum.CSValid(data) Then
            valid = True
        End If

    Case "p", "s"
        If checkSum.CSValid(data) Then
            If transPacketType = UCase(recPacketType) Then    'expected results
                valid = True
            End If
        End If

    Case "r"
        If checkSum.CSValid(data) Then
            If transPacketType = UCase(recPacketType) Then    'expected results
                valid = True
            End If
        End If
    End Select
End Sub

```

```

        End If
    End If
Case Else
    valid = False
End Select

If valid Then
    RaiseEvent receivedDataPacket(data)
    retryCount = 0
    packets.Remove (1)
    waitingForDevice = False
End If
End Sub

Private Sub emulator_ayscComm(packet As String)
    extractPacket (packet)
End Sub

Private Sub tmr_Timer()
    'receive packet response timed out
    If packets.Count > 0 Then
        waitingForDevice = False      'give up on the response and send again
        ' RaiseEvent comError(packets(1), Timeout)
        ' EventLog.AddMessage ("Receive response timed out retry # " & retryCount)
        tmr.Enabled = False
        sendPacket
    End If
End Sub

Private Function PacketToString(arg As String) As String
    Dim loop1 As Integer
    Dim result As String
    For loop1 = 1 To Len(arg)
        ' result = result & Hex(Asc(Mid(arg, loop1, 1))) & " "
        result = result & (Asc(Mid(arg, loop1, 1))) & " "
    Next loop1
    PacketToString = result
End Function

```

Listing A50. Configuration Class

```

Option Explicit
Const APPNAME As String = "Smart Charger Server" 'used with registry set/get
Const SECTION As String = "Settings"
Private Type ConfigType
    name As String
    value As String
End Type

Private mvarconfig() As ConfigType 'dynamic array of config objects
Private mvarPort1 As Integer 'local copy
Private mvarPort2 As Integer 'local copy
Private mvarCommPort As Integer
Private mvarMaxCurrent As Double 'local copy
Private mvarMinCurrent As Double 'local copy
Private mvarMaxVoltage As Double 'local copy
Private mvarMinVoltage As Double 'local copy
Private mvarMaxTemperature As Double 'local copy
Private mvarMinTemperature As Double 'local copy
Private mvarRetries As Integer 'local copy
Private mvarRetryDelayTime As Integer 'local copy

```

```
Public Property Get RetryDelayTime() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.RetryDelayTime
    RetryDelayTime = mvarRetryDelayTime
End Property
```

```
Public Property Get Retries() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.Retries
    Retries = mvarRetries
End Property
```

```
Public Property Get MinTemperature() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.MinTemperature
    MinTemperature = mvarMinTemperature
End Property
```

```
Public Property Get MaxTemperature() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.MaxTemperature
    MaxTemperature = mvarMaxTemperature
End Property
```

```
Public Property Get Port1() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.ComPort
    Port1 = mvarPort1
End Property
```

```
Public Property Get Port2() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.ComPort
    Port2 = mvarPort2
End Property
```

```
Public Property Get CommPort() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.ComPort
    CommPort = mvarCommPort
End Property
```

```
Public Property Get MaxCurrent() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.MaxCurrent
    MaxCurrent = mvarMaxCurrent
End Property
```

```
Public Property Get MinCurrent() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.PSupdate
    MinCurrent = mvarMinCurrent
End Property
```

```
Public Property Get MaxVoltage() As Double
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.MaxVoltage
    MaxVoltage = mvarMaxVoltage
End Property
```

```
Public Property Get MinVoltage() As Integer
'used when retrieving value of a property, on the right side of an assignment.
```

```

'Syntax: Debug.Print X.PSupdate
    MinVoltage = mvarMinVoltage
End Property

Public Sub SaveSettings()
    Dim name As String
    Dim defaultValue As String
    Dim i As Integer

    For i = 0 To NumberOfSettings - 1
        name = mvarconfig(i).name
        defaultValue = mvarconfig(i).value
        SaveSetting APPNAME, SECTION, name, defaultValue
    Next i
End Sub

Public Property Get NumberOfSettings() As Integer
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.NumberOfSettings
    NumberOfSettings = UBound(mvarconfig) + 1
End Property

Private Sub Class_Initialize()
    'create the mconfig object when the Configuration class is created

    Dim defaultValue As String    'default parameter value
    Dim name As String            'default parameter name
    Dim i As Integer              'index variable

    i = 0
    ReDim Preserve mvarconfig(i)
    name = "Maximum voltage"
    defaultValue = "22"
    mvarconfig(i).name = name
    mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
    mvarMaxVoltage = CDBl(mvarconfig(i).value)

    i = 1
    ReDim Preserve mvarconfig(i)
    name = "Maximum current"
    defaultValue = "11"
    mvarconfig(i).name = name
    mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
    mvarMaxCurrent = CDBl(mvarconfig(i).value)

    i = 2
    ReDim Preserve mvarconfig(i)
    name = "Minimum voltage"
    defaultValue = "0"
    mvarconfig(i).name = name
    mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
    mvarMinVoltage = CDBl(mvarconfig(i).value)

    i = 3
    ReDim Preserve mvarconfig(i)
    name = "Minimum current"
    defaultValue = "0"
    mvarconfig(i).name = name
    mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
    mvarMinCurrent = CDBl(mvarconfig(i).value)

    i = 4

```

```

ReDim Preserve mvarconfig(i)
name = "TCP IP Port1"
defaultValue = "23"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarPort1 = CInt(mvarconfig(i).value)

i = 5
ReDim Preserve mvarconfig(i)
name = "TCP IP Port2"
defaultValue = "1080"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarPort2 = CInt(mvarconfig(i).value)

i = 6
ReDim Preserve mvarconfig(i)
name = "Maximum Temperature"
defaultValue = "500"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarMaxTemperature = CDBl(mvarconfig(i).value)

i = 7
ReDim Preserve mvarconfig(i)
name = "Minimum Temperature"
defaultValue = "1"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarMinTemperature = CDBl(mvarconfig(i).value)

i = 8
ReDim Preserve mvarconfig(i)
name = "Retries"
defaultValue = "3"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarRetries = mvarconfig(i).value

i = 9
ReDim Preserve mvarconfig(i)
name = "Retry delay time(ms)"
defaultValue = "100"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarRetryDelayTime = mvarconfig(i).value

i = 10
ReDim Preserve mvarconfig(i)
name = "Comm port"
defaultValue = "1"
mvarconfig(i).name = name
mvarconfig(i).value = GetSetting(APPNAME, SECTION, name, defaultValue)
mvarCommPort = mvarconfig(i).value
End Sub

Public Property Get configName(Index As Integer) As String
    configName = mvarconfig(Index).name
End Property

Public Property Get configValue(Index As Integer) As String
    configValue = mvarconfig(Index).value

```

End Property

```
Public Property Let configValue(Index As Integer, arg As String)
'this procedure allow update of paramaters with error checking
'must synchronize this procedure with the initialize procedure
On Error GoTo handle
    If Index < 0 Or Index > NumberOfSettings Then GoTo handle
    Dim cname As String
    cname = configName(Index)
    Dim test As Variant
    Select Case cname
    Case "Maximum voltage"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Then GoTo handle 'drop out if less than 0
        mvarconfig(Index).value = arg
    Case "Maximum current"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Then GoTo handle 'drop out if less than 0
        mvarconfig(Index).value = arg
    Case "Minimum voltage"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Then GoTo handle 'drop out if less than 0
        mvarconfig(Index).value = arg
    Case "Minimum current"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Then GoTo handle 'drop out if less than 0
        mvarconfig(Index).value = arg
    Case "TCP IP Port1"
        test = CInt(arg) 'test if argument will cast to int
        If test < 0 Or test > 60000 Then GoTo handle 'what is the high limit on a port?
        mvarconfig(Index).value = arg
    Case "TCP IP Port2"
        test = CInt(arg) 'test if argument will cast to int
        If test < 0 Or test > 60000 Then GoTo handle 'what is the high limit on a port?
        mvarconfig(Index).value = arg
    Case "Maximum Temperature"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Then GoTo handle
        mvarconfig(Index).value = arg
    Case "Minimum Temperature"
        test = CSng(arg) 'test if argument will cast to single
        If test < 0 Then GoTo handle
        mvarconfig(Index).value = arg
    Case "Retries"
        test = CInt(arg) 'test if argument will cast to integer
        If test < 0 Then GoTo handle
        mvarconfig(Index).value = arg
    Case "Retry delay time(ms)"
        test = CInt(arg) 'test if argument will cast to integer
        If test < 0 Then GoTo handle
        mvarconfig(Index).value = arg
    Case "Comm port"
        test = CInt(arg) 'test if argument will cast to integer
        If test < 0 Or test > 4 Then GoTo handle
        mvarconfig(Index).value = arg
    End Select

handle:
End Property
```

Listing A51. ModMain Module

```
Option Explicit
Public config As Configuration
'Public EventLog As EventLogger

Public Sub main()
'program entry point, make global reference to configuration and show main form
If App.PrevInstance Then End 'only one instance can run

Set config = New Configuration
'Set EventLog = New EventLogger
frmMain.Show

End Sub
```

Listing A52. Power Supply Emulator Class

```
Option Explicit
'this class provides an interface with Smart Charger and emulates the behavior of
'the device
Const MAXV = 20
Const MAXI = 10
Public Enum MODETYPE
    voltreg = 0
    curreg = 1
End Enum
Public Enum CONTROLTYPE
    Device = 0
    PC = 1
End Enum

'local variable(s) to hold property value(s)
Private intAlg(8) As BytePair 'internal analog channels
Private extAlg(2) As BytePair 'external analog channels
Private voltageSP As BytePair
Private currentSP As BytePair
Private pwmValue As BytePair
Private checksum As New CheckSumCalculator
Private actualMode As MODETYPE
Private desiredMode As MODETYPE
Private R1 As Double
Private control As CONTROLTYPE
Private output As Byte
'To fire this event, use RaiseEvent with the following syntax:
'RaiseEvent ayscComm[(arg1, arg2, ... , argn)]
Public Event ayscComm(packet As String)

Public Sub simulateError(errnum As Integer)
    Dim pac As String
    pac = "e" & Asc(errnum)
    pac = pac & checksum.char(pac)
    RaiseEvent ayscComm(pac)
End Sub

Public Function ProcessPCrequest(msg As String) As String
'this sub emulates the decoding of a packet by the device and sends an appropriate
response
Dim firstchar As String
Dim response As String
```

```

Dim subString As String

'these models calculate v and i to return to the application
'VariableResistorModel (30) 'variable resistance .1 to arg to .1 to arg ... ohms
'FixedResistorModel (10) 'fixed resistor of 10 ohms
'sinWaveVoltageSensingModel
'BatteryModel1 'simple lead acid
BatteryModel2 'NiCd or NiMh with - slope
'constantSlopeModel

firstchar = Left(msg, 1)
Select Case firstchar
    Case "R" 'pc requests data
        response = "r" & extAlg(0).StringValue & extAlg(1).StringValue & _
            intAlg(2).StringValue & intAlg(3).StringValue
        response = response & checkSum.char(response)
    Case "V" 'pc sets voltage
        If checkSum.CSValid(msg) Then
            response = "a"
            subString = Mid(msg, 2, 2)
            voltageSP.StringValue = subString 'update voltage from request
            desiredMode = voltreg
            actualMode = voltreg 'required for alg
            control = PC
        Else
            response = "n" 'send not ack
        End If
    Case "I" 'pc sets current
        If checkSum.CSValid(msg) Then
            response = "a"
            subString = Mid(msg, 2, 2)
            currentSP.StringValue = subString 'update current from request
            desiredMode = curreg
            actualMode = curreg 'required for algorithm
            control = PC
        Else
            response = "n" 'send not ack
        End If

    Case "C"
        If checkSum.CSValid(msg) Then
            response = "a"
            subString = Mid(msg, 2, 1)
            control = Asc(subString)
        Else
            response = "n" 'send not ack
        End If

    Case "O"
        If checkSum.CSValid(msg) Then
            response = "a"
            subString = Mid(msg, 2, 1)
            output = Asc(subString)
        Else
            response = "n" 'send not ack
        End If

    Case "P"
        pwmValue.RawValue = 1000 * output
        response = "p" & pwmValue.StringValue
        response = response & checkSum.char(response)
        'BatteryModel2 (0) 'for testing, reset charge memory during getpwm

```



```

        Case "S"
            response = "s" & Chr(actualMode) & Chr(control)
            response = response & checksum.char(response)

        End Select
        RaiseEvent ayscComm(response)
        ProcessPCrequest = response 'return response
End Function

Private Sub sinWaveVoltageSensingModel()
Static t As Integer
    t = (t + 1) Mod 50
    extAlg(0).floatValue = (10 + 10 * Sin(t / 25 * 2 * 3.141592)) * output
    extAlg(1).RawValue = extAlg(1).RawValue * output

End Sub

Private Sub FixedResistorModel(r As Double)
RegulationSolution (r)
End Sub

Private Sub BatteryModel1()
Static t0 As Single
Static Qt As Double
Static i As Double

Const Vs = 12
Const Ro = 0.5
Const C = 100
Const Rsd = 1000

Dim dt As Double
Dim Vc As Double
Dim Vin As Double
Dim Vb As Double
Dim tn As Single

If t0 = 0 Then t0 = Timer
tn = Timer
dt = tn - t0
If dt > 10 Then Qt = 0 'reset memory if long delta time
Qt = Qt + (i * dt) - (Qt * dt) / (C * Rsd)
If Qt < 0 Then Qt = 0 'stored charge cannot be negative

Vc = Qt / C

Vb = Vs + i * Ro + Vc
Debug.Print "Qt= " & Qt & " Vc= " & Vc & " Vb= " & Vb & " Ib " & i
If actualMode = voltreg Then
    Vin = voltageSP.floatValue
    i = (-Vs - Vc + Vin) / Ro * output 'if output =0 then i=0
Else
    i = currentSP.floatValue
    Vin = Vb
End If
t0 = tn
RegulationSolution (Vin / (i + 0.00001)) 'avoid div by zero

If Vb > Vin Then 'voltage is vb due to series diode
    extAlg(0).floatValue = Vb
    intAlg(0).floatValue = Vb

```

```

End If
End Sub

Private Sub BatteryModel2(Optional q As Single = -1)
Static t0 As Single
Static Qt As Double
Static i As Double

Const Vs = 7#           'simulate 6 cell pac
Const Ro = 0.1
Const C = 400
Const Rsd = 1000
Const pi = 3.1415926
Dim dt As Double
Dim Vc As Double
Dim Vin As Double
Dim Vb As Double
Dim tn As Single
Dim ambTemp As Single
If q <> -1 Then Qt = q
If t0 = 0 Then t0 = Timer
tn = Timer
dt = tn - t0

If dt > 10 Then Qt = 0 'reset memory if long delta time

Qt = Qt + (i * dt) - (Qt * dt) / (C * Rsd)
If Qt < 0 Then Qt = 0 'stored charge cannot be negative
If Qt > 0.75 * C Then 'if sufficiently charged begin a positive then negative slope
    Vc = 0.5 * (Qt / C) + 0.2 * Sin(((Qt / C) * 2 * pi - 4.712))
Else
    Vc = 0.5 * (Qt / C)
End If

Vb = Vs + i * Ro + Vc

Debug.Print "Qt= " & Qt & " Vc= " & Vc & " Vb= " & Vb & " Ib " & i & " dt" & dt
If actualMode = voltreg Then
    Vin = voltageSP.floatValue
    i = (-Vs - Vc + Vin) / Ro * output 'if output =0 then i=0
Else
    i = currentSP.floatValue
    Vin = Vb
End If
t0 = tn
RegulationSolution (Vin / (i + 0.00001)) 'avoid div by zero

If Vb > Vin Then 'voltage is vb due to series diode
    extAlg(0).floatValue = Vb
    intAlg(0).floatValue = Vb
End If
ambTemp = 73
intAlg(2).floatValue = ambTemp 'ambient temperature
'simulate slight depression in temperature during endothermic charge, then positive
temp due to exothermic reaction
If Qt < C Then
    intAlg(3).floatValue = ambTemp - Qt / C * 2
Else
    intAlg(3).floatValue = ambTemp - Qt / C * 2 + (Qt / C - 1) * 10
End If

```

End Sub

```
Sub constantSlopeModel()  
Static lasttmr As Single  
Static lastv As Single  
Dim thisv As Single  
Dim thistmr As Single  
  
Static slope As Single  
thistmr = Timer  
'calculate a constant slope voltage up and down  
If lasttmr = 0 Then  
    thisv = 0  
    slope = 20000 / (60000) '20000mv/min  
Else  
    thisv = lastv + slope * (thistmr - lasttmr)  
End If  
  
If thisv > 20 Or thisv < 0 Then slope = slope * -1  
lasttmr = thistmr  
lastv = thisv
```

```
    extAlg(0).floatValue = thisv  
    intAlg(0).floatValue = thisv
```

End Sub

```
Sub RegulationSolution(r As Double)  
Dim v As Double  
Dim i As Double  
'this sub calculates voltage, current, and regulation mode for a given load resistance r  
  
If desiredMode = voltreg Then  
    If actualMode <> voltreg Then  
        If intAlg(0) < voltageSP Or currentSP.floatValue < MAXI Then  
            actualMode = voltreg  
        End If  
    End If  
    v = voltageSP.floatValue  
    i = v / (r + 0.01)  
    If i > MAXI Then  
        i = MAXI  
        v = i * r  
        actualMode = curreg  
    End If  
Else 'current regulation  
    If actualMode <> curreg Then  
        If intAlg(1) < currentSP Or voltageSP.floatValue < MAXV Then  
            actualMode = curreg  
        End If  
    End If  
  
    i = currentSP.floatValue  
    v = i * r  
    If v > MAXV Then  
        v = MAXV  
        i = v / r  
        actualMode = voltreg  
    End If  
End If  
'if output=0 then output is set to off
```

```

extAlg(0).floatValue = v
extAlg(1).floatValue = i
intAlg(0).floatValue = v
intAlg(1).floatValue = i
End Sub

Private Sub VariableResistorModel(maxValue As Double)
'simulates saw variable resistor
Dim v As Double
Dim i As Double
Static r As Double
Static up As Boolean

'simulate load changes
If up Then 'going up
    r = r + 0.1
    If r >= maxValue Then up = False
Else 'going down
    r = r - 0.1
    If r <= 0 Then
        r = 0.1
        up = True
    End If
End If
End Sub

RegulationSolution (r)

End Sub

Private Sub Class_Initialize()
'Set tmr = frmMain.TimerDevice
Dim loop1 As Integer

For loop1 = 0 To 7
    If loop1 < 2 Then Set extAlg(loop1) = New BytePair
    Set intAlg(loop1) = New BytePair

Next loop1
Set pwmValue = New BytePair
Set voltageSP = New BytePair
Set currentSP = New BytePair
extAlg(0).setScaleFactors (22) 'external voltage 0-22V sensed
extAlg(1).setScaleFactors (11) 'external current 0-11A sensed
voltageSP.setScaleFactors (20) 'setpoint 0-20V
currentSP.setScaleFactors (10) 'setpoint 0-10A
Call intAlg(0).setScaleFactors(22, &H3FF) 'internal voltage
Call intAlg(1).setScaleFactors(11, &H3FF) 'internal current
Call intAlg(2).setScaleFactors(500, &H3FF * 10) 'amb temperature
Call intAlg(3).setScaleFactors(500, &H3FF * 10) 'battery temperature

intAlg(2).floatValue = 70 'F
intAlg(3).floatValue = 70 'F
output = 1 'output is on
End Sub

Private Sub Class_Terminate()
Dim loop1 As Integer

For loop1 = 0 To 7
    If loop1 < 2 Then Set extAlg(loop1) = Nothing
    Set intAlg(loop1) = Nothing

```

Next loop1

```
Set pwmValue = Nothing
Set voltageSP = Nothing
Set currentSP = Nothing
End Sub
```

Listing A53. Socket Manager Class

```
Option Explicit
'This class provides a link between the PC com port and a TCP/IP socket
'run this server on a machine with rs232 communication and run hyper term or application
on remote
'machine with server name and port configured
'default values set in initialize method
'local variable(s) to hold property value(s)
'Private mvarServerPortTerm As Integer 'local copy
'Private mvarServerPortApp As Integer 'local copy

Private bpSetV As New BytePair
Private bpSetC As New BytePair
Private bpGetV As New BytePair
Private bpGetI As New BytePair
Private bpGetAmbTemp As New BytePair
Private bpGetBatTemp As New BytePair
Private bpPWMvalue As New BytePair
Private lastSendTime As Single 'last time packet was sent
Private lastTermRec As Date
Private lastAppRec As Date

Private WithEvents tmrLOCP As Timer 'loss of coms prevention
Private WithEvents com As CommManager
'hyper terminal interface socket on port 23
Private WithEvents socTerm As Winsock
'application interface on socket 107
Private WithEvents socApp As Winsock

'To fire this event, use RaiseEvent with the following syntax:
'RaiseEvent ClientServiceRequest[(arg1, arg2, ... , argn)]
Public Event ClientServiceRequest(service As SERVICETYPE, ClientIP As String, ClientPort
As Integer)
Public Event ConectionClosed(service As SERVICETYPE)
Public Event comPacket(packet As String)
Public Event socPacket(packet As String)
Public Enum SERVICETYPE
    Terminal = 1
    Application = 2
End Enum

Public Property Get getCommManager() As CommManager
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.getCommManager
    Set getCommManager = com
End Property

Public Property Get ServerIPAddress() As String
'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.ServerIPAddress
    ServerIPAddress = socTerm.LocalIP
End Property

Public Property Get ServerName() As String
```

```

'used when retrieving value of a property, on the right side of an assignment.
'Syntax: Debug.Print X.ServerName
    ServerName = socTerm.LocalHostName
End Property

Public Sub BeginListen()
If socTerm.State = sckClosed Then
    socTerm.LocalPort = config.Port1
    socTerm.Listen      'begin listening to port for a client
End If

If socApp.State = sckClosed Then
    socApp.LocalPort = config.Port2
    socApp.Listen      'begin listening to port for a client
End If
End Sub

Private Sub Class_Initialize()
Set socTerm = frmMain.Winsock1
Set socApp = frmMain.Winsock2
Set com = New CommManager
Set tmrLOCP = frmMain.tmrLOCP

bpSetV.setScaleFactors (config.MaxVoltage)
bpSetC.setScaleFactors (config.MaxCurrent)
bpGetV.setScaleFactors (config.MaxVoltage * 1.1)      'voltage measurement scaled to 22V
bpGetI.setScaleFactors (config.MaxCurrent * 1.1)      'voltage measurement scaled to 11A
Call bpGetAmbTemp.setScaleFactors(config.MaxTemperature / 10, &H3FF)
Call bpGetBatTemp.setScaleFactors(config.MaxTemperature / 10, &H3FF)
End Sub

Private Sub Class_Terminate()
'If com.PortOpen Then com.PortOpen = False
If socTerm.State <> sckClosed Then socTerm.Close
If socApp.State <> sckClosed Then socApp.Close
End Sub

Private Sub com_comError(pac As String, comError As COMFAULTTYPE)
If socTerm.State = sckConnected Then
    socTerm.SendData ("not acknowledged!" & vbCrLf)
End If
If socApp.State = sckConnected Then
    socApp.SendData ("n")
End If
End Sub

Private Sub com_receivedDataPacket(pac As String)
If socTerm.State = sckConnected Then
    decodeRecDataTerm (pac)
End If
If socApp.State = sckConnected Then
    decodeRecDataApp (pac)
End If
End Sub

Private Sub decodeRecDataTerm(pac As String)
Dim firstchar As String
Dim v As Single
Dim i As Single
firstchar = Left(pac, 1)
Select Case firstchar
Case "a"

```

```

        socTerm.SendData ("acknowledged" & vbCrLf)
        socTerm.SendData (vbCrLf) 'blank line
    Case "e"
        socTerm.SendData ("error #" & Chr(Right(pac, 1)))
        socTerm.SendData (vbCrLf) 'blank line
    Case "p"
        bpPWMvalue.StringValue = Mid(pac, 2, 2)
        socTerm.SendData ("PWM = " & bpPWMvalue.RawValue & vbCrLf)
        socTerm.SendData (vbCrLf) 'blank line
    Case "r"
        bpGetV.StringValue = Mid(pac, 2, 2)
        bpGetI.StringValue = Mid(pac, 4, 2)
        bpGetAmbTemp.StringValue = Mid(pac, 6, 2)
        bpGetBatTemp.StringValue = Mid(pac, 8, 2)
        'voltage measurements are with respect to 22V, 11A
        socTerm.SendData ("Voltage = " & Format(bpGetV.floatValue, "#0.0##") & vbCrLf)
        socTerm.SendData ("Current = " & Format(bpGetI.floatValue, "#0.0##") & vbCrLf)
        socTerm.SendData ("Ambient Temperature = " & Format(bpGetAmbTemp.floatValue,
"#0.0#") & vbCrLf)
        socTerm.SendData ("Battery Temperature = " & Format(bpGetBatTemp.floatValue,
"#0.0#") & vbCrLf)
        socTerm.SendData (vbCrLf) 'blank line
    Case "s"
        Dim mode As Integer
        Dim control As Integer
        mode = Asc(Mid(pac, 2, 1))
        control = Asc(Mid(pac, 3, 1))

        socTerm.SendData ("Mode = " & mode & vbCrLf)
        socTerm.SendData ("Control = " & control & vbCrLf)
        socTerm.SendData (vbCrLf) 'blank line
End Select
End Sub

Private Sub decodeRecDataApp(pac As String)
'this function decodes the com packet and sends responses
'suitable for the application
Dim firstchar As String
firstchar = Left(pac, 1)
Select Case firstchar
    Case "a"
        socApp.SendData ("a")
    Case "e"
        socApp.SendData ("e" & Chr(Right(pac, 1)))
    Case "p"
        bpPWMvalue.StringValue = Mid(pac, 2, 2)
        socApp.SendData ("p " & bpPWMvalue.RawValue & vbCrLf)
    Case "r"
        bpGetV.StringValue = Mid(pac, 2, 2)
        bpGetI.StringValue = Mid(pac, 4, 2)
        bpGetAmbTemp.StringValue = Mid(pac, 6, 2)
        bpGetBatTemp.StringValue = Mid(pac, 8, 2)
        'V wrt 22v and I wrt 11a
        socApp.SendData ("r " & bpGetV.floatValue & " " & _
bpGetI.floatValue & " " & bpGetAmbTemp.floatValue & " " & _
bpGetBatTemp.floatValue)
    Case "s"
        Dim mode As Integer
        Dim control As Integer
        mode = Asc(Mid(pac, 2, 1))
        control = Asc(Mid(pac, 3, 1))
        socApp.SendData ("s " & mode & " " & control)

```

```

End Select

End Sub

Public Sub socTerm_Close()
If socTerm.State <> sckClosed Then
    socTerm.Close
End If
RaiseEvent ConectionClosed(Terminal)
BeginListen
End Sub

Public Sub socApp_Close()
If socApp.State <> sckClosed Then
    socApp.Close
End If
RaiseEvent ConectionClosed(Application)
BeginListen
End Sub

Private Sub socTerm_ConnectionRequest(ByVal requestID As Long)
Dim msg As String
' Check if the control's State is closed. If not,
' close the connection before accepting the new
' connection.
If socTerm.State <> sckClosed Then socTerm.Close
socTerm.Accept requestID
msg = "Connected with " & socTerm.LocalHostName & vbCrLf
If com.Emulate Then
    msg = msg & "In emulation mode" & vbCrLf & vbCrLf
Else
    msg = msg & "On com port " & config.CommPort & vbCrLf & vbCrLf
End If
RaiseEvent ClientServiceRequest(Terminal, socTerm.RemoteHostIP, socTerm.RemotePort)
socTerm.SendData (msg)
sendHeader
lastTermRec = Now 'store timestamp of last rec packet for client inactivity timeout
detection
End Sub

Private Sub socApp_ConnectionRequest(ByVal requestID As Long)
If socApp.State <> sckClosed Then socApp.Close
socApp.Accept requestID
RaiseEvent ClientServiceRequest(Application, socApp.RemoteHostIP, socApp.RemotePort)
lastAppRec = Now 'store timestamp of last rec packet for client inactivity timeout
detection
End Sub

Private Sub sendHeader()
Dim msg As String
msg = "Enter command and press enter, ? for help" & vbCrLf
socTerm.SendData (msg)
End Sub

Private Sub sendHelpText()
Dim msg As String
socTerm.SendData ("Valid Commands:" & vbCrLf)
socTerm.SendData ("getdata - returns a sample" & vbCrLf)
socTerm.SendData ("getpwm - returns the current pwm value" & vbCrLf)
socTerm.SendData ("gets - returns current mode and control status" & vbCrLf)
socTerm.SendData ("setv arg - set the voltage setpoint, with arg=0-" & config.MaxVoltage
& vbCrLf)

```



```

socTerm.SendData ("seti arg - set the current setpoint, with arg=0-" & config.MaxCurrent
& vbCrLf)
socTerm.SendData ("seto arg - set the output on or off, with arg=0,1" & vbCrLf)
socTerm.SendData ("setc arg - set the controller, 0=device, 1=pc" & vbCrLf)
socTerm.SendData ("sett arg - set the temperature, with arg=0-" & config.MaxTemperature &
vbCrLf)
End Sub

```

```

Private Sub socTerm_DataArrival(ByVal bytesTotal As Long)
Dim strdata As String
Dim lm As Integer
Static message As String
socTerm.GetData strdata, vbString
'RaiseEvent socPacket(hexString(strdata))
lastTermRec = Now 'store timestamp of last rec packet for client inactivity timeout
detection
Select Case strdata
    Case vbBack
        lm = Len(message)
        message = Left(message, lm - (1 * lm))
    Case vbCr
        socTerm.SendData (vbCrLf)
        ParseCommandTerm (message)
        message = ""
    Case "?"
        socTerm.SendData (vbCrLf)
        sendHelpText
    Case Else
        message = message & strdata
End Select
End Sub

```

```

Private Sub socApp_DataArrival(ByVal bytesTotal As Long)
Dim strdata As String
lastAppRec = Now 'store timestamp of last rec packet for client inactivity timeout
detection

socApp.GetData strdata, vbString
ParseCommandApp (strdata)
End Sub

```

```

Private Sub ParseCommandTerm(msg As String)
'this function parses the command and encodes a device packet minus checksum
Dim arg As Variant
Dim val As Double
Dim ival As Integer
Dim pac As String
Dim command As String
On Error GoTo handle
arg = Split(msg, " ", 3)
If UBound(arg) >= 0 Then
command = Trim(UCase(arg(0)))
    lastSendTime = Timer
    Select Case command

        Case "SETV"
            val = arg(1)
            If val < 0 Or val > config.MaxVoltage Then GoTo handle 'argument invalid
            bpSetV.floatValue = val
            pac = "V" & bpSetV.StringValue
            'com.addPacket (pac)
        Case "SETI"

```

```

        val = arg(1)
        If val < 0 Or val > config.MaxCurrent Then GoTo handle    'argument invalid
        bpSetC.floatValue = val
        pac = "I" & bpSetC.StringValue
    Case "SETO"
        ival = arg(1)
        If ival < 0 Or ival > 1 Then GoTo handle    'argument invalid
        pac = "O" & Chr(ival)
    Case "SETC"
        ival = arg(1)
        If ival < 0 Or ival > 1 Then GoTo handle    'argument invalid
        pac = "C" & Chr(ival)
    Case "SETT"
        Dim bptemp As New BytePair
        Call bptemp.setScaleFactors(config.MaxTemperature)
        val = arg(1)
        If val < 0 Or val > 110 Then GoTo handle
        bptemp.floatValue = val
        pac = "V" & bptemp.StringValue
    Case "GETDATA"
        pac = "R"
    Case "GETPWM"
        pac = "P"
    Case "GETS"
        pac = "S"
        'com.addPacket (pac)

    Case Else
        GoTo handle
End Select
com.addPacket (pac)
End If
Exit Sub
handle:
socTerm.SendData ("Invalid command" & vbCrLf)
End Sub

Private Sub ParseCommandApp(msg As String)
'this function parses the command and encodes a device packet minus checksum
Dim arg As Variant
Dim val As Double
Dim ival As Integer
Dim pac As String
Dim command As String
On Error GoTo handle
arg = Split(msg, " ", 3)
If UBound(arg) >= 0 Then
command = arg(0)
    Select Case command
        Case "V"
            val = arg(1)
            If val < 0 Or val > config.MaxVoltage Then GoTo handle    'argument invalid
            bpSetV.floatValue = val
            pac = "V" & bpSetV.StringValue
            'com.addPacket (pac)
        Case "I"
            val = arg(1)
            If val < 0 Or val > config.MaxCurrent Then GoTo handle    'argument invalid
            bpSetC.floatValue = val
            pac = "I" & bpSetC.StringValue
        Case "O"
            ival = arg(1)

```

```

        If ival < 0 Or ival > 1 Then GoTo handle    'argument invalid
        pac = "O" & Chr(ival)
    Case "C"
        ival = arg(1)
        If ival < 0 Or val > 1 Then GoTo handle    'argument invalid
        pac = "C" & Chr(ival)
    Case "R"
        pac = "R"
    Case "P"
        pac = "P"
    Case "S"
        pac = "S"
        'com.addPacket (pac)
    Case Else
        GoTo handle
    End Select
    com.addPacket (pac)
End If
Exit Sub
handle:
socApp.SendData ("i")    'invalid command from application
End Sub

Private Sub socTerm_Error(ByVal Number As Integer, Description As String, ByVal Scode As
Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long,
CancelDisplay As Boolean)
If Number = 10053 Then
    frmMain dispText (Now & Description)
    socTerm_Close
End If
End Sub

Private Sub tmrLOCP_Timer()
Dim pac As String
Dim msg As String
Dim tmr As Single

Const Timeout = 60

If socTerm.State = sckConnected Then
    If Now > DateAdd("s", Timeout, lastTermRec) Then
        socTerm.SendData ("Client inactivity timeout after " & Timeout & " seconds " &
vbCrLf & "Closing connection" & vbCrLf)
        tmr = Timer
        Do
            'wait one second before closing connection
            DoEvents
        Loop While Abs(Timer - tmr) < 1
        frmMain dispText (Now & " Client inactivity timeout")
        socTerm_Close
    End If
End If

If socApp.State = sckConnected Then
    If Now > DateAdd("s", Timeout, lastAppRec) Then
        frmMain dispText (Now & " Client inactivity timeout")
        socApp_Close
    End If
End If

End Sub

```