

**LIBRARY
Michigan State
University**

This is to certify that the
thesis entitled

**DESIGN AND CONTROL OF A MULTILEVEL
INVERTER FOR ELECTRIC VEHICLES**

presented by

Arthur W. Matteson

has been accepted towards fulfillment
of the requirements for the

M.S. degree in Electrical Engineering

Michael Shumblatt
Major Professor's Signature

April 30, 2008
Date

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.
MAY BE RECALLED with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

DESIGN AND CONTROL OF A MULTILEVEL INVERTER
FOR ELECTRIC VEHICLES

By

Arthur W. Matteson

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

ELECTRICAL ENGINEERING

2008

ABSTRACT

DESIGN AND CONTROL OF A MULTILEVEL INVERTER FOR ELECTRIC VEHICLES

By

Arthur W. Matteson

Typical AC induction motor drive systems in electric vehicles use full-amplitude pulse width modulation inverters. These have many drawbacks including the requirement of a cooling system, lack of integrated battery management, and often non-integrated and/or low-power charging. There is also an issue of safety with a permanently connected series battery pack, and motor winding insulation is degraded in this type of controller. Moreover, the size and mass of these inverters reduce available vehicle storage space. An ideal solution is to provide a small inverter for each battery, so that its charge can be maintained individually and a low-distortion sine wave can be applied to the motor. A battery charger is then intrinsic in this layout. This work presents the development and implementation of a such a multilevel inverter. Two separate designs are constructed, each including a master module and six slave modules. Results from driving a GM EV₁ using the second design are presented. Hardware and software designs are attached, including PC interface code in 'C++' and algorithm simulation code in Python.

ACKNOWLEDGMENTS

I would first like to thank my advisor, Dr. Michael Shanblatt, and committee members, Dr. Norbert Mueller and Dr. Shantanu Chakrabartty. Dr. Shanblatt has been a great influence, especially during my undergraduate years here at Michigan State, by providing me the rare opportunity to conduct an individualized senior design project with Texas Instruments. The knowledge and experience I gained from this project are invaluable. I have also enjoyed working with Dr. Mueller on the Solar Car Team, of which he is the advisor, and on unrelated projects in his laboratory.

I would also like to thank Joel Anderson, Daniel West, and the rest of the Solar Car Team for their support. Joel originally told me of multilevel inverters; without him this project probably would not have been done. Daniel's assistance in construction of the inverter and his loyalty have been extremely valuable, and enough appreciation cannot be shown to him.

Furthermore, I would like to thank my friend Paul Kucher, who has been of immense support during the writing of this thesis. He provided me with L^AT_EX, Python, and matplotlib assistance, and his help made my authoring much more efficient. He also provided general moral support and advice on this work, from inception to completion.

My parents, Robert and Anne Matteson, my brother, Andrew Matteson, and my grandfather, Donald Matteson, have all been of help during this project. Andrew reviewed this document very thoroughly and provided great input to revise it to its final state. Donald gave me the opportunity to work on his own electric vehicle, from which I gained vast experience.

TABLE OF CONTENTS

LIST OF TABLES	vi
LIST OF FIGURES	vii
LIST OF ABBREVIATIONS	ix
1 Introduction	1
1.1 Shortcomings of Electric Vehicle Power Electronics	1
1.2 The Multilevel Inverter as a Solution	3
1.3 Requirements and Difficulties of Implementation	5
1.4 Approach to Overcome Challenges	6
2 Algorithm Development	8
2.1 Backbone Implementation Overview	8
2.2 Communication Design	11
2.3 Battery Selection Algorithm Design	15
2.3.1 Theory	15
2.3.2 Single-Cycle Discharge Distribution	17
2.3.3 Algorithm Evaluation	21
2.4 Battery Charging	27
2.5 Motor Control	30
3 Slave Module Development	31
3.1 Hardware Design One	31
3.1.1 Power Section	31
3.1.2 Microcontroller	34
3.1.3 DC/DC Converter	36
3.1.4 Printed Circuit Board	37
3.2 Redesign Considerations	39
3.3 Hardware Design Two	40
3.3.1 Power Section	40
3.3.2 Microcontroller	41
3.3.3 DC/DC Converter	41
3.3.4 Printed Circuit Board	43
3.4 Software	45
3.4.1 Main Loop	45
3.4.2 Serial Interrupt	45
3.4.3 Bootloader	48

4	Master Module Development	49
4.1	Hardware Design One	49
4.2	Hardware Design Two	52
4.3	Software Design One	54
4.3.1	Main Loop	54
4.3.2	USB Interrupt	56
4.3.3	Bootloader	57
4.4	Software Design Two	58
4.4.1	PIC18F2553 Code	58
4.4.2	DSC Main Loop	59
4.4.3	DSC Bootloader	59
4.4.4	Serial Port Interrupts	59
5	Results	60
6	Conclusions	67
6.1	Design One	67
6.2	Design Two	67
6.3	System and Topology	68
6.4	Future Work	68
	APPENDICES	71
A	PC Interface Software	71
A.1	Overview of Software	71
A.2	User Interface Layout: monitor.ui	73
A.3	Header File: monitor.h	78
A.4	Main Code: monitor.cpp	80
B	Slave Printed Circuit Board Layouts	121
C	Battery Discharge Distribution Python Script Example	128
D	Slave Assembly Code, Design Two	132
E	Master Code Listings	152
E.1	DSPIC30F4011 USB Assembly Code, Design One	152
E.2	PIC18F2553 Full Assembly Code, Design Two	173
E.3	DSPIC30F4011 Full Assembly Code, Design Two	182
	BIBLIOGRAPHY	230

LIST OF TABLES

2.1	Battery Selection Algorithm Simulation Results.	24
3.1	1 Megabaud Command Enumeration.	47
3.2	125 Kilobaud Bootloader Command Enumeration.	48

LIST OF FIGURES

1.1	Basic Overview of the Multilevel Inverter.	4
2.1	Positive Third Harmonic Injection Amplitude Increase.	10
2.2	Negative Third Harmonic Injection Amplitude Decrease.	10
2.3	Design One's Elementary Serial Transmission Buffering.	12
2.4	Worst-case Communication Bus Loading.	12
2.5	Battery Charge Regulation, Equal Charge.	16
2.6	Battery Charge Regulation, Inequal Charge.	16
2.7	Line-to-neutral Generated Waves.	18
2.8	Line-to-line Generated Waves.	18
2.9	Discharge Distribution, No Third Harmonic, $\sigma=0.112$	19
2.10	Discharge Distribution, Positive Third Harmonic, $\sigma=0.126$	19
2.11	Discharge Distribution, Negative Third Harmonic, $\sigma=0.053$	20
2.12	Discharge Dist., 8 Batteries, PF=0.85, Non-hybrid Pack.	22
2.13	Time-Spent-On Dist., 8 Batteries, PF=0.85, Non-hybrid Pack.	22
2.14	Discharge Dist., 16 Batteries, PF=0.85, Hybrid Pack.	23
2.15	Time-Spent-On Dist., 16 Batteries, PF=0.85, Hybrid Pack.	23
2.16	Discharge Dist., Battery One Initially 30 Charge Units Depleted.	25
2.17	Time-Spent-On Dist., Battery One Initially 30 Charge Units Depleted.	25
3.1	Slave Module Design One Schematic.	38
3.2	Slave Module Design Two Schematic.	44
3.3	Communication Bus Byte Format.	46
3.4	Slave Module Automatic Frequency Calibration.	46
4.1	Master Module Design One Schematic.	51
4.2	Master Module Design Two Schematic.	53
5.1	Slave Module Output Current Waveform, 130A Peak.	61
5.2	Slave Module MOSFET Avalanche Waveform, 23V Peak.	61
5.3	Slave Module PWM Illustrating an 800ns Dead Time.	62
5.4	Line-to-line Inverter Output Voltage, Six Total Slave Modules.	62
5.5	Design-One Master Multitasking During a USB Transfer.	63
5.6	EV ₁ Captured Motor Variables, $I_d^*=8A$, Slight Loading.	65
5.7	EV ₁ Captured Motor Variables, $I_d^*=6A$, Three Loadings.	65
5.8	EV ₁ Captured Motor Variables, $I_d^*=40A$, Driving.	66
5.9	EV ₁ Captured Motor Variables, $I_d^*=40A$, Driving.	66
A.1	Monitor v0.5 Screenshot with Six Battery Modules	72

B.1 Slave Design One PCB Top Layer 122

B.2 Slave Design One PCB Bottom Layer 123

B.3 Slave Design Two PCB Top Layer 124

B.4 Slave Design Two PCB Power Layer 125

B.5 Slave Design Two PCB Ground Layer 126

B.6 Slave Design Two PCB Bottom Layer 127

Images in this thesis are presented in color.

LIST OF ABBREVIATIONS

A/D	Analog to Digital
AGM	Absorbed Glass Mat
AWG	American Wire Gauge
CRC	Cyclic Redundancy Check
DIP	Dual In-line Package
DPST	Double-Pole, Single-Throw
DSC	Digital Signal Controller
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIFO	First In, First Out
FOC	Field-Oriented Control
ICE	Internal Combustion Engine
IGBT	Insulated-Gate Bipolar Transistor
kBd	Kilobaud (thousand bits per second)
LED	Light-Emitting Diode
MBd	Megabaud (million bits per second)
MIPS	Millions of Instructions per Second
MLCC	Multilayer Ceramic Capacitor
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
PCB	Printed Circuit Board
PFC	Power Factor Correction
PWM	Pulse Width Modulation
SOC	State of Charge
SPST	Single-Pole, Single-Throw
THD	Total Harmonic Distortion
μ C	Microcontroller

CHAPTER 1

Introduction

1.1 Shortcomings of Electric Vehicle Power Electronics

Factory-built electric vehicles (EVs) such as the GM EV₁ or Toyota RAV4-EV used a high-voltage battery pack consisting of many series cells, nominally 300V or above. An inverter employing full-amplitude pulse-width modulation (PWM) drove the main motor in both cases. PWM was used to generate a set of time-averaged, near-sinusoidal voltages for the motor. This configuration has many drawbacks. A battery pack of this potential is extremely hazardous when permanently connected in series. Many active and passive safety mechanisms must be in place to prevent electric shock to a service person. The electromagnetic interference caused by the PWM is substantial, and the signals are not healthy for the motor because of wire insulation breakdown (high dV/dt) and magnetic core loss. The efficiency of these insulated-gate bipolar transistor (IGBT) inverters is typically around 95% [1]. Unfortunately, IGBTs are not optimized to switch at low loads — those occurring most often in an electric vehicle — because of their relatively constant voltage drop. Some users find a low PWM frequency acoustically offensive, though it is more efficient. There are also issues with reliability because of the fast switching speeds and high voltages.

The charging system of these vehicles also had limitations. Though it is possible to use the motor as a boost inductor to do power factor correction (PFC) [2], the

input voltage is limited to about 265VAC (only one-phase), and neither the EV₁ nor RAV4-EV employed this technique. Both factory-built vehicles needed a large, expensive, external converter unit for quick charging, and this was only available in a few selected areas where it had been chosen to be installed. EV₁ owners were obliged to purchase a charging unit for their garage for over \$2,000 after installation costs [3]. Management and balancing for a series-cell pack of about 300V requires at least some repeated circuitry for each battery, or many wires routed to a main control unit (EV₁). Finally, the main power inverter and its liquid cooling system took up a considerable amount of space, which was already at a premium in the two-seat EV₁ (about 32x14x5in. under the hood was needed by the inverter alone).

AC Propulsion, a small company based in California, provides inverter/motor combinations for EVs. Their second-generation AC-150 is a 150kW IGBT inverter paired with an equally powerful custom-design induction motor, both air-cooled [2]. It contains a charger capable of up to 20kW, and uses the motor as a boost inductor to correct its power factor seen at an outlet. It also includes a powerful DC/DC converter for 12V accessories. It is capable of discharging the vehicle's batteries onto the power line, features traction control, and operates at a peak efficiency of 91%.

The second-generation AC-150's mass is 80kg, impressive for a system of that power considering a typical 20HP industrial motor is more massive [4], except that the large-dimension inverter constitutes 30kg of that. With EVs already stuffed full of batteries it is a challenge to find a large enough space for such a device. This 150kW power level provides performance comparable to a 300HP internal combustion engine (ICE). AC Propulsion's Lithium-Ion T-Zero demonstration vehicle, which uses the AC-150, accelerates from 0–60MPH in only 3.6 seconds [5]. This is not necessary for most vehicles, especially those with non-advanced batteries, but this system provides no means of downscaling without a redesign. The datasheet makes no claims of individual battery management capability, and the system does not support hybrid-

chemistry battery packs — which could make EVs more affordable, flexible, and upgradable — or at least not hybrid packs with differing capacity modules.

Still, the AC-150 sets a bar for performance to be matched or exceeded. PFC is basically a necessity for quick recharging. Regenerative braking is also extremely desirable for EVs; most DC-motor EVs do not have it. Smooth torque control is important in a replacement design. The 85-90% efficiency should be met or exceeded, especially under typical driving conditions. An alternative inverter should be smaller and manage the batteries as well as or better than a typical AC EV setup. There should also be at least the availability of a 150kW power output, but also the ability to meet demands of all sizes at a reasonable cost.

1.2 The Multilevel Inverter as a Solution

A multilevel power inverter is a system that can produce a varying output voltage from a given number of interconnected sources without the need for full-amplitude PWM [6]. The multilevel topology being considered here, intended exclusively for a drive motor application in an electric vehicle, consists of many batteries in series through full-bridge inverters that can act as three-state sources: positive twelve volts, zero volts, or negative twelve volts (roughly). In this design, the number of voltage output levels for each line-to-neutral voltage is twice the number of batteries per phase plus one. Typical drive motors are three-phase induction machines; this type will be the focus in this project. For this motor, the pack's batteries will be divided into three groups, all connected together at one end and connected to the three motor phases at the other end. A basic schematic can be seen in Figure 1.1. The aim of this thesis is to develop a control algorithm along with a complete implementation for this setup. The major goal is to achieve the highest possible output frequency while managing the batteries to have the highest range capacity and the most equal charge, while keeping hardware costs to a minimum.

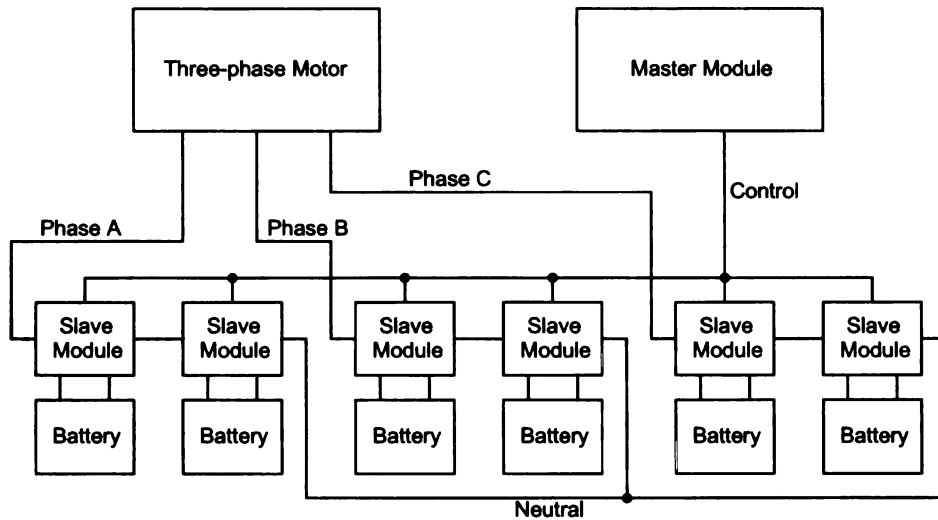


Figure 1.1. Basic Overview of the Multilevel Inverter.

The multilevel topology suggested here solves many of the problems associated with traditional PWM drives. The inverter modules (or battery modules) measure only 3.7x2.7x0.5in. each after thermal epoxy potting and conveniently sit atop each sealed lead-acid battery. Only a small amount of extra space is needed for other components, including the master controller. Since the heat loss is distributed, no active cooling mechanism is needed. Instead of IGBTs, metal oxide semiconductor field-effect transistors (MOSFETs) are used for this low-voltage application. MOSFETs are preferred because of their low losses, both conductive and switching, and will give the inverter an efficiency of 98% or above at typical power levels. The switching frequency is typically in the range of 300Hz, compared with 5–20kHz for a PWM inverter. The dV/dt going to the motor will be about $50V/\mu s$ as compared with $1,000V/\mu s$, prolonging its reliability and decreasing its heat loss.

A high-power charging system is inherent in the design of this multilevel inverter with the addition of a small inductor; contactors can switch the three module groups in series to support up to 480VAC directly with PFC (no external hardware needed). Battery balancing — even charging a weak battery — can be done during motoring by the control algorithm. A weak and/or damaged battery can be totally bypassed, a

feat impossible in the EV₁ or RAV4-EV. The highest voltage in a parked vehicle will be the highest battery voltage (about 12V). Finally, a high-power DC/DC converter will be distributed across the battery pack, already built into the battery modules. As few as 15 or as many as 120 of these modules will be supported.

1.3 Requirements and Difficulties of Implementation

In general, this topology increases the burden on software, and presenting a design challenge for the master control processor. Many tasks need to be implemented in the master. First of all, *module tokens* need to be generated to send on the serial bus, in the end form of three sinusoids. A module token usually consists of a seven-bit module address, a two-bit module command, and optional nine-bit arguments. The module command tells the addressed battery module to either turn on positive, turn on negative, turn off, or watch the bus for more extended data (arguments). In the first master design, these tokens are fed through a software FIFO (first in, first out) buffer so that they still can be sent out during an interrupt, though they typically will be sent out in the main loop. The control algorithm must take into account the charge present on each battery. More highly-charged batteries are selected more often to come out of an off state, so that the charge will be equalized among all. The same algorithm also attempts to keep the battery current as level as possible, though ripple is inherent in this hardware design.

There needs to be some means of controlling the motor past a simple V/Hz algorithm to help its dynamic response and efficiency. This involves many calculations and measurements, mainly measuring and processing the three motor phase currents. The microprocessor must devote time to read back accurate battery voltages from the modules, and at some point while the vehicle is parked, calibrate the modules against an accurate master battery voltage measurement by turning on only one at a time. The processor must also control accessories such as contactors and read the status

of the ignition, while monitoring temperatures. Finally, a communication interface to a modern computer and an accompanying multi-platform software package is a necessity for setup and troubleshooting.

1.4 Approach to Overcome Challenges

The processor choice for the multilevel inverter master control unit is the dsPIC30F4011 from Microchip Technology, Inc [7]. It is termed a *digital signal controller* (DSC) and is intended for use in motor control applications. It has a 10-bit, 12-input analog to digital (A/D) converter operating at up to 500kSPS (thousand samples per second). The DSC accepts up to 5V for main power and comes in a 40-pin dual in-line package (DIP), ideal for prototyping. It features a high-speed serial port — up to 1.875 megabaud (MBd) without overclocking — necessary for sending data to the individual inverter modules. It nominally runs at 30 million instructions per second (MIPS), comparatively fast for Microchip products. It has 83 base instructions each a minimum of 24 bits wide (mostly single-cycle), 48kB of Flash program memory, and 2kB of RAM. It also features predecrement, preincrement, postdecrement, and postincrement by one, two, or four for byte, word, or long addressing respectively, as well as other addressing modes — mainly useful for signal filtering.

Assembly language is used exclusively to program this DSC and the accompanying Atmel microcontrollers (μ Cs) inside the battery modules. Typical 'C' coding is relatively inefficient at memory and cycle usage. Memory and time resources will be at a premium due to the complex, real-time tasks this master is asked to perform. One of the only benefits of using 'C' would be portability, but that is not a concern at this time; developing and demonstrating the algorithm is the main goal.

A debugging and/or data collection interface for any complex system such as this one is an important inclusion. For the multilevel inverter, a USB connection is chosen. Nearly every computer has at least one USB port and availability is especially

important in a mass-produced design. USB includes error checking and flow control over only three wires. A fourth wire (+5V) can also signal to the DSC when the plug is inserted. Version numbers and strings can be built into the USB descriptors, so the user knows immediately which system they have after connection. Finally, master and slave firmware upgrades are possible through USB.

Unfortunately, the DSC does not include a USB peripheral. Due to the hardware and cost minimization efforts, it initially was decided to manually implement 1.5MBd USB in software. The Microchip PIC18F67J50 does include a full-speed (12MBd) USB peripheral and fast serial ports, but is only an eight-bit processor, comes only in surface-mount packages, and runs at 3.6V maximum [8]. The dsPIC30F4011 is a 16-bit processor, and its instruction set is far more powerful, even supporting a 17x17-bit multiply. There are (16) 16-bit working registers, while the 18F67J50 has only one eight-bit working register, comparatively. There are also 21 types of conditional branches in the DSC, and it has two 40-bit saturating accumulators for higher math. Finally, it runs at 30MIPS vs. the 18F67J50's 12MIPS.

In the second master design, a PIC18F2553 is added to the circuit since there were reliability issues with the DSC's bit-banging implementation. The 18F2553 is capable of full-speed USB, and allows a computer to record 64 bytes of inverter status data every millisecond. Included in the 64 bytes is the current status of every module — off, on positive, on negative, or performing PWM. The 30F4011 and 18F2553 communicate over a 2MBd serial bus. Though the first master design used a module token buffer, this is not necessary in the second version because the serial communications are received in hardware, not software. More details can be found in Chapter 4.

CH

Alg

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

CHAPTER 2

Algorithm Development

2.1 Backbone Implementation Overview

Individual battery modules in the pack must be turned on and off at specific time intervals to produce synthesized sine waves. This requires the modules to be connected by some sort of a communication bus. The DSC must calculate the time intervals for each module to be on and off, accounting for individual battery state of charge (SOC) and other factors. A motor control algorithm must work on top of all this, maximizing the efficiency of the drive system at all times. Simultaneous communication to a computer through USB is also a necessity.

The synthesized sine waves put out by the inverter set should support both positive and negative third-harmonic injection. This technique is considered because sometimes it could be used in the positive form to increase the line-to-line voltage at the motor for high-speed operation, allowing the practical use of 48 battery modules for a two-pole, 230VAC motor capable of around 4,500RPM. It also could be used in the non-traditional, negative form to spread out the charge among batteries by reducing the modulation index, though this depends on the chosen battery selection algorithm. A depiction of the ‘voltage-extending’ property of positive third-harmonic injection can be seen in Figure 2.1. An opposite injection can be seen in Figure 2.2.

AC induction motors are typically controlled with an approximately constant V/Hz ratio because of their nearly linear impedance increase with higher frequency operation. Typical motors, like the 15HP EM3314T from Baldor [9], require a three-phase source of $230V_{\text{rms}}$ line-to-line nominal at 60Hz. At 30Hz this motor would require about $115V_{\text{rms}}$ for the same torque output. A traditional PWM drive would require $230V_{\text{rms}}*\sqrt{2} = 325V$ of DC bus to operate at 60Hz, using either positive third harmonic injection or space vector PWM. This is equivalent to about 28 lead-acid batteries under load. To operate with full torque at 4,500RPM, this motor's maximum safe speed, about 36 batteries would be required. The three sections of the multilevel converter are connected in a wye fashion, so each section's output is a line-to-neutral voltage. These subtract to create line-to-line voltages as seen in Figures 2.1 and 2.2. The greatest attainable line-to-line voltage, shown in Figure 2.1, has a peak twice that of the line-to-neutral voltage. Therefore, for a $230V_{\text{rms}}$ motor at 60Hz, the necessary line-to-neutral voltage peak is about 163V or 14 batteries — half that of the PWM inverter for each section, for a total of 42. To operate at 4,500RPM, a total of 54 batteries would be needed. The quotient of the RMS line-to-line voltage divided by the peak line-to-neutral voltage is $\sqrt{2} \simeq 1.414$ for positive third harmonic injection, $2/\sqrt{6} \simeq 0.816$ for negative third harmonic injection, and $\sqrt{3}/2 \simeq 1.225$ for no injection, giving a total range of $\sqrt{3} \simeq 1.732$ by shape alteration alone.

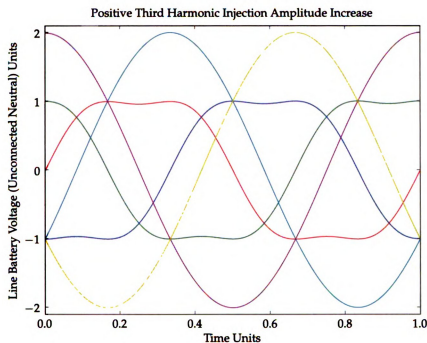


Figure 2.1. Positive Third Harmonic Injection Amplitude Increase.

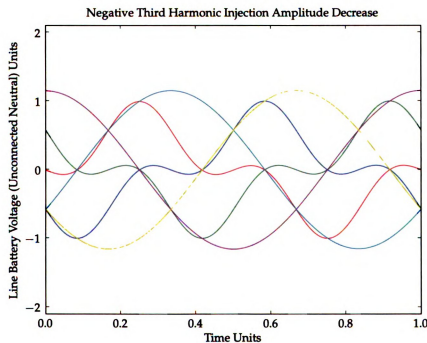


Figure 2.2. Negative Third Harmonic Injection Amplitude Decrease.

2.2 Communication Design

The DSC's asynchronous serial port features nine-bit transmission and reception. This is handy for addressing up to 128 objects along with a two-bit command. For safety, values 0 and 127 do not address modules, but perform other functions instead: an address of 0 tells all μ Cs to enter their bootloaders, and an address of 127 tells all μ Cs to recalibrate their clocks to the DSC's bit widths. A command of '11' (two passive bits) means turn off the addressed module, '01' means turn on positive, '10' means turn on negative, and '00' means data is to follow. Since the dsPIC is a 16-bit processor, these nine-bit values can be stored in a memory word by themselves. Modules carry an address of 1–120, and values 121–126 have other functions, such as “blink all module LEDs.”

Because of the need for USB and reception from modules, the design-one multilevel inverter requires two separate types of buffering, one done inside the DSC and the other done inside the μ Cs of the battery modules (Atmel ATTiny24s). Timestamps alongside the module tokens allow the DSC's USB interrupt routine to send out bytes when delaying for a bit transition on the USB data lines D+ and D-, no matter which way data is flowing on the lines. This buffer is 256 bytes long. By sending out two sets of nine-bit words, the DSC can also give the μ Cs a relative time in the future to turn on or off — a delayed command. This second type of buffering allows a module to send data back to the DSC (battery voltage) while delayed commands are being executed by the other modules. Both types of buffering are crucial to glitchless operation. An example of the latter type of buffering can be seen in Figure 2.3. The application of design-one DSC's FIFO buffer is described in Section 4.3.

The design-two inverter does not require the use of either buffering type because of the module serial bus' pseudo-full-duplex operation. In addition, USB is done externally to the dsPIC30F4011. The buffering concept is still being illustrated here for academic purposes. Results of both designs are discussed in Chapter 5.

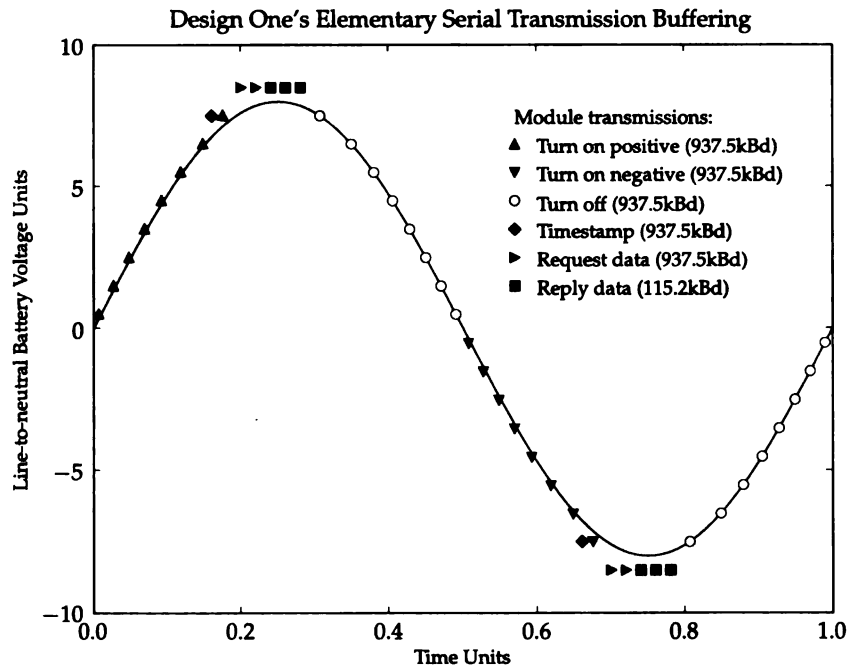


Figure 2.3. Design One's Elementary Serial Transmission Buffering.

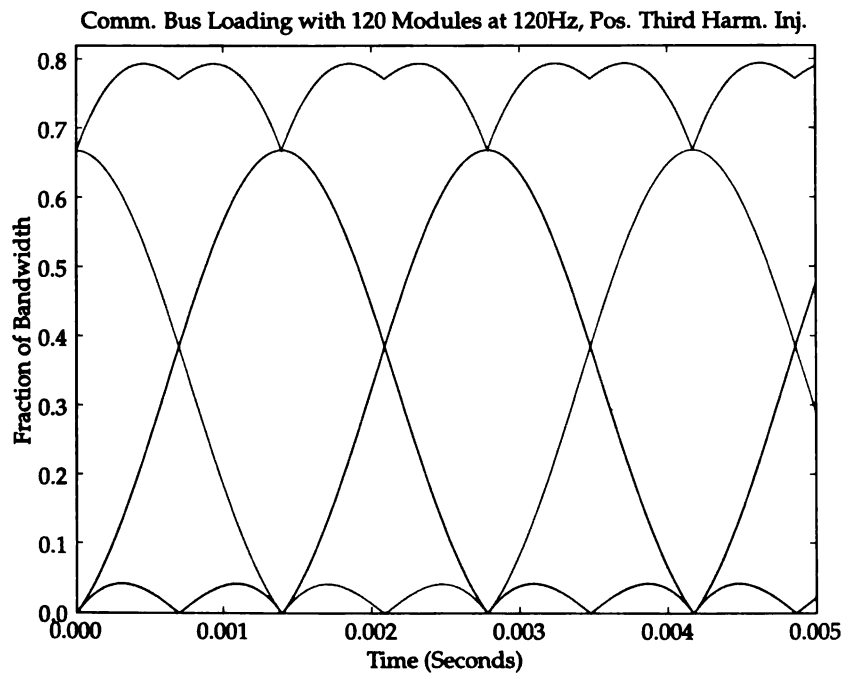


Figure 2.4. Worst-case Communication Bus Loading.

Figure 2.4 depicts the worst-case scenario of bus loading: a full 120 modules switching to make full-amplitude waves at 120Hz with positive third-harmonic injection. The topmost curve is the sum of the magnitudes of the output voltages' rates of change. The higher the rates of change, the more messages that need to be transmitted on the bus. Normally, only a one-word module token would be needed to command a slave module to turn its output on or off (slave buffering requires two-word tokens: one command and one timestamp).

The multilevel inverter output voltages in this scenario can be approximated in terms of equal battery voltage increments by:

$$U_{3pinj} = \frac{80}{\sqrt{3}} \left[\sin(240\pi t) + \frac{\sin(720\pi t)}{6} \right] \text{ battery voltage units,} \quad (2.1)$$

$$V_{3pinj} = \frac{80}{\sqrt{3}} \left[\sin\left(240\pi t + \frac{2\pi}{3}\right) + \frac{\sin(720\pi t)}{6} \right] \text{ battery voltage units,} \quad (2.2)$$

$$\text{and } W_{3pinj} = \frac{80}{\sqrt{3}} \left[\sin\left(240\pi t + \frac{4\pi}{3}\right) + \frac{\sin(720\pi t)}{6} \right] \text{ battery voltage units.} \quad (2.3)$$

The magnitude of the derivatives can be represented as follows:

$$U_{3pinj}' = \frac{19200\pi}{\sqrt{3}} \text{ abs} \left[\cos(240\pi t) + \frac{\sin(720\pi t)}{2} \right] \text{ messages/s,} \quad (2.4)$$

$$V_{3pinj}' = \frac{19200\pi}{\sqrt{3}} \text{ abs} \left[\cos\left(240\pi t + \frac{2\pi}{3}\right) + \frac{\sin(720\pi t)}{2} \right] \text{ messages/s,} \quad (2.5)$$

$$\text{and } W_{3pinj}' = \frac{19200\pi}{\sqrt{3}} \text{ abs} \left[\cos\left(240\pi t + \frac{4\pi}{3}\right) + \frac{\sin(720\pi t)}{2} \right] \text{ messages/s.} \quad (2.6)$$

The total approximate bus loading is the sum of these derivatives divided by the maximum number of messages per second. With a design-one serial line running at 937.5kBd, and nine-bit data with one start bit and two stop bits, 78,125 messages can be sent every second. The bus loading with no slave module buffering is therefore shown as the topmost curve on Figure 2.4. This is given by:

$$BusLoad_{3pinj} = \frac{U_{3pinj}' + V_{3pinj}' + W_{3pinj}'}{78125} \text{ fraction of bandwidth.} \quad (2.7)$$

It should be recognized that for 120 modules at 120Hz — which corresponds to either 3,500RPM or 7,000RPM at 600VAC line-to-line, nearly 150kW/200HP in a typical setup — the bus loading never exceeds 80%. Condensing these messages to the beginning of a bus loading cycle gives nearly 75% bus loading average in design one, leaving about 25% bandwidth open for other data. This extra time can be used to receive data from modules at 115.2kBd, up to five bytes' worth theoretically. A method was devised to accomplish this; however, it was never fully coded in design one. In design two, the ATTiny24s can reply at the full 1MBd, so no module token buffering nor condensation is necessary for proper operation. Though the average forward bus loading is reduced to less than 70%, a practical safe maximum output frequency is no higher than 125Hz for 120 modules and 250Hz for 60 modules.

2.3 Battery Selection Algorithm Design

2.3.1 Theory

Some type of algorithm to decide when each battery module should turn on or off is necessary to support variable-frequency and variable-amplitude waves. Three duplicate algorithms are needed, one for each motor phase. The inputs to the algorithms must be line-to-neutral phasors — either $\sin(t)$, $\sin(t) + \sin(3t)/6$, or $\sin(t) - \sin(3t)/2$, each shifted by 120 and 240 degrees of fundamental for algorithms two and three respectively. The processing between input and output is somewhat complicated, needing to take into account the charge remaining on each battery.

In an ideal environment, each battery contains an equal amount of charge and therefore should be discharged the same amount as the rest. A visualization of one algorithm's output is shown in Figure 2.5. It resembles a pyramid, where each battery is on for an equal amount of time. An equal charge will be taken out of the batteries in this case only for a constant-current load, which is quite different from the actual load — but the general technique can still be demonstrated.

For a pack of inequally charged batteries, a scheme such as that shown in Figure 2.6 could be used. Battery 3 has the highest remaining charge and Battery 4 has the lowest. Therefore, the amount of time Battery 3 is switched on is longer than that of Battery 4 (19 vs. 13 time units respectively). The disadvantage of such a scheme is the complication in attaining the goals set forth: 1) battery modules should only switch at most two times a half-cycle to reduce bus loading, and 2) there should be sufficient time between module transitions to keep inductive ringing to a minimum. The battery on-times have to be carefully planned inside the 'pyramid' waveform to remain contiguous. Because of the variety of output shapes that need to be accommodated, a lookup table for every amplitude and pack battery count is not practical. Even apart from these issues, only a constant current will produce the

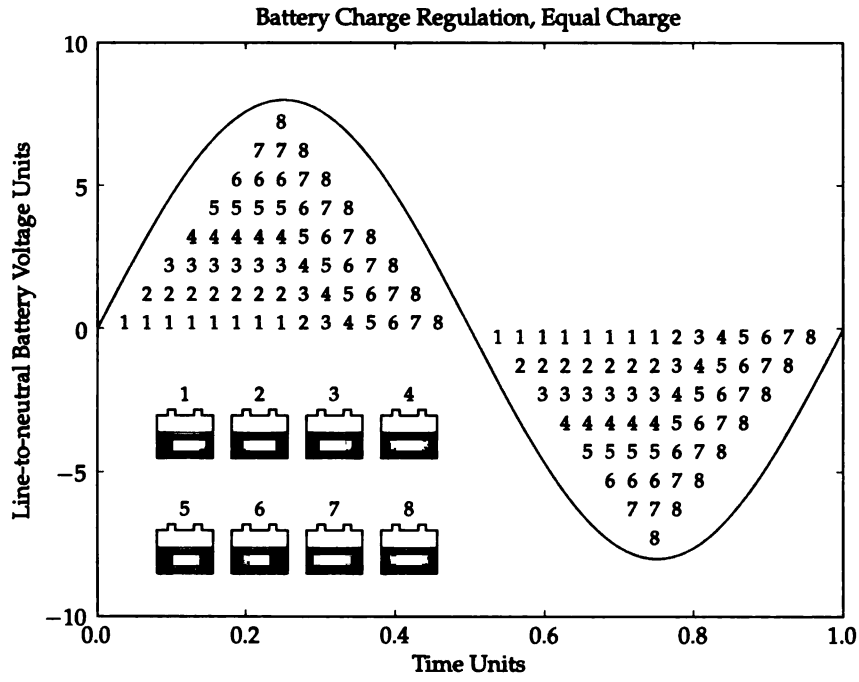


Figure 2.5. Battery Charge Regulation, Equal Charge.

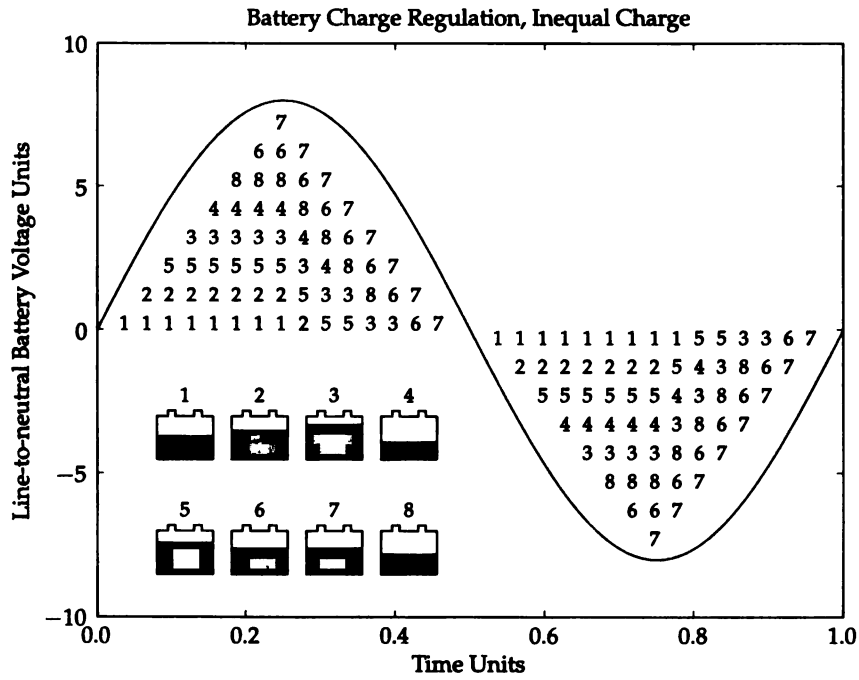


Figure 2.6. Battery Charge Regulation, Inequal Charge.

desired results. In conclusion, analysis needs to be done to derive the true discharge distribution for any given output waveform.

2.3.2 Single-Cycle Discharge Distribution

Figure 2.7 depicts the synthesized output voltage for a single-phase 40-battery module string, the largest to be accommodated. Figure 2.8 shows the associated difference — shifted only for clarity — between two subtracted line-to-neutral waveforms. Though the peak voltage in the former figure varies, the amplitudes of the differences do not. This is accomplished by third-harmonic injection. The switching scheme used here is similar to that in Figure 2.5, where batteries are turned on and off in the same order, with the exception of the small negative excursion on the negative third-harmonic injection wave. All of the graphs presented here have been generated in the Python language using matplotlib under Linux. This allows simulations to be carried out in a completely textual environment, much faster for development than a large, complex package such as Mathworks’s MATLAB or Simulink. The same matplotlib can be embedded in the open-source software interface in Trolltech’s Qt4, useful for comparing empirical to theoretical results. A selected script, used for the following section’s simulations, is included in Appendix C.

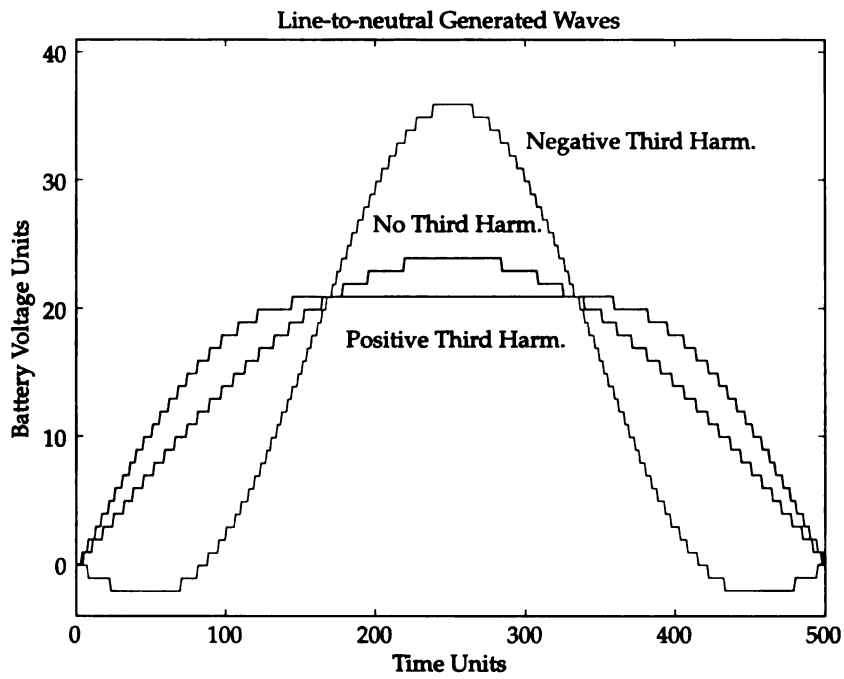


Figure 2.7. Line-to-neutral Generated Waves.

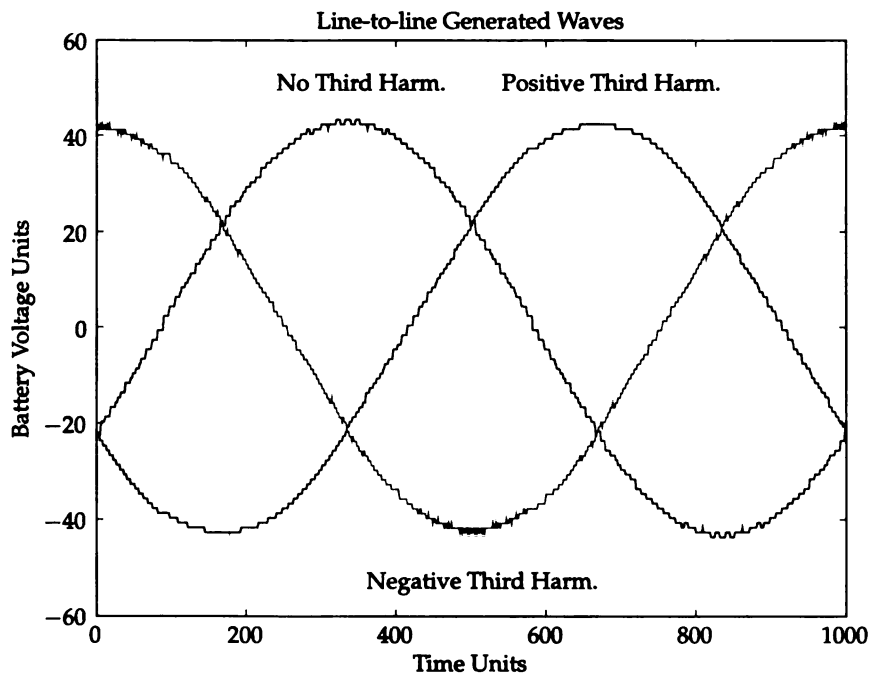


Figure 2.8. Line-to-line Generated Waves.

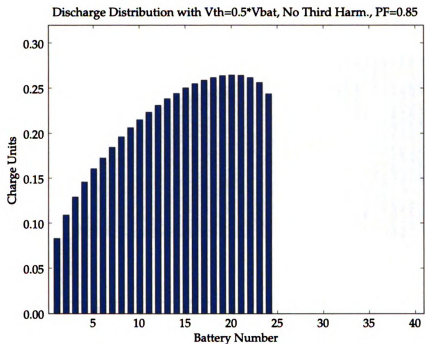


Figure 2.9. Discharge Distribution, No Third Harmonic, $\sigma=0.112$.

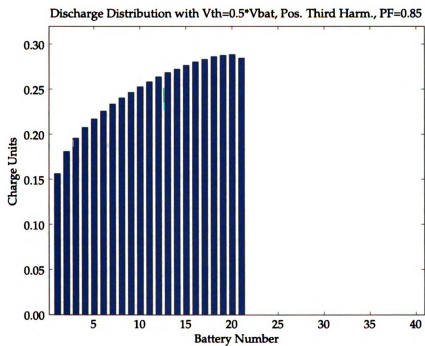


Figure 2.10. Discharge Distribution, Positive Third Harmonic, $\sigma=0.126$.

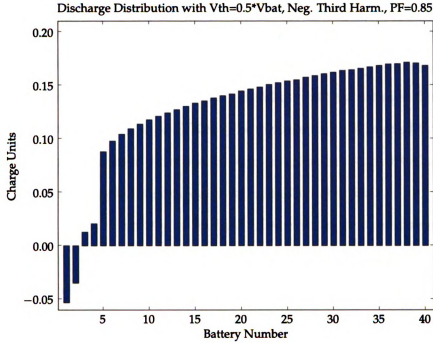


Figure 2.11. Discharge Distribution, Negative Third Harmonic, $\sigma=0.053$.

Figures 2.9, 2.10, and 2.11 show the distribution of charge taken out of the batteries for a half-cycle. Notice the much lower standard deviation (σ) for the negative third-harmonic injection. It would provide a way of spreading out the charge distribution for a simplistic algorithm. See the end of this chapter for a conclusion regarding injection. The unusual effects for the first four batteries in this graph are due to the negative excursion in Figure 2.7. During the second negative excursion, the involved batteries are actually charged instead of discharged (batteries #1 and #2). This can be applied to the weak cells in the pack, if any exist, or to random batteries otherwise. For algorithm simplicity, no battery will be used twice in a half-cycle.

Neglecting non-ideal effects such as battery manufacturing and/or temperature mismatch, each battery should be chosen for the #1–40 spot an equal amount of time so that all are discharged equally. This potentially could be done with a sort of shift register, but the problem would occur when less than 40 batteries were needed. Furthermore, transitioning between two different shift register lengths would require

various initializations so that the first battery would not be overly discharged.

Even with a pack of closely matched batteries, there will be small variances in capacity and SOC for which the selection algorithm needs to adjust. During operation, the master module will read the individual battery voltages from the slave modules and adjust their estimated SOC's. The entire system is limited by the lowest-SOC battery, so the most ideal situation is to have all at the same SOC all the time. This requires a more complex algorithm than a random chooser.

For flexibility, various battery capacities of the same or a different chemistry in the same string could be accommodated. Potentially a hybrid pack could be constructed to increase range per weight, containing both lead-acid and lithium-ion cells. Connecting different capacity batteries in series would be unhelpful for increasing range in a series pack, like the one the EV₁ used. The ability to use a combination of various chemistries together without any system modification besides an electrically erasable programmable read-only memory (EEPROM) data reprogram represents a major advantage of the multilevel inverter, especially as lithium-ion batteries are still impractically expensive in large quantities.

2.3.3 Algorithm Evaluation

An experimental algorithm was developed in and tested with Python and matplotlib. The single-phase routine turns on the highest-SOC batteries first and turns off the lowest-SOC batteries first. Note that SOC is unitless and is found by dividing remaining charge by capacity. Figures 2.12 and 2.13 depict the discharge and time-spent-on distribution for eight batteries producing a sine wave with a maximum line-to-neutral amplitude of eight battery voltage units. The graphs represent 100 fundamental cycles (about 50Hz), which corresponds to only two seconds. Figures 2.14 and 2.15 show distributions for a hybrid pack of 12 lead-acid and four lithium-ion batteries, where the lithium-ion have twice the capacity of the lead-acid.

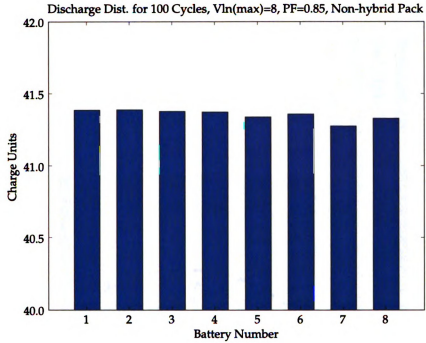


Figure 2.12. Discharge Dist., 8 Batteries, PF=0.85, Non-hybrid Pack.

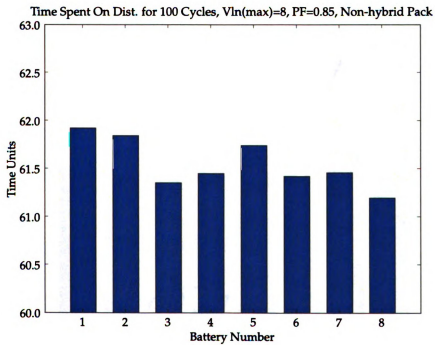


Figure 2.13. Time-Spent-On Dist., 8 Batteries, PF=0.85, Non-hybrid Pack.

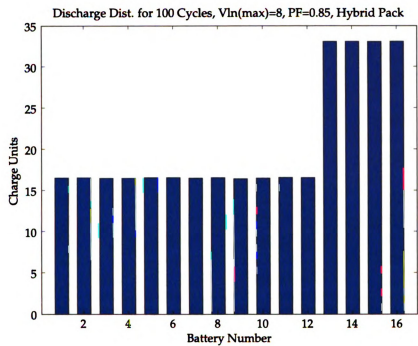


Figure 2.14. Discharge Dist., 16 Batteries, $PF=0.85$, Hybrid Pack.

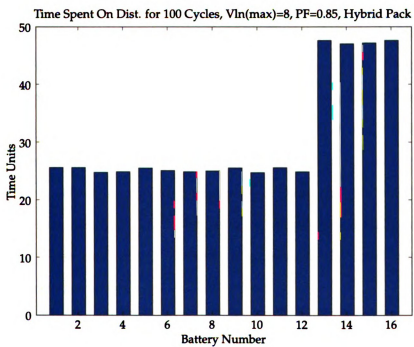


Figure 2.15. Time-Spent-On Dist., 16 Batteries, $PF=0.85$, Hybrid Pack.

Total	PbA	Lilon	Cycles	% Disch. Span	% Time Span
8	8	0	100	0.274	1.188
8	6	2	100	38.715	12.906
10	10	0	100	0.551	1.234
10	8	2	100	0.587	19.319
16	16	0	100	1.008	2.054
16	12	4	100	0.886	9.038
8	8	0	3000	0.009	0.039
8	6	2	3000	38.599	12.527
10	10	0	3000	0.018	0.041
10	8	2	3000	0.020	18.023
16	16	0	3000	0.033	0.061
16	12	4	3000	0.027	7.487
16	12	4	10000	0.010	7.367
16	12	4	100000	0.001	7.115

Table 2.1. Battery Selection Algorithm Simulation Results.

Table 2.1 gives the simulation results for the discharge and time-spent-on mismatch between batteries. The mismatch is calculated by dividing the maximum charge/time by the minimum, and subtracting 100%. Clearly this algorithm is totally satisfactory for any pack type, giving an incredibly low discharge mismatch of 0.001% for the 12-PbA/4-Lilon pack over the span of less than an hour of driving. If the batteries were completely matched, the vehicle's range would increase by a mere two feet over the tiny mismatch. Even if the batteries begin badly imbalanced, the algorithm can correct this; see Figures 2.16 and 2.17. It has a discharge mismatch of only 0.248% even though only 100 cycles passed. Note that the 8/6/2 lines in the table above represent the fact that if the required amplitude is eight batteries, then there must be at least that many batteries of approximately equal capacity for the algorithm to function properly.

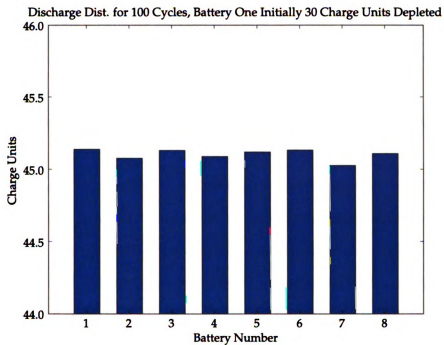


Figure 2.16. Discharge Dist., Battery One Initially 30 Charge Units Depleted.

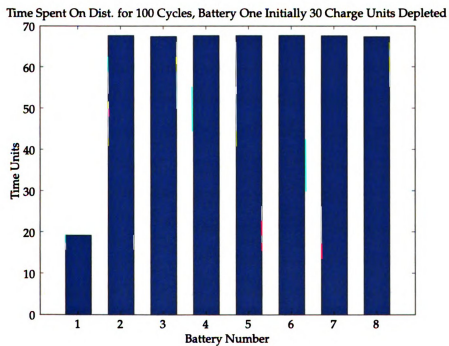


Figure 2.17. Time-Spent-On Dist., Battery One Initially 30 Charge Units Depleted.

This algorithm is easy to implement in the DSC. Its most intensive computation is the minimum- and maximum-searching, which will only be across 40 batteries per phase maximum. The implementation is briefly discussed in Section 4.4.

As shown, this algorithm works with no third-harmonic injection. Positive and negative injection capability could be added, but only positive injection would provide a benefit since it can increase the line-to-line motor voltage and hence motor speed. Negative injection is not needed to smooth the charge distribution, but it may provide secondary empirical benefits. This is yet to be determined and neither injection is supported in the second version of the DSC's code, to be left for further investigation. Also, a function could be implemented to bypass low-power, high-energy cells such as lithium-ion during periods of heavy acceleration, but this is also left as an expansion upon this work.

At low motor speeds, PWM will need to be used on a per-module basis to control current. PWM is implemented in a loop in each slave module; the master sends a module token with an argument of pulse width and polarity to a slave and the slave carries out eight PWM cycles in $364\mu\text{s}$ (11,650 clock pulses in the design-two master). The algorithm must handle PWM somewhat differently since the energy used in a PWM cycle depends on the pulse width, and since a PWM command is automatically turned off after eight periods. PWM also needs to be combined with normal module operation, and while one slave module cannot be used for PWM twice in a row because of communication overlap, only one module in each string will be used for this purpose at a time. A 15-battery system — the minimum limit because of 120VAC recharging capability — will have at most 12 modules on at any given time while performing PWM. The simplest solution for this algorithm accessory is to choose the least discharged available battery — one that had not been used for the previous PWM. This will not interrupt the algorithm's normal regulation and the discharge distribution will remain as it is in Table 2.1.

Ex. 1

Ex. 2

Ex. 3

Ex. 4

Ex. 5

Ex. 6

Ex. 7

Ex. 8

Ex. 9

Ex. 10

Ex. 11

Ex. 12

Ex. 13

Ex. 14

Ex. 15

Ex. 16

2.4 Battery Charging

The source of most, if not all, power for vehicle recharging will come from 120/240VAC single-phase grid outlets. During charging, the three module strings are connected in series and are placed across the outlet terminals along with a small series inductor to smooth the battery voltage transitions. The goal is to create a seemingly resistive load so that the outlet's load current waveform will exactly resemble its voltage in shape and phase. The voltage across the inductor, placed across two identical 60Hz AC voltage sources shifted in phase by 2θ can be represented as follows:

$$V_L = 170V \cdot [\cos(377t + 2\theta) - \cos(377t)] = -340V \cdot \sin(377t + \theta) \cdot \sin(\theta). \quad (2.8)$$

The inductor 60Hz current is then:

$$I_L = -\frac{340V \cdot \sin(\theta)}{L} \int \sin(377t + \theta) dt = \frac{340V \cdot \sin(\theta)}{377L} \cos(377t + \theta) \quad (2.9)$$

$$I_{L(rms)} = \frac{340V \cdot \sin(\theta)}{\sqrt{2} \cdot 377L} \text{ at 60Hz.} \quad (2.10)$$

The worst-case inductor ripple voltage can be approximated by a sine wave with peaks +/-10V and a frequency of half the average switching frequency of 15 batteries per quarter-cycle, or 1.8kHz. The ripple current that results is:

$$I_{Lr} = \frac{10V}{L} \int \sin(11310t) dt = -\frac{1V}{1131L} \cos(11310t) \quad (2.11)$$

$$I_{Lr(rms)} = \frac{1V}{\sqrt{2} \cdot 1131L} \text{ at 1.8kHz} \quad (2.12)$$

$$\frac{I_{Lr(rms)}}{I_{L(rms)}} \simeq \frac{0.001}{\sin(\theta)} \simeq 0.028 \text{ for } 2\theta = 4^\circ. \quad (2.13)$$

The power factor at the fundamental current angle θ , disregarding ripple, is:

$$PF = \cos(\theta) \simeq 0.99756 \text{ for } 2\theta = 4^\circ. \quad (2.14)$$

Both results are more than satisfactory for a high total power factor, which includes a total harmonic distortion (THD) element as well.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

Battery selection during charging will operate similarly to motoring, except that the minimum and maximum are reversed. Less-charged batteries will be turned on first and left on for the longest. The voltage phase angle will be modified depending on how much current is commanded; the battery voltage will be lagging the outlet voltage by about 4° during normal operation.

Absorbed glass mat (AGM) batteries, like all lead-acid batteries, require a careful charging algorithm to minimize hydrogen gassing which reduces their lifetime. Typically, a balance is struck between charging time and gassing, and the charging voltage is simply capped at about 2.4V/cell. Though this is often adjusted based on the cell temperature, it does not provide an interactive means of reducing gassing and may also unnecessarily increase charging time. Lead-acid cells have a charge acceptance that varies substantially with SOC. Practically, the cell acts as a varying current source. When the cell is discharged, it can accept a great amount of current, perhaps even surpassing 2.4V/cell when resistance is taken into account. When the cell reaches 70–80% SOC, the current source reduces in magnitude and increasing the voltage will not increase the current much, but will make the cell gas more.

In this inverter, the charging algorithm attempts to detect the current at which the battery impedance increases substantially and sets this as the current command. At 90% SOC, this current may be as low as 500mA for a 22Ah battery, but at 20% SOC, could be as high as 40A. No user input is needed to determine this charge acceptance current, besides the nominal battery capacities in the master's data EEPROM.

The intrinsic battery current pulsation due to the design of the multilevel inverter has implications for battery capacity, lifetime, and charge acceptance. A literature review was done for the effect of superimposed AC currents in lead-acid batteries. The Papers' results varied [10,11,12]. Some indicated that some frequencies of alternating current decreased capacity. Other results, including a test done for this work, showed moderate increases in capacity (10% or less). Techniques were developed in Wilkinson

and Covic to evaluate the best method of pulsing to increase charge acceptance [10]. Tests showed that an 800ms period with a 25% duty cycle vastly decreased the necessary charging time, allowing an 80% charge in only two hours. This is easily possible with the multilevel inverter and requires no extra hardware. A typical 60-battery system could separate the batteries into groups of 15 and charge each group for 200ms from 120VAC. This requires no extra hardware and the necessary software modifications are minimal, and adjustable at that.

In the second slave hardware design, DC outputs are supported. This would theoretically allow charging from a DC source such as a solar panel collection, an anticipated future source of EV recharge energy. The slave modules would still be in series and could be turned on and off depending on how much charge was still depleted in their associated batteries. This method is not covered nor attempted in this work, but the master module permits future code modification to allow this technique if it is ever needed. AC pulse charging, DC charging, and other optimizations are left as future improvements to this thesis.

2.5 Motor Control

AC motors are not as simple to control as DC motors because there are two variables to regulate instead of one: amplitude and frequency. This work does not take the approach of field-oriented control (FOC), which requires many motor parameters and a shaft speed sensor. There is no standard for shaft speed sensing; the EV₁ used a custom 128-line sensor built-in to the transmission assembly. Speed sensors are often costly and difficult to add, and most AC motors have no published parameters. Speed estimation is a difficult task, and V/Hz control is not sufficient by itself because it requires a speed command which is unsuitable for automobiles.

For the code used in this thesis, I_d and I_q are regulated by means of voltage and frequency, respectively. I_d is kept constant and the vehicle's throttle (and/or brake) determines I_q . In the master module, Clarke and Park transforms decompose three phase currents and an output voltage angle into I_d and I_q [13]. These currents are filtered and fed to the proportional controllers. The output voltage is then adjusted depending on the result of a comparison of I_d to I_d^* . The latter value is a reference, about 12A for a 230V, 15HP motor. The frequency also may be adjusted depending on the throttle value. If I_q is too high, there is too much slip, so the frequency is reduced; the converse is also true.

CHAPTER 3

Slave Module Development

3.1 Hardware Design One

3.1.1 Power Section

The most critical component selection of the slave module design is the power MOSFET switch. The IRF1324S-7P from International Rectifier is chosen for this design because of its low 24V V_{ds} rating, giving it a low $R_{ds(on)}$ of down to 0.8m Ω . At least four of these are needed per battery module because of the full bridge topology, so both positive, negative, and zero output voltages can be produced. This MOSFET comes in a seven-pin D²Pak, with five source pins to help reduce the resistance to such a low level. These extra pins also help moderately with heat dissipation. It is rated for 429A continuously at a case temperature of 25°C and 1640A maximum, one of the most current-dense MOSFETs on the market.

The conduction loss for the module operating at the maximum RMS phase current of 150A is continuously 45W assuming a 1m Ω on-resistance from temperature rise. This can be spread fairly evenly across the four MOSFETs with certain techniques. Though 45W is high for the module's size, it is encapsulated in thermal epoxy and such a current will be rare. Note that a 75A current will produce one-quarter this loss. Neither the motor nor the batteries will be able to stand 150A for long because of resistive heating. The total module switching loss for one transition cycle can be

approximated as:

$$E_{loss} = V_{bat}I_{phase}(t_r + t_f)/2 \text{ joules.} \quad (3.1)$$

For each transition, mainly one MOSFET will be dissipating energy because of the inductive load's constant current, and the average voltage across the switch is half the supply. To reduce dV/dt , a slower transition speed of about $1\mu s$ will be used. For a maximum phase current of 150A and a maximum battery voltage of 15V, only about 4.5mJ of energy is dissipated in one cycle. At 250Hz (half the maximum transition frequency of 500Hz) or a 125Hz output frequency, this represents just over 1.1W of loss total, or an average of 281mW per device. The switching loss is therefore almost negligible in this application. More typical values of loss — at half the motor's rated power at its rated speed — are closer to 25mW per module.

An *avalanche* rating is important for this device because there will be no capacitor across the full bridge. Any time battery power is removed from the phase string, the battery voltage at the MOSFETs will spike up because of stray inductance. The MOSFETs must be rated to absorb this spike. The energy stored in a stray inductor can be calculated with the following standard equation:

$$E = LI^2/2 \text{ joules.} \quad (3.2)$$

An inductance of 250nH, which includes wiring from the battery to the module and a fuse, could be considered worst-case. With this value and the maximum phase current of 150A, 2.8mJ is stored. This is far below the 230mJ single-pulse rating of the IRF1324S-7P, and considerably within the power dissipation limits. If the module were switching off at the maximum frequency of 250Hz, this 2.8mJ every 4ms would represent only 700mW of loss per module. Under typical driving conditions, this value will decrease similarly to the switching loss by a factor of over 40.

Clearly, the conduction losses will dominate the module's temperature and the inverter's efficiency. At full load with 120 modules and 200 feet of 4AWG wire, a

worst-case scenario, total losses are $(46\text{W} * 120) + (150\text{A}^2 * 0.2485\Omega/5) = 6.64\text{kW}$. This is high, but it should be noted that the total power output to the motor is approximately $40 * 11\text{V} * \sqrt{2} * 150\text{A} * \sqrt{3} = 161.6\text{kW}$, giving an inverter/wiring efficiency of nearly 96%. Again, typical efficiencies will be over 99% due to the greatly reduced MOSFET conduction loss at cruising power (about 1W per module).

Some type of gate drive IC is needed for the four MOSFETs because of the 12V gate bias required for both the high and low-side switches. The IRS2004, also from International Rectifier, is the ideal theoretical solution [14]. Each IC can drive two MOSFETs in a half-bridge configuration, and the part draws little quiescent current. The maximum output currents for source and sink are 130mA and 270mA respectively (worst-case).

These parts each require a *bootstrap* capacitor and diode to help provide the floating gate drive supply for the high-side MOSFETs, up to about 25V with reference to the battery negative terminal. If the high-side MOSFETs were P-channel, this would not be necessary; however, considerably fewer models of P-channel devices are produced because of the limited hole mobility and thus worse performance. In this gate drive IC's operation, a low voltage on the high-side source (also the low-side drain) charges the capacitor through the diode. Therefore, the high-side MOSFET cannot be turned on continuously. Beginning with a charged capacitor, and neglecting the charge taken to turn on the switch since it is relatively small, the decay time of the capacitor's voltage can be estimated from its leakage current as follows:

$$I_{leak} = C_{boot} \frac{dV_{BS}}{dt} \text{ amps.} \quad (3.3)$$

Therefore:

$$dt = C_{boot} \frac{dV_{BS}}{I_{leak}} \text{ seconds.} \quad (3.4)$$

Knowing the nominal leakage current is $30\mu\text{A}$ [14], allowing for a 2V V_{BS} drop, and choosing a $2.2\mu\text{F}$ capacitor, dt can be found to be 147ms. This is enough time for

the purposes here, though a longer time would be desirable at the expense of module cost. A lower capacitance value could also be used; however, $2.2\mu\text{F}$ allows non-PWM output frequencies down to about 3Hz without any algorithm complication. A multilayer ceramic capacitor (MLCC) was considered and tested for this part; unfortunately, there were many gate drive IC failures due to ceramic's DC bias effect. The chosen $2.2\mu\text{F}$ had only $0.4\mu\text{F}$ at 12.5V, causing the bootstrap supply to drop many volts and create too high of a dV_{BS}/dt when the V_S line returned to zero. The capacitor's recharging inrush current may have also contributed to the failures. Instead, a tantalum capacitor turned out to be the best choice for this $2.2\mu\text{F}$ part, having a medium internal series resistance of around 3.5Ω at 100kHz. These V_{BS} decay calculations assume a negligible Schottky diode leakage. The generic BAS70 has a very small leakage current compared to $30\mu\text{A}$, so it can be neglected [15].

3.1.2 Microcontroller

Cost is the most important factor in selecting microcontrollers for the slaves because of the sheer number (typically 45–60) that will be in a vehicle. The Atmel AVR series offers very competitive pricing for the speed and feature set, down to \$111 per hundred 20MHz ICs. The ATTiny24 offers an adjustable internal clock, a 10-bit A/D converter, nearly one instruction per cycle, data EEPROM, and self-programmable Flash memory [16]. A serial port peripheral is not important to have, since it easily can be implemented in software. The ATTiny24 can clock itself up to about 16MHz without an external crystal; 11.25MHz will be used to coincide with the 937.5kBd serial transmissions, giving twelve cycles. Up to 1000 instructions can be stored in the processor's non-volatile memory and it has enough pins for all the necessary functions. These include two main gate drive outputs (both with LEDs), a gate driver enable line, two DC/DC converter gate drive outputs, voltage and temperature sense lines, transmit and receive lines, and an input button (actually on the reset pin).

The ATTiny24 supports up to 5V operation. This supply voltage will be generated from the battery via an LP2950 regulator in a TO-92 package [17]. 3.3V could also have been used, but this would have been marginal for the 11.25MHz frequency according to Figure 20-2 in [16].

Isolation between slave modules' battery negative terminals is required in this inverter because of the multilevel topology. Since the IRS2004 gate drivers do not contain any isolation, it will have to be done on the communication bus side. Optocouplers are the simplest solution and are at worst moderately expensive, but still considerably less than the MOSFETs. A bidirectional communication scheme that requires only two wires is possible if both polarities are used. Reducing the number of wires is important if up to 120 modules are to be connected in parallel on the logic side. The master module can drive LEDs contained inside optocouplers from the two-wire bus, turning on the output transistor of each and signaling bits to the ATTiny24. This will be done at the rate of 937.5kBd because of the master's clock rate. An optocoupler with an internal photodiode sourcing current to a transistor's base is required in order to achieve this speed; a phototransistor was found to not switch fast enough. This circuit requires a DC voltage supply to operate, convenient on the battery side but not on the communication bus side since it would require an extra wire. The photodiode optocoupler (6N136S) is more expensive than a slower phototransistor optocoupler (CNY17F-4S), about \$0.65 vs. \$0.19. Since the slaves need to communicate little data back to the master, the CNY17F-4S can be used for the other data direction. Its output transistor will need to be placed in antiparallel with the 6N136S's LED so that it can work against a lower impedance than the master's driver IC. In order to allow the slave modules to communicate back, the master pulls the bus to -4V through about 75 Ω . The 6N136S's LEDs ignore the negative bias and allow the individual slave module's CNY17F-4S to operate against the 75 Ω load. The master reads these pulses on the bus and translates them to bytes

at approximately 115.2kBd, a reliable sending rate for the slower optocouplers.

Voltage sensing is very desirable for the slave modules, so each battery can be monitored and its usage adjusted appropriately in comparison to the rest. This can be done with a simple voltage divider consisting of two resistors. Values of 68k Ω and 33k Ω give a 0-5V A/D range that corresponds to a 0-15.3V battery voltage, plenty for lead-acid. Tolerances of 1% are used to increase measurement accuracy. The ATTiny24 implements a calibration byte in EEPROM to further improve readings.

A temperature reading is also desirable for the design, but is not as important as the voltage. The temperature should mainly depend on weather conditions, measurable from within the master enclosure. However, batteries with shorted cells or other problems may get warm when charged. The ATTiny24 does include an internal temperature sensor, but it does not accurately represent an overheated battery since the battery would get warm mainly on the sides, not the top (where the module is mounted). This necessitates an external thermistor mounted on the side of the battery. The same 1%, 68k Ω resistor as above can be used as the top part of a voltage divider across the 5V supply, the bottom of which is a 68k Ω thermistor. This setup gives a large range of temperature readings and both it and the internal sensor can be calibrated and compared to find an overheating battery very early, before it gasses excessively or even explodes.

3.1.3 DC/DC Converter

Due to the isolation of batteries, generating a 12V vehicle accessory supply (for headlights, *etc.*) is difficult. It requires each module to have an individual high-frequency transformer and two small MOSFETs driving it, here in a push-pull configuration for ease of gate drive. Each module should be able to supply about 2A of output current at 12V; 45 of these paralleled outputs would equal 90A total, comparable to an alternator in an internal combustion engine vehicle. The GM EV₁

requires a high-voltage supply, nominally the same as the twenty-six 12V batteries in series, 312V. Twenty-four 12V outputs in series (instead of parallel for conversion electric vehicles) would make 288V, perfectly acceptable for this purpose.

The MOSFET chosen for the push-pull power supply is the Toshiba 2SK2231, in a D-Pak package. It is available for \$0.35 each in a quantity of 100 or more and has a $0.12\Omega R_{ds(on)}$. Its gate driver is chosen to be the TC1427 from Microchip, only \$0.88 each per 100 purchased. It is one of the least expensive drivers available but still features two 1.2A totem-pole outputs, plenty sufficient for the 2SK2231s. The gate driver, MOSFETs, battery, and microcontroller all share the same ground (the negative battery terminal). The Atmel's OC1A and OC1B, with a dead time to prevent cross conduction, can be used to provide signals for the TC1427.

The transformer will consist of two center-tapped windings. The primary will go across the 2SK2231 drains with its center tap at battery 12V, and the secondary will go through two rectifiers to output positive with its center tap at output negative. The converter's rectifiers can be Schottky barrier diodes because of the low required reverse voltage, about 40V. Two 1N5822s will be used; they cost \$0.25 each in a quantity of 100. It was planned to use a small toroid for the DC/DC converter, but this was never implemented due to the unreliability of the first hardware design. See Section 3.3.3 for a description of the functioning redesign of the DC/DC converter.

3.1.4 Printed Circuit Board

The first hardware design does not call for a four-layer PCB; a two-layer with added copper thickness is acceptable. The extra copper aids in current conduction and heat spreading/dissipation. The board was designed in Cadence OrCAD Layout and produced by Silver Circuits in Malaysia. The design one slave's schematic can be seen in Figure 3.1.

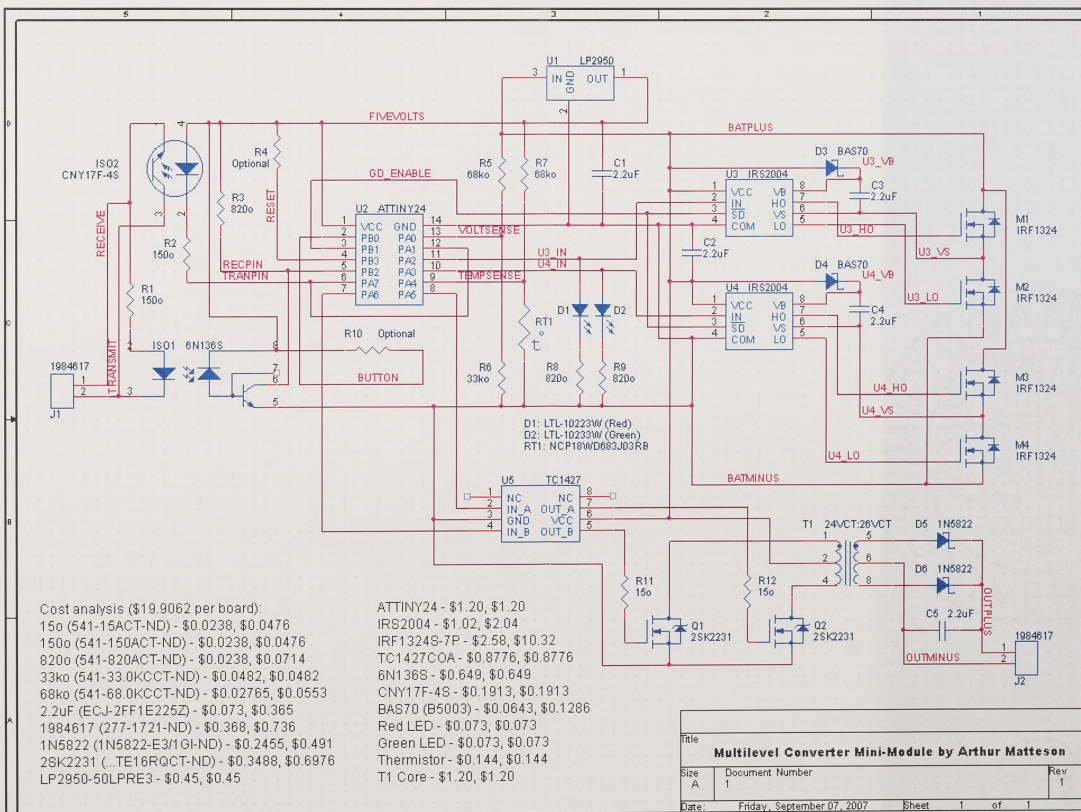


Figure 3.1. Slave Module Design One Schematic.

Sap

BEN

LST

Dns

Dns

Dns

S.H.

Egy

Ldt

Dns

I.

S.H.

Dns

Dns

Dns

Dns

Dns

Dns

Dns

Dns

Dns

Dns

Dns

3.2 Redesign Considerations

Six prototype design-one slave modules were constructed with two IRS2004s and four IRF1324Ss with voltage isolation between slave modules' microprocessors as described in Section 3.1.2. The gate drivers were unreliable and often exploded for seemingly no reason, even after changing to a tantalum bootstrap capacitor. Because of the high currents and quick voltage transitions involved, a decoupling or isolation was needed between the microprocessor and power grounds. This is not easily achievable with the IRS2004s since they share a logic and power ground. The Solar Car Team's GM EV₁ moved about five feet under the power of the six slave modules, but without a load, the modules would randomly fail even after just being repaired and the design was eventually abandoned. Still, many of its aspects remained in the second version.

The reason for failure was never fully investigated, but the first hardware design also had the undesirable limitation of requiring the master's DSC to change its serial port baud rate on the fly, since the slaves could not send at 937.5kBd. This would have been easy without buffering, but buffering was required in the DSC because of the manual implementation of USB. Without a token buffer, USB transactions would interrupt the flow of tokens to the slave modules. The token buffer caused issues with control, and the bit-banging USB implementation was slow and error-prone despite implementing CRC checking. The design-one slave modules were tedious to connect and disconnect, since they used miniature terminal blocks with screws. This problem was alleviated in the second slave hardware version by the use of 0.1in.-spacing headers and connectors paired with ribbon cable. The master redesign followed the slave redesign by about a month. Master designs one and two are covered in Chapter 4, and conclusions are discussed in Chapter 6.

3.3 Hardware Design Two

3.3.1 Power Section

The second slave module PCB was designed to support eight IRF1324Ss, configured in pairs, to reduce conduction loss in high-power installations. For the same phase current, this ideally reduces the conduction loss to only one-fourth of the original value per part or one-half per module. The avalanche losses will be similar for a given phase current, but since there are more MOSFETs, the module can support more avalanche current. Though the manufacturer recommends against sharing avalanche spikes between paralleled MOSFETs, it is acceptable in this design because of the similar temperature, close proximity, and identical date code of the parts (this must be ensured during PCB population).

It was decided to move the isolation barrier from between the communication bus and ATTiny24 to between the ATTiny24 and power MOSFETs, in order to eliminate feedback and spikes from the power section that had been causing problems in the first design. This would allow the μ Cs to both transmit and receive at 937.5kBd or 1MBd (depending on the master hardware version), and would not require the master's DSC to change its baud rate except when reprogramming the Atmels.

An optoisolator with a built-in gate driver was found for the second hardware version, the HCPL-314J from Avago Technologies. The ATTiny24 drives four internal LEDs in the HCPL-314Js to signal their output stages to be either high (12V) or low (0V). Each half of these parts requires an isolated voltage supply to drive the associated MOSFET gate, totaling four isolated supplies in all. These can be taken from the DC/DC converter. Small MLCC capacitors can be used to bypass the gate drivers, along with BAS70 diodes — the same used to create a bootstrapped supply in the first design — to half-wave rectify the transformer windings' alternating voltages.

3.3.2 Microcontroller

Since the Atmel μ Cs are isolated from the batteries, they can share a 5V supply with the master's DSC. Separate transmit and receive lines are shared among the slaves, making four signals total, spread across a ten-pin connector for redundancy and hence connection reliability (5V and ground are given three pins, and transmit and receive are given two). In order to allow each ATTiny24 to reply without requiring any to change its own port direction (input vs. output), a small Schottky barrier diode is placed with its anode to the bus and cathode to the μ C, and a pullup resistor is placed inside the master module.

The HCPL-314Js associated with the IRF1324Ss require a total of four signals from the μ C. A dead time needs to be manually coded in software so that the upper and lower IRF1324S don't both conduct at the same time while transitioning. Red and green LEDs are placed in parallel (through resistors) with the high-side gate drive LEDs in the optocouplers, so the module output voltage can always be known.

Battery voltage sensing is done through a CNY17F-4S optocoupler operated in an analog mode. A 4.7V Zener diode and $3.3\text{k}\Omega$ resistor form an ideal input circuit, and another $3.3\text{k}\Omega$ pull-down resistor on the output side is sourced current through the phototransistor, providing an A/D input for the Atmel. The external thermistor configuration remains the same as in the first design, except for the use of $22\text{k}\Omega$ each instead of $68\text{k}\Omega$.

3.3.3 DC/DC Converter

The same 2SK2231 MOSFETs are used in the second design as in the first, since they are still ideal for this purpose. However, due to them now being isolated from the logic, another HCPL-314J is used to drive their gates, with its supply directly taken from the battery. This gate supply does not need to be isolated since the DC/DC converter works with fairly small currents. The same two ATTiny24 pins are used to

provide gate drive signals, OC1A and OC1B.

The winding of a toroid was found to be quite difficult for producing even a small number of these modules. Instead, an *EFD* core is chosen because it can use a bobbin. It consists of two halves that somewhat resemble the letter 'E.' Surface mount bobbins from Transformer Bobbin Industrial in China were sampled, and EFD-15 (flat, 15mm wide) cores from Ferroxcube were purchased through Elna Magnetics. These bobbins each support six isolated windings. In order to achieve the 25-30W needed for the DC/DC converter's output, two transformers are used in the second slave design. Out of the twelve windings, four are used for MOSFET gate drive supplies and the other eight are split between the input and output and are center-tapped by connecting pairs of windings in series. 26AWG wire is used to wind the transformers, sufficient because it carries only 1A for half the time when the converter is outputting 2A to the vehicle's accessories. This pair of transformers, operating at about 250kHz, draws just below 60mA of current under no load. Reducing that drain may be possible, but it only amounts to less than one watt of loss.

The HCPL-314J has a quiescent load of about 3mA. This would create a nearly 10% monthly depletion in each battery's charge, quite undesirable if the vehicle were to be stored for any length of time since the lead-acid electrolyte freezes much easier when a battery is discharged. Since the optoisolator only draws about 15mA maximum, a small MOSFET — the SOT-23 NTR4101 from On Semiconductor (\$0.26) — can be inserted in the power supply loop, its gate controlled by another optoisolator, the LTV-816S (only \$0.15). With a clean PCB, this reduces the quiescent current down to less than 1 μ A.

3.3.4 Printed Circuit Board

Due to the number of connections in the second design, a four-layer board is required if it is to fit on top of the UB12220 battery — an important space-saving criterion. Both the old and new layouts can be seen in Appendix B. Special attention was paid to the high-current traces. As much copper as possible was laid down to help decrease the trace resistance and aid the spreading of heat away from the main MOSFETs. Figure 3.2 is the redesign's schematic.

3.4 Software

3.4.1 Main Loop

The only task of the μ Cs' main loop is to perform A/D conversions. This includes one battery voltage, one external temperature sensor, and one internal temperature sensor. These are interpreted through lookup tables and calibrated with data EEPROM values. The results can be read through the serial interrupt.

3.4.2 Serial Interrupt

Both the design-one and design-two serial interrupts are triggered by a low level on the Atmel's INT0 pin. Following a low start bit of approximate length 1μ s, nine data bits are read in, each the same length as the start bit. For a clock frequency of 12MHz, 12 cycles pass in 1μ s. This is ample time to process the transmitted value. It is interpreted as shown in Table 3.1. The first column, *Command*, is split into seven bits and two bits (see Figure 3.3). This table is for the second hardware design, but differs very little from the first hardware design command set.

Many of the commands are intuitive, but those such as "execute a roll call" and "calibrate clock frequency" have somewhat more complexity. The purpose of the roll call command is to find which modules are present in an orderly fashion and to also assign a new address to a previously unassigned module (address of 255). After a roll call command of 126 is sent out, values from 1–120 are sent by the master to the bus to find an attached module to which that address belongs. If a module is called, it echoes its address. At the end, a value of 255 is sent out; if a module has this invalid temporary address (default for an erased EEPROM byte), it assigns itself the first available spot and writes the value to EEPROM location \$01.

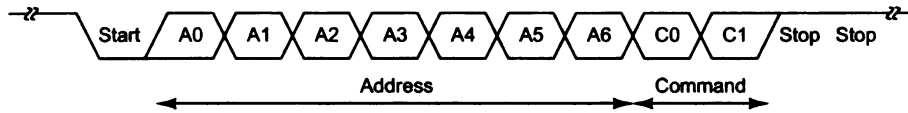


Figure 3.3. Communication Bus Byte Format.

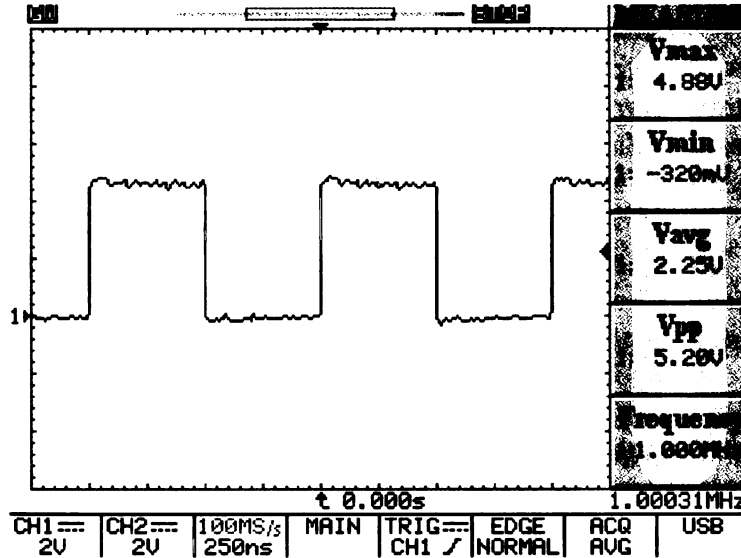


Figure 3.4. Slave Module Automatic Frequency Calibration.

The calibrate clock frequency command is used to ensure successful data reception by the slave modules. The design-two master contains an 8MHz precision quartz crystal oscillator, but the slaves have only internal clocks whose frequencies vary with temperature and possibly lifetime. It is therefore advisable to occasionally correct the slave oscillator calibration bytes (EEPROM location \$00) by way of the master. The value 127 is chosen intentionally as the command because only the start bit needs to be received properly. This greatly increases the chance an uncalibrated module will receive the transmission correctly. After sending two 127s, the master sends thirty-two 0s at 125kBd, to which the Atmel adjusts itself by means of a width measurement. Test results show that even from as high as 14.3MHz or as low as 9.7MHz, the Atmel could recover. Figure 3.4 shows a divide-by-12 output from a calibrated ATTiny24.

Command	Arguments	Function
0 0X		Enter bootloader (125kBd start/0 bit)
0 1X		Do nothing (dummy command)
1-120 11		Turn off module at given address
1-120 01		Turn on module pos. at given address
1-120 10		Turn on module neg. at given address
1-120 00	0000000 00	Echo next byte XOR module address
1-120 00	1000000 00	Put addr'd module in state "sleeping"
1-120 00	0100000 00	Put addr'd module in state "idle"
1-120 00	1100000 00	Put addr'd module in state "ready"
1-120 00	0010000 00	Put addr'd module in state "running"
1-120 00	1010000 00	Blink addr'd module for three seconds
1-120 00	0110000 00	Read EEPROM (address follows)
1-120 00	1110000 00	Write EEPROM (address, data follow)
1-120 00	0001000 00	Transmit DC battery voltage A/D value
1-120 00	1001000 00	Transmit AC battery voltage A/D value
1-120 00	0101000 00	Transmit local temperature A/D value
1-120 00	1101000 00	Transmit remote temperature A/D value
1-120 00	0011000 00	Erase addr'd bootloader flash memory
1-120 00	1011000 00	Fill addr'd bootloader memory buffer
1-120 00	0111000 00	Write addr'd bootloader flash memory
1-120 00	1111000 00	Verify addr'd bootloader flash memory
1-120 00	DutycP1 00	Eight PWM cycles with duty cycle, pol.
1-120 00	EEkey 1	EEPROM writing key, \$92 XOR mod. addr.
121 !00		Put all modules in state "sleeping"
122 !00		Put all modules in state "idle"
123 !00		Put all modules in state "ready"
124 !00		Put all modules in state "running"
125 !00		Blink all modules for three seconds
126 00	Address X	Execute a roll call
127 XX	1111111 XX	Calibrate clock frequency
Else		Bad command; ignore, exit interrupt

Table 3.1. 1 Megabaud Command Enumeration.

Data/Command	Function
\$AC \$53 Addr	Enter bootloader, module to address as third byte
XX XX \$00	Hang and let the watchdog reset all modules
AdrL AdrH \$01	Fill page buffer at address with data (call after \$88)
AdrL AdrH \$03	Erase Flash page at address
AdrL AdrH \$05	Write page buffer to Flash page at address
XX XX \$11	Clear temporary page buffer for Flash writing
Addr XX \$41	Return value of data EEPROM byte at address
Addr Data \$44	Erase; write given data to EEPROM byte at address
Addr XX \$54	Erase EEPROM byte at address
Addr Data \$64	Write given data to erased EEPROM byte at address
Addr XX \$81	Change currently addressed module to first byte
AdrL AdrH \$82	Indirect jump to address LSB/MSB in bytes one/two
AdrL AdrH \$84	Return value of program Flash byte at address
DatL DatH \$88	Load internal data buffer LSB/MSB in bytes one/two
XX XX \$FF	Exit bootloader normally (no watchdog reset)

Table 3.2. 125 Kilobaud Bootloader Command Enumeration.

3.4.3 Bootloader

The ability to update the μ Cs' code remotely is very important, especially during development. Non-critical updates, such as improving temperature or voltage measurement accuracy, and critical updates, such as fixing bugs, can both be applied through a firmware update. Unfortunately, modifying a certain section of the flash while executing out of it is difficult if not impossible. For this reason a bootloader was developed, placed from \$0370 to \$03FF in the Atmel flash memory. This small, robust code segment has the capability to rewrite the rest of the flash in a matter of four seconds. All bootloader commands are sent at 125kBd to ensure communication reliability. Verification is also done at this speed. The command set is shown in Table 3.2. Every command consists of three bytes sent as a group. Only the addressed module will respond to a programming command. The currently addressed module can be changed by the \$81 command; a value of zero addresses all modules globally. The design-two master has the built-in capability of reprogramming the slaves, given the correct data; direct computer control was found too unreliable.

CHAPTER 4

Master Module Development

4.1 Hardware Design One

The dsPIC30F4011 DSC is the heart of the master module [7]. It provides slave communication signals, throttle and current A/D inputs, contactor signal outputs, USB signals, as well as others. It is available as a 5V-capable, 40-pin DIP or 44-pin TQFP (thin quad flat pack). In this work, the DIP is chosen because it can fit into an inexpensive ZIF (zero insertion force) socket for replacement, instead of needing to be desoldered like the TQFP. Five volts is the only voltage node that supports full-speed operation, or 30MIPS, so the supply bus voltage is an easy decision.

The greatest design challenge for the first hardware version is the communication bus. It needs to be passive at 75Ω , -4V and active at less than 1Ω , +4V; a strong driver IC is required for this. The TC4452 from Microchip (dsPIC manufacturer) is a 12A driver with an allowable supply of up to 18V — double the necessary ratings. A diode's anode can be placed at its output, with the cathode to the communication bus, and a 75Ω resistor can be paralleled with the diode to give the desired impedances and voltages. Because the logic input will be referenced to -4V, a transistor is needed to turn the non-inverting device on. The dsPIC drives the PNP transistor's base through a resistor. The -4V supply is generated from a simple charge-pump, consisting of a UCC27423 gate driver from Texas Instruments, fed from the 7.5MHz oscillator. An

NPN transistor is placed across the 75Ω resistor and senses when a slave module tries to "short out" the bus. Its response is to pull down the DSC's receive pin.

The DSC needs to accept ignition and throttle/regeneration inputs as well as current sense signals. The ignition signals are typically 0V or 12V, so a $10k\Omega$ resistor can be placed in series with them since the current will be considerably below the maximum pin clamping current. The dsPIC will use seven analog inputs: two throttles (one for regeneration), three phase currents, one scaled/isolated charging plug voltage, and one scaled battery voltage. The plug voltage signals the DSC when the user plugs the car in to charge. The battery voltage allows calibration of slave modules, with the help of three DPST (double-pole, single-throw) reed relays that switch the phase voltages' grounds to the DSC's ground and average the scaled phase outputs together. When a single slave module is turned on before or after charging, its voltage is read by the dsPIC, and the slave module's voltage calibration byte is written to correct its measurement error. This at least guarantees relative accuracy between slave modules, even if the dsPIC's analog supply voltage is slightly inaccurate. Six additional SPST (single-pole, single-throw) reed relays drive other relays that make connections between the phase strings, such as connecting their negative ends to drive the motor or putting them in series to be charged from an outlet.

A 74ABT245 bidirectional bus transceiver from Texas Instruments is added to the design to interface to USB. The dsPIC will not directly interface to USB because the logic signals are not compatible; i.e., the dsPIC needs a $0.8 \cdot V_{dd} \simeq 4.0V$ input to trigger as high. A TI CAN transceiver, the 65HVD251D, is also added to the master.

Since the master module deals with nothing over 5A (for the relays), and dissipates little heat over few components, its size can be relatively small. Enclosure model 1554GGY from Hammond Manufacturing was chosen. It can accommodate a circuit board of approximately 3.6x3.1in. Therefore, all of the parts should fit in this space. The schematic of the first master module PCB design is shown in Figure 4.1.

4.2 Hardware Design Two

The main reasons for a master module redesign were the unreliability of the manually-implemented USB and the need for higher-precision A/D inputs. A PIC18F2553 processor is therefore added to the module. It supports full-speed (12MBd) USB, features an internal USB transceiver, and has ten 12-bit A/D inputs [18]. It also features a serial port with a rate up to 2MBd, necessary for communicating with the dsPIC. Because of the 2553's USB clock and the need for inter-processor communication, the dsPIC needs to be overclocked by 6.67% to 32MHz; this is not a reliability issue since it is operating well above the minimum voltage for 30MHz.

Much of the remaining hardware design is kept the same as the first version with a few exceptions. The reed relays are replaced with MOSFETs for higher current capability, since they were found to be unreliable above a 1A switching current. A boost converter (the PTN04050C from Texas Instruments) is added to allow accurate regulation of the analog 5V supply from about 10V when the USB port provides module power. An additional PTN78000W buck converter is introduced to prevent negative V_{DD} spikes when turning on the module bus. This is separate from the original 5V buck converter to allow a current reduction on the 12V master module supply when the vehicle is off. More optoisolators are added to sense the motor phase voltages in addition to the AC input (as before) to check for malfunctioning slave modules or for other unforeseen purposes. Finally, two LEDs are used in the new design: green for power/ignition and red for USB activity. The design-two PCB schematic is given in Figure 4.2.

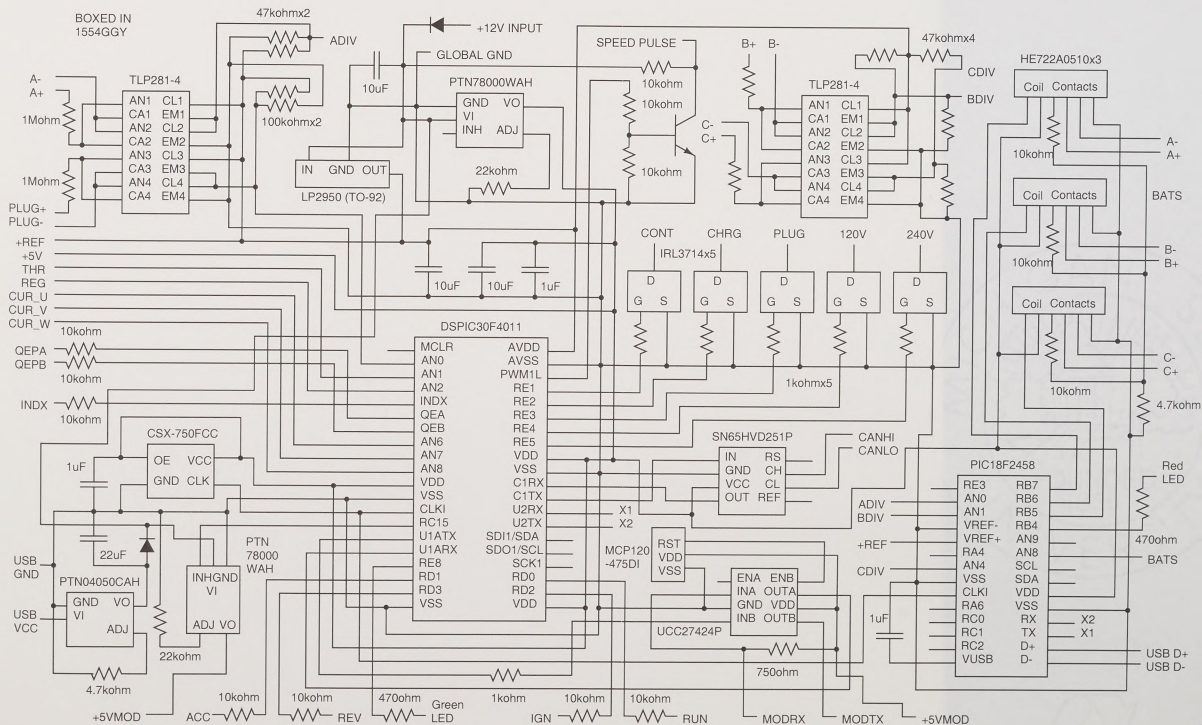


Figure 4.2. Master Module Design Two Schematic.

4.3 Software Design One

4.3.1 Main Loop

The first task in the dsPIC30F4011 code is variable initialization. This includes sampling each of the three phase currents to determine their idle values, in the case of small offsets in the current sensors or the DSC. Sixteen samples are taken of each channel and are averaged and saved. The module token buffer is cleared, USB variables are reset, and all peripherals are set up in this stage. Initialization constitutes almost 250 lines of code.

The functions implemented in the main loop are as follows: check the USB port's status, calculate the minimum and maximum voltage for the current frequency, find the current accumulated angle, buffer module tokens to generate sinusoidal output voltages, measure, transform, and filter the currents, calculate and average the power factor, make voltage and frequency adjustments based on the system variables, and poll the timer value until the $364\mu\text{s}$ loop is over. This is done in a matter of just over 500 lines of assembly language code; only six batteries are supported in this version. The design-one main loop code is excluded from the Appendix as it is essentially the same as the design-two main loop code.

Slope and offset constants (such as a and b in $y=ax+b$) define the minimum and maximum voltages for any given synchronous frequency. These are calculated by the 17x17-bit hardware multiplier. The limits are saved in a 16-bit format, with the low byte corresponding to fractional batteries and the high byte to integral batteries. Fractional batteries, with 32 divisions, are made by PWM; only one module per phase string conducts PWM at once. These minimum and maximum voltages are used later in the main loop.

The elapsed "future time," or buffer time, is calculated and multiplied by the frequency to find an angle increment. The increment is added to the previous angle

and sine values are found for offsets 0, 120, and 240 degrees. A sine lookup table is used with 10-bit input and output precision (0.352° and 0.002 result precision). These three results are multiplied by the amplitude and then added in the form of module tokens to the DSC's FIFO buffer. A simple toggling scheme is used in this prototype code to alternate between energizing batteries #1/4, #2/5, and #3/6; each of these pairs is in series, with the #1/2/3 negative outputs connected to neutral.

Tokens will be sent out by the 'SENDTOKEN' macro, shown at the beginning of Appendix E.1, during the USB interrupt and the main loop. This eight-cycle macro was designed to fit inside the USB interrupt without interrupting its processing. Working register W13, used as the buffer output pointer, automatically rolls around at the end of the buffer with help from hardware modulo addressing. The macro compares the timestamp word in the buffer to the current time. If the timestamp is zero, it assumes the buffer is empty. If the current time has surpassed the timestamp, it sends out the word in the buffer that follows and increments the output pointer. The tokens have a transmission time precision of about $2\mu\text{s}$ due to the spacing of 'SENDTOKEN.'

The three phase currents are next measured and centered using the initially measured idle values. These numbers are not averaged since doing so would create a phase delay with reference to the applied voltage, destroying the I_d and I_q calculations that depend on the voltage/current phase shift. A Clarke transform creates I_α and I_β through simple arithmetic, and a Park transform results in I_d and I_q by using the sine lookup table [13]. The Park transform outputs can be heavily filtered because they should be approximately DC currents. The 32-bit accumulators are used to calculate a 4096-weight infinite impulse response filter. This high precision is needed to prevent hysteresis problems, *i.e.* a current never approaching the steady-state value due to rounding error.

The power factor is calculated by dividing I_q by I_d , and taking the cosine of

the arctangent of the quotient. A 1024-entry lookup table provides this complex conversion. The power factor is not used in this version of the code, but gives the user an indication of what region the motor is operating in. There may be efficiency optimizations possible with this value, but that is left as an extension to this thesis. This value would be used while charging to ensure a high power factor at the outlet; however, a charging algorithm is not implemented in this code version.

Finally, the applied motor voltage and frequency are adjusted based on I_d and the throttle, respectively. A low I_d , or one below the reference I_d^* , will force the control loop to increase the applied voltage. Similarly, a high throttle (I_q^*) will result in an increased synchronous frequency, which will in turn increase I_q as desired. This control loop structure avoids the need for a rotor speed feedback.

4.3.2 USB Interrupt

The DSC's USB interrupt is triggered by a change on either the D+ or D- lines. A sync byte is read in, followed by the packet identification byte and the address/endpoint [19]. A cyclic redundancy check (CRC) calculation is performed on the packet's data as it comes in. For the design-one master's clock frequency of 30MHz, 20 cycles pass for every low-speed USB bit. The actual bit receive routine tests for an edge in a window of about eight cycles. If it sees no edge, it considers the bit a one. If it does see an edge, it considers the bit a zero and resynchronizes its timing. In this way, no timing errors will be accumulated over the course of a large packet. USB specifies a stuffed zero bit for every six one bits; the receive and transmit routines implement this and exclude these stuffed bits from the CRC.

This code implements a nearly complete enumeration sequence [19]. Nearly all functions have been included; even string descriptors are supported. The processor has four software endpoints: EP0 in, EP0 out, EP1 in, and EP1 out. EP0 is used for control transfers and EP1 is used for interrupt transfers, the latter representing all

software communications such as firmware upgrades and data collection. The software package developed for the inverter, named “Monitor” as described in Appendix A.1, has the ability to send individual commands to the modules through this USB interrupt. Multiple reports can be sent to the computer, including many different eight-byte combinations of motor variables such as currents and voltages. DSC RAM can also be read or written by the computer to alter control functionality, such as changing loop gain constants.

The USB interrupt code is included in Appendix E.1.

4.3.3 Bootloader

The design-one master has another copy of the USB interrupt code at DSC flash address 0x006000 to act as a bootloader, when the master firmware needs to be upgraded. The second copy of the code is not an interrupt but actually operates “inside of” the original interrupt and polls the D+ and D- lines in a loop instead. From this bootloader, all 36,864 bytes of memory from 0x000000–0x005FFE can be updated in about three minutes. All 1024 bytes of data EEPROM can be updated and verified from the bootloader. The main loop’s USB interrupt includes similar flash and EEPROM modification code, so the bootloader itself can also be updated. This code mostly ranges from lines 611–749 in Appendix E.1.

The main loop contains a routine to update the slave modules’ Atmel memory upon request. The PC writes the new Atmel program to DSC memory, and after telling the DSC which slaves to update, the DSC performs a communications check with each slave and reprograms its memory block-by-block (32 bytes at a time). Direct PC control was found to be unreliable and unnecessarily slow for this task. This routine can update each slave’s entire flash memory in less than four seconds.

4.4 Software Design Two

4.4.1 PIC18F2553 Code

The PIC18F2553 takes the place of the dsPIC30F4011's USB interrupt with its full-speed USB peripheral. It communicates with the DSC at 2MBd over two serial lines. Each processor maintains a 256-byte area in RAM containing data for both USB transfer directions, since each of the four endpoints is 64 bytes. When an interrupt OUT command is sent over USB, the resultant data is automatically written to the corresponding location in 2553 RAM by the peripheral. This is then forwarded in software to the DSC over one of the 2MBd lines. Similarly, the DSC periodically sends motor variable data over the other serial line which is then stored in the associated buffer in the 2553. The PC may then read it by an interrupt IN command. The 2553 has a red indicator LED to show USB activity.

The 2553 also has the duty of converting four analog values at 12-bit precision. Three of these values are optoisolated phase voltages and the other is a resistor-scaled combination of non-isolated phase voltages. Before and after battery charging, slave modules are turned on in order and calibrated against the standard measurement of the 2553 with the help of three reed relays shown in Figure 4.2 (HE722A, top right). This will give the slave modules better relative accuracy so their respective batteries will not get as mismatched during charging and discharging. The results from these conversions are sent to the DSC occasionally over the serial bus.

This part does not contain a bootloader in this project and cannot update its own flash memory; however, this was deemed unnecessary due to its fairly general functionality. It does have the hardware capability to update itself and this function could potentially be written if a need for it arises.

44.

45.

46.

47.

48.

49.

50.

51.

52.

53.

54.

55.

56.

57.

58.

59.

60.

61.

62.

63.

64.

65.

66.

67.

68.

4.4.2 DSC Main Loop

The design-two main loop greatly resembles the design-one main loop with a couple of exceptions. There is no longer a DSC module token buffer. This is unnecessary here because the main loop will be only lightly interrupted. Due to the absence of a USB interrupt, all flash and EEPROM modification code is in the main loop. The routine to reprogram the slave modules remains in the main loop and is not at all changed. Motor variables are written serially to the 2553 after being recalculated.

4.4.3 DSC Bootloader

A bootloader for the design-two master is located at 0x007C00 in dsPIC30F4011 memory. It begins on line 2577 in Appendix E.3. It is nearly identical to the design-one bootloader except it utilizes both EP0 out and EP1 out for a total of 96 program data bytes and can update the DSC's flash memory in only five seconds as opposed to nearly two minutes for design one. As in the first design, the flash and EEPROM modification code appears twice in the listing, so all code can be updated: once before 0x007C00 and once after.

4.4.4 Serial Port Interrupts

Both the PIC18F2553 and dsPIC30F4011 use serial port interrupts that trigger when bytes are received. The communication method uses nine bits, where the top bit is zero for an address byte or one for a data byte. A current address is maintained in each processor, and each writes to the other's memory nearly transparently. The processor with information to give away will first write an address command, then follow it with one or more data commands. The data commands automatically increment the writing address so that address commands do not need to be constantly sent when large areas of shadow memory are being copied. The DSC bootloader also contains an identical serial port interrupt.

CHAPTER 5

Results

Various results have been recorded in this work. Most are either oscilloscope graphs or **matplotlib** graphs of USB-acquired data from the master module. The oscilloscope used was a GW-Instek GDS-820C.

Figures 5.1, 5.2, and 5.3 are oscilloscope plots of a slave module's operation. The first shows a current waveform with $\pm 130\text{A}$ peaks, demonstrated with an approximately resistive load (halogen light bulbs). The second depicts a slave MOSFET going into avalanche conduction. Since the MOSFETs are rated for 24V, the 23V peak is fairly accurate. The duration of the avalanche pulse is $1.16\mu\text{s}$. The module in question had only four IRF1324Ss populated. The third shows two 800ns dead-times of a module performing PWM. These dead-times are necessary to prevent cross-conduction and energy waste.

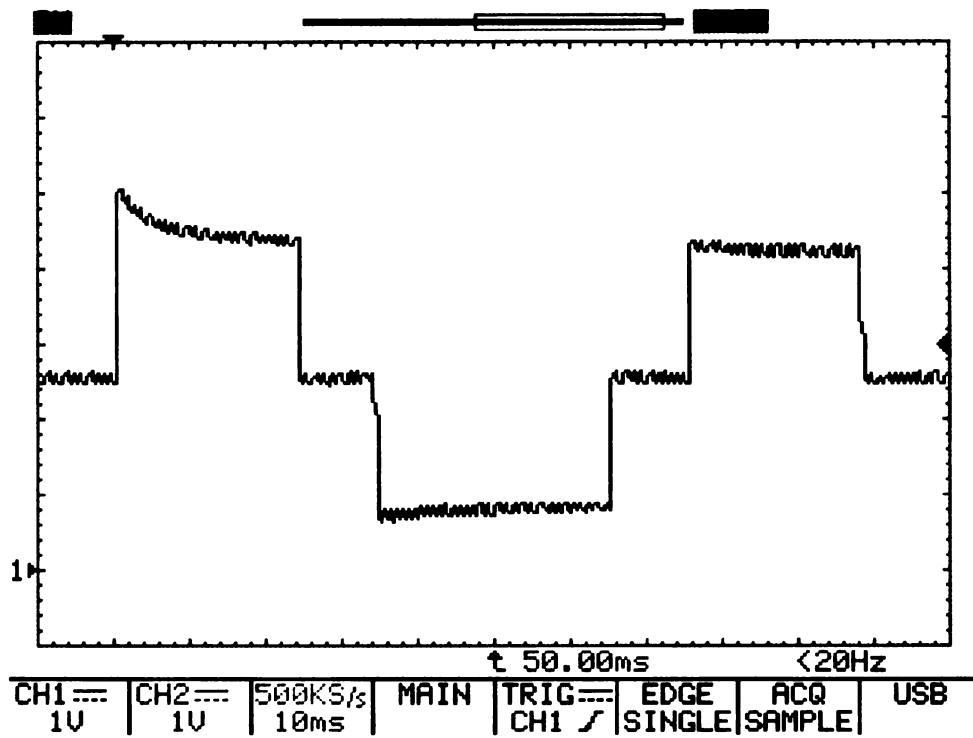


Figure 5.1. Slave Module Output Current Waveform, 130A Peak.

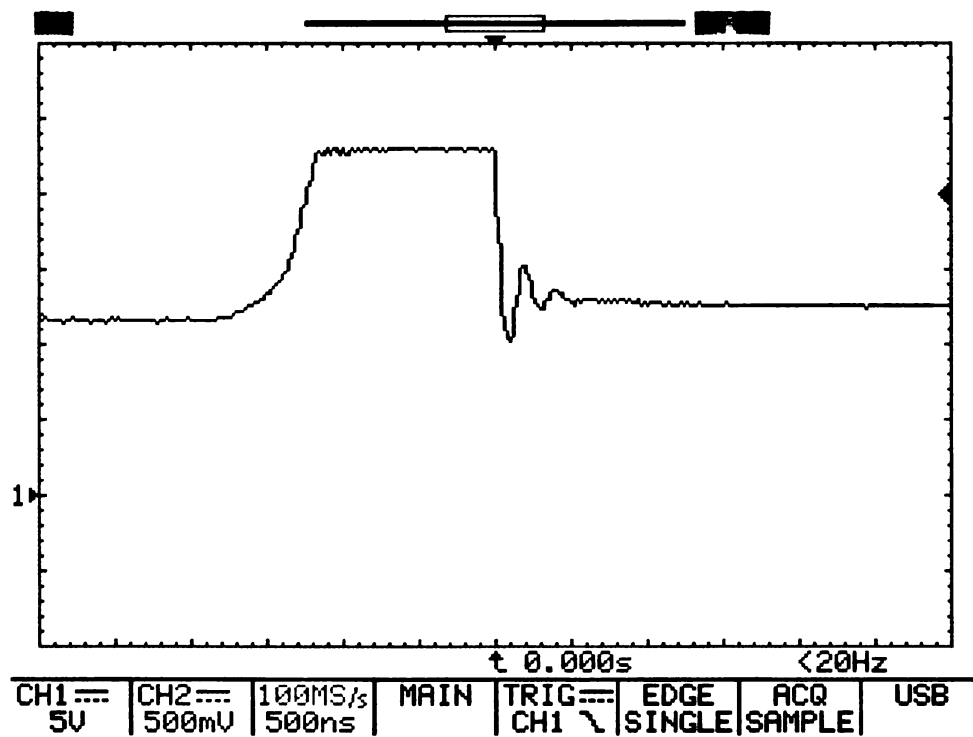


Figure 5.2. Slave Module MOSFET Avalanche Waveform, 23V Peak.

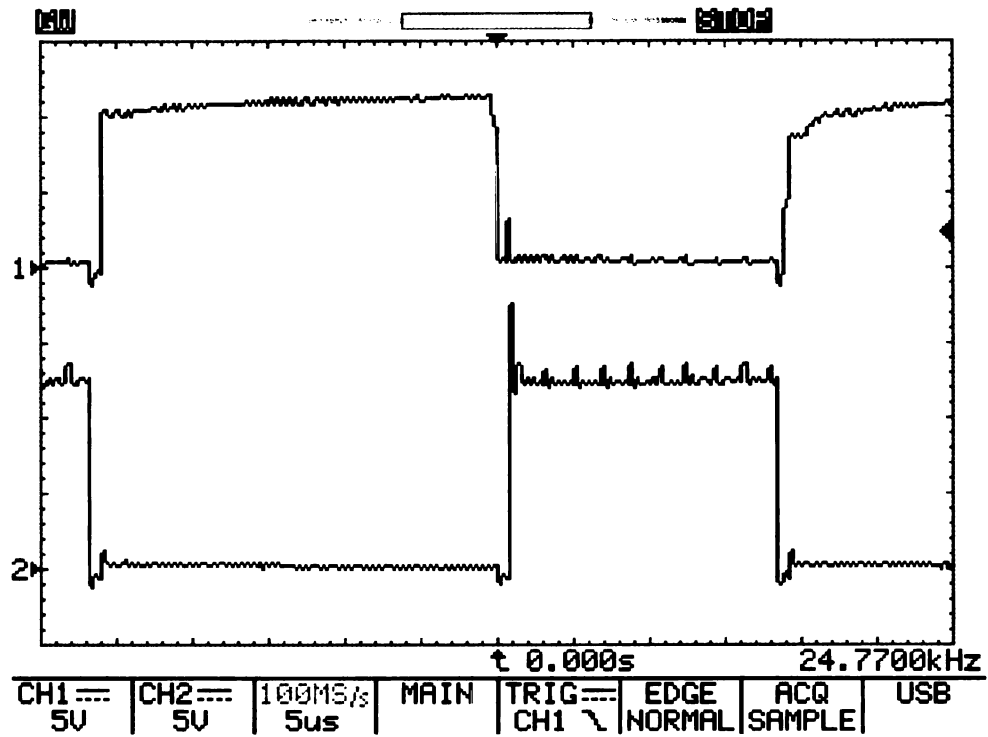


Figure 5.3. Slave Module PWM Illustrating an 800ns Dead Time.

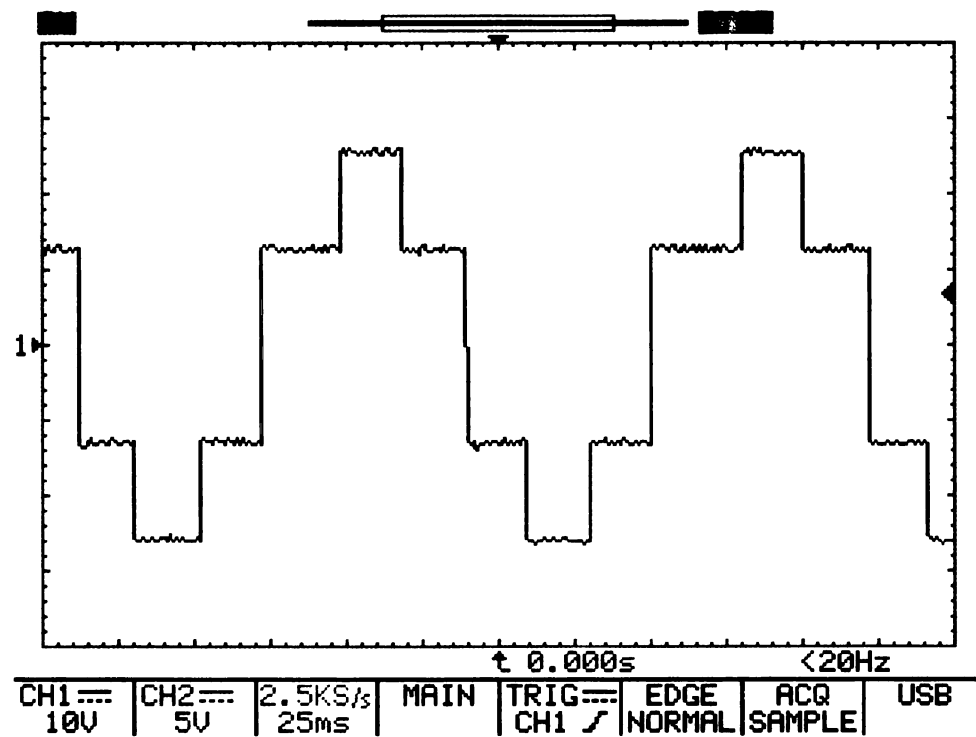


Figure 5.4. Line-to-line Inverter Output Voltage, Six Total Slave Modules.

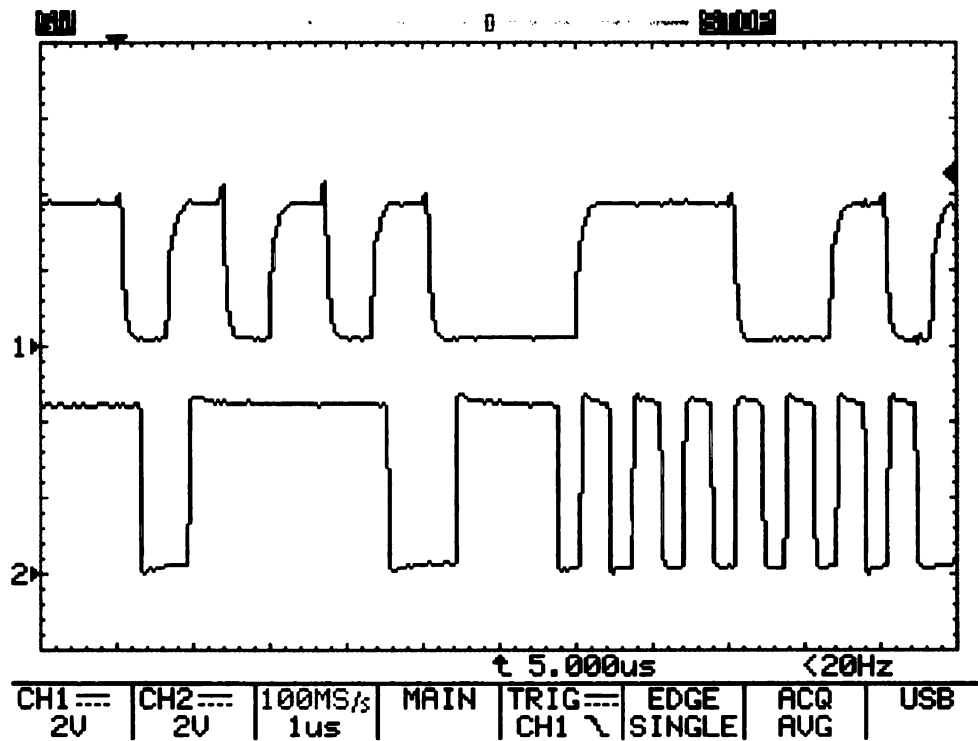


Figure 5.5. Design-One Master Multitasking During a USB Transfer.

Figure 5.4 is a plot of a line-to-line voltage with six total batteries and slave modules. No PWM is being conducted here.

Figure 5.5 shows the design-one master's availability during a USB transfer. The lower curve's high time represents time during which the master can send out module tokens. The upper curve shows the USB D- line.

Figures 5.6—5.9 are a demonstration of the EV₁ motor’s operation under the multilevel inverter. 5.6 and 5.7 were captured with the EV₁’s left front wheel off the ground while providing only a slight load torque by hand. Both plots show a synchronous frequency of 22Hz for the tests. I_d and I_q are quite noisy, even with a relatively constant amplitude (in units of 1/32 line-neutral battery voltage). Figure 5.6 has a slight load applied at six seconds, where I_q can be seen to increase slightly and stabilize, then drop off as the load is removed at almost nine seconds. This I_q increase is much more prominent in the three torque loadings of Figure 5.7, where the amplitude adjusts slightly to maintain an average I_d of 6A.

5.8 and 5.9 were captured while driving 3–4MPH in a parking lot, both peaking at a frequency of about 20Hz at 4MPH. I_d cannot be regulated at 40A as desired because of the conservative voltage limits used here. These were enacted to prevent high currents, though even with the limits in place, I_q still reaches over 120A for about a second. A throttle speed command was used for the driving tests. A polished torque-control algorithm and better current limiting would probably reduce the severe motor oscillations, but the project’s concept is very clearly demonstrated in these tests.

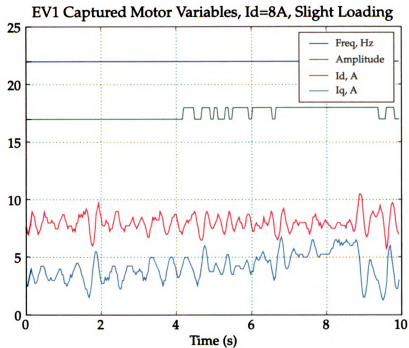


Figure 5.6. EV₁ Captured Motor Variables, $I_d^*=8A$, Slight Loading.

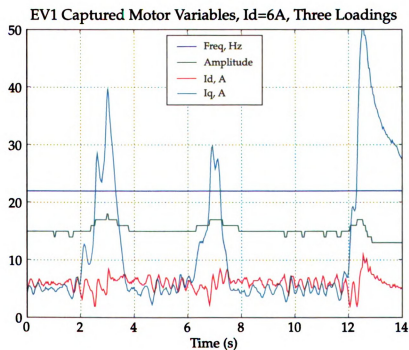


Figure 5.7. EV₁ Captured Motor Variables, $I_d^*=6A$, Three Loadings.

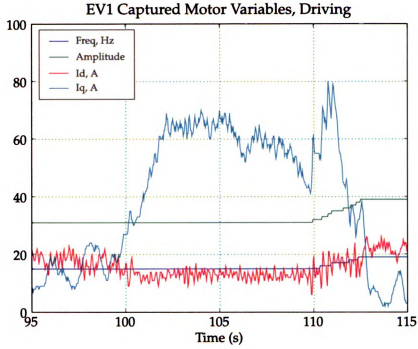


Figure 5.8. EV₁ Captured Motor Variables, $I_d^*=40A$, Driving.

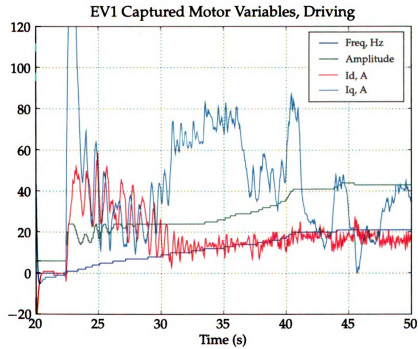


Figure 5.9. EV₁ Captured Motor Variables, $I_d^*=40A$, Driving.

CHAPTER 6

Conclusions

6.1 Design One

The first system design was functional enough to demonstrate the multilevel inverter topology. It moved the EV₁ about five feet, limited only by the garage length, with six batteries and battery modules. Torque control was demonstrated on a rewound 3/4HP induction motor. The slave modules unfortunately had reliability issues, mainly under no load. The IRS2004 gate drivers never functioned properly and would explode without warning. The master's token buffering was difficult to implement and only moderately reliable — some of the problems were never fully debugged. Not being able to turn on the slaves' outputs continuously was a major disadvantage for debugging. Finally, the master's USB interface was second-rate.

6.2 Design Two

Most of the issues with design one, except the EV₁ motor's oscillations, were resolved in design two. There were no slave module failures during testing, and the master's USB implementation was much improved, especially in terms of speed. A workable, inexpensive solution to surface-mount transformers was found for the slave modules, and the 10-pin connector greatly eased prototyping and installation. The master redesign greatly improved module communication reliability as well.

6.3 System and Topology

This work clearly demonstrates the ability of a multilevel inverter to drive a motor. The implemented system is cost-effective though still more expensive than competing PWM inverters, but its features — including individual battery balancing — somewhat make up for the difference. The slave modules and the master module are quite small and light compared to PWM inverters. The USB interface is extremely useful for debugging and collecting data, and proved an invaluable resource for development. Firmware upgrades are very reliable in the second design, and the system gives the user the ability to mix lead-acid and lithium-ion cells, a feat impractical in series-connected battery packs. The lack of a permanently series-connected pack greatly increases the inherent safety of the system and makes servicing much more convenient. Finally, the scalability of the project without any necessary redesign is crucial for supporting a varied product base, from golf carts to partially or fully electric trucks and buses.

6.4 Future Work

A major unresolved issue with the EV₁ inverter implementation is motor oscillation. This is evident in Figures 5.8 and 5.9 as unsteady currents I_d and I_q . Without a frequency-dependent limit on amplitude, it would also oscillate. Unfortunately, the limit prevents the vehicle from having enough torque to overcome small obstacles such as potholes. This is in contrast with the author's previous work, a Renault LeCar converted to use an AC motor and a standard PWM inverter [20], which could climb a fairly steep hill in third gear from a dead stop. The multilevel inverter does not make use of the speed sensor in a feedback loop and this may contribute to the oscillations. There should be otherwise no differences between the two types of inverters, and the EV₁'s motor may be simply more difficult to control. It is almost certainly possible

to resolve the oscillations with the correct application of control theory.

Although the theory for a battery selection algorithm was developed it was never fully coded or tested *in situ*. Currently only six batteries are supported and the inverter state (motoring/charging) is determined manually in ‘master.s.’ Contactors and relays for reconfiguring the inverter’s connections to allow motoring or charging were accommodated for but never installed in the inverter. There may be beneficial additions to the master module’s software such as positive or negative third harmonic injection as discussed in Section 2.1, efficiency improvements by means of power factor regulation during motoring, or DC charging capability to allow a solar or wind energy source. None of these changes require any hardware modification and all can be done by means of a USB software upgrade.

APPENDICES

APPENDIX A

PC Interface Software

A.1 Overview of Software

A program named Monitor was developed solely for the purpose of configuring and debugging the multilevel inverter. It is based on Trolltech's Qt4 'C++' software development kit for multi-platform compilation and makes use of libusb to communicate with the master module. The most recent version, Version 0.5, allows the user to test master-slave communications, view the current slave module list with slave temperatures, voltages, and hardware/firmware versions, send global and local commands to change the state of the slaves, upgrade the master and slave firmwares, and even reprogram slave data EEPROM to manually modify calibration values. For example, the user can choose to blink any slave's green LED to test its connection to the module bus. Any battery module can be turned on positive or negative continuously to test or debug power wiring. A "Roll Call" command assigns a new module an address, which can be changed with the "Readdress Module" command. All critical functions, such as rewriting flash or EEPROM, perform a communications check before executing. Module layouts (phase string configurations) can be read from or written to files on disk, and written to master EEPROM. Finally, communication errors are fully reported and all activity is logged to a file.

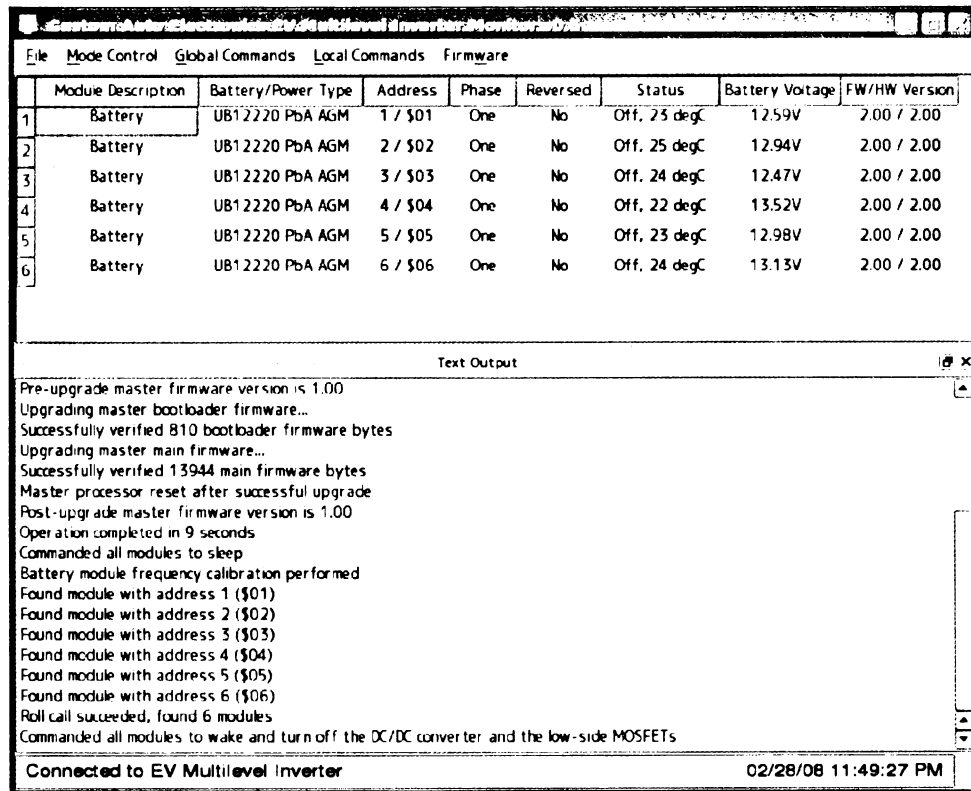


Figure A.1. Monitor v0.5 Screenshot with Six Battery Modules

Figure A.1 shows a screenshot of real-time updating temperatures and voltages from six actual battery modules after a master firmware upgrade was performed. The major portions of the Monitor v0.5 code can be seen below.

Future versions could potentially support motor performance graphing and logging as well, perhaps with an integration of matplotlib, the same software package used to generate many figures in this work. A large number of variables can be obtained from the master module, including the applied motor amplitude, untransformed and transformed currents, and an entire list of which slave modules are turned on. Another possible feature is automatic tuning of the inverter parameters for a given motor through test runs conducted by Monitor.

A.2 User Interface Layout: monitor.ui

```
1 <ui version="4.0" >
2   <class>Monitor</class>
3   <widget class="QMainWindow" name="Monitor" >
4     <property name="windowTitle" >
5       <string>Custom Auto Electronics EV Multilevel Inverter Monitor v0.5</string>
6     </property>
7     <widget class="QWidget" name="centralwidget" />
8     <widget class="QMenuBar" name="menubar" >
9       <property name="geometry" >
10        <rect>
11          <x>0</x>
12          <y>0</y>
13          <width>800</width>
14          <height>29</height>
15        </rect>
16      </property>
17      <widget class="QMenu" name="menuFile" >
18        <property name="title" >
19          <string>&File</string>
20        </property>
21        <addaction name="actionNewSettings" />
22        <addaction name="actionOpenSettings" />
23        <addaction name="actionSaveSettings" />
24        <addaction name="separator" />
25        <addaction name="actionReadSettings" />
26        <addaction name="actionWriteSettings" />
27        <addaction name="separator" />
28        <addaction name="actionExit" />
29      </widget>
30      <widget class="QMenu" name="menuModeControl" >
31        <property name="title" >
32          <string>&Mode Control</string>
33        </property>
34        <addaction name="actionEmergencyOff" />
35        <addaction name="separator" />
36        <addaction name="actionDiagnosticMode" />
37        <addaction name="actionMonitorMode" />
38      </widget>
39      <widget class="QMenu" name="menuGlobal" >
40        <property name="title" >
41          <string>&Global Commands</string>
42        </property>
43        <addaction name="actionSleepAll" />
44        <addaction name="actionWakeAll" />
45        <addaction name="actionDCDCOffAll" />
46        <addaction name="actionDCDCOnAll" />
47        <addaction name="actionBlinkAll" />
48        <addaction name="separator" />
49        <addaction name="actionCalibrate" />
50        <addaction name="actionRollCall" />
51        <addaction name="actionCheckCommAll" />
52      </widget>
53      <widget class="QMenu" name="menuLocal" >
54        <property name="title" >
55          <string>&Local Commands</string>
56        </property>
57        <addaction name="actionSleepM" />
58        <addaction name="actionWakeM" />
```

```

59     <addaction name="actionDCDCOffM" />
60     <addaction name="actionDCDCOnM" />
61     <addaction name="actionOnPM" />
62     <addaction name="actionOnNM" />
63     <addaction name="actionBlinkM" />
64     <addaction name="separator" />
65     <addaction name="actionCheckCommM" />
66     <addaction name="actionReadEEPROM" />
67     <addaction name="actionWriteEEPROM" />
68     <addaction name="actionReaddressM" />
69     <addaction name="actionChangePhase" />
70 </widget>
71 <widget class="QMenu" name="menuFirmware" >
72     <property name="title" >
73         <string>Firm&ware</string>
74     </property>
75     <addaction name="actionUpgradeMFW" />
76     <addaction name="actionUpgradeBFW" />
77 </widget>
78 <addaction name="menuFile" />
79 <addaction name="menuModeControl" />
80 <addaction name="menuGlobal" />
81 <addaction name="menuLocal" />
82 <addaction name="menuFirmware" />
83 </widget>
84 <widget class="QStatusBar" name="statusbar" />
85 <action name="actionNewSettings" >
86     <property name="text" >
87         <string>&New Module Layout</string>
88     </property>
89 </action>
90 <action name="actionOpenSettings" >
91     <property name="text" >
92         <string>&Open Module Layout</string>
93     </property>
94 </action>
95 <action name="actionSaveSettings" >
96     <property name="text" >
97         <string>&Save Module Layout</string>
98     </property>
99 </action>
100 <action name="actionReadSettings" >
101     <property name="text" >
102         <string>&Read Layout from EV</string>
103     </property>
104 </action>
105 <action name="actionWriteSettings" >
106     <property name="text" >
107         <string>&Write Layout to EV</string>
108     </property>
109 </action>
110 <action name="actionExit" >
111     <property name="text" >
112         <string>E&xit</string>
113     </property>
114 </action>
115 <action name="actionEmergencyOff" >
116     <property name="text" >
117         <string>&Emergency Off</string>
118     </property>
119 </action>
120 <action name="actionDiagnosticMode" >

```



```

121     <property name="checkable" >
122         <bool>true</bool>
123     </property>
124     <property name="checked" >
125         <bool>>false</bool>
126     </property>
127     <property name="text" >
128         <string>&Diagnostic Mode</string>
129     </property>
130 </action>
131 <action name="actionMonitorMode" >
132     <property name="checkable" >
133         <bool>true</bool>
134     </property>
135     <property name="checked" >
136         <bool>true</bool>
137     </property>
138     <property name="text" >
139         <string>&Monitor Mode</string>
140     </property>
141 </action>
142 <action name="actionSleepAll" >
143     <property name="text" >
144         <string>Put All to &Sleep</string>
145     </property>
146 </action>
147 <action name="actionWakeAll" >
148     <property name="text" >
149         <string>&Wake All (DC/DC, FETs Off)</string>
150     </property>
151 </action>
152 <action name="actionDCDCOffAll" >
153     <property name="text" >
154         <string>DC/DC Conv. On, FETs &Off All</string>
155     </property>
156 </action>
157 <action name="actionDCDCOnAll" >
158     <property name="text" >
159         <string>&DC/DC Conv. On, FETs On All</string>
160     </property>
161 </action>
162 <action name="actionBlinkAll" >
163     <property name="text" >
164         <string>&Blink All Modules LEDs</string>
165     </property>
166 </action>
167 <action name="actionCalibrate" >
168     <property name="text" >
169         <string>Calibrate &Frequency</string>
170     </property>
171 </action>
172 <action name="actionRollCall" >
173     <property name="text" >
174         <string>Perform &Roll Call</string>
175     </property>
176 </action>
177 <action name="actionCheckCommAll" >
178     <property name="text" >
179         <string>&Verify Communications</string>
180     </property>
181 </action>
182 <action name="actionSleepM" >

```

```

183     <property name="text" >
184         <string>Put Module to &Sleep</string>
185     </property>
186 </action>
187 <action name="actionWakeM" >
188     <property name="text" >
189         <string>&Wake (DC/DC, FETs Off)</string>
190     </property>
191 </action>
192 <action name="actionDCDCOffM" >
193     <property name="text" >
194         <string>DC/DC Conv. On, FETs &Off</string>
195     </property>
196 </action>
197 <action name="actionDCDCOnM" >
198     <property name="text" >
199         <string>&DC/DC Conv. On, FETs On</string>
200     </property>
201 </action>
202 <action name="actionOnPM" >
203     <property name="text" >
204         <string>Turn Module On &Positive</string>
205     </property>
206 </action>
207 <action name="actionOnNM" >
208     <property name="text" >
209         <string>Turn Module On &Negative</string>
210     </property>
211 </action>
212 <action name="actionBlinkM" >
213     <property name="text" >
214         <string>&Blink Module LED</string>
215     </property>
216 </action>
217 <action name="actionCheckCommM" >
218     <property name="text" >
219         <string>&Verify Communications</string>
220     </property>
221 </action>
222 <action name="actionReadEEPROM" >
223     <property name="text" >
224         <string>Read &EEPROM</string>
225     </property>
226 </action>
227 <action name="actionWriteEEPROM" >
228     <property name="text" >
229         <string>Wr&ite EEPROM</string>
230     </property>
231 </action>
232 <action name="actionReaddressM" >
233     <property name="text" >
234         <string>Re&address Module</string>
235     </property>
236 </action>
237 <action name="actionChangePhase" >
238     <property name="text" >
239         <string>&Change Phase</string>
240     </property>
241 </action>
242 <action name="actionUpgradeMFW" >
243     <property name="text" >
244         <string>Upgrade &Master...</string>

```

```
245     </property>
246 </action>
247 <action name="actionUpgradeBFW" >
248   <property name="text" >
249     <string>Upgrade & Battery Modules...</string>
250   </property>
251 </action>
252 </widget>
253 <resources/>
254 <connections/>
255 </ui>
```

A.3 Header File: monitor.h

```
1  #ifndef MONITOR_H
2  #define MONITOR_H
3
4
5  #include "ui_monitor.h"
6  #include <QApplication>
7  #include <QMainWindow>
8  #include <QWidget>
9  #include <QRect>
10 #include <QTableWidget>
11 #include <QListWidget>
12 #include <QStatusBar>
13 #include <QLabel>
14 #include <QTimer>
15 #include <QDockWidget>
16 #include <QFileDialog>
17 #include <QInputDialog>
18 #include <QMessageBox>
19 #include <sys/io.h>
20 #include <sys/types.h>
21 #include <stdio.h>
22 #include <stdlib.h>
23 #include <math.h>
24 #include <time.h>
25 #include <usb.h>
26
27
28 #define VENDOR_ID      0x04D8
29 #define PRODUCT_ID     0xDCBA
30
31 #define ERASE_PROG_ROW    1
32 #define WRITE_PROG_ROW   2
33 #define ERASE_DATA_WORD  3
34 #define ERASE_DATA_ROW   4
35 #define WRITE_DATA_WORD  5
36 #define WRITE_DATA_ROW   6
37 #define WRITE_CONFIG     7
38 #define READ_ROM_LONG    8
39 #define READ_ROM_HROW    9
40 #define WRITE_RAM_WORD   10
41 #define READ_RAM_LONG    11
42 #define JUMP_OR_RESET    12
43 #define CHECK_COMM_MOD   13
44 #define REPROGRAM_MODS   14
45 #define SEND_MODS_FAST   15
46 #define SEND_MODS_SLOW   16
47 #define RECEIVE_MODS     17
48 #define CLEAR_STATUS     31
49
50
51 class Monitor : public QMainWindow
52 {
53     Q_OBJECT
54
55 public:
56     Monitor(QMainWindow *parent=0);
57     ~Monitor();
58
```

```

59 private slots:
60     void keep_time();
61     void on_actionNewSettings_activated();
62     void on_actionOpenSettings_activated();
63     void on_actionSaveSettings_activated();
64     void on_actionReadSettings_activated();
65     void on_actionWriteSettings_activated();
66     void on_actionExit_activated();
67     void on_actionEmergencyOff_activated();
68     void on_actionDiagnosticMode_activated();
69     void on_actionMonitorMode_activated();
70     void on_actionSleepAll_activated();
71     void on_actionWakeAll_activated();
72     void on_actionDCDCOffAll_activated();
73     void on_actionDCDCOnAll_activated();
74     void on_actionBlinkAll_activated();
75     void on_actionCalibrate_activated();
76     void on_actionRollCall_activated();
77     void on_actionCheckCommAll_activated();
78     void on_actionSleepM_activated();
79     void on_actionWakeM_activated();
80     void on_actionOnPM_activated();
81     void on_actionOnNM_activated();
82     void on_actionDCDCOffM_activated();
83     void on_actionDCDCOnM_activated();
84     void on_actionBlinkM_activated();
85     void on_actionCheckCommM_activated();
86     void on_actionReadEEPROM_activated();
87     void on_actionWriteEEPROM_activated();
88     void on_actionReaddressM_activated();
89     void on_actionChangePhase_activated();
90     void on_actionUpgradeMFW_activated();
91     void on_actionUpgradeBFW_activated();
92
93 private:
94     Ui::Monitor ui;
95 };
96
97
98 #endif

```

A.4 Main Code: monitor.cpp

```
1  #include "monitor.h"
2
3
4  #define DEBUG
5
6
7  char statustext[300], timetext[50], manufacturer[80], product[80], buf[400],
   logged[400], bytestext[300];
8  int received=0, pleaseprint=0, addresses[120], nummods=0, firmware[2][2][120];
9  unsigned int address=0;
10 long timebegin=0;
11 unsigned long bufx[64];
12 struct usb_device *usb_dev=0;
13 struct usb_dev_handle *usb_handle=0;
14 struct usb_bus *usb_bus;
15 struct usb_device *dev;
16 QLabel *statuslabel;
17 QTimer *timer;
18 QTableWidget *biglist;
19 QTableWidgetItem *module_entries[128][8];
20 QListWidget *outputList;
21 QDockWidget *outputDock;
22
23
24 unsigned int readhex(char inhex);
25 int roll_call(bool onlynew);
26 void calibrate(void);
27 bool check_comm(int address);
28 bool check_boot(void);
29 bool issue_prog(long paddress, long pdata, char pcommand);
30 bool wait_finish(void);
31 bool write_fast(long val1, long val2, long val3, long val4);
32 bool write_slow(long val1, long val2, long val3, long val4);
33 long read_word(void);
34 void write_log(const char *logtext);
35
36
37 Monitor::Monitor(QMainWindow *parent)
38     : QMainWindow(parent)
39 {
40     int i;
41
42     ui.setupUi(this);
43
44     outputDock=new QDockWidget(tr("Text Output"), this);
45     outputDock->setAllowedAreas(Qt::TopDockWidgetArea|Qt::BottomDockWidgetArea);
46     outputList=new QListWidget(outputDock);
47     outputDock->setWidget(outputList);
48     outputDock->setMinimumWidth(200);
49     addDockWidget(Qt::BottomDockWidgetArea, outputDock);
50
51     QFont fixedFont("Helvetica", 11);
52     fixedFont.setStretch(110);
53     statuslabel=new QLabel(this);
54     statuslabel->setFont(fixedFont);
55     statuslabel->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::
        Fixed));
56
```

```

57     delete ui.centralwidget;
58     biglist=new QTableWidgetItem();
59     setCentralWidget(biglist);
60     biglist->setColumnCount(8);
61     biglist->setHorizontalHeaderLabels(QStringList()
62         <<"      Module Description      "
63         <<"      Battery/Power Type      "
64         <<"      Address      "
65         <<"      Phase      "
66         <<"      Reversed      "
67         <<"      Status      "
68         <<"Battery Voltage"
69         <<"FW/HW Version");
70     biglist->resizeColumnToContents(0);
71     biglist->resizeColumnToContents(1);
72     biglist->resizeColumnToContents(2);
73     biglist->resizeColumnToContents(3);
74     biglist->resizeColumnToContents(4);
75     biglist->resizeColumnToContents(5);
76     biglist->resizeColumnToContents(6);
77     biglist->resizeColumnToContents(7);
78
79     for(i=0;i<120;i++) addresses[i]=0;
80
81     sprintf(statustext, "<table width=100%><tr><td align=left>&nbsp;Unconnected</td>
82         <td align=right>&nbsp;</td></tr></table>");
83     statuslabel->setText(statustext);
84     statuslabel->setTextFormat(Qt::RichText);
85
86     usb_init();
87
88     timer=new QTimer(this);
89     connect(timer, SIGNAL(timeout()), SLOT(keep_time()));
90     timer->start(900);
91
92     statusBar()->addWidget(statuslabel, 1);
93
94     showMaximized();
95 }
96
97 Monitor::~Monitor()
98 {
99     if(usb_handle!=NULL) usb_close(usb_handle);
100 }
101
102
103 void Monitor::keep_time()
104 {
105     int i, j, k;
106     unsigned long volts, loctemp;
107     struct tm *timestruct;
108     time_t timenow;
109
110     timenow=time(NULL);
111     timestruct=localtime((const time_t *) &timenow);
112     if(timestruct->tm_hour==0) {
113         sprintf(timetext, "%02d/%02d/%02d&nbsp;12:%02d:%02d AM", timestruct->tm_mon+1,
114             timestruct->tm_mday, timestruct->tm_year%100, timestruct->tm_min,
115             timestruct->tm_sec);
116     } else if((timestruct->tm_hour>0)&&(timestruct->tm_hour<12)) {
117         sprintf(timetext, "%02d/%02d/%02d&nbsp;%2d:%02d:%02d AM", timestruct->tm_mon

```

```

        +1, timestruct->tm_mday, timestruct->tm_year%100, timestruct->tm_hour,
        timestruct->tm_min, timestruct->tm_sec);
116 } else if(timestruct->tm_hour==12) {
117     sprintf(timetext, "%02d/%02d/%02d 12:%02d:%02d PM", timestruct->tm_mon+1,
        timestruct->tm_mday, timestruct->tm_year%100, timestruct->tm_min,
        timestruct->tm_sec);
118 } else {
119     sprintf(timetext, "%02d/%02d/%02d %2d:%02d:%02d PM", timestruct->tm_mon
        +1, timestruct->tm_mday, timestruct->tm_year%100, timestruct->tm_hour-12,
        timestruct->tm_min, timestruct->tm_sec);
120 }
121
122 if(usb_dev==NULL) {
123     usb_find_busses();
124     usb_find_devices();
125     usb_busses=usb_get_busses();
126     for(usb_bus=usb_busses;usb_bus;usb_bus=usb_bus->next) {
127         for(dev=usb_bus->devices;dev;dev=dev->next) {
128             if((dev->descriptor.idVendor==VENDOR_ID)&&
129                 (dev->descriptor.idProduct==PRODUCT_ID)) usb_dev=dev;
130         }
131     }
132
133     if(usb_dev!=NULL) {
134         usb_handle=usb_open(usb_dev);
135         if(usb_handle!=NULL) {
136             received=usb_claim_interface(usb_handle, 0);
137             if(received==0) {
138                 received=usb_get_string_simple(usb_handle, 1, manufacturer, 80);
139                 // printf("Manufacturer: ");
140                 // for(i=0;i<received;i++) printf("%c", manufacturer[i]); printf("\n");
141                 received=usb_get_string_simple(usb_handle, 2, product, 80);
142                 // printf("Product: ");
143                 // for(i=0;i<received;i++) printf("%c", product[i]); printf("\n");
144                 received=usb_interrupt_read(usb_handle, 0x81, buf, 64, 0);
145                 // if(received>0) printf("Interrupt read works\n");
146             } else {
147                 // printf("Interface could not be claimed: %d\n", received);
148                 // printf("%s\n", usb_strerror());
149             }
150         }
151     } else {
152         nummods=0;
153         for(i=0;i<120;i++) {
154             addresses[i]=0;
155             firmware[0][0][i]=firmware[0][1][i]=firmware[1][0][i]=firmware[1][1][i]=0;
156         }
157     }
158 } else {
159     for(i=0;i<10;i++) {
160         received=usb_interrupt_read(usb_handle, 0x81, buf, 64, 0);
161         // printf("%d\n", received);
162         if(received>0) break;
163     }
164     if(received<=0) {
165         usb_reset(usb_handle);
166         usb_close(usb_handle);
167         usb_dev=0; usb_handle=0;
168     } else {
169         for(i=0;i<biglist->rowCount();i++) {
170             // Find the actual module address
171             sscanf(module_entries[i][2]->text().toAscii(), "%d", &j);

```



```

172
173 // Read the battery voltage
174 write_fast(0x8180+j, 0, 0, 0);
175 k=read_word();
176 volts=(unsigned long) k*2;
177
178 // Read the local temperature byte
179 write_fast(0x8000+j, 0x800A, 0, 0);
180 loctemp=(unsigned long) (read_word() & 0x00FF);
181
182 if(k!=-1) {
183     if(strcmp(module_entries[i][5]->text().toAscii(), "(Not present)")) {
184         delete module_entries[i][5];
185         module_entries[i][5]=new QTableWidgetItem("(Not present)");
186         module_entries[i][5]->setTextAlignment(Qt::AlignHCenter);
187         biglist->setItem((int) i, 5, module_entries[i][5]);
188     }
189
190     if(strcmp(module_entries[i][6]->text().toAscii(), "N/A")) {
191         delete module_entries[i][6];
192         module_entries[i][6]=new QTableWidgetItem("N/A");
193         module_entries[i][6]->setTextAlignment(Qt::AlignHCenter);
194         biglist->setItem((int) i, 6, module_entries[i][6]);
195     }
196
197     if(strcmp(module_entries[i][7]->text().toAscii(), "N/A")) {
198         delete module_entries[i][7];
199         module_entries[i][7]=new QTableWidgetItem("N/A");
200         module_entries[i][7]->setTextAlignment(Qt::AlignHCenter);
201         biglist->setItem((int) i, 7, module_entries[i][7]);
202     }
203 } else {
204     sprintf(buf, "Off, %ld degC", loctemp-128);
205     if(strcmp(module_entries[i][5]->text().toAscii(), buf)) {
206         delete module_entries[i][5];
207         module_entries[i][5]=new QTableWidgetItem(buf);
208         module_entries[i][5]->setTextAlignment(Qt::AlignHCenter);
209         biglist->setItem((int) i, 5, module_entries[i][5]);
210     }
211
212     sprintf(buf, "%5.2fV ", 7.5+((double) (volts))*0.0095);
213     if(strcmp(module_entries[i][6]->text().toAscii(), buf)) {
214         delete module_entries[i][6];
215         module_entries[i][6]=new QTableWidgetItem(buf);
216         module_entries[i][6]->setTextAlignment(Qt::AlignHCenter);
217         biglist->setItem((int) i, 6, module_entries[i][6]);
218     }
219
220     sprintf(buf, "%d.%02X / %d.%02X", firmware[0][1][addresses[i]-1],
221             firmware[0][0][addresses[i]-1], firmware[1][1][addresses[i]-1],
222             firmware[1][0][addresses[i]-1]);
223     if(strcmp(module_entries[i][7]->text().toAscii(), buf)) {
224         delete module_entries[i][7];
225         module_entries[i][7]=new QTableWidgetItem(buf);
226         module_entries[i][7]->setTextAlignment(Qt::AlignHCenter);
227         biglist->setItem((int) i, 7, module_entries[i][7]);
228     }
229 }
230 // roll_call(TRUE);
231 }

```

```

232     }
233
234     if(received>0) {
235         sprintf(statustext, "<table width=100%><tr><td align=left>&nbsp;   Connected to
236             %s</td><td align=right>%s&nbsp;   </td></tr></table>", product, timetext);
237         statuslabel->setText(statustext);
238         statuslabel->setTextFormat(Qt::RichText);
239     } else {
240         sprintf(statustext, "<table width=100%><tr><td align=left>&nbsp;   Unconnected </
241             td><td align=right>%s&nbsp;   </td></tr></table>", timetext);
242         statuslabel->setText(statustext);
243         statuslabel->setTextFormat(Qt::RichText);
244     }
245     timer->start(750);
246 }
247
248 int roll_call(bool onlynew)
249 {
250     char buf[80];
251     int i, j, k, lastfound=0, newfound=0;
252
253     lastfound=addresses[nummods-1];
254     if(!onlynew) nummods=0;
255
256     if(!write_fast(0x807E, 0, 0, 0)) {
257         outputList->addItem("Failed to command modules to roll call, check USB signal"
258             );
259         outputList->scrollToItem(outputList->item(outputList->count()-1));
260         sprintf(logged, "roll_call:    Notified \"Failed to command modules to roll
261             call, check USB signal\\\"\\n");
262         write_log(logged);
263         return -2;
264     }
265
266     if(!onlynew) {
267         for(j=1;j<=120;j++) {
268             // Try to see if the addressed module (address of j) is present
269             if(!write_fast(0x8000+j, 0, 0, 0)) {
270                 outputList->addItem("Failed to command modules to roll call, check USB
271                     signal");
272                 outputList->scrollToItem(outputList->item(outputList->count()-1));
273                 sprintf(logged, "roll_call:    Notified \"Failed to command modules to
274                     roll call, check USB signal\\\"\\n");
275                 write_log(logged);
276                 return -2;
277             }
278
279             // Get the reply, if one came
280             k=read_word();
281             if(k==2) {
282                 outputList->addItem("Failed to command modules to roll call, check USB
283                     signal");
284                 outputList->scrollToItem(outputList->item(outputList->count()-1));
285                 sprintf(logged, "roll_call:    Notified \"Failed to command modules to
286                     roll call, check USB signal\\\"\\n");
287                 write_log(logged);
288                 return -2;
289             }
290
291             // Interpret the result

```

```

286     if(k!=-1) {
287         if((k&0xFF)==j) {
288             sprintf(buf, "Found module with address %d ($%02X)", j, j);
289             outputList->addItem(buf);
290             outputList->scrollToItem(outputList->item(outputList->count()-1));
291             sprintf(logged, "roll_call:    Notified \"%s\"\n", buf);
292             write_log(logged);
293             addresses[nummods++]=j; addresses[nummods]=0; lastfound=j;
294         } else {
295             sprintf(buf, "Reply error for address %d ($%02X), incorrectly %d ($%02X)
296                 ", j, j, k&0x00FF, k&0x00FF);
297             outputList->addItem(buf);
298             outputList->scrollToItem(outputList->item(outputList->count()-1));
299             sprintf(logged, "roll_call:    Notified \"%s\"\n", buf);
300             write_log(logged);
301             return -1;
302         }
303     }
304 }
305
306 if((nummods<120)&&(lastfound<120)) {
307     // Try to see if the addressed module (address of lastfound+1) is present
308     if(!write_fast(0x8001+lastfound, 0, 0, 0)) {
309         outputList->addItem("Failed to command modules to roll call, check USB
310             signal");
311         outputList->scrollToItem(outputList->item(outputList->count()-1));
312         sprintf(logged, "roll_call:    Notified \"Failed to command modules to roll
313             call, check USB signal\"\n");
314         write_log(logged);
315         return -2;
316     }
317
318     // Command the non-addressed module to assign itself an address
319     if(!write_fast(0x80FF, 0, 0, 0)) {
320         outputList->addItem("Failed to command modules to roll call, check USB
321             signal");
322         outputList->scrollToItem(outputList->item(outputList->count()-1));
323         sprintf(logged, "roll_call:    Notified \"Failed to command modules to roll
324             call, check USB signal\"\n");
325         write_log(logged);
326         return -2;
327     }
328
329     // Get the reply, if one came
330     k=read_word();
331     if(k==-2) {
332         outputList->addItem("Failed to command modules to roll call, check USB
333             signal");
334         outputList->scrollToItem(outputList->item(outputList->count()-1));
335         sprintf(logged, "roll_call:    Notified \"Failed to command modules to roll
336             call, check USB signal\"\n");
337         write_log(logged);
338         return -2;
339     }
340
341     // Interpret the result
342     if(k!=-1) {
343         if((k&0xFF)==0xFF) {
344             sprintf(buf, "Assigned module new address %d ($%02X)", lastfound+1,
345                 lastfound+1);
346             outputList->addItem(buf);

```

```

340     outputList->scrollToItem(outputList->item(outputList->count()-1));
341     addresses[nummods++]=lastfound+1; addresses[nummods]=0; newfound++;
342     sprintf(logged, "roll_call:    Notified \"%s\"\n", buf);
343     write_log(logged);
344 } else {
345     sprintf(buf, "Warning: Unusual reply %d ($%02X) while testing for
        unassigned modules, should be none or $FF", k&0x00FF, k&0x00FF);
346     outputList->addItem(buf);
347     outputList->scrollToItem(outputList->item(outputList->count()-1));
348     sprintf(logged, "roll_call:    Notified \"%s\"\n", buf);
349     write_log(logged);
350 }
351 }
352 }
353
354 if(!write_fast(0x8000, 0, 0, 0)) {
355     outputList->addItem("Failed to command modules to roll call, check USB signal"
        );
356     outputList->scrollToItem(outputList->item(outputList->count()-1));
357     sprintf(logged, "roll_call:    Notified \"Failed to command modules to roll
        call, check USB signal\"\n");
358     write_log(logged);
359     return -2;
360 }
361
362 if(!onlynew) {
363     if(nummods==0) {
364         sprintf(buf, "Roll call succeeded, found no modules");
365     } else if(nummods==1) {
366         sprintf(buf, "Roll call succeeded, found 1 module");
367     } else {
368         sprintf(buf, "Roll call succeeded, found %d modules", nummods);
369     }
370     outputList->addItem(buf);
371     outputList->scrollToItem(outputList->item(outputList->count()-1));
372 }
373
374 if((onlynew)&&(!newfound)) return 0;
375
376 // Command the battery modules to enter their bootloaders
377 if(!write_slow(0x81AC, 0x8153, 0x8100, 0)) {
378     outputList->addItem("Failed to command modules to enter bootloader, check USB
        signal");
379     outputList->scrollToItem(outputList->item(outputList->count()-1));
380     sprintf(logged, "roll_call:    Notified \"Failed to command modules to enter
        bootloader, check USB signal\"\n");
381     write_log(logged);
382     return -2;
383 }
384
385 for(i=(onlynew?(nummods-1):0);i<nummods;i++) {
386     // Target the selected module
387     if(!write_slow(0x8100+addresses[i], 0x8100+addresses[i], 0x8181, 0)) {
388         outputList->addItem("Failed to address module in bootloader, check USB
            signal");
389         outputList->scrollToItem(outputList->item(outputList->count()-1));
390         sprintf(logged, "roll_call:    Notified \"Failed to address module in
            bootloader, check USB signal\"\n");
391         write_log(logged);
392         return -2;
393     }
394 }

```

```

395 // Get the reply, if one came
396 k=read_word();
397 if(k== -2) {
398     outputList->addItem("Failed to address module in bootloader, check USB
        signal");
399     outputList->scrollToItem(outputList->item(outputList->count()-1));
400     sprintf(logged, "roll_call:    Notified \"Failed to address module in enter
        bootloader, check USB signal\\\"\\n");
401     write_log(logged);
402     return -2;
403 }
404
405 // Interpret the result
406 if((k== -1) || ((k & 0xFF) != (0x4B ^ addresses[i]))) {
407     sprintf(buf, "Enter bootloader failed on module with address $%02X",
        addresses[i]);
408     outputList->addItem(buf);
409     outputList->scrollToItem(outputList->item(outputList->count()-1));
410     sprintf(logged, "roll_call:    Notified \"%s\\\"\\n", buf);
411     write_log(logged);
412     return -2;
413 }
414
415 firmware[0][0][addresses[i]-1]=0x00;
416 firmware[0][1][addresses[i]-1]=0x00;
417 firmware[1][0][addresses[i]-1]=0x00;
418 firmware[1][1][addresses[i]-1]=0x00;
419
420 // Read the firmware version low byte
421 if(!write_slow(0x8122, 0x8100, 0x8184, 0)) {
422     outputList->addItem("Failed to read module firmware version from bootloader,
        check USB signal");
423     outputList->scrollToItem(outputList->item(outputList->count()-1));
424     sprintf(logged, "roll_call:    Notified \"Failed to read module firmware
        version from bootloader, check USB signal\\\"\\n");
425     write_log(logged);
426     return -2;
427 }
428
429 // Get the reply, if one came
430 k=read_word();
431 if(k== -2) {
432     outputList->addItem("Failed to read module firmware version from bootloader,
        check USB signal");
433     outputList->scrollToItem(outputList->item(outputList->count()-1));
434     sprintf(logged, "roll_call:    Notified \"Failed to read module firmware
        version from bootloader, check USB signal\\\"\\n");
435     write_log(logged);
436     return -2;
437 }
438 if(k!= -1) firmware[0][0][addresses[i]-1]=k & 0xFF;
439
440 // Read the firmware version high byte
441 if(!write_slow(0x8123, 0x8100, 0x8184, 0)) {
442     outputList->addItem("Failed to read module firmware version from bootloader,
        check USB signal");
443     outputList->scrollToItem(outputList->item(outputList->count()-1));
444     sprintf(logged, "roll_call:    Notified \"Failed to read module firmware
        version from bootloader, check USB signal\\\"\\n");
445     write_log(logged);
446     return -2;
447 }

```

```

448
449 // Get the reply, if one came
450 k=read_word();
451 if(k==2) {
452     outputList->addItem("Failed to read module firmware version from bootloader,
453                          check USB signal");
454     outputList->scrollToItem(outputList->item(outputList->count()-1));
455     sprintf(logged, "roll_call:    Notified \"Failed to read module firmware
456                  version from bootloader, check USB signal\\\"\\n");
457     write_log(logged);
458     return -2;
459 }
460 if(k!=1) firmware[0][1][addresses[i]-1]=k&0xFF;
461
462 // Read the bootloader version low byte
463 if(!write_slow(0x81E0, 0x8106, 0x8184, 0)) {
464     outputList->addItem("Failed to read module firmware version from bootloader,
465                          check USB signal");
466     outputList->scrollToItem(outputList->item(outputList->count()-1));
467     sprintf(logged, "roll_call:    Notified \"Failed to read module firmware
468                  version from bootloader, check USB signal\\\"\\n");
469     write_log(logged);
470     return -2;
471 }
472
473 // Get the reply, if one came
474 k=read_word();
475 if(k==2) {
476     outputList->addItem("Failed to read module firmware version from bootloader,
477                          check USB signal");
478     outputList->scrollToItem(outputList->item(outputList->count()-1));
479     sprintf(logged, "roll_call:    Notified \"Failed to read module firmware
480                  version from bootloader, check USB signal\\\"\\n");
481     write_log(logged);
482     return -2;
483 }
484 if(k!=1) firmware[1][0][addresses[i]-1]=k&0xFF;
485
486 // Read the bootloader version high byte
487 if(!write_slow(0x81E1, 0x8106, 0x8184, 0)) {
488     outputList->addItem("Failed to read module firmware version from bootloader,
489                          check USB signal");
490     outputList->scrollToItem(outputList->item(outputList->count()-1));
491     sprintf(logged, "roll_call:    Notified \"Failed to read module firmware
492                  version from bootloader, check USB signal\\\"\\n");
493     write_log(logged);
494     return -2;
495 }
496
497 // Get the reply, if one came
498 k=read_word();
499 if(k==2) {
500     outputList->addItem("Failed to read module firmware version from bootloader,
501                          check USB signal");
502     outputList->scrollToItem(outputList->item(outputList->count()-1));
503     sprintf(logged, "roll_call:    Notified \"Failed to read module firmware
504                  version from bootloader, check USB signal\\\"\\n");
505     write_log(logged);
506     return -2;
507 }
508 if(k!=1) firmware[1][1][addresses[i]-1]=k&0xFF;
509 }

```

```

500
501 // Command the battery modules to exit their bootloaders softly
502 if(!write_slow(0x81FF, 0x81FF, 0x81FF, 0)) {
503     outputList->addItem("Failed to command modules to exit bootloader, check USB
        signal");
504     outputList->scrollToItem(outputList->item(outputList->count()-1));
505     sprintf(logged, "roll_call:    Notified \"Failed to command modules to exit
        bootloader, check USB signal\"\\n");
506     write_log(logged);
507     return -2;
508 }
509
510 return (onlynew?1:nummods);
511 }
512
513
514 void calibrate(void)
515 {
516     int i;
517
518     if(!usb_handle) return;
519
520     // Send two 0x0179s to make the modules sleep
521     if(!write_fast(0x8179, 0x8179, 0, 0)) {
522         outputList->addItem("Failed to command modules to sleep, check USB signal");
523         outputList->scrollToItem(outputList->item(outputList->count()-1));
524         sprintf(logged, "calibrate:    Notified \"Failed to command modules to sleep,
            check USB signal\"\\n");
525         write_log(logged);
526         return;
527     }
528     outputList->addItem("Commanded all modules to sleep");
529     outputList->scrollToItem(outputList->item(outputList->count()-1));
530     sprintf(logged, "calibrate:    Notified \"Commanded all modules to sleep\"\\n");
531     write_log(logged);
532
533     // Get the modules into the synchronization routine
534     if(!write_fast(0x81FF, 0x81FF, 0, 0)) {
535         outputList->addItem("Failed to command modules to calibrate, check USB signal"
            );
536         outputList->scrollToItem(outputList->item(outputList->count()-1));
537         sprintf(logged, "calibrate:    Notified \"Failed to command modules to
            calibrate, check USB signal\"\\n");
538         write_log(logged);
539         return;
540     }
541
542     // Send 125kBd all-zero words to the modules to help them synchronize
543     for(i=0;i<32;i++) {
544         if(!write_slow(0x8000, 0, 0, 0)) {
545             outputList->addItem("Failed to command modules to calibrate, check USB
                signal");
546             outputList->scrollToItem(outputList->item(outputList->count()-1));
547             sprintf(logged, "calibrate:    Notified \"Failed to command modules to
                calibrate, check USB signal\"\\n");
548             write_log(logged);
549             return;
550         }
551     }
552
553     // Send a $56 and a $78 to the modules to tell them that this function is valid
554     if(!write_slow(0x8156, 0x8178, 0, 0)) {

```

```

555     outputList->addItem("Failed to command modules to calibrate, check USB signal"
556     );
556     outputList->scrollToItem(outputList->item(outputList->count()-1));
557     sprintf(logged, "calibrate:    Notified \"Failed to command modules to
558         calibrate, check USB signal\\\"\\n");
558     write_log(logged);
559     return;
560 }
561
562 outputList->addItem("Battery module frequency calibration performed");
563 outputList->scrollToItem(outputList->item(outputList->count()-1));
564 sprintf(logged, "calibrate:    Notified \"Battery module frequency calibration
565     performed\\\"\\n");
565 write_log(logged);
566
567 // Delay to let the modules rewrite their calibration bytes (EEPROM $00)
568 for(i=0;i<20;i++) received=usb_interrupt_read(usb_handle, 0x82, buf, 64, 0);
569 }
570
571
572 bool check_comm(int address)
573 {
574     // Send a command to the DSPIC to check a battery module's communications
575     if(!issue_prog(0, address, CHECK_COMM_MOD)) {
576         outputList->addItem("Failed to check battery module's communications, check
577             USB signal");
577         outputList->scrollToItem(outputList->item(outputList->count()-1));
578         sprintf(logged, "check_comm:    Notified \"Failed to check battery module's
579             communications, check USB signal\\\"\\n");
579         write_log(logged);
580         return FALSE;
581     }
582
583     // Report the result
584     if(bufx[16]==1) {
585         sprintf(buf, "Check communications succeeded on module with address %d ($%02X)
586             ", address, address);
586         outputList->addItem(buf);
587         outputList->scrollToItem(outputList->item(outputList->count()-1));
588         sprintf(logged, "check_comm:    Notified \"%s\\\"\\n", buf);
589         write_log(logged);
590         return TRUE;
591     } else if(bufx[16]==2) {
592         sprintf(buf, "Check communications failed on module with address %d ($%02X) -
593             timeout", address, address);
593         outputList->addItem(buf);
594         outputList->scrollToItem(outputList->item(outputList->count()-1));
595         sprintf(logged, "check_comm:    Notified \"%s\\\"\\n", buf);
596         write_log(logged);
597         return FALSE;
598     } else if(bufx[16]==3) {
599         sprintf(buf, "Check communications failed on module with address %d ($%02X) -
600             bad byte", address, address);
600         outputList->addItem(buf);
601         outputList->scrollToItem(outputList->item(outputList->count()-1));
602         sprintf(logged, "check_comm:    Notified \"%s\\\"\\n", buf);
603         write_log(logged);
604         return FALSE;
605     } else {
606         sprintf(buf, "Check communications failed on module with address %d ($%02X) -
607             unknown", address, address);
607         outputList->addItem(buf);

```



```

608     outputList->scrollToItem(outputList->item(outputList->count()-1));
609     sprintf(logged, "check_comm:   Notified \"%s\"\n", buf);
610     write_log(logged);
611     return FALSE;
612 }
613 }
614
615
616 bool check_boot(void)
617 {
618     int i=0, k;
619
620     while(addresses[i]!=0x00) {
621         // Command the battery modules to enter their bootloaders
622         if(!write_slow(0x81AC, 0x8153, 0x8100+addresses[i], 0)) {
623             outputList->addItem("Failed to command modules to enter bootloader, check
                USB signal");
624             outputList->scrollToItem(outputList->item(outputList->count()-1));
625             sprintf(logged, "check_boot:   Notified \"Failed to command modules to enter
                bootloader, check USB signal\"\n");
626             write_log(logged);
627             return FALSE;
628         }
629
630         // Get the reply, if one came
631         k=read_word();
632         if(k==-2) {
633             outputList->addItem("Failed to command modules to roll call, check USB
                signal");
634             outputList->scrollToItem(outputList->item(outputList->count()-1));
635             sprintf(logged, "check_boot:   Notified \"Failed to command modules to roll
                call, check USB signal\"\n");
636             write_log(logged);
637             return FALSE;
638         }
639
640         // Command the battery modules to exit their bootloaders softly
641         if(!write_slow(0x81FF, 0x81FF, 0x81FF, 0)) {
642             outputList->addItem("Failed to command modules to exit bootloader, check USB
                signal");
643             outputList->scrollToItem(outputList->item(outputList->count()-1));
644             sprintf(logged, "check_boot:   Notified \"Failed to command modules to exit
                bootloader, check USB signal\"\n");
645             write_log(logged);
646             return FALSE;
647         }
648
649         // Interpret the result
650         if((k!=-1)||((k&0xFF)!=(0x4B^addresses[i]))) {
651             sprintf(buf, "Enter bootloader failed on module with address %02X",
                addresses[i]);
652             outputList->addItem(buf);
653             outputList->scrollToItem(outputList->item(outputList->count()-1));
654             sprintf(logged, "check_boot:   Notified \"%s\"\n", buf);
655             write_log(logged);
656             return FALSE;
657         }
658         i++;
659     }
660
661     sprintf(buf, "Enter bootloader succeeded on all %d modules", i);
662     outputList->addItem(buf);

```

```

663     outputList->scrollToItem(outputList->item(outputList->count()-1));
664     sprintf(logged, "check_boot:   Notified \"%s\"\\n", buf);
665     write_log(logged);
666     return TRUE;
667 }
668
669
670 bool issue_prog(long paddress, long pdata, char pcommand)
671 {
672     int i;
673
674     // Write out a data, address, and programming command, and wait for completion
675     // Note: For WRITE_PROG_ROW or WRITE_DATA_ROW, buf[32..63] will not be ruined
676     for(i=0;i<32;i++) buf[i]=0;
677     buf[8]=(char) (paddress&0xFF);
678     buf[9]=(char) ((paddress>>8)&0xFF);
679     buf[10]=(char) ((paddress>>16)&0xFF);
680     buf[12]=(char) (pdata&0xFF);
681     buf[13]=(char) ((pdata>>8)&0xFF);
682     buf[11]=0x00; buf[15]=pcommand;
683     if(usb_interrupt_write(usb_handle, 0x01, buf, 64, 0)!=64) return FALSE;
684     for(i=0;i<64;i++) sprintf(bytestext+(i*3), " %02X", (unsigned char) buf[i]);
685     sprintf(logged, "issue_prog:   Written to USB 0x01: %s\\n", bytestext);
686     write_log(logged);
687     if(pcommand==WRITE_RAM_WORD) return TRUE;
688     if(!wait_finish()) return FALSE;
689     return TRUE;
690 }
691
692
693 bool wait_finish(void)
694 {
695     int i;
696
697     // Test 'progstat' bit six to see if the operation completed yet
698     do {
699         received=usb_interrupt_read(usb_handle, 0x82, buf, 64, 0);
700         if(received==64) {
701             for(i=0;i<64;i++) bufx[i]=(unsigned long) ((unsigned char) buf[i]);
702         } else {
703             return FALSE;
704         }
705     } while(!(bufx[1]&0x40));
706     for(i=0;i<64;i++) sprintf(bytestext+(i*3), " %02X", (unsigned char) buf[i]);
707     sprintf(logged, "wait_finish:   Read from USB 0x82: %s\\n", bytestext);
708     write_log(logged);
709
710     // Send a programming command of 'CLEAR_STATUS' (31) to clear this flag
711     for(i=0;i<64;i++) buf[i]=((i==15)?CLEAR_STATUS:0x00);
712     if(usb_interrupt_write(usb_handle, 0x01, buf, 64, 0)!=64) return FALSE;
713     for(i=0;i<64;i++) sprintf(bytestext+(i*3), " %02X", (unsigned char) buf[i]);
714     sprintf(logged, "wait_finish:   Written to USB 0x01: %s\\n", bytestext);
715     write_log(logged);
716
717     // 'bufx[64]' contains the last USB 0x82 interrupt read data
718     return TRUE;
719 }
720
721
722 bool write_fast(long val1, long val2, long val3, long val4)
723 {
724     int i;

```

```

725
726 // Write out a set of four or less 1MBd words to the battery modules
727 // Note: write_fast(0x8123, 0x8000, 0, 0) sends 0x0123 and 0x0000
728 for(i=0;i<64;i++) buf[i]=0;
729 buf[0]=(char) (val1&0xFF);
730 buf[1]=(char) ((val1>>8)&0xFF);
731 buf[2]=(char) (val2&0xFF);
732 buf[3]=(char) ((val2>>8)&0xFF);
733 buf[4]=(char) (val3&0xFF);
734 buf[5]=(char) ((val3>>8)&0xFF);
735 buf[6]=(char) (val4&0xFF);
736 buf[7]=(char) ((val4>>8)&0xFF);
737 buf[15]=SEND_MODS_FAST;
738 if(usb_interrupt_write(usb_handle, 0x01, buf, 64, 0)!=64) return FALSE;
739 for(i=0;i<64;i++) sprintf(bytestext+(i*3), " %02X", (unsigned char) buf[i]);
740 sprintf(logged, "write_fast:   Written to USB 0x01: %s\n", bytestext);
741 write_log(logged);
742 return TRUE;
743 }
744
745
746 bool write_slow(long val1, long val2, long val3, long val4)
747 {
748     int i;
749
750     // Write out a set of four or less 125kBd words to the battery modules
751     // Note: write_slow(0x8123, 0x8000, 0, 0) sends 0x0123 and 0x0000
752     for(i=0;i<64;i++) buf[i]=0;
753     buf[0]=(char) (val1&0xFF);
754     buf[1]=(char) ((val1>>8)&0xFF);
755     buf[2]=(char) (val2&0xFF);
756     buf[3]=(char) ((val2>>8)&0xFF);
757     buf[4]=(char) (val3&0xFF);
758     buf[5]=(char) ((val3>>8)&0xFF);
759     buf[6]=(char) (val4&0xFF);
760     buf[7]=(char) ((val4>>8)&0xFF);
761     buf[15]=SEND_MODS_SLOW;
762     if(usb_interrupt_write(usb_handle, 0x01, buf, 64, 0)!=64) return FALSE;
763     for(i=0;i<64;i++) sprintf(bytestext+(i*3), " %02X", (unsigned char) buf[i]);
764     sprintf(logged, "write_slow:   Written to USB 0x01: %s\n", bytestext);
765     write_log(logged);
766     return TRUE;
767 }
768
769
770 long read_word(void)
771 {
772     int i;
773
774     // Write out a command to read a word from a battery module
775     for(i=0;i<64;i++) buf[i]=0;
776     buf[15]=RECEIVE_MODS;
777     if(usb_interrupt_write(usb_handle, 0x01, buf, 64, 0)!=64) return -1;
778     for(i=0;i<64;i++) sprintf(bytestext+(i*3), " %02X", (unsigned char) buf[i]);
779     sprintf(logged, "read_word:   Written to USB 0x01: %s\n", bytestext);
780     write_log(logged);
781
782     // Read the DSPIC's reply (no delay is needed)
783     received=usb_interrupt_read(usb_handle, 0x82, buf, 64, 0);
784     if(received==64) {
785         for(i=0;i<64;i++) bufx[i]=(unsigned long) ((unsigned char) buf[i]);
786     } else {

```

```

787     return -2;
788 }
789 for(i=0;i<64;i++) sprintf(bytestext+(i*3), " %02X", (unsigned char) buf[i]);
790 sprintf(logged, "read_word:      Read from USB 0x82: %s\n", bytestext);
791 write_log(logged);
792
793 // Test 'progstat' bit seven to see if a word was received
794 if(bufx[1]&0x80) {
795     // Send a programming command of 'CLEAR_STATUS' (31) to clear this flag
796     for(i=0;i<64;i++) buf[i]=((i==15)?CLEAR_STATUS:0x00);
797     if(usb_interrupt_write(usb_handle, 0x01, buf, 64, 0)!=64) return -1;
798     for(i=0;i<64;i++) sprintf(bytestext+(i*3), " %02X", (unsigned char) buf[i]);
799     sprintf(logged, "read_word:      Written to USB 0x01: %s\n", bytestext);
800     write_log(logged);
801
802     // Return the word's value
803     return (bufx[0]|((bufx[1]&0x01)<<8));
804 } else {
805     return -1;
806 }
807 }
808
809
810 void write_log(const char *logtext)
811 {
812     printf("%s", logtext);
813 }
814
815
816 void Monitor::on_actionNewSettings_activated()
817 {
818     int i, j, retval;
819
820     if(!usb_handle) return;
821
822     if(biglist->rowCount()!=0) {
823         retval=QMessageBox::warning(this, "Create New Module Layout", "Creating a new
            module layout will replace the module table contents with the result of a
            roll call. The roll call will discover all present modules and optionally
            assign an address to one new module. Alternatively, the Global Commands
            >> Roll Call menu item will only add to the table, not subtract from it.
            Continuing the New Module Layout command will destroy the existing table
            contents.\n\nTo save the current layout as a file and continue, press Save
            .\nTo discard the current layout and continue, press Discard.\nTo
            discontinue New Module Layout, press Cancel.", QMessageBox::Cancel|
            QMessageBox::Save|QMessageBox::Discard, QMessageBox::Cancel);
824         if(retval==QMessageBox::Cancel) return;
825         if(retval==QMessageBox::Save) on_actionSaveSettings_activated();
826     }
827
828     calibrate();
829     roll_call(FALSE);
830
831     biglist->setRowCount(nummods);
832     for(i=0;i<nummods;i++) {
833         module_entries[i][0]=new QTableWidgetItem("Battery");
834         module_entries[i][1]=new QTableWidgetItem("UB12220 PbA AGM");
835         sprintf(buf, "%d / %02X", addresses[i], addresses[i]);
836         module_entries[i][2]=new QTableWidgetItem(buf);
837         module_entries[i][3]=new QTableWidgetItem("One");
838         module_entries[i][4]=new QTableWidgetItem("No");
839         module_entries[i][5]=new QTableWidgetItem("(Not present)");

```

```

840     module_entries[i][6]=new QTableWidgetItem("N/A");
841     module_entries[i][7]=new QTableWidgetItem("N/A");
842
843     for(j=0;j<8;j++) {
844         module_entries[i][j]->setTextAlignment(Qt::AlignHCenter);
845         biglist->setItem((int) i, j, module_entries[i][j]);
846     }
847     biglist->resizeRowToContents(i);
848 }
849 }
850
851
852 void Monitor::on_actionOpenSettings_activated()
853 {
854     char x, moddesc[40], battype[40], address[40], phase[40], reversed[40];
855     int i, j, retval;
856
857     if(biglist->rowCount()!=0) {
858         retval=QMessageBox::warning(this, "Open Module Layout", "Creating a new module
            layout will replace the module table contents with the selected file's
            contents. Continuing the Open Layout command will destroy the existing
            table contents.\n\nTo save the current layout as a file and continue,
            press Save.\nTo discard the current layout and continue, press Discard.\n
            To discontinue Open Module Layout, press Cancel.", QMessageBox::Cancel|
            QMessageBox::Save|QMessageBox::Discard, QMessageBox::Cancel);
859         if(retval==QMessageBox::Cancel) return;
860         if(retval==QMessageBox::Save) on_actionSaveSettings_activated();
861     }
862
863     QString s=QFileDialog::getOpenFileName(this, "Open Module Layout", "layout.cae",
        "Module Layout Files (*.cae)");
864     QFile findata(s);
865
866     if(s.size()<2) return;
867
868     if(!findata.open(QIODevice::ReadOnly|QIODevice::Text)) {
869         sprintf(buf, "Cannot open %s for read", (const char *) s.toAscii());
870         outputList->addItem(buf);
871         outputList->scrollToItem(outputList->item(outputList->count()-1));
872         return;
873     }
874
875     findata.readLine(buf, 10);
876     sscanf(buf, "%d", &i);
877     if((i<1)||(i>120)) {
878         sprintf(buf, "%s is not a valid module layout file", (const char *) s.toAscii
            ());
879         outputList->addItem(buf);
880         outputList->scrollToItem(outputList->item(outputList->count()-1));
881         return;
882     }
883     biglist->setRowCount(i);
884
885     if(usb_handle!=NULL) {
886         calibrate();
887         roll_call(FALSE);
888     }
889
890     for(i=0;i<biglist->rowCount();i++) {
891         j=0;
892         do {
893             findata.getChar(&x);

```

```

894         if((x!=0x0D)&&(x!=0x0A)) moddesc[j++]= (x==0x09)?0x00:x;
895     } while(x!=0x09);
896     j=0;
897     do {
898         findata.getChar(&x);
899         battype[j++]=(x==0x09)?0x00:x;
900     } while(x!=0x09);
901     j=0;
902     do {
903         findata.getChar(&x);
904         address[j++]=(x==0x09)?0x00:x;
905     } while(x!=0x09);
906     j=0;
907     do {
908         findata.getChar(&x);
909         phase[j++]=(x==0x09)?0x00:x;
910     } while(x!=0x09);
911     j=0;
912     do {
913         findata.getChar(&x);
914         reversed[j++]=((x==0x0D)||(x==0x0A))?0x00:x;
915     } while((x!=0x0D)&&(x!=0x0A));
916
917     module_entries[i][0]=new QTableWidgetItem(moddesc);
918     module_entries[i][1]=new QTableWidgetItem(battype);
919     module_entries[i][2]=new QTableWidgetItem(address);
920     module_entries[i][3]=new QTableWidgetItem(phase);
921     module_entries[i][4]=new QTableWidgetItem(reversed);
922     module_entries[i][5]=new QTableWidgetItem("(Not present)");
923     module_entries[i][6]=new QTableWidgetItem("N/A");
924     module_entries[i][7]=new QTableWidgetItem("N/A");
925
926     for(j=0;j<8;j++) {
927         module_entries[i][j]->setTextAlignment(Qt::AlignHCenter);
928         biglist->setItem((int) i, j, module_entries[i][j]);
929     }
930     biglist->resizeRowToContents(i);
931 }
932
933 findata.close();
934
935 sprintf(buf, "Module layout loaded from %s", (const char *) s.toAscii());
936 outputList->addItem(buf);
937 outputList->scrollToItem(outputList->item(outputList->count()-1));
938 }
939
940
941 void Monitor::on_actionSaveSettings_activated()
942 {
943     int i;
944
945     if(biglist->rowCount()==0) return;
946
947     QString s=QFileDialog::getSaveFileName(this, "Save Module Layout", "layout.cae",
948         "Module Layout Files (*.cae)");
949     QFile foutdata(s);
950
951     if(s.size()<2) return;
952
953     if(!foutdata.open(QIODevice::WriteOnly|QIODevice::Truncate|QIODevice::Text)) {
954         sprintf(buf, "Cannot open %s for write", (const char *) s.toAscii());
955         outputList->addItem(buf);
956     }

```

```

955     outputList->scrollToItem(outputList->item(outputList->count()-1));
956     return;
957 }
958
959     sprintf(buf, "%d\r\n", biglist->rowCount());
960     foutdata.write(buf, strlen(buf));
961
962     for(i=0;i<biglist->rowCount();i++) {
963         sprintf(buf, "%s\t%s\t%s\t%s\t%s\r\n",
964             (const char *) module_entries[i][0]->text().toAscii(),
965             (const char *) module_entries[i][1]->text().toAscii(),
966             (const char *) module_entries[i][2]->text().toAscii(),
967             (const char *) module_entries[i][3]->text().toAscii(),
968             (const char *) module_entries[i][4]->text().toAscii());
969         foutdata.write(buf, strlen(buf));
970     }
971
972     foutdata.close();
973
974     sprintf(buf, "Current module layout saved to %s", (const char *) s.toAscii());
975     outputList->addItem(buf);
976     outputList->scrollToItem(outputList->item(outputList->count()-1));
977 }
978
979
980 void Monitor::on_actionReadSettings_activated()
981 {
982 }
983
984
985 void Monitor::on_actionWriteSettings_activated()
986 {
987     int i, j;
988     unsigned long outprom[144];
989
990     if(!usb_handle) return;
991
992     // Set up the list of active modules (zero meaning not present)
993     for(i=0;i<144;i++) outprom[i]=0;
994     for(i=0;i<biglist->rowCount();i++) {
995         sscanf(module_entries[i][2]->text().toAscii(), "%d", &j);
996         j--;
997         if(strcmp(module_entries[i][3]->text().toAscii(), "One")==0) {
998             outprom[j>>3]|=(1<<((j%8)*2));
999         } else if(strcmp(module_entries[i][3]->text().toAscii(), "Two")==0) {
1000             outprom[j>>3]|=(2<<((j%8)*2));
1001         } else if(strcmp(module_entries[i][3]->text().toAscii(), "Three")==0) {
1002             outprom[j>>3]|=(3<<((j%8)*2));
1003         }
1004         if(strcmp(module_entries[i][4]->text().toAscii(), "Yes")==0)
1005             outprom[(j>>4)+15]|=(1<<((j%16)));
1006         outprom[j+24]=22000>>2;
1007     }
1008
1009     for(i=0;i<144;i++) printf("write_set:    outprom[%d]=%04lX\n", i, outprom[i]);
1010
1011     for(i=0;i<144;i+=2) {
1012         // Erase two words of data memory
1013         if(!issue_prog(0x007FFC00+(i<<1), 0, ERASE_DATA_WORD)) goto ws_commerror;
1014         if(!issue_prog(0x007FFC02+(i<<1), 0, ERASE_DATA_WORD)) goto ws_commerror;
1015
1016         // Write out EEPROM words in pairs

```

```

1017     if(!issue_prog(0x007FFC00+(i<<1), outprom[i], WRITE_DATA_WORD))
1018         goto ws_commererror;
1019     if(!issue_prog(0x007FFC02+(i<<1), outprom[i+1], WRITE_DATA_WORD))
1020         goto ws_commererror;
1021
1022     // Verify EEPROM words in pairs
1023     if(!issue_prog(0x007FFC00+(i<<1), 0, READ_ROM_LONG)) goto ws_commererror;
1024     if((bufx[16]!=(outprom[i]&0xFF))||((bufx[17]!=(outprom[i]>>8)&0xFF))||
1025         (bufx[19]!=(outprom[i+1]&0xFF))||((bufx[20]!=(outprom[i+1]>>8)&0xFF)))
1026         goto ws_promerror;
1027 }
1028
1029 // Reset the master processor
1030 if(!issue_prog(0, 0, JUMP_OR_RESET)) goto ws_commererror;
1031 for(i=0;i<64;i++) buf[i]=((i==15)?JUMP_OR_RESET:0x00);
1032 if(usb_interrupt_write(usb_handle, 0x01, buf, 64, 0)!=64) goto ws_commererror;
1033 outputList->addItem("Master processor reset after successful module layout write
        to EEPROM");
1034 outputList->scrollToItem(outputList->item(outputList->count()-1));
1035 sprintf(logged, "write_set:    Notified \"Master processor reset after
        successful module layout write to EEPROM\\\"\\n");
1036 write_log(logged);
1037
1038 // Delay for at least two seconds
1039 for(i=0;i<2000;i++) if(usb_interrupt_read(usb_handle, 0x82, buf, 64, 0)!=64)
1040     goto ws_commererror;
1041 return;
1042
1043 ws_commererror:
1044 outputList->addItem("Communication error while writing master EEPROM");
1045 outputList->scrollToItem(outputList->item(outputList->count()-1));
1046 sprintf(logged, "write_set:    Notified \"Communication error while writing
        master EEPROM\\\"\\n");
1047 write_log(logged);
1048 return;
1049
1050 ws_promerror:
1051 outputList->addItem("Verification error while writing master EEPROM");
1052 outputList->scrollToItem(outputList->item(outputList->count()-1));
1053 sprintf(logged, "write_set:    Notified \"Verification error while writing
        master EEPROM\\\"\\n");
1054 write_log(logged);
1055 }
1056
1057
1058 void Monitor::on_actionExit_activated()
1059 {
1060     QApplication::exit(0);
1061 }
1062
1063
1064 void Monitor::on_actionEmergencyOff_activated()
1065 {
1066 }
1067
1068
1069 void Monitor::on_actionDiagnosticMode_activated()
1070 {
1071     ui.actionDiagnosticMode->setChecked(TRUE);
1072     ui.actionMonitorMode->setChecked(FALSE);
1073 }
1074

```



```

1075
1076 void Monitor::on_actionMonitorMode_activated()
1077 {
1078     ui.actionDiagnosticMode->setChecked(FALSE);
1079     ui.actionMonitorMode->setChecked(TRUE);
1080 }
1081
1082
1083 void Monitor::on_actionSleepAll_activated()
1084 {
1085     if(!usb_handle) return;
1086
1087     // Send one 0x0179 to make the modules enter the state "sleeping"
1088     if(!write_fast(0x8179, 0, 0, 0)) {
1089         outputList->addItem("Failed to communicate with master, check USB signal");
1090         outputList->scrollToItem(outputList->item(outputList->count()-1));
1091         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
            check USB signal\\\"\\n");
1092         write_log(logged);
1093         return;
1094     }
1095     outputList->addItem("Commanded all modules to sleep");
1096     outputList->scrollToItem(outputList->item(outputList->count()-1));
1097     sprintf(logged, "monitor:      Notified \"Commanded all modules to sleep\\\"\\n");
1098     write_log(logged);
1099 }
1100
1101
1102 void Monitor::on_actionWakeAll_activated()
1103 {
1104     if(!usb_handle) return;
1105
1106     // Send one 0x017A to make the modules enter the state "idle"
1107     if(!write_fast(0x817A, 0, 0, 0)) {
1108         outputList->addItem("Failed to communicate with master, check USB signal");
1109         outputList->scrollToItem(outputList->item(outputList->count()-1));
1110         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
            check USB signal\\\"\\n");
1111         write_log(logged);
1112         return;
1113     }
1114     outputList->addItem("Commanded all modules to wake and turn off the DC/DC
        converter and the low-side MOSFETs");
1115     outputList->scrollToItem(outputList->item(outputList->count()-1));
1116     sprintf(logged, "monitor:      Notified \"Commanded all modules to wake and turn
        off the DC/DC converter and the low-side MOSFETs\\\"\\n");
1117     write_log(logged);
1118 }
1119
1120
1121 void Monitor::on_actionDCDCOffAll_activated()
1122 {
1123     if(!usb_handle) return;
1124
1125     // Send one 0x017B to make the modules enter the state "ready"
1126     if(!write_fast(0x817B, 0, 0, 0)) {
1127         outputList->addItem("Failed to communicate with master, check USB signal");
1128         outputList->scrollToItem(outputList->item(outputList->count()-1));
1129         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
            check USB signal\\\"\\n");
1130         write_log(logged);
1131         return;

```

```

1132     }
1133     outputList->addItem("Commanded all modules to turn on the DC/DC converter and
        turn off the low-side MOSFETs");
1134     outputList->scrollToItem(outputList->item(outputList->count()-1));
1135     sprintf(logged, "monitor:      Notified \"Commanded all modules to turn on the
        DC/DC converter and turn off the low-side MOSFETs\\\"\\n");
1136     write_log(logged);
1137 }
1138
1139
1140 void Monitor::on_actionDCDCOnAll_activated()
1141 {
1142     if(!usb_handle) return;
1143
1144     // Send one 0x017C to make the modules enter the state "running"
1145     if(!write_fast(0x817C, 0, 0, 0)) {
1146         outputList->addItem("Failed to communicate with master, check USB signal");
1147         outputList->scrollToItem(outputList->item(outputList->count()-1));
1148         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
            check USB signal\\\"\\n");
1149         write_log(logged);
1150         return;
1151     }
1152     outputList->addItem("Commanded all modules to turn on the DC/DC converter and
        the low-side MOSFETs");
1153     outputList->scrollToItem(outputList->item(outputList->count()-1));
1154     sprintf(logged, "monitor:      Notified \"Commanded all modules to turn on the
        DC/DC converter and the low-side MOSFETs\\\"\\n");
1155     write_log(logged);
1156 }
1157
1158
1159 void Monitor::on_actionBlinkAll_activated()
1160 {
1161     if(!usb_handle) return;
1162
1163     // Send one 0x017D to make the modules blink
1164     if(!write_fast(0x817D, 0, 0, 0)) {
1165         outputList->addItem("Failed to communicate with master, check USB signal");
1166         outputList->scrollToItem(outputList->item(outputList->count()-1));
1167         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
            check USB signal\\\"\\n");
1168         write_log(logged);
1169         return;
1170     }
1171     outputList->addItem("Commanded all modules to sleep and blink the green LED for
        three seconds");
1172     outputList->scrollToItem(outputList->item(outputList->count()-1));
1173     sprintf(logged, "monitor:      Notified \"Commanded all modules to sleep and
        blink the green LED for three seconds\\\"\\n");
1174     write_log(logged);
1175 }
1176
1177
1178 void Monitor::on_actionCalibrate_activated()
1179 {
1180     if(!usb_handle) return;
1181
1182     calibrate();
1183 }
1184
1185

```

```

1186 void Monitor::on_actionRollCall_activated()
1187 {
1188     if(!usb_handle) return;
1189
1190     roll_call(FALSE);
1191 }
1192
1193
1194 void Monitor::on_actionCheckCommAll_activated()
1195 {
1196     int i, numer=0, denom=0;
1197
1198     if(!usb_handle) return;
1199
1200     calibrate();
1201     if(roll_call(FALSE)<=0) return;
1202
1203     for(i=0;i<256;i++) {
1204         if(addresses[i]) {
1205             if(check_comm(addresses[i])) numer++;
1206             denom++;
1207         } else {
1208             break;
1209         }
1210     }
1211
1212     if((denom)&&(numer==denom)) {
1213         sprintf(buf, "Check communications succeeded on all %d modules", denom);
1214         outputList->addItem(buf);
1215         outputList->scrollToItem(outputList->item(outputList->count()-1));
1216         sprintf(logged, "verify_comm:  Notified \"%s\\n\"", buf);
1217         write_log(logged);
1218     } else if((denom)&&(numer!=denom)) {
1219         sprintf(buf, "Check communications succeeded on only %d of %d modules", numer,
1220             denom);
1221         outputList->addItem(buf);
1222         outputList->scrollToItem(outputList->item(outputList->count()-1));
1223         sprintf(logged, "verify_comm:  Notified \"%s\\n\"", buf);
1224         write_log(logged);
1225     }
1226
1227
1228 void Monitor::on_actionSleepM_activated()
1229 {
1230     bool ok;
1231     int adr;
1232
1233     if(!usb_handle) return;
1234
1235     adr=QInputDialog::getInteger(this, "Put Module to Sleep", "Enter module address:
1236                                     ", 1, 1, 120, 1, &ok);
1237
1238     if(!ok) return;
1239
1240     // Send one 0x0001 to make the addressed module enter the state "sleeping"
1241     if(!write_fast(0x8000+adr, 0x8001, 0, 0)) {
1242         outputList->addItem("Failed to communicate with master, check USB signal");
1243         outputList->scrollToItem(outputList->item(outputList->count()-1));
1244         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
1245                             check USB signal\\n\\n\"");
1246         write_log(logged);
1247         return;

```

```

1245     }
1246     sprintf(buf, "Commanded module with address %d ($%02X) to sleep", adr, adr);
1247     outputList->addItem(buf);
1248     outputList->scrollToItem(outputList->item(outputList->count()-1));
1249     sprintf(logged, "monitor:      Notified \"%s\\\"\\n", buf);
1250     write_log(logged);
1251 }
1252
1253
1254 void Monitor::on_actionWakeM_activated()
1255 {
1256     bool ok;
1257     int adr;
1258
1259     if(!usb_handle) return;
1260
1261     adr=QInputDialog::getInteger(this, "Turn Off / Wake Module", "Enter module
        address:                                ", 1, 1, 120, 1, &ok);
1262     if(!ok) return;
1263
1264     // Send one 0x0002 to make the addressed module enter the state "idle"
1265     if(!write_fast(0x8000+adr, 0x8002, 0, 0)) {
1266         outputList->addItem("Failed to communicate with master, check USB signal");
1267         outputList->scrollToItem(outputList->item(outputList->count()-1));
1268         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1269         write_log(logged);
1270         return;
1271     }
1272     sprintf(buf, "Commanded module with address %d ($%02X) to wake and turn off the
        DC/DC converter and the low-side MOSFETs", adr, adr);
1273     outputList->addItem(buf);
1274     outputList->scrollToItem(outputList->item(outputList->count()-1));
1275     sprintf(logged, "monitor:      Notified \"%s\\\"\\n", buf);
1276     write_log(logged);
1277 }
1278
1279
1280 void Monitor::on_actionOnPM_activated()
1281 {
1282     bool ok;
1283     int adr;
1284
1285     if(!usb_handle) return;
1286
1287     adr=QInputDialog::getInteger(this, "Turn Module On Positive", "Enter module
        address:                                ", 1, 1, 120, 1, &ok);
1288     if(!ok) return;
1289
1290     // Send one 0x0100 to make the addressed module turn on positive
1291     if(!write_fast(0x8100+adr, 0, 0, 0)) {
1292         outputList->addItem("Failed to communicate with master, check USB signal");
1293         outputList->scrollToItem(outputList->item(outputList->count()-1));
1294         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1295         write_log(logged);
1296         return;
1297     }
1298     sprintf(buf, "Commanded module with address %d ($%02X) to turn on positive
        continuously", adr, adr);
1299     outputList->addItem(buf);
1300     outputList->scrollToItem(outputList->item(outputList->count()-1));

```

```

1301     sprintf(logged, "monitor:      Notified \"%s\"\n", buf);
1302     write_log(logged);
1303 }
1304
1305
1306 void Monitor::on_actionOnNM_activated()
1307 {
1308     bool ok;
1309     int adr;
1310
1311     if(!usb_handle) return;
1312
1313     adr=QInputDialog::getInteger(this, "Turn Module On Negative", "Enter module
        address:                                ", 1, 1, 120, 1, &ok);
1314     if(!ok) return;
1315
1316     // Send one 0x0080 to make the addressed module turn on positive
1317     if(!write_fast(0x8080+adr, 0, 0, 0)) {
1318         outputList->addItem("Failed to communicate with master, check USB signal");
1319         outputList->scrollToItem(outputList->item(outputList->count()-1));
1320         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
            check USB signal\"\n");
1321         write_log(logged);
1322         return;
1323     }
1324     sprintf(buf, "Commanded module with address %d ($%02X) to turn on negative
        continuously", adr, adr);
1325     outputList->addItem(buf);
1326     outputList->scrollToItem(outputList->item(outputList->count()-1));
1327     sprintf(logged, "monitor:      Notified \"%s\"\n", buf);
1328     write_log(logged);
1329 }
1330
1331
1332 void Monitor::on_actionDCDCOffM_activated()
1333 {
1334     bool ok;
1335     int adr;
1336
1337     if(!usb_handle) return;
1338
1339     adr=QInputDialog::getInteger(this, "DC/DC Converter Off", "Enter module address:
        ", 1, 1, 120, 1, &ok);
1340     if(!ok) return;
1341
1342     // Send one 0x0003 to make the addressed module enter the state "idle"
1343     if(!write_fast(0x8000+adr, 0x8003, 0, 0)) {
1344         outputList->addItem("Failed to communicate with master, check USB signal");
1345         outputList->scrollToItem(outputList->item(outputList->count()-1));
1346         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
            check USB signal\"\n");
1347         write_log(logged);
1348         return;
1349     }
1350     sprintf(buf, "Commanded module with address %d ($%02X) to turn on the DC/DC
        converter and turn off the low-side MOSFETs", adr, adr);
1351     outputList->addItem(buf);
1352     outputList->scrollToItem(outputList->item(outputList->count()-1));
1353     sprintf(logged, "monitor:      Notified \"%s\"\n", buf);
1354     write_log(logged);
1355 }
1356

```

```

1357
1358 void Monitor::on_actionDCDCOnM_activated()
1359 {
1360     bool ok;
1361     int adr;
1362
1363     if(!usb_handle) return;
1364
1365     adr=QInputDialog::getInteger(this, "DC/DC Converter On", "Enter module address:
        ", 1, 1, 120, 1, &ok);
1366
1367     if(!ok) return;
1368
1369     // Send one 0x0004 to make the addressed module enter the state "idle"
1370     if(!write_fast(0x8000+adr, 0x8004, 0, 0)) {
1371         outputList->addItem("Failed to communicate with master, check USB signal");
1372         outputList->scrollToItem(outputList->item(outputList->count()-1));
1373         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1374         write_log(logged);
1375         return;
1376     }
1377     sprintf(buf, "Commanded module with address %d ($%02X) to turn on the DC/DC
        converter and the low-side MOSFETs", adr, adr);
1378     outputList->addItem(buf);
1379     outputList->scrollToItem(outputList->item(outputList->count()-1));
1380     sprintf(logged, "monitor:      Notified \"%s\\\"\\n", buf);
1381     write_log(logged);
1382 }
1383
1384 void Monitor::on_actionBlinkM_activated()
1385 {
1386     bool ok;
1387     int adr;
1388
1389     if(!usb_handle) return;
1390
1391     adr=QInputDialog::getInteger(this, "Blink Module LED", "Enter module address:
        ", 1, 1, 120, 1, &ok);
1392
1393     if(!ok) return;
1394
1395     // Send one 0x0005 to make the addressed module blink
1396     if(!write_fast(0x8000+adr, 0x8005, 0, 0)) {
1397         outputList->addItem("Failed to communicate with master, check USB signal");
1398         outputList->scrollToItem(outputList->item(outputList->count()-1));
1399         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1400         write_log(logged);
1401         return;
1402     }
1403     sprintf(buf, "Commanded module with address %d ($%02X) to sleep and blink the
        green LED for three seconds", adr, adr);
1404     outputList->addItem(buf);
1405     outputList->scrollToItem(outputList->item(outputList->count()-1));
1406     sprintf(logged, "monitor:      Notified \"%s\\\"\\n", buf);
1407     write_log(logged);
1408 }
1409
1410 void Monitor::on_actionCheckCommM_activated()
1411 {
1412     bool ok;

```

```

1413     int adr;
1414
1415     if(!usb_handle) return;
1416
1417     adr=QInputDialog::getInteger(this, "Verify Communications", "Enter module
        address:                                ", 1, 1, 120, 1, &ok);
1418     if(!ok) return;
1419
1420     check_comm(adr);
1421 }
1422
1423
1424 void Monitor::on_actionReadEEPROM_activated()
1425 {
1426     bool ok;
1427     int adr, eeadl, eedata;
1428
1429     if(!usb_handle) return;
1430
1431     eeadl=QInputDialog::getInteger(this, "Read Module EEPROM", "Enter EEPROM address
        :\n$00: Oscillator calibration byte\n$01: Module address\n$02: Voltage
        offset calibration byte\n$03: Temperature offset calibration byte
        ", 4, 0, 127, 1, &ok);
1432     if(!ok) return;
1433
1434     adr=QInputDialog::getInteger(this, "Read Module EEPROM", "Enter module address:
        ", 1, 1, 120, 1, &ok);
1435     if(!ok) return;
1436
1437     if(!check_comm(adr)) return;
1438
1439     // Read from the addressed module's EEPROM
1440     if(!write_fast(0x8000+adr, 0, 0, 0)) {
1441         outputList->addItem("Failed to communicate with master, check USB signal");
1442         outputList->scrollToItem(outputList->item(outputList->count()-1));
1443         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
            check USB signal\\\"\\n");
1444         write_log(logged);
1445         return;
1446     }
1447     if(!write_fast(0x8006, 0, 0, 0)) {
1448         outputList->addItem("Failed to communicate with master, check USB signal");
1449         outputList->scrollToItem(outputList->item(outputList->count()-1));
1450         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
            check USB signal\\\"\\n");
1451         write_log(logged);
1452         return;
1453     }
1454     if(!write_fast(0x8000+eeadl, 0, 0, 0)) {
1455         outputList->addItem("Failed to communicate with master, check USB signal");
1456         outputList->scrollToItem(outputList->item(outputList->count()-1));
1457         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
            check USB signal\\\"\\n");
1458         write_log(logged);
1459         return;
1460     }
1461
1462     // Read and interpret the module's reply
1463     eedata=read_word();
1464     if(eedata>=0) {
1465         eedata&=0xFF;
1466         switch(eeadl) {

```

```

1467     case 0x00:
1468         sprintf(buf, "EEPROM $00 (oscillator calibration byte) of module %d ($%02X
            ) is %d ($%02X)", adr, adr, eedata, eedata);
1469         break;
1470     case 0x01:
1471         sprintf(buf, "EEPROM $01 (module address) of module %d ($%02X) is %d ($%02
            X)", adr, adr, eedata, eedata);
1472         break;
1473     case 0x02:
1474         sprintf(buf, "EEPROM $02 (voltage calibration byte) of module %d ($%02X)
            is %d ($%02X)", adr, adr, eedata, eedata);
1475         break;
1476     case 0x03:
1477         sprintf(buf, "EEPROM $03 (temperature calibration byte) of module %d ($%02
            X) is %d ($%02X)", adr, adr, eedata, eedata);
1478         break;
1479     default:
1480         sprintf(buf, "EEPROM $%02X of module %d ($%02X) is %d ($%02X)", eeadl, adr
            , adr, eedata, eedata);
1481     }
1482 } else if(eedata==--1) {
1483     sprintf(buf, "EEPROM of module %d ($%02X) failed to read", adr, adr);
1484 } else {
1485     sprintf(buf, "Failed to communicate with master, check USB signal");
1486 }
1487 outputList->addItem(buf);
1488 outputList->scrollToItem(outputList->item(outputList->count()-1));
1489 sprintf(logged, "monitor:         Notified \"%s\\n\"", buf);
1490 write_log(logged);
1491 }
1492
1493
1494 void Monitor::on_actionWriteEEPROM_activated()
1495 {
1496     bool ok;
1497     int adr, eeadl, eedr, eedata, retval;
1498
1499     if(!usb_handle) return;
1500
1501     eeadl=QInputDialog::getInteger(this, "Write Module EEPROM", "Enter EEPROM
        address:\\n$00: Oscillator calibration byte\\n$01: Module address\\n$02:
        Voltage offset calibration byte\\n$03: Temperature offset calibration byte
        ", 4, 0, 127, 1, &ok);
1502     if(!ok) return;
1503
1504     if(eeadl==0x00) {
1505         retval=QMessageBox::warning(this, "Write Module EEPROM", "EEPROM address $00
            is the 12MHz calibration byte. This location can be automatically
            adjusted via the global Calibrate Frequency command. Modifying it
            manually may create unpredictable results and is not recommended. Are you
            sure you want to overwrite it?", QMessageBox::Yes|QMessageBox::No,
            QMessageBox::No);
1506         if(retval==QMessageBox::No) return;
1507     } else if(eeadl==0x01) {
1508         on_actionReaddressM_activated();
1509     } else if(eeadl==0x02) {
1510         retval=QMessageBox::warning(this, "Write Module EEPROM", "EEPROM address $02
            is the voltage offset calibration byte. This location is automatically
            written during normal inverter operation. Modifying it manually may
            create unpredictable results and is not recommended. Are you sure you
            want to overwrite it?", QMessageBox::Yes|QMessageBox::No, QMessageBox::No)
            ;

```



```

1511     if(retval==QMessageBox::No) return;
1512 } else if(eeadl==0x03) {
1513     retval=QMessageBox::warning(this, "Write Module EEPROM", "EEPROM address $03
        is the temperature offset calibration byte. This location is
        automatically written during normal inverter operation. Modifying it
        manually may create unpredictable results and is not recommended. Are you
        sure you want to overwrite it?", QMessageBox::Yes|QMessageBox::No,
        QMessageBox::No);
1514     if(retval==QMessageBox::No) return;
1515 }
1516
1517 eedr=QInputDialog::getInteger(this, "Write Module EEPROM", "Enter EEPROM data:
        ", 0, 0, 255, 1, &ok);
1518
1519 if(!ok) return;
1520
1521 adr=QInputDialog::getInteger(this, "Write Module EEPROM", "Enter module address:
        ", 1, 1, 120, 1, &ok);
1522
1523 if(!ok) return;
1524
1525 if(!check_comm(adr)) return;
1526
1527 // Send the EEPROM key ($92~address) to the module to allow temporary writing
1528 if(!write_fast(0x8000+adr, 0, 0, 0)) {
1529     outputList->addItem("Failed to communicate with master, check USB signal");
1530     outputList->scrollToItem(outputList->item(outputList->count()-1));
1531     sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1532     write_log(logged);
1533     return;
1534 }
1535
1536 if(!write_fast(0x8192~adr, 0, 0, 0)) {
1537     outputList->addItem("Failed to communicate with master, check USB signal");
1538     outputList->scrollToItem(outputList->item(outputList->count()-1));
1539     sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1540     write_log(logged);
1541     return;
1542 }
1543
1544 // Write to the addressed module's EEPROM
1545 if(!write_fast(0x8000+adr, 0, 0, 0)) {
1546     outputList->addItem("Failed to communicate with master, check USB signal");
1547     outputList->scrollToItem(outputList->item(outputList->count()-1));
1548     sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1549     write_log(logged);
1550     return;
1551 }
1552
1553 if(!write_fast(0x8007, 0, 0, 0)) {
1554     outputList->addItem("Failed to communicate with master, check USB signal");
1555     outputList->scrollToItem(outputList->item(outputList->count()-1));
1556     sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1557     write_log(logged);
1558     return;
1559 }
1560
1561 if(!write_fast(0x8000+eeadl, 0, 0, 0)) {
1562     outputList->addItem("Failed to communicate with master, check USB signal");
1563     outputList->scrollToItem(outputList->item(outputList->count()-1));
1564     sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1565     write_log(logged);

```

```

1561     return;
1562 }
1563 if(!write_fast(0x8000+eedr, 0, 0, 0)) {
1564     outputList->addItem("Failed to communicate with master, check USB signal");
1565     outputList->scrollToItem(outputList->item(outputList->count()-1));
1566     sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1567     write_log(logged);
1568     return;
1569 }
1570
1571 // Delay to let the module rewrite its EEPROM byte
1572 for(retval=0;retval<20;retval++) if(usb_interrupt_read(usb_handle, 0x82, buf,
        64, 0)!=64) return;
1573
1574 // Read and interpret the module's reply
1575 eedata=read_word();
1576 if((eedata>0)&&((eedata&0xFF)==eedr)) {
1577     eedata&=0xFF;
1578     switch(eeadl) {
1579     case 0x00:
1580         sprintf(buf, "Wrote EEPROM $00 (oscillator calibration byte) to new value
            %d ($%02X) in module %d ($%02X)", eedr, eedr, adr, adr);
1581         break;
1582     case 0x01:
1583         sprintf(buf, "Wrote EEPROM $01 (module address) to new value %d ($%02X) in
            module %d ($%02X)", eedr, eedr, adr, adr);
1584         break;
1585     case 0x02:
1586         sprintf(buf, "Wrote EEPROM $02 (voltage offset calibration byte) to new
            value %d ($%02X) in module %d ($%02X)", eedr, eedr, adr, adr);
1587         break;
1588     case 0x03:
1589         sprintf(buf, "Wrote EEPROM $03 (temperature offset calibration byte) to
            new value %d ($%02X) in module %d ($%02X)", eedr, eedr, adr, adr);
1590         break;
1591     default:
1592         sprintf(buf, "Wrote EEPROM $%02X to new value %d ($%02X) in module %d ($
            %02X)", eeadl, eedr, eedr, adr, adr);
1593     }
1594 } else if(eedata==--1) {
1595     sprintf(buf, "EEPROM of module %d ($%02X) failed to read", adr, adr);
1596 } else if(eedata==--2) {
1597     sprintf(buf, "Failed to communicate with master, check USB signal");
1598 } else {
1599     sprintf(buf, "Value %d ($%02X) in module %d ($%02X) failed to store in EEPROM
        $%02X - verified %d ($%02X) instead", eedr, eedr, adr, adr, eeadl, eedata,
        eedata);
1600 }
1601 outputList->addItem(buf);
1602 outputList->scrollToItem(outputList->item(outputList->count()-1));
1603 sprintf(logged, "monitor:      Notified \"%s\\\"\\n", buf);
1604 write_log(logged);
1605 }
1606
1607
1608 void Monitor::on_actionReaddressM_activated()
1609 {
1610     bool ok;
1611     int adrold, adrnew, eedata, i;
1612
1613     if(!usb_handle) return;

```

```

1614
1615     adrold=QInputDialog::getInteger(this, "Readdress Module", "Enter old module
        address:                                ", 1, 1, 120, 1, &ok);
1616     if(!ok) return;
1617
1618     adrnew=QInputDialog::getInteger(this, "Readdress Module", "Enter new module
        address:                                ", 1, 1, 255, 1, &ok);
1619     if(!ok) return;
1620
1621     if(!check_comm(adrold)) return;
1622
1623     if((adrnew==0)||((adrnew>120)&&(adrnew!=255))) {
1624         outputList->addItem("New address must be 1..120 or 255");
1625         outputList->scrollToItem(outputList->item(outputList->count()-1));
1626         sprintf(logged, "monitor:      Notified \"New address must be 1..120 or 255\\\"\\n");
1627         write_log(logged);
1628         return;
1629     }
1630
1631     for(i=0;i<nummods;i++) if(addresses[i]==adrnew) break;
1632     if(i!=nummods) {
1633         outputList->addItem("New address must not be held by any existing module");
1634         outputList->scrollToItem(outputList->item(outputList->count()-1));
1635         sprintf(logged, "monitor:      Notified \"New address must not be held by any
        existing module\\\"\\n");
1636         write_log(logged);
1637         return;
1638     }
1639
1640     // Send the EEPROM key ($92^address) to the module to allow temporary writing
1641     if(!write_fast(0x8000+adrold, 0, 0, 0)) {
1642         outputList->addItem("Failed to communicate with master, check USB signal");
1643         outputList->scrollToItem(outputList->item(outputList->count()-1));
1644         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1645         write_log(logged);
1646         return;
1647     }
1648     if(!write_fast(0x8192^adrold, 0, 0, 0)) {
1649         outputList->addItem("Failed to communicate with master, check USB signal");
1650         outputList->scrollToItem(outputList->item(outputList->count()-1));
1651         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1652         write_log(logged);
1653         return;
1654     }
1655
1656     // Write to the addressed module's EEPROM
1657     if(!write_fast(0x8000+adrold, 0, 0, 0)) {
1658         outputList->addItem("Failed to communicate with master, check USB signal");
1659         outputList->scrollToItem(outputList->item(outputList->count()-1));
1660         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1661         write_log(logged);
1662         return;
1663     }
1664     if(!write_fast(0x8007, 0, 0, 0)) {
1665         outputList->addItem("Failed to communicate with master, check USB signal");
1666         outputList->scrollToItem(outputList->item(outputList->count()-1));
1667         sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");

```

```

1668     write_log(logged);
1669     return;
1670 }
1671 if(!write_fast(0x8001, 0, 0, 0)) {
1672     outputList->addItem("Failed to communicate with master, check USB signal");
1673     outputList->scrollToItem(outputList->item(outputList->count()-1));
1674     sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1675     write_log(logged);
1676     return;
1677 }
1678 if(!write_fast(0x8000+adrnew, 0, 0, 0)) {
1679     outputList->addItem("Failed to communicate with master, check USB signal");
1680     outputList->scrollToItem(outputList->item(outputList->count()-1));
1681     sprintf(logged, "monitor:      Notified \"Failed to communicate with master,
        check USB signal\\\"\\n");
1682     write_log(logged);
1683     return;
1684 }
1685
1686 // Delay to let the module rewrite its EEPROM address byte ($01)
1687 for(i=0;i<20;i++) if(usb_interrupt_read(usb_handle, 0x82, buf, 64, 0)!=64)
1688     return;
1689
1690 // Read and interpret the module's reply
1691 eedata=read_word();
1692 if(eedata!=-1) {
1693     eedata&=0xFF;
1694     if(eedata==adrnew) {
1695         if(adrnew!=255) {
1696             firmware[0][0][adrnew]=firmware[0][0][adroid];
1697             firmware[0][1][adrnew]=firmware[0][1][adroid];
1698             firmware[1][0][adrnew]=firmware[1][0][adroid];
1699             firmware[1][1][adrnew]=firmware[1][1][adroid];
1700             firmware[0][0][adroid]=0x00;
1701             firmware[0][1][adroid]=0x00;
1702             firmware[1][0][adroid]=0x00;
1703             firmware[1][1][adroid]=0x00;
1704         }
1705         sprintf(buf, "Wrote EEPROM $01 (module address) to new value %d ($%02X) of
            previous module %d ($%02X)", adrnew, adrnew, adroid, adroid);
1706     } else {
1707         sprintf(buf, "New address %d ($%02X) of previous module %d ($%02X) failed to
            store in EEPROM $01 - verified %d ($%02X) instead", adrnew, adrnew,
            adroid, adroid, eedata, eedata);
1708     }
1709 } else {
1710     sprintf(buf, "New address %d ($%02X) of previous module %d ($%02X) failed to
        store in EEPROM $01 - no verification reply", adrnew, adrnew, adroid,
        adroid);
1711 }
1712 outputList->addItem(buf);
1713 outputList->scrollToItem(outputList->item(outputList->count()-1));
1714 sprintf(logged, "monitor:      Notified \"%s\\\"\\n", buf);
1715 write_log(logged);
1716
1717 // roll_call(FALSE);
1718 }
1719
1720
1721 void Monitor::on_actionChangePhase_activated()
1722 {

```

```

1723 }
1724
1725
1726 void Monitor::on_actionUpgradeMFW_activated()
1727 {
1728     char x;
1729     unsigned char bufsaved[96];
1730     unsigned char bigmemused[512];
1731     unsigned int i, j, intel_len, intel_addr, intel_type, intel_highaddr;
1732     unsigned long readin, bigmem[16384], numbytes[2];
1733     time_t timenow;
1734
1735     numbytes[0]=numbytes[1]=0;
1736
1737     if(!usb_handle) return;
1738
1739     // Read the old master firmware version at 0x00000100
1740     if(!issue_prog(0x00000100, 0, READ_ROM_LONG)) {
1741         outputList->addItem("Failed to read master firmware version, check connection"
1742             );
1743         outputList->scrollToItem(outputList->item(outputList->count()-1));
1744         sprintf(logged, "upgrade_MFW:  Notified \"Failed to read master firmware
1745             version, check connection\\\"\\n");
1746         write_log(logged);
1747         return;
1748     }
1749
1750     // Inform the user of the current version number
1751     sprintf(buf, "Current master firmware version is %ld.%02ld.", bufx[17], bufx
1752         [16]);
1753     sprintf(logged, "upgrade_MFW:  Notified \"%s\\\"\\n", buf);
1754     write_log(logged);
1755     QMessageBox::information(this, "Upgrade Master Firmware", buf, QMessageBox::Ok,
1756         QMessageBox::Ok);
1757
1758     // Open a dialog box for the user to choose a new firmware file
1759     QString s=QFileDialog::getOpenFileName(this, "Choose New Master Firmware", "", "
1760         Firmware Files (*.hex)");
1761     QFile findata(s);
1762     if(s.size()<2) return;
1763     sprintf(logged, "upgrade_MFW:  User selected \"%s\" for master firmware upgrade\\
1764         n", (const char *) s.toAscii());
1765     write_log(logged);
1766
1767     // Try to open the file
1768     if(!findata.open(QIODevice::ReadOnly|QIODevice::Text)) {
1769         sprintf(buf, "Cannot open %s for read", (const char *) s.toAscii());
1770         outputList->addItem(buf);
1771         outputList->scrollToItem(outputList->item(outputList->count()-1));
1772         sprintf(logged, "upgrade_MFW:  Notified \"%s\\\"\\n", buf);
1773         write_log(logged);
1774         return;
1775     }
1776
1777     timenow=time(NULL);
1778
1779     // Check the file size
1780     if(findata.size()<6000) {
1781         sprintf(buf, "%s appears to be a battery module firmware file instead", (const
1782             char *) s.toAscii());
1783         outputList->addItem(buf);
1784         outputList->scrollToItem(outputList->item(outputList->count()-1));

```

```

1778     finddata.close();
1779     sprintf(logged, "upgrade_MFW:  Notified \"%s\\n", buf);
1780     write_log(logged);
1781     goto mfw_noerror;
1782 } else if(finddata.size()<40000) {
1783     sprintf(buf, "%s does not appear to be a battery module nor master firmware
        file", (const char *) s.toAscii());
1784     outputList->addItem(buf);
1785     outputList->scrollToItem(outputList->item(outputList->count()-1));
1786     finddata.close();
1787     sprintf(logged, "upgrade_MFW:  Notified \"%s\\n", buf);
1788     write_log(logged);
1789     goto mfw_noerror;
1790 }
1791
1792 sprintf(buf, "Pre-upgrade master firmware version is %ld.%02ld", bufx[17], bufx
    [16]);
1793 outputList->addItem(buf);
1794 outputList->scrollToItem(outputList->item(outputList->count()-1));
1795
1796 // Clear out the memory space and the memory used markers
1797 for(j=0;j<16384;j++) bigmem[j]=0;
1798 for(j=0;j<512;j++) bigmemused[j]=0;
1799
1800 intel_highaddr=0x0000;
1801
1802 do {
1803     do {
1804         finddata.getChar(&x);
1805     } while((!finddata.atEnd())&&(x!=':'));
1806     if(finddata.atEnd()) break;
1807
1808     if(finddata.read(buf, 8)<8) break;
1809     buf[8]=0x00;
1810     sscanf(buf, "%2X%4X%2X", &intel_len, &intel_addr, &intel_type);
1811
1812     // We measure in words, not bytes
1813     intel_addr>>=1; intel_len>>=1;
1814
1815     // Check for the type of message/line
1816     if(intel_type==0x04) {
1817         // Read the address MSBs
1818         finddata.getChar(&x);
1819         intel_highaddr=readhex(x)<<12;
1820         finddata.getChar(&x);
1821         intel_highaddr+=readhex(x)<<8;
1822         finddata.getChar(&x);
1823         intel_highaddr+=readhex(x)<<4;
1824         finddata.getChar(&x);
1825         intel_highaddr+=readhex(x);
1826         intel_highaddr>>=1;
1827     } else if((intel_type==0x00)&&(intel_highaddr==0x0000)) {
1828         // Write out program memory (actually just store for now)
1829         for(i=intel_addr+intel_highaddr;i<intel_addr+intel_highaddr+intel_len;i+=2)
            {
1830             finddata.getChar(&x);
1831             readin=readhex(x)<<4;
1832             finddata.getChar(&x);
1833             readin+=readhex(x);
1834             finddata.getChar(&x);
1835             readin+=readhex(x)<<12;
1836             finddata.getChar(&x);

```

```

1837         readin+=readhex(x)<<8;
1838         findata.getChar(&x);
1839         readin+=readhex(x)<<20;
1840         findata.getChar(&x);
1841         readin+=readhex(x)<<16;
1842         findata.getChar(&x);
1843         readin+=readhex(x)<<28;
1844         findata.getChar(&x);
1845         readin+=readhex(x)<<24;
1846         bigmem[i>>1]=readin&0x00FFFFFF;
1847         bigmemused[i>>6]|=1;
1848         if(intel_addr<0x00007C00) {
1849             numbytes[0]+=3;
1850         } else {
1851             numbytes[1]+=3;
1852         }
1853     }
1854 }
1855 } while(intel_type!=0x01);
1856
1857 findata.close();
1858
1859 outputList->addItem("Upgrading master bootloader firmware...");
1860 outputList->scrollToItem(outputList->item(outputList->count()-1));
1861 sprintf(logged, "upgrade_MFW:  Notified \"Upgrading master bootloader firmware
1862 ...\"\\n\\n");
1863 write_log(logged);
1864
1865 for(i=2;i<512;i++) {
1866     if((bigmemused[i])&&((i<4)||i>=496)) {
1867         // Erase a row of program memory
1868         sprintf(logged, "upgrade_MFW:  Erasing locations 0x%08X-0x%08X...\\n", i<<6,
1869             (i<<6)+62);
1870         write_log(logged);
1871         if(!issue_prog(i<<6, 0, ERASE_PROG_ROW)) goto mbl_commererror;
1872
1873         // Write a row of program memory
1874         sprintf(logged, "upgrade_MFW:  Writing locations 0x%08X-0x%08X...\\n", i<<6,
1875             (i<<6)+62);
1876         write_log(logged);
1877         for(j=0;j<16;j++) {
1878             buf[j*6+32]=(bigmem[(i<<5)+(j<<1)+0]&0x000000FF);
1879             buf[j*6+33]=(bigmem[(i<<5)+(j<<1)+0]&0x0000FF00)>>8;
1880             buf[j*6+34]=(bigmem[(i<<5)+(j<<1)+0]&0x00FF0000)>>16;
1881             buf[j*6+35]=(bigmem[(i<<5)+(j<<1)+1]&0x000000FF);
1882             buf[j*6+36]=(bigmem[(i<<5)+(j<<1)+1]&0x0000FF00)>>8;
1883             buf[j*6+37]=(bigmem[(i<<5)+(j<<1)+1]&0x00FF0000)>>16;
1884         }
1885         for(j=0;j<64;j++)
1886             sprintf(bytestext+(j*3), " %02X", (unsigned char) buf[j+64]);
1887         sprintf(logged, "upgrade_MFW:  Written to USB 0x02: %s\\n", bytestext);
1888         write_log(logged);
1889         if(usb_interrupt_write(usb_handle, 0x02, buf+64, 64, 0)!=64)
1890             goto mbl_commererror;
1891         for(j=0;j<64;j++)
1892             sprintf(bytestext+(j*3), " %02X", (unsigned char) buf[j]);
1893         sprintf(logged, "upgrade_MFW:  Written to USB 0x01: %s\\n", bytestext);
1894         write_log(logged);
1895         for(j=0;j<96;j++) bufsaved[j]=(unsigned char) buf[j+32];
1896         if(usb_interrupt_write(usb_handle, 0x01, buf, 64, 0)!=64)
1897             goto mbl_commererror;
1898         if(!issue_prog(i<<6, 0, WRITE_PROG_ROW)) goto mbl_commererror;

```

```

1896
1897 // Verify the written locations, first half row
1898 sprintf(logged, "upgrade_MFW: Verifying locations 0x%08X-0x%08X...\n", i
    <<6, (i<<6)+30);
1899 write_log(logged);
1900 if(!issue_prog(i<<6, 0, READ_ROM_HROW)) goto mbl_commererror;
1901 for(j=0;j<48;j++) if(bufsaved[j]!=bufx[j+16]) goto mbl_vererror;
1902
1903 // Verify the written locations, second half row
1904 sprintf(logged, "upgrade_MFW: Verifying locations 0x%08X-0x%08X...\n", (i
    <<6)+32, (i<<6)+62);
1905 write_log(logged);
1906 if(!issue_prog((i<<6)+32, 0, READ_ROM_HROW)) goto mbl_commererror;
1907 for(j=0;j<48;j++) if(bufsaved[j+48]!=bufx[j+16]) goto mbl_vererror;
1908 }
1909 }
1910
1911 sprintf(buf, "Successfully verified %ld bootloader firmware bytes", numbytes[1])
    ;
1912 outputList->addItem(buf);
1913 outputList->scrollToItem(outputList->item(outputList->count()-1));
1914 sprintf(logged, "upgrade_MFW: Notified \"%s\"\n", buf);
1915 write_log(logged);
1916
1917 // Jump to the bootloader so the main code can be updated
1918 if(!issue_prog(0, 0, JUMP_OR_RESET)) goto mfw_commererror;
1919 if(!(bufx[1]&0x20)) {
1920     outputList->addItem("Could not enter master bootloader");
1921     outputList->scrollToItem(outputList->item(outputList->count()-1));
1922     sprintf(logged, "upgrade_MFW: Notified \"Could not enter master bootloader\"\n"
        n");
1923     write_log(logged);
1924     goto mfw_noerror;
1925 }
1926
1927 outputList->addItem("Upgrading master main firmware...");
1928 outputList->scrollToItem(outputList->item(outputList->count()-1));
1929 sprintf(logged, "upgrade_MFW: Notified \"Upgrading master main firmware...\"\n"
    );
1930 write_log(logged);
1931
1932 for(i=0;i<496;i++) {
1933     if((bigmemused[i])&&(i!=2)&&(i!=3)) {
1934         // Erase a row of program memory
1935         sprintf(logged, "upgrade_MFW: Erasing locations 0x%08X-0x%08X...\n", i<<6,
            (i<<6)+62);
1936         write_log(logged);
1937         if(!issue_prog(i<<6, 0, ERASE_PROG_ROW)) goto mfw_commererror;
1938
1939         // Write a row of program memory
1940         sprintf(logged, "upgrade_MFW: Writing locations 0x%08X-0x%08X...\n", i<<6,
            (i<<6)+62);
1941         write_log(logged);
1942         for(j=0;j<16;j++) {
1943             buf[j*6+32]=(bigmem[(i<<5)+(j<<1)+0]&0x000000FF);
1944             buf[j*6+33]=(bigmem[(i<<5)+(j<<1)+0]&0x0000FF00)>>8;
1945             buf[j*6+34]=(bigmem[(i<<5)+(j<<1)+0]&0x00FF0000)>>16;
1946             buf[j*6+35]=(bigmem[(i<<5)+(j<<1)+1]&0x000000FF);
1947             buf[j*6+36]=(bigmem[(i<<5)+(j<<1)+1]&0x0000FF00)>>8;
1948             buf[j*6+37]=(bigmem[(i<<5)+(j<<1)+1]&0x00FF0000)>>16;
1949         }
1950         for(j=0;j<64;j++)

```



```

1951     sprintf(bytestext+(j*3), " %02X", (unsigned char) buf[j+64]);
1952     sprintf(logged, "upgrade_MFW:  Written to USB 0x02: %s\n", bytestext);
1953     write_log(logged);
1954     if(usb_interrupt_write(usb_handle, 0x02, buf+64, 64, 0)!=64)
1955         goto mfw_commererror;
1956     for(j=0;j<64;j++)
1957         sprintf(bytestext+(j*3), " %02X", (unsigned char) buf[j]);
1958     sprintf(logged, "upgrade_MFW:  Written to USB 0x01: %s\n", bytestext);
1959     write_log(logged);
1960     for(j=0;j<96;j++) bufsaved[j]=(unsigned char) buf[j+32];
1961     if(usb_interrupt_write(usb_handle, 0x01, buf, 64, 0)!=64)
1962         goto mfw_commererror;
1963     if(!issue_prog(i<<6, 0, WRITE_PROG_ROW)) goto mfw_commererror;
1964
1965     // Verify the written locations, first half row
1966     sprintf(logged, "upgrade_MFW:  Verifying locations 0x%08X-0x%08X...\n", i
        <<6, (i<<6)+30);
1967     write_log(logged);
1968     if(!issue_prog(i<<6, 0, READ_ROM_HROW)) goto mfw_commererror;
1969     for(j=0;j<48;j++) if(bufsaved[j]!=bufx[j+16]) goto mfw_vererror;
1970
1971     // Verify the written locations, second half row
1972     sprintf(logged, "upgrade_MFW:  Verifying locations 0x%08X-0x%08X...\n", (i
        <<6)+32, (i<<6)+62);
1973     write_log(logged);
1974     if(!issue_prog((i<<6)+32, 0, READ_ROM_HROW)) goto mfw_commererror;
1975     for(j=0;j<48;j++) if(bufsaved[j+48]!=bufx[j+16]) goto mfw_vererror;
1976 }
1977 }
1978
1979 sprintf(buf, "Successfully verified %ld main firmware bytes", numbytes[0]);
1980 outputList->addItem(buf);
1981 outputList->scrollToItem(outputList->item(outputList->count()-1));
1982 sprintf(logged, "upgrade_MFW:  Notified \"%s\"\n", buf);
1983 write_log(logged);
1984
1985 // Reset the master processor
1986 for(i=0;i<64;i++) buf[i]=((i==15)?JUMP_OR_RESET:0x00);
1987 if(usb_interrupt_write(usb_handle, 0x01, buf, 64, 0)!=64) goto mfw_commererror;
1988 outputList->addItem("Master processor reset after successful upgrade");
1989 outputList->scrollToItem(outputList->item(outputList->count()-1));
1990 sprintf(logged, "upgrade_MFW:  Notified \"Master processor reset after
        successful upgrade\"\n");
1991 write_log(logged);
1992
1993 // Delay for at least two seconds
1994 for(i=0;i<2000;i++) if(usb_interrupt_read(usb_handle, 0x82, buf, 64, 0)!=64)
1995     goto mfw_commererror;
1996
1997 // Read the new master firmware version at 0x00000100
1998 if(!issue_prog(0x00000100, 0, READ_ROM_LONG)) {
1999     outputList->addItem("Failed to read master firmware version after upgrade");
2000     outputList->scrollToItem(outputList->item(outputList->count()-1));
2001     sprintf(logged, "upgrade_MFW:  Notified \"Failed to read master firmware
        version after upgrade\"\n");
2002     write_log(logged);
2003     goto mfw_commererror;
2004 } else {
2005     sprintf(buf, "Post-upgrade master firmware version is %ld.%02ld", bufx[17],
        bufx[16]);
2006     outputList->addItem(buf);
2007     outputList->scrollToItem(outputList->item(outputList->count()-1));

```

```

2008     sprintf(logged, "upgrade_MFW: Notified \"%s\\\"\\n", buf);
2009     write_log(logged);
2010 }
2011
2012     goto mfw_noerror;
2013
2014 mfw_commererror:
2015     outputList->addItem("Communication error while upgrading master main firmware");
2016     outputList->scrollToItem(outputList->item(outputList->count()-1));
2017     sprintf(logged, "upgrade_MFW: Notified \"Communication error while upgrading
        master main firmware\\\"\\n");
2018     write_log(logged);
2019     goto mfw_noerror;
2020
2021 mfw_vererror:
2022     outputList->addItem("Verification error while upgrading master main firmware");
2023     outputList->scrollToItem(outputList->item(outputList->count()-1));
2024     sprintf(logged, "upgrade_MFW: Notified \"Verification error while upgrading
        master main firmware\\\"\\n");
2025     write_log(logged);
2026     goto mfw_noerror;
2027
2028 mbl_commererror:
2029     outputList->addItem("Communication error while upgrading master bootloader
        firmware");
2030     outputList->scrollToItem(outputList->item(outputList->count()-1));
2031     sprintf(logged, "upgrade_MFW: Notified \"Communication error while upgrading
        master bootloader firmware\\\"\\n");
2032     write_log(logged);
2033     goto mfw_noerror;
2034
2035 mbl_vererror:
2036     outputList->addItem("Verification error while upgrading master bootloader
        firmware");
2037     outputList->scrollToItem(outputList->item(outputList->count()-1));
2038     sprintf(logged, "upgrade_MFW: Notified \"Verification error while upgrading
        master bootloader firmware\\\"\\n");
2039     write_log(logged);
2040
2041 mfw_noerror:
2042     timenow=time(NULL)-timenow;
2043     sprintf(buf, "Operation completed in %ld seconds", (unsigned long) timenow);
2044     outputList->addItem(buf);
2045     outputList->scrollToItem(outputList->item(outputList->count()-1));
2046     sprintf(logged, "upgrade_MFW: Notified \"%s\\\"\\n", buf);
2047     write_log(logged);
2048 }
2049
2050
2051 void Monitor::on_actionUpgradeBFW_activated()
2052 {
2053     char x, allmodules=0;
2054     int i, j, k, received;
2055     unsigned int flash[64][16], intel_len, intel_addr, intel_type, proginfo[16];
2056     unsigned long readin;
2057     time_t timenow;
2058
2059     if(!usb_handle) return;
2060
2061     calibrate();
2062
2063     if(roll_call(FALSE)<=0) return;

```

```

2064
2065     for(i=0;i<16;i++) proginfo[i]=0x0000;
2066
2067     for(i=0;i<120;i++) {
2068         if(addresses[i]) {
2069             if(!check_comm(addresses[i])) return;
2070             proginfo[(addresses[i]-1)>>3]|=(1<<(((addresses[i]-1)%8)*2));
2071         } else {
2072             break;
2073         }
2074     }
2075
2076     if(!check_boot()) return;
2077
2078     // Open a dialog box for the user to choose a new firmware file
2079     QString s=QFileDialog::getOpenFileName(this, "Choose New Module Firmware", "", "
        Firmware Files (*.hex)");
2080     QFile findata(s);
2081     if(s.size()<2) return;
2082     sprintf(logged, "upgrade_BFW: User selected \"%s\" for master firmware upgrade\\
        n", (const char *) s.toAscii());
2083     write_log(logged);
2084
2085     // Try to open the file
2086     if(!findata.open(QIODevice::ReadOnly|QIODevice::Text)) {
2087         sprintf(buf, "Cannot open %s for read", (const char *) s.toAscii());
2088         outputList->addItem(buf);
2089         outputList->scrollToItem(outputList->item(outputList->count()-1));
2090         sprintf(logged, "upgrade_BFW: Notified \"%s\"\\n", buf);
2091         write_log(logged);
2092         return;
2093     }
2094
2095     timenow=time(NULL);
2096
2097     // Check the file size
2098     if(findata.size()>=40000) {
2099         sprintf(buf, "%s appears to be a master firmware file instead", (const char *)
            s.toAscii());
2100         outputList->addItem(buf);
2101         outputList->scrollToItem(outputList->item(outputList->count()-1));
2102         findata.close();
2103         sprintf(logged, "upgrade_BFW: Notified \"%s\"\\n", buf);
2104         write_log(logged);
2105         goto bwf_noerror;
2106     } else if(findata.size()>=6000) {
2107         sprintf(buf, "%s does not appear to be a battery module nor master firmware
            file", (const char *) s.toAscii());
2108         outputList->addItem(buf);
2109         outputList->scrollToItem(outputList->item(outputList->count()-1));
2110         findata.close();
2111         sprintf(logged, "upgrade_BFW: Notified \"%s\"\\n", buf);
2112         write_log(logged);
2113         goto bwf_noerror;
2114     }
2115
2116     for(i=0;i<64;i++) for(j=0;j<16;j++) flash[i][j]=0x00;
2117
2118     i=0;
2119     do {
2120         do {
2121             findata.getChar(&x);

```

```

2122     } while((!findata.atEnd())&&(x!=':'));
2123     if(findata.atEnd()) break;
2124
2125     if(findata.read(buf, 8)<8) break;
2126     buf[8]=0x00;
2127     sscanf(buf, "%2X%4X%2X", &intel_len, &intel_addr, &intel_type);
2128     intel_addr>>=1; intel_len>>=1;
2129
2130     if(intel_type==0x00) {
2131         for(j=0;j<(signed int) intel_len;j++) {
2132             findata.getChar(&x);
2133             readin=readhex(x)<<4;
2134             findata.getChar(&x);
2135             readin+=readhex(x);
2136             findata.getChar(&x);
2137             readin+=readhex(x)<<12;
2138             findata.getChar(&x);
2139             readin+=readhex(x)<<8;
2140             i=j+intel_addr;
2141             flash[i>>4][i&0x000F]=readin;
2142         }
2143     }
2144 } while(intel_type!=0x01);
2145
2146 findata.close();
2147
2148 // Write the list of modules to be programmed to the master's memory
2149 for(j=0;j<32;j+=2) if(!issue_prog(0x0820+j, proginfo[j>>1], WRITE_RAM_WORD))
2150     goto bfw_commererror;
2151
2152 // Write the modules' new firmware to the master's memory
2153 for(i=0;i<56;i++)
2154     for(j=0;j<32;j+=2)
2155         if(!issue_prog(0x0840+(i<<5)+j, flash[i][j>>1], WRITE_RAM_WORD))
2156             goto bfw_commererror;
2157
2158 // Issue a reprogram battery modules command to the master
2159 if(!issue_prog(0, 0, REPROGRAM_MODS)) goto bfw_commererror;
2160
2161 // Check for individual module completions
2162 for(j=0;j<120;j++) {
2163     if((((proginfo[j>>3]>>((j%8)<<1)))&0x03)!=0x01) continue;
2164
2165     // Read master RAM location 0x0820+(j>>3)*2 to check the completed module's
2166     // status
2167     if(!issue_prog(0x0820+(j>>3)*2, 0, READ_RAM_LONG)) goto bfw_commererror;
2168
2169     // Interpret the results
2170     switch((((bufx[16]+(bufx[17]<<8))>>((j%8)*2)))&0x03) {
2171     case 0x00:
2172         for(k=0;k<nummods;k++) {
2173             if(addresses[k]==(j+1)) {
2174                 firmware[0][0][j]=flash[0x01][0x01]&0xFF;
2175                 firmware[0][1][j]=flash[0x01][0x01]>>8;
2176                 firmware[1][0][j]=flash[0x37][0x00]&0xFF;
2177                 firmware[1][1][j]=flash[0x37][0x00]>>8;
2178                 for(i=0;i<biglist->rowCount();i++) {
2179                     sscanf(module_entries[i][2]->text().toAscii(), "%d", &received);
2180                     if(received==j+1) {
2181                         sprintf(buf, "%d.%02X / %d.%02X", firmware[0][1][j], firmware
2182                             [0][0][j], firmware[1][1][j], firmware[1][0][j]);
2183                         delete module_entries[i][7];

```

```

2182         module_entries[i][7]=new QTableWidgetItem(buf);
2183         module_entries[i][7]->setTextAlignment(Qt::AlignHCenter);
2184         biglist->setItem((int) i, 7, module_entries[i][7]);
2185     }
2186 }
2187 }
2188 }
2189     sprintf(buf, "Firmware upgrade succeeded on module with address %d ($%02X)
        ", j+1, j+1);
2190     break;
2191 case 0x01:
2192     sprintf(buf, "Firmware upgrade on module with address %d ($%02X) failed -
        could not enter bootloader", j+1, j+1);
2193     allmodules++;
2194     break;
2195 case 0x02:
2196     sprintf(buf, "Firmware upgrade on module with address %d ($%02X) failed -
        no reply during verification", j+1, j+1);
2197     allmodules++;
2198     break;
2199 case 0x03:
2200     sprintf(buf, "Firmware upgrade on module with address %d ($%02X) failed -
        no reply during verification", j+1, j+1);
2201     allmodules++;
2202     break;
2203 }
2204 outputList->addItem(buf);
2205 outputList->scrollToItem(outputList->item(outputList->count()-1));
2206 sprintf(logged, "upgrade_BFW:  Notified \"%s\"\n", buf);
2207 write_log(logged);
2208 }
2209
2210 if(allmodules) {
2211     sprintf(buf, "Firmware upgrade failed on %d modules", allmodules);
2212 } else {
2213     sprintf(buf, "Firmware upgrade succeeded on all %d modules", nummods);
2214 }
2215 outputList->addItem(buf);
2216 outputList->scrollToItem(outputList->item(outputList->count()-1));
2217 sprintf(logged, "upgrade_BFW:  Notified \"%s\"\n", buf);
2218 write_log(logged);
2219 goto bfw_noerror;
2220
2221 bfw_commererror:
2222     outputList->addItem("Communication error while upgrading battery module firmware
        ");
2223     outputList->scrollToItem(outputList->item(outputList->count()-1));
2224     sprintf(logged, "upgrade_BFW:  Notified \"Communication error while upgrading
        battery module firmware\"\n");
2225     write_log(logged);
2226
2227 bfw_noerror:
2228     timenow=time(NULL)-timenow;
2229     sprintf(buf, "Operation completed in %ld seconds", (unsigned long) timenow);
2230     outputList->addItem(buf);
2231     outputList->scrollToItem(outputList->item(outputList->count()-1));
2232     sprintf(logged, "upgrade_BFW:  Notified \"%s\"\n", buf);
2233     write_log(logged);
2234     return;
2235 }
2236
2237

```

```

2238 unsigned int readhex(char inhex)
2239 {
2240     switch(inhex) {
2241         case 'A':
2242         case 'a':
2243             return 10;
2244         case 'B':
2245         case 'b':
2246             return 11;
2247         case 'C':
2248         case 'c':
2249             return 12;
2250         case 'D':
2251         case 'd':
2252             return 13;
2253         case 'E':
2254         case 'e':
2255             return 14;
2256         case 'F':
2257         case 'f':
2258             return 15;
2259         default:
2260             return inhex-'0';
2261     }
2262 }

```

APPENDIX B

Slave Printed Circuit Board Layouts

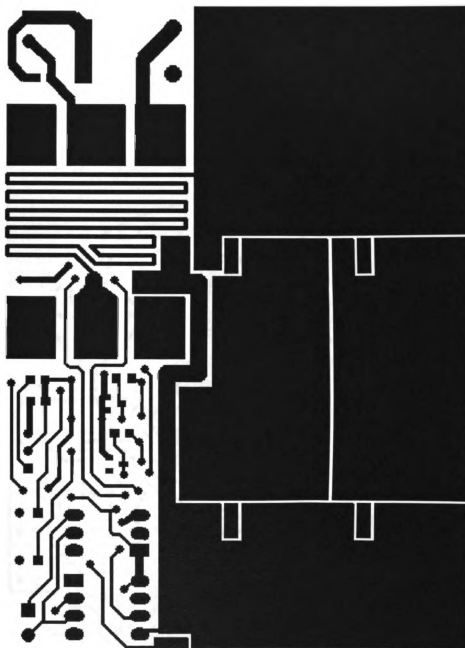


Figure B.1. Slave Design One PCB Top Layer

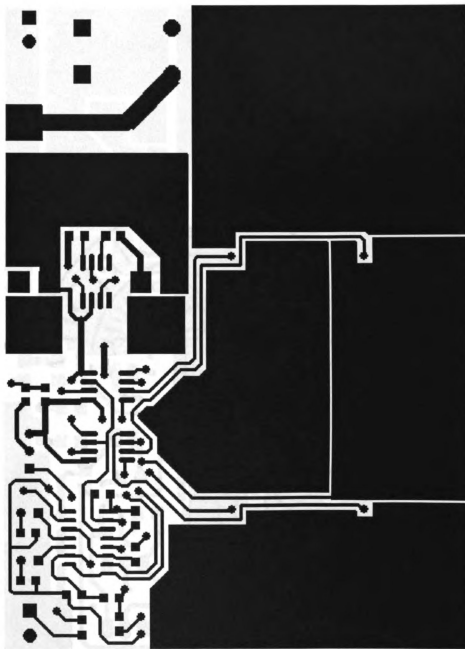


Figure B.2. Slave Design One PCB Bottom Layer

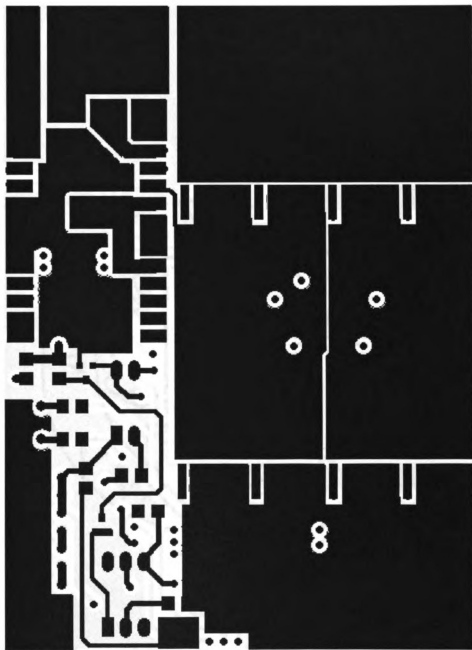


Figure B.3. Slave Design Two PCB Top Layer

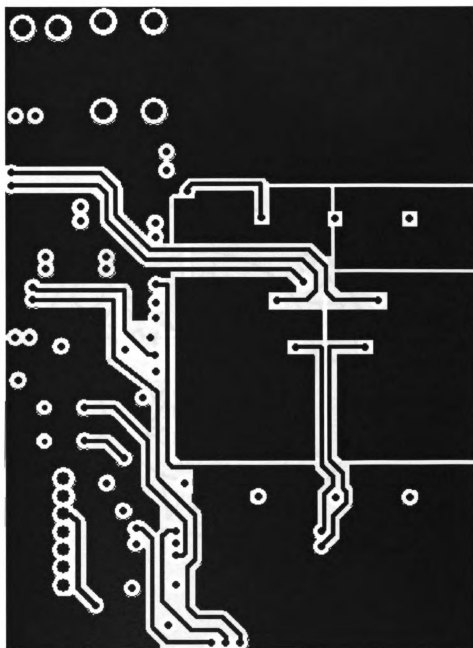


Figure B.4. Slave Design Two PCB Power Layer

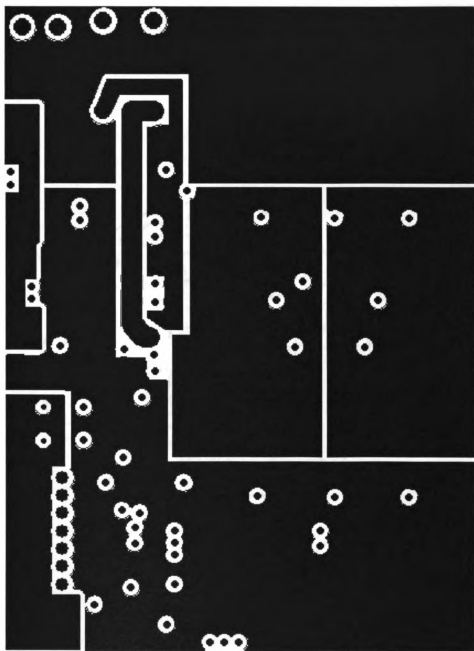


Figure B.5. Slave Design Two PCB Ground Layer

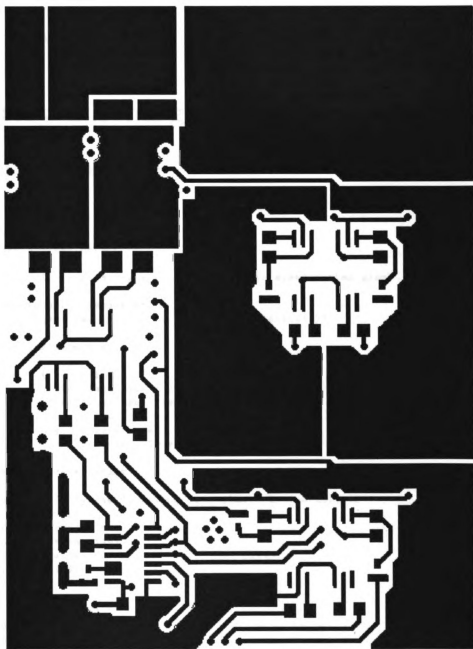


Figure B.6. Slave Design Two PCB Bottom Layer

APPENDIX C

Battery Discharge Distribution Python Script Example

```
1  #!/usr/bin/env python
2
3  # Script for generating a battery discharge distribution graph, choosebats.py
4
5  from matplotlib import rc
6  rc('font',**{'family':'serif','serif':['Palatino']})
7
8  from pylab import *
9
10 params = {'backend': 'ps',
11           'axes.labelsize': 14,
12           'text.fontsize': 14,
13           'xtick.labelsize': 14,
14           'ytick.labelsize': 14,
15           'text.usetex': True}
16
17 rcParams.update(params)
18
19 # This is 100 cycles ('howlong' calculated by 100*1000+1)
20 howlong=100001
21 batchcharges=range(16)
22 batpercents=range(16)
23 timeon=range(16)
24 baton=range(16)
25 curvolt=0
26
27 for i in range(16):
28     batchcharges[i]=0
29     batpercents[i]=0
30     timeon[i]=0
31     baton[i]=0
32
33 for i in arange(0, howlong/1000.0, 0.001):
34     thiscur=sin(2*pi*i-0.555)
35     thisvolt=7.7*sin(2*pi*i)
36
37     # Accumulate the counters for each battery that's on
38     for j in range(16):
39         if(baton[j]>0.5):
```

```

40         batchcharges[j]=batchcharges[j]+(0.001*thiscur)
41         timeon[j]=timeon[j]+0.001
42     if(baton[j]<-0.5):
43         batchcharges[j]=batchcharges[j]-(0.001*thiscur)
44         timeon[j]=timeon[j]+0.001
45
46     # Account for a hybrid battery pack
47     batpercents[0]=batchcharges[0]
48     batpercents[1]=batchcharges[1]
49     batpercents[2]=batchcharges[2]
50     batpercents[3]=batchcharges[3]
51     batpercents[4]=batchcharges[4]
52     batpercents[5]=batchcharges[5]
53     batpercents[6]=batchcharges[6]
54     batpercents[7]=batchcharges[7]
55     batpercents[8]=batchcharges[8]
56     batpercents[9]=batchcharges[9]
57     batpercents[10]=batchcharges[10]
58     batpercents[11]=batchcharges[11]
59     batpercents[12]=batchcharges[12]/2.0
60     batpercents[13]=batchcharges[13]/2.0
61     batpercents[14]=batchcharges[14]/2.0
62     batpercents[15]=batchcharges[15]/2.0
63
64     # This is for 0-180 degrees
65     if(thisvolt>=0):
66         # This is for 0-90 degrees
67         if(thisvolt>0.5+curvolt):
68             # Find any battery to compare mindisch to
69             for j in range(16):
70                 if(baton[j]==0):
71                     mindisch=j
72
73             # Find the least discharged battery to turn on first
74             for j in range(16):
75                 if(baton[j]==0):
76                     if(batpercents[j]<batpercents[mindisch]):
77                         mindisch=j
78
79             # Turn this battery positive (made sure above it's not on)
80             baton[mindisch]=1
81             curvolt=curvolt+1
82
83     # This is for 90-180 degrees
84     if(thisvolt<curvolt-0.5):
85         # Find any battery to compare maxdisch to
86         for j in range(16):
87             if(baton[j]>0.5):
88                 maxdisch=j
89
90         # Find the most discharged battery to turn off first
91         for j in range(16):
92             if(baton[j]>0.5):
93                 if(batpercents[j]>batpercents[maxdisch]):
94                     maxdisch=j
95
96         # Turn this battery off (made sure above it's on)
97         baton[maxdisch]=0
98         curvolt=curvolt-1
99
100     # This is for 180-360 degrees
101     if(thisvolt<0):

```

```

102     # This is for 180-270 degrees
103     if(thisvolt<curvolt-0.5):
104         # Find any battery to compare mindisch to
105         for j in range(16):
106             if(baton[j]==0):
107                 mindisch=j
108
109         # Find the least discharged battery
110         for j in range(16):
111             if(baton[j]==0):
112                 if(batpercents[j]<batpercents[mindisch]):
113                     mindisch=j
114
115         # Turn this battery negative (made sure above it's not on)
116         baton[mindisch]=-1
117         curvolt=curvolt-1
118
119     # This is for 270-360 degrees
120     if(thisvolt>0.5+curvolt):
121         # Find any battery to compare maxdisch to
122         for j in range(16):
123             if(baton[j]<-0.5):
124                 maxdisch=j
125
126         # Find the most discharged battery to turn off first
127         for j in range(16):
128             if(baton[j]<-0.5):
129                 if(batpercents[j]>batpercents[maxdisch]):
130                     maxdisch=j
131
132         # Turn this battery off (made sure above it's on)
133         baton[maxdisch]=0
134         curvolt=curvolt+1
135
136
137     # Plot the discharge distribution
138     subplot(111)
139     bar([0.7, 1.7, 2.7, 3.7, 4.7, 5.7, 6.7, 7.7, 8.7, 9.7, 10.7, 11.7, 12.7,
140         13.7, 14.7, 15.7], batchcharges, width=0.6)
141     title('Discharge Dist. for 100 Cycles, Vln(max)=8, PF=0.85, Hybrid Pack',
142         size='large', color='k')
143     ylabel('Charge Units', size=14, color='k')
144     xlabel('Battery Number', size=14, color='k')
145     axis([0.1, 16.9, 0, 35])
146     show()
147
148     # Plot the time-spent-on distribution
149     subplot(111)
150     bar([0.7, 1.7, 2.7, 3.7, 4.7, 5.7, 6.7, 7.7, 8.7, 9.7, 10.7, 11.7, 12.7,
151         13.7, 14.7, 15.7], timeon, width=0.6)
152     title('Time Spent On Dist. for 100 Cycles, Vln(max)=8, PF=0.85, Hybrid Pack',
153         size='large', color='k')
154     ylabel('Time Units', size=14, color='k')
155     xlabel('Battery Number', size=14, color='k')
156     axis([0.1, 16.9, 0, 50])
157     show()
158
159     # Account for a hybrid battery pack
160     timeon[12]=timeon[12]/2.0
161     timeon[13]=timeon[13]/2.0
162     timeon[14]=timeon[14]/2.0
163     timeon[15]=timeon[15]/2.0

```



```

164
165 # Initialize the statistical variables
166 mindisch=0
167 maxdisch=0
168 mintime=0
169 maxtime=0
170
171 # Calculate the statistics
172 for i in range(16):
173     # Find the most minimally discharged battery
174     if batpercents[i]<batpercents[mindisch]:
175         mindisch=i
176
177     # Find the most maximally discharged battery
178     if batpercents[i]>batpercents[maxdisch]:
179         maxdisch=i
180
181     # Find the most minimally turned-on battery
182     if timeon[i]<timeon[mintime]:
183         mintime=i
184
185     # Find the most maximally turned-on battery
186     if timeon[i]>timeon[maxtime]:
187         maxtime=i
188
189 # Print the statistical results
190 print "Charges and Time: Min, Max, % Difference"
191 print batpercents[mindisch], batpercents[maxdisch],
192       ((batpercents[maxdisch]/batpercents[mindisch])-1)*100.0
193 print timeon[mintime], timeon[maxtime],
194       ((timeon[maxtime]/timeon[mintime])-1)*100.0

```

APPENDIX D

Slave Assembly Code, Design Two

```
1 ;
2 ; Multilevel Converter Module Code
3 ; By Arthur Matteson, 3/23/2008
4 ;
5 ; Port pin assignments:
6 ; PA0 - Voltage sense A/D input
7 ; PA1 - Temperature sense A/D input
8 ; PA2 - Positive high side MOSFET gate drive output
9 ; PA3 - Negative high side MOSFET gate drive output
10 ; PA4 - Positive low side MOSFET gate drive output
11 ; PA5/PA6 - DC/DC converter gate drive outputs, complementary
12 ; PA7 - Negative low side MOSFET gate drive output
13 ; PB0/PB1 - Paralleled transmit line for 1MBd, active low
14 ; PB2 - Receive line for 1MBd, active low
15 ; PB3 - Battery voltage enable line, active low
16 ;
17 ; Register usage:
18 ; R0 - Bootloader programming data low byte
19 ; R1 - Bootloader programming data high byte
20 ; R2 - 12MHz calibration value (EEPROM $00)
21 ; R3 - 8MHz calibration value
22 ; R4 - Voltage A/D reading DC amplitude (average)
23 ; R5 - Voltage A/D reading AC amplitude (ripple) - bits 1:0 are R4's bits 9:8
24 ; R6 - Voltage A/D offset calibration value (EEPROM $02)
25 ; R7 - Temperature A/D reading DC amplitude (average)
26 ; R8 - Temperature A/D offset calibration value (EEPROM $03)
27 ; R9 -
28 ; R10 -
29 ; R11 -
30 ; R12 -
31 ; R13 -
32 ; R14 - EEPROM writing key (normally zero, $92 allows writing next time)
33 ; R15 - Diagnostic variable, status register backup in interrupt
34 ; R16 - Temporary variable in interrupt
35 ; R17 - Delay counter and temporary variable in bootloader and interrupt
36 ; R18 - First received byte in interrupt
37 ; R19 - Extra received bits and temporary variable in interrupt
38 ; R20 - Byte argument to write_byte and temporary variable in interrupt
39 ; R21 - Temporary counter in write_byte
40 ; R22 - Roll call temporary variable in interrupt
41 ; R23 - Module address (EEPROM $01)
42 ; R24 - Received byte count in interrupt
```

```

43 ; R25 - Temporary variable in mainloop
44 ; R26 (XL) - Temporary variable in mainloop
45 ; R27 (XH) - Temporary variable in mainloop
46 ; R28 (YL) -
47 ; R29 (YH) -
48 ; R30 (ZL) - IJMP's ZL in bootloader
49 ; R31 (ZH) - IJMP's ZH in bootloader
50 ;
51
52
53 .DEVICE ATTINY24
54
55 .EQU PRR=0x00
56 .EQU DIDR=0x01
57 .EQU ADCSRB=0x03
58 .EQU ADCL=0x04
59 .EQU ADCH=0x05
60 .EQU ADCSRA=0x06
61 .EQU ADMUX=0x07
62 .EQU ACSR=0x08
63 .EQU TIFR1=0x0B
64 .EQU TIMSK1=0x0C
65 .EQU USICR=0x0D
66 .EQU USISR=0x0E
67 .EQU USIDR=0x0F
68 .EQU USIBR=0x10
69 .EQU PCMSK0=0x12
70 .EQU GPIOR0=0x13
71 .EQU GPIOR1=0x14
72 .EQU GPIOR2=0x15
73 .EQU PINB=0x16
74 .EQU DDRB=0x17
75 .EQU PORTB=0x18
76 .EQU PINA=0x19
77 .EQU DDRA=0x1A
78 .EQU PORTA=0x1B
79 .EQU EECR=0x1C
80 .EQU EEDR=0x1D
81 .EQU EEARL=0x1E
82 .EQU EEARH=0x1F
83 .EQU PCMSK1=0x20
84 .EQU WDTCSR=0x21
85 .EQU TCCR1C=0x22
86 .EQU GTCCR=0x23
87 .EQU ICR1L=0x24
88 .EQU ICR1H=0x25
89 .EQU CLKPR=0x26
90 .EQU DWDR=0x27
91 .EQU OCR1BL=0x28
92 .EQU OCR1BH=0x29
93 .EQU OCR1AL=0x2A
94 .EQU OCR1AH=0x2B
95 .EQU TCNT1L=0x2C
96 .EQU TCNT1H=0x2D
97 .EQU TCCR1B=0x2E
98 .EQU TCCR1A=0x2F
99 .EQU TCCROA=0x30
100 .EQU OSCCAL=0x31
101 .EQU TCNT0=0x32
102 .EQU TCCROB=0x33
103 .EQU MCUSR=0x34
104 .EQU MCUCR=0x35

```

```

105 .EQU OCR0A=0x36
106 .EQU SPMCSR=0x37
107 .EQU TIFR0=0x38
108 .EQU TIMSK0=0x39
109 .EQU GIFR=0x3A
110 .EQU GIMSK=0x3B
111 .EQU OCR0B=0x3C
112 .EQU SPL=0x3D
113 .EQU SPH=0x3E
114 .EQU SREG=0x3F
115
116
117 .ORG 0x0000
118
119 RJMP reset
120 RJMP ext_int0
121 RJMP reset
122 RJMP reset
123 RJMP reset
124 RJMP reset
125 RJMP reset
126 RJMP reset
127 RJMP reset
128 RJMP reset
129 RJMP reset
130 RJMP reset
131 RJMP reset
132 RJMP reset
133 RJMP reset
134 RJMP reset
135 RJMP reset
136 .DW 0x0200 ; The main code firmware version (2.00)
137
138 reset:
139 ; Initialize the stack pointer address
140 LDI R26,0xDE
141 OUT SPL,R26
142 LDI R26,0x00
143 OUT SPH,R26
144
145 ; Configure the watchdog timer for one second (execute a timed sequence)
146 LDI R26,0x9E
147 OUT WDTCR,R26
148 LDI R26,0x8E
149 OUT WDTCR,R26
150
151 ; Set up the output ports and INTO trigger
152 LDI R26,0x40
153 OUT MCUCR,R26
154 LDI R26,0x00
155 OUT PORTA,R26
156 LDI R26,0xFC
157 OUT DDRA,R26
158 LDI R26,0x0B
159 OUT PORTB,R26
160 LDI R26,0x0B
161 OUT DDRB,R26
162
163 ; Set up the pin change interrupt and clear the flag for safety
164 LDI R26,0x40
165 OUT GIMSK,R26
166 LDI R26,0x40

```

```

167     OUT GIFR,R26
168
169     ; Set up the A/D converter
170     LDI R26,0x00
171     OUT ADCSRB,R26
172     LDI R26,0xFF
173     OUT DIDR,R26
174     LDI R26,0x96
175     OUT ADCSRA,R26
176
177     ; Set up the timer/counter
178     LDI R26,0x00
179     OUT TCCR1A,R26
180     LDI R26,0x11
181     OUT TCCR1B,R26
182     LDI R26,0x00
183     OUT TCNT1H,R26
184     OUT TCNT1L,R26
185     OUT ICR1H,R26
186     LDI R26,0x15
187     OUT ICR1L,R26
188     LDI R26,0x00
189     OUT OCR1AH,R26
190     LDI R26,0x0C
191     OUT OCR1AL,R26
192     LDI R26,0x00
193     OUT OCR1BH,R26
194     LDI R26,0x09
195     OUT OCR1BL,R26
196
197     ; Check the calibration value in EEPROM
198     LDI R26,0x00
199     OUT EEARL,R26
200     OUT EEARH,R26
201     SBI EECR,0
202     IN R26,EEDR
203     SBR R26,0xC0
204     MOV R2,R26
205
206     ; Check the module address in EEPROM
207     LDI R26,0x01
208     OUT EEARL,R26
209     SBI EECR,0
210     IN R23,EEDR
211
212     ; Check the voltage calibration byte in EEPROM
213     LDI R26,0x02
214     OUT EEARL,R26
215     SBI EECR,0
216     IN R6,EEDR
217
218     ; Check the temperature calibration byte in EEPROM
219     LDI R26,0x03
220     OUT EEARL,R26
221     SBI EECR,0
222     IN R8,EEDR
223
224     ; Change the initial oscillator frequency to 12MHz
225     IN R3,OSCCAL
226     RCALL speed_up
227
228     CLR R14                ; Clear the EEPROM writing key ($92 allows writing)

```

```

229
230     SBI PORTA,3           ; Turn the red LED on
231     RCALL diag_blink
232     SBI PORTA,2           ; Turn the green LED on
233     RCALL diag_blink
234
235     LDI R26,0x04           ; Set up the 8-bit timer with a divide-by-256 prescaler
236     OUT TCCR0B,R26
237     CLR R26
238     OUT TCNT0,R26
239
240     ; Enable interrupts
241     CLR R20
242     SEI
243
244     mainloop:
245         WDR                 ; Reset the watchdog timer periodically
246
247         ; Read the voltage into R4/R5 (DC/AC) calibrated by R6
248         LDI R26,0x00        ; Select ADC0 (PA0)
249         OUT ADMUX,R26
250         SBI ADCSRA,6        ; Start the conversion
251         SBIS ADCSRA,4        ; Test for a completed conversion
252         RJMP PC-1
253         SBI ADCSRA,4        ; Clear the interrupt flag by writing a one
254         SBI ADCSRA,6        ; Start a conversion again after the reference changed
255         SBIS ADCSRA,4        ; Test for a completed conversion
256         RJMP PC-1
257         SBI ADCSRA,4        ; Clear the interrupt flag by writing a one
258         IN R26,ADCL         ; Read in the low byte of the voltage A/D result
259         IN R27,ADCH         ; Read in the high byte (two bits) of the A/D result
260         MOV R25,R6          ; Buffer this value so it doesn't change in the middle
261         TST R25             ; Is the offset positive or negative?
262         BRMI voltad_offmi
263         ADD R26,R25         ; Add the offset calibration byte onto the low byte
264         BRCC voltad_done
265         INC R27
266         RJMP voltad_done
267     voltad_offmi:
268         ADD R26,R25         ; Add the offset calibration byte onto the low byte
269         BRCS voltad_done
270         DEC R27
271     voltad_done:
272         MOVW R4,R26         ; Simultaneously move R27:R26 to R5:R4 (interrupt care)
273
274         ; Read the temperature into R7 calibrated by R8
275         ; LDI R26,0x04        ; Select ADC4 (PA4)
276         LDI R26,0xA2        ; Select ADC8 (temperature)
277         OUT ADMUX,R26
278         SBI ADCSRA,6        ; Start the conversion
279         SBIS ADCSRA,4        ; Test for a completed conversion
280         RJMP PC-1
281         SBI ADCSRA,4        ; Clear the interrupt flag by writing a one
282         SBI ADCSRA,6        ; Start a conversion again after the reference changed
283         SBIS ADCSRA,4        ; Test for a completed conversion
284         RJMP PC-1
285         SBI ADCSRA,4        ; Clear the interrupt flag by writing a one
286         IN R26,ADCL         ; Read in the low byte of the voltage A/D result
287         IN R27,ADCH         ; Read in the high byte (two bits) of the A/D result
288         ADD R26,R8          ; Add the offset calibration byte onto the low byte
289         MOV R7,R26          ; Save the result in R7
290

```

```

291     RJMP mainloop
292
293     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
294
295     diag_blink:                ; Blink quickly
296         LDI R19,0x30           ; Delay about one-tenth of a second
297         RCALL diag_short
298         CBI PORTA,2           ; Turn the green LED off
299         CBI PORTA,3           ; Turn the red LED off
300         LDI R19,0x30           ; Delay about one-tenth of a second
301         RCALL diag_short
302         RET
303
304
305     diag_dash:                 ; Delay about one and a half seconds
306         RCALL diag_dot
307         RCALL diag_dot
308     diag_dot:                  ; Delay about one half second
309         LDI R19,0xB4           ; Loop 180 times
310     diag_short:
311         LDI R18,0xFA           ; Loop 250 times
312         LDI R17,0x78           ; Delay 120 cycles plus four more
313         RCALL delay_one
314         DEC R18                ; Subtract one from the remaining loop count
315         BRNE PC-3              ; Repeat the loop if the counter is not zero yet
316         LDI R17,0x7A           ; Delay 122 cycles plus one more
317         RCALL delay_one
318         LDI R17,0x7A           ; Delay 122 cycles plus two more
319         RCALL delay_one
320         WDR                    ; Reset the watchdog timer periodically
321         DEC R19                ; Subtract one from the remaining loop count
322         BRNE PC-11             ; Repeat the loop if the counter is not zero yet
323         RET
324
325     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
326
327     ext_int0:                   ; Handle a low-level trigger on INTO (PB2)
328         IN R15,SREG            ; Save the status register
329         RJMP PC+1              ; Ten delays is optimal for +/-4% clock
330         RJMP PC+1              ; Thirteen delays is maximal, six minimal at 12MHz
331         RJMP PC+1
332         RJMP PC+1
333         WDR
334         NOP
335
336     ; Read in seven bits to R18, 1MBd; module address or broadcast command
337         LDI R18,0x40           ; Load seven bits
338         SBIC PINB,2            ; Test the value of PB2
339         SBR R18,0x80           ; Set bit seven if on
340         NOP                    ; Each bit is twelve clock cycles
341         RJMP PC+1
342         RJMP PC+1
343         RJMP PC+1
344         LSR R18                ; Shift right (place a zero in the top bit)
345         BRCC PC-7              ; Locate the initially placed bit
346
347     ; Read in two bits to R19, 1MBd; '00'=data, '01'=pos., '10'=neg., '11'=off
348         CLR R19                ; Clear the follow-up bits first
349         SBIC PINB,2            ; Test the value of PB2
350         SBR R19,0x04           ; Set bit two if on
351         TST R18                ; Was the address zero?
352         BRNE pi_skipb11        ; If so, this is actually a 125kBd byte of $AC probably

```

```

353     TST R19
354     BRNE pi_skipbl2      ; If this was a dummy $180 command don't enter bootloader
355     LDI R16,0x00         ; Turn the DC/DC off
356     OUT TCCR1A,R16
357     OUT PORTA,R16        ; Turn everything off
358     LDI R16,0x0B         ; Disable the battery voltage to the DC/DC converter
359     OUT PORTB,R16
360     LDI R16,0x40         ; Clear the interrupt flag now
361     OUT GIFR,R16
362     RJMP bootloader      ; Jump to the bootloader now
363 pi_skipbl1:
364     NOP                  ; Each bit is twelve clock cycles
365     NOP
366 pi_skipbl2:
367     NOP
368     BST R19,2            ; Copy the first received bit to location four
369     BLD R19,4
370     LDI R22,0x01         ; Initialize the roll call address early
371     SER R24              ; Make the input byte counter $FF (-1)
372     SBIC PINB,2          ; Test the value of PB2
373     SBR R19,0x08         ; Set bit three if on
374     BST R19,3            ; Copy the second received bit to location seven
375     BLD R19,7
376
377     CPI R18,0x7F         ; Is this a synchronize clock speed command byte one?
378     BREQ pi_nextbytepre ; If so, make sure the next byte is also 0x7F
379
380 ; PA2 - Positive high side MOSFET gate drive output
381 ; PA3 - Negative high side MOSFET gate drive output
382 ; PA4 - Positive low side MOSFET gate drive output
383 ; PA7 - Negative low side MOSFET gate drive output
384
385 ; Test for a single-module on/off command, and check to see the address
386     TST R19              ; Message follows? (See if R19='00')
387     BREQ pi_nextbytepre ; Receive the next byte in that case
388     CP R18,R23           ; Is this module being addressed? If not, skip next
389     BRNE pi_testmore    ; Never will R19='00' get here
390     LDI R16,0x00         ; Turn off everything and wait a dead time
391     OUT PORTA,R16
392     LDI R16,0x0C         ; Change the polarity of the pos. HS and neg. HS
393     EOR R19,R16
394     MOV R20,R4           ; Use an 800ns dead time (do setup for pi_sendvolts)
395     LSR R20              ; Disregard the battery voltage low byte bit zero
396     CPI R19,0x90         ; Was the command '11'? If so, echo the battery voltage
397     BRNE PC+2
398     RJMP pi_sendvolts
399     NOP
400     OUT PORTA,R19        ; Output this to the port after the dead time
401     RJMP pi_done         ; If it got here, that's all we need to do
402
403 ; Test for single-byte broadcast commands or ignore the single-module command
404 pi_testmore:
405     LDI R16,0x87         ; Test for broadcast commands (values over 120)
406     ADD R16,R18
407     BRCC pi_done         ; If no overflow, this was a single-module command
408
409     CPI R18,0x79         ; Is it all sleep (all off)?
410     BREQ pi_sleep
411     CPI R18,0x7A         ; Is it all wake (opto. on, DC/DC off, MOSFETs off)?
412     BREQ pi_wake
413     CPI R18,0x7B         ; Is it all DC/DC converter on, MOSFETs off?
414     BREQ pi_dcdcfetsoff

```



```

415   CPI R18,0x7C           ; Is it all DC/DC converter on, MOSFETs on?
416   BREQ pi_dcdcfetson
417   CPI R18,0x7D           ; Is it all blink?
418   BREQ pi_blink
419   RJMP pi_done
420
421 pi_blink:
422   LDI R16,0x00           ; Turn the DC/DC off
423   OUT TCCR1A,R16
424   OUT PORTA,R16          ; Turn everything off
425   LDI R16,0x0B           ; Disable the battery voltage to the DC/DC converter
426   OUT PORTB,R16
427   LDI R19,0x30           ; Delay about one-tenth of a second
428   RCALL diag_short
429   LDI R17,0x04           ; Turn the green LED on and the rest off
430   OUT PORTA,R17
431   RCALL diag_dash        ; Wait about three seconds
432   RCALL diag_dash
433   RJMP pi_off            ; Turn everything off
434 pi_dcdcfetsoff:
435   LDI R16,0xE0           ; Turn the DC/DC on
436   OUT TCCR1A,R16
437   RJMP pi_wakex
438 pi_dcdcfetson:
439   LDI R16,0xE0           ; Turn the DC/DC on
440   OUT TCCR1A,R16
441   LDI R16,0x90           ; Turn the low side MOSFETs on
442   OUT PORTA,R16
443   LDI R16,0x03           ; Enable the battery voltage to the DC/DC converter
444   OUT PORTB,R16
445   RJMP pi_done
446 pi_wake:
447   LDI R16,0x00           ; Turn the DC/DC off
448   OUT TCCR1A,R16
449 pi_wakex:
450   LDI R16,0x03           ; Enable the battery voltage to the DC/DC converter
451   OUT PORTB,R16
452   RJMP pi_off
453
454   ; Wait for another start bit
455 pi_nextbytepre:
456   RJMP PC+1              ; Delay past the end of the last bit
457   RJMP PC+1
458   RJMP PC+1
459   RJMP PC+1
460   RJMP pi_nextbyte
461
462 pi_sleep:
463   LDI R16,0x00           ; Turn the DC/DC off
464   OUT TCCR1A,R16
465   LDI R16,0x0B           ; Disable the battery voltage to the DC/DC converter
466   OUT PORTB,R16
467 pi_off:
468   LDI R16,0x00           ; Turn everything off
469   OUT PORTA,R16
470 pi_done:
471   LDI R16,0x40           ; Put here for branch distance
472   OUT GIFR,R16           ; Clear the interrupt flag
473   CLR R14                ; Clear the EEPROM writing key
474   OUT SREG,R15           ; Recall the status register
475   RETI
476

```

```

477 pi_nextbyte:
478     SBIC PINB,2           ; Test the value of PB2 (wait for a low)
479     RJMP PC-1
480     RJMP PC+1             ; Thirteen delays works well
481     RJMP PC+1             ; Seventeen cycles maximal, ten minimal at 12MHz
482     RJMP PC+1
483     RJMP PC+1
484     RJMP PC+1
485     NOP
486     RJMP pi_readmore     ; Skip the next function
487
488 pi_writeeprom:
489     LDI R17,0x92           ; Check to see if the EEPROM key is correct
490     CP R14,R17
491     BRNE pi_done          ; If not, exit immediately
492     RCALL speed_down       ; Change the oscillator frequency down to 8MHz
493     LDI R16,0x04
494     OUT EECR,R16
495     SBI EECR,1             ; Set the EEPE bit to enable programming
496     SBIC EECR,1           ; Delay until the possible program operation completes
497     RJMP PC-1
498     SBI EECR,0             ; Read the value right back in to verify it
499     IN R20,EECR
500     RCALL speed_up        ; Change the oscillator frequency back up to 12MHz
501     RCALL write_byte      ; Echo the written value
502     RJMP pi_off
503
504 ; Read in eight bits to R19, 1MBd; either data or a single-module command
505 pi_readmore:
506     LDI R19,0x80           ; Load eight bits
507     LSR R19                ; Shift right (place a zero in the top bit)
508     SBIC PINB,2           ; Test the value of PB2
509     SBR R19,0x80          ; Set bit seven if on
510     NOP                    ; These make the loop 12 cycles total
511     RJMP PC+1
512     RJMP PC+1
513     RJMP PC+1
514     BRCC PC-7              ; Locate the initially placed bit
515     INC R24                ; Keep track of the number of bytes received
516     IN R16,PINB           ; Save the value of the last bit temporarily
517
518     CPI R18,0x7F           ; Does the first byte suggest a synchronize command?
519     BRNE PC+6
520     MOV R20,R19            ; Test for a synchronize clock speed command byte two
521     ANDI R20,0x7F
522     CPI R20,0x7F           ; Is this a synchronize clock speed command byte two?
523     BRNE PC+2              ; If not, check for a roll call command
524     RJMP pi_sync
525
526 ; Handle the module roll call command
527     CPI R18,0x7E           ; Is it roll call?
528     BRNE pi_check_el
529     TST R19                ; Is it over? (Roll call sent a $00)
530     BRNE PC+2
531     RJMP pi_done
532     WDR                    ; Reset the watchdog timer during the roll call process
533     CP R19,R23              ; Is the byte the same address as this module?
534     BRNE PC+5              ; If not, don't reply and let another module reply
535     MOV R20,R23            ; Reply with this module's address
536     RCALL write_byte
537     CPI R19,0xFF           ; Is this a signal for new module on the string?
538     BREQ PC+3              ; If so, set this module's address in line with the rest

```

```

539     MOV R22,R19                ; Store the previous address in case it needs assignment
540     RJMP pi_nextbyte
541     CPI R22,0x79                ; Is the address above 120?
542     BRCS PC+2                  ; If so, don't set it
543     RJMP pi_done
544     TST R22                    ; Also don't write a zero address
545     BRNE PC+2
546     RJMP pi_done
547     MOV R23,R22
548     LDI R16,0x01                ; Write the new module address to EEPROM $01
549     OUT EEARL,R16
550     OUT EEDR,R23                ; R23 contains the new module address
551     RJMP pi_wroteeprom
552
553 pi_check_el:
554 ; Handle a single-module, multi-byte command
555     SBRS R16,2                  ; Was the ninth bit set? If so, this is an EEPROM key
556     RJMP pi_noteepromkey
557     CP R18,R23                  ; Is this module being addressed? If not, take no action
558     BREQ PC+2
559     RJMP pi_done                ; This will automatically clear the EEPROM writing key
560     EOR R19,R23                ; XOR this with the module address just to be unique
561     MOV R14,R19                 ; Save this EEPROM key whether it's valid or not ($92)
562     LDI R16,0x40                ; Clear the interrupt flag
563     OUT GIFR,R16
564     OUT SREG,R15                ; Recall the status register
565     RETI
566
567 pi_noteepromkey:
568     TST R24                    ; Is this the first byte?
569     BREQ PC+2
570     RJMP pi_notfirst
571     MOV R17,R19                ; Save this command byte for later comparison
572     SBRC R17,6                  ; Is it perform eight PWM cycles (22kHz)?
573     RJMP pi_pwm
574     TST R17                    ; Is it echo byte (following)?
575     BRNE PC+2
576     RJMP pi_nextbyte
577     CPI R17,0x06                ; Is it read EEPROM?
578     BRNE PC+2
579     RJMP pi_nextbyte
580     CPI R17,0x07                ; Is it write EEPROM?
581     BRNE PC+2
582     RJMP pi_nextbyte
583     CPI R17,0x0C                ; Is it erase flash?
584     BRNE PC+2
585     RJMP pi_nextbyte
586     CPI R17,0x0D                ; Is it fill flash buffer?
587     BRNE PC+2
588     RJMP pi_nextbyte
589     CPI R17,0x0E                ; Is it write flash?
590     BRNE PC+2
591     RJMP pi_nextbyte
592     CPI R17,0x0F                ; Is it verify flash?
593     BRNE PC+2
594     RJMP pi_nextbyte
595     CP R18,R23                  ; Is this module being addressed? If not, take no action
596     BREQ PC+2
597     RJMP pi_done
598     CPI R17,0x01                ; Is it sleep (all off)?
599     BRNE PC+2
600     RJMP pi_sleep

```

```

601 CPI R17,0x02          ; Is it wake (opto. on, DC/DC off, MOSFETs off)?
602 BRNE PC+2
603 RJMP pi_wake
604 CPI R17,0x03          ; Is it DC/DC converter on, MOSFETs off?
605 BRNE PC+2
606 RJMP pi_dcdcfetsoff
607 CPI R17,0x04          ; Is it DC/DC converter on, MOSFETs on?
608 BRNE PC+2
609 RJMP pi_dcdcfetson
610 CPI R17,0x05          ; Is it blink?
611 BRNE PC+2
612 RJMP pi_blink
613 CPI R17,0x08          ; Is it recall DC battery voltage A/D reading?
614 BRNE pi_notrecallvdc
615 MOV R20,R4
616 RCALL write_byte      ; Transmit the data at 1MBd
617 RJMP pi_done
618 pi_notrecallvdc:
619 CPI R17,0x09          ; Is it recall AC battery voltage A/D reading?
620 BRNE pi_notrecallvac
621 MOV R20,R5
622 RCALL write_byte      ; Transmit the data at 1MBd
623 RJMP pi_done
624 pi_notrecallvac:
625 CPI R17,0x0A          ; Is it recall local temperature A/D reading?
626 BRNE pi_notrecallloctemp
627 MOV R20,R7
628 RCALL write_byte      ; Transmit the data at 1MBd
629 RJMP pi_done
630 pi_notrecallloctemp:
631 CPI R17,0x0B          ; Is it recall remote temperature A/D reading?
632 BRNE pi_notrecallremtemp
633 ; MOV R20,R7
634 ; RCALL write_byte      ; Transmit the data at 1MBd
635 RJMP pi_done
636 pi_notrecallremtemp:
637 RJMP pi_done
638
639 pi_notfirst:
640 TST R17                ; Is it echo byte (in R19, XOR with module address)?
641 BRNE pi_nf_notecho
642 CP R18,R23             ; Is this module being addressed? If not, take no action
643 BREQ PC+2
644 RJMP pi_done
645 MOV R20,R19            ; Echo the received byte to test communications
646 EOR R20,R23            ; XOR this with the module address just to be unique
647 RCALL write_byte
648 RJMP pi_done
649
650 pi_nf_notecho:
651 CPI R17,0x06          ; Is it read EEPROM?
652 BRNE pi_nf_notreadeprom
653 CP R18,R23             ; Is this module being addressed? If not, take no action
654 BREQ PC+2
655 RJMP pi_done
656 OUT EEARL,R19          ; Read and send out the data byte at the given argument
657 SBI EECR,0
658 IN R20,EEDR
659 RCALL write_byte
660 RJMP pi_done
661
662 pi_nf_notreadeprom:

```

```

663 CPI R17,0x07          ; Is it write EEPROM?
664 BRNE pi_nf_notwriteeprom
665 MOV R16,R24
666 CPI R16,0x01          ; If this is the first argument byte, save it
667 BRNE PC+3
668 OUT EEARL,R19          ; It can be written to the EEARL register right now
669 RJMP pi_nextbyte
670 CP R18,R23             ; Is this module being addressed? If not, take no action
671 BREQ PC+2
672 RJMP pi_done
673 IN R16,EEARL           ; Recall the address to write in case it was $00..$03
674 TST R16                ; Was the address $00? If so, save the value in R2
675 BRNE PC+3
676 MOV R2,R19             ; This is the new module calibration byte
677 RJMP pi_nf_ee_done     ; (It will be applied soon in the speed_up function)
678 DEC R16                ; Was the address $01? If so, save the value in R23
679 BRNE PC+9
680 CPI R19,0x79           ; Is the address above 120?
681 BRCS PC+2              ; If so, don't set it
682 RJMP pi_done
683 TST R18                ; Also don't write a zero address
684 BRNE PC+2
685 RJMP pi_done
686 MOV R23,R19            ; This is the new module address
687 RJMP pi_nf_ee_done
688 DEC R16                ; Was the address $02? If so, save the value in R6
689 BRNE PC+3
690 MOV R6,R19             ; This is the new voltage offset calibration byte
691 RJMP pi_nf_ee_done
692 DEC R16                ; Was the address $03? If so, save the value in R8
693 BRNE pi_nf_ee_done
694 MOV R8,R19             ; This is the new temperature offset calibration byte
695 pi_nf_ee_done:
696 OUT EEDR,R19           ; R19 contains the EEPROM byte to be written
697 RJMP pi_writeeprom
698
699 pi_nf_notwriteeprom:
700 CPI R17,0x0C           ; Is it erase flash?
701 BRNE pi_nf_noteraseflash
702 CP R18,R23             ; Is this module being addressed? If not, take no action
703 BREQ PC+2
704 RJMP pi_done
705 LDI R17,0x92           ; Check to see if the EEPROM key is correct
706 CP R14,R17
707 BREQ PC+2
708 RJMP pi_done          ; If not, exit immediately
709 PUSH R30
710 PUSH R31
711 LDI R31,0x07           ; Access memory only from 0x700 to 0x7FF (0x380 to 0x3FF)
712 MOV R30,R19
713 RCALL speed_down       ; Change the oscillator frequency down to 8MHz for SPM
714 LDI R16,0x03
715 OUT SPMCSR,R16         ; Write $03 to SPMCSR and execute SPM (erase flash page)
716 ; SPM                  ; Store (or modify) program memory
717 RCALL speed_up         ; Change the oscillator frequency back up to 12MHz
718 POP R31
719 POP R30
720 RJMP pi_done
721
722 pi_nf_noteraseflash:
723 CPI R17,0x0D           ; Is it fill flash buffer?
724 BRNE pi_nf_notfillflash

```

```

725     MOV R16,R24
726     CPI R16,0x01          ; If this is the first argument byte, save it
727     BRNE PC+3
728     MOV R0,R19            ; It can be written to the R0 register now
729     RJMP pi_nextbyte
730     CPI R16,0x02          ; If this is the second argument byte, save it
731     BRNE PC+3
732     MOV R1,R19            ; It can be written to the R1 register now
733     RJMP pi_nextbyte
734     CP R18,R23            ; Is this module being addressed? If not, take no action
735     BREQ PC+2
736     RJMP pi_done
737     LDI R17,0x92          ; Check to see if the EEPROM key is correct
738     CP R14,R17
739     BREQ PC+2
740     RJMP pi_done          ; If not, exit immediately
741     PUSH R30
742     PUSH R31
743     LDI R31,0x07          ; Access memory only from 0x700 to 0x7FF (0x380 to 0x3FF)
744     MOV R30,R19
745     RCALL speed_down      ; Change the oscillator frequency down to 8MHz for SPM
746     LDI R16,0x01
747     OUT SPMCSR,R16        ; Write $01 to SPMCSR and execute SPM (fill flash buffer)
748 ; SPM                    ; Store (or modify) program memory
749     RCALL speed_up        ; Change the oscillator frequency back up to 12MHz
750     POP R31
751     POP R30
752     RJMP pi_done
753
754 pi_nf_notfillflash:
755     CPI R17,0x0E          ; Is it write flash?
756     BRNE pi_nf_notwriteflash
757     CP R18,R23            ; Is this module being addressed? If not, take no action
758     BREQ PC+2
759     RJMP pi_done
760     LDI R17,0x92          ; Check to see if the EEPROM key is correct
761     CP R14,R17
762     BREQ PC+2
763     RJMP pi_done          ; If not, exit immediately
764     PUSH R30
765     PUSH R31
766     LDI R31,0x07          ; Access memory only from 0x700 to 0x7FF (0x380 to 0x3FF)
767     MOV R30,R19
768     RCALL speed_down      ; Change the oscillator frequency down to 8MHz for SPM
769     LDI R16,0x05
770     OUT SPMCSR,R16        ; Write $05 to SPMCSR and execute SPM (write flash page)
771 ; SPM                    ; Store (or modify) program memory
772     RCALL speed_up        ; Change the oscillator frequency back up to 12MHz
773     POP R31
774     POP R30
775     RJMP pi_done
776
777 pi_nf_notwriteflash:
778     CPI R17,0x0F          ; Is it verify flash?
779     BRNE pi_nf_notreadflash
780     CP R18,R23            ; Is this module being addressed? If not, take no action
781     BREQ PC+2
782     RJMP pi_done
783     PUSH R30
784     PUSH R31
785     LDI R31,0x07          ; Access memory only from 0x700 to 0x7FF (0x380 to 0x3FF)
786     MOV R30,R19

```

```

787     LPM R20,Z           ; Recall this program data byte to send off
788     POP R31
789     POP R30
790     RCALL write_byte
791     RJMP pi_done
792
793 pi_nf_notreadflash:
794     RJMP pi_done
795
796 pi_sendvolts:
797     OUT PORTA,R19        ; Output this to the port after the dead time
798     BST R5,0             ; Copy the volt. high byte bit zero to R20 location seven
799     BLD R20,7            ; Continue what was started above
800     RCALL write_byte     ; Transmit the data at 1MBd (send nine bits, also R5<1>)
801     RJMP pi_done        ; If it got here, that's all we need to do
802
803 ; PA2 - Positive high side MOSFET gate drive output
804 ; PA3 - Negative high side MOSFET gate drive output
805 ; PA4 - Positive low side MOSFET gate drive output
806 ; PA7 - Negative low side MOSFET gate drive output
807
808 ; Handle the single-module eight-PWM command (0x00=off, 0x01-0x1F=duty cycle)
809 pi_pwm:
810     CP R18,R23           ; Is this module being addressed? If not, take no action
811     BREQ PC+2
812     RJMP pi_done
813     LDI R18,0x08         ; Loop for eight PWM cycles
814     LDI R24,0x00         ; This is the value for PORTA to occupy during dead times
815     LDI R20,0x90         ; Turn off the high-sides when the time is appropriate
816     LDI R19,0x18         ; Turn on neg. high side, pos. low side if R17<5>=0
817     SBRC R17,5           ; Find the polarity of the PWM (0=negative, 1=positive)
818     LDI R19,0x84         ; Turn on pos. high side, neg. low side if R17<5>=1
819     ANDI R17,0x1F        ; Mask off the highest three bits to find the duty cycle
820     BREQ pi_pwmmonfull   ; If the duty cycle is zero, turn on for the whole time
821     NOP                 ; Equalize delays with pi_pwmmonfull
822
823 pi_pwmloop:
824     OUT PORTA,R24        ; Output the dead time state to the port
825     RJMP PC+1           ; Use a dead time of 800ns
826     RJMP PC+1
827     RJMP PC+1
828     RJMP PC+1
829     OUT PORTA,R19        ; Output the commanded on state to the port
830     ANDI R17,0x1F        ; Mask off the highest three bits to find the duty cycle
831     LDI R16,0x1F        ; Loop 31 times, once for each duty cycle possibility
832     NOP                 ; This makes a duty cycle of one, 16 cycles exactly
833
834 pi_pwmrepeat:
835     NOP                 ; Delay one cycle for a total of 16
836     DEC R17             ; Decrement the duty cycle to see if it is zero
837     BRNE pi_pwmnohit    ; If not, don't change the port value
838     OUT PORTA,R24        ; Output the dead time state to the port
839     RJMP PC+1           ; Use a dead time of 800ns
840     RJMP PC+1
841     RJMP PC+1
842     RJMP PC+1
843     OUT PORTA,R20        ; Output the commanded off state to the port
844     DEC R16
845     BRNE pi_pwmrepeat   ; Loop 31 times for a total of 495 cycles
846     DEC R17             ; Make the duty cycle counter roll over
847     DEC R18
848     BRNE pi_pwmloop     ; Repeat the same operation eight times (4095 cycles total)

```

```

849     RJMP pi_pwmdone
850
851 pi_pwmnohit:
852     RJMP PC+1           ; Delay nine cycles for a total of 16
853     RJMP PC+1
854     RJMP PC+1
855     RJMP PC+1
856     NOP
857     DEC R16
858     BRNE pi_pwmrepeat   ; Loop 31 times for a total of 495 cycles
859     DEC R17             ; Make the duty cycle counter roll over
860     DEC R18
861     BRNE pi_pwmloop     ; Repeat the same operation eight times (4095 cycles total)
862     RJMP pi_pwmdone
863
864 pi_pwmdone:
865     ; LDI R17,0x7F       ; Delay 256 cycles before exiting to skip communications
866     ; RCALL delay_one
867     ; LDI R17,0x7F
868     ; RCALL delay_one
869     RJMP pi_done
870
871 pi_pwmmonfull:
872     OUT PORTA,R24        ; Output the dead time state to the port
873     RJMP PC+1           ; Use a dead time of 800ns
874     RJMP PC+1
875     RJMP PC+1
876     RJMP PC+1
877     OUT PORTA,R19        ; Output the commanded on state to the port
878     LDI R16,0xFF         ; Delay 4080 cycles
879     RJMP PC+1           ; Delay 16 cycles per loop; 255 loops total
880     RJMP PC+1
881     RJMP PC+1
882     RJMP PC+1
883     RJMP PC+1
884     RJMP PC+1
885     NOP
886     DEC R16
887     BRNE PC-8
888     RJMP PC+1           ; Delay six more cycles for a total of 4086
889     RJMP PC+1
890     RJMP PC+1
891     OUT PORTA,R24        ; Output the dead time state to the port
892     RJMP PC+1           ; Use a dead time of 800ns
893     RJMP PC+1
894     RJMP PC+1
895     RJMP PC+1
896     OUT PORTA,R20        ; Output the commanded off state to the port
897     RJMP PC+1           ; Equalize delays with pi_pwmloop
898     RJMP PC+1
899     NOP
900     RJMP pi_pwmdone
901
902 pi_sync:
903     LDI R16,0x20         ; Time out after 32 bytes so as not to hang
904     CLR R17             ; Clear R17 to signify uncompletion of this calibration
905 pi_sync_restart:
906     WDR
907     LDI R20,0x91         ; Clear the temporary counter to an offset
908     SBIC PINB,2          ; Test the value of PB2 (wait for a low)
909     RJMP PC-1
910     INC R20             ; Increment the temporary counter

```



```

911 SBIS PINB,2          ; Test the value of PB2 (wait for a high)
912 RJMP PC-2
913 DEC R16              ; Time out if the calibration is right on the edge
914 BREQ pi_write        ; Also time out after 32 bytes received
915 TST R17
916 BRNE pi_sync_restart
917 CPI R20,0x80         ; Compare with the nominal 239 loops (956 cycles)
918 BRCS pi_sync_slow
919 BRNE pi_sync_fast
920 SER R17              ; Remember that this value was correct, and skip $00s
921 RJMP pi_sync_restart
922 pi_write:
923 RCALL bl_read_byte   ; Wait for another 125kBd byte to come in
924 CPI R18,0x56         ; Was a $56 sent?
925 BREQ PC+2            ; If not, this could be noise; exit
926 RJMP pi_done         ; Prevent spurious EEPROM writes
927 RCALL bl_read_byte   ; Wait for another 125kBd byte to come in
928 CPI R18,0x78         ; Was a $78 sent?
929 BREQ PC+2            ; If not, this could be noise; exit
930 RJMP pi_done         ; Prevent spurious EEPROM writes
931 RCALL speed_down     ; Change the oscillator frequency down to 8MHz
932 CLR R16              ; Write the EEPROM address to zero
933 OUT EEARL,R16
934 OUT EEARH,R16
935 OUT EEDR,R2          ; Write the correct calibration value to EEPROM
936 LDI R16,0x04         ; The control value used to erase and write at once
937 OUT EECR,R16
938 SBI EECR,1           ; Set the EEPE bit to enable programming
939 RCALL speed_up       ; Change the oscillator frequency back up to 12MHz
940 SBIC EECR,1          ; Delay until the program operation completes
941 RJMP PC-1
942 RJMP pi_done
943 pi_sync_slow:
944 INC R2               ; If the timer is too slow, increase the calibration
945 OUT OSCCAL,R2
946 RJMP pi_sync_restart
947 pi_sync_fast:
948 DEC R2              ; If the timer is too fast, decrease the calibration
949 OUT OSCCAL,R2
950 RJMP pi_sync_restart
951
952
953 write_byte:          ; Place byte to transmit LSB first at 1MBd in R20
954 IN R17,PORTB         ; Save the enable status (don't change it)
955 ANDI R17,0x08
956 LDI R16,0x00         ; Turn the transmitter on (PB0/PB1, active low)
957 OR R16,R17
958 OUT PORTB,R16
959 LDI R21,0x08         ; Clock eight bits
960 RJMP PC+1
961 RJMP PC+1
962 NOP
963 ROR R20              ; Shift right
964 LDI R16,0x00         ; Turn the transmitter on, possibly
965 BRCC PC+2            ; Is the bit clear? If so, skip the next instruction
966 LDI R16,0x03         ; Override R16=0 and turn the transmitter off
967 OR R16,R17
968 OUT PORTB,R16        ; Write the new value out to the port
969 RJMP PC+1            ; Each bit is twelve clock cycles
970 NOP
971 DEC R21              ; Decrement the bit counter
972 BRNE PC-9            ; Jump back up to the ROR R20 if R21 is not zero yet

```

```

973     NOP
974     BST R5,1                ; Copy the voltage high byte bit one to the T status bit
975     LDI R16,0x00            ; Turn the transmitter on, possibly
976     BRTC PC+2              ; Is the bit clear? If so, skip the next instruction
977     LDI R16,0x03            ; Override R16=0 and turn the transmitter off
978     OR R16,R17
979     OUT PORTB,R16           ; Write the new value out to the port
980     RJMP PC+1              ; Each bit is twelve clock cycles
981     RJMP PC+1
982     RJMP PC+1
983     RJMP PC+1
984     NOP
985     LDI R16,0x03            ; Turn the transmitter off (PB0/PB1, active low)
986     OR R16,R17
987     OUT PORTB,R16
988     RJMP PC+1
989     RJMP PC+1
990     RJMP PC+1
991     RJMP PC+1
992     RET
993
994
995     speed_up:                ; Increase the calibration value up to 12MHz
996     IN R16,OSCCAL           ; Recall the current calibration value
997     CP R16,R2               ; Is it 12MHz yet?
998     BRNE PC+2              ; Skip exiting if the Z bit is not set yet
999     RET                    ; Exit since 12MHz has been reached
1000    INC R16                 ; Increase the speed one more notch
1001    OUT OSCCAL,R16
1002    RJMP PC-5              ; Keep looping until 12MHz is reached
1003
1004
1005    speed_down:              ; Decrease the calibration value down to 8MHz
1006    IN R16,OSCCAL           ; Recall the current calibration value
1007    CP R16,R3               ; Is it 8MHz yet? (R3 is approximately $80)
1008    BRNE PC+2              ; Skip exiting if the Z bit is not set yet
1009    RET                    ; Exit since 8MHz has been reached
1010    DEC R16                 ; Increase the speed one more notch
1011    OUT OSCCAL,R16
1012    RJMP PC-5              ; Keep looping until 8MHz is reached
1013
1014
1015    delay_one:                ; Call with delay in R17 (0x10-0x7F), number of cycles
1016    SUBI R17,0x0C           ; Account for extra cycles in calling and other (twelve)
1017    ASR R17                 ; Divide the delay by two
1018    BRCS PC+1              ; Waste one more cycle on a positive initial bit zero
1019    ASR R17                 ; Divide the delay by two (four total)
1020    BRCS PC+1              ; Waste two more cycles on a positive initial bit one
1021    BRCS PC+1
1022    DEC R17                 ; Subtract one from the remaining delay
1023    NOP                    ; This delay makes a four-cycle loop
1024    BRNE PC-2              ; Repeat the loop if the counter is not zero yet
1025    RET
1026
1027
1028    .ORG 0x0370              ; The bootloader origin (accessory functions follow)
1029    .DW 0x0200              ; The hardware/bootloader version (2.00)
1030
1031    ;
1032    ; Bootloader commands:
1033    ;
1034    ; $00 - hang (independent of address) and let the watchdog reset the device

```

```

1035 ; $01 - fill page buffer with data (after $88), address in bytes one and two
1036 ; $03 - erase flash page address LSB in byte one, MSB in byte two
1037 ; $05 - write page buffer to flash page address LSB in byte one, MSB in byte two
1038 ; $11 - clear temporary page buffer for flash writing
1039 ; $41 - read EEPROM byte at address of byte one and return
1040 ; $44 - erase and write EEPROM byte of value byte two at address of byte one
1041 ; $54 - erase EEPROM byte at address of byte one
1042 ; $64 - write EEPROM byte of value byte two at address of byte one
1043 ; $81 - change address with new programming module address in byte one
1044 ; $82 - indirect jump to address LSB in byte one, MSB in byte two
1045 ; $84 - read flash data byte; address LSB in byte one, MSB in byte two
1046 ; $88 - put byte one into LSB of word flash data to write, byte two into MSB
1047 ; $FF - exit the EXT_INT0 interrupt normally (independent of address); no reset
1048 ;
1049
1050 bootloader:                ; This is the function to call from the EXT_INT0
1051     RCALL bl_start_bit      ; Skip the start bit wait loop
1052     CPI R18,0xAC            ; Was an $AC sent?
1053     BRNE bl_exit            ; If not, this could be noise; exit
1054     RCALL bl_read_byte      ; Wait for another byte to come in
1055     CPI R18,0x53            ; Was a $53 sent?
1056     BRNE bl_exit            ; If not, this could be noise; exit
1057     RCALL bl_read_byte      ; Wait for another byte to come in
1058 bl_chadr:
1059     CP R18,R23              ; Is this module being addressed?
1060     BRNE PC+4               ; Skip three instructions if not
1061     LDI R20,0x4B            ; Reply with a $4B to let the master know presence
1062     EOR R20,R23             ; Exclusive-OR this with the module address
1063     RCALL bl_write_byte
1064     MOV R19,R18             ; Save this address for comparison later
1065
1066 bl_mainloop:
1067     RCALL bl_read_byte      ; Wait for another byte to come in (address/data low)
1068     MOV R30,R18             ; Load the byte into ZL
1069     RCALL bl_read_byte      ; Wait for another byte to come in (address/data high)
1070     MOV R31,R18             ; Load the byte into ZH
1071     RCALL bl_read_byte      ; Wait for another byte to come in (always a command)
1072     TST R18                 ; If the command is $00, exit the bootloader
1073     BREQ PC                 ; Hang if $00 and therefore let the watchdog take over
1074
1075     CPI R18,0xFF            ; Is this an exit soft command?
1076     BREQ bl_exit
1077
1078     CPI R18,0x81            ; Is this a change address command?
1079     BRNE bl_notchadr
1080     MOV R18,R30             ; Modify the address by the first sent byte (ZL)
1081     RJMP bl_chadr
1082
1083 bl_notchadr:
1084     CP R19,R23              ; Is this module being addressed?
1085     BREQ bl_local           ; If so, skip the set red LED and clear green LED instrs.
1086     TST R19                 ; Is the address zero (global reprogram)?
1087     BRNE bl_mainloop        ; If not, keep waiting for an exit or an address change
1088     LDI R16,0x08            ; Turn the red LED on and green LED off (global mode)
1089     OUT PORTA,R16
1090
1091 bl_local:
1092     CPI R18,0x82            ; Is this an indirect jump (switch bootloaders) command?
1093     BRNE bl_notijmp
1094     IJMP                   ; Jump to the address stored in R31:R30 (ZH:ZL)
1095
1096 bl_notijmp:

```

```

1097     CPI R18,0x84           ; Is this a read/verify data command?
1098     BRNE bl_notread
1099     LPM R20,Z               ; Recall this program data byte to send off
1100     RCALL bl_write_byte
1101     RJMP bl_mainloop
1102
1103 bl_notread:
1104     CPI R18,0x88           ; Is this a move R31:R30 into R1:R0 (load data) command?
1105     BRNE bl_notdata
1106     MOVW R0,R30            ; Place R31:R30 into R1:R0 (address to data conversion)
1107     RJMP bl_mainloop       ; Loop again to receive the address of this data
1108
1109 bl_notdata:
1110     SBRS R18,6             ; Is this a read/write EEPROM instruction?
1111     RJMP bl_noteprom
1112
1113     RCALL bl_speed_down    ; Change the oscillator frequency down to 8MHz
1114     OUT EEARL,R30          ; Write the first received byte as an EEPROM address
1115     OUT EEDR,R31           ; Write the second received byte as data (if writing)
1116     OUT EECR,R18           ; Write the third received byte as an EEPROM control byte
1117     IN R20,EEDR            ; Input the EEPROM data to R20 to transmit
1118     SBI EECR,1             ; Set the EEPE bit to enable programming if needed or not
1119     RCALL bl_speed_up      ; Change the oscillator frequency back up to 12MHz
1120     SBIC EECR,1            ; Delay until the possible program operation completes
1121     RJMP PC-1
1122     TST R19                ; If this is a global command, don't reply
1123     BREQ bl_mainloop       ; Keep waiting for more bytes to come in
1124     RCALL bl_write_byte
1125     RJMP bl_mainloop       ; Keep waiting for more bytes to come in
1126
1127 bl_noteprom:
1128     SBRS R18,0             ; Test the LSB of the command
1129     RJMP bl_mainloop       ; If zero, this must be noise data, so wait for a $00
1130
1131     RCALL bl_speed_down    ; Change the oscillator frequency down to 8MHz for SPM
1132     OUT SPMCSR,R18         ; Write $01/$03/$05 to SPMCSR and execute SPM
1133     SPM                    ; Store (or modify) program memory
1134     RCALL bl_speed_up      ; Change the oscillator frequency back up to 12MHz
1135     RJMP bl_mainloop       ; Keep waiting for more bytes to come in
1136
1137 bl_exit:
1138     LDI R16,0x00           ; Turn off the LEDs
1139     OUT PORTA,R16
1140     OUT SREG,R15           ; Recall the status register
1141     RETI
1142
1143
1144 bl_speed_up:
1145     IN R16,OSCCAL          ; Increase the calibration value up to 12MHz
1146     CP R16,R2              ; Recall the current calibration value
1147     BRNE PC+2              ; Is it 12MHz yet?
1148     RET                    ; Skip exiting if the Z bit is not set yet
1149     INC R16                ; Exit since 12MHz has been reached
1150     OUT OSCCAL,R16         ; Increase the speed one more notch
1151     RJMP PC-5              ; Keep looping until 12MHz is reached
1152
1153
1154 bl_speed_down:
1155     IN R16,OSCCAL          ; Decrease the calibration value down to 8MHz
1156     CP R16,R3              ; Recall the current calibration value
1157     BRNE PC+2              ; Is it 8MHz yet? (R3 is approximately $80)
1158     RET                    ; Skip exiting if the Z bit is not set yet
1159     RET                    ; Exit since 8MHz has been reached

```

```

1159     DEC R16                ; Increase the speed one more notch
1160     OUT OSCCAL,R16
1161     RJMP PC-5              ; Keep looping until 8MHz is reached
1162
1163
1164     bl_delay_one:          ; Call with delay in R17 (0x10-0x7F), number of cycles
1165     SUBI R17,0x0C          ; Account for extra cycles in calling and other (twelve)
1166     ASR R17                ; Divide the delay by two
1167     BRCS PC+1              ; Waste one more cycle on a positive initial bit zero
1168     ASR R17                ; Divide the delay by two (four total)
1169     BRCS PC+1              ; Waste two more cycles on a positive initial bit one
1170     BRCS PC+1
1171     DEC R17                ; Subtract one from the remaining delay
1172     NOP                    ; This delay makes a four-cycle loop
1173     BRNE PC-2              ; Repeat the loop if the counter is not zero yet
1174     RET
1175
1176
1177     bl_write_byte:         ; Place byte to transmit LSB first at 125kBd in R20
1178     LDI R16,0x04           ; Turn the red LED off and green LED on (local mode)
1179     OUT PORTA,R16
1180     LDI R16,0x08           ; Turn the transmitter on (PB0/PB1, active low)
1181     OUT PORTB,R16
1182     LDI R21,0x08           ; Clock eight bits
1183     LDI R17,0x59           ; Delay exactly 89 cycles
1184     RCALL bl_delay_one
1185     ROR R20                ; Shift right
1186     LDI R16,0x08           ; Turn the transmitter on, possibly
1187     BRCC PC+2              ; Is the bit clear? If so, skip the next instruction
1188     LDI R16,0x0B           ; Override R16=0 and turn the transmitter off
1189     OUT PORTB,R16          ; Write the new value out to the port
1190     LDI R17,0x57           ; Delay exactly 87 cycles
1191     RCALL bl_delay_one
1192     DEC R21                ; Decrement the bit counter
1193     BRNE PC-8              ; Jump back up to the ROR R20 if R21 is not zero yet
1194     LDI R16,0x0B           ; Turn the transmitter off (PB0/PB1, active low)
1195     OUT PORTB,R16
1196     LDI R17,0x5A           ; Delay exactly 90 cycles for the stop bit
1197     RCALL bl_delay_one
1198     RET
1199
1200
1201     bl_read_byte:         ; Receive a byte at 125kBd for bootloader number one
1202     WDR                    ; Make sure the watchdog timer doesn't expire
1203     SBIC PINB,2            ; Test the value of PB2
1204     RJMP PC-2              ; If the pin is still set, keep waiting for a start bit
1205     LDI R17,0x72           ; Delay exactly 114 cycles on a found start bit
1206     RCALL bl_delay_one
1207     bl_start_bit:         ; This can be called after reading a zero 1MBd byte
1208     LDI R17,0x1D           ; Delay exactly 27 cycles
1209     RCALL bl_delay_one
1210     LDI R18,0x80           ; Load eight bits into R18
1211     CLC                    ; Prepare carry flag
1212     ROR R18                ; Shift right
1213     SBIC PINB,2            ; Test the value of PB2
1214     SBR R18,0x80           ; Set bit seven if on
1215     IN R16,SREG             ; Save the carry flag
1216     LDI R17,0x58           ; Delay exactly 88 cycles
1217     RCALL bl_delay_one
1218     OUT SREG,R16           ; Recall the carry flag
1219     BRCC PC-7              ; Locate the initially placed bit
1220     RET

```

APPENDIX E

Master Code Listings

E.1 DSPIC30F4011 USB Assembly Code, Design One

```
1 ; The SENDTOKEN macro, for quickly sending words to the slave modules
2
3 .MACRO SENDTOKEN ; This macro tests the FIFO buffer for a token to send
4 DEC.W [W13],W10 ; Test for a zero token timestamp and simultaneously load
5 BTSC.W SR,#C ; If there was a borrow (C=0), don't load the timer value
6 MOV.W TMR2,W10 ; Recall the low word of the 32-bit timer value
7 CP.W W10,[W13] ; Perform (TMR2)-(timestamp); if borrow above, result=-1
8 BRA N,+.4 ; Skip the next instruction if the time is too early
9 MOV.W [++W13],[W14] ; Transmit the token; make use of modulo addressing here
10 BRA N,+.4 ; Skip the next instruction if the time is too early
11 MOV.W [++W13],W10 ; Have the address generator automatically perform modulo
12 .ENDM
13
14
15 ; The USB interrupt routine (activated by a level change on RC13/RC14)
16
17 __CNInterrupt:
18 PUSH.S ; Save the status register, W0, W1, W2, and W3
19 PUSH.W W8 ; Save the working register W8
20 PUSH.W W9 ; Save the working register W9
21 MOV.W PORTC,W9 ; Clear the notification condition
22 MOV.W #0x6000,W0 ; Mask the D+/D- bits respectively
23 AND.W W9,W0,W9
24 BRA NZ,notse0 ; If SE0, don't branch and check for a reset
25
26 CLR.W TMR1 ; Clear the timer value
27 BCLR.W T1CON,#TON ; Turn the timer off before changing the prescaler
28 BSET.W T1CON,#TCKPS1 ; Choose a 1:64 prescaler for 140ms maximum
29 BSET.W T1CON,#TON ; Start the timer counting
30 BRA __ICN_end ; Wait until something happens
31
32 notse0:
33 BTSS.W T1CON,#TON ; Was the timer on?
34 BRA syncbyte ; If not, this must be a sync byte
35 MOV.W #MINRESET,W0 ; Test a pulse of 4215*64/30MHz (9ms)
36 SUB.W TMR1,WREG ; Compute W0-TMR1-W0
37 BRA GTU,resetit ; A long single-ended zero means reset the part
38 BCLR.W T1CON,#TON ; Disable the timer for now since the bus is idle
39 BRA __ICN_end ; Wait until the bus is non-idle
```

```

40
41 syncbyte:
42     MOV.W #0x4000,W0      ; Test for a RC14 (D+) high and RC13 (D-) low
43     XOR.W W9,W0,W0        ; If the bus was not D+ high and D- low (K), exit
44     BRA NZ, __ICN_end
45
46     MOV.W #0x00E8,W0      ; Timeout after 24 cycles cumulatively (0x0100-0x0018)
47     INC.B W0,W0           ; Increment the timeout counter
48     BTSS.W PORTC,#13      ; Wait for a D+ low and D- high (J)
49     BRA NZ,.-4            ; Branch if no bit and no timeout
50     BRA Z,resetit         ; Exit early on a timeout
51     SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
52     REPEAT #13            ; Delay past most of the sync byte for 15 cycles
53     NOP
54     SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
55     REPEAT #13            ; Delay past most of the sync byte for 15 cycles
56     NOP
57     SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
58     REPEAT #13            ; Delay past most of the sync byte for 15 cycles
59     NOP
60     SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
61     INC.B W0,W0           ; Increment the timeout counter
62     BTSS.W PORTC,#14      ; Wait for a D+ high and D- low (K)
63     BRA NZ,.-4            ; Branch if no bit and no timeout
64     BRA Z,resetit         ; Exit early on a timeout
65
66 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; RECEIVE FUNCTION ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
67
68     SETM.W W3              ; Set W3 to all ones; this is the CRC "shift register"
69     MOV.W #0xA001,W8       ; Set W8 to the CRC-16 polynomial
70     MOV.W #0x00FB,W2       ; Load (0x0100-0x0005) into W2 for the bitstuffing count
71     MOV.W #inbuffer+1,W0   ; Set up W0 and W9 to point to the input buffer areas
72     MOV.W #inbuffer+2,W9
73     SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
74     REPEAT #18            ; Delay 20 cycles to find the center of the next bit
75     NOP
76 getpid_j:                ; This label is called after a J to K transition is found
77     SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
78     REPEAT #2             ; Delay four cycles for a total of 20
79     NOP
80     INC.B W2,W2           ; Increment the bitstuffing count; it may be cleared next
81     BSET.W SR,#C          ; Set the carry bit early in case a one bit comes
82     BTSC.W PORTC,#14      ; If D+ is now low, this is a zero bit
83     BRA getpid_j_one      ; If not, this is surely a one bit
84     MOV.W #0x00FA,W2      ; Load (0x0100-0x0006) into W2 for the bitstuffing count
85     LSR.B [W0],[W0]       ; Shift the location right to put a zero at bit seven
86     BRA NC,getpid_k       ; If the placed bit has not appeared yet, keep looping
87     BRA waitbit_k        ; This branch ends seven cycles after the PORTC test
88 getpid_j_one:
89     RRC.B [W0],[W0]       ; Shift the location right to put a one at bit seven
90     BRA NC,getpid_j       ; If the placed bit has not appeared yet, keep looping
91     BRA waitbit_j        ; This branch ends seven cycles after the PORTC test
92 getpid_k:                ; This label is called after a K to J transition is found
93     SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
94     REPEAT #2             ; Delay four cycles for a total of 20
95     NOP
96     INC.B W2,W2           ; Increment the bitstuffing count; it may be cleared next
97     BSET.W SR,#C          ; Set the carry bit early in case a one bit comes
98     BTSC.W PORTC,#13      ; If D- is now low, this is a zero bit
99     BRA getpid_k_one      ; If not, this is surely a one bit
100    MOV.W #0x00FA,W2      ; Load (0x0100-0x0006) into W2 for the bitstuffing count
101    LSR.B [W0],[W0]       ; Shift the location right to put a zero at bit seven

```

```

102     BRA NC,getpid_j      ; If the placed bit has not appeared yet, keep looping
103     BRA waitbit_j        ; This branch ends seven cycles after the PORTC test
104 getpid_k_one:
105     RRC.B [W0],[W0]      ; Shift the location right to put a one at bit seven
106     BRA NC,getpid_k      ; If the placed bit has not appeared yet, keep looping
107     BRA waitbit_k        ; This branch ends seven cycles after the PORTC test
108 waitbit_k_done:
109     LSR.W W3,W3          ; A zero K bit was found; store it and put it in the CRC
110     BTSC.W SR,#C         ; Shift the checksum right by one bit into carry
111     XOR.W W3,W8,W3       ; Is the previous CRC low bit one?
112     LSR.B [W9],[W9]      ; XOR the words together if the bits differ
113     ADDC.W #0,W9         ; Shift the location right to put a zero at bit seven
114     MOV.W #0x00FA,W2     ; If the placed one has shown up, increment the address
115     SENDTOKEN            ; Load (0x0100-0x0006) into W2 for the bitstuffing count
116                          ; Insert the SENDTOKEN macro (eight cycles)
117 waitbit_j:
118     BTSC.W PORTC,#13     ; Wait for a D+ low and D- high (J)
119     BRA waitbit_j_done   ; Skip on a falling edge of D- in case of a stop bit
120     BTSC.W PORTC,#13     ; Wait for a D+ low and D- high (J)
121     BRA waitbit_j_done   ; Skip on a falling edge of D- in case of a stop bit
122     BTSC.W PORTC,#13     ; Wait for a D+ low and D- high (J)
123     BRA waitbit_j_done   ; Skip on a falling edge of D- in case of a stop bit
124     BTSC.W PORTC,#13     ; Wait for a D+ low and D- high (J)
125     BRA waitbit_j_done   ; Skip on a falling edge of D- in case of a stop bit
126     BTSS.W PORTC,#14     ; Test to see if both D+ and D- are low; if so, branch
127     BRA bitstream_done  ; If this is a single-ended zero, the receive is over
128     LSR.W W3,W3          ; Shift the checksum right by one bit into carry
129     BTSS.W SR,#C         ; Is the previous CRC low bit zero?
130     XOR.W W3,W8,W3       ; XOR the words together if the bits differ
131     BSET.W SR,#C         ; Set the carry bit early in case a one bit comes
132     RRC.B [W9],[W9]      ; Shift the location right to put a one at bit seven
133     ADDC.W #0,W9         ; If the placed one has shown up, increment the address
134     NOP                  ; This makes 20 cycles in the loop total, for each bit
135     INC.B W2,W2          ; Increment the bitstuffing count since a one just came
136     BRA NZ,waitbit_j     ; If the count didn't carry over to 0x00, wait for a one
137     MOV.W #0x00F6,W0     ; Timeout after 10 cycles (0x0100-0x000A)
138     INC.B W0,W0          ; Increment the timeout counter
139     BTSC.W PORTC,#14     ; Wait for a D+ low and D- high (J)
140     BRA NZ, -4           ; Resynchronize on a stuffed bit edge as well
141     BRA Z,resetit        ; Exit early on a timeout
142     SENDTOKEN            ; Insert the SENDTOKEN macro (eight cycles)
143     NOP                  ; Ignore the stuffed bit for one more cycle
144     MOV.W #0x00FA,W2     ; Load (0x0100-0x0006) into W2 for the bitstuffing count
145     BRA waitbit_k        ; Insert the SENDTOKEN macro (eight cycles)
146
147 waitbit_j_done:
148     LSR.W W3,W3          ; A zero J bit was found; store it and put it in the CRC
149     BTSC.W SR,#C         ; Shift the checksum right by one bit into carry
150     XOR.W W3,W8,W3       ; Is the previous CRC low bit one?
151     LSR.B [W9],[W9]      ; XOR the words together if the bits differ
152     ADDC.W #0,W9         ; Shift the location right to put a zero at bit seven
153     MOV.W #0x00FA,W2     ; If the placed one has shown up, increment the address
154     SENDTOKEN            ; Load (0x0100-0x0006) into W2 for the bitstuffing count
155                          ; Insert the SENDTOKEN macro (eight cycles)
156 waitbit_k:
157     BTSC.W PORTC,#14     ; Wait for a D+ high and D- low (K)
158     BRA waitbit_k_done   ; Skip on a falling edge of D+ in case of a stop bit
159     BTSC.W PORTC,#14     ; Wait for a D+ high and D- low (K)
160     BRA waitbit_k_done   ; Skip on a falling edge of D+ in case of a stop bit
161     BTSC.W PORTC,#14     ; Wait for a D+ high and D- low (K)
162     BRA waitbit_k_done   ; Skip on a falling edge of D+ in case of a stop bit
163     BTSC.W PORTC,#14     ; Wait for a D+ high and D- low (K)

```



```

164   BRA waitbit_k_done      ; Skip on a falling edge of D+ in case of a stop bit
165   BTSS.W PORTC,#13       ; Test to see if both D+ and D- are low; if so, branch
166   BRA bitstream_done     ; If this is a single-ended zero, the receive is over
167   LSR.W W3,W3            ; Shift the checksum right by one bit into carry
168   BTSS.W SR,#C           ; Is the previous CRC low bit zero?
169   XOR.W W3,W8,W3        ; XOR the words together if the bits differ
170   BSET.W SR,#C          ; Set the carry bit early in case a one bit comes
171   RRC.B [W9],[W9]        ; Shift the location right to put a one at bit seven
172   ADDC.W #0,W9          ; If the placed one has shown up, increment the address
173   NOP                   ; This makes 20 cycles in the loop total, for each bit
174   INC.B W2,W2           ; Increment the bitstuffing count since a one just came
175   BRA NZ,waitbit_k       ; If the count didn't carry over to 0x00, wait for a one
176   MOV.W #0x00F6,W0      ; Timeout after 10 cycles (0x0100-0x000A)
177   INC.B W0,W0           ; Increment the timeout counter
178   BTSC.W PORTC,#13      ; Wait for a D+ high and D- low (K)
179   BRA NZ,.-4            ; Resynchronize on a stuffed bit edge as well
180   BRA Z,resetit         ; Exit early on a timeout
181   SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
182   NOP                   ; Ignore the stuffed bit for one more cycle
183   MOV.W #0x00FA,W2      ; Load (0x0100-0x0006) into W2 for the bitstuffing count
184   BRA waitbit_j
185
186 bitstream_done:
187   COM.W W3,W8            ; Complement the optional CRC to complete the calculation
188   MOV.B [W9-2],W1       ; Recall the next-to-last byte received, maybe CRC LSB
189   SWAP.W W1             ; Push this to the MSB location temporarily
190   MOV.B [W9-1],W1       ; Add on the last byte received, maybe CRC MSB
191   SWAP.W W1             ; Swap the two bytes in W1 for the proper orientation
192   MOV.W W9,W3           ; Save the ending address in W3
193   MOV.W #inbuffer,W9    ; Load the buffer address into W9
194   SUB.W W3,W9,W3        ; W3 is now the number of bytes received
195   SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
196   REPEAT #15            ; Delay past the single-ended zero end-of-packet
197   NOP
198   SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
199
200 ;;;;;;;;;;;;;;;;;;;;;;;;;; SETUP TOKEN COMPARISON ;;;;;;;;;;;;;;;;;;;;;;;;;;
201
202   MOV.B [++W9],W0        ; Test the first byte of the data buffer (exclude sync)
203   SUB.B #0x2D,W0         ; Compare to the SETUP token at address/endpoint zero
204   BRA NZ,notsetup       ; If not equal, this is not a SETUP token packet
205   MOV.B [++W9],W0        ; Test the second byte of the buffer
206   MOV.B WREG,lastaddr    ; Save the address and endpoint of this token packet
207   BRA __ICN_end
208
209 ;;;;;;;;;;;;;;;;;;;;;;;;;; IN TOKEN COMPARISON ;;;;;;;;;;;;;;;;;;;;;;;;;;
210
211 notsetup:
212   MOV.B [W9],W0          ; Test the first byte of the buffer
213   SUB.B #0x69,W0         ; Compare to the IN token at address/endpoint zero
214   BRA NZ,notin          ; If not equal, this is not an IN packet
215   MOV.B [++W9],W0        ; Test the second byte of the buffer
216   MOV.B WREG,lastaddr    ; Save the address and endpoint of this token packet
217   SUB.B address,WREG     ; Is the address correct and one endpoint bit zero?
218   BRA Z,in_endp0        ; If equal, this is data for endpoint zero
219   AND.W #0x007F,W0       ; Mask off the top bit to test for endpoint one
220   BRA NZ,badpacket      ; If zero, this is data for endpoint one
221
222 ;;;;;;;;;;;;;;;;;;;;;;;;;; INPUT INTERRUPT PROCESSING FUNCTION ;;;;;;;;;;;;;;;;;;;;;;;;;;
223
224   CP0.W nextreport      ; Was a report recently requested?
225   BRA NZ,report

```

```

226
227 ; Try to read in words from the hardware UART receive FIFO buffer
228 MOV.W #uartin,W2 ; Test to see if the software UART FIFO buffer is full
229 BTSC.W [W2],#15 ; If the high bit is set
230 BRA uart0full
231 BTSS.W U1STA,#URXDA ; Is there at least one available word?
232 BRA uartread_end
233 MOV.W U1RXREG,W0 ; Recall the first received word
234 BSET.W W0,#15 ; Set the top bit to let the PC know this is real data
235 MOV.W W0,[W2] ; Store the received word + 0x8000 to the output buffer
236 uart0full:
237 BTSC.W [++W2],#15 ; If the high bit is set
238 BRA uart1full
239 BTSS.W U1STA,#URXDA ; Is there another available word?
240 BRA uartread_end
241 MOV.W U1RXREG,W0 ; Recall the first received word
242 BSET.W W0,#15 ; Set the top bit to let the PC know this is real data
243 MOV.W W0,[W2] ; Store the received word + 0x8000 to the output buffer
244 uart1full:
245 BTSC.W [++W2],#15 ; If the high bit is set
246 BRA uart2full
247 BTSS.W U1STA,#URXDA ; Is there another available word?
248 BRA uartread_end
249 MOV.W U1RXREG,W0 ; Recall the first received word
250 BSET.W W0,#15 ; Set the top bit to let the PC know this is real data
251 MOV.W W0,[W2] ; Store the received word + 0x8000 to the output buffer
252 uart2full:
253 BTSC.W [++W2],#15 ; If the high bit is set
254 BRA uartread_end
255 BTSS.W U1STA,#URXDA ; Is there another available word?
256 BRA uartread_end
257 MOV.W U1RXREG,W0 ; Recall the first received word
258 BSET.W W0,#15 ; Set the top bit to let the PC know this is real data
259 MOV.W W0,[W2] ; Store the received word + 0x8000 to the output buffer
260 uartread_end:
261
262 ; An interrupt IN packet just came in; test for bytes needing to be sent
263 MOV.W #uartin,W1 ; Load the software UART FIFO buffer address
264 MOV.W #outbuffer,W11 ; Load the output buffer address
265 MOV.W TMR2,W2 ; Load the 32-bit timer value and send it to the PC
266 MOV.W TMR3HLD,W3 ; The four bytes of the long will be little Endian
267 MOV.D W2,[W11++] ; Place a long into the USB output buffer
268
269 BTSS.W [W1],#15 ; Is there at least one available word?
270 BRA in_int_0 ; If not, clear out the entire USB output buffer
271 MOV.W [W1],[W11++] ; Store the received word + 0x8000 to the output buffer
272 BTSS.W [++W1],#15 ; Is there another available word?
273 BRA in_int_1 ; If not, clear out part of the USB output buffer
274 MOV.W [W1],[W11++] ; Store the received word + 0x8000 to the output buffer
275 BRA in_int_done ; Don't clear out any of the USB output buffer
276 in_int_0:
277 CLR.W [W11++] ; Clear out the USB output buffer first word
278 in_int_1:
279 CLR.W [W11++] ; Clear out the USB output buffer second word
280 in_int_done:
281 MOV.W #outbuffer,W11 ; Load the output buffer address
282 MOV.W #8,W3 ; Load the byte count into W3
283 BRA senddata_wait
284
285 report:
286 MOV.W nextreport,W0 ; Test to see which report was requested, and answer
287 CLR.W W1 ; Clear the answer initially

```

```

288     CP.W W0,#1           ; Is this a throttle position report request?
289     BRA NZ,report_notthr
290     MOV.W throttle,W1
291     BRA report_done
292
293 report_notthr:
294     CP.W W0,#2           ; Is this a regen position report request?
295     BRA NZ,report_notreg
296     MOV.W regen,W1
297     BRA report_done
298
299 report_notreg:
300     CP.W W0,#3           ; Is this a random number seed?
301     BRA NZ,report_notseed
302     MOV.W TMR2,W2        ; Load the 32-bit timer value to seed with
303     MOV.W TMR3HLD,W3
304     MOV.W W2,rand32l
305     MOV.W W3,rand32h
306     CLR.W W1
307     BRA report_done
308
309 report_notseed:
310     CP.W W0,#4           ; Is this a random number request?
311     BRA NZ,report_notrand
312     RCALL make_random
313     MOV.W rand32l,W0
314     MOV.W rand32h,W1
315     BRA report_done
316
317 report_notrand:
318     CP.W W0,#5           ; Is this a currents U and V request?
319     BRA NZ,report_notcuruv
320     MOV.W curu,W0
321     MOV.W curv,W1
322     BRA report_done
323
324 report_notcuruv:
325     CP.W W0,#6           ; Is this a currents alpha, beta, D, and Q request?
326     BRA NZ,report_notcurabdq
327     MOV.W curalpha,W0
328     MOV.W curbeta,W1
329     MOV.W curd,W2
330     MOV.W curq,W3
331     MOV.W #outbuffer,W11 ; Load the output buffer address
332     MOV.W W0,[W11++]
333     MOV.W W1,[W11++]
334     MOV.W W2,[W11++]
335     MOV.W W3,[W11++]
336     MOV.W #outbuffer,W11 ; Load the output buffer address
337     MOV.W #8,W3          ; Load the byte count into W3
338     BRA senddata_wait
339
340 report_notcurabdq:
341     CP.W W0,#7           ; Is this a number seven request?
342     BRA NZ,report_notnum7
343     MOV.W frequency,W0
344     MOV.W pfavg32h,W1
345     MOV.W curd,W2
346     MOV.W curq,W3
347     MOV.W #outbuffer,W11 ; Load the output buffer address
348     MOV.W W0,[W11++]
349     MOV.W W1,[W11++]

```

```

350     MOV.W W2,[W11++]
351     MOV.W W3,[W11++]
352     MOV.W #outbuffer,W11 ; Load the output buffer address
353     MOV.W #8,W3          ; Load the byte count into W3
354     BRA senddata_wait
355
356 report_notnum7:
357     CP.W W0,#8           ; Is this a number eight request?
358     BRA NZ,report_notnum8
359     MOV.W frequency,W0
360     MOV.W pfavg32h,W1
361     MOV.W ampl32l,W2      ; Send off the amplitude in PWM units (32 per low word)
362     MOV.W ampl32h,W3
363     SL.W W3,#5,W3        ; Make room for the five lower amplitude bits
364     LSR.W W2,#11,W2      ; Rid of the unnecessary 11 lowest amplitude bits
365     IOR.W W2,W3,W2
366     MOV.W currms,W3
367     MOV.W #outbuffer,W11 ; Load the output buffer address
368     MOV.W W0,[W11++]
369     MOV.W W1,[W11++]
370     MOV.W W2,[W11++]
371     MOV.W W3,[W11++]
372     MOV.W #outbuffer,W11 ; Load the output buffer address
373     MOV.W #8,W3          ; Load the byte count into W3
374     BRA senddata_wait
375
376 report_notnum8:
377     CP.W W0,#9           ; Is this a number nine request?
378     BRA NZ,report_notnum9
379     MOV.W frequency,W0
380     MOV.W pfavg32h,W1
381     MOV.W ampl32l,W2      ; Send off the amplitude in PWM units (32 per low word)
382     MOV.W ampl32h,W3
383     SL.W W3,#5,W3        ; Make room for the five lower amplitude bits
384     LSR.W W2,#11,W2      ; Rid of the unnecessary 11 lowest amplitude bits
385     IOR.W W2,W3,W2
386     MOV.W curq,W3
387     MOV.W #outbuffer,W11 ; Load the output buffer address
388     MOV.W W0,[W11++]
389     MOV.W W1,[W11++]
390     MOV.W W2,[W11++]
391     MOV.W W3,[W11++]
392     MOV.W #outbuffer,W11 ; Load the output buffer address
393     MOV.W #8,W3          ; Load the byte count into W3
394     BRA senddata_wait
395
396 report_notnum9:
397     CP.W W0,#10          ; Is this an echo bytes request?
398     BRA NZ,report_notecho
399     MOV.W pc_loop,W1
400     MOV.W pc_pfkp,W2
401     MOV.W pc_pfka,W3
402     MOV.W #outbuffer,W11 ; Load the output buffer address
403     MOV.W W0,[W11++]
404     MOV.W W1,[W11++]
405     MOV.W W2,[W11++]
406     MOV.W W3,[W11++]
407     MOV.W #outbuffer,W11 ; Load the output buffer address
408     MOV.W #8,W3          ; Load the byte count into W3
409     BRA senddata_wait
410
411 report_notecho:

```

```

412     CP.W W0,#11          ; Is this a check battery module comm. request?
413     BRA NZ,report_notcc
414     MOV.W readhigh,W1    ; Recall the return value
415     CPO.W readhigh
416     BRA Z,report_done
417     CLR.W usbcommand
418     BRA report_done
419
420 report_notcc:
421     CP.W W0,#24          ; Is this an echo read bytes request?
422     BRA NZ,report_notread
423     MOV.W bootaddr1,W1
424     MOV.W bootaddrh,W2
425     MOV.W readhigh,W3
426     MOV.W #outbuffer,W11 ; Load the output buffer address
427     MOV.W W0,[W11++]
428     MOV.W W1,[W11++]
429     MOV.W W2,[W11++]
430     MOV.W W3,[W11++]
431     MOV.W #outbuffer,W11 ; Load the output buffer address
432     MOV.W #8,W3          ; Load the byte count into W3
433     BRA senddata_wait
434
435 report_notread:
436
437 report_done:
438     MOV.W #outbuffer,W11 ; Load the output buffer address
439     MOV.W TMR2,W2        ; Load the 32-bit timer value and send it to the PC
440     MOV.W TMR3HLD,W3     ; The four bytes of the long will be little Endian
441     MOV.D W2,[W11++]     ; Place a long into the USB output buffer
442     MOV.W W0,[W11++]     ; Set the USB output buffer third word to the report type
443     MOV.W W1,[W11++]     ; Set the USB output buffer fourth word to the data
444     MOV.W #outbuffer,W11 ; Load the output buffer address
445     MOV.W #8,W3          ; Load the byte count into W3
446     BRA senddata_wait
447
448 in_endp0:
449     MOV.B addrbuf,WREG    ; Update the address only in this status stage
450     MOV.B WREG,address
451
452     MOV.B nextlen,WREG    ; Load the previously set byte count into W3
453     MOV.B W0,W3
454     AND.W #0x00FF,W3
455     BRA senddata_wait
456
457 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; OUT TOKEN COMPARISON ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
458
459 notin:
460     MOV.B [W9],W0         ; Test the first byte of the buffer
461     SUB.B #0xE1,W0        ; Compare to the OUT token at address/endpoint zero
462     BRA NZ,notout         ; If not equal, this is not an OUT packet
463     MOV.B [++W9],W0       ; Test the second byte of the buffer
464     MOV.B WREG,lastaddr   ; Save the address and endpoint of this token packet
465     SUB.B address,WREG    ; Is the address correct and one endpoint bit zero?
466     BRA Z,out_endp0       ; If equal, this is data for endpoint zero
467     AND.W #0x007F,W0     ; Mask off the top bit to test for endpoint one
468     BRA NZ,badpacket      ; If zero, this is data for endpoint one
469     BRA __ICN_end
470
471 out_endp0:
472     MOV.B addrbuf,WREG    ; Update the address only in this status stage
473     MOV.B WREG,address

```

```

474     BRA __ICN_end
475
476     ;;;;;;;;;;;;;; DATA0 TOKEN COMPARISON ;;;;;;;;;;;;;;
477
478     notout:
479         MOV.B [W9],W0          ; Test the first byte of the buffer
480         SUB.B #0xC3,W0        ; Compare to the DATA0 token
481         BRA NZ,notdata0       ; If not equal, this is not a DATA0 packet
482         MOV.W #0x00C3,W0      ; Record that this was a DATA0 packet for toggling
483         BRA datapacket
484
485     ;;;;;;;;;;;;;; DATA1 TOKEN COMPARISON ;;;;;;;;;;;;;;
486
487     notdata0:
488         MOV.B [W9],W0          ; Test the first byte of the buffer
489         SUB.B #0x4B,W0        ; Compare to the DATA1 token
490         BRA NZ,notdata       ; If not equal, this is not a DATA1 packet
491         MOV.W #0x004B,W0      ; Record that this was a DATA1 packet for toggling
492     datapacket:
493         MOV.B WREG,lastdata
494
495         ; The received CRC is in W1, and the calculated CRC is in W8. Unfortunately,
496         ; the calculated CRC is also over the received CRC. In order to validate the
497         ; packet, the received CRC must be recalculated over itself.
498         MOV.W #16,W2          ; Loop 16 times, once for each CRC bit
499         COM.W W1,W3           ; Undo the CRC complement applied by the host, into W3
500         MOV.W #0xA001,W0      ; Set W0 to the CRC-16 polynomial
501         LSR.W W3,W3           ; Shift the checksum right by one bit into carry
502         RRNC.W W1,W1          ; Rotate the byte right to send LSB first as specified
503         BTSS.W SR,#N          ; Is the data low bit zero?
504         XOR.W W3,W0,W3        ; XOR the words together (destroys negative status bit)
505         BTSS.W SR,#C          ; Is the previous CRC low bit zero?
506         XOR.W W3,W0,W3        ; XOR the words together (perhaps undoing the above XOR)
507         DEC.W W2,W2           ; Decrement the remaining bit count
508         BRA NZ,.-14           ; If the count is not zero, keep looping
509         COM.W W3,W3           ; Complement W3 as specified by the USB standard
510         CP.W W3,W8            ; Did the CRC's compare equally? If not, exit now
511         BRA NZ,badpacket
512         RCALL sendack         ; Send an acknowledgment if the packet's CRC was good
513
514         MOV.B lastaddr,WREG    ; Recall the token packet's address and low endpoint bit
515         SUB.B address,WREG     ; Is the address correct and one endpoint bit zero?
516         BRA Z,processdata     ; Start processing this endpoint zero DATAx packet
517         AND.W #0x007F,W0      ; Mask off the top bit to test for endpoint one
518         BRA NZ,badpacket     ; If zero, this is data for endpoint one
519
520     ;;;;;;;;;;;;;; OUTPUT INTERRUPT PROCESSING FUNCTION ;;;;;;;;;;;;;;
521
522     ; A DATAx packet just came in, with interrupt data
523     processint:                ; Process an endpoint one packet
524         INC.W W9,W9           ; Point to the data buffer
525
526         MOV.W [W9++],W0       ; Load in the first word of the USB input buffer
527         BTSC.W W0,#12         ; If the fourth-to-top bit is set, this is a report req.
528         BRA pi_report
529         CLR.W nextreport      ; If data is requested instead, clear this variable
530         BTSC.W W0,#11         ; If the fifth-to-top bit is set, this is a variable sent
531         BRA pi_vartrans
532         BTSC.W W0,#10         ; If the sixth-to-top bit is set, this is programming
533         BRA pi_program
534         MOV.W #0x0001,W1
535         BTSS.W W0,#13         ; If the third-to-top bit is set, this is a 937.5kBd byte

```

```

536 MOV.W #0x000F,W1 ; If the bit is clear, this is a 115.2kBd byte (or four)
537 MOV.W W1,U2BRG ; Modify the baud rate on the fly
538 BTSC.W W0,#15 ; If the top bit is set, this is real data to transmit
539 MOV.W W0,U2TXREG ; Transmit the nine-bit word
540 BTSS.W W0,#14 ; If the next-to-top bit it set, shift the UART buffer
541 BRA pi_not0
542 MOV.W #uartin,W2 ; Load the software UART FIFO buffer first word address
543 MOV.W #uartin+2,W3 ; Load the software UART FIFO buffer second word address
544 MOV.W [W3++],[W2++] ; Move the second word into the first word
545 MOV.W [W3++],[W2++] ; Move the third word into the second word
546 MOV.W [W3],[W2++] ; Move the fourth word into the third word
547 CLR.W [W3] ; Clear the fourth word
548 pi_not0:
549
550 MOV.W [W9++],W0 ; Load in the second word of the USB input buffer
551 BTSC.W W0,#15 ; If the top bit is set, this is real data to transmit
552 MOV.W W0,U2TXREG ; Transmit the nine-bit word
553 BTSS.W W0,#14 ; If the next-to-top bit it set, shift the UART buffer
554 BRA pi_not1
555 MOV.W #uartin,W2 ; Load the software UART FIFO buffer first word address
556 MOV.W #uartin+2,W3 ; Load the software UART FIFO buffer second word address
557 MOV.W [W3++],[W2++] ; Move the second word into the first word
558 MOV.W [W3++],[W2++] ; Move the third word into the second word
559 MOV.W [W3],[W2++] ; Move the fourth word into the third word
560 CLR.W [W3] ; Clear the fourth word
561 pi_not1:
562
563 MOV.W [W9++],W0 ; Load in the third word of the USB input buffer
564 BTSC.W W0,#15 ; If the top bit is set, this is real data to transmit
565 MOV.W W0,U2TXREG ; Transmit the nine-bit word
566 BTSS.W W0,#14 ; If the next-to-top bit it set, shift the UART buffer
567 BRA pi_not2
568 MOV.W #uartin,W2 ; Load the software UART FIFO buffer first word address
569 MOV.W #uartin+2,W3 ; Load the software UART FIFO buffer second word address
570 MOV.W [W3++],[W2++] ; Move the second word into the first word
571 MOV.W [W3++],[W2++] ; Move the third word into the second word
572 MOV.W [W3],[W2++] ; Move the fourth word into the third word
573 CLR.W [W3] ; Clear the fourth word
574 pi_not2:
575
576 MOV.W [W9++],W0 ; Load in the fourth word of the USB input buffer
577 BTSC.W W0,#15 ; If the top bit is set, this is real data to transmit
578 MOV.W W0,U2TXREG ; Transmit the nine-bit word
579 BTSS.W W0,#14 ; If the next-to-top bit it set, shift the UART buffer
580 BRA pi_not3
581 MOV.W #uartin,W2 ; Load the software UART FIFO buffer first word address
582 MOV.W #uartin+2,W3 ; Load the software UART FIFO buffer second word address
583 MOV.W [W3++],[W2++] ; Move the second word into the first word
584 MOV.W [W3++],[W2++] ; Move the third word into the second word
585 MOV.W [W3],[W2++] ; Move the fourth word into the third word
586 CLR.W [W3] ; Clear the fourth word
587 pi_not3:
588
589 BRA __ICN_end
590
591 pi_report:
592 MOV.W #0x0FFF,W1
593 AND.W W0,W1,W0
594 MOV.W W0,nextreport
595 CP.W W0,#11 ; Was this a check battery module comm. request?
596 BRA NZ,__ICN_end
597 MOV.W [W9++],W0 ; If so, save the address of the battery module

```

```

598     MOV.W W0,usbcommand
599     CLR.W readhigh
600     BRA  __ICN_end
601
602 pi_vartrans:
603     MOV.W [W9++],W0           ; Load in the second word of the USB input buffer
604     MOV.W W0,pc_loop
605     MOV.W [W9++],W0           ; Load in the third word of the USB input buffer
606     MOV.W W0,pc_pfkp
607     MOV.W [W9++],W0           ; Load in the fourth word of the USB input buffer
608     MOV.W W0,pc_pfka
609     BRA  __ICN_end
610
611 pi_program:
612     CP.B W0,#16               ; Is this a load 23-bit address operation request?
613     BRA NZ,pp_not16
614     MOV.W [W9++],W0           ; Load in the second word of the USB input buffer
615     MOV.W W0,bootaddr1
616     MOV.W [W9++],W0           ; Load in the third word of the USB input buffer
617     MOV.W W0,bootaddrh
618     BRA  __ICN_end
619
620 pp_not16:
621     CP.B W0,#17               ; Is this an erase program memory row operation request?
622     BRA NZ,pp_not17
623     MOV.W [W9++],W0           ; Load in the second word of the USB input buffer
624     MOV.W W0,NVMADR
625     MOV.W W0,bootaddr1
626     MOV.W [W9++],W0           ; Load in the third word of the USB input buffer
627     MOV.W W0,NVMADRU
628     MOV.W W0,bootaddrh
629     MOV.W #0x4041,W0          ; Set up NVMCON for erasing a program memory row
630     MOV.W W0,NVMCON
631     BRA  __ICN_prog
632
633 pp_not17:
634     CP.B W0,#18               ; Is this a write program memory row operation request?
635     BRA NZ,pp_not18
636     MOV.W [W9++],W0           ; Load in the second word of the USB input buffer
637     MOV.W W0,NVMADR
638     MOV.W [W9++],W0           ; Load in the third word of the USB input buffer
639     MOV.W W0,NVMADRU
640     MOV.W #0x4001,W0          ; Set up NVMCON for writing a program memory row
641     MOV.W W0,NVMCON
642     BRA  __ICN_prog
643
644 pp_not18:
645     CP.B W0,#19               ; Is this an erase data memory word operation request?
646     BRA NZ,pp_not19
647     MOV.W [W9++],W0           ; Load in the second word of the USB input buffer
648     MOV.W W0,NVMADR
649     MOV.W W0,bootaddr1
650     MOV.W [W9++],W0           ; Load in the third word of the USB input buffer
651     MOV.W W0,NVMADRU
652     MOV.W W0,bootaddrh
653     MOV.W #0x4044,W0          ; Set up NVMCON for erasing a data memory word
654     MOV.W W0,NVMCON
655     BRA  __ICN_prog
656
657 pp_not19:
658     CP.B W0,#20               ; Is this an erase data memory row operation request?
659     BRA NZ,pp_not20

```



```

660     MOV.W [W9++],W0          ; Load in the second word of the USB input buffer
661     MOV.W W0,NVMADR
662     MOV.W W0,bootaddr1
663     MOV.W [W9++],W0          ; Load in the third word of the USB input buffer
664     MOV.W W0,NVMADRU
665     MOV.W W0,bootaddrh
666     MOV.W #0x4045,W0         ; Set up NVMCON for erasing a data memory row
667     MOV.W W0,NVMCON
668     BRA __ICN_prog
669
670 pp_not20:
671     CP.B W0,#21              ; Is this a write data memory word operation request?
672     BRA NZ,pp_not21
673     MOV.W [W9++],W0          ; Load in the second word of the USB input buffer
674     MOV.W [W9++],W1          ; Load in the third word of the USB input buffer
675     MOV.W W1,TBLPAG
676     TBLWTL.W [W9++],[W0]     ; Load in the fourth word of the USB input buffer
677     MOV.W #0x4004,W0         ; Set up NVMCON for writing a data memory word
678     MOV.W W0,NVMCON
679     BRA __ICN_prog
680
681 pp_not21:
682     CP.B W0,#22              ; Is this a write data memory row operation request?
683     BRA NZ,pp_not22
684     MOV.W [W9++],W0          ; Load in the second word of the USB input buffer
685     MOV.W W0,NVMADR
686     MOV.W [W9++],W0          ; Load in the third word of the USB input buffer
687     MOV.W W0,NVMADRU
688     MOV.W #0x4005,W0         ; Set up NVMCON for writing a data memory row
689     MOV.W W0,NVMCON
690     BRA __ICN_prog
691
692 pp_not22:
693     CP.B W0,#23              ; Is this a write config. memory operation request?
694     BRA NZ,pp_not23
695     MOV.W [W9++],W0          ; Load in the second word of the USB input buffer
696     MOV.W [W9++],W1          ; Load in the third word of the USB input buffer
697     MOV.W W1,TBLPAG
698     TBLWTL.W [W9++],[W0]     ; Load in the fourth word of the USB input buffer
699     MOV.W #0x4008,W0         ; Set up NVMCON for writing a configuration memory word
700     MOV.W W0,NVMCON
701     BRA __ICN_prog
702
703 pp_not23:
704     CP.B W0,#24              ; Is this a read memory long operation request?
705     BRA NZ,pp_not24
706     MOV.W [W9++],W3          ; Load in the second word of the USB input buffer
707     MOV.W [W9++],W1          ; Load in the third word of the USB input buffer
708     MOV.W W1,TBLPAG
709     TBLRDL.W [W3++],W1       ; Read the low memory at offsets zero and two
710     TBLRDL.W [W3],W2
711     MOV.W W1,bootaddr1       ; Save memory by reusing the address variables
712     MOV.W W2,bootaddrh
713     TBLRDH.B [W3--],W1       ; Read the high memory at offsets two and zero
714     SWAP.W W1
715     TBLRDH.B [--W3],W1
716     MOV.W W1,readhigh
717     AND.W #0x00FF,W0
718     MOV.W W0,nextreport
719     BRA __ICN_end
720
721 pp_not24:

```

```

722 CP.B W0,#25 ; Is this a jump to other bootloader operation request?
723 BRA NZ,pp_not25
724 MOV.W #0x8080,W0 ; Return the input buffer to its original condition
725 MOV.W #inbuffer,W9 ; Load the buffer address into W9
726 REPEAT #5 ; Clear out 12 bytes (maximum overall packet length)
727 MOV.W W0,[W9++]
728 GOTO bootloader
729
730 pp_not25:
731 CP.B W0,#26 ; Is this a reset processor operation request?
732 BRA NZ,pp_not26
733 RESET
734
735 pp_not26: ; If none of the above, this is a load write latches op.
736 AND.W #0x000F,W0 ; Mask off all bits but the lower four
737 SL.W W0,#2,W0 ; Multiply this address by four (access 32 instr. words)
738 ADD.W bootaddr1,WREG ; Add this to the commanded low address
739 MOV.W bootaddrh,W1
740 MOV.W W1,TBLPAG
741 MOV.W [W9++],W1 ; Load in the second word of the USB input buffer
742 TBLWTL.W W1,[W0++]
743 MOV.W [W9++],W1 ; Load in the third word of the USB input buffer
744 TBLWTL.W W1,[W0--]
745 MOV.W [W9++],W1 ; Load in the fourth word of the USB input buffer
746 TBLWTH.W W1,[W0++]
747 SWAP.W W1 ; Now access the high byte of the fourth word, to write
748 TBLWTH.W W1,[W0]
749 BRA __ICN_end
750
751 processdata: ; Start processing this DATAx packet
752 CP.B W3,#4 ; Were there only four bytes (sync/PID/CRC-16)?
753 BRA NZ,notzerolength
754
755 ;;;;;;;;;;;;;; ZERO-LENGTH PACKET ;;;;;;;;;;;;;;
756
757 ; A DATAx packet just came in, with zero data
758 BRA __ICN_end
759
760 notzerolength:
761 MOV.B [++W9],W0 ; Test the second byte of the buffer
762 CPO.B W0 ; Test the byte for zero
763 BRA Z,notdatadevhost ; If zero, this is not a "Device to Host" request
764 MOV.B [++W9],W0 ; Test the second byte of the buffer
765 SUB.B #0x06,W0 ; Is this the "Get Descriptor" data?
766 BRA NZ,notdatadesc ; If not equal, this is not a "Get Descriptor" request
767 MOV.B [++W9],W3 ; Save the third byte of the buffer (string desc. number)
768 MOV.B [++W9],W0 ; Test the fourth byte of the buffer
769 SUB.B #0x01,W0 ; Is this the "Get Device Descriptor" data?
770 BRA NZ,notdatadevd ; If not equal, this is not a "Get Device Descriptor"
771
772 ;;;;;;;;;;;;;; DEVICE DESCRIPTOR REQUEST ;;;;;;;;;;;;;;
773
774 MOV.W #nextlen,W3 ; First, read the packet lengths into nextlen
775 CLR.W TBLPAG ; Set up TBLRD.L to grab the string descriptor packets
776 MOV.W #tbloffset(dDevice),W9
777 REPEAT #7 ; Read eight bytes into nextlen
778 TBLRD.L.B [W9++],[W3++]
779 MOV.W #outbuffer,W11 ; Load the output buffer address
780 TBLRD.L.B [W9],W0 ; Load the data length and adjust for REPEAT
781 AND.W #0x00FF,W0
782 INC.W W0,W0
783 ASR.W W0,W0

```

```

784     DEC.W W0,W0
785     REPEAT W0                ; Copy up to 64 bytes of data into the output buffer
786     TBLRDL.W [W9++],[W11++]
787
788     ; A DATAx packet just came in asking for the device descriptor
789     MOV.W #outbuffer,W11 ; Load the output buffer address for later
790     BRA __ICN_end
791
792 notdatadevd:
793     MOV.B [W9],W0            ; Test the fourth byte of the buffer
794     SUB.B #0x02,W0           ; Is this the "Get Configuration Descriptor" data?
795     BRA NZ,notdataacond      ; If not equal, this is not a "Get Config. Descriptor"
796
797     ;;;;;;;;;;;;;;;;;; CONFIGURATION DESCRIPTOR REQUEST ;;;;;;;;;;;;;;;;;;
798
799     PUSH.W W9
800     MOV.W #nextlen,W3        ; First, read the packet lengths into nextlen
801     CLR.W TBLPAG             ; Set up TBLRDL to grab the string descriptor packets
802     MOV.W #tbloffset(dConfiguration),W9
803     REPEAT #7                ; Read eight bytes into nextlen
804     TBLRDL.B [W9++],[W3++]
805     MOV.W #outbuffer,W11 ; Load the output buffer address
806     TBLRDL.W [++W9],W0       ; Load the data length (actually second word) and adjust
807     INC.W W0,W0
808     ASR.W W0,W0
809     DEC.W W0,W0
810     DEC2.W W9,W9
811     REPEAT W0                ; Copy up to 64 bytes of data into the output buffer
812     TBLRDL.W [W9++],[W11++]
813     POP.W W9
814
815     INC2.W W9,W9             ; Skip the fifth and sixth bytes (zero)
816     MOV.B [++W9],W0          ; Test the seventh byte of the buffer for short/complete
817     SUB.B #0x09,W0           ; Is this the "Get Configuration Descriptor header" data?
818     BRA NZ,condescfull       ; If not equal, this is not a "Get Config. Desc. header"
819
820     ; Shorten the configuration descriptor for now; limit it to nine data bytes
821     MOV.W #8,W0              ; Set the length of data to send in bytes for next time
822     MOV.B WREG,nextlen
823     MOV.W #1,W0              ; The second packet is shorter
824     MOV.B WREG,nextlen+1
825     CLR.B nextlen+2          ; The third packet has zero data (if necessary)
826
827 condescfull:
828     ; A DATAx packet just came in asking for the configuration descriptor
829     MOV.W #outbuffer,W11 ; Load the output buffer address for later
830     BRA __ICN_end
831
832 notdataacond:
833     MOV.B [W9],W0            ; Test the fourth byte of the buffer
834     SUB.B #0x03,W0           ; Is this the "Get String Descriptor" data?
835     BRA NZ,notdatastrd       ; If not equal, this is not a "Get String Descriptor"
836
837     ;;;;;;;;;;;;;;;;;; STRING DESCRIPTOR REQUEST ;;;;;;;;;;;;;;;;;;
838
839     BSET.W PORTF,#6          ; Turn the LED on
840
841     MOV.W #outbuffer,W11 ; Load the output buffer address
842     CPO.B W3                 ; Test the saved string descriptor number
843     BRA NZ,notdatastrd0      ; If not equal, this is not a "Get String Descriptor 0"
844
845     ; This is preparing for after the IN packet comes in

```

```

846     MOV.W #0x04,W0          ; This is the string descriptor length
847     MOV.B W0,[W11++]
848     MOV.W #0x03,W0          ; This is the string descriptor type
849     MOV.B W0,[W11++]
850     MOV.W #0x0409,W0        ; This is the United States English language code
851     MOV.W W0,[W11++]
852
853     MOV.W #4,W0              ; Set the length of data to send in bytes for next time
854     MOV.B WREG,nextlen
855     BRA dontcopystr
856
857 notdatastrd0:
858     DEC.B W3,W3              ; Is this the "Get String Descriptor 1" data?
859     BRA NZ,notdatastrd1     ; If not equal, this is not a "Get String Descriptor 1"
860     MOV.W #tbloffset(sManufacturer),W9
861     BRA donestringd
862
863 notdatastrd1:
864     DEC.B W3,W3              ; Is this the "Get String Descriptor 2" data?
865     BRA NZ,notdatastrd2     ; If not equal, this is not a "Get String Descriptor 2"
866     MOV.W #tbloffset(sProduct),W9
867     BRA donestringd
868
869 notdatastrd2:
870     BRA dontcopystr          ; There was an error in asking for a string descriptor
871
872 donestringd:
873     MOV.W #nextlen,W3        ; First, read the packet lengths into nextlen
874     CLR.W TBLPAG              ; Set up TBLRDL to grab the string descriptor packets
875     REPEAT #7                 ; Read eight bytes into nextlen
876     TBLRDL.B [W9++],[W3++]
877     TBLRDL.B [W9],W0          ; Load the data length and adjust for REPEAT
878     AND.W #0x00FF,W0
879     INC.W W0,W0
880     ASR.W W0,W0
881     DEC.W W0,W0
882     REPEAT W0                 ; Copy up to 64 bytes of data into the output buffer
883     TBLRDL.W [W9++],[W11++]
884
885 dontcopystr:
886     ; A DATAx packet just came in asking for the string descriptor
887     MOV.W #outbuffer,W11 ; Load the output buffer address for later
888     BRA __ICN_end
889
890 notdatastrd:
891     MOV.B [W9],W0            ; Test the fourth byte of the buffer
892     SUB.B #0x22,W0           ; Is this the "Get HID Descriptor" data?
893     BRA NZ,notdatarepd       ; If not equal, this is not a "Get Report Descriptor"
894
895     ;;;;;;;;;;;;;;; REPORT DESCRIPTOR REQUEST ;;;;;;;;;;;;;;;
896
897     MOV.W #nextlen,W3        ; First, read the packet lengths into nextlen
898     CLR.W TBLPAG              ; Set up TBLRDL to grab the string descriptor packets
899     MOV.W #tbloffset(dReport),W9
900     REPEAT #7                 ; Read eight bytes into nextlen
901     TBLRDL.B [W9++],[W3++]
902     MOV.W #outbuffer,W11 ; Load the output buffer address
903     TBLRDL.B [++W9],W0        ; Load the data length (actually second byte) and adjust
904     AND.W #0x00FF,W0
905     INC.W W0,W0
906     ASR.W W0,W0
907     DEC.W W0,W0

```

```

908     DEC.W W9,W9
909     REPEAT W0                ; Copy up to 64 bytes of data into the output buffer
910     TBLRDL.W [W9++],[W11++]
911
912     ; A DATAx packet just came in asking for the report descriptor
913     MOV.W #outbuffer,W11 ; Load the output buffer address for later
914     BRA __ICN_end
915
916 notdatarepd:
917     BRA __ICN_end
918
919 notdatadesc:
920     BRA __ICN_end
921
922 notdatadevhost:
923     MOV.B [++W9],W0          ; Test the second byte of the buffer
924     SUB.B #0x05,W0           ; Is this the "Set Address" data?
925     BRA NZ,notdatasetadd ; If not equal, this is not a "Set Address" request
926
927     ;:::::::::::::::::::::::::::: SET ADDRESS REQUEST ::::::::::::::::::::::::::::::
928
929     MOV.B [++W9],W0          ; Read the address from the packet
930     MOV.B WREG,addrbuf       ; Store the address for this device in the buffer
931
932     CLR.B nextlen            ; Set the length of data bytes to zero for next time
933
934     ; A DATAx packet just came in, with the address
935     BRA __ICN_end
936
937 notdatasetadd:
938     MOV.B [W9],W0            ; Test the second byte of the buffer
939     SUB.B #0x09,W0           ; Is this the "Set Configuration" data?
940     BRA NZ,notdatasetcon ; If not equal, this is not a "Set Configuration" request
941
942     ;:::::::::::::::::::::::::::: SET CONFIGURATION REQUEST ::::::::::::::::::::::::::::::
943
944     MOV.B [++W9],W0          ; Read the configuration number from the packet
945     CLR.B nextlen            ; Set the length of data bytes to zero for next time
946
947     ; A DATAx packet just came in, with the configuration
948     BRA __ICN_end
949
950 notdatasetcon:
951     BRA __ICN_end
952
953     ;:::::::::::::::::::::::::::: ACKNOWLEDGEMENT TOKEN COMPARISON ::::::::::::::::::::::::::::::
954
955 notdata:
956     MOV.B [W9],W0            ; Test the first byte of the buffer
957     SUB.B #0xD2,W0           ; Compare to the ACK token at address/endpoint zero
958     BRA NZ,notack            ; If not equal, this is not an ACK packet
959
960     MOV.B nextlen+1,WREG ; Shift the buffer of data lengths to send upon request
961     MOV.B WREG,nextlen
962     MOV.B nextlen+2,WREG
963     MOV.B WREG,nextlen+1
964     MOV.B nextlen+3,WREG
965     MOV.B WREG,nextlen+2
966     MOV.B nextlen+4,WREG
967     MOV.B WREG,nextlen+3
968     MOV.B nextlen+5,WREG
969     MOV.B WREG,nextlen+4

```

```

970     MOV.B nextlen+6,WREG
971     MOV.B WREG,nextlen+5
972     MOV.B nextlen+7,WREG
973     MOV.B WREG,nextlen+6
974     CLR.B nextlen+7
975     BRA __ICN_end
976
977 notack:
978     BRA __ICN_end
979
980 badpacket:
981     BRA __ICN_end
982
983 ;;;;;;;;;;;;;;;;;; ACKNOWLEDGEMENT TRANSMIT FUNCTION ;;;;;;;;;;;;;;;;;;
984
985 sendack:
986     MOV.W #0x2000,W0      ; D+ should be low (non-inverted) since low-speed
987     MOV.W W0,PORTC        ; D- should be high (non-inverted) since low-speed
988     CLR.W TRISC           ; This will put the bus into a J (idle) state
989     REPEAT #11            ; Delay 13 cycles to make an idle J for 20 cycles
990     MOV.W #0x6000,W0      ; Load the bit locations to toggle
991     MOV.W #0xD280,W3      ; Load the acknowledgement reply
992
993     DO #15,sendack_end    ; Send 16 bits of the sync/PID
994     RRNC.W W3,W3          ; Rotate the word right to send LSB first as specified
995     BTSS.W SR,#N          ; If the wrapped-around LSB was one, don't toggle PORTC
996     XOR.W PORTC           ; Toggle the pins together for a zero bit
997     SENDTOKEN             ; Insert the SENDTOKEN macro (eight cycles)
998     REPEAT #6             ; Delay eight more cycles for a total of 20
999     NOP
1000 sendack_end:
1001     NOP
1002
1003     NOP                  ; Delay twice more for 20 cycles total
1004     NOP
1005     CLR.W PORTC          ; This creates a single-ended zero with RC14 and RC13
1006     SENDTOKEN            ; Insert the SENDTOKEN macro (eight cycles)
1007     REPEAT #21           ; Delay 23 more cycles (nearly two bit lengths)
1008     NOP
1009     SENDTOKEN            ; Insert the SENDTOKEN macro (eight cycles)
1010     BSET.W PORTC,#13     ; D- should be high (non-inverted) since low-speed
1011     REPEAT #8            ; Delay 10 cycles (nearly one bit length)
1012     NOP
1013     SENDTOKEN            ; Insert the SENDTOKEN macro (eight cycles)
1014     MOV.W #0x6000,W0      ; Load the original TRISC value
1015     MOV.W W0,TRISC       ; This will release the bus to the host
1016     BSET.W PORTC,#15     ; Turn the 74LS244 to input from the bus
1017     RETURN
1018
1019 ;;;;;;;;;;;;;;;;;; DATA TRANSMIT FUNCTION ;;;;;;;;;;;;;;;;;;
1020
1021 senddata_wait:          ; This label can be branched to, to delay before sending
1022     SENDTOKEN            ; Insert the SENDTOKEN macro (eight cycles)
1023     REPEAT #14           ; Delay a short amount of time
1024     NOP
1025     SENDTOKEN            ; Insert the SENDTOKEN macro (eight cycles)
1026     REPEAT #14           ; Delay a short amount of time
1027     NOP
1028     SENDTOKEN            ; Insert the SENDTOKEN macro (eight cycles)
1029     REPEAT #14           ; Delay a short amount of time
1030     NOP
1031     SENDTOKEN            ; Insert the SENDTOKEN macro (eight cycles)

```

```

1032 REPEAT #14 ; Delay a short amount of time
1033 NOP
1034 SENDTOKEN ; Insert the SENDTOKEN macro (eight cycles)
1035
1036 senddata:
1037 MOV.W #0x8088,W0 ; Load in the data-toggle constant and the sync byte
1038 XOR.B lastdata,WREG ; Toggle the last DATAx bits for this transfer
1039 MOV.B WREG,lastdata ; Save this result for next time
1040 SWAP.W W0 ; Swap the bytes of W0 in order to transmit properly
1041 MOV.W W0,W9 ; This word must be moved into W9
1042 MOV.W #1,W2 ; Keep track of how many ones in a row for stuffing
1043 BTSC.W W9,#15 ; If we're sending a 0xC380 instead of a 0x4B80, the
1044 INC.W W2,W2 ; bitstuff count should be two instead of one
1045 MOV.W #0x2000,W0 ; D+ should be low (non-inverted) since low-speed
1046 MOV.W W0,PORTC ; D- should be high (non-inverted) since low-speed
1047 CLR.W TRISC ; This will put the bus into a J (idle) state
1048 MOV.W #0x6000,W0 ; Load the bit locations to toggle
1049
1050 DO #15,senddata_end0 ; Send 16 bits of the sync/PID
1051 SENDTOKEN ; Insert the SENDTOKEN macro (eight cycles)
1052 REPEAT #3 ; Delay five more cycles for a total of 20
1053 NOP
1054 RRNC.W W9,W9 ; Rotate the word right to send LSB first as specified
1055 BTSS.W SR,#N ; If the wrapped-around LSB was one, don't toggle PORTC
1056 XOR.W PORTC ; Toggle the pins together for a zero bit
1057 NOP ; For some reason, a REPEAT here takes one cycle too many
1058 NOP ; Instead of a REPEAT #1, use three NOPs and a final
1059 NOP
1060 senddata_end0:
1061 NOP
1062
1063 SETM.W W9 ; Set W9 to all ones; this is the CRC "shift register"
1064 CPO.W W3 ; Should only the CRC be sent? Check the bit count
1065 BRA Z,senddata_crcz ; This will branch in the case of a zero-data packet
1066 SL.W W3,#3,W3 ; Turn the byte count into a bit count
1067 DEC.W W3,W3 ; Decrement it by one because the DO loop needs this
1068 CLR.W W1 ; Clear the bit count (actually shifted left by five)
1069 MOV.W #0xA001,W8 ; Set W8 to the CRC-16 polynomial
1070
1071 DO W3,senddata_end1 ; Send W3+1 bits total
1072 LSR.W W9,W9 ; Shift the checksum right by one bit into carry
1073 RRNC.B [W11],[W11] ; Rotate the byte right to send LSB first as specified
1074 BRA N,sd_notzero1 ; If the wrapped-around LSB was one, don't toggle RB7/RB6
1075 CLR.W W2 ; Clear the stuffing bit count since a zero was sent
1076 BTSC.W SR,#C ; Is the previous CRC low bit one?
1077 XOR.W W9,W8,W9 ; XOR the words together if the bits differ (zero vs one)
1078 XOR.W PORTC ; Toggle the pins together for a zero bit
1079 NOP ; Equalize the zero/one delays
1080 BRA sd_continue1 ; Skip all the next instructions
1081 sd_notzero1:
1082 BTSS.W SR,#C ; Is the previous CRC low bit zero?
1083 XOR.W W9,W8,W9 ; XOR the words together if the bits differ (zero vs one)
1084 INC.W W2,W2 ; This bit of one may create a need for stuffing
1085 CP.W W2,#6 ; Test to see if stuffing is needed (six ones in a row)
1086 BRA NZ,sd_continue1 ; If the comparison was nonzero, continue
1087 SENDTOKEN ; Insert the SENDTOKEN macro (eight cycles)
1088 REPEAT #7 ; Still send out a one first (delay nine cycles)
1089 NOP
1090 XOR.W PORTC ; Toggle the pins together for a stuffed zero bit
1091 CLR.W W2 ; Clear the stuffing bit count since a zero was sent
1092 REPEAT #1 ; Delay three more cycles to equalize with a normal zero
1093 NOP

```

```

1094 sd_continue1:
1095     SENDTOKEN                ; Insert the SENDTOKEN macro (eight cycles)
1096     ADD.B #32,W1             ; Add '00100000' to W1 to test for the next buffer byte
1097 senddata_end1:              ; This label points to the last instruction in the loop
1098     ADDC.W #0,W11            ; If it carries, point to the next byte in the buffer
1099
1100 senddata_crc:               ; Note that the CRC is inverted at this point
1101     DO #15,senddata_end2     ; Send 16 bits of the CRC
1102     RRNC.W W9,W9             ; Rotate the byte right to send LSB first as specified
1103     BRA NN,sd_notzero2       ; If the wrapped-around LSB was one, don't toggle RB7/RB6
1104     CLR.W W2                 ; Clear the stuffing bit count since a zero was sent
1105     NOP                     ; Equalize the zero/one delays
1106     XOR.W PORTC              ; Toggle the pins together for a zero bit
1107     BRA sd_continue2        ; Skip all the next instructions
1108
1109 sd_notzero2:
1110     INC.W W2,W2              ; This bit of one may create a need for stuffing
1111     CP.W W2,#6               ; Test to see if stuffing is needed (six ones in a row)
1112     BRA NZ,sd_continue2     ; If the comparison was nonzero, continue
1113     SENDTOKEN                ; Insert the SENDTOKEN macro (eight cycles)
1114     REPEAT #8                ; Still send out a one first (delay 10 cycles)
1115     NOP
1116     XOR.W PORTC              ; Toggle the pins together for a stuffed zero bit
1117     CLR.W W2                 ; Clear the stuffing bit count since a zero was sent
1118     NOP                     ; Delay one more cycle to equalize with a normal zero
1119 sd_continue2:
1120     NOP
1121     NOP
1122     SENDTOKEN                ; Insert the SENDTOKEN macro (eight cycles)
1123     NOP
1124     NOP
1125 senddata_end2:              ; This label points to the last instruction in the loop
1126     NOP
1127
1128     REPEAT #2                ; Delay four cycles for 20 cycles total
1129     NOP
1130     CLR.W PORTC              ; This creates a single-ended zero with RC14 and RC13
1131     SENDTOKEN                ; Insert the SENDTOKEN macro (eight cycles)
1132     REPEAT #21               ; Delay 23 more cycles (nearly two bit lengths)
1133     NOP
1134     SENDTOKEN                ; Insert the SENDTOKEN macro (eight cycles)
1135     BSET.W PORTC,#13         ; D- should be high (non-inverted) since low-speed
1136     REPEAT #8                ; Delay 10 cycles (nearly one bit length)
1137     NOP
1138     SENDTOKEN                ; Insert the SENDTOKEN macro (eight cycles)
1139     MOV.W #0x6000,W0         ; Load the original TRISC value
1140     MOV.W W0,TRISC           ; This will release the bus to the host
1141     BSET.W PORTC,#15         ; Turn the 74LS244 to input from the bus
1142     BRA __ICN_end
1143
1144 senddata_crcz:
1145     NOP                     ; Delay five cycles before sending the zero CRC
1146     NOP
1147     NOP
1148     BRA senddata_crc
1149
1150 resetit:
1151     BCLR.W PORTF,#6          ; Turn the LED off
1152     RCALL reset_usb          ; Re-initialize all the USB variables
1153     MOV.W PORTC,W0           ; Clear the notification condition
1154     BCLR.W IFS0,#CNIF        ; Clear the CN interrupt flag
1155     POP.W W9                 ; Restore the working register W9

```



```

1156     POP.W W8                ; Restore the working register W8
1157     POP.S                   ; Restore the status register, W0, W1, W2, and W3
1158     RETFIE
1159
1160 __ICN_prog:
1161     DISI #5                  ; Execute a generic programming cycle for many operations
1162     MOV.W #0x55,W0
1163     MOV.W W0,NVMKEY
1164     MOV.W #0xAA,W1
1165     MOV.W W1,NVMKEY
1166     BSET.W NVMCON,#WR
1167     NOP
1168     NOP
1169
1170 __ICN_end:
1171     MOV.W #0x8080,W0        ; Return the input buffer to its original condition
1172     MOV.W #inbuffer,W9      ; Load the buffer address into W9
1173     REPEAT #5               ; Clear out 12 bytes (maximum overall packet length)
1174     MOV.W W0,[W9++]
1175     MOV.W PORTC,W0          ; Clear the notification condition
1176     BCLR.W IFS0,#CNIF      ; Clear the CN interrupt flag
1177     POP.W W9                ; Restore the working register W9
1178     POP.W W8                ; Restore the working register W8
1179     POP.S                   ; Restore the status register, W0, W1, W2, and W3
1180     RETFIE
1181
1182
1183 ; USB descriptors (0-32 words real data)
1184
1185 bl_dDevice:
1186     .WORD 0x0808            ; The nextlen packet lengths (LSB first)
1187     .WORD 0x0002
1188     .WORD 0x0000
1189     .WORD 0x0000
1190     .WORD 0x0112            ; The device descriptor length in bytes and number
1191     .WORD 0x0110            ; The USB specification number (version 1.1)
1192     .WORD 0x0000            ; The device class/subclass (specified in interf. desc.)
1193     .WORD 0x0800            ; The device protocol and the maximum packet size
1194     .WORD 0x04D8            ; The vendor ID (from Microchip)
1195     .WORD 0xABCD            ; The product ID
1196     .WORD 0x0100            ; The device release number
1197     .WORD 0x0201            ; The manufacturer/product string descriptor addresses
1198     .WORD 0x0100            ; The serial number string addr. and configuration count
1199
1200 bl_dConfiguration:
1201     .WORD 0x0808            ; The nextlen packet lengths (LSB first)
1202     .WORD 0x0808
1203     .WORD 0x0108
1204     .WORD 0x0000
1205     .WORD 0x0209            ; The config. descriptor length in bytes and number
1206     .WORD 0x0029            ; The total length of bytes returned
1207     .WORD 0x0101            ; The number of interfaces and the configuration value
1208     .WORD 0xC000            ; The config. string address and the config. attributes
1209     .WORD 0x0980            ; The maximum current (by 2mA) and interface desc. length
1210     .WORD 0x0004            ; The interface descriptor number and which one this is
1211     .WORD 0x0200            ; The alternate setting number and endpoint count
1212     .WORD 0x0003            ; The interface class (HID) and subclass
1213     .WORD 0x0000            ; The interface protocol and string descriptor address
1214     .WORD 0x2109            ; The HID descriptor length in bytes and number
1215     .WORD 0x0110            ; The HID specification number (version 1.1)
1216     .WORD 0x0100            ; The country code and the number of subordinate descs.
1217     .WORD 0x2D22            ; The report descriptor length in bytes and number

```

```

1218 .WORD 0x0700 ; The report desc. length MSB and IN interrupt desc. len.
1219 .WORD 0x8105 ; The IN int. desc. number and endpoint number/direction
1220 .WORD 0x0803 ; The transfer type and maximum packet size LSB
1221 .WORD 0x0A00 ; The maximum packet size MSB and polling interval
1222 .WORD 0x0507 ; The OUT int. desc. length and number
1223 .WORD 0x0301 ; The endpoint number and direction and transfer type
1224 .WORD 0x0008 ; The maximum packet size MSB/LSB
1225 .WORD 0x000A ; The polling interval in milliseconds
1226
1227 bl_dReport:
1228 .WORD 0x0808 ; The nextlen packet lengths (LSB first)
1229 .WORD 0x0808
1230 .WORD 0x0808
1231 .WORD 0x0010
1232 .WORD 0x2D22 ; The report descriptor length in bytes and number
1233 .WORD 0x0500
1234 .WORD 0x0984
1235 .WORD 0xA159
1236 .WORD 0x0901
1237 .WORD 0x151A
1238 .WORD 0x2600
1239 .WORD 0x00FF
1240 .WORD 0x0895
1241 .WORD 0x0875
1242 .WORD 0x0281
1243 .WORD 0x1C09
1244 .WORD 0x0015
1245 .WORD 0xFF26
1246 .WORD 0x9500
1247 .WORD 0x7508
1248 .WORD 0x9108
1249 .WORD 0x0902
1250 .WORD 0x151E
1251 .WORD 0x2600
1252 .WORD 0x00FF
1253 .WORD 0x0895
1254 .WORD 0x0875
1255 .WORD 0x02B1
1256 .WORD 0x00C0
1257
1258
1259 ; USB strings (2-32 words real data, including 0x03xx header; 1-31 characters)
1260
1261 bl_sManufacturer:
1262 .WORD 0x0808 ; The nextlen packet lengths (LSB first)
1263 .WORD 0x0808
1264 .WORD 0x0808
1265 .WORD 0x0000
1266 .WORD 0x0330 ; The string descriptor number and the length in bytes
1267 .WORD 'C','u','s','t','o','m',' ','A','u','t','o',' '
1268 .WORD 'E','l','e','c','t','r','o','n','i','c','s'
1269
1270 bl_sProduct:
1271 .WORD 0x0808 ; The nextlen packet lengths (LSB first)
1272 .WORD 0x0808
1273 .WORD 0x0608
1274 .WORD 0x0000
1275 .WORD 0x032E ; The string descriptor number and the length in bytes
1276 .WORD 'E','V',' ','M','u','l','t','i','l','e','v','e','l',' '
1277 .WORD 'I','n','v','e','r','t','e','r'

```

E.2 PIC18F2553 Full Assembly Code, Design Two

```
1 ;
2 ; USB Interface Code for the EV Multilevel Inverter
3 ; Arthur W. Matteson - 2/8/2008
4 ;
5
6 LIST P=PIC18F2550 ; Set the processor type
7
8 #INCLUDE p18f2550.inc
9
10
11 ; Variables
12 loopval0 EQU 0x00
13 loopval1 EQU 0x01
14 savedw EQU 0x02
15 newaddr EQU 0x03
16 savedl EQU 0x04
17 savedh EQU 0x05
18 ledonl EQU 0x06
19 ledonh EQU 0x07
20
21
22 ORG 0x0000 ; Start at the reset vector
23 BRA main ; We don't need to do anything else
24 ORG 0x0008 ; Start at the high priority interrupt vector
25 BRA isr ; Point to our interrupt service routine
26 ORG 0x0018 ; Start at the low priority interrupt vector
27 BRA isr ; Point to our interrupt service routine
28
29 main:
30 RCALL delay_50ms ; Delay if we're programming so pins don't short
31 RCALL delay_50ms
32
33 MOVLW B'01111111' ; PORTA data direction register
34 MOVWF TRISA,A
35 MOVLW B'11101111' ; PORTB data direction register
36 MOVWF TRISB,A
37 MOVLW B'10000111' ; PORTC data direction register
38 MOVWF TRISC,A
39 MOVLW B'00000000' ; PORTA output data
40 MOVWF PORTA,A
41 MOVLW B'00000000' ; PORTB output data
42 MOVWF PORTB,A
43 MOVLW B'00000000' ; PORTC output data
44 MOVWF PORTC,A
45
46 MOVLW B'00000101' ; Set the serial port up for 2MBd
47 MOVWF SPBRG,A
48 CLRF SPBRGH,A
49 MOVLW B'00001000' ; Set up the 16-bit baud rate generator
50 MOVWF BAUDCON,A
51 MOVLW B'01100100' ; Enable serial port transmissions
52 MOVWF TXSTA,A
53 MOVLW B'11010000' ; Enable serial port receptions
54 MOVWF RCSTA,A
55 BSF PIE1,RCIE,A ; Enable interrupts upon reception
56 BSF INTCON,PEIE,A
57 BSF INTCON,GIE,A
58
```

```

59   CLRF ledonl,A           ; Clear the LED on-time counter
60   CLRF ledonh,A
61
62   LFSR 0,0x0000           ; Clear out the RAM beginning at 0
63   clearout:
64       CLRF INDF0,A         ; Clear out the data location specified by FSRO
65       INCFSR FSR0L,F,A     ; Increment the address to clear
66       BTFSC STATUS,Z,A     ; Was there an overflow?
67       INCFSR FSR0H,F,A     ; Increment the high address as well
68       BTFSS FSR0H,3,A      ; Test for an address of 0x0800
69       BRA clearout         ; Branch if not there yet
70
71       MOVLW B'00010100'    ; Set up high-speed operation
72       MOVWF UCFG,A
73       MOVLW B'10011111'    ; Enable all error reporting
74       MOVWF UEIE,A
75       MOVLW B'00001000'    ; Enable USB and don't suspend
76       MOVWF UCON,A
77
78       LFSR 2,0x0600        ; Load FSR2 with the data stream block address
79
80   mainloop:
81       BCF UCON,SUSPND,A     ; Always take the USB module out of suspend
82
83       DECFSZ ledonl,F,A     ; Decrement the counter
84       BRA noledoff
85       DCFSNZ ledonh,F,A
86       BCF PORTB,4,A        ; Turn the LED off when the counter expires
87   noledoff:
88
89       BTFSS UIR,URSTIF,A    ; Test the USB reset interrupt flag
90       BRA notreset
91
92       CLRF UIR,A            ; Clear the USB interrupt flag(s)
93       CLRF UIR,A            ; Clear the USB interrupt flag(s)
94       CLRF UIR,A            ; Clear the USB interrupt flag(s)
95       CLRF UIR,A            ; Clear the USB interrupt flag(s)
96       CLRF UIR,A            ; Clear the USB interrupt flag(s)
97       MOVLB 0x04            ; Access bank 4 (USB RAM buffer descriptors)
98
99       MOVLW 0x40             ; Set up BDOCNT for 64 bytes of buffer
100      MOVWF 0x01,B
101      MOVLW 0x00              ; Set up BDOADRL to point to 0x500
102      MOVWF 0x02,B
103      MOVLW 0x05              ; Set up BDOADRH to point to 0x500
104      MOVWF 0x03,B
105      MOVLW B'10000000'      ; Set up BD0STAT for 64 bytes of buffer
106      MOVWF 0x00,B           ; This descriptor is for EPO OUT (control)
107      CLRF 0x05,B            ; Clear the BD1CNT buffer length (no sending yet)
108      MOVLW 0x40              ; Set up BD1ADRL to point to 0x540
109      MOVWF 0x06,B
110      MOVLW 0x05              ; Set up BD1ADRH to point to 0x540
111      MOVWF 0x07,B
112      MOVLW B'00000000'      ; Set up BD1STAT for 64 bytes of buffer
113      MOVWF 0x04,B           ; This descriptor is for EPO IN (control)
114      MOVLW B'00010110'      ; Set up endpoint zero (with control transfers)
115      MOVWF UEPO,A
116
117      MOVLW 0x40              ; Set up BD2CNT for 64 bytes of buffer
118      MOVWF 0x09,B
119      MOVLW 0x80              ; Set up BD2ADRL to point to 0x680
120      MOVWF 0x0A,B

```

```

121    MOVLW 0x06                ; Set up BD2ADRH to point to 0x680
122    MOVWF 0x0B,B
123    MOVLW B'10000000'        ; Set up BD2STAT for 64 bytes of buffer
124    MOVWF 0x08,B            ; This descriptor is for EP1 OUT (DSPIC control)
125    MOVLW 0x40                ; Set up BD3CNT for 64 bytes of buffer
126    MOVWF 0x0D,B
127    MOVLW 0x00                ; Set up BD3ADRL to point to 0x600
128    MOVWF 0x0E,B
129    MOVLW 0x06                ; Set up BD3ADRH to point to 0x600
130    MOVWF 0x0F,B
131    MOVLW B'10000000'        ; Set up BD3STAT for 64 bytes of buffer
132    MOVWF 0x0C,B            ; This descriptor is for EP1 IN (data stream)
133    MOVLW B'00011110'        ; Set up endpoint one (no control transfers)
134    MOVWF UEP1,A
135
136    MOVLW 0x40                ; Set up BD4CNT for 64 bytes of buffer
137    MOVWF 0x11,B
138    MOVLW 0xC0                ; Set up BD4ADRL to point to 0x6C0
139    MOVWF 0x12,B
140    MOVLW 0x06                ; Set up BD4ADRH to point to 0x6C0
141    MOVWF 0x13,B
142    MOVLW B'10000000'        ; Set up BD4STAT for 64 bytes of buffer
143    MOVWF 0x10,B            ; This descriptor is for EP2 OUT (DSPIC control)
144    MOVLW 0x40                ; Set up BD5CNT for 64 bytes of buffer
145    MOVWF 0x15,B
146    MOVLW 0x40                ; Set up BD5ADRL to point to 0x640
147    MOVWF 0x16,B
148    MOVLW 0x06                ; Set up BD5ADRH to point to 0x640
149    MOVWF 0x17,B
150    MOVLW B'10000000'        ; Set up BD5STAT for 64 bytes of buffer
151    MOVWF 0x14,B            ; This descriptor is for EP2 IN (DSPIC control)
152    MOVLW B'00011110'        ; Set up endpoint one (no control transfers)
153    MOVWF UEP2,A
154
155    CLRF newaddr,A            ; Clear the buffered device address
156    CLRF UADDR,A            ; Clear the actual device address
157    BRA mainloop
158
159    notreset:
160    BTFSS UIR,SOFIF,A        ; Test the USB start-of-frame interrupt flag
161    BRA notsof
162
163    BCF UIR,SOFIF,A          ; Clear the USB interrupt flag
164    BCF UCON,PKTDIS,A        ; Take the USB module out of packet disable
165    BRA mainloop
166
167    notsof:
168    BTFSS UIR,TRNIF,A        ; Test the USB transaction interrupt flag
169    BRA nottran
170
171    BSF PORTB,4,A            ; Turn the LED on
172    CLRF ledonl,A            ; Set the counter to 4096
173    MOVLW 0x10
174    MOVWF ledonh,A
175
176    MOVLW 0x04                ; Select the INDF0 to be USB buf. desc. RAM space
177    MOVWF FSROH,A
178    MOVF USTAT,W,A
179    BCF UIR,TRNIF,A          ; Clear the interrupt flag after reading USTAT
180    ANDLW B'01111100'        ; Find the offset address of the endpoint
181    MOVWF FSROL,A
182    MOVF INDF0,W,A            ; Check the BDnSTAT byte for the token bits

```

```

183     ANDLW B'00111100'
184
185     ADDLW B'11111100'           ; Was this an OUT token transaction?
186     BNZ tran_notout
187
188     MOVLW 0x00
189     CPFSEQ FSR0L,A             ; Is this endpoint 0 OUT?
190     BRA copy_out              ; If not, FSR0L should be 0x08 or 0x10
191     BRA no_copy_out
192
193 ; Possibly endpoint 1 or 2 (interrupt OUT)
194
195 copy_out:
196     MOVF FSR0L,W,A             ; Save the FSR0 low byte for later
197     MOVWF savedl,A
198     MOVF FSR0H,W,A             ; Save the FSR0 high byte for later
199     MOVWF savedh,A
200
201     LFSR 1,0x0680              ; Load FSR1 with the new data block address
202     INCF FSR0L,F,A             ; Access BD2ADRL/BD4ADRL after the pre-increment
203     MOVF PREINC0,W,A           ; Recall the low address byte and store in FSR1L
204     MOVWF FSR1L,A              ; Now, this newly received data can be sent out
205     BCF TXSTA,TX9D,A           ; A clear ninth bit means this is an address byte
206     MOVWF TXREG,A              ; Write this start address to the transmit buffer
207
208     MOVLW 0x40                 ; Transfer 64 bytes to the DSPIC buffer
209     MOVWF savedw,A             ; Save the working register (byte count)
210     BSF TXSTA,TX9D,A           ; A set ninth bit means this is a data byte
211 out_loop:
212     BTFSS TXSTA,TRMT,A         ; Wait until the transmit buffer is empty
213     BRA out_loop
214     MOVF POSTINC1,W,A           ; Recall the USB-sent value in memory (0x0680...)
215     MOVWF TXREG,A              ; Write this to the transmit buffer
216     DECFSZ savedw,F,A          ; Decrement the loop counter
217     BRA out_loop
218
219     MOVF savedl,W,A             ; Restore the FSR0 low byte for BDnSTAT
220     MOVWF FSR0L,A
221     MOVF savedh,W,A             ; Restore the FSR0 high byte for BDnSTAT
222     MOVWF FSR0H,A
223
224 no_copy_out:
225     MOVLB 0x04                 ; Access bank 4 (USB RAM buffer descriptors)
226     MOVLW B'10000000'          ; Set up BDnSTAT for 64 bytes of buffer
227     MOVWF POSTINC0,A           ; This descriptor is for EPO/EP1/EP2 OUT
228     MOVLW 0x40                 ; Set up BDnCNT for 64 bytes of buffer
229     MOVWF INDF0,A
230     BCF UCON,PKTDIS,A          ; Take the USB module out of packet disable
231     BRA mainloop
232
233 tran_notout:
234     ADDLW B'11100000'          ; Was this an IN token transaction?
235     BNZ tran_notin
236
237 ; Possibly endpoint 1 or 2 (interrupt IN)
238
239     MOVLW 0x04
240     CPFSEQ FSR0L,A             ; Is this endpoint 0 IN?
241     BRA tran_in_toggle         ; If not, FSR0L should be 0x0C or 0x14
242
243     MOVF newaddr,W,A           ; Set the new address in the status stage (IN)
244     MOVWF UADDR,A

```

```

245     BCF UCON,PKTDIS,A           ; Take the USB module out of packet disable
246     MOVLB 0x04                 ; Access bank 4 (USB RAM buffer descriptors)
247     CLRF 0x05,B               ; Write the zero length out to EPO IN's B.D.
248     MOVLW B'11000000'         ; Set up BDOSTAT for sending data (just in case)
249     MOVWF 0x04,B              ; This descriptor is for EPO IN
250     BRA mainloop
251
252 tran_in_toggle:
253     MOVLW 0x40                 ; Set up BD3CNT/BD5CNT for 64 bytes of buffer
254     MOVWF PREINCO,A           ; Write this to 0x0D or 0x15 (BD3CNT/BD5CNT)
255     MOVWF 0x15,B
256     DECF FSR0L,F,A            ; Write next to 0x0C or 0x14 (BD3STAT/BD5STAT)
257     MOVLW B'01000000'         ; Clear out all bits but the data toggle bit
258     ANDWF INDF0,F,A
259     BTG INDF0,6,A              ; Toggle the data toggle synchronization bit
260     BSF INDF0,7,A              ; Make sure the USB peripheral can read the data
261     BCF UCON,PKTDIS,A         ; Take the USB module out of packet disable
262     BRA mainloop
263
264 tran_notin:
265     MOVF PREINCO,W,A           ; This must have been a SETUP token transaction
266     MOVLB 0x05                 ; Check the BDnCNT byte for the packet size
267     MOVF 0x01,W,B              ; Read the bmRequestType and bRequest bytes
268     BTFSC 0x00,7,B             ; Recall the bRequest byte into W
269     BRA tran_get               ; Was it a SET_xxx request?
270                                ; If not, it was a GET_xxx request
271     ADDLW 0xFB                 ; Is this a SET_ADDRESS request?
272     BNZ tran_notsetaddress
273
274     MOVF 0x02,W,B              ; This is the new device address
275     MOVWF newaddr,A            ; Don't set it quite yet
276     MOVLB 0x04                 ; Access bank 4 (USB RAM buffer descriptors)
277     CLRF 0x05,B               ; Write the zero length out to EPO IN's B.D.
278     MOVLW B'11000000'         ; Set up BDOSTAT for sending data
279     MOVWF 0x04,B              ; This descriptor is for EPO IN
280     BRA tran_setupdone         ; EPO IN should now be set up to send out data
281
282 tran_notsetaddress:
283     ADDLW 0xFC                 ; Is this a SET_CONFIGURATION request?
284     BNZ tran_notsetconfig
285
286     MOVLB 0x04                 ; Access bank 4 (USB RAM buffer descriptors)
287     CLRF 0x05,B               ; Write the zero length out to EPO IN's B.D.
288     MOVLW B'11000000'         ; Set up BDOSTAT for sending data
289     MOVWF 0x04,B              ; This descriptor is for EPO IN
290     BRA tran_setupdone         ; EPO IN should now be set up to send out data
291
292 tran_notsetconfig:
293     BRA tran_setupdone
294
295 tran_get:
296     ADDLW 0xFA                 ; Is this a GET_DESCRIPTOR request?
297     BNZ tran_notgetdesc
298
299     MOVF 0x03,W,B              ; Recall the wValue high byte into W (desc. type)
300     ADDLW 0xFF                 ; Is this a GET_DEVICE_DESCRIPTOR request?
301     BNZ tran_notgetdd
302
303     MOVLW 0x10                 ; Access the device descriptor in Flash
304     MOVWF TBLPTRH,A
305     BRA tran_get_copy
306

```



```

307 tran_notgetdd:
308     ADDLW 0xFF                                ; Is this a GET_CONFIGURATION_DESCRIPTOR request?
309     BNZ tran_notgetcd
310
311     MOVLW 0x11                                ; Access the configuration descriptor in Flash
312     MOVWF TBLPTRH,A
313     BRA tran_get_copy
314
315 tran_notgetcd:
316     ADDLW 0xFF                                ; Is this a GET_STRING_DESCRIPTOR request?
317     BNZ tran_notgetsd
318
319     MOVF 0x02,W,B                             ; Recall the wValue low byte into W (string num.)
320     BNZ tran_sd_not0                          ; Is this a "Get String Descriptor 0" request?
321
322     MOVLW 0x12                                ; Access the string descriptor zero in Flash
323     MOVWF TBLPTRH,A
324     BRA tran_get_copy
325
326 tran_sd_not0:
327     ADDLW 0xFF                                ; Is this a "Get String Descriptor 1" request?
328     BNZ tran_sd_not1
329
330     MOVLW 0x13                                ; Access the string descriptor one in Flash
331     MOVWF TBLPTRH,A
332     BRA tran_get_copy
333
334 tran_sd_not1:
335     ADDLW 0xFF                                ; Is this a "Get String Descriptor 2" request?
336     BNZ tran_sd_not2
337
338     MOVLW 0x14                                ; Access the string descriptor two in Flash
339     MOVWF TBLPTRH,A
340     BRA tran_get_copy
341
342 tran_sd_not2:
343 tran_notgetsd:
344 tran_notgetdesc:
345 tran_setupdone:
346     MOVLW 0x40                                ; Set up BDnCNT for 64 bytes of buffer
347     MOVWF POSTDECO,A
348     MOVLW B'10000000'                        ; Set up BDOSTAT for 64 bytes of buffer
349     MOVWF INDF0,A                             ; This descriptor is for EPO OUT
350     BCF UCON,PKTDIS,A                        ; Take the USB module out of packet disable
351     BRA mainloop
352
353 tran_get_copy:
354     MOVLW 0x05                                ; Select INDF1 to be EPO's IN space for outputting
355     MOVWF FSR1H,A
356     MOVLW 0x40
357     MOVWF FSR1L,A
358     CLRF TBLPTRU,A                             ; Access the descriptor in Flash
359     CLRF TBLPTRL,A
360     TBLRD++                                    ; Read the descriptor length byte
361     MOVF TABLAT,W,A
362     TBLRD++                                    ; Skip past the length high byte
363     MOVWF savedw,A
364     MOVF 0x06,W,B                             ; Don't send more bytes than the request asked
365     CPFSGT savedw,A
366     MOVF savedw,W,A
367     MOVLB 0x04                                ; Access bank 4 (USB RAM buffer descriptors)
368     MOVWF 0x05,B                             ; Write the packet length out to EPO IN's B.D.

```

```

369     RCALL copy_data           ; Copy the length of bytes in W to INDF1
370     MOVLW B'11000000'        ; Set up BD1STAT for sending data
371     MOVWF 0x04,B              ; This descriptor is for EPO IN
372     BRA tran_setupdone        ; EPO IN should now be set up to send out data
373
374 nottran:
375 noterror:
376     BRA mainloop
377
378
379 blink:
380     BSF PORTB,4,A              ; Turn the LED on
381     RCALL delay_50ms
382     RCALL delay_50ms
383     RCALL delay_50ms
384     RCALL delay_50ms
385     BCF PORTB,4,A              ; Turn the LED off
386     RCALL delay_50ms
387     RCALL delay_50ms
388     RCALL delay_50ms
389     RCALL delay_50ms
390     BRA blink
391
392
393 copy_data:
394     MOVWF savedw,A             ; Copy the length of bytes in W to INDF1
395     cd_loop:
396         TBLRD**                ; Read the Flash byte value
397         MOVF TABLAT,W,A         ; Recall the read value
398         MOVWF POSTINC1,A        ; Write this to the output buffer
399         DECFSZ savedw,F,A       ; Decrement the loop counter
400         BRA cd_loop
401         RETURN
402
403
404 delay_100us:
405     MOVLW D'239'               ; Our cycles are 0.0833us long (48MHz/4)
406     MOVWF loopval0,A
407     here_100us:
408         NOP                     ; These make it 5 cycles total
409         NOP
410         DECFSZ loopval0,F,A
411         BRA here_100us
412         RETURN
413
414
415 delay_5ms:
416     MOVLW D'50'                ; 100us*50=5ms
417     MOVWF loopval1,A
418     here_5ms:
419         RCALL delay_100us
420         DECFSZ loopval1,F,A
421         BRA here_5ms
422         RETURN
423
424
425 delay_50ms:
426     MOVLW D'250'               ; 200us*250=50ms
427     MOVWF loopval1,A
428     here_50ms:
429         RCALL delay_100us
430         RCALL delay_100us

```

```

431     DECFSZ loopval1,F,A
432     BRA here_50ms
433     RETURN
434
435
436     isr:
437         BCF PIR1,RCIF,A                ; Clear the received byte interrupt flag
438         MOVF RCREG,W,A                ; Recall the received byte
439         BTFSS RCSTA,RX9D,A            ; Was the received byte a data byte?
440         BRA setaddress                ; If not, it was an FSR2L address
441         MOVWF POSTINC2,A              ; Store the byte at the indirect pointer address
442         RETFIE 1                      ; Restore the contents of W and STATUS
443     setaddress:
444         LFSR 2,0x0600                ; Load FSR2 with the data stream block address
445         MOVWF FSR2L,A                ; Store it in the indirect address low pointer
446     isrdone:
447         RETFIE 1                      ; Restore the contents of W and STATUS
448
449
450     ORG 0x001000
451     DW 0x0012                        ; The device descriptor is 18 bytes long
452     DW 0x0112                        ; The device descriptor length in bytes and number
453     DW 0x0110                        ; The USB specification number (version 1.1)
454     DW 0x0000                        ; The device class/subclass (specified in interf. desc.)
455     DW 0x4000                        ; The device protocol and the maximum packet size
456     DW 0x04D8                        ; The vendor ID (from Microchip)
457     DW 0xDCBA                        ; The product ID
458     DW 0x0100                        ; The device release number
459     DW 0x0201                        ; The manufacturer/product string descriptor addresses
460     DW 0x0100                        ; The serial number string addr. and configuration count
461
462     ORG 0x001100
463     DW 0x002E                        ; The configuration descriptor is 46 bytes as returned
464     DW 0x0209                        ; The config. descriptor length in bytes and number
465     DW 0x002E                        ; The total length of bytes returned
466     DW 0x0101                        ; The number of interfaces and the configuration value
467     DW 0xC000                        ; The config. string address and the config. attributes
468     DW 0x0980                        ; The maximum current (by 2mA) and interface desc. length
469     DW 0x0004                        ; The interface descriptor number and which one this is
470     DW 0x0400                        ; The alternate setting number and endpoint count
471     DW 0x00FF                        ; The interface class (vendor-specific) and subclass
472     DW 0x0000                        ; The interface protocol and string descriptor address
473     DW 0x0507                        ; The IN interrupt #1 descriptor length and number
474     DW 0x0381                        ; The endpoint number/dir. and transfer type
475     DW 0x0040                        ; The maximum packet size MSB/LSB
476     DW 0x0701                        ; The polling interval and OUT interrupt #1 desc. length
477     DW 0x0105                        ; The OUT int. #1 desc. number and endpoint number/dir.
478     DW 0x4003                        ; The transfer type and maximum packet size LSB
479     DW 0x0100                        ; The maximum packet size MSB and polling interval (ms)
480     DW 0x0507                        ; The IN interrupt #2 descriptor length and number
481     DW 0x0382                        ; The endpoint number/dir. and transfer type
482     DW 0x0040                        ; The maximum packet size MSB/LSB
483     DW 0x0701                        ; The polling interval and OUT interrupt #2 desc. length
484     DW 0x0205                        ; The OUT int. #2 desc. number and endpoint number/dir.
485     DW 0x4003                        ; The transfer type and maximum packet size LSB
486     DW 0x0100                        ; The maximum packet size MSB and polling interval (ms)
487
488     ORG 0x001200
489     DW 0x0004                        ; The string descriptor zero is four bytes long
490     DW 0x0304                        ; The string descriptor length in bytes and number
491     DW 0x0409                        ; The United States English language code
492

```

```

493   ORG 0x001300
494   DW 0x0030           ; The string descriptor zero is 48 bytes long
495   DW 0x0330           ; The string descriptor number and the length in bytes
496   DW 'C','u','s','t','o','m',' ','A','u','t','o',' '
497   DW 'E','l','e','c','t','r','o','n','i','c','s'
498
499   ORG 0x001400
500   DW 0x002E           ; The string descriptor zero is 46 bytes long
501   DW 0x032E           ; The string descriptor number and the length in bytes
502   DW 'E','V',' ','M','u','l','t','i','l','e','v','e','l',' '
503   DW 'I','n','v','e','r','t','e','r'
504
505   ORG 0x300000         ; Point to the configuration word
506   DW 0x0621           ; 8MHz oscillator, use high-speed USB and 48MHz CPU
507   DW 0x003E
508   DW 0x0300
509   DW 0x0081
510
511   END

```

E.3 DSPIC30F4011 Full Assembly Code, Design Two

```
1 ;
2 ; Multilevel Converter Master Code
3 ; By Arthur Matteson, 3/23/2008
4 ;
5 ; Pin assignments:
6 ; Port B:
7 ; RB0 (2): Analog plug voltage sense input (AN0 function)
8 ; RB1 (3): Analog throttle sense input (AN1 function)
9 ; RB2 (4): Analog regen sense input (AN2 function)
10 ; RB3 (5): QEP channel index input, 5-12V logic high (INDX function)
11 ; RB4 (6): QEP channel A input, 5-12V logic high (QEA function)
12 ; RB5 (7): QEP channel B input, 5-12V logic high (QEB function)
13 ; RB6 (8): Analog phase U current input, 2.5V centered (AN6 function)
14 ; RB7 (9): Analog phase V current input, 2.5V centered (AN7 function)
15 ; RB8 (10): Analog phase W current input, 2.5V centered (AN8 function)
16 ; Port C:
17 ; RC13 (15): 1MBd module data stream output (U1ATX function)
18 ; RC14 (16): 1MBd module data stream input (U1ARX function)
19 ; RC15 (14): Module bus power PTN78000W inhibit line (inhibit low)
20 ; Port D:
21 ; RD0 (23): Non-inverted "Run" signal input, 5-12V logic high
22 ; RD1 (18): Non-inverted "Accessory" signal input, 5-12V logic high
23 ; RD2 (22): Non-inverted "Ignition" signal input, 5-12V logic high
24 ; RD3 (19): Non-inverted "Reverse" signal input, 5-12V logic high
25 ; Port E:
26 ; RE0 (38): 4000 pulses/mile vehicle speed sensor output, to transistor
27 ; RE1 (37): Non-inverted series connection relay output, to reed relay
28 ; RE2 (36): Non-inverted plug connection relay output, to reed relay
29 ; RE3 (35): Non-inverted 240VAC connection relay output, to reed relay
30 ; RE4 (34): Non-inverted 120VAC connection relay output, to reed relay
31 ; RE8 (17): Non-inverted LED output
32 ; Port F:
33 ; RF0 (30): CAN receive input
34 ; RF1 (29): CAN transmit output
35 ; RF4 (28): 2MBd intra-circuit data stream input (U2RX function)
36 ; RF5 (27): 2MBd intra-circuit data stream output (U2TX function)
37 ;
38
39
40 ; Register definitions
41 .EQU RCOUNT,0x0036
42 .EQU CORCON,0x0044
43 .EQU INTCON1,0x0080
44 .EQU INTCON2,0x0082
45 .EQU IFS1,0x0086
46 .EQU IEC1,0x008E
47 .EQU TMR1,0x0100
48 .EQU PR1,0x0102
49 .EQU T1CON,0x0104
50 .EQU TMR2,0x0106
51 .EQU TMR3HLD,0x0108
52 .EQU TMR3,0x010A
53 .EQU PR2,0x010C
54 .EQU PR3,0x010E
55 .EQU T2CON,0x0110
56 .EQU T3CON,0x0112
57 .EQU QEICON,0x0122
58 .EQU DFLTCN,0x0124
```

```

59 .EQU POSCNT,0x0126
60 .EQU MAXCNT,0x0128
61 .EQU OC1RS,0x0180
62 .EQU OC1CON,0x0184
63 .EQU OC2RS,0x0186
64 .EQU OC2CON,0x018A
65 .EQU OC3RS,0x018C
66 .EQU OC3CON,0x0190
67 .EQU OC4RS,0x0192
68 .EQU OC4CON,0x0196
69 .EQU PTCON,0x01C0
70 .EQU PTMR,0x01C2
71 .EQU PTPER,0x01C4
72 .EQU PWMCON1,0x01C8
73 .EQU PWMCON2,0x01CA
74 .EQU PDC1,0x01D6
75 .EQU PDC2,0x01D8
76 .EQU PDC3,0x01DA
77 .EQU U1MODE,0x020C
78 .EQU U1STA,0x020E
79 .EQU U1TXREG,0x0210
80 .EQU U1RXREG,0x0212
81 .EQU U1BRG,0x0214
82 .EQU U2MODE,0x0216
83 .EQU U2STA,0x0218
84 .EQU U2TXREG,0x021A
85 .EQU U2RXREG,0x021C
86 .EQU U2BRG,0x021E
87 .EQU ADCBUF0,0x0280
88 .EQU ADCBUF1,0x0282
89 .EQU ADCBUF2,0x0284
90 .EQU ADCBUF3,0x0286
91 .EQU ADCON1,0x02A0
92 .EQU ADCON2,0x02A2
93 .EQU ADCON3,0x02A4
94 .EQU ADCHS,0x02A6
95 .EQU ADPCFG,0x02A8
96 .EQU TRISB,0x02C6
97 .EQU PORTB,0x02C8
98 .EQU TRISC,0x02CC
99 .EQU PORTC,0x02CE
100 .EQU TRISD,0x02D2
101 .EQU PORTD,0x02D4
102 .EQU TRISE,0x02D8
103 .EQU PORTE,0x02DA
104 .EQU TRISF,0x02DE
105 .EQU PORTF,0x02E0
106 .EQU RCON,0x0740
107 .EQU OSCCON,0x0742
108 .EQU NVMCON,0x0760
109 .EQU NVMADR,0x0762
110 .EQU NVMADRU,0x0764
111 .EQU NVMKEY,0x0766
112
113 ; Bit definitions
114 .EQU 0A,0x000F
115 .EQU 0B,0x000E
116 .EQU 0A,0x000D
117 .EQU 0B,0x000C
118 .EQU 0AB,0x000B
119 .EQU 0AB,0x000A
120 .EQU 0A,0x0009

```

```

121 .EQU DC,0x0008
122 .EQU IPL2,0x0007
123 .EQU IPL1,0x0006
124 .EQU IPL0,0x0005
125 .EQU RA,0x0004
126 .EQU N,0x0003
127 .EQU OV,0x0002
128 .EQU Z,0x0001
129 .EQU C,0x0000
130 .EQU TON,0x000F
131 .EQU SWDTEN,0x0005
132 .EQU UARTEN,0x000F
133 .EQU UTXEN,0x000A
134 .EQU UTRMT,0x0008
135 .EQU URXDA,0x0000
136 .EQU TCKPS1,0x0005
137 .EQU TCKPS0,0x0004
138 .EQU T32,0x0003
139 .EQU U2RXIE,0x0008
140 .EQU U2RXIF,0x0008
141 .EQU SAMP,0x0001
142 .EQU DONE,0x0000
143 .EQU IF,0x0000
144 .EQU WR,0x000F
145
146 ; Variable definitions
147 .EQU proginfo,0x0820
148 .EQU proglast,0x083E
149 .EQU progdone,0x083F
150 .EQU oldsoc,0x0820 ; Each module has its own word; previous SOC (0-100%)
151 .EQU batvolts,0x0910 ; Each module has its own word; voltage (unsigned mV)
152 .EQU batamph,0x0A00 ; Each module has its own word; capacity (unsigned mAh/4)
153 .EQU batsoc,0x0AF0 ; Each module has its own long; percent SOC (0-100%)
154 .EQU charges,0x0CD0 ; Each module has its own long; charge (unsng. mAh*16384)
155 .EQU endtables,0x0EB0
156 .EQU pc_loop,0x0EB0
157 .EQU pc_idkp,0x0EB2
158 .EQU pc_idka,0x0EB4
159 .EQU timeroldl,0x0EB6
160 .EQU timeroldh,0x0EB8
161 .EQU nextadjust,0x0EBA
162 .EQU angle32l,0x0EBC
163 .EQU angle32h,0x0EBE
164 .EQU curcentu,0x0EC0
165 .EQU curcentv,0x0EC2
166 .EQU curcentw,0x0EC4
167 .EQU curalpha,0x0EC6
168 .EQU curbeta,0x0EC8
169 .EQU ampl32l,0x0ECA
170 .EQU ampl32h,0x0ECC
171 .EQU curd32l,0x0ECE
172 .EQU curd32h,0x0ED0
173 .EQU curq32l,0x0ED2
174 .EQU curq32h,0x0ED4
175 .EQU pfavg32l,0x0ED6
176 .EQU pfavg32h,0x0ED8
177 .EQU angleold,0x0EDA
178 .EQU vmin32l,0x0EDC
179 .EQU vmin32h,0x0EDE
180 .EQU vmax32l,0x0EE0
181 .EQU vmax32h,0x0EE2
182 .EQU curu,0x0EE4

```

```

183 .EQU curv,0x0EE6
184 .EQU curw,0x0EE8
185 .EQU curd,0x0EEA
186 .EQU curq,0x0EEC
187 .EQU frequency,0x0EEE
188 .EQU pwrfact,0x0EF0
189 .EQU temp,0x0EF2
190 .EQU currms,0x0EF4
191 .EQU linevolts,0x0EF6
192 .EQU throttle,0x0EF8
193 .EQU regen,0x0EFA
194 .EQU linecen,0x0EFC
195 .EQU batupdate,0x0EFE
196 .EQU atmldata,0x0840
197 .EQU atmldatalastbyte,0x0F3F
198 .EQU carstat,0x0F00
199 .EQU thrbyte,0x0F01
200 .EQU regbyte,0x0F02
201 .EQU freqbyte,0x0F03
202 .EQU amplitude,0x0F04
203 .EQU amplmin,0x0F05
204 .EQU amplmax,0x0F06
205 .EQU curubyte,0x0F07
206 .EQU curvbyte,0x0F08
207 .EQU curwbyte,0x0F09
208 .EQU curdbyte,0x0F0A
209 .EQU curqbyte,0x0F0B
210 .EQU rmsbyte,0x0F0C
211 .EQU pfbbyte,0x0F0D
212 .EQU angbyte,0x0F0E
213 .EQU linebyte,0x0F0F ; Line voltage
214 .EQU timebyte,0x0F10
215 .EQU lineneg,0x0F11 ; A temporary byte used while charging
216 .EQU onlist,0x0F22 ; Encoding: 00: Off, 01: Negative, 10: Positive, 11: PWM
217 .EQU atmelrx,0x0F40 ; This is the reply from an Atmel if 0x0F41>=0x80
218 .EQU progstat,0x0F41 ; This byte indicates whether or not programming is done
219 .EQU dataout,0x0F50 ; A returned RAM, EEPROM, or flash data set (<=48 bytes)
220 .EQU atmeltx1,0x0F80 ; When >0x8000, this is the first 9-bit word to send out
221 .EQU atmeltx2,0x0F82 ; When >0x8000, this is the second 9-bit word to send out
222 .EQU atmeltx3,0x0F84 ; When >0x8000, this is the third 9-bit word to send out
223 .EQU atmeltx4,0x0F86 ; When >0x8000, this is the fourth 9-bit word to send out
224 .EQU addressl,0x0F88 ; The DSPIC programming address (RAM, EEPROM, or flash)
225 .EQU addressh,0x0F8A
226 .EQU progdata,0x0F8C ; The DSPIC programming data word (RAM or EEPROM)
227 .EQU whosecar,0x0F8E ; >=0x80, USB control of vehicle; contactors in this byte
228 .EQU progcmd,0x0F8F ; The DSPIC memory programming command (written last)
229 .EQU flashbuf,0x0FA0 ; This 96-byte buffer is to program DSPIC flash
230
231 ; 0x0F00-0x0F3F, endpoint 0x81, is output data stream (motor values)
232 ; 0x0F40-0x0F7F, endpoint 0x82, is output control (verify DSPIC flash)
233 ; 0x0F80-0x0FBF, endpoint 0x01, is input control (program DSPIC flash)
234 ; 0x0FC0-0x0FFF, endpoint 0x02, is input control (program DSPIC flash)
235
236 ; Constant definitions
237 .EQU MINRESET,4215
238 .EQU MINTHR,0x0080 ; Throttle values <= this are considered zero
239 .EQU MAXTHR,0x0340 ; Throttle values > this are considered W.O.T.
240 .EQU MINFREQ,24 ; The minimum value of 'frequency' (Hz*2.097152)
241 .EQU MAXFREQ,200 ; The maximum value of 'frequency' (Hz*2.097152)
242 .EQU FREQSCALE,4000 ; MAXFREQ/(MAXTHR-MINTHR)*65536
243 .EQU MINMINV,6 ; The lowest lower limit of line-neutral amplitude
244 .EQU MINMAXV,24 ; The lowest upper limit of line-neutral amplitude

```



```

245 .EQU MAXMINV,36      ; The highest lower limit of line-neutral amplitude
246 .EQU MAXMAXV,62      ; The highest upper limit of line-neutral amplitude
247 .EQU MINVPERHZ,240    ; The line-neutral peak batteries per 'frequency' (12.3V)
248 .EQU MAXVPERHZ,2000   ; The line-neutral peak batteries per 'frequency' (12.3V)
249 .EQU IDOPTIMAL,120    ; The optimal D current in amps*4 (30A)
250 .EQU LOOP,50         ; How many 2.048ms increments to close the loop after
251 .EQU IDKP,100        ; The proportional constant for ID regulation
252 .EQU IDKA,4          ; The integral constant for ID regulation
253 .EQU IQKP,21846       ; The proportional constant for IQ regulation
254
255 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
256 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
257
258 .SECTION .text
259 .WORD 0x0100          ; Master firmware version 1.00
260 .GLOBAL __reset
261 __reset:
262     BCLR.W RCON,#SWDTEN ; Disable the watchdog timer
263
264 ; Initialize the transmit pin early
265     MOV.W #0x0028,W0
266     MOV.W W0,PORTF
267
268 ; Set up the data direction registers
269     MOV.W #0x01FF,W0
270     MOV.W W0,TRISB
271     MOV.W #0x4000,W0
272     MOV.W W0,TRISC
273     MOV.W #0x000F,W0
274     MOV.W W0,TRISD
275     MOV.W #0x0000,W0
276     MOV.W W0,TRISE
277     MOV.W #0x0011,W0
278     MOV.W W0,TRISF
279
280 ; Initialize the port values
281     MOV.W #0x0000,W0
282     MOV.W W0,PORTB
283     MOV.W #0x2000,W0
284     MOV.W W0,PORTC
285     MOV.W #0x0000,W0
286     MOV.W W0,PORTD
287     MOV.W #0x0000,W0
288     MOV.W W0,PORTE
289     MOV.W #0x0020,W0
290     MOV.W W0,PORTF
291
292 ; Set up serial port number two at 2MBd
293     MOV.W #0x0007,W0
294     MOV.W W0,U2MODE
295     MOV.W #0x0000,W0
296     MOV.W W0,U2BRG
297     BSET.W U2MODE,#UARTEN
298     BSET.W U2STA,#UTXEN
299
300 ; Disable the two QEP-shared A/D inputs on PORTB, so they can be read digital
301     MOV.W #0x0038,W0
302     MOV.W W0,ADPCFG
303
304 ; Set up the quadrature encoder interface
305     MOV.W #0x0600,W0
306     MOV.W W0,QEICON

```

```

307     CLR.W POSCNT
308
309 ; Set up the rest of the A/D converter for AN0-AN2/AN6-AN8 (TAD=183.3ns)
310     MOV.W #0x1FOA,W0
311     MOV.W W0,ADCON3
312     MOV.W #0x80E0,W0
313     MOV.W W0,ADCON1
314
315 ; Set up Timer2 and Timer3 as a 32-bit timer
316     SETM.W PR2
317     SETM.W PR3
318     BSET.W T2CON,#T32      ; Set up the timers together as a 32-bit pair
319     BSET.W T2CON,#TON      ; Start the timers counting with no period limitation
320
321 ; Initialize the A/D center values (three phase currents and a line voltage)
322     RCALL init_centers
323
324     MOV.W #batsoc,W0      ; Clear out the percent SOC array
325     REPEAT #239
326     CLR.W [W0++]          ; Each module has its own long; percent SOC (0-100%)
327
328 ; Delay and turn on the battery modules' power
329     BSET.W PORTD,#2
330     DO #999,endstart1
331     REPEAT #16383          ; Delay about a half second
332     NOP
333 endstart1:
334     NOP
335     BCLR.W PORTD,#2
336     BSET.W TRISC,#15      ; Turn on the module power
337     BSET.W PORTD,#2
338     DO #1999,endstart2
339     REPEAT #16383          ; Delay about a second
340     NOP
341 endstart2:
342     NOP
343     BCLR.W PORTD,#2
344
345 ; Set up serial port number one at 1MBd
346     MOV.W #0x0407,W0
347     MOV.W W0,U1MODE
348     MOV.W #0x0001,W0
349     MOV.W W0,U1BRG
350     BSET.W U1MODE,#UARTEN
351     BSET.W U1STA,#UTXEN
352
353 ; Clear out the hardware UART input receive buffers
354     MOV.W U1RXREG,W0
355     MOV.W U1RXREG,W0
356     MOV.W U1RXREG,W0
357     MOV.W U1RXREG,W0
358     MOV.W U2RXREG,W0
359     MOV.W U2RXREG,W0
360     MOV.W U2RXREG,W0
361     MOV.W U2RXREG,W0
362
363 ; Load the battery capacities from EEPROM into RAM
364     MOV.W #0x007F,W4      ; Load TBLPAG with 0x007F, the EEPROM location
365     MOV.W W4,TBLPAG
366     MOV.W #0xFC30,W4      ; Load the beginning EEPROM address into W1
367     MOV.W #batvolts,W5    ; Load the voltage table address into W5
368     MOV.W #batamph,W6     ; Load the capacity table address into W6

```

```

369 CLR.W W1 ; Start verifying at address one (see INC.W below)
370 DO #119,endloadcap ; Copy 120 words of capacity data from EEPROM to RAM
371 INC.W W1,W1 ; Increment the module address
372 CLR.W [W5++] ; Clear the voltage table simultaneously
373 TBLRDL.W [W4++],W0
374 CPO.W W0 ; Is the capacity zero? If so, don't verify comm.
375 BRA Z,endloadcap
376 RCALL check_comm ; Perform a check communications test on this module
377 CP.W W2,#1 ; Was it successful?
378 BRA Z,endloadcap
379 CLR.W W0 ; If the module wasn't found, don't use it
380 endloadcap:
381 MOV.W W0,[W6++] ; Store the capacity in RAM (batamph, mAh/4)
382
383 MOV.W #0x017B,W1 ; Send a wake (and turn off) command to all modules
384 MOV.W W1,U1TXREG
385
386 ; Clear variables (the rest are done below under active:)
387 CLR.W throttle
388 CLR.W regen
389 MOV.W #LOOP,W0
390 MOV.W W0,pc_loop
391 MOV.W #IDKP,W0
392 MOV.W W0,pc_idkp
393 MOV.W #IDKA,W0
394 MOV.W W0,pc_idka
395 CLR.B carstat
396 CLR.B whosecar
397 CLR.W TMR2 ; Clear the timer initially
398 CLR.W TMR3
399 CLR.W timeroldl
400 CLR.W timeroldh
401 MOV.W pc_loop,W0
402 MOV.W W0,nextadjust ; Also calculate the next time to close the loop
403 CLR.W temp
404 CLR.W batupdate
405 CLR.W angle32l
406 CLR.W angle32h
407 CLR.B lineneg
408
409 ; Clear the entire 128-byte output buffer and output this to the PIC18F2553
410
411 MOV.W #carstat,W2 ; Send the entire section data to the PIC18F2553
412 AND.W #0x00FF,W2
413 MOV.W W2,U2TXREG
414 REPEAT #190
415 NOP
416 MOV.W #0x0100,W1
417 MOV.W #carstat,W2
418 DO #127,.,+14 ; Send 128 data byte commands to the PIC18F2553
419 CLR.B [W2++]
420 MOV.W W1,U2TXREG
421 REPEAT #190
422 NOP
423 NOP
424
425 ; Perform other initialization and enable the UART2 receive interrupt
426 MOV.W #carstat,W14 ; This is the current writing address for U2RXREG bytes
427 CLR.B progcmd ; Clear the command so nothing happens accidentally
428 CLR.B progstat ; Clear the programming read status byte
429 BCLR.W IFS1,#U2RXIF ; Clear the U2RX interrupt flag in IFS1
430 BSET.W IEC1,#U2RXIF ; Enable the U2RX interrupt in IEC1

```

```

431
432 ; Read the battery voltages of those which are present, and update the SOC %
433 CLR.W W12
434 readvoltsloop:
435     RCALL voltcheck
436     INC.W W12,W12
437     BRA NZ,readvoltsloop
438
439     BSET.W PORTE,#8      ; Turn the LED on
440
441 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
442
443 mainloop:
444     MOV.W POSCNT,W0      ; Recall the QEI counter value
445     MOV.W #54613,W1      ; This is a scale factor for the vehicle speed sensor
446     MUL.UU W0,W1,W2      ; W3:W2=W0*W1
447     BTSC.W W3,#9         ; Copy the scaled output bit to the vehicle speed sensor
448     BRA .+6
449     BCLR.W PORTE,#0
450     BRA .+4
451     BSET.W PORTE,#0
452
453     RCALL handle_USB     ; Handle USB programming commands
454
455     MOV.W #0x0000,W1      ; Select channel AN0 for A/D conversion (line voltage)
456     MOV.W W1,ADCHS
457     BSET.W ADCON1,#SAMP   ; Start an A/D conversion
458     BTSS.W ADCON1,#DONE
459     BRA .-2              ; Keep looping until it completes
460     BCLR.W ADCON1,#DONE
461     CLR.W W0
462     MOV.W linevolts,W0    ; Recall the running average
463     MUL.SU W0,#3,W2       ; Weight the previous value by three
464     MOV.W ADCBUFO,W0      ; Weight the current value by one
465     SUBR.W linecen,WREG   ; Center the result with the zero volts result
466     ADD.W W0,W2,W0
467     ADD.W #2,W0          ; Add one-half before dividing by four
468     ASR.W W0,#2,W0
469     MOV.W W0,linevolts    ; Store the result (integer format)
470     ASR.W W0,#2,W0        ; Get rid of the lower two bits of the new result
471     MOV.B WREG,linebyte   ; Store the result (integer format)
472     MOV.W linevolts,W0
473     ADD.W #40,W0          ; Is the line voltage reasonably negative?
474     BRA NN,+.4
475     BSET.B lineneg,#0     ; If so, set the line negative flag temporarily
476     MOV.W linevolts,W0
477     SUB.W #40,W0         ; Is the line voltage reasonably positive?
478     BRA N,+.4
479     BCLR.B lineneg,#0    ; If so, clear the line negative flag temporarily
480
481     MOV.B carstat,WREG    ; Recall the status byte low nibble
482     AND.B #0x0F,W0
483     BTSC.W PORTD,#0       ; Is the "Run" input high? If so, set W0<4>
484     BSET.W W0,#4
485     BTSC.W PORTD,#1       ; Is the "Accessory" input high? If so, set W0<5>
486     BSET.W W0,#5
487     BTSC.W PORTD,#2       ; Is the "Ignition" input high? If so, set W0<6>
488     BSET.W W0,#6
489     BTSC.W PORTD,#3       ; Is the "Reverse" input high? If so, set W0<7>
490     BSET.W W0,#7
491     MOV.B WREG,carstat
492

```

```

493     MOV.W #0x0001,W1      ; Select channel AN1 for A/D conversion (throttle)
494     MOV.W W1,ADCHS
495     BSET.W ADCON1,#SAMP    ; Start an A/D conversion
496     BTSS.W ADCON1,#DONE
497     BRA .-2                ; Keep looping until it completes
498     BCLR.W ADCON1,#DONE
499     CLR.W W0
500     MOV.W throttle,W0      ; Recall the running average
501     MUL.UU W0,#7,W2        ; Weight the previous value by seven
502     MOV.W ADCBUF0,W0       ; Weight the current value by one
503     ADD.W W0,W2,W0
504     ADD.W #4,W0            ; Add one-half before dividing by eight
505     LSR.W W0,#3,W0
506     MOV.W W0,throttle      ; Store the result (integer format)
507     LSR.W W0,#2,W0         ; Get rid of the lower two bits of the new result
508     MOV.B WREG,thrbyte     ; Store the result (integer format)
509
510     MOV.W #carstat,W2      ; Send the carstat address command to the PIC18F2553
511     AND.W #0x00FF,W2
512     MOV.W W2,U2TXREG
513     REPEAT #190
514     NOP
515     MOV.B carstat,WREG
516     BSET.W W0,#8           ; Send the carstat data byte command to the PIC18F2553
517     MOV.W W0,U2TXREG
518     REPEAT #190
519     NOP
520     MOV.B thrbyte,WREG
521     BSET.W W0,#8           ; Send the throttle data byte command to the PIC18F2553
522     MOV.W W0,U2TXREG
523     REPEAT #190
524     NOP
525
526 ; Send off the line voltage and current time to the PIC18F2553, even for car off
527
528     MOV.W #linebyte,W2     ; Send the line voltage address command to the PIC18F2553
529     AND.W #0x00FF,W2
530     MOV.W W2,U2TXREG
531     REPEAT #190
532     NOP
533     MOV.B linebyte,WREG
534     BSET.W W0,#8           ; Send the line voltage data byte to the PIC18F2553
535     MOV.W W0,U2TXREG
536     REPEAT #190
537     NOP
538     MOV.W TMR3,W0          ; Recall the current timer high word
539     MOV.B WREG,timebyte
540     BSET.W W0,#8           ; Send the time data byte to the PIC18F2553
541     MOV.W W0,U2TXREG
542     REPEAT #190
543     NOP
544
545 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
546
547 ; BRA chargemain
548
549 ; Test the ignition for a "Run" signal
550
551 ; CPO.B disable            ; Is USB commanding the system be in diagnostic mode?
552 ; BRA NZ,+.6              ; If so, don't active the inverter
553 BTST.W PORTD,#0           ; Is the "Run" signal asserted?
554 BRA NZ,active

```

```

555
556 ; Periodically query the battery voltages in sequence when the car is off
557
558 ; RCALL voltcheck
559
560 ; Since the "Run" signal was off, execute an off sequence
561
562 BTST.B carstat,#0 ; Was the inverter recently on?
563 BRA Z,mainloop ; If it's been off, don't bother to sleep the modules
564
565 MOV.W #0x0001,W0 ; Set the baud rate to 1MBd
566 MOV.W W0,U1BRG
567 MOV.W #0x017B,W1 ; Send a wake (and turn off) command to all modules
568 MOV.W W1,U1TXREG
569 BCLR.B carstat,#0 ; Set carstat<0> to zero
570 BCLR.W PORTE,#8 ; Turn the LED off
571
572 ; Clear almost the entire output buffer and output this to the PIC18F2553
573
574 MOV.W #freqbyte,W1
575 REPEAT #12
576 CLR.B [W1++]
577 MOV.W #onlist,W1
578 REPEAT #14
579 CLR.W [W1++]
580
581 MOV.W #carstat,W2 ; Send the entire section data to the PIC18F2553
582 AND.W #0x00FF,W2
583 MOV.W W2,U2TXREG
584 REPEAT #190
585 NOP
586 MOV.W #carstat,W2
587 DO #63,+.14 ; Send 64 data byte commands to the PIC18F2553
588 MOV.B [W2++],W1
589 BSET.W W1,#8
590 MOV.W W1,U2TXREG
591 REPEAT #190
592 NOP
593 NOP
594
595 BRA mainloop ; Keep sampling the throttle to see if it comes back on
596
597 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
598
599 active:
600 ; Since the "Run" signal was on, execute an on sequence
601
602 BTST.B carstat,#0 ; Was the inverter recently on?
603 BRA NZ,act_wason ; If it's been on, don't bother to wake the modules
604 BTST.W PORTD,#2 ; Is the "Ignition" signal asserted?
605 BRA Z,mainloop ; If not, keep waiting
606
607 MOV.W #0x0001,W0 ; Set the baud rate to 1MBd
608 MOV.W W0,U1BRG
609 MOV.W #0x017C,W1 ; Send a wake (and turn on) command to all modules
610 MOV.W W1,U1TXREG
611 REPEAT #383 ; Wait for this command to execute
612 NOP
613 MOV.W #MINFREQ,W0 ; Initialize many variables
614 MOV.W W0,frequency
615 CLR.W angle32l
616 CLR.W angle32h

```

```

617 CLR.W curu
618 CLR.W curv
619 CLR.W curw
620 CLR.W curalpha
621 CLR.W curbeta
622 CLR.W curd
623 CLR.W curq
624 MOV.W #MINMINV,W0
625 SL.W W0,#11,W0
626 MOV.W W0,ampl32l
627 CLR.W ampl32h
628 CLR.W curd32l
629 CLR.W curd32h
630 CLR.W curq32l
631 CLR.W curq32h
632 CLR.W pwrfact
633 CLR.W pfavg32l
634 MOV.W #700,W0
635 MOV.W W0,pfavg32h
636 CLR.W currms
637 MOV.W TMR2,W2
638 MOV.W TMR3HLD,W3
639 MOV.W W2,timeroldl
640 MOV.W W3,timeroldh
641 MOV.W pc_loop,W2
642 ADD.W W1,W3,W1
643 MOV.W W1,nextadjust
644 BSET.B carstat,#0 ; Set carstat<0> to one
645 BSET.W PORTE,#8 ; Turn the LED on
646
647 MOV.W #carstat,W2 ; Send the entire section data to the PIC18F2553
648 AND.W #0x00FF,W2
649 MOV.W W2,U2TXREG
650 REPEAT #190
651 NOP
652 MOV.W #carstat,W2
653 DO #63,+.14 ; Send 64 data byte commands to the PIC18F2553
654 MOV.B [W2++],W1
655 BSET.W W1,#8
656 MOV.W W1,U2TXREG
657 REPEAT #190
658 NOP
659 NOP
660 act_wason:
661
662 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
663
664 ; Interpret the sampled throttle and generate a frequency command
665 CLR.W W0
666 MOV.W throttle,W0 ; Recall the throttle command (0-1023)
667 SUB.W #MINTHR,W0 ; Subtract off the minimum throttle value
668 BRA NN,+.4 ; Skip the next instruction on a non-zero throttle
669 CLR.W W0 ; Saturate to zero if less than zero
670 MOV.W #FREQSCALE,W2 ; Load in the constant to multiply by, to find frequency
671 MUL.UU W0,W2,W2 ; W3:W2=W0*W2 (W3 is the frequency)
672 MOV.W W3,W1 ; Move this result to W1 to conserve register space
673 MOV.W W1,frequency ; Save the frequency (Hz*2.097152)
674
675 MOV.W TMR2,W13 ; Remember the beginning timer low word
676 ; MOV.W #10923,W1
677 MOV.W #18500,W1
678 ADD.W W1,W13,W13

```

```

679
680     RCALL find_minmax      ; Find the minimum and maximum voltages for motoring
681     RCALL do_timer        ; Find the accumulated time
682     RCALL output_PWM      ; Generate module commands with PWM for motoring
683     RCALL convert_curs    ; Convert the current A/D channels
684     RCALL clarke          ; Perform a Clarke transform of the currents
685     RCALL park            ; Perform a Park transform of the currents
686 ;   RCALL find_rms        ; Find the RMS current
687     RCALL find_pf         ; Find the instantaneous power factor and average it
688     RCALL motor_loop      ; Close the loop for motoring
689     RCALL send_vars       ; Transfer data to the PIC18F2553
690
691     MOV.W W13,W0
692     CP.W TMR2             ; Compare the low word of the 32-bit timer with W0
693     BRA N,-2              ; If TMR2 hasn't passed W0 yet, keep waiting
694     BRA mainloop
695
696 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
697
698 chargemain:
699     MOV.W #126,W0         ; Set the frequency to as close to 60Hz as possible
700     MOV.W W0,frequency
701     MOV.W #40960,W0       ; Set the fractional battery amplitude to five-eighths
702     MOV.W W0,ampl32l
703     MOV.W #6,W0           ; Set the integral battery amplitude to six
704     MOV.W W0,ampl32h
705
706     CP0.B lineneg        ; Is the negative line input voltage flag set?
707     BRA Z,charge_notneg
708     CP0.W linevolts       ; Is the line voltage crossing zero just now?
709     BRA NZ,charge_notneg
710     CLR.B lineneg        ; If so, clear the flag and reset the angles
711     CLR.W angle32l
712     MOV.W #145,W0         ; 145 is good (160 gives no current flow)
713     MOV.W W0,angle32h
714 charge_notneg:
715
716     RCALL do_timer        ; Find the accumulated time
717     RCALL output_charge   ; Generate module commands with no PWM for charging
718     RCALL send_vars       ; Transfer data to the PIC18F2553
719     BRA mainloop
720
721 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
722 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
723
724 init_centers:             ; Initialize A/D center values (currents and voltage)
725     MOV.W #0x0006,W1      ; Select channel AN6 for A/D conversion
726     MOV.W W1,ADCHS
727     CLR.W W2              ; Clear the total at the beginning
728     DO #15,avg_curu       ; Sample the U current sensor 16 times and average
729     BSET.W ADCON1,#SAMP   ; Start an A/D conversion
730     NOP
731     NOP
732     NOP
733     NOP
734     BTSS.W ADCON1,#DONE
735     BRA .-2              ; Keep looping until it completes
736     BCLR.W ADCON1,#DONE
737     MOV.W ADCBUFO,W1
738     SL.W W1,#2,W1         ; Multiply the ten-bit integer result by four
739     INC2.W W1,W1          ; Add one-half of four because of the shift
740 avg_curu:

```



```

741     ADD.W W1,W2,W2      ; Add the result to the total and maybe sample again
742     LSR.W W2,#4,W2      ; Divide the total by 16
743     MOV.W W2,curcentu    ; Store the result in counts*four (integer format)
744
745     MOV.W #0x0007,W1     ; Select channel AN7 for A/D conversion
746     MOV.W W1,ADCHS
747     CLR.W W2             ; Clear the total at the beginning
748     DO #15,avg_curv      ; Sample the V current sensor 16 times and average
749     BSET.W ADCON1,#SAMP  ; Start an A/D conversion
750     NOP
751     NOP
752     NOP
753     NOP
754     BTSS.W ADCON1,#DONE
755     BRA .-2              ; Keep looping until it completes
756     BCLR.W ADCON1,#DONE
757     MOV.W ADCBUFO,W1
758     SL.W W1,#2,W1        ; Multiply the ten-bit integer result by four
759     INC2.W W1,W1         ; Add one-half of four because of the shift
760     avg_curv:
761     ADD.W W1,W2,W2      ; Add the result to the total and maybe sample again
762     LSR.W W2,#4,W2      ; Divide the total by 16
763     MOV.W W2,curcentv    ; Store the result in counts*four (integer format)
764
765     MOV.W #0x0008,W1     ; Select channel AN8 for A/D conversion
766     MOV.W W1,ADCHS
767     CLR.W W2             ; Clear the total at the beginning
768     DO #15,avg_curv      ; Sample the W current sensor 16 times and average
769     BSET.W ADCON1,#SAMP  ; Start an A/D conversion
770     NOP
771     NOP
772     NOP
773     NOP
774     BTSS.W ADCON1,#DONE
775     BRA .-2              ; Keep looping until it completes
776     BCLR.W ADCON1,#DONE
777     MOV.W ADCBUFO,W1
778     SL.W W1,#2,W1        ; Multiply the ten-bit integer result by four
779     INC2.W W1,W1         ; Add one-half of four because of the shift
780     avg_curw:
781     ADD.W W1,W2,W2      ; Add the result to the total and maybe sample again
782     LSR.W W2,#4,W2      ; Divide the total by 16
783     MOV.W W2,curcentw    ; Store the result in counts*four (integer format)
784
785     MOV.W #0x0000,W1     ; Select channel AN0 for A/D conversion (line voltage)
786     MOV.W W1,ADCHS
787     CLR.W W2             ; Clear the total at the beginning
788     DO #15,avg_line      ; Sample the line voltage sensor 16 times and average
789     BSET.W ADCON1,#SAMP  ; Start an A/D conversion
790     NOP
791     NOP
792     NOP
793     NOP
794     BTSS.W ADCON1,#DONE
795     BRA .-2              ; Keep looping until it completes
796     BCLR.W ADCON1,#DONE
797     MOV.W ADCBUFO,W1
798     avg_line:
799     ADD.W W1,W2,W2      ; Add the result to the total and maybe sample again
800     LSR.W W2,#4,W2      ; Divide the total by 16
801     MOV.W W2,linecen     ; Store the result in counts (integer format)
802

```

```

803     RETURN
804
805     ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
806
807     check_comm:                ; Module address in W1 (uncorrupted), return value in W2
808         MOV.W U1RXREG,W2      ; Clear out the receive buffer first, to prevent errors
809         MOV.W U1RXREG,W2
810         MOV.W U1RXREG,W2
811         MOV.W U1RXREG,W2
812         CLR.W W3              ; Reset the loop counter (value to be echoed)
813     cc_loop:
814         MOV.W W1,U1TXREG      ; Address the module in W1
815         REPEAT #400
816         NOP
817         MOV.W #0x0000,W2      ; Next, send out a zero (echo next byte command)
818         MOV.W W2,U1TXREG      ; (The timeout counter is simultaneously cleared above)
819         REPEAT #400
820         NOP
821         MOV.W W3,U1TXREG      ; Finally, send out the loop counter value
822         BTSC.W U1STA,#URXDA   ; Is there at least one available word?
823         BRA .+10
824         INC.W W2,W2           ; If not, keep waiting; increment the timeout counter
825         BTSS.W W2,#12        ; If the timer overflowed to 4096, exit with W2 as one
826         BRA .-8              ; If the timer has not overflowed, keep waiting
827         RETLW.W #2,W2        ; Return with an unsuccessful (two) value in W2
828         MOV.W U1RXREG,W2      ; Grab the received byte
829         BCLR.W W2,#8         ; Ignore the ninth bit
830         XOR.W W1,W2,W2        ; Undo the applied XOR
831         CP.B W2,W3           ; Is the value the same as the loop counter?
832         BRA NZ, .+10
833         INC.W W3,W3          ; Increment the loop counter
834         BTSS.W W3,#8         ; Has the counter hit 0x0100?
835         BRA cc_loop          ; If not, keep looping with the new counter value
836         RETLW.W #1,W2        ; Return with a successful (one) value in W2
837         RETLW.W #3,W2        ; Return with an unsuccessful (three) value in W2
838
839     ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
840
841     find_minmax:              ; Find the minimum and maximum voltages for motoring
842         MOV.W frequency,W1    ; Load in the frequency calculated by the PID controller
843         MOV.W #MINVPERHZ,W2   ; Load in the constant to multiply by, to find amplitude
844         MUL.UU W1,W2,W2       ; W3:W2=W1*W2 (W3:W2 is the minimum amplitude by slope)
845         MOV.W #MINMINV,W6     ; Compare the lower limit to the lowest legal value
846         ASR.W W6,#5,W7        ; Divide by 32 (create a high word limit)
847         SL.W W6,#11,W6        ; Multiply by 2048 (create a low word limit)
848         SUB.W W2,W6,W8        ; Subtract the minimum low word from the new low word
849         SUBB.W W3,W7,W9       ; Subtract the minimum high word from the new high word
850         BRA NN, .+4           ; If the calculated minimum is legal, don't adjust it
851         MOV.D W6,W2           ; Replace the calculated minimum with the lowest minimum
852         MOV.W #MAXMINV,W6     ; Compare the lower limit to the highest legal value
853         ASR.W W6,#5,W7        ; Divide by 32 (create a high word limit)
854         SL.W W6,#11,W6        ; Multiply by 2048 (create a low word limit)
855         SUB.W W6,W2,W8        ; Subtract the new low word from the maximum low word
856         SUBB.W W7,W3,W9       ; Subtract the new high word from the maximum high word
857         BRA NN, .+4           ; If the calculated minimum is legal, don't adjust it
858         MOV.D W6,W2           ; Replace the calculated minimum with the highest minimum
859         MOV.W W2,vmin32l      ; Save the minimum ampl. low word (fractional batteries)
860         MOV.W W3,vmin32h      ; Save the minimum ampl. high word (integral batteries)
861         MOV.W #MAXVPERHZ,W2   ; Load in the constant to multiply by, to find amplitude
862         MUL.UU W1,W2,W2       ; W3:W2=W1*W2 (W3:W2 is the maximum amplitude by slope)
863         MOV.W #MINMAXV,W6     ; Compare the upper limit to the lowest legal value
864         ASR.W W6,#5,W7        ; Divide by 32 (create a high word limit)

```

```

865     SL.W W6,#11,W6      ; Multiply by 2048 (create a low word limit)
866     SUB.W W2,W6,W8      ; Subtract the minimum low word from the new low word
867     SUBB.W W3,W7,W9     ; Subtract the minimum high word from the new high word
868     BRA NN, .+4         ; If the calculated maximum is legal, don't adjust it
869     MOV.D W6,W2         ; Replace the calculated maximum with the lowest maximum
870     MOV.W #MAXMAXV,W6   ; Compare the upper limit to the highest legal value
871     ASR.W W6,#5,W7      ; Divide by 32 (create a high word limit)
872     SL.W W6,#11,W6      ; Multiply by 2048 (create a low word limit)
873     SUB.W W6,W2,W8      ; Subtract the new low word from the maximum low word
874     SUBB.W W7,W3,W9     ; Subtract the new high word from the maximum high word
875     BRA NN, .+4         ; If the calculated maximum is legal, don't adjust it
876     MOV.D W6,W2         ; Replace the calculated maximum with the highest maximum
877     MOV.W W2,vmax32l    ; Save the maximum ampl. low word (fractional batteries)
878     MOV.W W3,vmax32h    ; Save the maximum ampl. high word (integral batteries)
879     RETURN
880
881     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
882
883 charge_on:                ; Find the least-charged battery of the whole group
884     MOV.W #batsoc,W0     ; Load the location of the battery module SOC percents
885     MOV.W #onlist,W1     ; Load the location of the battery module "on-list"
886     CLR.W W2             ; W2 is the current module (clear at the beginning)
887     CLR.W W3             ; W3 is the minimum module (clear at the beginning)
888     SETM.W W4            ; W5:W4 is the highest possible state of charge (100%)
889     SETM.W W5
890     MOV.W #0x0003,W8     ; Initialize the battery module status mask
891     DO #119,c_on_loop    ; Search through the entire battery module list
892     MOV.D [W0++],W6      ; Recall the SOC percent of the tested module
893     ADD.W W6,W7,W9       ; Is the tested long zero? If so, exit now
894     BRA C, .+4           ; If W6+W7 is coincidentally zero, don't exit
895     BRA Z,c_on_inc       ; Only exit if both W6 and W7 are zero
896     AND.W W8,[W1],W9     ; Mask off the status of the current battery module
897     BRA NZ,c_on_inc      ; If it isn't currently off, exit now
898     SUB.W W4,W6,W10      ; Subtract the tested low word from the minimum SOC
899     SUBB.W W5,W7,W11     ; Subtract the tested high word from the minimum SOC
900     BRA LTU,c_on_inc     ; An equal or lesser tested module will replace the min.
901     MOV.D W6,W4          ; Since this module had less charge, save its parameters
902     INC.W W2,W3
903 c_on_inc:
904     INC.W W2,W2          ; Increment the current module address
905     RLNC.W W8,W8         ; Rotate the battery module status mask left w/o carry
906     BRA NN, .+4         ; If the '11' was at the top, increment the list pointer
907     INC2.W W1,W1
908 c_on_loop:
909     RLNC.W W8,W8
910     RETURN              ; W3 is the least-charged battery (zero if none found)
911
912     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
913
914 charge_off:               ; Find the most-charged battery of the whole group
915     MOV.W #batsoc,W0     ; Load the location of the battery module SOC percents
916     MOV.W #onlist,W1     ; Load the location of the battery module "on-list"
917     CLR.W W2             ; W2 is the current module (clear at the beginning)
918     CLR.W W3             ; W3 is the maximum module (clear at the beginning)
919     CLR.W W4             ; W5:W4 is the lowest possible state of charge (0%)
920     CLR.W W5
921     MOV.W #0x0003,W8     ; Initialize the battery module status mask
922     DO #119,c_off_loop   ; Search through the entire battery module list
923     MOV.D [W0++],W6      ; Recall the SOC percent of the tested module
924     ADD.W W6,W7,W9       ; Is the tested long zero? If so, exit now
925     BRA C, .+4           ; If W6+W7 is coincidentally zero, don't exit
926     BRA Z,c_off_inc      ; Only exit if both W6 and W7 are zero

```

```

927     AND.W W8,[W1],W9      ; Mask off the status of the current battery module
928     BRA Z,c_off_inc       ; If it isn't currently on, exit now
929     SUBR.W W4,W6,W10      ; Subtract the maximum low word from the tested SOC
930     SUBBR.W W5,W7,W11     ; Subtract the maximum high word from the tested SOC
931     BRA LTU,c_off_inc     ; An equal or greater tested module will replace the max.
932     MOV.D W6,W4           ; Since this module had less charge, save its parameters
933     INC.W W2,W3
934 c_off_inc:
935     INC.W W2,W2           ; Increment the current module address
936     RLNC.W W8,W8         ; Rotate the battery module status mask left w/o carry
937     BRA NN,+.4           ; If the '11' was at the top, increment the list pointer
938     INC2.W W1,W1
939 c_off_loop:
940     RLNC.W W8,W8
941     RETURN                ; W3 is the most-charged battery (zero if none found)
942
943 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
944
945 do_timer:                ; Find the accumulated time
946     MOV.W frequency,W1    ; Load in the frequency calculated by the PID controller
947     MOV.W timeroldl,W3    ; Load the old timer value low word
948     MOV.W timeroldh,W4    ; Load the old timer value high word
949     MOV.W TMR2,W5         ; Load the new timer value low word
950     MOV.W TMR3HLD,W6      ; Load the new timer value high word
951     SUB.W W5,W3,W3        ; Subtract the old low word from the new low word
952     SUBB.W W6,W4,W4       ; Subtract the old high word from the new high word
953     BRA NN,+.10          ; If the result is negative, take the absolute value
954     NEG.W W3,W3          ; Complement and increment the low word
955     BRA NZ,+.4           ; If there was an overflow, decrement the high word
956     DEC.W W4,W4          ; This decrement will change into an increment next
957     COM.W W4,W4          ; Negate the high word by complementing
958     MOV.W W5,timeroldl    ; Save the timer value low word for next time
959     MOV.W W6,timeroldh    ; Save the timer value high word for next time
960     RETURN
961
962 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
963
964 output_PWM:              ; Generate module commands with PWM for motoring
965     MUL.UU W1,W3,W6       ; W7:W6=W1*W3 (W7 is part of the low product, an angle)
966     MUL.UU W1,W4,W4       ; W5:W4=W1*W4 (W4 is part of the high product, an angle)
967     ADD.W W4,W7,W3        ; W3:W6 is the accumulated angle from frequency
968     BTST.W PORTD,#3      ; Is the "Reverse" signal active?
969     BRA NZ,goreverse
970     MOV.W angle32l,W1     ; Recall the old accumulated angle low word
971     ADD.W W1,W6,W6        ; Add the old angle low word to the new angle low word
972     MOV.W angle32h,W1     ; Recall the old accumulated angle high word
973     MOV.W W1,angleold     ; Save this angle for calculations below
974     ADDC.W W1,W3,W3       ; Add the old angle high word to the new angle high word
975     BRA goforward
976 goreverse:
977     MOV.W angle32l,W1     ; Recall the old accumulated angle low word
978     SUB.W W1,W6,W6        ; Subtract new angle low from old angle low
979     MOV.W angle32h,W1     ; Recall the old accumulated angle high word
980     MOV.W W1,angleold     ; Save this angle for calculations below
981     SUBB.W W1,W3,W3       ; Subtract new angle high from old angle high
982 goforward:
983     MOV.W W6,angle32l     ; Save the accumulated angle low word for next time
984     MOV.W W3,angle32h     ; Save the accumulated angle high word for next time
985     RCALL sine_table      ; Return the sin(W3&0x03FF) in W6, 0x0001 to 0x03FF
986     MOV.W W6,W4          ; Save U's amplitude in W4
987     ADD.W #341,W3        ; Add about 120 degrees
988     RCALL sine_table      ; Return the sin(W3&0x03FF) in W6, 0x0001 to 0x03FF

```

```

989     MOV.W W6,W5           ; Save V's amplitude in W5
990     ADD.W #342,W3         ; Add about 120 degrees
991     RCALL sine_table      ; Return the sin(W3&0x03FF) in W6, 0x0001 to 0x03FF
992
993     CP0.W frequency       ; Is the frequency zero?
994     BRA NZ,notzerof
995     MOV.W #0x0200,W4      ; If so, effectively short out the motor
996     MOV.W #0x0200,W5
997     MOV.W #0x0200,W6
998     MOV.W #MINMINV,W0     ; Also reduce the initial applied voltage
999     SL.W W0,#11,W0
1000    MOV.W W0,amp132l
1001    CLR.W amp132h
1002    notzerof:
1003
1004    ; Generate module commands for all three phases
1005
1006    MOV.W #0x0060,W1       ; Attempt to turn module #1/#4 on positive
1007    SUB.W #0x0200,W4       ; If U's amplitude is below zero, invert it
1008    BRA NN,.,+6
1009    MOV.W #0x0040,W1       ; Turn module #1/#4 on negative instead
1010    NEG.W W4,W4           ; Negate the duty cycle
1011    MOV.W amp132l,W2       ; Recall the fractional battery amplitude
1012    MOV.W amp132h,W3       ; Recall the integral battery amplitude
1013    MUL.UU W2,W4,W8        ; W9:W8=W2*W4 (W9 is U's amplitude in part battery units)
1014    MUL.UU W3,W4,W2        ; W3:W2=W3*W4 (W2 is U's amplitude in full battery units)
1015    ADD.W W2,W9,W9         ; Add the two results together to get the real amplitude
1016    ASR.W W9,#9,W8        ; Transfer the integral battery amplitude to W8
1017    BRA Z,no_ufull        ; If no full batteries are on, just do normal PWM
1018    MOV.W #0x0004,W9       ; Address module #4/#1 with extra data
1019    MOV.W temp,W0          ;
1020    SUB.W W9,W0,W9         ;
1021    MOV.W W9,U1TXREG
1022    REPEAT #420
1023    NOP
1024    MOV.W W1,U1TXREG       ; Send the zero PWM duty cycle (full) out to module #4/#1
1025    REPEAT #420
1026    NOP
1027    no_ufull:
1028    ASR.W W9,#4,W9        ; Since W9 was up to 511, make it up to 31
1029    AND.W #0x001F,W9      ; Consider only fractional batteries
1030    BRA Z,upwm_done
1031    ADD.W W1,W9,W9         ; Add on the duty cycle to the PWM command with polarity
1032    MOV.W #0x0001,W1       ; Address module #1/#4 with extra data
1033    MOV.W temp,W0          ;
1034    ADD.W W1,W0,W1         ;
1035    MOV.W W1,U1TXREG
1036    REPEAT #420
1037    NOP
1038    MOV.W W9,U1TXREG       ; Send the PWM duty cycle out to module #1/#4
1039    REPEAT #420
1040    NOP
1041    upwm_done:
1042
1043    MOV.W #0x0060,W1       ; Attempt to turn module #2/#5 on positive
1044    SUB.W #0x0200,W5       ; If V's amplitude is below zero, invert it
1045    BRA NN,.,+6
1046    MOV.W #0x0040,W1       ; Turn module #2/#5 on negative instead
1047    NEG.W W5,W5           ; Negate the duty cycle
1048    MOV.W amp132l,W2       ; Recall the fractional battery amplitude
1049    MOV.W amp132h,W3       ; Recall the integral battery amplitude
1050    MUL.UU W2,W5,W8        ; W9:W8=W2*W5 (W9 is V's amplitude in part battery units)

```

```

1051     MUL.UU W3,W5,W2      ; W3:W2=W3*W5 (W2 is V's amplitude in full battery units)
1052     ADD.W W2,W9,W9      ; Add the two results together to get the real amplitude
1053     ASR.W W9,#9,W8      ; Transfer the integral battery amplitude to W8
1054     BRA Z,no_vfull      ; If no full batteries are on, just do normal PWM
1055     MOV.W #0x0005,W9     ; Address module #5/#2 with extra data
1056     MOV.W temp,W0        ;
1057     SUB.W W9,W0,W9       ;
1058     MOV.W W9,U1TXREG
1059     REPEAT #420
1060     NOP
1061     MOV.W W1,U1TXREG     ; Send the zero PWM duty cycle (full) out to module #5/#2
1062     REPEAT #420
1063     NOP
1064 no_vfull:
1065     ASR.W W9,#4,W9       ; Since W9 was up to 511, make it up to 31
1066     AND.W #0x001F,W9     ; Consider only fractional batteries
1067     BRA Z,vpwm_done
1068     ADD.W W1,W9,W9       ; Add on the duty cycle to the PWM command with polarity
1069     MOV.W #0x0002,W1     ; Address module #2/#5 with extra data
1070     MOV.W temp,W0        ;
1071     ADD.W W1,W0,W1       ;
1072     MOV.W W1,U1TXREG
1073     REPEAT #420
1074     NOP
1075     MOV.W W9,U1TXREG     ; Send the PWM duty cycle out to module #2/#5
1076     REPEAT #420
1077     NOP
1078 vpwm_done:
1079
1080     MOV.W #0x0060,W1     ; Attempt to turn module #3/#6 on positive
1081     SUB.W #0x0200,W6     ; If W's amplitude is below zero, invert it
1082     BRA NN,+.6
1083     MOV.W #0x0040,W1     ; Turn module #3/#6 on negative instead
1084     NEG.W W6,W6          ; Negate the duty cycle
1085     MOV.W amp132l,W2     ; Recall the fractional battery amplitude
1086     MOV.W amp132h,W3     ; Recall the integral battery amplitude
1087     MUL.UU W2,W6,W8      ; W9:W8=W2*W6 (W9 is W's amplitude in part battery units)
1088     MUL.UU W3,W6,W2      ; W3:W2=W3*W6 (W2 is W's amplitude in full battery units)
1089     ADD.W W2,W9,W9      ; Add the two results together to get the real amplitude
1090     ASR.W W9,#9,W8      ; Transfer the integral battery amplitude to W8
1091     BRA Z,no_wfull      ; If no full batteries are on, just do normal PWM
1092     MOV.W #0x0006,W9     ; Address module #6/#3 with extra data
1093     MOV.W temp,W0        ;
1094     SUB.W W9,W0,W9       ;
1095     MOV.W W9,U1TXREG
1096     REPEAT #420
1097     NOP
1098     MOV.W W1,U1TXREG     ; Send the zero PWM duty cycle (full) out to module #6/#3
1099     REPEAT #420
1100     NOP
1101 no_wfull:
1102     ASR.W W9,#4,W9       ; Since W9 was up to 511, make it up to 31
1103     AND.W #0x001F,W9     ; Consider only fractional batteries
1104     BRA Z,wpwm_done
1105     ADD.W W1,W9,W9       ; Add on the duty cycle to the PWM command with polarity
1106     MOV.W #0x0003,W1     ; Address module #3/#6 with extra data
1107     MOV.W temp,W0        ;
1108     ADD.W W1,W0,W1       ;
1109     MOV.W W1,U1TXREG
1110     REPEAT #420
1111     NOP
1112     MOV.W W9,U1TXREG     ; Send the PWM duty cycle out to module #3/#6

```

```

1113 REPEAT #420
1114 NOP
1115 wpwm_done:
1116
1117 BTG.W temp,#0 ;
1118 BTG.W temp,#1 ;
1119 RETURN
1120
1121 ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
1122
1123 output_charge: ; Generate module commands with no PWM for charging
1124 MUL.UU W1,W3,W6 ; W7:W6=W1*W3 (W7 is part of the low product, an angle)
1125 MUL.UU W1,W4,W4 ; W5:W4=W1*W4 (W4 is part of the high product, an angle)
1126 ADD.W W4,W7,W3 ; W3:W6 is the accumulated angle from frequency
1127 MOV.W angle32l,W1 ; Recall the old accumulated angle low word
1128 ADD.W W1,W6,W6 ; Add the old angle low word to the new angle low word
1129 MOV.W angle32h,W1 ; Recall the old accumulated angle high word
1130 MOV.W W1,angleold ; Save this angle for calculations below
1131 ADDC.W W1,W3,W3 ; Add the old angle high word to the new angle high word
1132 MOV.W W6,angle32l ; Save the accumulated angle low word for next time
1133 MOV.W W3,angle32h ; Save the accumulated angle high word for next time
1134 RCALL sine_table ; Return the sin(W3&0x03FF) in W6, 0x0001 to 0x03FF
1135 MOV.W ampl32l,W2 ; Recall the fractional battery amplitude
1136 MOV.W ampl32h,W3 ; Recall the integral battery amplitude
1137 MOV.W temp,W4 ; Recall the current output amplitude
1138 SUB.W #0x0200,W6 ; If the sine amplitude is below zero, invert it
1139 BRA NN,oc_pos
1140
1141 ; The charging output reference value is negative
1142 NEG.W W6,W6 ; Negate the sine amplitude if it's negative
1143 NEG.W W4,W4 ; Also negate the current output amplitude
1144 CPO.W W4 ; Is the current output amplitude positive?
1145 BRA N,oc_pos_off ; If so, turn off positives before turning on negatives
1146 MUL.UU W2,W6,W0 ; W1:W0=W2*W6 (W1 is the amplitude in part battery units)
1147 MUL.UU W3,W6,W2 ; W3:W2=W3*W6 (W2 is the amplitude in full battery units)
1148 ADD.W W1,W2,W2 ; Add the two results together to get the real amplitude
1149 SUB.W #256,W2 ; Subtract half a battery (the minimum threshold)
1150 BRA NN,+.4 ; If the reference value undersaturated, set it to zero
1151 CLR.W W2
1152 ASR.W W2,#9,W2 ; Divide the value by the sine wave's peak value (512)
1153 CP.W W2,W4 ; Compute the reference value minus the current value
1154 BRA NZ,+.4 ; If they're equal, do nothing and return immediately
1155 RETURN
1156 BRA N,oc_neg_off
1157 RCALL charge_on ; Find the least-charged battery of the whole group (W3)
1158 CPO.W W3 ; Were none found?
1159 BRA NZ,+.4 ; If none were found, return immediately
1160 RETURN
1161 DEC.W temp ; Decrement the current output voltage
1162 BSET.W W3,#7 ; Set bit seven to signify turning on negative
1163 BRA oc_done
1164 oc_neg_off:
1165 RCALL charge_off ; Find the most-charged battery of the whole group (W3)
1166 CPO.W W3 ; Were none found?
1167 BRA NZ,+.4 ; If none were found, return immediately
1168 RETURN
1169 INC.W temp ; Increment the current output voltage
1170 BRA oc_done
1171
1172 ; The charging output reference value is positive
1173 oc_pos:
1174 CPO.W W4 ; Is the current output amplitude negative?

```

```

1175     BRA N,oc_neg_off      ; If so, turn off negatives before turning on positives
1176     MUL.UU W2,W6,W0       ; W1:W0=W2*W6 (W1 is the amplitude in part battery units)
1177     MUL.UU W3,W6,W2       ; W3:W2=W3*W6 (W2 is the amplitude in full battery units)
1178     ADD.W W1,W2,W2        ; Add the two results together to get the real amplitude
1179     SUB.W #256,W2         ; Subtract half a battery (the minimum threshold)
1180     BRA NN,+.4            ; If the reference value undersaturated, set it to zero
1181 oc_pos_zero:
1182     CLR.W W2
1183     ASR.W W2,#9,W2        ; Divide the value by the sine wave's peak value (512)
1184     CP.W W2,W4             ; Compute the reference value minus the current value
1185     BRA NZ,+.4            ; If they're equal, do nothing and return immediately
1186     RETURN
1187     BRA N,oc_pos_off
1188     RCALL charge_on       ; Find the least-charged battery of the whole group (W3)
1189     CPO.W W3              ; Were none found?
1190     BRA NZ,+.4            ; If none were found, return immediately
1191     RETURN
1192     INC.W temp            ; Increment the current output voltage
1193     BSET.W W3,#8          ; Set bit eight to signify turning on negative
1194     BRA oc_done
1195 oc_pos_off:
1196     RCALL charge_off      ; Find the most-charged battery of the whole group (W3)
1197     CPO.W W3              ; Were none found?
1198     BRA NZ,+.4            ; If none were found, return immediately
1199     RETURN
1200     DEC.W temp            ; Decrement the current output voltage
1201
1202 oc_done:
1203     MOV.W #onlist,W2      ; W3 has the module command (off or on positive/negative)
1204     DEC.W W3,W4           ; Load the location of the battery module "on-list"
1205     ASR.W W4,#2,W5        ; Addresses in memory start at zero
1206     AND.W #0x001E,W5      ; Find the offset address of the status bits
1207     ADD.W W2,W5,W2        ; Mask off the module command and the lowest bit
1208     SL.W W4,#1,W5         ; W2 contains the status bits' address (word boundary)
1209     INC.W W5,W6           ; W5 contains the status bits' offset (crumb boundary)
1210     MOV.W #0x0180,W0      ; Also create a second offset
1211     AND.W W0,W3,W0        ; Test to see if this is an "off" command
1212     BRA Z,oc_off          ; If so, don't check for reversal; get the voltage word
1213                             ; sent afterward by the module, and save it in the array
1214     MOV.W #0x007F,W0      ; Load TBLPAG with 0x007F, the EEPROM location
1215     MOV.W W0,TBLPAG
1216     MOV.W #0xFCOF,W0      ; Load the beginning EEPROM address into W0
1217     ASR.W W4,#3,W1        ; Find the offset address of the reversal bit
1218     AND.W #0x000E,W1      ; Mask off the module command and the lowest bit
1219     ADD.W W0,W1,W0
1220     TBLRDL.W [W0],W1      ; Load the reversal-specifying EEPROM word into W1
1221     BTST.C W1,W3          ; Test the bit associated with this module
1222     BRA NC,+.6
1223     BTG.W W3,#7           ; Toggle bits seven and eight because of reversal
1224     BTG.W W3,#8
1225     BTST.Z W3,#7         ; Copy these polarity bits to the "on-list"
1226     BTST.C W3,#8
1227     BSW.Z [W2],W5        ; Signify that this module is either on pos. or neg.
1228     BSW.C [W2],W6
1229     MOV.W W3,U1TXREG      ; Send a turn on positive or turn on negative command
1230     REPEAT #420
1231     NOP
1232     RETURN
1233
1234 oc_off:
1235     BTST.Z W3,#7          ; The module will be turned off; get the voltage word
1236     BTST.C W3,#8          ; Copy these polarity bits to the "on-list"

```



```

1237 BSW.Z [W2],W5 ; Signify that this module is now off
1238 BSW.C [W2],W6
1239 ADD.W #0x0180,W3
1240 MOV.W W3,U1TXREG ; Send a turn off command to the addressed module
1241 REPEAT #1000
1242 NOP
1243 BTST.W U1STA,#URXDA ; Is there at least one available word?
1244 BRA NZ,.+4 ; If not, return immediately
1245 RETURN
1246 MOV.W U1RXREG,W1 ; Recall the received word
1247 AND.W #0x01FF,W1 ; Mask off the top seven bits
1248 SL.W W1,#1,W1 ; Multiply this result by two (one bit is not sent now)
1249 MOV.W W3,W0 ; This is for vc_entering (it needs the module address)
1250 AND.W #0x007F,W0 ; Mask off bits seven and eight
1251 BRA vc_entering ; Send execution to the routine below (slave voltage A/D)
1252
1253 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1254
1255 voltcheck: ; Incrementally check the battery voltages and store
1256 MOV.W #0x0001,W0 ; Set the baud rate to 1MBd
1257 MOV.W W0,U1BRG
1258 MOV.W batupdate,W0 ; Recall the current battery address to query
1259 INC.W W0,W0
1260 MOV.W W0,U1TXREG ; Send an extended data command to the addressed module
1261 REPEAT #420
1262 NOP
1263 MOV.W #0x0008,W1 ; Send a battery voltage low byte request
1264 MOV.W W1,U1TXREG
1265 REPEAT #1100
1266 NOP
1267 BTST.W U1STA,#URXDA ; Is there at least one available word?
1268 BRA Z,vc_skipread
1269 MOV.W U1RXREG,W1 ; Recall the received word
1270 AND.W #0x00FF,W1 ; Mask off the top eight bits
1271 MOV.W W0,U1TXREG ; Send an extended data command to the addressed module
1272 REPEAT #420
1273 NOP
1274 MOV.W #0x0009,W2 ; Send a battery voltage high byte request
1275 MOV.W W2,U1TXREG
1276 REPEAT #1100
1277 NOP
1278 BTST.W U1STA,#URXDA ; Is there at least one available word?
1279 BRA Z,vc_skipread
1280 MOV.W U1RXREG,W2 ; Recall the received word
1281 AND.W #0x0003,W2 ; Mask off the lower two bits
1282 SWAP.W W2
1283 ADD.W W1,W2,W1 ; Add the high byte to the low byte for a 10-bit result
1284
1285 vc_entering: ; This may be called by other functions
1286 SL.W W1,#4,W1 ; Scale this by 16 to get more constant precision
1287 MOV.W #38912,W2 ; This is a scale factor to reach mV
1288 MUL.UU W1,W2,W4 ; W5:W4=W1*W2 (W5 is the new relative voltage in mV)
1289 MOV.W #7500,W4 ; Find the new absolute voltage in mV (add 7.5V)
1290 ADD.W W4,W5,W6
1291 SL.W W0,#1,W0 ; Multiply the address by two
1292 DEC2.W W0,W0
1293 MOV.W #batvolts,W1 ; Find the appropriate array address
1294 ADD.W W0,W1,W1
1295 MOV.W [W1],W4 ; Recall the previous voltage
1296 LSR.W W4,#2,W4 ; Divide it by four
1297 MUL.UU W4,#7,W4 ; Weight it by seven
1298 ADD.W #4,W4 ; Add one-half (about four)

```

```

1299     LSR.W W6,#2,W6      ; Divide the new value by four also
1300     ADD.W W4,W6,W6      ; Average the new and old battery voltages
1301     LSR.W W6,#1,W6      ; Divide the new total by two
1302     MOV.W W6,[W1]        ; Store the final value in the voltage array
1303
1304     MOV.W #11800,W1      ; Find the percent SOC (0-100%)
1305     SUB.W W6,W1,W6      ; Subtract off the zero-SOC voltage (11.8V)
1306     BRA NN,+.4          ; If the measured voltage was lower, assume 0% SOC
1307     CLR.W W6
1308     MOV.W #2400,W1       ; Compare the measured voltage with 14.2V
1309     CP.W W6,W1          ; Perform W6-W1
1310     BRA N,+.4
1311     MOV.W #2400,W6       ; Saturate to 14.2V if above (assume 100% SOC)
1312     MOV.W #27,W1        ; This is a scale factor to reach SOC percent high
1313     MUL.UU W1,W6,W4      ; W5:W4=W1*W6 (W4 is the new SOC percent high)
1314     MOV.W #20700,W1     ; This is a scale factor to reach SOC percent low
1315     MUL.UU W1,W6,W2     ; W3:W2=W1*W6 (W3:W2 is the new SOC percent high/low)
1316     ADD.W W3,W4,W3      ; Do a 32-bit calculation for more precision
1317     INC.W W3,W3         ; Make the maximum SOC high word 65535
1318     SL.W W0,#1,W0       ; Multiply the address by two again (four total)
1319     MOV.W #batsoc,W1    ; Find the appropriate array address
1320     ADD.W W0,W1,W1
1321     MOV.D W2,[W1]       ; Store the final value in the percent SOC array
1322
1323     vc_done:
1324     MOV.W batupdate,W0   ; Recall the current battery address to query
1325     INC.W W0,W0          ; Increment the randomness counter and update it
1326     MOV.W W0,W1
1327     SUB.W #120,W1        ; Has the address overflowed?
1328     BRA N,+.4
1329     CLR.W W0             ; If so, set it back to zero
1330     MOV.W W0,batupdate
1331     RETURN
1332
1333     vc_skipread:        ; This battery module is not present; zero voltage/SOC
1334     SL.W W0,#1,W0       ; Multiply the address by two
1335     DEC2.W W0,W0
1336     MOV.W #batvolts,W1  ; Find the appropriate array address
1337     ADD.W W0,W1,W1
1338     CLR.W [W1]          ; Store the final value in the voltage array
1339     SL.W W0,#1,W0       ; Multiply the address by two again (four total)
1340     MOV.W #batsoc,W1    ; Find the appropriate array address
1341     ADD.W W0,W1,W1
1342     CLR.W [W1++]        ; Store the final value in the percent SOC array
1343     CLR.W [W1]
1344     BRA vc_done
1345
1346     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1347
1348     convert_curs:       ; Convert the current A/D channels
1349     MOV.W #0x0006,W1    ; Select channel AN6 for A/D conversion (current U)
1350     MOV.W W1,ADCHS
1351     BSET.W ADCON1,#SAMP ; Start an A/D conversion
1352     BTSS.W ADCON1,#DONE
1353     BRA .-2            ; Keep looping until it completes
1354     BCLR.W ADCON1,#DONE
1355     MOV.W ADCBUFO,W1    ; Weight the current value by one but make it counts*four
1356     SL.W W1,#2,W1       ; Multiply the ten-bit integer result by four
1357     MOV.W curcentu,W2   ; Recall the center value (zero amps)
1358     SUB.W W1,W2,W1      ; Perform W1=W1-W2
1359     MOV.W W1,curu       ; Store the result (two's complement format, counts*four)
1360

```

```

1361 MOV.W #0x0007,W1      ; Select channel AN7 for A/D conversion (current V)
1362 MOV.W W1,ADCHS
1363 BSET.W ADCON1,#SAMP    ; Start an A/D conversion
1364 BTSS.W ADCON1,#DONE
1365 BRA .-2                ; Keep looping until it completes
1366 BCLR.W ADCON1,#DONE
1367 MOV.W ADCBUF0,W1       ; Weight the current value by one but make it counts*four
1368 SL.W W1,#2,W1          ; Multiply the ten-bit integer result by four
1369 MOV.W curcentv,W2       ; Recall the center value (zero amps)
1370 SUB.W W1,W2,W1         ; Perform W1=W1-W2
1371 MOV.W W1,curv          ; Store the result (two's complement format, counts*four)
1372
1373 MOV.W #0x0008,W1       ; Select channel AN8 for A/D conversion (current W)
1374 MOV.W W1,ADCHS
1375 BSET.W ADCON1,#SAMP    ; Start an A/D conversion
1376 BTSS.W ADCON1,#DONE
1377 BRA .-2                ; Keep looping until it completes
1378 BCLR.W ADCON1,#DONE
1379 MOV.W ADCBUF0,W1       ; Weight the current value by one but make it counts*four
1380 SL.W W1,#2,W1          ; Multiply the ten-bit integer result by four
1381 MOV.W curcentw,W2       ; Recall the center value (zero amps)
1382 SUB.W W1,W2,W1         ; Perform W1=W1-W2
1383 MOV.W W1,curw          ; Store the result (two's complement format, counts*four)
1384
1385 RETURN
1386
1387 ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
1388
1389 clarke:
1390 ; PERFORM A CLARKE TRANSFORM OF THE CURRENTS JUST MEASURED
1391 ;
1392 ; Formulas:
1393 ; curalpha = curu-((curu+curv+curw)/3)
1394 ; curbeta = (curu+2*curv-(curu+curv+curw))/sqrt(3) = (curv-curw)/sqrt(3)
1395 ;
1396 ; Note:
1397 ; Because curw is not normally used in the calculations, it can be utilized
1398 ; here to center the currents properly. The sum of all three currents should
1399 ; be zero; if not, there is an offset error. Assuming the offset error is
1400 ; equal between the three sensors (which it should be approximately), the
1401 ; formulas above cancel it out without destroying the original readings.
1402
1403 MOV.W curu,W1           ; Recall the current U reading in counts*4
1404 MOV.W curv,W2           ; Recall the current V reading in counts*4
1405 MOV.W curw,W3           ; Recall the current W reading in counts*4
1406 ADD.W W1,W2,W4          ; Sum the three together to find the offset error
1407 ADD.W W3,W4,W4
1408 MOV.W #0x5555,W5        ; Load one-third into W5 as a multiply constant
1409 MUL.SU W4,W5,W4          ; W5:W4=W4*W5 (take one-third of the total offset error)
1410 MOV.W #0x8000,W9        ; Add one-half to get better accuracy (+/-0.5 vs. +/-1.0)
1411 ADD.W W4,W9,W4
1412 ADDC.W #0,W5
1413 SUB.W W1,W5,W1          ; Calculate curu-((curu+curv+curw)/3)
1414 MOV.W W1,curalpha       ; Store in curalpha
1415 SUB.W W2,W3,W2          ; Calculate curv-curw
1416 MOV.W #0x93CD,W5        ; Load 1/sqrt(3) into W5 as a multiply constant
1417 MUL.SU W2,W5,W4          ; W5:W4=W2*W5 (divide (curv-curw) by sqrt(3))
1418 ADD.W W4,W9,W4
1419 ADDC.W #0,W5
1420 MOV.W W5,curbeta        ; Store in curbeta
1421 RETURN
1422

```

```

1423 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1424
1425 park:
1426 ; PERFORM A PARK TRANSFORM OF THE CURRENTS JUST CALCULATED
1427 ;
1428 ; The alpha-beta currents aren't too useful without being oriented to the flux
1429 ; angle. This angle, when moving, basically represents the synchronous speed
1430 ; of the motor with Id and Iq that we can control. Id (current along the D-
1431 ; axis) is the magnetizing current, and Iq (current along the Q-axis) controls
1432 ; the torque. Id is constant and Iq is proportional to the throttle reading.
1433 ;
1434 ; Formulas: curd = -curalpha*cos(angle32h)+curbeta*sin(angle32h)
1435 ;             curq = curalpha*sin(angle32h)+curbeta*cos(angle32h)
1436
1437 MOV.W angleold,W3 ; Recall the old accumulated angle high word
1438 RCALL sine_table ; Return the sin(W3&0x03FF) in W6, 0x0001 to 0x03FF
1439 SUB.W #0x0200,W6 ; Change the range of 1 to 1023 to -511 to 511
1440 SL.W W6,#6,W1 ; Change the -511 to 511 value into -32704 to 32704
1441 ADD.W #32,W1 ; Add one-half for accuracy
1442 ADD.W #256,W3 ; Add about 90 degrees to change sine into cosine
1443 RCALL sine_table ; Return the sin(W3&0x03FF) in W6, 0x0001 to 0x03FF
1444 SUB.W #0x0200,W6 ; Change the range of 1 to 1023 to -511 to 511
1445 SL.W W6,#6,W6 ; Change the -511 to 511 value into -32704 to 32704
1446 ADD.W #32,W6 ; Add one-half for accuracy
1447 MOV.W curalpha,W7
1448 MOV.W curbeta,W8
1449 MOV.W #0x8000,W9 ; Add one-half to get better accuracy (+/-0.5 vs. +/-1.0)
1450 MOV.W #24000,W0 ; This is a scale factor to reach amps*4 at the end
1451
1452 MUL.SS W6,W7,W2 ; W3:W2=W6*W7 (curalpha*cos(angle32h))
1453 ADD.W W2,W9,W2
1454 ADDC.W #0,W3
1455 MUL.SS W1,W8,W4 ; W5:W4=W1*W8 (curbeta*sin(angle32h))
1456 ADD.W W4,W9,W4
1457 ADDC.W #0,W5
1458 SUB.W W5,W3,W3 ; Calculate curbeta*sin(angle32h)-curalpha*cos(angle32h)
1459 MOV.W curd32h,W5 ; Load the old current D into both accumulators
1460 LAC W5,#0,A
1461 LAC W5,#0,B
1462 MOV.W curd32l,W4 ; Note that the sine and cosine here are +/-0.5 maximum
1463 MOV.W W4,ACCAL
1464 MOV.W W4,ACCBL
1465 SFTAC B,#6 ; Divide accumulator B by 64
1466 SUB A ; Subtract this from the original value for a 63/64 ratio
1467 LAC W3,#0,B ; Load the new value into accumulator B
1468 SFTAC B,#7 ; Divide the 65536 multiplication by 128 for times 512
1469 ADD A ; Sum these two weighted parts
1470 MOV.W ACCAL,W4 ; Save this value for next time (with great accuracy)
1471 MOV.W W4,curd32l
1472 MOV.W ACCAH,W5
1473 MOV.W W5,curd32h
1474 SFTAC A,#14 ; Divide accumulator A by 64*256 for a 16-bit result
1475 MOV.W ACCAL,W3
1476 MUL.SU W3,W0,W4 ; W5:W4=W3*W0 (W5 is the new D current in amps*4)
1477 BTSC.W PORTD,#3 ; Is the "Reverse" signal active?
1478 NEG.W W5,W5 ; If so, negate the D current
1479 MOV.W W5,curd
1480
1481 MUL.SS W1,W7,W2 ; W3:W2=W6*W7 (curalpha*sin(angle32h))
1482 ADD.W W2,W9,W2
1483 ADDC.W #0,W3
1484 MUL.SS W6,W8,W4 ; W5:W4=W1*W8 (curbeta*cos(angle32h))

```

```

1485     ADD.W W4,W9,W4
1486     ADDC.W #0,W5
1487     ADD.W W3,W5,W3      ; Calculate curalpha*sin(angle32h)+curbeta*cos(angle32h)
1488     MOV.W curq32h,W5    ; Load the old current Q into both accumulators
1489     LAC W5,#0,A
1490     LAC W5,#0,B
1491     MOV.W curq32l,W4     ; Note that the sine and cosine here are +/-0.5 maximum
1492     MOV.W W4,ACCAL
1493     MOV.W W4,ACCB
1494     SFTAC B,#6           ; Divide accumulator B by 64
1495     SUB A                ; Subtract this from the original value for a 63/64 ratio
1496     LAC W3,#0,B         ; Load the new value into accumulator B
1497     SFTAC B,#7           ; Divide the 65536 multiplication by 128 for times 512
1498     ADD A               ; Sum these two weighted parts
1499     MOV.W ACCAL,W4       ; Save this value for next time (with great accuracy)
1500     MOV.W W4,curq32l
1501     MOV.W ACCAH,W5
1502     MOV.W W5,curq32h
1503     SFTAC A,#14          ; Divide accumulator A by 64*256 for a 16-bit result
1504     MOV.W ACCAL,W3
1505     MUL.SU W3,W0,W4      ; W5:W4=W3*W0 (W5 is the new Q current in amps*4)
1506     MOV.W W5,curq
1507
1508     RETURN
1509
1510     ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
1511
1512     find_rms:
1513     ; Calculate the 64-point averaged RMS phase current in amps*4 from Id and Iq
1514     MOV.W curd,W5        ; Recall the averaged current D in amps*4
1515     MOV.W curq,W7        ; Recall the averaged current Q in amps*4
1516     BSET.W CORCON,#IF    ; Enable integer mult. (as opposed to fractional)
1517     MPY W5*W5,A          ; Square the D current to accumulator A
1518     MAC W7*W7,A          ; Square the Q current and add to accumulator A
1519     SFTAC A,#-7          ; Pretend to multiply by 256; account for Id/q vs. Iu/v/w
1520     ; The result high word can range from 0 to 4095, unsigned. In order to get a
1521     ; better relative precision result without a huge lookup table, the magnitude
1522     ; of the result will be recorded, divided by two, then added back to the
1523     ; result of the square root. The magnitude will be in powers of four, to
1524     ; simplify the readdition of the magnitude. Magnitudes from 4^5 to 4^0 are
1525     ; supported. The most significant 10 bits of the value are used by a lookup
1526     ; table to find the square root value (the result has a scaling of 32). The
1527     ; first quarter of the table is used only for 4^0 for better accuracy.
1528     MOV.W ACCAH,W1       ; Extract the high word from accumulator A
1529     FFIL W1,W2           ; W2|4: 5/6|5, 7/8|4, 9/10|3, 11/12|2, 13/14|1, 15/16|0|0
1530     BRA NC,#+4           ; If W2 ends up to be zero, the carry will be set
1531     MOV.W #16,W2         ; If W1 was zero, assume 4^0 scaling
1532     SUB.W W2,#7,W3       ; W3|4^x: -2/-1|5, 0/1|4, 2/3|3, 4/5|2, 6/7|1, 8/9|0
1533     BCLR.W W3,#0        ; W3|4^x now: -2|5, 0|4, 2|3, 4|2, 6|1, 8|0
1534     NEG.W W3,W3          ; W3|4^x now: 2|5, 0|4, -2|3, -4|2, -6|1, -8|0
1535     SFTAC A,W3           ; Shift the 10 significant bits to the low part of ACCAH
1536     MOV.W ACCAH,W1       ; Again extract the high word from accumulator A
1537     RCALL sqrt_table     ; Return the sqrt(W1&0x03FF) in W1, 0x0000 to 0x03FF
1538     SUB.W W2,#5,W2       ; W2|4^x now: 0/1|5, 2/3|4, 4/5|3, 6/7|2, 8/9|1, 10/11|0
1539     ASR.W W2,#1,W2       ; W2|4^x now: 0|5, 1|4, 2|3, 3|2, 4|1, 5|0
1540     ASR.W W1,W2,W1       ; Divide the square root by up to 32 (the table scaling)
1541     MOV.W W1,currms      ; The phase current amplitude in amps*4 (0.00-255.75Arms)
1542     RETURN
1543
1544     ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
1545
1546     find_pf:             ; Find the instantaneous power factor and average it

```

```

1547 ; Calculate the power factor, stored to a scale of 1000 to pwrfact
1548 MOV.W curd,W2 ; Recall the calculated currents D and Q
1549 MOV.W curq,W4
1550 CPO.W W2 ; Saturate curd to at least zero
1551 BRA NN, .+4
1552 CLR.W W2
1553 MOV.W W2,W3 ; Copy curd into W3; thus multiply W2 by 256 into W3:W2
1554 AND.W #0xFF,W2 ; Only retain the original W2 low byte
1555 SWAP.W W2 ; Swap high and low bytes of W2
1556 SWAP.W W3 ; Swap high and low bytes of W3
1557 AND.W #0xFF,W3 ; Only retain the original W2 high byte
1558 CPO.W W4 ; If curq is zero or under, saturate to 1023
1559 BRA LE, .+14
1560 REPEAT #17 ; W3:W2/W4 quotient to W0, W3:W2/W4 remainder to W1
1561 DIV.UD W2,W4
1562 BRA OV, .+8 ; If there was an overflow, saturate to 1023
1563 MOV.W W0,W1
1564 SUB.W #0x03FF,W1 ; Was the result over 1023?
1565 BRA N, .+4
1566 MOV.W #0x03FF,W0 ; If so, saturate to 1023 (on three separate conditions)
1567 RCALL pf_table ; Return the cos(atan(W0&0x03FF)) in W1, 0x00F2 to 0x03E8
1568 MOV.W W1,pwrfact
1569
1570 ; Take an average of the power factor for slower (more stable) corrections
1571 MOV.W pfavg32h,W5 ; Load the old power factor average into the accumulators
1572 LAC W5,#0,A
1573 LAC W5,#0,B
1574 MOV.W pfavg32l,W4
1575 MOV.W W4,ACCAL
1576 MOV.W W4,ACCBL
1577 SFTAC B,#10 ; Divide accumulator B by 1024
1578 SUB A ; Subtract this from the original value for 1023/1024
1579 LAC W1,#0,B ; Load the new power factor value into accumulator B
1580 SFTAC B,#10 ; Divide the 65536 multiplication by 1024 for times 64
1581 ADD A ; Sum these two weighted parts
1582 MOV.W ACCAL,W4 ; Save this value for next time (with great accuracy)
1583 MOV.W W4,pfavg32l
1584 MOV.W ACCAH,W5
1585 MOV.W W5,pfavg32h ; This can be compared to pwrfact (it is scaled the same)
1586
1587 RETURN
1588
1589 ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
1590
1591 motor_loop: ; Close the loop for motoring
1592 ; Close the loop - adjust the applied voltage to set the D current to optimal
1593 MOV.W TMR3,W1 ; Recall the current 32-bit timer high word
1594 MOV.W nextadjust,W2 ; Recall the calculated next time to adjust
1595 CP.W W1,W2 ; If they aren't the same, don't adjust
1596 BRA NZ,noadjust ; Only close the loop every LOOP*2.18453ms
1597 MOV.W pc_loop,W2
1598 ADD.W W1,W2,W1 ; Calculate and store the next time to close the loop
1599 MOV.W W1,nextadjust
1600
1601 MOV.W curd,W1 ; Recall the averaged D current
1602 MOV.W #IDOPTIMAL,W2 ; Compare it to the optimal D current
1603 SUB.W W2,W1,W1 ; Calculate the ID command minus the measured ID
1604 MOV.W pc_idkp,W4 ; Multiply the ID difference by the proportional constant
1605 MUL.SU W1,W4,W6 ; W7:W6=W1*W4 (amplitude adjustment due to proportion)
1606 MOV.W W6,W4
1607 MOV.W ampl32l,W2 ; Recall the current amplitude
1608 MOV.W ampl32h,W3

```

```

1609 CPO.W W4 ; Test the sign of the PF difference in voltage units
1610 BRA NN, .+10
1611 NEG.W W4, W4 ; Negate the difference so it can be subtracted
1612 SUB.W W2, W4, W2 ; The measured was less than the command (less volts)
1613 SUBB.W #0, W3
1614 BRA .+6
1615 ADD.W W2, W4, W2 ; The measured was more than the command (more volts)
1616 ADDC.W #0, W3
1617
1618 MOV.W vmin32l, W6 ; Compare the new amplitude to the minimum at this speed
1619 MOV.W vmin32h, W7
1620 SUB.W W2, W6, W8 ; Subtract the minimum low word from the new low word
1621 SUBB.W W3, W7, W9 ; Subtract the minimum high word from the new high word
1622 BRA NN, .+4 ; If the calculated amplitude is legal, don't adjust it
1623 MOV.D W6, W2 ; Replace the calculated amplitude with the minimum
1624 MOV.W vmax32l, W6 ; Compare the new amplitude to the maximum at this speed
1625 MOV.W vmax32h, W7
1626 SUB.W W6, W2, W8 ; Subtract the new low word from the maximum low word
1627 SUBB.W W7, W3, W9 ; Subtract the new high word from the maximum high word
1628 BRA NN, .+4 ; If the calculated amplitude is legal, don't adjust it
1629 MOV.D W6, W2 ; Replace the calculated amplitude with the maximum
1630 MOV.W W2, ampl32l
1631 MOV.W W3, ampl32h
1632
1633 BRA noadjust
1634
1635 ; Also close the loop by adjusting the frequency based on the desired torque
1636 MOV.W throttle, W1 ; Recall the throttle command (0-1023)
1637 SUB.W #MINTHR, W1 ; Subtract off the minimum throttle value
1638 ASR.W W1, #3, W1 ; Change the 0 to 895 value to 0 to 111 (Iq command)
1639 MOV.W curq, W2 ; Recall the averaged current Q
1640 SUB.W W1, W2, W1 ; Calculate the Iq command minus the measured Iq
1641 MOV.W #IQKP, W2 ; Multiply the Iq difference by the proportional constant
1642 MUL.SU W1, W2, W2 ; W3:W2=W1*W2 (W3=frequency adjustment due to proportion)
1643 MOV.W frequency, W1 ; Recall the current frequency
1644 CPO.W W3 ; Test the sign of the Iq difference in frequency units
1645 BRA Z, noadjust ; If the difference is zero, don't adjust the frequency
1646 BRA NN, .+6
1647 DEC.W W1, W1 ; The command was less than the measured (less speed)
1648 BRA .+4
1649 INC.W W1, W1 ; The command was more than the measured (more speed)
1650 MOV.W #MINFREQ, W2
1651 CP.W W1, W2 ; Test to see if the frequency is too low
1652 BRA GEU, .+4
1653 MOV.W W2, W1 ; If so, limit the frequency based on MINFREQ
1654 MOV.W #MAXFREQ, W2
1655 CP.W W1, W2 ; Test to see if the frequency is too high
1656 BRA LEU, .+4
1657 MOV.W W2, W1 ; If so, limit the frequency based on MAXFREQ
1658 MOV.W W1, frequency
1659
1660 noadjust:
1661 RETURN
1662
1663 ;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
1664
1665 send_vars:
1666 ; Convert many variables to one byte in order to send out to the PIC18F2553
1667
1668 MOV.W frequency, W1 ; Recall the frequency word
1669 MOV.W #31250, W2 ; This is a scale factor to reach Hz from Hz*2.097152
1670 MUL.UU W1, W2, W4 ; W5:W4=W1*W2 (W5 is the new frequency in Hz)

```

```

1671     MOV.W W5,W0
1672     MOV.B WREG,freqbyte
1673
1674     MOV.W ampl32l,W0
1675     LSR.W W0,#11,W0
1676     BTSC.W ampl32h,#0
1677     BSET.W W0,#5
1678     MOV.B WREG,amplitude
1679
1680     MOV.W vmin32l,W0
1681     LSR.W W0,#11,W0
1682     BTSC.W vmin32h,#0
1683     BSET.W W0,#5
1684     MOV.B WREG,amplmin
1685
1686     MOV.W vmax32l,W0
1687     LSR.W W0,#11,W0
1688     BTSC.W vmax32h,#0
1689     BSET.W W0,#5
1690     MOV.B WREG,amplmax
1691
1692     MOV.W curu,W1           ; Recall the U (A) current word in counts*4
1693     MOV.W curv,W2           ; Recall the V (B) current word in counts*4
1694     MOV.W curw,W3           ; Recall the W (C) current word in counts*4
1695     MOV.W #3000,W6          ; This is a scale factor to reach amps/2 at the end
1696     MUL.SU W1,W6,W4          ; W5:W4=W1*W6 (W5 is the new U current in amps/2)
1697     MOV.W W5,W0
1698     MOV.B WREG,curubyte
1699     MUL.SU W2,W6,W4          ; W5:W4=W2*W6 (W5 is the new V current in amps/2)
1700     MOV.W W5,W0
1701     MOV.B WREG,curvbyte
1702     MUL.SU W3,W6,W4          ; W5:W4=W3*W6 (W5 is the new W current in amps/2)
1703     MOV.W W5,W0
1704     MOV.B WREG,curwbyte
1705
1706     MOV.W curd,W0           ; Recall the D current word in amps*4
1707     ASR.W W0,#3,W0           ; Convert it to amps/2
1708     MOV.B WREG,curdbyte
1709
1710     MOV.W curq,W0           ; Recall the Q current word in amps*4
1711     ASR.W W0,#3,W0           ; Convert it to amps/2
1712     MOV.B WREG,curqbyte
1713
1714     MOV.W currms,W0          ; Recall the RMS phase current word in amps*4
1715     ASR.W W0,#3,W0          ; Convert it to amps/2
1716     MOV.B WREG,rmsbyte
1717
1718     MOV.W pfavg32h,W1        ; Recall the power factor in value*1000
1719     MOV.W #16777,W2          ; This is a scale factor to reach value*256
1720     MUL.UU W1,W2,W4          ; W5:W4=W1*W2 (W5 is the new power factor in value*256)
1721     MOV.W W5,W0
1722     MOV.B WREG,pfbyte
1723
1724     MOV.W angle32h,W0         ; Recall the angle word in degrees*1024/360
1725     ASR.W W0,#2,W0           ; Convert it to degrees*256/360
1726     MOV.B WREG,angbyte
1727
1728     MOV.W TMR3,W0            ; Recall the current timer high word
1729     MOV.B WREG,timebyte
1730
1731 ; Take care of linebyte
1732     MOV.W #freqbyte,W2       ; Send almost the entire section data to the PIC18F2553

```



```

1733     AND.W #0x00FF,W2
1734     MOV.W W2,U2TXREG
1735     REPEAT #190
1736     NOP
1737     MOV.W #freqbyte,W2
1738     DO #14,.,+14          ; Send 14 data byte commands to the PIC18F2553
1739     MOV.B [W2++],W1
1740     BSET.W W1,#8
1741     MOV.W W1,U2TXREG
1742     REPEAT #190
1743     NOP
1744     NOP
1745
1746     RETURN
1747
1748     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1749
1750 handle_USB:                ; Handle USB programming commands
1751     MOV.B progcmd,WREG      ; Test the DSPIC programming command
1752     CPO.B W0                ; Is there anything here?
1753     BRA Z,pp_none
1754
1755     CP.B W0,#1              ; Is this an erase program memory row operation request?
1756     BRA NZ,pp_not1
1757     MOV.W addressl,W0       ; Recall the low address word and store in NVMADR
1758     MOV.W W0,NVMADR
1759     MOV.W addressh,W0       ; Recall the high address word and store in NVMADRU
1760     MOV.W W0,NVMADRU
1761     MOV.W #0x4041,W0        ; Set up NVMCON for erasing a program memory row
1762     MOV.W W0,NVMCON
1763     BRA pp_prog
1764
1765 pp_not1:
1766     CP.B W0,#2              ; Is this a write program memory row operation request?
1767     BRA NZ,pp_not2
1768     MOV.W addressl,W2       ; Recall the low address word and store in NVMADR
1769     MOV.W W2,NVMADR
1770     MOV.W addressh,W0       ; Recall the high address word and store in NVMADRU
1771     MOV.W W0,NVMADRU
1772     MOV.W W0,TBLPAG         ; Also store it in TBLPAG for the write latches
1773     MOV.W #flashbuf,W1      ; Write 96 bytes of flash memory to the write latches
1774     DO #31,pp2_end          ; Loop 16 times for a total of 32 instruction words
1775     TBLWTL.B [W1++],[W2++]
1776     TBLWTL.B [W1++],[W2--]
1777     TBLWTH.B [W1++],[W2]
1778     INC2.W W2,W2
1779 pp2_end:
1780     NOP
1781     MOV.W #0x4001,W0        ; Set up NVMCON for writing a program memory row
1782     MOV.W W0,NVMCON
1783     BRA pp_prog
1784
1785 pp_not2:
1786     CP.B W0,#3              ; Is this an erase data memory word operation request?
1787     BRA NZ,pp_not3
1788     MOV.W addressl,W0       ; Recall the low address word and store in NVMADR
1789     MOV.W W0,NVMADR
1790     MOV.W addressh,W0       ; Recall the high address word and store in NVMADRU
1791     MOV.W W0,NVMADRU
1792     MOV.W #0x4044,W0        ; Set up NVMCON for erasing a data memory word
1793     MOV.W W0,NVMCON
1794     BRA pp_prog

```

```

1795
1796 pp_not3:
1797     CP.B W0,#4           ; Is this an erase data memory row operation request?
1798     BRA NZ,pp_not4
1799     MOV.W addressl,W0    ; Recall the low address word and store in NVMAADR
1800     MOV.W W0,NVMAADR
1801     MOV.W addressh,W0    ; Recall the high address word and store in NVMAADR
1802     MOV.W W0,NVMAADR
1803     MOV.W #0x4045,W0     ; Set up NVMCN for erasing a data memory row
1804     MOV.W W0,NVMCN
1805     BRA pp_prog
1806
1807 pp_not4:
1808     CP.B W0,#5           ; Is this a write data memory word operation request?
1809     BRA NZ,pp_not5
1810     MOV.W addressl,W0    ; Recall the low address word
1811     MOV.W addressh,W1    ; Recall the high address word and store in TBLPAG
1812     MOV.W W1,TBLPAG
1813     MOV.W #progdata,W2   ; Write the data word to the temporary buffer
1814     TBLWTL.W [W2],[W0]
1815     MOV.W #0x4004,W0     ; Set up NVMCN for writing a data memory word
1816     MOV.W W0,NVMCN
1817     BRA pp_prog
1818
1819 pp_not5:
1820     CP.B W0,#6           ; Is this a write data memory row operation request?
1821     BRA NZ,pp_not6
1822     MOV.W addressl,W2    ; Recall the low address word and store in NVMAADR
1823     MOV.W W2,NVMAADR
1824     MOV.W addressh,W0    ; Recall the high address word and store in NVMAADR
1825     MOV.W W0,NVMAADR
1826     MOV.W W0,TBLPAG     ; Also store it in TBLPAG for the write latches
1827     MOV.W #flashbuf,W1   ; Write 96 bytes of flash memory to the write latches
1828     REPEAT #15           ; Loop 16 times for a total of 16 data EEPROM words
1829     TBLWTL.W [W1++],[W2++]
1830     MOV.W #0x400A,W0     ; Set up NVMCN for writing a data memory row
1831     MOV.W W0,NVMCN
1832     BRA pp_prog
1833
1834 pp_not6:
1835     CP.B W0,#7           ; Is this a write config. memory operation request?
1836     BRA NZ,pp_not7
1837     MOV.W addressl,W0    ; Recall the low address word
1838     MOV.W addressh,W1    ; Recall the high address word and store in TBLPAG
1839     MOV.W W1,TBLPAG
1840     MOV.W #progdata,W2   ; Write the data word to the temporary buffer
1841     TBLWTL.W [W2],[W0]
1842     MOV.W #0x4008,W0     ; Set up NVMCN for writing a configuration memory word
1843     MOV.W W0,NVMCN
1844     BRA pp_prog
1845
1846 pp_not7:
1847     CP.B W0,#8           ; Is this a read memory long operation request?
1848     BRA NZ,pp_not8
1849     MOV.W addressl,W1    ; Recall the low address of the flash/EEPROM to read
1850     MOV.W addressh,W2    ; Recall the high address of the flash/EEPROM to read
1851     MOV.W W2,TBLPAG
1852     MOV.W #dataout,W2    ; Write six bytes of flash/EEPROM memory to the buffer
1853     TBLRDL.B [W1++],[W2++]
1854     TBLRDL.B [W1--],[W2++]
1855     TBLRDH.B [W1],[W2++]
1856     INC2.W W1,W1

```

```

1857     TBLRD.L.B [W1++],[W2++]
1858     TBLRD.L.B [W1--],[W2++]
1859     TBLRD.H.B [W1],[W2++]
1860     MOV.W #dataout,W2      ; Recall the address to which the bytes were written
1861     AND.W #0x00FF,W2      ; Send an address command to the PIC18F2553
1862     MOV.W W2,U2TXREG
1863     REPEAT #190
1864     NOP
1865     MOV.W #dataout,W2      ; Recall the written bytes
1866     DO #5,pp8_end         ; Send six data byte commands to the PIC18F2553
1867     MOV.B [W2++],W1
1868     BSET.W W1,#8
1869     MOV.W W1,U2TXREG
1870     REPEAT #190
1871     NOP
1872 pp8_end:
1873     NOP
1874     MOV.W #progstat,W2     ; Send an address command to the PIC18F2553
1875     BSET.B [W2],#6        ; Set bit six of progstat to indicate read completion
1876     BCLR.B [W2],#5        ; Clear bit five of progstat to indicate not bootloader
1877     MOV.B [W2],W1
1878     AND.W #0x00FF,W2
1879     MOV.W W2,U2TXREG
1880     REPEAT #190
1881     NOP
1882     BSET.W W1,#8          ; Send the progstat data byte command to the PIC18F2553
1883     MOV.W W1,U2TXREG
1884     REPEAT #190
1885     NOP
1886     BRA pp_done
1887
1888 pp_not8:
1889     CP.B W0,#9            ; Is this a read 48 memory bytes operation request?
1890     BRA NZ,pp_not9
1891     MOV.W addressl,W1     ; Recall the low address of the flash/EEPROM to read
1892     MOV.W addressh,W2     ; Recall the high address of the flash/EEPROM to read
1893     MOV.W W2,TBLPAG
1894     MOV.W #dataout,W2
1895     DO #15,pp9_enda       ; Write 48 bytes of flash/EEPROM memory to the buffer
1896     TBLRD.L.B [W1++],[W2++]
1897     TBLRD.L.B [W1--],[W2++]
1898     TBLRD.H.B [W1],[W2++]
1899     INC2.W W1,W1
1900 pp9_enda:
1901     NOP
1902     MOV.W #dataout,W2     ; Recall the address to which the bytes were written
1903     AND.W #0x00FF,W2     ; Send an address command to the PIC18F2553
1904     MOV.W W2,U2TXREG
1905     REPEAT #190
1906     NOP
1907     MOV.W #dataout,W2     ; Recall the written bytes
1908     DO #47,pp9_endb       ; Send 48 data byte commands to the PIC18F2553
1909     MOV.B [W2++],W1
1910     BSET.W W1,#8
1911     MOV.W W1,U2TXREG
1912     REPEAT #190
1913     NOP
1914 pp9_endb:
1915     NOP
1916     MOV.W #progstat,W2    ; Send an address command to the PIC18F2553
1917     BSET.B [W2],#6        ; Set bit six of progstat to indicate read completion
1918     BCLR.B [W2],#5        ; Clear bit five of progstat to indicate not bootloader

```

```

1919     MOV.B [W2],W1
1920     AND.W #0x00FF,W2
1921     MOV.W W2,U2TXREG
1922     REPEAT #190
1923     NOP
1924     BSET.W W1,#8           ; Send the progstat data byte command to the PIC18F2553
1925     MOV.W W1,U2TXREG
1926     REPEAT #190
1927     NOP
1928     BRA pp_done
1929
1930 pp_not9:
1931     CP.B W0,#10           ; Is this a write RAM word operation request?
1932     BRA NZ,pp_not10
1933     MOV.W address1,W1     ; Recall the low address word
1934     MOV.W progdata,W2
1935     BCLR.W W1,#0          ; Prevent non-word-boundary writes
1936     MOV.W W2,[W1]         ; Transfer a word of data to the RAM location
1937     BRA pp_done          ; No acknowledgement is needed for this operation
1938
1939 pp_not10:
1940     CP.B W0,#11           ; Is this a read RAM long operation request?
1941     BRA NZ,pp_not11
1942     MOV.W address1,W1     ; Recall the address of the RAM to read
1943     MOV.W #dataout,W2    ; Write this to the output buffer
1944     BCLR.W W1,#0          ; Prevent non-word-boundary reads
1945     MOV.W [W1++],[W2++]   ; Transfer a long of data to the output buffer
1946     MOV.W [W1++],[W2--]
1947     AND.W #0x00FF,W2     ; Send an address command to the PIC18F2553
1948     MOV.W W2,U2TXREG
1949     REPEAT #190
1950     NOP
1951     MOV.W #dataout,W2    ; Recall the written bytes
1952     DO #3,pp11_end        ; Send four data byte commands to the PIC18F2553
1953     MOV.B [W2++],W1
1954     BSET.W W1,#8
1955     MOV.W W1,U2TXREG
1956     REPEAT #190
1957     NOP
1958 pp11_end:
1959     NOP
1960     MOV.W #progstat,W2    ; Send an address command to the PIC18F2553
1961     BSET.B [W2],#6        ; Set bit six of progstat to indicate read completion
1962     BCLR.B [W2],#5        ; Clear bit five of progstat to indicate not bootloader
1963     MOV.B [W2],W1
1964     AND.W #0x00FF,W2
1965     MOV.W W2,U2TXREG
1966     REPEAT #190
1967     NOP
1968     BSET.W W1,#8         ; Send the progstat data byte command to the PIC18F2553
1969     MOV.W W1,U2TXREG
1970     REPEAT #190
1971     NOP
1972     BRA pp_done
1973
1974 pp_not11:
1975     CP.B W0,#12           ; Is this a jump to other bootloader operation request?
1976     BRA NZ,pp_not12
1977     REPEAT #12798        ; Delay 400us so no more bytes come in from PIC18F2553
1978     NOP
1979     BSET.W INTCON2,#15    ; Switch to the alternate interrupt vector table
1980     BRA bootloader       ; The bootloader updates up to 0x00007BFE, then resets

```

```

1981
1982 pp_not12:
1983     CP.B W0,#13           ; Is this a check battery modules' communication request?
1984     BRA NZ,pp_not13
1985     REPEAT #12798         ; Delay 400us so no more bytes come in from PIC18F2553
1986     NOP
1987     MOV.W #0x0001,W0      ; Set the baud rate to 1MBd
1988     MOV.W W0,U1BRG
1989     MOV.W progdata,W1     ; Recall the desired module address to test this time
1990     RCALL check_comm      ; Send 256 different bytes to the module, check echoes
1991     SL.W W1,#8,W1         ; Form a return value for the computer (address, retval)
1992     ADD.W W1,W2,W1
1993     MOV.W W1,dataout      ; This value will in turn get transmitted back over USB
1994     MOV.W #dataout,W2    ; Recall the address to which the bytes were written
1995     AND.W #0x00FF,W2     ; Send an address command to the PIC18F2553
1996     MOV.W W2,U2TXREG
1997     REPEAT #190
1998     NOP
1999     MOV.W #dataout,W2    ; Recall the written bytes
2000     DO #1,pp13_end       ; Send two data byte commands to the PIC18F2553
2001     MOV.B [W2++],W1
2002     BSET.W W1,#8
2003     MOV.W W1,U2TXREG
2004     REPEAT #190
2005     NOP
2006 pp13_end:
2007     NOP
2008     MOV.W #progstat,W2   ; Send an address command to the PIC18F2553
2009     BSET.B [W2],#6       ; Set bit six of progstat to indicate read completion
2010     BCLR.B [W2],#5       ; Clear bit five of progstat to indicate not bootloader
2011     MOV.B [W2],W1
2012     AND.W #0x00FF,W2
2013     MOV.W W2,U2TXREG
2014     REPEAT #190
2015     NOP
2016     BSET.W W1,#8        ; Send the progstat data byte command to the PIC18F2553
2017     MOV.W W1,U2TXREG
2018     REPEAT #190
2019     NOP
2020     BRA pp_done
2021
2022 pp_not13:
2023     CP.B W0,#14           ; Is this a reprogram battery modules operation request?
2024     BRA NZ,pp_not14
2025     REPEAT #12798         ; Delay 400us so no more bytes come in from PIC18F2553
2026     NOP
2027
2028 ; Send $AC $53 $00 - enter the bootloader and address all modules globally
2029     MOV.W #0x000F,W3      ; Set the baud rate to 125kBd for the Atmel bootloader
2030     MOV.W W3,U1BRG
2031     MOV.W #0x01AC,W3
2032     MOV.W W3,U1TXREG
2033     REPEAT #3068         ; Delay about 3070 cycles
2034     NOP
2035     MOV.W #0x0153,W3
2036     MOV.W W3,U1TXREG
2037     REPEAT #3068         ; Delay about 3070 cycles
2038     NOP
2039     MOV.W #0x0100,W3
2040     MOV.W W3,U1TXREG
2041     REPEAT #3998         ; Delay about 4000 cycles
2042     NOP

```

```

2043
2044 CLR.W W0 ; Clear the module address counter
2045 MOV.W #proginfo,W4 ; Load the address of the Atmels-to-program list into W1
2046 MOV.W U1RXREG,W1 ; Clear out the receive buffer first, to prevent errors
2047 MOV.W U1RXREG,W1
2048 MOV.W U1RXREG,W1
2049 MOV.W U1RXREG,W1
2050 DO #14,atmelloop
2051 atmelnnext:
2052 SL.W W0,#1,W5 ; Create a doubled module address to find the bits
2053 INC.W W0,W0 ; Increment the counter early (addresses start at one)
2054 INC.W W5,W6
2055 BTST.C [W4],W5 ; Is the pointed module commanded to be programmed?
2056 BRA NC,atmelnoprog ; If not, keep looking for one that is
2057
2058 ; Send $00 $00 $81 - change the bootloader address to the global address (zero)
2059 MOV.W #0x0100,W1
2060 MOV.W W1,U1TXREG
2061 REPEAT #3068 ; Delay about 3070 cycles
2062 NOP
2063 MOV.W W1,U1TXREG
2064 REPEAT #3068 ; Delay about 3070 cycles
2065 NOP
2066 MOV.W #0x0181,W1
2067 MOV.W W1,U1TXREG
2068 REPEAT #3998 ; Delay about 4000 cycles
2069 NOP
2070
2071 ; Send $00 $00 $88 - send a dummy command to turn the red LEDs on
2072 MOV.W #0x0100,W1
2073 MOV.W W1,U1TXREG
2074 REPEAT #3068 ; Delay about 3070 cycles
2075 NOP
2076 MOV.W W1,U1TXREG
2077 REPEAT #3068 ; Delay about 3070 cycles
2078 NOP
2079 MOV.W #0x0188,W1
2080 MOV.W W1,U1TXREG
2081 REPEAT #3998 ; Delay about 4000 cycles
2082 NOP
2083
2084 ; Send XX XX $81 - change the bootloader address to this module's address
2085 MOV.W W0,W2
2086 BSET.W W2,#8
2087 MOV.W W2,U1TXREG
2088 REPEAT #3068 ; Delay about 3070 cycles
2089 NOP
2090 MOV.W W2,U1TXREG
2091 REPEAT #3068 ; Delay about 3070 cycles
2092 NOP
2093 MOV.W #0x0181,W3
2094 MOV.W W3,U1TXREG
2095 REPEAT #9998 ; Delay about 10000 cycles; wait for a reception
2096 NOP
2097 BTSS.W U1STA,#URXDA ; Is there at least one available word?
2098 BRA atmelfailcc ; If not, this is an enter bootloader failure
2099 MOV.W U1RXREG,W3 ; If so, check to see if it's $4B XOR address
2100 MOV.W #0x004B,W2
2101 XOR.W W0,W2,W2
2102 CP.B W2,W3
2103 BRA NZ,atmelfailcc ; If it wasn't this there was an enter bootloader failure
2104 REPEAT #998 ; Delay about 1000 cycles

```

```

2105     NOP
2106
2107     MOV.W #atmeldata,W7 ; Load the Atmel program data address into W7
2108     CLR.W W8             ; Clear the Atmel address pointer, W8
2109     atmelprog:
2110     ; Send YY ZZ $03 - erase flash page $ZZYY
2111     MOV.W W8,W3
2112     BSET.W W3,#8
2113     MOV.W W3,U1TXREG
2114     REPEAT #3068          ; Delay about 3070 cycles
2115     NOP
2116     MOV.W W8,W3
2117     MOV.W W8,W9
2118     SWAP.W W3
2119     BSET.W W3,#8
2120     MOV.W W3,U1TXREG
2121     REPEAT #3068          ; Delay about 3070 cycles
2122     NOP
2123     MOV.W #0x0103,W3
2124     MOV.W W3,U1TXREG
2125     CLR.W W3              ; Delay over 150,000 cycles (five milliseconds)
2126     INC.W W3,W3
2127     BRA NZ,.-2
2128
2129     ; Send $11 $11 $11 - clear the temporary page buffer for flash writing
2130     MOV.W #0x0111,W2
2131     MOV.W W2,U1TXREG
2132     REPEAT #3068          ; Delay about 3070 cycles
2133     NOP
2134     MOV.W W2,U1TXREG
2135     REPEAT #3068          ; Delay about 3070 cycles
2136     NOP
2137     MOV.W W2,U1TXREG
2138     MOV.W #-10000,W3      ; Delay about 30000 cycles
2139     INC.W W3,W3
2140     BRA NZ,.-2
2141
2142     atmelmore:
2143     ; Send YY ZZ $88 - load flash data into the Atmel's temporary flash buffer (#1)
2144     MOV.B [W7++],W3
2145     BSET.W W3,#8
2146     MOV.W W3,U1TXREG
2147     REPEAT #3068          ; Delay about 3070 cycles
2148     NOP
2149     MOV.B [W7++],W3
2150     BSET.W W3,#8
2151     MOV.W W3,U1TXREG
2152     REPEAT #3068          ; Delay about 3070 cycles
2153     NOP
2154     MOV.W #0x0188,W3
2155     MOV.W W3,U1TXREG
2156     MOV.W #-10000,W3      ; Delay about 30000 cycles
2157     INC.W W3,W3
2158     BRA NZ,.-2
2159
2160     ; Send YY ZZ $01 - load flash data into the Atmel's temporary flash buffer (#2)
2161     MOV.W W8,W3
2162     BSET.W W3,#8
2163     MOV.W W3,U1TXREG
2164     REPEAT #3068          ; Delay about 3070 cycles
2165     NOP
2166     MOV.W W8,W3

```

```

2167     INC2.W W8,W8
2168     SWAP.W W3
2169     BSET.W W3,#8
2170     MOV.W W3,U1TXREG
2171     REPEAT #3068             ; Delay about 3070 cycles
2172     NOP
2173     MOV.W #0x0101,W3
2174     MOV.W W3,U1TXREG
2175     MOV.W #-10000,W3        ; Delay about 30000 cycles
2176     INC.W W3,W3
2177     BRA NZ,.-2
2178
2179     AND.W W8,#0x001F,W3    ; Has the end of this page been reached?
2180     BRA NZ,atmelmore       ; If not, keep loading this page buffer
2181
2182 ; Send YY ZZ $05 - write flash page $ZZYY
2183     MOV.W W9,W3
2184     BSET.W W3,#8
2185     MOV.W W3,U1TXREG
2186     REPEAT #3068             ; Delay about 3070 cycles
2187     NOP
2188     MOV.W W9,W3
2189     SWAP.W W3
2190     BSET.W W3,#8
2191     MOV.W W3,U1TXREG
2192     REPEAT #3068             ; Delay about 3070 cycles
2193     NOP
2194     MOV.W #0x0105,W3
2195     MOV.W W3,U1TXREG
2196     CLR.W W3                ; Delay over 150,000 cycles (five milliseconds)
2197     INC.W W3,W3
2198     BRA NZ,.-2
2199
2200     MOV.W W9,W8             ; Rewind in memory and verify the written bytes
2201     MOV.W #atmeldata,W7     ; Load the Atmel program data address into W7
2202     ADD.W W7,W8,W7          ; Access the beginning of this flash page in DSPIC RAM
2203
2204 atmelverify:
2205 ; Send YY ZZ $84 - read flash location $ZZYY
2206     MOV.W W8,W3
2207     BSET.W W3,#8
2208     MOV.W W3,U1TXREG
2209     REPEAT #3068             ; Delay about 3070 cycles
2210     NOP
2211     MOV.W W8,W3
2212     INC.W W8,W8
2213     SWAP.W W3
2214     BSET.W W3,#8
2215     MOV.W W3,U1TXREG
2216     REPEAT #3068             ; Delay about 3070 cycles
2217     NOP
2218     MOV.W #0x0184,W3
2219     MOV.W W3,U1TXREG
2220     REPEAT #9998             ; Delay about 10000 cycles; wait for a reception
2221     NOP
2222     BTSS.W U1STA,#URXDA     ; Is there at least one available word?
2223     BRA atmelfailnoreply    ; If not, this is a communications error
2224     MOV.W U1RXREG,W3        ; If so, check to see if it's the same as the written
2225     CP.B W3,[W7++]
2226     BRA NZ,atmelfailbad     ; If it wasn't the same, this is a verification error
2227     REPEAT #998             ; Delay about 1000 cycles
2228     NOP

```



```

2229
2230 AND.W W8,#0x001F,W3 ; Has the end of this page been reached?
2231 BRA NZ,atmelverify ; If not, keep verifying the written bytes
2232
2233 ADD.W #0x0140,W9 ; Has the end of the program been reached (0x06C0 done)?
2234 BTSS.W W9,#11 ; The above addition would create an overflow to 0x0800
2235 BRA atmelprog ; This sequence programs bytes 0x0000-0x06DF
2236
2237 BCLR.W SR,#C
2238 BSW.C [W4],W5 ; A return value of '00' means success
2239 BSW.C [W4],W6
2240 BRA atmelnoprog
2241 atmelfailcc:
2242 BSET.W SR,#C
2243 BSW.C [W4],W5 ; A return value of '01' means entering bootloader failed
2244 BCLR.W SR,#C
2245 BSW.C [W4],W6
2246 BRA atmelnoprog
2247 atmelfailnoreply:
2248 BCLR.W SR,#C
2249 BSW.C [W4],W5 ; A return value of '10' means a no reply verify error
2250 BSET.W SR,#C
2251 BSW.C [W4],W6
2252 BRA atmelnoprog
2253 atmelfailbad:
2254 BSET.W SR,#C
2255 BSW.C [W4],W5 ; A return value of '11' means a bad byte verify error
2256 BSW.C [W4],W6
2257 atmelnoprog:
2258 MOV.B WREG,proglast ; Note the last processed module address
2259 AND.W W0,#0x0007,W1 ; Test the lower three bits to know if this word is done
2260 BRA NZ,atmelnext ; If the bits are nonzero, there are still more data left
2261 INC2.W W4,W4
2262 atmelloop:
2263 NOP
2264
2265 ; Send $00 $00 $81 - change the bootloader address to the global address (zero)
2266 MOV.W #0x0100,W1
2267 MOV.W W1,U1TXREG
2268 REPEAT #3068 ; Delay about 3070 cycles
2269 NOP
2270 MOV.W W1,U1TXREG
2271 REPEAT #3068 ; Delay about 3070 cycles
2272 NOP
2273 MOV.W #0x0181,W1
2274 MOV.W W1,U1TXREG
2275 REPEAT #3998 ; Delay about 4000 cycles
2276 NOP
2277
2278 ; Send $00 $00 $88 - send a dummy command to turn the last module's red LED on
2279 MOV.W #0x0100,W1
2280 MOV.W W1,U1TXREG
2281 REPEAT #3068 ; Delay about 3070 cycles
2282 NOP
2283 MOV.W W1,U1TXREG
2284 REPEAT #3068 ; Delay about 3070 cycles
2285 NOP
2286 MOV.W #0x0188,W1
2287 MOV.W W1,U1TXREG
2288 REPEAT #3998 ; Delay about 4000 cycles
2289 NOP
2290

```

```

2291 ; Send $00 $00 $00 - hang the modules and let the watchdogs reset them
2292 MOV.W #0x0100,W1
2293 MOV.W W1,U1TXREG
2294 REPEAT #3068 ; Delay about 3070 cycles
2295 NOP
2296 MOV.W W1,U1TXREG
2297 REPEAT #3068 ; Delay about 3070 cycles
2298 NOP
2299 MOV.W W1,U1TXREG
2300 REPEAT #3068 ; Delay about 3070 cycles
2301 NOP
2302
2303 ; Inform the computer upon completion
2304 MOV.W #0x0001,W1 ; Set the baud rate back to 1MBd
2305 MOV.W W1,U1BRG
2306 MOV.W #progrstat,W2 ; Send an address command to the PIC18F2553
2307 BSET.B [W2],#6 ; Set bit six of progrstat to indicate read completion
2308 BCLR.B [W2],#5 ; Clear bit five of progrstat to indicate not bootloader
2309 MOV.B [W2],W1
2310 AND.W #0x00FF,W2
2311 MOV.W W2,U2TXREG
2312 REPEAT #190
2313 NOP
2314 BSET.W W1,#8 ; Send the progrstat data byte command to the PIC18F2553
2315 MOV.W W1,U2TXREG
2316 REPEAT #190
2317 NOP
2318 BRA pp_done
2319
2320 pp_not14:
2321 CP.B W0,#15 ; Is this a send 1MBd data to battery modules request?
2322 BRA NZ,pp_not15
2323 MOV.W #0x0001,W0 ; Set the baud rate to 1MBd
2324 MOV.W W0,U1BRG
2325 REPEAT #12798 ; Delay 400us so no more bytes come in from PIC18F2553
2326 NOP
2327 BTSS.W atmeltx1,#15 ; Is the top bit of atmeltx1 set? If so, send out a word
2328 BRA .+6
2329 MOV.W atmeltx1,W0
2330 MOV.W W0,U1TXREG
2331 REPEAT #380
2332 NOP
2333 BTSS.W atmeltx2,#15 ; Is the top bit of atmeltx2 set? If so, send out a word
2334 BRA .+6
2335 MOV.W atmeltx2,W0
2336 MOV.W W0,U1TXREG
2337 REPEAT #380
2338 NOP
2339 BTSS.W atmeltx3,#15 ; Is the top bit of atmeltx3 set? If so, send out a word
2340 BRA .+6
2341 MOV.W atmeltx3,W0
2342 MOV.W W0,U1TXREG
2343 REPEAT #380
2344 NOP
2345 BTSS.W atmeltx4,#15 ; Is the top bit of atmeltx4 set? If so, send out a word
2346 BRA .+6
2347 MOV.W atmeltx4,W0
2348 MOV.W W0,U1TXREG
2349 REPEAT #380
2350 NOP
2351 CLR.W atmeltx1
2352 CLR.W atmeltx2

```

```

2353 CLR.W atmeltx3
2354 CLR.W atmeltx4
2355 BRA pp_done
2356
2357 pp_not15:
2358 CP.B W0,#16 ; Is this a send 125kBd data to battery modules request?
2359 BRA NZ,pp_not16
2360 MOV.W #0x000F,W0 ; Set the baud rate to 125kBd
2361 MOV.W W0,U1BRG
2362 REPEAT #12798 ; Delay 400us so no more bytes come in from PIC18F2553
2363 NOP
2364 BTSS.W atmeltx1,#15 ; Is the top bit of atmeltx1 set? If so, send out a word
2365 BRA .+6
2366 MOV.W atmeltx1,W0
2367 MOV.W W0,U1TXREG
2368 REPEAT #3068
2369 NOP
2370 BTSS.W atmeltx2,#15 ; Is the top bit of atmeltx2 set? If so, send out a word
2371 BRA .+6
2372 MOV.W atmeltx2,W0
2373 MOV.W W0,U1TXREG
2374 REPEAT #3068
2375 NOP
2376 BTSS.W atmeltx3,#15 ; Is the top bit of atmeltx3 set? If so, send out a word
2377 BRA .+6
2378 MOV.W atmeltx3,W0
2379 MOV.W W0,U1TXREG
2380 REPEAT #3068
2381 NOP
2382 BTSS.W atmeltx4,#15 ; Is the top bit of atmeltx4 set? If so, send out a word
2383 BRA .+6
2384 MOV.W atmeltx4,W0
2385 MOV.W W0,U1TXREG
2386 REPEAT #3068
2387 NOP
2388 CLR.W atmeltx1
2389 CLR.W atmeltx2
2390 CLR.W atmeltx3
2391 CLR.W atmeltx4
2392 BRA pp_done
2393
2394 pp_not16:
2395 CP.B W0,#17 ; Is this a check battery module reply operation request?
2396 BRA NZ,pp_not17
2397 BTSS.W U1STA,#URXDA ; Is there at least one available word?
2398 BRA pp_noreply ; If not, clear progstat and send it to the PIC18F2553
2399 MOV.W U1RXREG,W1 ; Recall this available word
2400 BSET.W W1,#15 ; Set bit seven of progstat to indicate a received word
2401 BCLR.W W1,#13 ; Clear bit five of progstat to indicate not bootloader
2402 MOV.W W1,atmelrx ; This value will in turn get transmitted back over USB
2403 MOV.W #atmelrx,W2 ; Recall the address to which the bytes were written
2404 AND.W #0x00FF,W2 ; Send an address command to the PIC18F2553
2405 MOV.W W2,U2TXREG
2406 REPEAT #190
2407 NOP
2408 MOV.W #atmelrx,W2 ; Recall the written bytes
2409 DO #1,pp17_end ; Send two data byte commands to the PIC18F2553
2410 MOV.B [W2++],W1 ; Note that one of these bytes is progstat
2411 BSET.W W1,#8
2412 MOV.W W1,U2TXREG
2413 REPEAT #190
2414 NOP

```

```

2415 pp17_end:
2416     NOP
2417     BRA pp_done
2418
2419 pp_not17:
2420     CP.B W0,#31          ; Is this a clear output buffer operation request?
2421     BRA NZ,pp_done
2422 pp_noreply:
2423     MOV.W #progstat,W2   ; Send an address command to the PIC18F2553
2424     CLR.B [W2]           ; Clear progstat to indicate no completion (UART or read)
2425     AND.W #0x00FF,W2
2426     MOV.W W2,U2TXREG
2427     REPEAT #190
2428     NOP
2429     MOV.W #0x0100,W1     ; Send the progstat data byte command to the PIC18F2553
2430     MOV.W W1,U2TXREG
2431     REPEAT #190
2432     NOP
2433     BRA pp_done
2434
2435 pp_prog:
2436     REPEAT #12798        ; Delay 400us so no more bytes come in from PIC18F2553
2437     NOP
2438     DISI #5              ; Execute a generic programming cycle for many operations
2439     MOV.W #0x55,W0
2440     MOV.W W0,NVMKEY
2441     MOV.W #0xAA,W1
2442     MOV.W W1,NVMKEY
2443     BSET.W NVMCON,#WR
2444     NOP
2445     NOP
2446     BTSC.W NVMCON,#WR    ; Wait until the operation completes (for EEPROM)
2447     BRA .-2
2448     MOV.W #progstat,W2   ; Send an address command to the PIC18F2553
2449     BSET.B [W2],#6        ; Set bit six of progstat to indicate read completion
2450     BCLR.B [W2],#5        ; Clear bit five of progstat to indicate not bootloader
2451     MOV.B [W2],W1
2452     AND.W #0x00FF,W2
2453     MOV.W W2,U2TXREG
2454     REPEAT #190
2455     NOP
2456     BSET.W W1,#8          ; Send the progstat data byte command to the PIC18F2553
2457     MOV.W W1,U2TXREG
2458     REPEAT #190
2459     NOP
2460
2461 pp_done:
2462     CLR.B progcmd        ; Clear the command so nothing happens accidentally
2463
2464 pp_none:
2465     RETURN
2466
2467 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2468 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2469
2470 ; Interrupt service routine
2471 .global __U2RXInterrupt
2472 __U2RXInterrupt:
2473     PUSH.S                ; Save the status register, W0, W1, W2, and W3
2474
2475     MOV.W U2RXREG,W0
2476     BTSS.W W0,#8          ; Is bit eight low? If so, this is a new writing address

```

```

2477     BRA .+6
2478     MOV.B W0,[W14++]      ; Write the received data to the local buffer at 0x0F00
2479     BRA .+6
2480     MOV.W #carstat,W14    ; Start the buffer address off at 0x0F00
2481     ADD.W W0,W14,W14      ; Save this byte as the new memory buffer writing address
2482
2483 U2RXInt_end:
2484     BCLR.W IFS1,#U2RXIF   ; Clear the U2RX interrupt flag in IFS1
2485     POP.S                 ; Restore the status register, W0, W1, W2, and W3
2486     RETFIE
2487
2488 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2489
2490 ; Sine table
2491
2492 sine_table:                ; 1024 entries, scaled/offset by 512, 0 to <360 degrees
2493     AND.W #0x03FF,W3      ; Disallow any data access past 1023 (round to ten bits)
2494     INC2.W W3,W3           ; Account for the "DEC2 W3,W3" and "RETURN" instructions
2495     RCALL W3
2496     DEC2.W W3,W3           ; Put W3 back to what it was before sine_table was called
2497     RETURN
2498     RETLW.W #0x0200,W6     ; sin(000.0) = +0.0000 (0512)
2499     RETLW.W #0x0203,W6     ; sin(000.4) = +0.0061 (0515)
2500     RETLW.W #0x0206,W6     ; sin(000.7) = +0.0123 (0518)
2501     RETLW.W #0x0209,W6     ; sin(001.1) = +0.0184 (0521)
2502     RETLW.W #0x020C,W6     ; sin(001.4) = +0.0245 (0524)
2503     RETLW.W #0x020F,W6     ; sin(001.8) = +0.0307 (0527)
2504     RETLW.W #0x0212,W6     ; sin(002.1) = +0.0368 (0530)
2505     RETLW.W #0x0215,W6     ; sin(002.5) = +0.0429 (0533)
2506     RETLW.W #0x0219,W6     ; sin(002.8) = +0.0491 (0537)
2507     RETLW.W #0x021C,W6     ; sin(003.2) = +0.0552 (0540)
2508     RETLW.W #0x021F,W6     ; sin(003.5) = +0.0613 (0543)
2509     RETLW.W #0x0222,W6     ; sin(003.9) = +0.0674 (0546)
2510     RETLW.W #0x0225,W6     ; sin(004.2) = +0.0736 (0549)
2511     RETLW.W #0x0228,W6     ; sin(004.6) = +0.0797 (0552)
2512     RETLW.W #0x022B,W6     ; sin(004.9) = +0.0858 (0555)
2513     RETLW.W #0x022E,W6     ; sin(005.3) = +0.0919 (0558)
2514
2515 (Only 16 of 1024 entries shown)
2516
2517 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2518
2519 ; Power factor table
2520
2521 pf_table:                  ; 1024 entries, 1000*cos(atan(ratio)), 0 to 1023/256
2522     AND.W #0x03FF,W0      ; Disallow any data access past 1023 (round to ten bits)
2523     INC2.W W0,W0           ; Account for the "DEC2 W0,W0" and "RETURN" instructions
2524     RCALL W0
2525     DEC2.W W0,W0           ; Put W0 back to what it was before pf_table was called
2526     RETURN
2527     RETLW.W #0x03E8,W1     ; 100*cos(atan(0.000)) = 100.00 (1000)
2528     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.004)) = 999.99 (0999)
2529     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.008)) = 999.99 (0999)
2530     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.012)) = 999.99 (0999)
2531     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.016)) = 999.98 (0999)
2532     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.020)) = 999.98 (0999)
2533     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.023)) = 999.97 (0999)
2534     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.027)) = 999.96 (0999)
2535     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.031)) = 999.95 (0999)
2536     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.035)) = 999.93 (0999)
2537     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.039)) = 999.92 (0999)
2538     RETLW.W #0x03E7,W1     ; 100*cos(atan(0.043)) = 999.90 (0999)

```

```

2539     RETLW.W #0x03E6,W1      ; 100*cos(atan(0.047)) = 099.89 (0998)
2540     RETLW.W #0x03E6,W1      ; 100*cos(atan(0.051)) = 099.87 (0998)
2541     RETLW.W #0x03E6,W1      ; 100*cos(atan(0.055)) = 099.85 (0998)
2542     RETLW.W #0x03E6,W1      ; 100*cos(atan(0.059)) = 099.82 (0998)
2543
2544     (Only 16 of 1024 entries shown)
2545
2546     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2547
2548     ; Square root table
2549
2550     sqrt_table:              ; 1024 entries, sqrt(value), 0 to 1023
2551     AND.W #0x03FF,W1        ; Disallow any data access past 1023 (round to ten bits)
2552     INC2.W W1,W1            ; Account for the "DEC2 W1,W1" and "RETURN" instructions
2553     RCALL W1
2554     RETURN
2555     RETLW.W #0x0000,W1      ; 32*sqrt(0000) = 0000.00 (0000)
2556     RETLW.W #0x0020,W1      ; 32*sqrt(0001) = 0032.00 (0032)
2557     RETLW.W #0x002D,W1      ; 32*sqrt(0002) = 0045.25 (0045)
2558     RETLW.W #0x0037,W1      ; 32*sqrt(0003) = 0055.43 (0055)
2559     RETLW.W #0x0040,W1      ; 32*sqrt(0004) = 0064.00 (0064)
2560     RETLW.W #0x0047,W1      ; 32*sqrt(0005) = 0071.55 (0071)
2561     RETLW.W #0x004E,W1      ; 32*sqrt(0006) = 0078.38 (0078)
2562     RETLW.W #0x0054,W1      ; 32*sqrt(0007) = 0084.66 (0084)
2563     RETLW.W #0x005A,W1      ; 32*sqrt(0008) = 0090.51 (0090)
2564     RETLW.W #0x0060,W1      ; 32*sqrt(0009) = 0096.00 (0096)
2565     RETLW.W #0x0065,W1      ; 32*sqrt(0010) = 0101.19 (0101)
2566     RETLW.W #0x006A,W1      ; 32*sqrt(0011) = 0106.13 (0106)
2567     RETLW.W #0x006E,W1      ; 32*sqrt(0012) = 0110.85 (0110)
2568     RETLW.W #0x0073,W1      ; 32*sqrt(0013) = 0115.38 (0115)
2569     RETLW.W #0x0077,W1      ; 32*sqrt(0014) = 0119.73 (0119)
2570     RETLW.W #0x007B,W1      ; 32*sqrt(0015) = 0123.94 (0123)
2571
2572     (Only 16 of 1024 entries shown)
2573
2574     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2575     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2576
2577     ; The bootloader location (a custom section at 0x007C00)
2578     .SECTION boot,code,address(0x007C00)
2579
2580     ; Bootloader's interrupt service routine
2581     .global __AltU2RXInterrupt
2582     __AltU2RXInterrupt:
2583     PUSH.S                  ; Save the status register, W0, W1, W2, and W3
2584     MOV.W U2RXREG,W0
2585     BTSS.W W0,#8            ; Is bit eight low? If so, this is a new writing address
2586     BRA .+6
2587     MOV.B W0,[W14++]        ; Write the received data to the local buffer at 0x0F00
2588     BRA .+6
2589     MOV.W #carstat,W14      ; Start the buffer address off at 0x0F00
2590     ADD.W W0,W14,W14        ; Save this byte as the new memory buffer writing address
2591     BCLR.W IFS1,#U2RXIF     ; Clear the U2RX interrupt flag in IFS1
2592     POP.S                   ; Restore the status register, W0, W1, W2, and W3
2593     RETFIE
2594
2595     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2596
2597     bootloader:             ; The bootloader for updating 0x00000000-0x00007BFE flash
2598     CLR.B progcmd           ; Clear the command so nothing happens accidentally
2599     MOV.W #progstat,W2      ; Send an address command to the PIC18F2553
2600     BSET.B [W2],#6          ; Set bit six of progstat to indicate read completion

```

```

2601     BSET.B [W2],#5           ; Set bit five of progstat to indicate bootloader
2602     MOV.B [W2],W1
2603     AND.W #0x00FF,W2
2604     MOV.W W2,U2TXREG
2605     REPEAT #190
2606     NOP
2607     BSET.W W1,#8             ; Send the progstat data byte command to the PIC18F2553
2608     MOV.W W1,U2TXREG
2609     REPEAT #190
2610     NOP
2611
2612 bp_loop:
2613     MOV.B progcmd,WREG       ; Test the DSPIC programming command
2614     CPO.B W0                 ; Is there anything here?
2615     BRA Z,bp_loop
2616
2617     CP.B W0,#1               ; Is this an erase program memory row operation request?
2618     BRA NZ,bp_not1
2619     MOV.W addressl,W0        ; Recall the low address word and store in NVMADR
2620     MOV.W W0,NVMADR
2621     MOV.W addressh,W0        ; Recall the high address word and store in NVMADRU
2622     MOV.W W0,NVMADRU
2623     MOV.W #0x4041,W0         ; Set up NVMCON for erasing a program memory row
2624     MOV.W W0,NVMCON
2625     BRA bp_prog
2626
2627 bp_not1:
2628     CP.B W0,#2               ; Is this a write program memory row operation request?
2629     BRA NZ,bp_not2
2630     MOV.W addressl,W2        ; Recall the low address word and store in NVMADR
2631     MOV.W W2,NVMADR
2632     MOV.W addressh,W0        ; Recall the high address word and store in NVMADRU
2633     MOV.W W0,NVMADRU
2634     MOV.W W0,TBLPAG          ; Also store it in TBLPAG for the write latches
2635     MOV.W #flashbuf,W1       ; Write 96 bytes of flash memory to the write latches
2636     DO #31,bp2_end           ; Loop 16 times for a total of 32 instruction words
2637     TBLWTL.B [W1++],[W2++]
2638     TBLWTL.B [W1++],[W2--]
2639     TBLWTH.B [W1++],[W2]
2640     INC2.W W2,W2
2641 bp2_end:
2642     NOP
2643     MOV.W #0x4001,W0         ; Set up NVMCON for writing a program memory row
2644     MOV.W W0,NVMCON
2645     BRA bp_prog
2646
2647 bp_not2:
2648     CP.B W0,#3               ; Is this an erase data memory word operation request?
2649     BRA NZ,bp_not3
2650     MOV.W addressl,W0        ; Recall the low address word and store in NVMADR
2651     MOV.W W0,NVMADR
2652     MOV.W addressh,W0        ; Recall the high address word and store in NVMADRU
2653     MOV.W W0,NVMADRU
2654     MOV.W #0x4044,W0         ; Set up NVMCON for erasing a data memory word
2655     MOV.W W0,NVMCON
2656     BRA bp_prog
2657
2658 bp_not3:
2659     CP.B W0,#4               ; Is this an erase data memory row operation request?
2660     BRA NZ,bp_not4
2661     MOV.W addressl,W0        ; Recall the low address word and store in NVMADR
2662     MOV.W W0,NVMADR

```

```

2663     MOV.W addressh,W0      ; Recall the high address word and store in NVMADRU
2664     MOV.W W0,NVMADRU
2665     MOV.W #0x4045,W0      ; Set up NVMCON for erasing a data memory row
2666     MOV.W W0,NVMCON
2667     BRA bp_prog
2668
2669     bp_not4:
2670     CP.B W0,#5             ; Is this a write data memory word operation request?
2671     BRA NZ,bp_not5
2672     MOV.W addressl,W0      ; Recall the low address word
2673     MOV.W addressh,W1      ; Recall the high address word and store in TBLPAG
2674     MOV.W W1,TBLPAG
2675     MOV.W #progdata,W2     ; Write the data word to the temporary buffer
2676     TBLWTL.W [W2],[W0]
2677     MOV.W #0x4004,W0      ; Set up NVMCON for writing a data memory word
2678     MOV.W W0,NVMCON
2679     BRA bp_prog
2680
2681     bp_not5:
2682     CP.B W0,#6             ; Is this a write data memory row operation request?
2683     BRA NZ,bp_not6
2684     MOV.W addressl,W2      ; Recall the low address word and store in NVMADR
2685     MOV.W W2,NVMADR
2686     MOV.W addressh,W0      ; Recall the high address word and store in NVMADRU
2687     MOV.W W0,NVMADRU
2688     MOV.W W0,TBLPAG        ; Also store it in TBLPAG for the write latches
2689     MOV.W #flashbuf,W1     ; Write 96 bytes of flash memory to the write latches
2690     REPEAT #15             ; Loop 16 times for a total of 16 data EEPROM words
2691     TBLWTL.W [W1++],[W2++]
2692     MOV.W #0x400A,W0      ; Set up NVMCON for writing a data memory row
2693     MOV.W W0,NVMCON
2694     BRA bp_prog
2695
2696     bp_not6:
2697     CP.B W0,#7             ; Is this a write config. memory operation request?
2698     BRA NZ,bp_not7
2699     MOV.W addressl,W0      ; Recall the low address word
2700     MOV.W addressh,W1      ; Recall the high address word and store in TBLPAG
2701     MOV.W W1,TBLPAG
2702     MOV.W #progdata,W2     ; Write the data word to the temporary buffer
2703     TBLWTL.W [W2],[W0]
2704     MOV.W #0x4008,W0      ; Set up NVMCON for writing a configuration memory word
2705     MOV.W W0,NVMCON
2706     BRA bp_prog
2707
2708     bp_not7:
2709     CP.B W0,#8             ; Is this a read memory long operation request?
2710     BRA NZ,bp_not8
2711     MOV.W addressl,W1      ; Recall the low address of the flash/EEPROM to read
2712     MOV.W addressh,W2      ; Recall the high address of the flash/EEPROM to read
2713     MOV.W W2,TBLPAG
2714     MOV.W #dataout,W2      ; Write six bytes of flash/EEPROM memory to the buffer
2715     TBLRDL.B [W1++],[W2++]
2716     TBLRDL.B [W1--],[W2++]
2717     TBLRDH.B [W1],[W2++]
2718     INC2.W W1,W1
2719     TBLRDL.B [W1++],[W2++]
2720     TBLRDL.B [W1--],[W2++]
2721     TBLRDH.B [W1],[W2++]
2722     MOV.W #dataout,W2      ; Recall the address to which the bytes were written
2723     AND.W #0x00FF,W2      ; Send an address command to the PIC18F2553
2724     MOV.W W2,U2TXREG

```



```

2725     REPEAT #190
2726     NOP
2727     MOV.W #dataout,W2      ; Recall the written bytes
2728     DO #5,bp8_end         ; Send six data byte commands to the PIC18F2553
2729     MOV.B [W2++],W1
2730     BSET.W W1,#8
2731     MOV.W W1,U2TXREG
2732     REPEAT #190
2733     NOP
2734 bp8_end:
2735     NOP
2736     MOV.W #progstat,W2     ; Send an address command to the PIC18F2553
2737     BSET.B [W2],#6        ; Set bit six of progstat to indicate read completion
2738     BSET.B [W2],#5        ; Set bit five of progstat to indicate bootloader
2739     MOV.B [W2],W1
2740     AND.W #0x00FF,W2
2741     MOV.W W2,U2TXREG
2742     REPEAT #190
2743     NOP
2744     BSET.W W1,#8          ; Send the progstat data byte command to the PIC18F2553
2745     MOV.W W1,U2TXREG
2746     REPEAT #190
2747     NOP
2748     BRA bp_done
2749
2750 bp_not8:
2751     CP.B W0,#9            ; Is this a read 48 memory bytes operation request?
2752     BRA NZ,bp_not9
2753     MOV.W addressl,W1     ; Recall the low address of the flash/EEPROM to read
2754     MOV.W addressh,W2     ; Recall the high address of the flash/EEPROM to read
2755     MOV.W W2,TBLPAG
2756     MOV.W #dataout,W2
2757     DO #15,bp9_enda      ; Write 48 bytes of flash/EEPROM memory to the buffer
2758     TBLRD.L B [W1++],[W2++]
2759     TBLRD.L B [W1--],[W2++]
2760     TBLRD.H B [W1],[W2++]
2761     INC2.W W1,W1
2762 bp9_enda:
2763     NOP
2764     MOV.W #dataout,W2     ; Recall the address to which the bytes were written
2765     AND.W #0x00FF,W2     ; Send an address command to the PIC18F2553
2766     MOV.W W2,U2TXREG
2767     REPEAT #190
2768     NOP
2769     MOV.W #dataout,W2     ; Recall the written bytes
2770     DO #47,bp9_endb      ; Send 48 data byte commands to the PIC18F2553
2771     MOV.B [W2++],W1
2772     BSET.W W1,#8
2773     MOV.W W1,U2TXREG
2774     REPEAT #190
2775     NOP
2776 bp9_endb:
2777     NOP
2778     MOV.W #progstat,W2     ; Send an address command to the PIC18F2553
2779     BSET.B [W2],#6        ; Set bit six of progstat to indicate read completion
2780     BSET.B [W2],#5        ; Set bit five of progstat to indicate bootloader
2781     MOV.B [W2],W1
2782     AND.W #0x00FF,W2
2783     MOV.W W2,U2TXREG
2784     REPEAT #190
2785     NOP
2786     BSET.W W1,#8          ; Send the progstat data byte command to the PIC18F2553

```

```

2787     MOV.W W1,U2TXREG
2788     REPEAT #190
2789     NOP
2790     BRA bp_done
2791
2792 bp_not9:
2793     CP.B W0,#10             ; Is this a write RAM word operation request?
2794     BRA NZ,bp_not10
2795     MOV.W address1,W1       ; Recall the low address word
2796     MOV.W progdata,W2
2797     BCLR.W W1,#0            ; Prevent non-word-boundary writes
2798     MOV.W W2,[W1]          ; Transfer a word of data to the RAM location
2799     BRA bp_done            ; No acknowledgement is needed for this operation
2800
2801 bp_not10:
2802     CP.B W0,#11             ; Is this a read RAM long operation request?
2803     BRA NZ,bp_not11
2804     MOV.W address1,W1       ; Recall the address of the RAM to read
2805     MOV.W #dataout,W2       ; Write this to the output buffer
2806     BCLR.W W1,#0            ; Prevent non-word-boundary reads
2807     MOV.W [W1++],[W2++]     ; Transfer a long of data to the output buffer
2808     MOV.W [W1++],[W2--]
2809     AND.W #0x00FF,W2       ; Send an address command to the PIC18F2553
2810     MOV.W W2,U2TXREG
2811     REPEAT #190
2812     NOP
2813     MOV.W #dataout,W2       ; Recall the written bytes
2814     DO #3,bp11_end         ; Send four data byte commands to the PIC18F2553
2815     MOV.B [W2++],W1
2816     BSET.W W1,#8
2817     MOV.W W1,U2TXREG
2818     REPEAT #190
2819     NOP
2820 bp11_end:
2821     NOP
2822     MOV.W #progstat,W2      ; Send an address command to the PIC18F2553
2823     BSET.B [W2],#6          ; Set bit six of progstat to indicate read completion
2824     BSET.B [W2],#5          ; Set bit five of progstat to indicate bootloader
2825     MOV.B [W2],W1
2826     AND.W #0x00FF,W2
2827     MOV.W W2,U2TXREG
2828     REPEAT #190
2829     NOP
2830     BSET.W W1,#8            ; Send the progstat data byte command to the PIC18F2553
2831     MOV.W W1,U2TXREG
2832     REPEAT #190
2833     NOP
2834     BRA bp_done
2835
2836 bp_not11:
2837     CP.B W0,#12             ; Is this a reset processor operation request?
2838     BRA NZ,bp_not12
2839     REPEAT #12798           ; Delay 400us so no more bytes come in from PIC18F2553
2840     NOP
2841     RESET
2842
2843 bp_not12:
2844     CP.B W0,#31             ; Is this a clear output buffer operation request?
2845     BRA NZ,bp_done
2846     MOV.W #progstat,W2      ; Send an address command to the PIC18F2553
2847     CLR.B [W2]              ; Clear progstat to indicate no completion (UART or read)
2848     BSET.B [W2],#5          ; Set bit five of progstat to indicate bootloader

```

```

2849     AND.W #0x00FF,W2
2850     MOV.W W2,U2TXREG
2851     REPEAT #190
2852     NOP
2853     MOV.W #0x0120,W1      ; Send the progstata data byte command to the PIC18F2553
2854     MOV.W W1,U2TXREG
2855     REPEAT #190
2856     NOP
2857     BRA bp_done
2858
2859 bp_prog:
2860     REPEAT #12798         ; Delay 400us so no more bytes come in from PIC18F2553
2861     NOP
2862     DISI #5               ; Execute a generic programming cycle for many operations
2863     MOV.W #0x55,W0
2864     MOV.W W0,NVMKEY
2865     MOV.W #0xAA,W1
2866     MOV.W W1,NVMKEY
2867     BSET.W NVMCON,#WR
2868     NOP
2869     NOP
2870     BTSC.W NVMCON,#WR     ; Wait until the operation completes (for EEPROM)
2871     BRA .-2
2872     MOV.W #progstata,W2   ; Send an address command to the PIC18F2553
2873     BSET.B [W2],#6        ; Set bit six of progstata to indicate read completion
2874     BSET.B [W2],#5        ; Set bit five of progstata to indicate bootloader
2875     MOV.B [W2],W1
2876     AND.W #0x00FF,W2
2877     MOV.W W2,U2TXREG
2878     REPEAT #190
2879     NOP
2880     BSET.W W1,#8          ; Send the progstata data byte command to the PIC18F2553
2881     MOV.W W1,U2TXREG
2882     REPEAT #190
2883     NOP
2884
2885 bp_done:
2886     CLR.B progcmd         ; Clear the command so nothing happens accidentally
2887     BRA bp_loop
2888
2889 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2890 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2891
2892 ; Configuration fuses
2893
2894 .SECTION __FOSC.sec,psv
2895 .WORD 0xC30F
2896 .SECTION __FWDT.sec,psv
2897 .WORD 0x003F
2898 .SECTION __FBORPOR.sec,psv
2899 .WORD 0x07B3
2900 .SECTION __FGS.sec,psv
2901 .WORD 0x0007
2902
2903 .END

```

BIBLIOGRAPHY

BIBLIOGRAPHY

- [1] Metric Mind Engineering, "EV power inverters for street vehicle or boat conversion," Online, Revised 18 August 2007.
- [2] AC Propulsion, Inc., "AC-150 Gen-2 EV Power System: Integrated Drive and Charging for Electric Vehicles," Online, Accessed March 2008.
- [3] Kodama, Dave and Jean Kodama, "EV₁ Chronicles: EV₁ Chargers," Online, Accessed March 2008.
- [4] Baldor Electric Company, "Product Overview: EJMM4106T," Online, Accessed March 2008.
- [5] AC Propulsion, Inc., "tZero Electric Sports Car Presentation," Online, Accessed March 2008.
- [6] Corzine, Keith A., "Operation and Design of Multilevel Inverters," University of Missouri–Rolla, June 2005.
- [7] Microchip Technology, Inc., "dsPIC30F4011/4012 Data Sheet: High-Performance, 16-Bit Digital Signal Controllers," DS70135E, Revised January 2007.
- [8] Microchip Technology, Inc., "PIC18F87J50 Family Data Sheet: 64/80-Pin High-Performance, 1-Mbit Flash USB Microcontrollers with nanoWatt Technology," DS39775B, Revised May 2007.
- [9] Baldor Electric Company, "Product Overview: EM3314T," Online, Accessed March 2008.
- [10] J. J. A. Wilkinson and G. A. Covic, "A New Pulse Charging Methodology for Lead Acid Batteries," IPENZ Transactions, Vol. 25, No. 1, 5 March 1998.
- [11] Okazaki S., S. Higuchi, O. Nakamura, and S. Takahashi, "Influence of Superimposed Alternating Current on Capacity and Cycle Life for Lead-acid Batteries," Journal of Applied Electrochemistry, Vol. 16, No. 6, November 1986, pp. 894-898.
- [12] Harris L. B. and J. P. D. Martin, "The Influence of Pulsed Discharge on the Capacity of Lead/Acid Batteries," Journal of Power Sources, Vol. 12, No. 1, 1984, pp. 71-80.
- [13] Ross, Dave, John Theys, and Steve Bowling, "Using the dsPIC30F for Vector Control of an ACIM," Microchip, Inc., AN908, 2004.

- [14] International Rectifier Corp., “IRS2004(S)PbF: Half-Bridge Driver,” PD60270, 27 November 2006.
- [15] Infineon Technologies AG, “BAS70.../BAS170W: Silicon Schottky Diode,” 19 September 2007.
- [16] Atmel Corporation, “8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable Flash: ATtiny24/44/84 Preliminary,” DOC8006G, Revised January 2008.
- [17] Texas Instruments Inc., “LP2950, LP2951: Adjustable Micropower Voltage Regulators with Shutdown,” SLVS582D, Revised March 2007.
- [18] Microchip Technology, Inc., “PIC18F2458/2553/4458/4553 Data Sheet: 28/40/44-Pin High-Performance, Enhanced Flash, USB Microcontrollers with 12-Bit A/D and nanoWatt Technology,” DS39887B, Revised June 2007.
- [19] Axelson, Jan, “USB Complete: Third Edition,” Lakeview Research LLC, 2005.
- [20] Matteson, Arthur. “Arthur Matteson / Custom Auto Electronics Homepage,” <http://home.comcast.net/~awmatt>.