

Remote Observation Station

Entry #F2040

Introduction

My wife Sherry's parents live on a hobby farm in northern Wisconsin near the town of Ashland, which sits on the shore of Lake Superior. During our visits to their home I often hear tales of the wildlife that is to be seen living on their mostly wooded three hundred acre farm. Stories of black bears with cubs, endangered timber wolves, and rare albino white tail deer are told often. Yet, try as I might, I never see any of them for myself, or only at extreme distances. Indeed, if they are in the vicinity, just walking into the woods to get a better look is enough to scare them away!

After hearing these stories enough times and never seeing much wildlife for myself, I hit upon a technological solution to my problem that would either provide us with great views of the indigenous wildlife, or prove that the stories were of the fish variety. My solution is the Remote Observation Station. I wanted a relatively inexpensive, easy to use, and easily maintainable piece of equipment that we could move to various locations without relying on utility power and one that would not require frequent trips to retrieve a battery for recharging. I also wanted to be able to simply turn on the TV to watch the alleged wildlife without any other special receiving equipment, except perhaps a good antenna.

Background

The first system I developed was a simple one. It consisted of an inexpensive color video camera module, a 433Mhz ATV transmitter, a high-gain antenna, a 12 volt VRLA battery, and a 21 Watt photovoltaic (PV) solar panel. A very simple control board provided power control (assembled on proto-board in few hours), consisting of two linear regulators, a MOSFET, and a relay. The system was installed in the middle of winter and initially it worked well. Unfortunately when spring came, the system became less and less reliable. First, the battery's charge was no longer being maintained by the solar panel due to the partial shading from the new tree canopy. Then the mechanical relay failed several times, possibly due to increased humidity. A linear regulator failed several times, which may explain why the RF transmitter it powered also failed. Although these power board failures were irritating, they were not completely unexpected. I knew the design was weak because I was using linear voltage regulators and no charge controller of any kind. But the problem that bothered me the most was not having any idea as to the state of charge of the battery. If the system stopped working I didn't know if it was due to a low battery or a component failure. In fact there were many questions I did not have answers for relating to system performance. Since this was a solar powered system that was located in a woodland in northern Wisconsin, a simple solar sizing calculation to determine available power at a particular site would probably give erroneous results. Trial and error often turns out to be the only practical way to find a setup that can be continuously maintained by solar power. I wanted a simple way to keep a close eye on the system's performance, without resorting to data loggers or similar equipment.

Just around the time I was contemplating an improved design, I learned of the latest Motorola design contest, sponsored by Circuit Cellar, based on their recent entry into the 8 and 16 pin MCU market. These Nitron family devices looked like they had everything my project needed. A perfect fit for my project and my first Circuit Cellar contest to boot! Off to the computer...

Design Objectives

My overall objectives for the Remote Observation Station included:

- High Portability – A small form factor, solar-powered, system that can operate without access to utility power.
- Increased Reliability – Power regulation components that can handle the wide voltage swings possible in a solar powered system. Charge control to prevent overcharging the battery.
- Battery Status Reporting – A way to easily see if the system's battery is being recharged, without a trip to the remote station.
- Low Cost – A single board solution based on an inexpensive microcontroller. Configurable via a PC to minimize user interface costs.

The heart of the Remote Observation Station is the System Control Board, which I call the Photovoltaic Charge Controller, or PVCC. The PVCC's primary purpose in this system is to prevent the battery from being overcharged. Feedback about the battery's state is designed to spot conditions that would leave the battery undercharged. The easiest way to instantly know the state of charge of a battery is to measure the battery's voltage. Because the station could be miles away from the user, and because it was already transmitting a standard NTSC video signal, it seemed natural to simply overlay the performance information on the video signal in the form of characters to be displayed on the bottom of the TV screen (an on-screen display, or OSD).

The major design objectives for the PVCC were as follows:

- Create a design general enough that it could be used as the control board for a variety of remote solar observation and data collection projects.
- Integrate a simple photovoltaic (PV) based charge controller (PVCC) that supports up to 10 Amps of current from the PV panel, and supports charging a variety of battery types.
- Monitor battery voltage and temperature for charge control and user feedback of current battery state. Provide feedback in easy non-technical way via On Screen Display (OSD) overlay.
- Provide a simple Windows configuration program that can be used to configure the control board MCU behavior. Retain configuration data through power cycles.
- Provide 5 volt logic power and auxiliary device power for cameras, transmitter, data loggers, and so forth, using high efficiency DC/DC converters which can handle high input voltages that may occur in PV systems. Avoid the use of inefficient linear regulators.
- Provide in-circuit programming capabilities for in-field firmware upgrades¹.
- Provide programming and debugging interface for use during development. A production version could eliminate this feature to reduce cost.
- Provide a low power mode using a slower MCU clock, and inhibiting the OSD feature.

This system can have many uses beyond observing wildlife and may be located in a myriad of locations. I wanted to provide the user with enough information to spot trends in the system's performance. I was not looking to generate detailed logs files, but simply a way to keep my eye on system's battery performance. If the solar panel was suddenly being shaded too often as the seasons changed, it would be nice to have some idea that performance was being negatively affected before the system just quit working. My control board design tries to address the issue of poor reliability in the original design and provide a simple way to monitor system performance. Overall, the goal is to add just enough technical sophistication to increase reliability and flexibility while keeping product costs low enough for this type of product to appeal to others trying to set up remote observation/data collection stations.

To meet the first two design objectives listed above, I chose a microcontroller-based design. A microcontroller allows the design to accommodate far more functionality without adding the complexity of the user interface (e.g. switches, buttons, and pots), which would tend to drive up costs. Flexibility is also greatly enhanced. Features can often be added with no change to the product's hardware design. For example, currently the charge controller uses a simple On/Off charge algorithm. In a future revision of the firmware, a PWM charging algorithm can be added, which will help maximize the battery's service life.

¹ A bootloader to support this feature has not been written yet, however the hardware is in place to support this feature.

System Overview

The Remote Observation Station integrates six electronic devices including: a CCD video camera, a PV solar panel, a rechargeable battery, a temperature sensor, an RF video transmitter, and the system control board (PVCC). The control board is based on Motorola's MC68HC908QY4 MCU and sits at the center of the system providing a PV charge controller, two high efficiency power regulators, a video sync separator, and an RS232 serial interface. Figure 1 shows a block diagram of the overall system. Photograph 1 shows the heart of the Remote Observation Station, enclosed in a water-tight aluminum box.

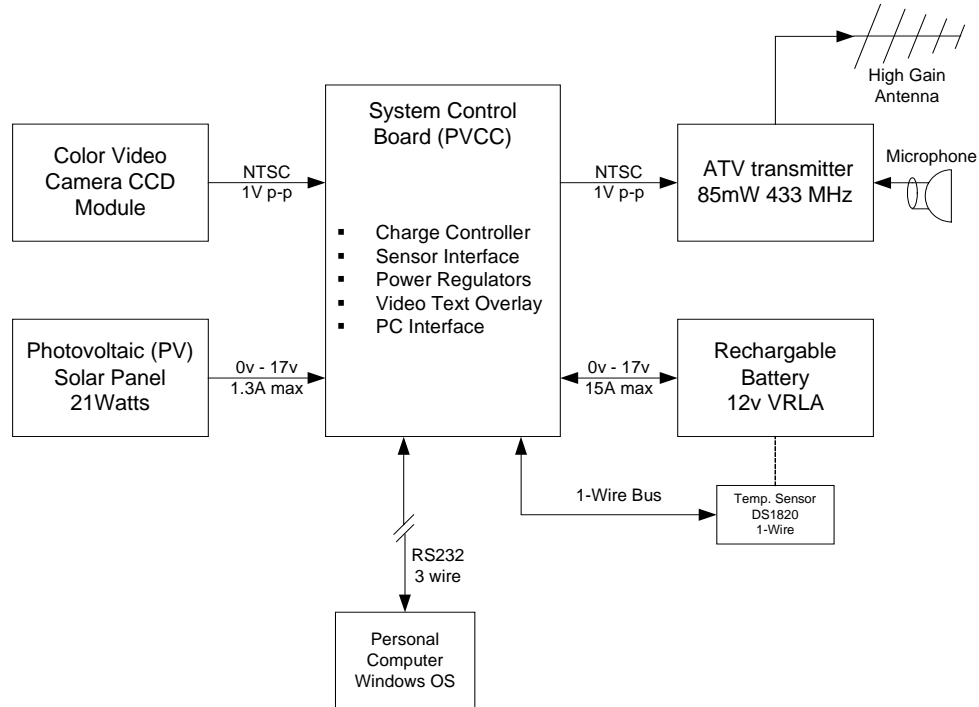
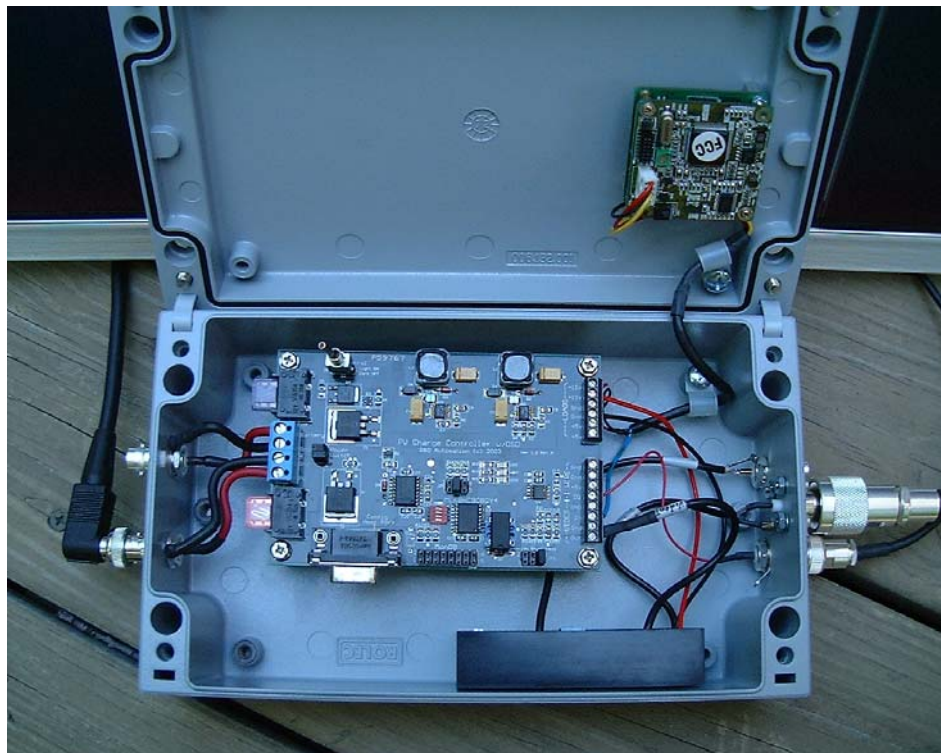


Figure 1: Remote Observation Station Block Diagram



Photograph 1: The Remote Observation Station (Not Pictured: Battery and Temp Sensor)

System Control Board

The PVCC control board provides a simple On/Off battery charger using the PV solar panel as a power source. In this system the total load is small (< 0.5 Amps). Therefore the primary job of the charge controller is to prevent the battery from being overcharged. The control board also adds battery state information in the form of a text overlay to the video signal from the CCD camera before it reaches the video transmitter. Additionally, a PC can be connected to the control board via DB9 connector J1.

The PVCC control board has three operating modes: the Configuration Mode, which allows the user to set up its behavior; the OSD Inactive Mode, which uses the charge controller but not the OSD; and the OSD Active Mode, where a text overlay is added to the video signal for feedback to the user. The configuration mode is selected via jumper JP5. Switch S2 controls when the unit operates; during daylight hours only, or all of the time.

Video Transmitter

The video transmitter I selected for this project operates in the 433MHz Amateur Television (ATV) band. It is model Z70A made by VideoLynx and has a rated output of 50 to 100mW while using 300mA at 10 volts. I chose this transmitter because it provides a very clean and stable signal and operates on a frequency that corresponds to a specific channel on a cable-ready TV set. Actually one of 4 different channels can be selected. This eliminates the need for special receiving equipment. To span distances in the 1 to 2 mile range requires a high gain antenna. I chose to build an antenna designed by N6NB, a 15 element Quagi for the 70cm band. This antenna provides a gain of approximately 14 to 15 dBd.

The use of an ATV transmitter requires an Amateur Radio License; which is covered under Part 97 in the FCC rules. Other possible choices for video transmitters that don't require special licensing are those covered under Part 15 in the FCC rules. These include video transmitters in the 2.4GHz band. These units are common and inexpensive today, but would probably require an upgrade to their antenna system to extend operating range. Another unlicensed option might be to use 900MHz spread spectrum equipment operating in the ISM band. In general, any transmitter can be used with this system as long as it requires operating current less than 1 Amp, and operating voltage in the 8 to 11 volt range.

PV Solar Panel

The solar panel I am using for this system is a UniSolar 21 watt unit. Its primary function is to recharge the system's battery. It has a rated output of 1.27A at 17 volts. Because my controller basically connects the battery directly to the PV panel when charging, the voltage output of the panel will be pulled down by the battery, which will cause the peak power of the panel to fall to about 17 watts.

The PV panel is also used as a light sensor for the Day On/Night Off load switch circuit on the PVCC board. Main battery power is supplied to the two switching regulators via MOSFET Q4. This transistor is controlled manually by switch S2 or automatically by the PV panel via Q3. Switch S2 acts as a bypass switch of Q3 allowing the PVCC to operate day and night, or when a PV panel is not present in the system.

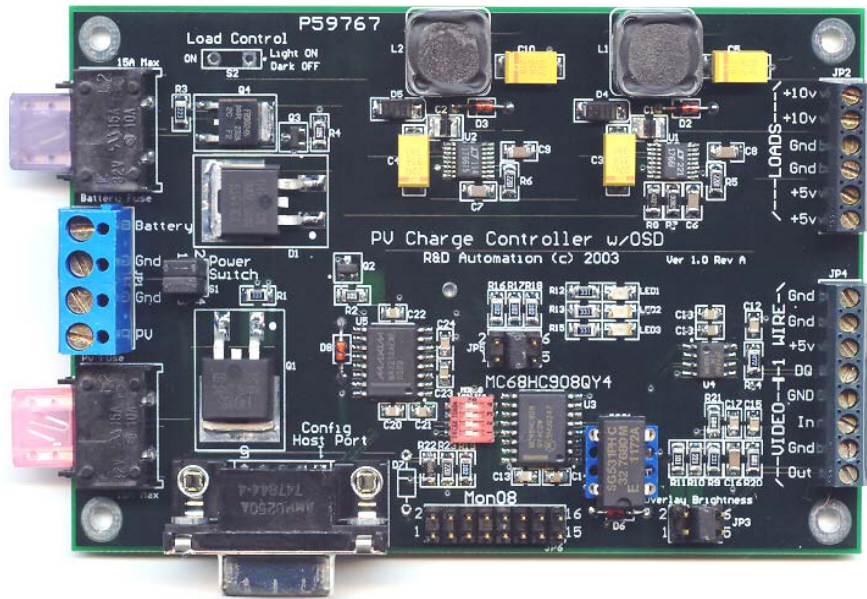
Rechargeable Battery

I use a 12 volt, Valve Regulated Lead Acid (VRLA) battery to power the system. It is monitored by the MCU to determine the charge state using the MCU's integrated 8-bit analog/digital converter. The battery's temperature is also monitored using a Dallas DS1820 1-Wire temperature sensor, which is interfaced to the MCU through a single I/O pin and some software bit-banging to instruct the MCU to behave like a 1-Wire master. A starting point for the 1-Wire master code was found in Motorola Application Note AN1292 and Dallas Semiconductor Application Note 162.

The charge control algorithm implements simple On/Off control. Because even relatively small PV panels can overcharge a battery under sunny conditions, the primary job of the charge controller is to prevent overcharging the battery. Overcharging a lead-acid battery will tend to cause gassing, loss of electrolyte, and possibly even damage to the battery's plates. Even if there is no serious damage, the battery working life can be greatly reduced. Additional battery life can be obtained by using a PWM charging algorithm. In the next major revision of firmware, I plan to implement a PWM charging algorithm..

PVCC Control Board

The first prototype of the PVCC board is shown in photograph 2. Figure 2 depicts the functional block diagram. The electrical schematics for this board are shown at the end of this section in figures 3a and 3b. The board has a 4500mil by 3000mil form factor, and was made on FR4 1oz double-sided copper PCB.



Photograph 2: PVCC control board

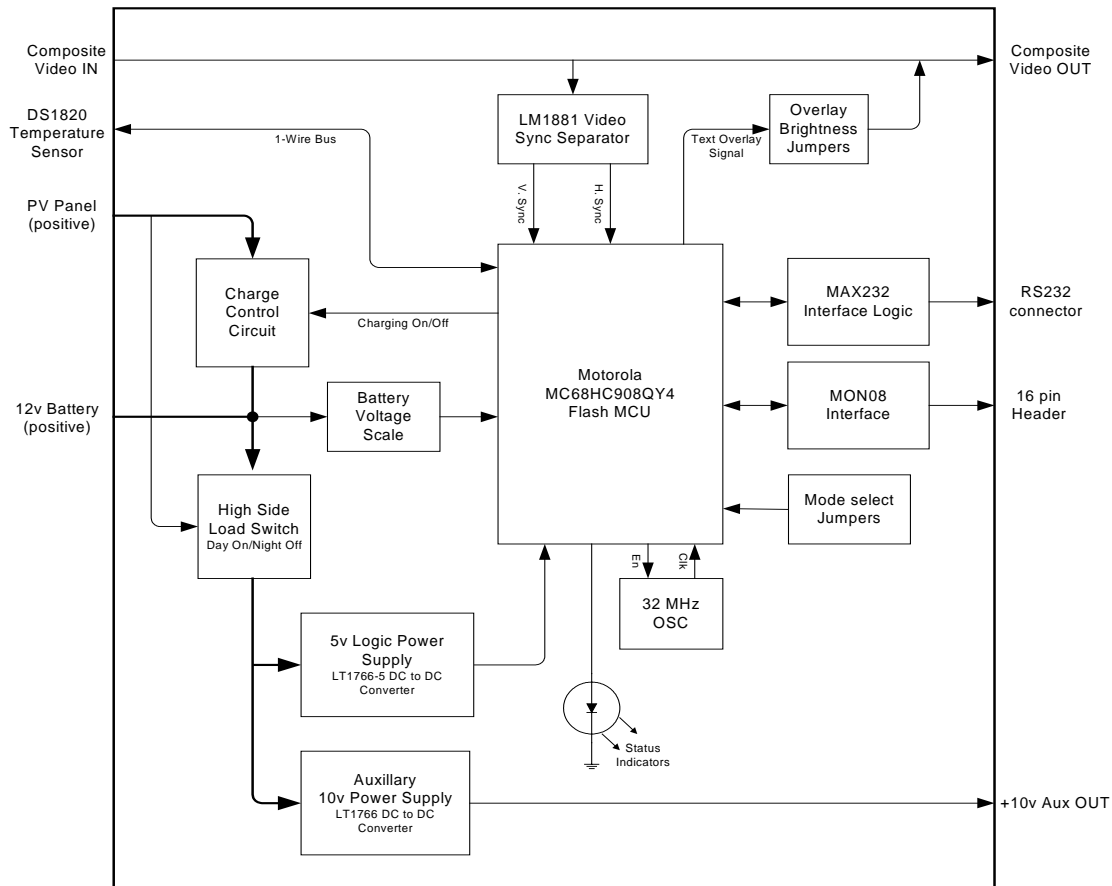


Figure 2: Block Diagram PVCC control Board

Microcontroller

A Motorola MC68HC908QY4CDW (U3) is the microcontroller (MCU) used in this design. A member of the Nitron Family, this part is a 16-pin SOIC which has 4K of FLASH memory program space, 128 bytes of RAM, and 14 I/O pins. The MCU can be powered at 3 or 5 volts. I chose the five-volt supply to allow the MCU to reach its maximum clock speed of 32 MHz. Table 1 breaks down the MCU resources and their use in this design.

QY4 MCU Resource	MCU Pin #	PVCC Usage
IRQ	9	Video Horizontal Sync (start of video scan line)
PTA3	8	Interface DS1820 1-Wire Bus Temp Sensor
OSC1	4	32MHz External Oscillator Input for OSD Active Mode
PTA4	5	Charge Enable Output
AD1	12	Analog Input to Measure Battery Voltage
PTA0	13	Communications with User's PC during Configuration
PTB0	15	Video Vertical Sync Detection (start of new video frame)
PTB1	14	Output to Enable/Disable External Oscillator
PTB2	11	Charge Control Mode Request Jumper
PTB3	10	Configuration Mode Request Jumper (has precedence over PTB2)
PTB4-6	7, 6, 3	LED Status Indicator Outputs
PTB7	2	Overlay Text Output Signal (added to video signal)

Table 1: MCU resource allocation

The PVCC has three modes of operation, which the user selects via jumper block JP5. The primary characteristic that differentiates these modes is the clock source: the internal RC oscillator or an external 32MHz oscillator. The external clock is needed when OSD is active. The MCU clock rate directly determines the maximum resolution of the overlay text. In the 908QY4's case, a maximum clock frequency of 32MHz results in a CPU clock rate of 8 MHz and a video dot rate of approximately 150 dots per line². When OSD is not active, maximum MCU speed is unnecessary; a lower clock speed reduces current drain from the battery. In this mode the external oscillator is disabled to further reduce power consumption.

Serial Interface

A serial communications connection to the PVCC is provided via J1 (DB9 connector) and U5 (MAX232). Both transmit and receive lines on the logic side of U5, and connect to PTA0. This arrangement follows Motorola's recommended low cost serial monitor port, which could be used to provide programming access to the MCU's flash memory. In this case, it is used to communicate with a personal computer to allow the user to configure the PVCC using the ConfigPVCC Win32 configuration program. The single pin arrangement is accomplished via a diode D8 and pull-up resistor R19. Although this arrangement forces communication to be half-duplex (simplex), it is more than adequate for sending and receiving product configuration information.

Mon08 Port

Programming access to the MCU's Flash memory is provided via JP6. This connector is wired per Motorola's MON08 specification. It is accompanied by dipswitch S3 (MON08 isolate), which is used to disconnect four specific MCU pins, needed by the MON08 interface. This port is included in this prototype design, but may be excluded from production versions. Switch S3 (MON08 Isolate) may not be reliable in the long term, but for development purposes a mechanical switch is simple and cost effective.

² Each NTSC scan line represents approximately a 62.5uS time span, although not all of this time is useable due to overscan on the screen edges.

Power Regulators

Two DC-DC converter type power regulators are built onto the PVCC board. Both regulators are based on Linear Technologies LT1766IGN chips. This device is a monolithic buck 200KHz switching regulator which accepts a wide input voltage range of 5.5 to 60 volts. This is important in PV charging applications, if the battery is suddenly disconnected while charging, some PV Panel can potentially produce open-circuit³ potentials as high as 30 volts (typically 23volts). Integrated circuit U2 is the five volt version of the LT1766, and provides Vcc (5 volts) for all of the IC's on the board. Integrated circuit U1 is an auxiliary supply used to power off board devices, in this case, the CCD video camera module and the 433MHz ATV transmitter. The voltage regulation point is set via resistors R7 and R8. In this design those resistors are fixed for an output voltage of 10 volts. The design of these converters closely follows the manufacturer's reference design, with only a few component value variations to enhance operation when the input voltage is close to the regulation voltage. The PCB layout for these devices also tries to follow manufacturer's recommendations to minimize switching noise, output ripple, and overall instability.

Video Sync Separator

The purpose of the LM1881 video sync separator is to provide timing signals to the MCU that allows it to generate a stable video overlay signal synchronized to the original video signal. This overlay signal is added back into the original video signal to create the overlay effect. Two signals are created by the LM1881; a vertical sync signal that is connected to port PTB0, and a composite horizontal sync signal that is connected to the MCU via the IRQ input. I studied a variety of hardware and software schemes before settling on this arrangement. Using the IRQ to sense the horizontal sync signal produces a consistently repeatable response by the MCU to the signal's arrival. The horizontal sync pulse can be thought of as the "Start of Line" signal that is used by the MCU to synchronize the output of "dots" via PTB7 in creating the overlay text. The MCU external hardware interrupt input allows the MCU to generate acceptably steady text that would not be possible with a conventional input. See the software section for a description of the techniques used to generate stable overlay text.

Charger Power Circuit

The charging control circuit provides the MCU with a way to control the connection of the PV solar panel to the battery. The MCU output PTA4 is used to drive a load switch consisting of a low power N-channel MOSFET and a high power P-channel MOSFET. Use of this load switch arrangement eliminates the need for high voltage used in high-side N-channel MOSFET switches. The charger power circuit consists of transistors Q1, Q2, and D1. Low power MOSFET Q2 provides the interface driver between the MCU port pin and high power MOSFET Q1. Q1 has an R_{DS} of 20m Ω , dissipating 2W at 10A. Diode D1 is a 20A schottky diode with a V_f in the .25 to .4 volt range. This diode provides protection against reverse current flow from the battery into the PV panel in dark or low light conditions. When the MCU wants to allow battery charging from the PV panel, it sets PTA4 output high. When the MCU wants to inhibit charging it clears PTA4.

³ Light loads, although technically not open-circuits, can still be exposed to high voltages from a PV panel if the battery is disconnected

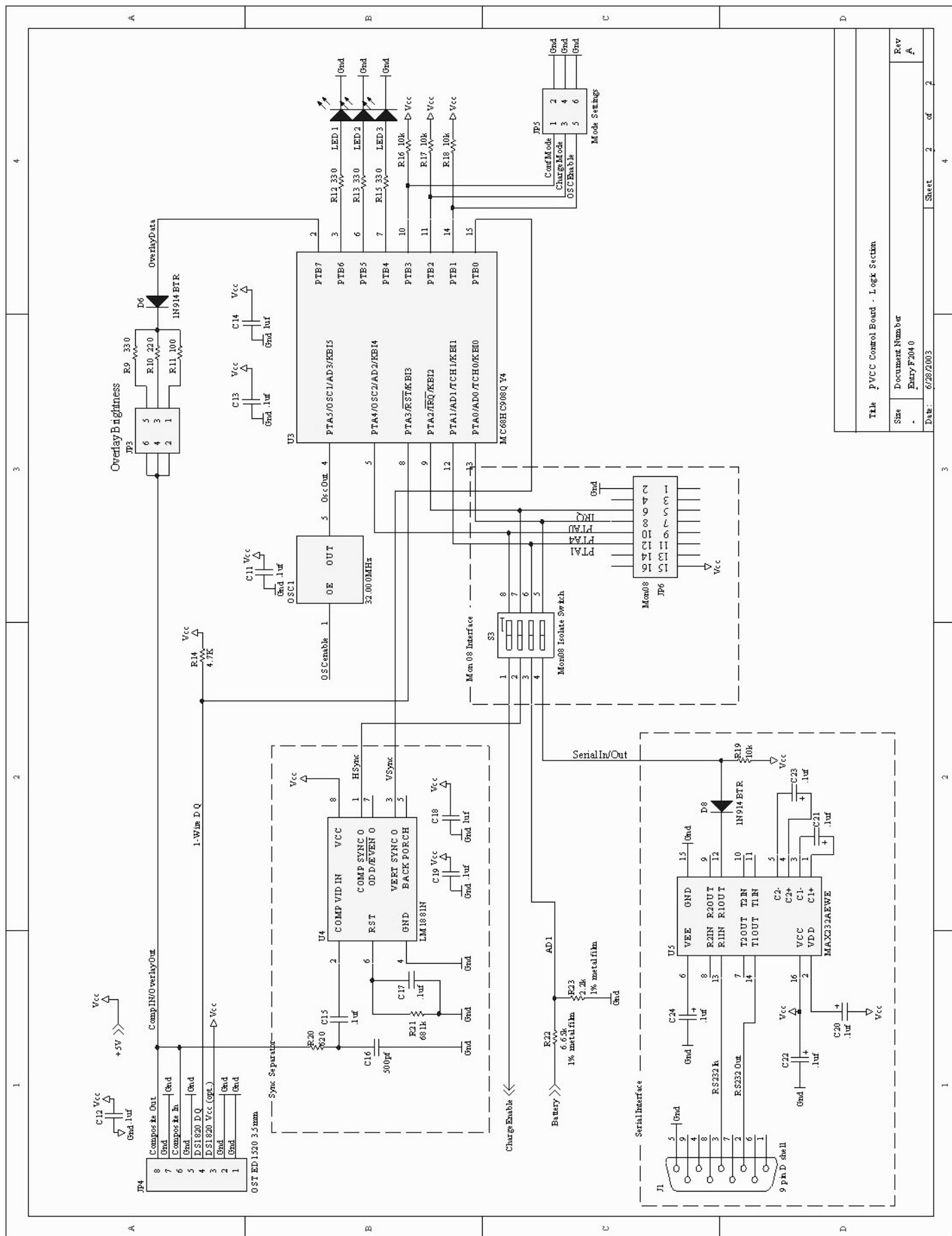


Figure 3b: PVCC schematic – logic section

Firmware

The MCU firmware for this project was developed in Assembly language using Metrowerks' Code Warrior version 2.1, and consists of three files: PVCC.asm, CharGenData.inc, and qtqy_registers inc. During development, the QY4's FLASH memory was programmed via a MON08 interface header using a MON08MULTILINK pod from P&E Micro. An overview of the firmware is presented here. See figure 4a and 4b for a procedural flow chart of the firmware executed by the MC68HC908QY4.

I devised three modes of operation and implemented them in three separate software loops. Functionality common to more than one mode was implemented in subroutines that could be called from any mode. The larger 4K FLASH memory of the QY4 made this arrangement possible, and I chose it for this project because I was not sure what code size I would have at the project's end. Table 2 summarizes the operating modes.

Operating Mode	Operating Characteristics	Clock Source
OSD Inactive Mode	Battery charge controller active External OSC disabled Lower power consumption (uses MCU's WAIT mode) COP enabled	Internal RC Oscillator Freq = 12.8MHz (3.2MHz bus)
OSD Active Mode	Battery charge controller active Text overlay active (battery's voltage & temperature) COP enable	External Canned Oscillator Freq = 32.000MHz (8MHz bus)
Configuration Mode	Communication between PVCC board and PC No charge controller active COP disabled	Internal RC Oscillator Freq = 12.8MHz (3.2MHz bus)

Table 2: Operating Modes

Regardless of the mode selected, just after startup, the MCU checks the current configuration data stored in FLASH for validity. The MCU performs a 16 bit cyclical redundancy check (CRC16) on the configuration data. That value is then compared to the value stored in the first two bytes of the configuration data. If the values do not match, the configuration data is assumed to be corrupt and is overwritten with "factory" default values stored in another region of FLASH. This should eliminate or greatly reduce the chance of damaging the system through the use of corrupt configuration data.

Configuration Mode

The Configuration Mode allows the user to configure the behavior of the PVCC board. In this mode the MCU will wait indefinitely for a single byte command from the PC. If a valid command is received, the MCU carries out the command and returns to wait for the next command. This mode is exited by moving the shorting block on JP5 and then power cycling.

Two commands are currently defined: Send Configuration Data and Get Configuration Data⁴. When the host PC issues a Send command the MCU waits for a packet of configuration data. A 16 bit CRC is computed on the data and compared to the CRC stored in the data packet. If they match, it is assumed that no corruption of data occurred and the data is programmed into the configuration data area in FLASH memory at \$FD00.

This mode makes heavy use of ROM resident communications and FLASH programming routines, listed in table 3. These function are used to establish a communications link between your PC and the PVCC board. I wrote a Windows compatible program called ConfigPVCC which runs on the PC to allow easy setup of the PVCC board. Refer to the Software section for details on the features can be configured using ConfigPVCC.

⁴ The directional sense is defined with respect to the user's PC (host)

ROM Resident Function Name	Entry Address in QY4	Description
GETBYTE	\$2800	Read one byte from PTA0
PUTBYTE	\$FEA1	Send one byte out PTA0
RDVRRNG	\$2803	Read & verify range of FLASH (send memory range out PTA0)
ERARNGE	\$2806	FLASH erase range
PRGRNGE	\$2809	FLASH program range

Table 3: Rom resident software routines

The communications baud rate generated by these routines with a correctly trimmed oscillator is approximately 12800 baud when the MCU is using the internal oscillator and in user mode. The PC achieves this baud rate using a UART clock divisor of 9. Although not a standard baud rate, all PC UARTs are capable of this speed. I found the following sources of information helpful when working with the QY4's resident ROM routines: Motorola application Note: AN1831 "*Using MC68HC908 On-Chip FLASH Programming Routines*" and AN2346: "*EEPROM Emulation Using FLASH in MC68HC908QY/QT MCUs*"

Charge Control w/OSD Inactive Mode

In this mode, the MCU clock is derived from the MCU's internal oscillator. After initializing the I/O ports, the oscillator trim, the A/D port, and the system timer, the system enters the WAIT mode, lowering power consumption. The WAIT mode is exited every 4ms in order to reset the COP timer and check a semaphore to see if the user specified sample period has expired. Then the battery's voltage and temperature are sampled. Battery voltage is read via the integral analog/digital converter via input AD1. Resistors R22 and R23 form a resistor divider to scale 20 volts down to the 5 volts needed by the analog input. The temperature sensed by the 1-Wire DS1820 sensor is read and checked against a user defined battery temperature limit. The sampled battery voltage is compared to the user-defined set points and the charging circuit is turned on or off accordingly. This is controlled by output PTA4.

Charge Control w/OSD Active Mode

In this mode, the MCU clock is derived from an external 32MHz oscillator. The I/O ports, the system timer, and the A/D converter systems are initialized, and the main loop is entered. The charge control algorithm works just as it does in the "OSD Inactive Mode", but in this mode, the on-screen display logic is active and is responsible for "drawing" nine characters on the video signal generated by the camera module. The video signal and the firmware synchronize via the video's horizontal sync component. This component signal is obtained by an LM1881 video sync separator chip and is connected to the MCU's IRQ input. The vertical sync signal is also made available to the MCU via port PTB0. Each horizontal sync pulse that triggers an interrupt represents the start of a new scan line. These can be counted in order to determine when "drawing" should begin on the screen in a particular frame. The vertical sync signal indicates when a new video frame is beginning and that the scan line counter should be reset to prepare for the new frame.



Picture 2: Text overlay on video

To create overlay text with a high enough resolution to be useful and minimal jitter, required that I take several creative steps. The MCU's response to the horizontal sync (Hsync) pulse had to be as consistent (from scan line to scan line) as possible. A variation of only three clock cycles (at 8 MHz) would be noticeable to the viewer as a side-to-side jitter. Variations due to three or four extra instructions being executed would create a very blurry jitter in the overlay text. Minimizing jitter ruled out monitoring the horizontal sync signal with a normal input and a tight software loop to detect its transitions. Use of the IRQ input had the best chance of reducing jitter because of its inherent repeatability. But, even the IRQ mechanism has a

variable latency based on the instruction that is being executed when the interrupt occurs. The HC08 CPU family enforces run to completion (RTC) execution of instructions. This means a possible response of 1 to 9 clock cycles, with an average variation of about 3 clock cycles for most code. To eliminate this variation, I used a tricky but effective technique. When the scan line counter (incremented by each Hsync pulse in the IRQ ISR) reaches the “starting scan line” minus one, the IRQ ISR is forced to no longer return to the interrupted background code but instead to return to a WAIT instruction (this is done by modifying the stack in the ISR). When the next IRQ interrupt occurs approximately 62.5uS later, the WAIT mode aborts and again the interrupt is serviced. This is repeated for the next 16 interrupts (scan lines) as the text is “drawn” onto the video signal. After the text overlay is completed for the current frame, the IRQ ISR returns to the originally interrupted code. The whole process takes approximately $16 * 62.5\mu\text{S}$ or 1mS. Because of the consistency of interrupting the WAIT instruction, this technique produces nearly jitterless overlay text.

This technique may have negative side effects on the background code, due to interrupted execution for an entire millisecond. During development I had a problem with the Dallas DS1820 temperature sensor code failing periodically. The solution was to simply synchronize the access of the DS1820 with the execution of the video overlay code. This was done via a semaphore (a bit) that was set at the end of the video overlay process. With this semaphore in place, it assured all accesses to the DS1820 would not be interrupted by the overlay code. This is because no 1-Wire bus command was longer than the video inter-frame time of 16.66mS. Another possible, but less glamorous, solution to this problem would have been to disable video overlay operations for a short time period while the temperature sensor was being accessed. I decided not to use this approach because of the blinking effect it would have caused. The use of semaphores, even simple ones, can be a source of many problems. I took extra care to address the possible loss of the video signal, in order to prevent a deadlock in code waiting for the video overlay to finish. Now any potential deadlock created by the loss of the video signal is avoided by the modulo TIM interrupt. If the video signal is lost for more than two TIM modulo counter overflow interrupts, a release semaphore is set to release any code that would normally wait for the overlay complete semaphore. In the event of the video signal returning, the release semaphore is cleared and the system returns to its normal behavior.

The OSD feature currently displays 9 characters in an 8 row by 7 column per character format. Obtaining a high enough resolution to be useful for my system, meant finding a fast way to shift the character data, or “dots”, out of port PTB7. After four attempts I believe I have found the most efficient way possible using an HC08 family processor. When the IRQ ISR has determined that it is time to commence “drawing” on the video signal, the background code has already converted the ASCII characters to be display, via lookup table, into character font data. That data is placed into a Video RAM memory area that is arranged such that one scan line of each character, for 9 consecutive characters, is placed in 9 contiguous memory locations. This is followed by the next scan line, repeating until the bottom of the character cell is reached. So it is a simple matter for the interrupt routine to shift 9 consecutive bytes from RAM out port PTB7. Any time spent accessing the next byte in the video RAM will show up on the screen as an inter-character space. To minimize this space, I first unroll all loops to get rid of the overhead of checking for loop termination conditions. Next, I push all of the data for one scan line onto the stack in the reverse order it is to be displayed. Stack instructions take only two clock cycles and require no overhead to increment index registers. See source file PVCC.asm starting at line 1500 for the code described here. To summarize, one “dot” period is 3 clock cycles @ 8MHz or 375nS. The inter-character space⁵ period is 2 clock cycles or 250nS. Assuming a total horizontal scan time of 62.5uS for NTSC video, this gives a display resolution of approximately 167 dots per line. But every TV/Monitor will use some of that scan time to “overscan” off the edges of the screen, reducing the resolution to somewhere in the 140 to 150 range. The total time for one character and its inter-character space is 22 clock cycles or 2.75uS. This means one line could in theory display about 18 to 20 characters. Alas, my technique uses 7 bytes of RAM for each character being displayed, so I elected to display 9 characters, consuming 63 bytes of RAM or half the total RAM on the QY4 MCU for the video OSD feature.

⁵ The first bit of every row in the character ROM is always a zero. This makes the inter-character space always appear to be 625nS wide.

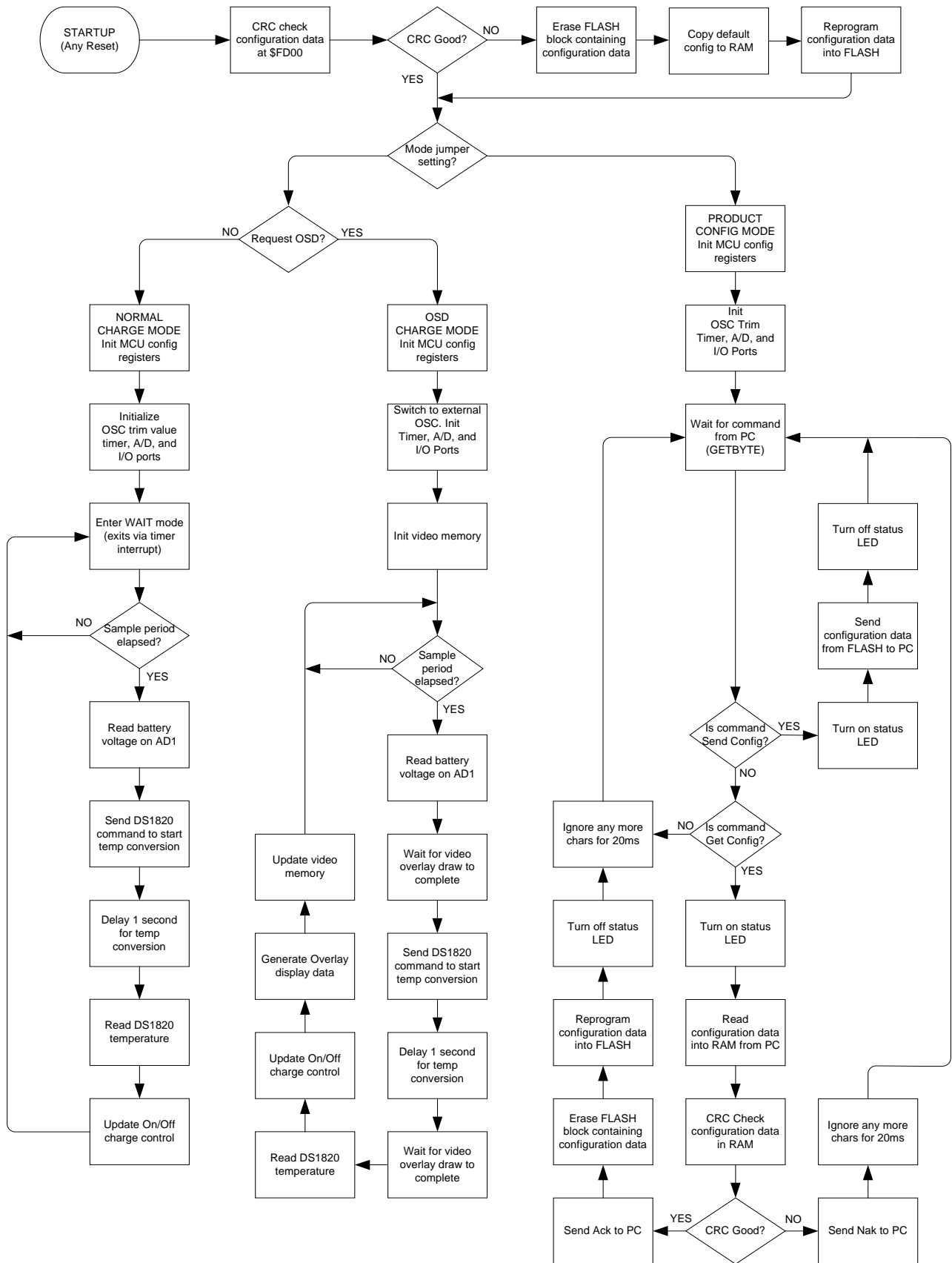


Figure 4a: Firmware flowchart, background code

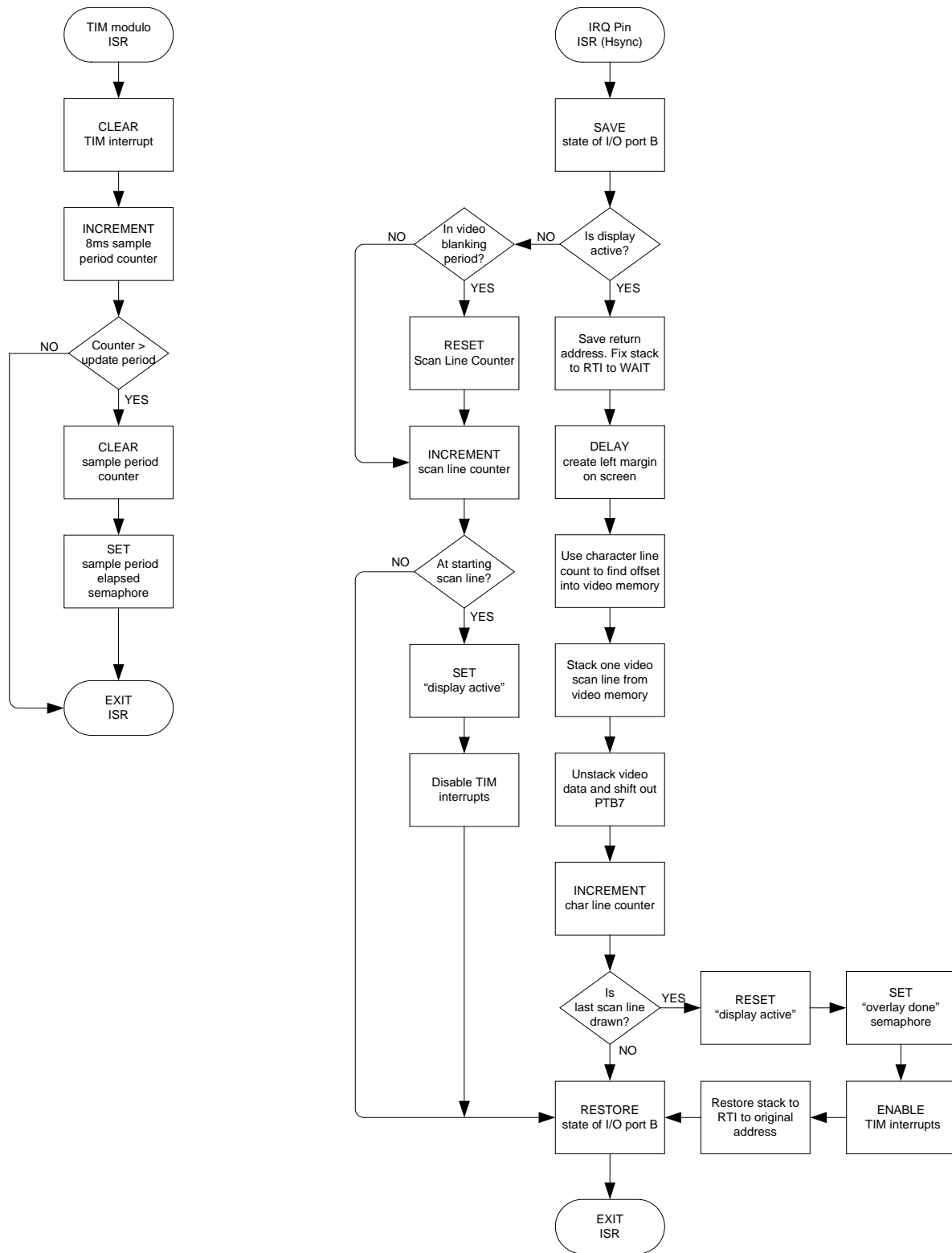


Figure 4b: Firmware flowchart, Interrupt service routines

PC Configuration Software

I developed a configuration utility in C/C++ using Microsoft Visual C++ 6.0 that allows the user to configure the PVCC board's behavior. The PVCC Configuration Utility communicates with the MCU via serial cable while in the Configuration Mode. This program is a MFC dialog-based application with two main threads of execution; the GUI thread and a communications (COM) thread. The GUI thread communicates with the COM thread via event semaphores. The COM thread implements the communications protocol with a simple state machine. The program's interface is shown below in figure 5 and is very self explanatory.

The most important product behaviors that can be controlled by the PVCC Configuration Utility are the charge On/Off set points and the OSD. The user can specify the PV full charge set point (stop charging) and the PV reconnect set point (start charging). To turn on the OSD (OSD Active Mode), the user selects the Enable OSD check box. When OSD is on, the unit consumes more power due to the need to clock the MCU at its maximum frequency of 32MHz using an external oscillator. When in the OSD Inactive Mode, the unit uses less power by using the MCU's internal oscillator. In this mode the external canned oscillator is disabled to further reduce power consumption.

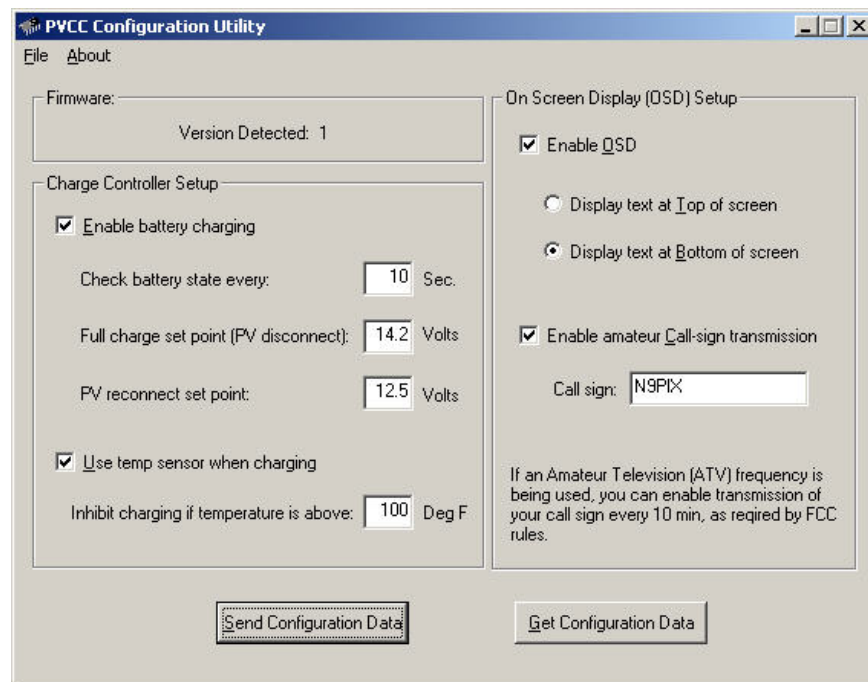


Figure 5: The PVCC Configuration Utility interface.

When the user clicks the Get Configuration Data button the utility will request the configuration data currently stored in the MCU's FLASH memory and display it on the dialog. The user can enter any desired changes to the settings and then click Send Configuration Data. This will send the revised data to the MCU to be programmed into FLASH, overwriting any previous data. A CRC16 check is used in both Get and Send data packets in order to avoid accepting corrupt data.

Future Enhancements

There are a variety of ways one might enhance this project. Listed below are several enhancements I have started or have planned:

- I plan to add a simple boot-loader to this product to allow in-field FLASH firmware upgrades. The ConfigPVCC program will be extended to parse .S19 files, and transfer the data to the PVCC via the existing serial port. A good starting point for this can be found in AN2295/D Developer's Serial Bootloader for M68HC08
- I plan to add simple linear temperature compensation to the charge control algorithm, which will adjust the "full charge set point" based on the battery's temperature. This will allow for more accurate charging and less chance of overcharging in cold weather (Wisconsin has plenty of that).
- A possible enhancement to this product is the use of a PWM charging algorithm. This technique is very popular today in many commercial charge controllers. PWM based charging will tend to extend the working life of the battery when compared to the On/Off charge control method.
- When another spin of the PCB is done, I want to add a Low Voltage Disconnect (LVD) feature to the charge controller. This will allow the MCU to control whether the loads are on or off. If the battery's voltage drops below a LVD threshold, the MCU will turn off the loads to prevent draining the battery to the point where the controller can no longer operate, and require user intervention.

Conclusion

This prototype system is going to be installed over the July 4th weekend to replace the original less reliable system described above in the History section. With the experience gained from this installation I hope to further improve the design, keeping an eye on broadening the project's usefulness in other solar applications. The market for solar charge controllers is over supplied and cut throat, but few offer integrated solutions for remote solar powered data and observation stations.

Working with the FLASH-based QY4 MCU was easier than I expected it to be. The CodeWarrior tools were free and worked very well. There is plenty of online documentation to get one started, and the Motorola application notes are a treasure trove of engineering development help. There is also a great resource in the Yahoo hosted microcontroller discussion groups. The 68HC05_08 group had a lot of discussions related to the Nitron parts.

The only stumbling block I experienced was with the P&E programming software initially being flaky under Windows XP. After some tinkering I discovered that boosting the execution priority of the process to "Above Normal" eliminated the problems. I created a batch file to speed the process, which contains:

```
start /AboveNormal C:\Progra~1\pemicro\pkg08sz\prog08sz.exe
```

Thank you, Motorola and Circuit Cellar for considering my Flash Innovation contest entry. I am honored to be entering my design.

Bill of Materials

QTY	Part # or Name	PCB designation	Description
1	LT1766IGN	U1	1.5A, 200kHz Step-Down Switching Regulator
1	LT1766IGN-5	U2	1.5A, 200kHz Step-Down Switching Regulator 5v
1	LM1881N	U4	Video Sync Separator
1	MC68HC908QY4	U3	FLASH 8 Bit HC08 Microcontroller 16 pin SOIC
1	MAX232AEWE	U5	Multi-Channel RS-232 Driver/Receiver
1	IRF4905S	Q1	Power MOSFET SMD220
1	IRFR9024	Q4	Power MOSFET DPAK
2	IRLML2803	Q2, Q3	Power MOSFET SOT23
1	20L15TS	D1	Schottky Diode
4	1N914BTR	D2, D3, D6, D8	Small signal Diode
2	10MQ060N	D4, D5	Schottky Diode
1	32.000MHz Can	OSC1	Half size DIP high freq. crystal oscillator
2	1% metal film res	R7, R8	Resistor SMD1206
2	1% metal film res	R22, R23	Resistor SMD1206
20	Resistor	R1, R2, R3, R4, R5, R6, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R24	Resistor SMD1206
2	ECPU1C105MA5	C1, C2	1uf 1210 SMD Capacitor
2	ECU-V1H221KBM	C8, C9	220pf 50v 1206 SMD Capacitor
2	ECU-V1H223KBM	C6, C7	.022uf 50v 1206 SMD Capacitor
1	T494X107M020AS	C5	Low ESR 100uf Capacitor
2	T495X226K035AS	C3, C4	Low ESR 22uf Capacitor
1	T494D107M010AS	C10	Low ESR 100uf Capacitor
9	Cap 1205	C11, C12, C13, C14, C15, C16, C17, C18, C19	0805/1206 SMD Capacitors
5	Cap 1205	C20, C21, C22, C23, C24	0805/1206 SMD Capacitors
2	Inductor	L1, L2	100uH Sumida CRDH Series Inductor
3	LED2	LED1, LED2, LED3	Light Emitting Diode
2	Littlefuse ATO	Batt Fuse, PV Fuse	Fuse
1	9 pin D shell	J1	Receptacle Assembly, 9 Position, Right Angle
1	Mode Settings	JP5	Header, 3-Pin, Dual row
1	Mon08	JP6	Header, 8-Pin, Dual row
1	OST ED1518 3.5mm	JP2	Connector, 6-Pin
1	OST ED1520 3.5mm	JP4	Connector, 8-Pin
1	OST ED2227 5.08mm	JP1	Connector, 4-Pin
1	Overlay Brightness	JP3	Header, 3-Pin, Dual row
1	Switch header	S1	Double-Pole, Single-Throw Switch (Optional)
1	Switch header	S2	Single-Pole, Single-Throw Switch
1	Mon08 Isolate Switch	S3	DIP Switch 4-Pin

References

Motorola AN1292 - [Adding a Voice User Interface to M68HC05 Applications](#)

Motorola AN2312 - [MC68HC908QY4 Internal Oscillator Usage Notes](#)

Motorola AN1831 - [Using MC68HC908 On-Chip FLASH Programming Routines](#)

Motorola AN2295 - [Developer's Serial Bootloader for M68HC08](#)

Motorola 68HC05_08 discussion group - http://groups.yahoo.com/group/68HC05_08/

Dallas Semi. AN162 - [Interfacing the DS18X20/DS1822 1-Wire Temperature Sensor in a Microcontroller environment](#)

The Quagi Antenna Turns 30 - <http://commfaculty.fullerton.edu/woverbeck/quagi.htm> - Wayne Overbeck, N6NB

Sandia National Labs: Batteries and Charge Control in Stand-Alone Photovoltaic Systems Fundamentals and Application - James P. Dunlop, P.E.