# Doing More with Fewer Wires in the Harness: A New Approach to Spacecraft On-Board Command and Telemetry Interfacing

by
Bdale Garbee, KB0G, Chuck Green, N0ADI,
Lyle Johnson, KK7P, and Stephen Moraco, KC0FTQ

## Introduction

*The new approach to handling on-board command and telemetry described here replaces much of a traditional spacecraft wiring harness with a CAN bus connecting standard module interface boards. Using this approach reduces mass and integration complexity, provides flexibility, and simplifies and facilitates testing of individual spacecraft modules and groups of modules before full satellite integration.*

*A new, lightweight protocol designed to take advantage of unique features of the CAN bus architecture delivers a simple, robust, and powerful solution for multiplexing large numbers of control signals and telemetry channels on a single twisted-pair in the wiring harness. A flexible module interface board and associated spacecraft bus interface connector become standard components, simplifying module design and allowing payload experimenters to concentrate on their payloads, not spacecraft interfaces.*

*The hardware and software detailed in this paper are scheduled to fly on the AMSAT-NA Eagle spacecraft intended for high-altitude elliptical Earth orbit. While the initial focus is on meeting all of Eagle's requirements, components and protocols designed for this system have wide applicability to other spacecraft of similar complexity.*

*Prototypes of the module interface boards have passed radiation and other relevant tests. Final module circuit board designs and flight software are nearing completion.*

## Background

Traditionally, most satellites used a point-to-point wiring harness to connect all of the onboard electrical and electronic components. As spacecraft become larger and advances in technology allow more features to be packed into each module, the complexity of this wiring harness increases. As the complexity increases so does the mass and the administrative burden of keeping track of which wires go where in the wiring harness design.

One of the successful innovations in the original AMSAT Microsat design was the use of a multi-drop serial bus for command and telemetry. The Microsat bus was based on an obsolete Motorola part called the AART, which provided a modest number of modules in close proximity with a modest number of control and telemetry points. More recent micro-satellites have employed other serial bus technologies like SPI with similar success.

Early in the Phase 3D program that resulted in AO-40, CAN bus was proposed as an alternative to the traditional wiring harness. The CAN (Controller Area Network) bus was originally designed primarily for use in automotive electronics. While it has grown in popularity and is fairly widely used within the embedded systems community today, in the early days of the Phase 3D program it was brand new with no real track record. To manage risks, the decision was taken to build AO-40 with a traditional wiring harness, which turned out to be the most complex such harness ever built for an AMSAT spacecraft! However, a CAN bus was flown as an experimental interconnect, and serves as the primary communications path between RUDAK and all of the scientific experiments. This CAN experiment on AO-40 was highly successful, and CAN bus technology has also now flown on a number of other small satellites including several built by SSTL.

## The Idea

For the AMSAT-NA Eagle project the size and complexity of the spacecraft suggested to us that it might be a good time to revisit the idea of replacing at least part of the wiring harness with a modern, robust multi-drop serial bus design. After considering various alternatives, CAN bus was selected as the base technology.

Drawing from the design of the AART boards in the Microsats and the CAN SmartNode boards on AO-40, we have designed and are building a number of identical small circuit boards, one of which will be placed in each module to provide a standard interface between the spacecraft bus and the module's electronics. These circuit boards contain an Atmel microcontroller with CAN interface, power switch, temperature and current sensors, connectors for the spacecraft and module interfaces and signal conditioning circuitry for all of the inputs and outputs.

The Eagle team intends to use a new housekeeping computer design called IHU-3. This IHU is derived from the designs flown on previous Phase 3 spacecraft including AO-10, AO-13 and AO-40 and is intended to reuse much of the software developed for those missions. Thus, one of the design constraints for this project was to provide a conceptual equivalent to the I/O multiplexer that was associated with previous AMSAT IHU designs. In effect, what we have done is to design a system for distributing the I/O multiplexing among the various modules instead of centralizing it in the IHU. Fortunately, this turns out to be an excellent match to what CAN bus does best.

We also realized during the Phase 3D project that having each module use a different set of connectors and pinouts for attaching to the wiring harness made bench testing of modules difficult. Special test boxes and wiring harnesses had to be developed for each module or set of modules, and these were hard to keep up with in the AMSAT lab environment. A major benefit of this project is that key interfaces between each module and the spacecraft harness are identical for every module, and a single test harness can be used to test many individual modules or groups of modules.

As we surveyed the needs of the satellite module builders it quickly became apparent that there were at least three classes of *consumers* for the CAN module. Three main operating modes are implemented that are easily selectable by the module builder. We refer to these as normal, multiplexed and byte-pipe modes.

## Electrical Design

There are two components to the hardware in this system. The first is inclusion of a CAN interface on the IHU(s) on the spacecraft, which is left to the IHU-3 team to document.

The second is implementation of the small circuit boards that will be included in each payload module and we describe here.

The core component on the *widget board* is an Atmel T89C51CC01 microcontroller. This part is a derivative of the venerable Intel 8051 architecture including a number of digital input and output lines, an integrated analog to digital converter, a CAN interface, onboard Flash program memory and data RAM.

Jumper locations on the board allow for six bits of address selection allowing up to 64 modules and two bits of mode selection. Currently we implement three of the four possible modes - normal mode, a *multiplexed* mode supporting expansion of the digital control and telemetry line count and a byte pipe mode for communication with a processor in the module's electronics.

In normal mode the features provided by the widget board include a power switch, current sensor, temperature sensor, 12 digital output lines, 8 digital input lines and 5 user-defined analog sensor channels.

In multiplexed mode the digital output and input lines from normal mode are replaced by support for up to 8 banks of external multiplexers. The output lines are allocated as 8 for the mux data bus, 3 for mux select and one for a strobe. The input lines are converted to a mux data bus using the same select and strobe lines as the outputs. Thus, in this mode the widget board provides a power switch, current sensor, temperature sensor, up to 63 lines of digital output, 64 lines of digital input and 5 user-defined analog sensor channels.

In byte-pipe mode the widget implements distinct 8-bit input and output busses with simple handshaking. The feature set in this mode includes the power switch, current sensor, temperature sensor, 8-bit input and output busses each with strobe/ack handshake lines, two independent output lines and three user-defined analog sensor channels.

Various ideas have been discussed for other modes we might support including various flavors of serial byte pipes. There are also proposals for reusing this hardware with fully custom firmware for some special sensor applications. No commitments have yet been made by the team to support any of these other applications.

### Circuit Description

Please refer to the schematics on pages 16 and 17 for this discussion.

A 15-pin male D-sub connector (P1) interfaces to the spacecraft and brings in nominal 14V power, provides CAN connections and passes through up to five user-definable signal lines which connect directly to the associated module. Redundant CAN connections are provided so the spacecraft wiring harness can *loop through* the module without splicing wires. An *EB* line is provided passed through to the user module, which allows tapping the IHU engineering beacon data stream if required by the module.

A 40-pin connector (J5) organized as 2 rows of 20 pins all on 2.00 mm centers attaches to the module and brings all available signals to the module.

Incoming power is fused, filtered and applied to a power switch (FET Q2 and associated electronics) and to a switching regulator (U7) which provides the required +5V to run the CAN module.

Analog circuitry includes a temperature sensor (U12) and a means to monitor the current consumption of the attached module (U8). Remaining analog circuitry conditions the ADC inputs with filtering and clamping. Analog signals are then applied to the eight ADC inputs of the microcontroller (U5).

CAN signals are filtered by U3 and applied to U4, which converts from the CAN physical layer-signaling scheme to standard CMOS 5V logic levels. U4 has its own power supply filtering to minimize noise effects from the CAN bus. The CMOS signals are interfaced directly to U5.

U5 includes clock circuitry and is set to 8 MHz by crystal Y1. U6 provides a regulated analog voltage reference independent of the accuracy of the 5V supply.

Shift register U2 allows the microcontroller to read the jumpers at reset or on command. The jumpers set the widget address as well as its operating mode.

Schottky diodes and current limiting resistors isolate the U5 digital outputs. The user module must provide pull-ups to not more than +5V and the pull-up value should be at least 10K ohms.

### Radiation Testing Results

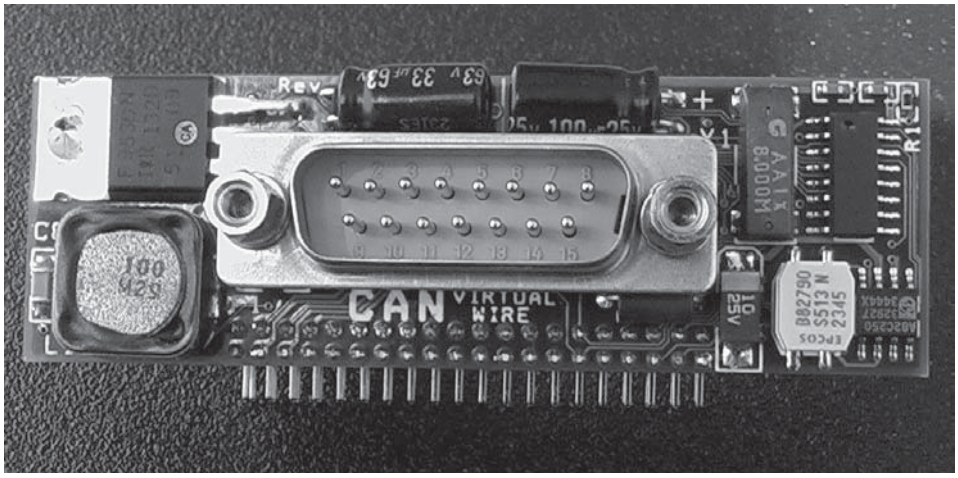
We were pleasantly surprised by how well the Atmel microcontroller survived our radiation testing. By arrangement with the University of Virginia Medical Center, prototype CAN widgets were exposed to a calibrated radioactive source in controlled dosages. The module was then evaluated for power consumption and functionality. These tests were repeated until failure, which occurred at some 60 kRads.

In addition to the confidence this testing provided, we were able to directly apply some of its results to the circuit design. For example, the use of the clock oscillator on the microcontroller, which saves tens of milliwatts of continuous power consumption, was only decided upon after evaluation of the radiation results. This may result in a power savings of a watt or so in a moderately complex spacecraft like Eagle.

To further enhance reliability in the radiation intense GTO orbit environment AMSAT will continue our traditional practice of applying additional shielding material to critical ICs. This shielding will likely be implemented with 1mm of Lead affixed to the relevant components.

### Mechanical Design

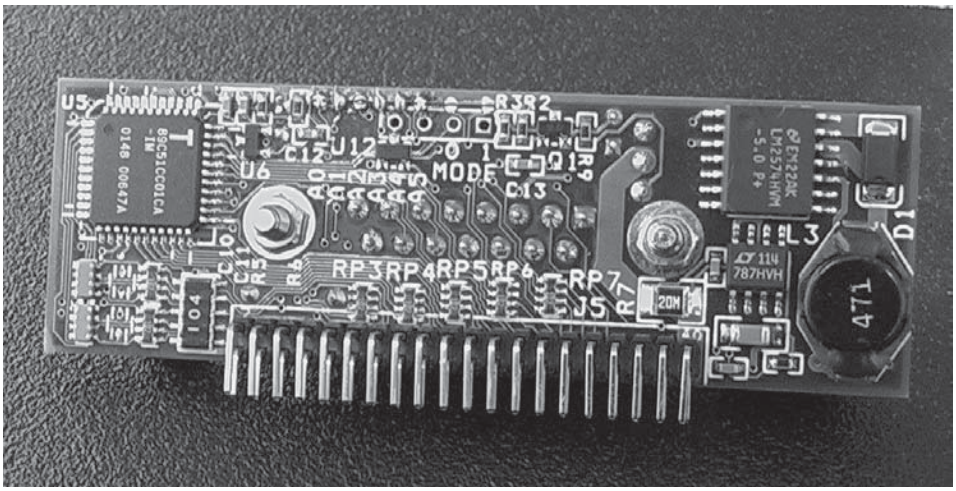
As usual, the process of laying out the PCB was more an exercise in mechanical engineering than electrical engineering. The mechanical objectives were simple to state but a bit more difficult to implement. Keep it as small as possible while maintaining a form factor that would fit into any size box available to the module developers. And provision for mounting the board must be adequate to




*Circuit Board Showing Sub-D Connector*

withstand vibration levels anticipated during launch.

Throughout this section we freely mix metric and imperial units. When working with electronic components any attempt to standardize is futile. And trying to standardize on a single set of dimensional units will only lead to errors.




***PCB with Downward Facing Plug***

The minimum box thickness is one inch so a board height of 24mm was chosen. The width would be whatever was necessary to contain all the parts. There are a total of 67 parts on the board (not counting radiation shields) with sizes ranging from 0603 resistors to the 15-pin Sub-D connector. The width turned out to be 74mm.

Efficient use of volumetric space was also desirable. The 15-pin Sub-D connector established the space needed between the PCB and the inside edge of the box it is mounted in to about 1/4 inch. So all high-profile parts should be on this side of the PCB to make maximum use of this space. Only low profile parts can go on the backside of the PCB. The tallest part on the back of the PCB stands out about 3.5mm with the exception of the 40-pin connector that plugs into the PCB containing the module function.

In some cases it may be desired by the module developer to extend their PCB all the way to the connector side of the box. If the box height is sufficient this can be done by inverting the orientation of the 40-pin connector so that the right angle pins point down rather than up. This connector is located at the bottom edge of the PCB to accommodate this choice and centered left-to-right along this edge.

The 15-pin Sub-D connector is located in the geometric center of the PCB. It is mounted to the PCB by stand-offs so that when the connector is secured to the side of the box the PCB is also securely held in place.

When a part requiring a radiation shield (IC's and FET transistors) is located on the PCB a similar part should be located on the opposite side of the PCB. This way, a radiation shield attached to the top of one part can also shield the bottom of another part thereby reducing the total number of shields needed.

The FET module power switch should be mounted on the side of the PCB facing the box edge to maximize heat radiation coupling to the side of the box rather than back into the module contained in the box. Similarly, the thermistor should be on the backside of the PCB facing the application module.

The above requirements made component placement challenging. But an even bigger challenge was connecting everything properly. Some traces needed to be wide to allow for significant currents. Other traces could be very narrow. However, if you make them too narrow the PCB manufacturer will experience manufacturing problems. The narrowest traces on the PCB are 0.006 inches wide and the smallest spacing between traces is also 0.006 inches. This is a four-layer board.

Most components are SMD and these units will likely be assembled by hand.

## Protocol Design

Our protocol design is based on a simple transaction model that takes full advantage of the unique characteristics of the CAN bus. We use the 11-bit addressing mode on the CAN bus instead of the longer 29-bit mode to reduce packet size since the shorter addresses used in the 11-bit mode are more than sufficient. Modules generally do not speak unless spoken to (input packets from a module in byte pipe mode are the exception), and the transaction model is as close to stateless between transactions as possible.

CAN is a message oriented bus so each packet contains a single message or stream address. The data payload of each packet is small - a maximum of eight bytes. All of the low-level packet handling is implemented in the CAN controller hardware, including error checking and retransmission.

The design of the CAN bus encodes packet priority in message addresses. When two CAN devices try to transmit at the same time the higher priority packet succeeds and the lower priority packet is deferred until it is the highest priority packet. Our allocation of address bits makes the module address less significant than the stream type, which reflects our sense of priorities in the spacecraft environment. For example, an IHU can be confident of always being able to write output state vectors to all modules even if some modules are responding oddly or not at all.

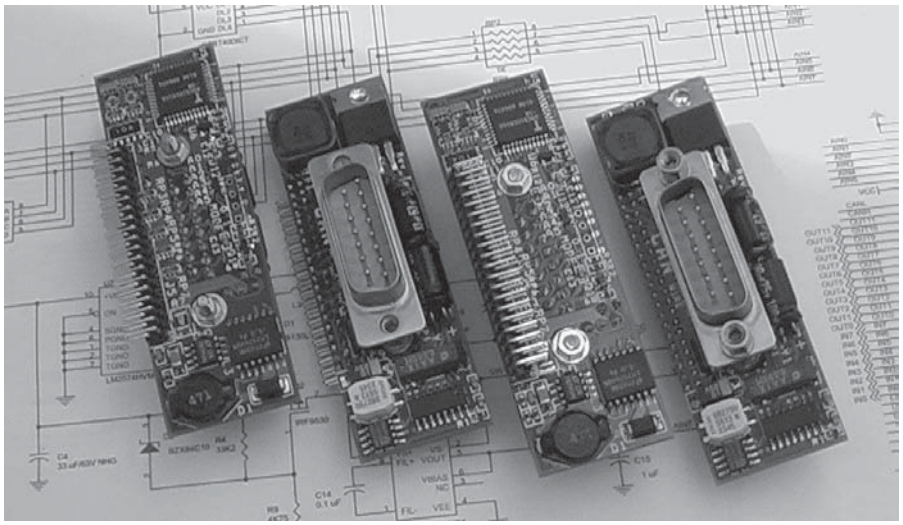
In normal operation an IHU sends a single CAN packet to each module that contains a complete state vector to be written to the module's outputs. Each module then responds with a short burst of packets that contain a state vector representing the complete input state of the analog and digital inputs to from the module. In this way each transaction completely refreshes all of the control lines and gathers a complete set of telemetry points from a module.

In normal and multiplexed modes the output state is contained in a single CAN packet and the input state requires two or three CAN packets for normal or multiplexed mode respectively. In byte pipe mode the output state, other than data for the byte pipe, is contained in a single CAN packet and the inputs require two reply packets. The data to be piped is sent and received in single CAN packets of eight data bytes each that are separate and at a lower bus priority than the command/telemetry packets.

For maximum compatibility with IPS software expectations the protocol is designed around the idea that each module might be asked to engage in a command/telemetry transaction as often as 50 times per second. At this repetition rate it is not possible to interact with a full set of modules all in multiplexed mode on a 800 kbit/sec CAN bus. However, all reasonable configurations of modules for all spacecraft currently contemplating use of this design are easily handled with bandwidth to spare.

## Firmware Design

From the onset of this project, we intended to release the entire design as Open Source. This has guided many aspects of the firmware development from the choice of tools we use to our internal documentation style. All of our tools are themselves Open Source. Where one did not exist, we created it and are releasing it as Open Source. The tools we found are an 8051 family assembler and a disassembler. We created our own flash utility for reprogramming the microcontroller as we were not able to find an Open Source tool for this purpose. Our internal documentation is quite descriptive, perhaps more than typical for a project of this type. The reason, in addition to this design being open, will be described shortly.



***Circuit Board Group***

Other philosophies guide our implementation. This device implements our spacecraft wiring harness. As such the expectation is that it *just works*. In fact, it must or we can easily lose our spacecraft. To meet this expectation we adopted a philosophy seen elsewhere in AMSAT designs - what is not flown in orbit will not break in orbit. To a firmware designer this means keep the amount of code to a minimum. Make each routine simple yet efficient. Use only the features within the microcontroller we need to accomplish our function.

Another constraint on our firmware design is that we want this to work in the traditional AMSAT spacecraft environment. The currently selected IHU is the master controller of the CAN bus. Modules speak only when spoken to. Our firmware design supports all modules on the CAN bus being spoken to 50 times per second. The only exception to this "speak when spoken to" policy is the byte-pipe traffic. Byte pipe traffic is implemented using lower priority messages so that it will not interfere with the high priority effort of writing state vectors and reading telemetry from the modules. It consumes bus bandwidth not used by the higher priority traffic.

Delving a little deeper into the firmware implementation, the Atmel T89C51CC01 part is rich in onboard devices. Our current firmware makes use of the 256 Bytes of on-chip RAM, 32K Bytes of on-chip Flash Memory, two of the three 16-bit timers/counters, one of the channels of the 5 channel 16-bit programmable counter array, the hardware watchdog timer, the 10-bit resolution Analog to Digital Converter with 8 multiplexed inputs, all four I/O ports and the onboard CAN controller. While this seems like quite a list there remain more capabilities which we don't use such as the 2K Byte on-chip EEPROM or a 1K Byte on-chip XRAM. Again, we only use what we actually need.

The onboard CAN controller handles all CAN bus interaction. The CAN controller provides 15 independent message objects each of which can be configured for either transmit or receive. The firmware uses only a couple of the 15. Each type of message is allocated a receiver object. The CAN controller handles the entire CAN protocol (receiving, acknowledging, retransmission, etc.) and notifies the firmware when a message has been completely received. The firmware then copies the incoming message to RAM, performs whatever actions are indicated by the message and then tells the CAN controller that it can start listening for that type of message again.

On the transmit side firmware hands the entire message off to the CAN controller and tells it to start sending. The firmware is then later told that the transmission has completed.

We clock the T89C51CC01 at 8MHz, well below the 20 MHz maximum for the part. Instructions are executed in a minimum of a single machine cycle, which is 6 or 12 clock periods long. We run the part in what's called X2 mode that selects the 6 clock period machine cycle. The resulting machine cycle time is 750 nanoseconds. The instruction set contains a mix of 1, 2 and 4 machine cycle instructions and, by rough calculation, our code can be running anywhere from 650,000 to 1.3 million instructions per second.

For the standard and multiplex modes the firmware copies the state vector bits from each incoming message to the output ports including turning on/off the power as requested in the message. It then reads all the digital inputs, digitizes the analog inputs, places all these telemetry values into packets and initiates transmission of the telemetry packets over the CAN bus.

The act of digitizing the analog values is the only place that we use any of the special reduced power modes of the microcontroller. When the firmware samples the analog channels it follows an Atmel recommendation to instruct the microcontroller to power itself down during the conversion and reawaken once the conversion is complete. This reduces digital noise during the conversion and increases the precision of the analog readings.

In the unlikely event that our firmware stops working as expected a hardware watchdog will reset the microcontroller. Since the firmware is effectively stateless between transactions the watchdog allows the firmware to restart and continue operating with only the loss of the transaction underway at the time of the lockup. Each watchdog event resets the CPU putting all of the output lines in a default state. To speed overall system recovery the firmware keeps multiple copies of the last output state vectors received in RAM. Upon restart these copies are evaluated and if there are at least two copies that agree, these are immediately written to the output lines. This means that it is likely that the firmware can endure a watchdog restart and still be able to restore the last known output state if memory is not entirely corrupted.

In byte-pipe mode the firmware processes CAN messages for all functions other than the pipe normally. In addition, it accepts output CAN messages and transfers those bytes to the associated module electronics and simultaneously can accept bytes from the module to be sent over the CAN bus. To

prevent lockups if something goes wrong in the module electronics our firmware enforces a minimum transfer rate for the module developers to meet. We implement a timer to make sure each transfer completes in the allotted time. If this timer runs out the current transfer is abandoned. Any practical use of the byte-pipe mode will involve an end-to-end protocol between the IHU and module involved so this defensive approach makes good sense.

Even though we use hardware watchdogs and timers to watch over the system we still want to do our best to ensure that these mechanisms only come into play under extreme circumstances. To help ensure this all of our firmware is written in handcrafted assembly language because we want to know exactly what the code is doing. Before flight experienced software engineers in the AMSAT community will be invited to review our work. We fully expect this code review effort to find ways we can improve the firmware. One of the key benefits of the Open Source approach is the notion that with enough eyeballs looking at the code, all problems become visible. The high level of internal firmware documentation mentioned earlier is intended to facilitate these inspections by rapidly communicating the intent and desired effect of each routine in the code.

Finally, we use our firmware in these *widget boards* on the ground before the satellite is even assembled. The module designers are given the widget boards to be incorporated into their module, CAN bus interfaces for their PC's and software that mimics the way an IHU will interact with the modules on board the satellite. This means that all of the module implementers will be testing the behavior of these widget boards throughout the development of their components of the satellite. Any issues the module developers point out which could affect flight will be addressed in the firmware before it is flown.

### Conclusions

The project is nearing completion at the time of this writing. Prototype modules have passed functional and radiation tests. Flight firmware is nearing completion and enough PC software exists to allow functional testing of new modules and rudimentary control of the modules by satellite module builders. Still remaining are improvements to the ground software, implementation of the required software in IPS to enable IHU interaction with the modules and completion of the documentation package for satellite builders.

Several other AMSAT projects involving spacecraft of similar complexity are expected to adopt this design for their onboard command and telemetry systems, and at least one small satellite company has expressed interest.

Additional information about this project will be published on the web at: **http://www.amsat.org/amsat/projects/can-do/**

## Support AMSAT-NA!

## Bird Watchers

IT'S A HAWK! IT'S A MAN!
IT'S MIKE, N1JEZ!

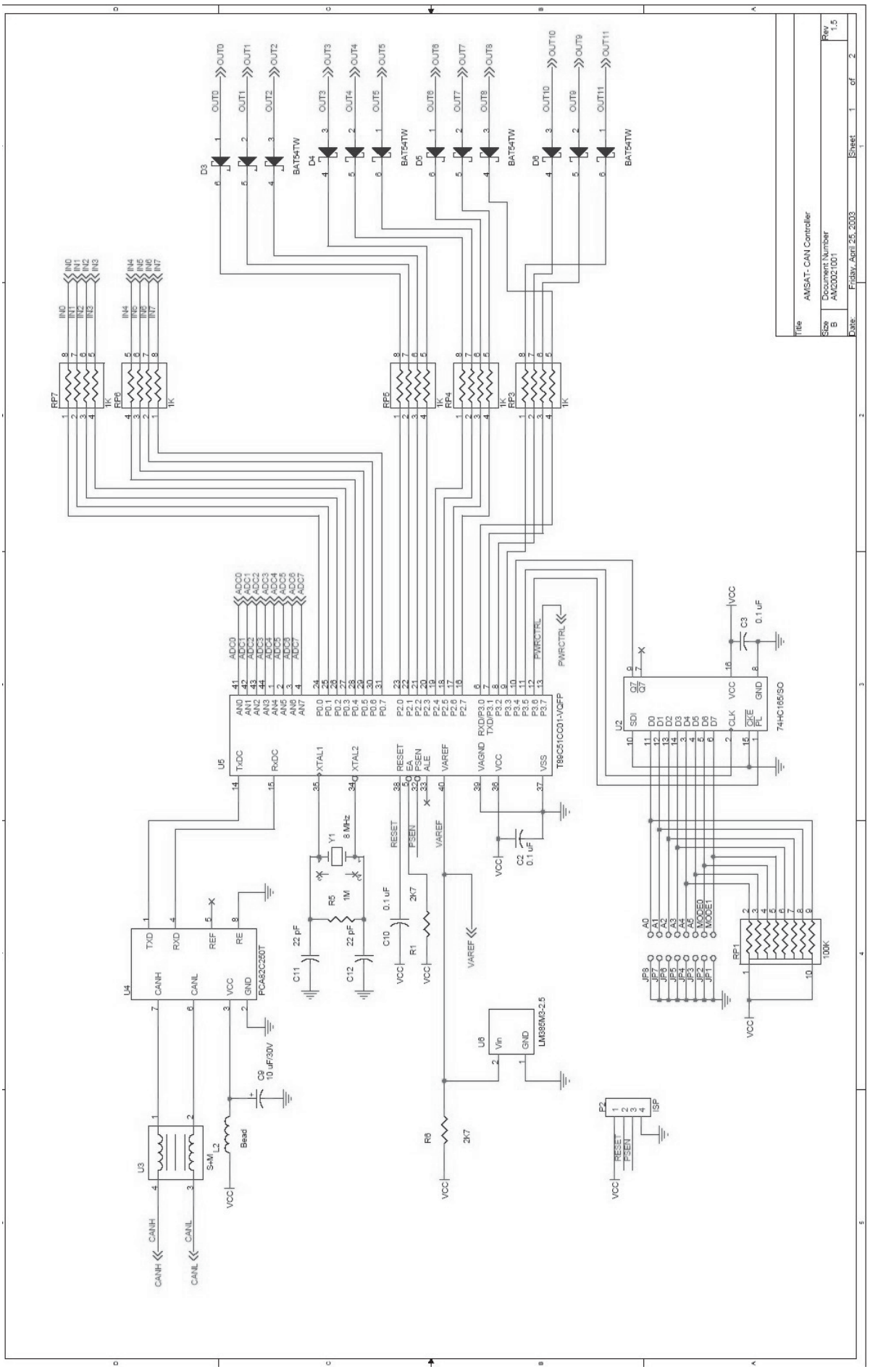
Last Sunday Dave, (N9PVF), his charming wife and I went to the top of Mt. Wachusetts, MA to watch the Hawk population migrating to the south as they do every year. We had expected to see many of them following the air currents. The hilltop was filled with many other bird watchers with their telescopes and binoculars of all different makes and models. Many of them had come from many miles away just to see this event.

I noticed one with a microwave dish jumping back and forth from his antenna and his car. I stood along side him for a moment and heard him say FN41 you are acknowledged. Somehow or other I knew this was not a bird watcher. Then I heard him say this is N1JEZ FN42BL. I froze in my tracks and said to him "Hello Mike" I am N1ORC, Arthur, and he replied I knew your voice the moment I heard you speak. Previously we had conversations on the Mt. Washington, NH repeater (146.655) and on UO-14. Mike had driven four hours from his home in Burlington, VT to reach this location to participate in the Microwave contesting that took place on Sept. 21, 2003. All I can add is that you never know whom you will meet in any part of this small world.

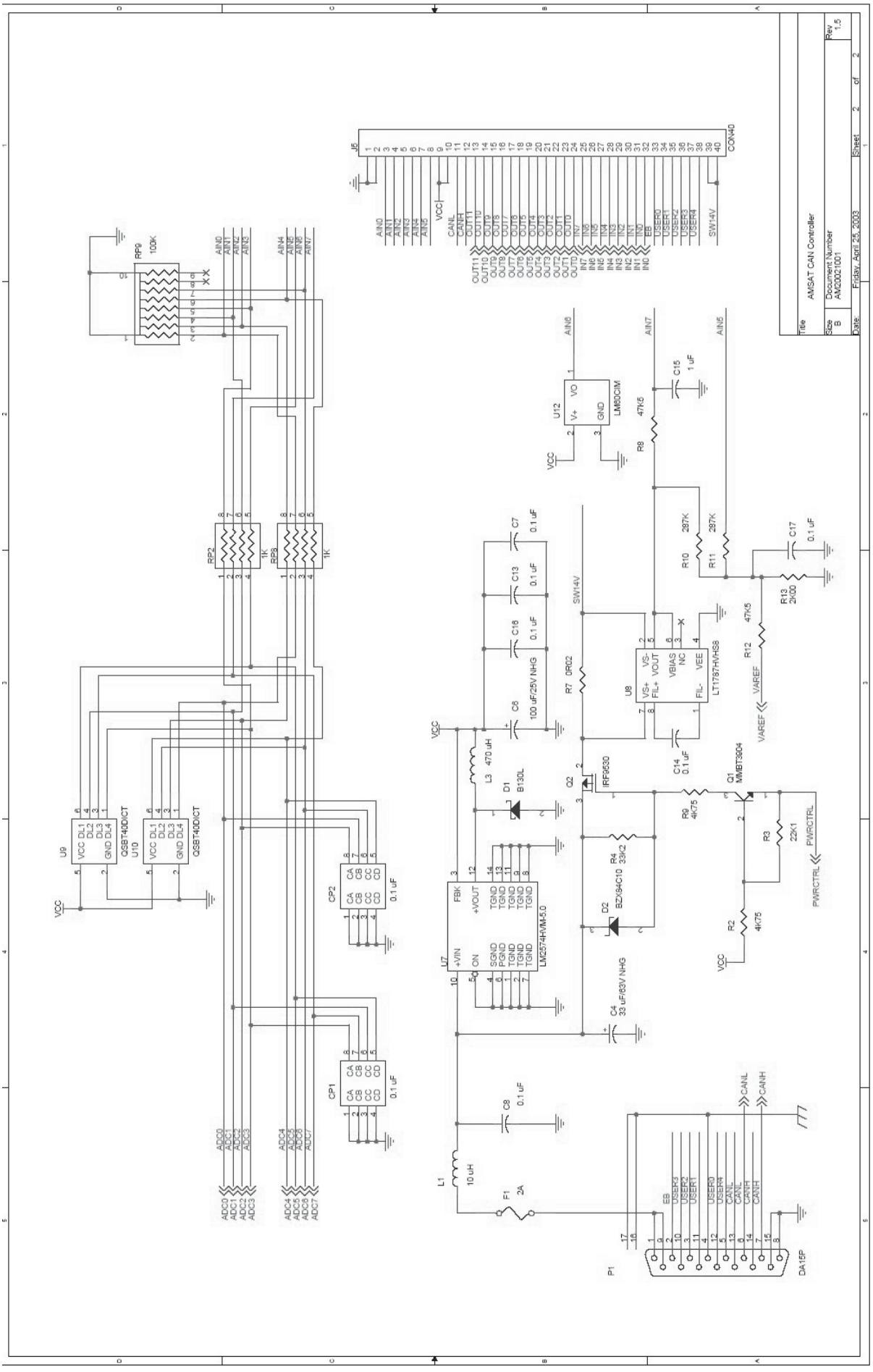
Arthur, N1ORC

P.S. I guess we were too early to see any Hawks but we did see several Sea Gulls.

(Ed. Note - See Mike's article on page 25.)




***Can-Do!***
***Processor Schematic***




***Can-Do!***
***Power Supply and Interface Schematic***