At the Repair Bench – Antenna Rotator Controller – December 2023

Back around the end of May, Al KB2AYU came to me and told me that the antenna rotator controller (Figure 1) that I had built and given to the club for use in the VHF room was "hosed". A quick look at it showed that he was correct – the display, which should have been showing azimuthal readings between zero degrees and three hundred and sixty degrees, was displaying values way up into the four- and five-digit area (Figure 2), as well as blanking out some of the letters in the word "Azimuth", which should have been displayed as well.

Figure 1 - Antenna rotator controller

This controller consists of some very basic parts… (1) an H-bridge motor driver to handle the output current to the rotator motor, including directional (polarity) control, (2) a simple five-volt voltage regulator that drops the incoming 24VDC down to the 5VDC required by the brains of the unit, and (3) an Arduino Mega2560 microprocessor board. A plain Mega2560 shield was built up to bring out the necessary pin connections as well as serve as the PCB for a low-pass filter that is a crucial part of the circuit.

Figure 2 - Erroneous display on controller

When I say "to bring out the necessary pin connections", it must be understood that there are many more +5VDC connections and ground connections made back to the Arduino than there are native pins on the Arduino to handle. As a result, a row of eight or nine pins of each type -- +5VDC and ground – are set up on the shield. In addition to the LPF already mentioned, the shield also carries the contrast control for the front panel display, which is a blue and white backlit LCD panel with 16- x 2-character capability.

The shield also carries the interface pins for the other controls on the front panel, which include directional motion switches (CW and CCW), preset, park, and speed controls, as well as the bulk of the front panel display connections. This shield board, of course, simply plugs piggy-back style onto the Arduino Mega2560.

I initially thought that maybe a quick reload of the program on the Arduino might solve the problem, in that it may have somehow gotten corrupted. Deciding to try that on the spot, I brought my laptop and a USB-A to USB-B cable into the VHF room, hooked it up to the controller, fired up the Arduino IDE software, and uploaded the program into the Arduino. No joy – the problem remained. I decided then that I would bring the controller unit home for repair, and I would swap it out with a second controller that I had already built for my own use, but was sitting idle. Originally, I was going to bring the replacement controller to the clubhouse on Tuesday evening when I went there to teach the General class (Monday was a holiday, and the Technician

class was cancelled for that evening). The way things worked out, I was able to bring the fully repaired controller back to the clubhouse on Tuesday evening instead. Here is what I found…

I began by verifying the operation of the +5VDC voltage regulator, because if that voltage was screwy, the Arduino operation would also be wonky if it worked at all. The five-volt regulator was fine, with a steady output under load of +4.996 volts. No problem there.

Now, as I have already explained in describing the controller, this thing is actually quite simple with a limited number of places for a problem to crop up. Those problems, discounting a wire connection issue, are limited to the voltage regulator, the H-bridge, the display, and the Arduino. The voltage regulator has already been cleared, and the display was evidently operational, as it displayed information—just not the correct information. Simply in an effort to be thorough, I disconnected the plug the carries control signals from the Arduino to the H-bridge, and rebooted the controller, only to find that the same condition existed. Three out of four of the possibilities have now been eliminated, leaving only the Arduino as the culpable component.

Knowing that the programming had already been uploaded to the Arduino once since this problem began, and also knowing that the upload did not resolve the issue, I was left with the base Arduino as the fault source. Now… the Arduino obviously operated, as it *was* sending data to the H-bridge and the display, even if that data was incorrect. My reasoning told me to try to recover the Arduino and restore its proper operation.

There is a piece of software embedded on the Arduino called the *bootloader* which basically sets up the operating condition of the Arduino in use. That is the only thing, apart from a possible hardware failure, that determines how the Arduino behaves. Because of the fact that this is a piece of embedded software, the possibility exists that this software could have become corrupted. I decided to erase the Arduino and to burn a new bootloader into place on the board.

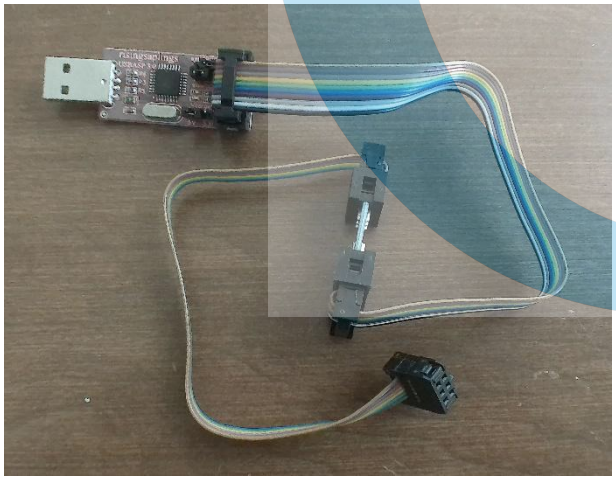

Figure 3 - USBasp programming device

Burning the bootloader can be done in a couple of ways. One way is to use a second Arduino as a programming interface. Another way is to use a dedicated programming device that is compatible with the type of Arduino at hand. In this case, with the Arduino being a Mega2560, I would be able to burn the bootloader using a device known as a USBasp (USB Atmel Serial Programmer). The USBasp (Figure 3) simply plugs into a USB port on the host PC, and then connects to the ICSP (*in circuit serial programming*) header on the Arduino board. Easily enough done, this procedure does require removal of the shield board in order to access the ICSP header, which in turn requires disconnection of all of the myriad pin connections made on the shield board. Note that the intermediate board in the data cable of the USBasp in the Figure 3 photo is an adapter interface, changing the ten-pin output of the USBasp to six pins so that it can be connected to a six-pin ICSP header such as that used in the Mega2560.

The USBasp works seamlessly, provided that two preliminary steps are taken. First is to set the on-board *Slow SCK* jumper on the USBasp to its closed (active) position. This introduces some wait states in the data stream so as to enable successful communications with the Mega2560. Second is to install the correct driver for the USBasp under your operating system. I am using Windows 10, and this step was as simple as running the *ZADIG* software and installing the driver from within *ZADIG.* Once the driver is installed, launch the Arduino IDE (if it is not already running), go to the *Tools* > *Programmer* menu item and select *USBasp* as the programmer in use. Then, with the USBasp connected to both the PC and the Arduino, go to *Tools* > *Burn Bootloader* and let the software do its job.

Figure 4 - Fully operational rotator controller

I did all of this, and then I re-uploaded the operational program to the Arduino again. Next, I completely re-assembled the controller (without its cover), including installing the shield board and re-connecting all of the pin connections. Finally, I connected up the 24VDC power supply to the controller and turned it on. *Voilá!* It worked normally, as it should have done. Note that the "damaged" digit in the Figure 4 image is a digit that was in the process of changing when the photo was snapped.

Finally, I installed the top cover, and the job was complete. I connected it up to a Yaesu rotator unit that I have on hand here, and it worked flawlessly. Thus, it was able to go back to the clubhouse that evening.

The moral of the story here is that when everything else has been eliminated, what remains must be the problem, regardless how unlikely it may seem. I would have expected the operational program to become corrupted before I would expect that to happen to the bootloader. However, the proof is in the repair, and the bootloader was certainly at fault here.

See you next month!

