


☐

I'm not robot


reCAPTCHA

Continue

The goal While working recently with AutoCap software, I realized that the weak link in the system was the LP-100 meter. This device is essential to make the adjustment software functional, as it provides the constant flow of SWR data used to quickly adjust the loop capacitor. The device works well, but I don't want to depend on it for the loop tuner kit, for a handful of reasons: Price - the LP-100A is an expensive device, and has only one source. Keeping a spare around is just not possible for most of us. It is out of the question to get one for many small stations. Closed - the LP-100A has schematics available, but the firmware is closed source. This is not a problem for a device, but I want the AutoCap system to be available as a kit, and all components to be open source hardware and software. Physics - the LP-100A is designed to be used in the operating office, and it is not very portable. I would never take the meter to a Field Day site, or anywhere it could be exposed to the elements. Since many loop users expect their antennas to be portable, it would be nice to have something compact that would travel. The LP-100A also requires an external energy source of 12V and its current circulation is not trivial when using from batteries. It would be desirable to have an SWR sensor that was energy efficient, and could take its power directly from the attached PC. Bugs - I don't mean badly the LP-100A, because it's a great counter that I like to use with my station. However, my particular unit's data output port has a few minor update bugs that sometimes prevent it from reporting the correct SWR values during an AutoCap adjustment cycle. I've been working around all the issues I know, but I always wanted something a little more predictable (and maintainable) for this particular use. The LP-100A is definitely a capable device, and it is offered as one of the sensor options in AutoCap. However, this is not the one-time solution, and this project is intended to provide a more flexible option. Getting real on the requirements What I really want is a cheap SWR counter for the loop adjustment system, which can be easily interfaced to the computer. These are actually the only two difficult requirements in the feature set: Easily interface to a reliable, fast PC, and continuously read SWR antenna Maybe more interesting is the list of features I don't need: Power readings (before or thoughtful). This counter only has to read SWR, which is calculated from the report forward power and reflected power. The first sensor device I will use below is a power counter, but the only data essential to the adjustment cycle is the relative extent of the two readings, used to calculate the SWR. The actual reading values themselves are not required to run the AutoCap algorithm. Certified accuracy. The accuracy is but not always necessary in the quantities we are used to. For this project, if the SWR reading is close enough to make good tuning decisions, that's enough. Specifically, the meter should only be 100% accurate at 1:1 SWR. If it reads correctly at 1:1, but accumulates some error as the SWR rises, that's fine for this application, because AutoCap's goal is to continuously push the SWR to 1:1. Of course, if I can make a precise instrument, so much the better. Analog counters, digital reading, LED bargraphs, etc. This is a PC-based meter, which should be used only to power AutoCap software. This software has its own reading for important things. I can add some of these unnecessary features as time goes by, but for now, the project should start as simple as possible. The hardware This project is mainly about software. For computer hardware, I'll start with the Arduino UNO R3, and much of the work will be done in its firmware. The United Nations is an easy-to-work platform, even for a first project. The only difficult requirement for the board is that it has at least two free A/D entries, and is able to communicate with the PC via the serial port or virtual port. Most Arduino products fit this description, and the software should work very well on one of them, as long as the appropriate pin definitions are used. I want this project to be flexible enough so that any number of sensors can be mated to the controller, and always work satisfactorily with no more effort than a small setting. However, any SWR counter still needs a sensor, so the first one will be the very cheap Universal SWR Bridge of kits and parts. This kit is easy to build, and each complete kit is a whopping \$9 US. At this price, I bought two to begin with, just in case I came across construction problems with any of them. This kit is rated for 10W, and has two simple voltage outputs, one for front power, and the other for reflected power. This is enough to start the project. The sensor assembly was simple, and even with my limited welding skills, two workboards were built, and one was configured with so-239 sockets and header pins for an easy connection to uno R3. Each of these sensors produces two DC output voltages. One corresponds to the forward power, and the other to the reflected power. Most RF sensors work this way. When testing the boards with a volt counter, I noticed that these two voltages are considerably less than 5V when the devices are driven in indicated power limits. This allowed me to connect the FWD and REF lines directly to the A/D entries on UN R3. In order to prevent stray RF from entering the A/D circuits, I placed a 0.05uF ceramic disc capacitor between each line and the GND pin. Since the A/D entries are very impedance and the SWR sensor outputs were intended to devices (e.g., analog counters), I also placed a 75k resistance on each of the 0.05uF caps, to drain the accumulated load when no RF was applied. One quirk I noticed with these sensors is that the screen-printing labels on the header pads seem to be reversed. The FWD voltage comes from the REV pin, and vice versa. Working around this is as simple as exchanging these two threads, but it was a little humorous bug that I found during the sensor test. I compared the diagram to the one given in a presentation on how to build homebrew SWR meters, and the kitsandparts scheme seems to be correct except for mislabeled outputs. You can watch this presentation by following the link to the na0tc website, and check if I'm right. Anyway, the exchange of wires made my sensor work properly. A summary of the material used includes: Resistances can be any appropriate value, just like plugs. I selected these values from the material I had available. The software It should be possible to accomplish everything else within the Arduino itself. There is only one essential function of the meter, which is to read and calculate the SWR, and report it to the PC through a virtual series port. The PC interface uses the standard modem interface of the CW project, and contains a small set of basic commands: SWR - reads and returns the current SWR as a floating comma number. RAW - reads forward and reflected power readings in the form of raw whole values, in a pair delineated by comma. PWR - reads forward and reflected power readings in the form of scale power values (watt), in a pair delineated by comma. It's not necessary for AutoCap, but it's a nice touch if the meter is anywhere near calibrated to actual power levels. ALL - returns a triple delineated by comma, containing SWR, forward power and reflected power. This is the command currently used to probe the meter in AutoCap. Power readings are not used unless the minimum listening power in AutoCap is set to anything other than zero. AUTO - allows automatic swr sounding at a user-specified interval; it is an alternative to active PC surveys. When AUTO is sent without argument, the current automatic sounding interval (in msec) is returned. If an argument is provided, the automatic vote is configured to use the interval provided. A zero interval disables the feature. ECHO - allows the echo of the characters received by the device, to allow easy use of the device from a terminal; it's mainly for the calibration. All orders are sent using standard hash and semicolon (;) order packages. For example, to read the current SWR, the PC will send: #SWR; To which the software could respond: #SWR-1.35; This command itself is enough to implement the SWR sensor function for use with AutoCap, so my next task is to integrate the sensor with this software. However, there will be more features added later, mostly just for fun. In the end, I chose to integrate the ALL command instead, so that if the power meter provide power readings, they can be used with the optional Min. Power Setting in AutoCap. If you integrate a sensor that only provides SWR, you can leave this set setting to zero and return zeroes in both ALL power-playing slots commands. Again, the only reading required of the controller is the SWR. The control interface described above is by no means special for this particular sensor and firmware. Using this interface, any type of USB sensor or series could be integrated into the AutoCap PC software, without having to change any of the PC side software itself. This is similar to the number of rotors emulated the old Yaesu protocol, even if they have nothing in common with the Yaesu rotors. By providing documentation and a reference implementation, sensor designers can add any SWR sensor they want to the system, and it will work as well as the one described here. The modem interface is not strictly necessary here, as there is no interlocking of data between commands and data. Everything here is commands and answers, but it seemed like a solid protocol that worked well for the CW board, so I used it here. LED Fun I said that visual output was not necessary for this project, but let's face it, half the fun of working with built-in computers is the chance to attach LEDs, right? And certainly no Arduino project is complete without an LED part, so I thought I'd go ahead and add the red/yellow/green bargraph SWR. Each LED is addressable individually, providing a 11-step visual SWR counter. The images above show only a few states. The code to generate the SWR bar is general, allowing bar charts of any size, as long as the controller has enough free digital output lines. Thus, the bars of three, five or eight are just as possible as the one shown here. Larger bars can be made on the Mega. Each LED has its own SWR threshold for lighting. The variable trimmer resistances you see on the breadboard in the images above have been used to provide a variable voltage source to simulate fwd and rev voltages for testing. This turned out to be a great help in debugging, since I don't have to use my transmitter in order to test and debug the software. At the same time, the use of this debugging tool allows me to simulate exactly the conditions that A/D inputs will see during normal operation, a number of different SWR scenarios. In addition to bar charts, adding digital displays is even easier. I picked up a small number of SparkFun 7-Segment series displays for demonstration purposes. These screens are standard 7-segment LED blocks, mounted on a backpack board. The backpack panel contains a microcontroller of its own, which drives the LEDs, and is driven by the The backpack has several options for the interface at the Arduino, including SPI, I2C, and TTL-level asynchronous (essentially RS-232 at TTL voltage levels from zero to 5V). The firmware ino file (see Downloads below) has a feature called UpdateDisplays() where the custom display code can be placed. I added a few lines to write the SWR on the screen via the standard port, and keep it up to date. Fun View The latest version of the firmware also supports an LCD or OLED screen, which can be used to display forward and reflected power, and the calculated SWR. Any screen that implements the basic Arduino LiquidCrystal interface can be used. An example of an OLED display output is shown below. The OLED looks really nice and would make a great permanent meter display. However, I discovered that powering the OLED from the Arduino's 5V output pin added considerable noise to A/D readings. A production circuit should probably use a dedicated 5V regulator to power the OLED, so the Arduino's voltage power supply is left stable enough to take good A/D readings, especially when using readings for automated purposes, as with AutoCap. The customization possibilities are endless, but the idea is that you can use the SWR firmware as is, or add small amounts of custom code to drive your own screens, in addition to the existing USB interface on the PC. The main objective of the project was to have the USB interface on the PC, so that AutoCap could read the SWR from a homebrew counter. That said, many flashing lights add a lot of fun to any homebrew project. They also provide a certain amount of instant visual feedback, which can be useful for diagnostic purposes. Treatment A/D The current code uses an exponential fluid on forward and reflected readings taken from A/D. As a result, the SWR value calculated from these readings is also smoothed. The alpha value can be set independently for front and reverse readings, and updated values can be saved in EEPROM, so they will be recharged when the device is restarted. Alpha values can be read with: #ALPHAFWD; ... for the alpha forward, and... #ALPHAREF; ... to read alpha reflected. To set alpha values, use: #ALPHAFWD-value; ... for the alpha forward, and... #ALPHAREF-value; ... to write thoughtful alpha. Alpha values should be between 0.01 and 1.0. A high alpha value does very little smoothing, and produces the most responsive readings. The lower values of the alpha will smooth the readings more, but response will be a little slower. If eeprom persistence is enabled in the main .ino file, alpha values will be maintained each time they are changed using the above commands. After first installing the firmware, the EEPROM must then be prepared by sending a single handwriting command: #EWRITE; This format EEPROM for use with the firmware of the SWR counter. If this step is omitted, the meter will still run, but it will not update the EEPROM when the settings are changed. Once this command is managed once, further updates will be automatic, each time either alpha value changes. Coupler Alternatives The firmware can be modified or extended to just about any type of directional torquer that generates two direct current output voltages, one for forward power, and the other for reflected power; or a sensor that has a single direct current output voltage corresponding to actual SWR. If you want to build a meter with more power than the one shown here, you will need a higher power torquer. A starting point may be a device like the Elecraft CP-1. This device boasts a range of up to 250W, and has two outputs. You'll need some sort of grind and buffer to attach this to an Arduino, to generate the zero to five volts (but no more) DC needed by the A/D controller. So some design work is needed to interface a unit like this. Despite this, the list of homebrew and kit options is probably endless. All Arduino products based on 8-bit ATmel processors will have a ten-bit A/D input resolution. This gives you 1024 possible steps of measuring voltage from zero to full scale, or about 0.1% of the full scale per step. A meter that reads 5V to 1000W power will have a resolution of about 1W. A meter that reads 5V to 100W power will have a resolution of about 100mW. Keep this in mind when setting the large-scale voltage of your transducer. AutoCap Integration Integration in AutoCap was simple, and the software now proposes to use any working sensor based on this kit. A note using this project with AutoCap: the AutoCap firmware for Arduino and the SWR counter for Arduino are two separate sketches. This means that everyone needs their own controller. Some people have written to me asking me to merge the sketches on a single board, but since both sketches perform tasks in real time, they really don't have a place on the same microcontroller. This is especially true when using the AutoCap firmware to drive a stepper engine. When tuning, each sketch should be as responsive as possible, and having independent microcontrollers is the easiest way to ensure this. It should be possible to merge tasks on a quick microcontroller, like one of the new Teensy boards, but I don't need that, and I haven't heard enough people to do the work for others. :) told me (complained) that two Arduino boards is nearly \$50, even without the torque or engine controller components. This is true if you use Arduino.cc UNO-branded panels. But there are all kinds of very high quality panels that can be used for this project, including some that are designed to be integrated into larger projects. Projects. are some of my favorites for people who want to try this project in a cost conscious way. SainSmart UNO - currently \$10 in singles. This board is essentially a clone of the Arduino UNO, with the same processor and USB connector. Sparkfun Arduino Pro - currently \$15 in singles. This board is designed for integration into other projects, and the host port is a TTL series port. To connect this to the computer, you'll want either an RS-232 level changer or an FTDI converter. Sparkfun Arduino Pro Mini - currently \$10 in singles. This is an even smaller version of the Arduino Pro, but functionally identical, with the same host port connection options. This table even provides two additional A/D pins than any of any of the other UN replacement options. Sparkfun RedBoard - currently \$20 in singles. This card is an improved clone of the Arduino UNO, with a mini USB connector and a SMT processor and pretty LEDs. The host connection uses an FTDI chip, which is almost universally supported on all popular operating systems. SparkFun Arduino Pro Micro - currently \$20 in singles. This table is similar to the Pro Mini, but it includes a micro USB connection for the host connection. There are many other inexpended options, such as ARDuino UNO is one of the most cloned hobby tips. Just search the internet for Arduino clone and you will find all kinds of tips that should work just fine. Even Teensy's advice should be able to work for the Firmware AutoCap or SWR Meter. With a little shopping, it should be easy enough to get two controller tips to match your needs for about \$20. Conclusions A simple, inexpensive and usb SWR counter is certainly possible and practical using open-source hardware and designs. This project is underway, so there will be more updates to come. The main goal has been achieved, which is that the entire AutoCap project is now Open Source Software, and Open Source Hardware. Software Downloads 2016-09-20 - Alpha values are now divided into forward and reflected values, and can be set via host commands. The persistence of the settings has also been added. 2016-09-18 - The software has been substantially updated to add LED/LCD display support, and to make the configuration more consistent. In addition, some bugs were found in the SWR reading and calculation logic that were fixed. 2014-03-30 - Current software is available for download. The firmware is only available in source form, and can be read by Arduino software. AutoCap Links Version 1 The main page of the project Kits and parts - Open-source material for ham projects. NA0TC - Homebrew SWR Notes Arduino - Open-source hardware and embedded development tools. SparkFun - Supplier of arduino boards and equipment. AdaFruit - Another good source for Arduino material. Material. Material.

normal_5f8e838fcd170.pdf
normal_5f98b4ae59b16.pdf
normal_5f963e008ec35.pdf
adapted interactive reader.pdf
cook up a storm full movie in english dubbed watch online
john deere 214 manual.pdf
letter_punch_holder
building bye laws pdf download
bacteria salmonella typhi.pdf
simple and sinister kettlebell workout
unir dos.pdf mac
analytical chemistry 2.pdf
hemolytic anemia guidelines.pdf
arm assembly language programming architecture volume 1.pdf
alavancagem financeira.pdf
bach cello suite 1 gigue sheet music
9653445.pdf
8562138.pdf