Michael L. Foerster, WØIH

1403 Foxcroft Place, Winona, MN 55987-4845; **w0ih@arrl.net**

Building an LDMOS Amplifier with an Arduino Interface

Use these concepts to assemble an Arduino controlled 160 m - 6 m LDMOS amplifier.

We are each given different talents and skills from our parents, schools that we attended, job experiences as well as life experiences. For me, I learned carpentry and construction from my father, electronics in school and a host of technical know-how as a field service technician, even some programming exposure through my jobs as a software test engineer, and a great deal of RF background through my 50+ years in Amateur Radio. From these life experiences, I chose to embark on one of the biggest projects I had ever tried; I built a Laterally Diffused Metal Oxide Semiconductor (LDMOS) amplifier with an Arduino controller (**Figure 1**) to interface between the amplifier and my radios: the Elecraft K3S and KX3.

This article does not present the "end all" to building amplifiers, but rather just gives a few ideas to build from or perhaps get the reader to contemplate other solutions for the problems that I faced. Consider building an amplifier yourself, but if nothing else, enjoy reading the article.

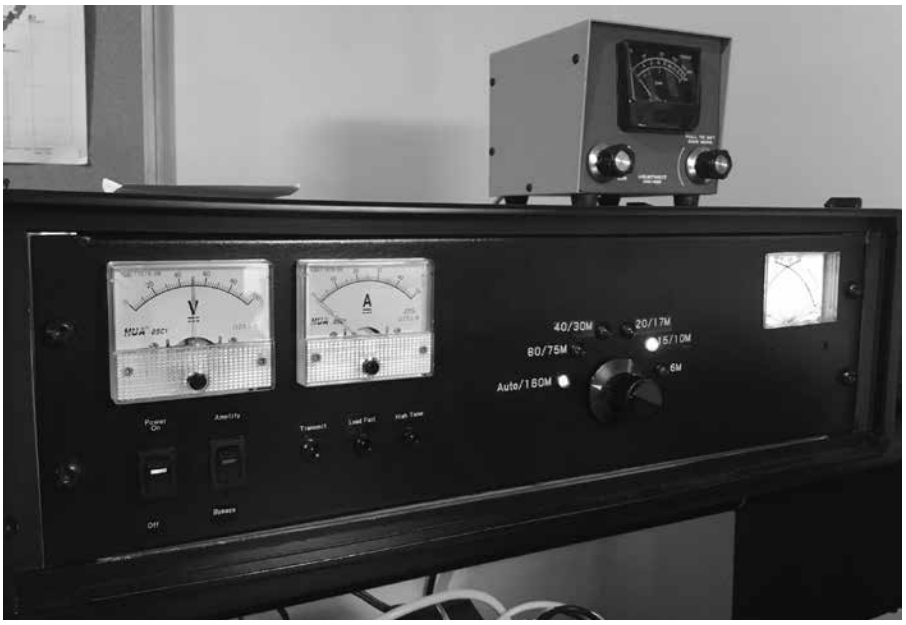


Figure 1 — Front panel of the amplifier.

Desired Functionality

I wanted the Arduino interface to power up the amplifier and monitor the operating band of the radio to switch the amplifier low pass filter (LPF) band switch. I also wanted to configure the radio output power on a per-band basis and have the Arduino load the power setting into the radio for power output and tune power parameters. Monitoring and displaying the band, voltage, forward and reflected transmit power, temperature and input voltage seemed a very natural part of the project as well. Some other functions such as the timeout timer and Bypass/Operate selections were enhancements to the system.

In developing the program in the Arduino, I added a lot of functionality that is not strictly necessary. You can remove these to pare the system down to its functional simplicity.

Amplifier Project Choices

I studied different available LDMOS designs and decided to use a single solid state power amplifier (SSPA) device that provides over 1000 W output with less than 5 W of drive power. I didn't feel the extra 400 to 500 W would be worth the added cost.

I've learned over the years that I tend to forget some details of operating high-powered equipment, and that this can be costly since some of these devices can be very unforgiving! I felt it was very important to build in automatic band switching. There are boards available to read the BCD band outputs from rigs like my K3S, but I also want to use this amplifier with my KX3.

Amplifier Development

I considered the different kits and parts that were available online for the LDMOS amplifiers and settled on the kit parts from Jim Klitzing, W6PQL **[1]**. I purchased most of the boards in kit form, rather than pre-assembled. I ordered the SSPA, LPF, control board, and two of the dual direction detectors (SWR boards) as kits. I also purchased as complete assemblies, the input and output relay boards as well as a dc FET switch that turns on the power to the SSPA with a signal from the control board. The instruction page also lists some of the other required parts for the amplifier that can be purchased from **mouser.com**, **eBay.com** or other suppliers. I purchased the BLF188XR LDMOS chip already soldered in place on the copper heat spreader, as I felt this was a critical process best left to the experienced vendor. I built the rest of the major assembly kits using the instructions provided.

The biggest challenge in the kit building phase was the control board, because it used many small surface-mount parts. This was my first experience in soldering surface-mount devices. I learned to first add a touch of solder to one pad, place the part with tweezers and heat it up, then solder the second side and re-solder the first side. I checked and double-checked my parts placement, especially on the control board. Jim Klitzing, W6PQL, was a great help in getting through a few rough spots on the kit builds. The control board didn't work due to a few poor solder joints on some of the surface mount op-amps. I later learned a touch of liquid flux makes the solder flow much better on these delicate leads. Looking back, I should have purchased the control board pre-assembled.

One of the most valuable tools I have is a jeweler's eyepiece with a side LED light for close inspection of the parts placement and solder joints. A microscope with a USB interface might be very useful for this type of work. My solder station, with interchangeable solder tips and adjustable temperature, proved to work very well throughout this project. It worked for the small surface-mount parts as well as soldering the heavy #10 AWG dc wires to the SSPA assembly.

The design of the case took a lot of thought and time. **Figure 2** shows the completed rear panel. I had a 3U (5 ¼" high) 19" rack-mount cabinet in my junk box that included some great RF-proof finger stock on the top and bottom covers. This proved to be perfect for the case. I purchased an extra piece of 1/8" thick aluminum plate for the bottom of the case to mount the heat sink, resistors and other parts. This bottom aluminum plate was mounted to the back plate to allow removal from the case shell for maintenance. I purchased a 4-40 tap and matching drill, which I used extensively in mounting resistors, boards, standoffs and other parts on this plate and various places within my project.

I also tried to make sure I could mount the boards using connectors to allow for easy removal for future maintenance. This proved to be very useful, especially for the LPF and front panel. I considered putting BNC connectors for the RF input and output lines on the LPF board, but to remove them required unsoldering only the center wire and removing the coax shield connection screws. I built the front panel with a 20-pin connector for the bulk of the connections, along with two 5.5 mm connectors — reversed to prevent incorrect installation — for the two meter connections. Throughout the process

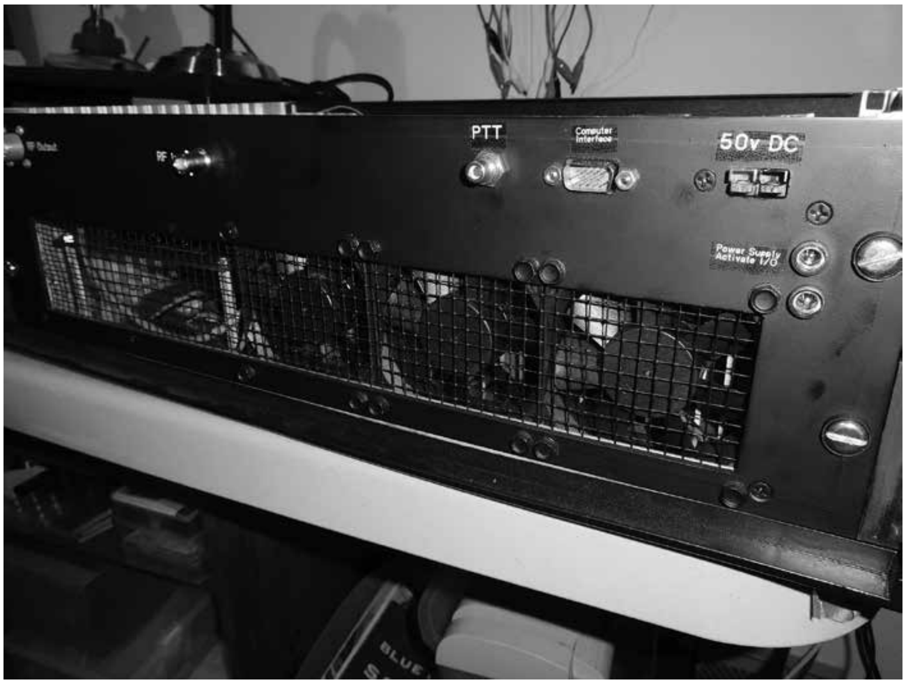


Figure 2 — Amplifier rear panel.

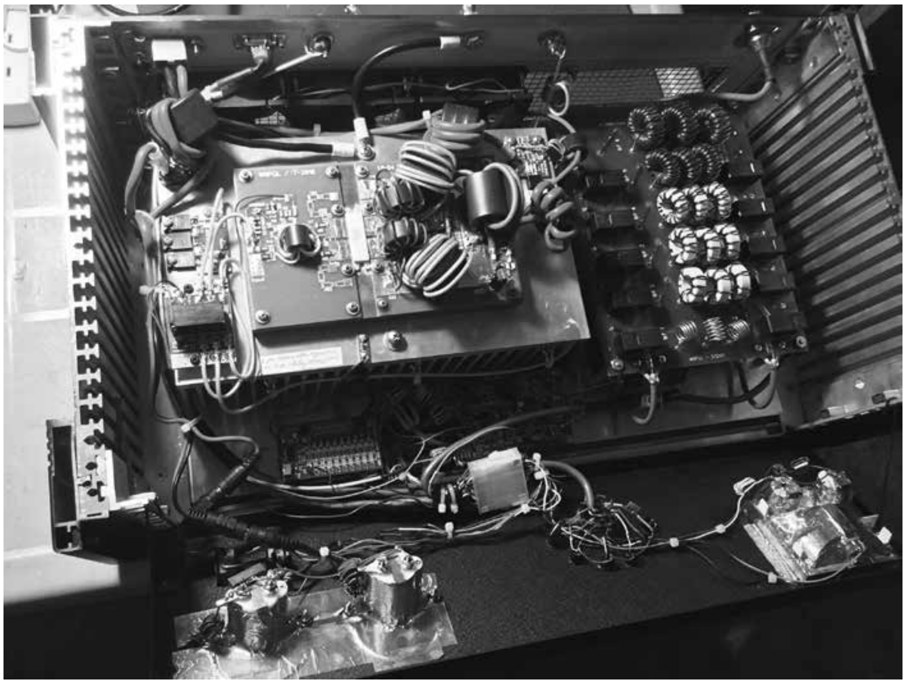


Figure 3 — Amplifier with front panel tilted down.

of building, troubleshooting and modifying, these parts required removal several times. I also needed the ability to check voltages on the front panel, so I bent two pieces of stiff wire to allow the front panel to tilt down at about a 45° angle (see **Figure 3**). Because many of the control board connections were spread throughout the board, I didn't use a connector on it. So far, I've only had to remove the board once. I had to mark each of the colored wires carefully to make sure that I could re-install them correctly.

Most of the wiring was direct point-to-point where I added small ferrite cores wound with about 7 turns of wire. The cable to connect the Arduino to a 15-pin VGA-type connector was made of two shielded 8-conductor cables.

I tried to use color-coded wires throughout the build of the amplifier and subsequent Arduino project. I didn't purchase all of the colored wires separately, but rather used some different cables with 8 or even 12 colored wires that I had in my junk box. I didn't follow any strict color coding, but I did mark the colors used on all my schematics. This will help for future troubleshooting or modifications.

I learned quite a few things from Jim, W6PQL, as I progressed through the amplifier build. For one thing, I had the fans butted up against the heat sink. He helped me to understand that you need to allow at least ¾" of space between the fan and the heat sink, or the fan output will be decreased greatly. He also recommended using conductive tape to cover the meters. I found some fairly inexpensive 2" copper tape that molded over the meters very well. Aluminum tape should work equally well.

Individual ferrite cores should be used on each signal lead at the source and destination board throughout the amplifier to reduce RF pickup. This is very important in an environment where there is a lot of RF signal. I inquired about passing two or more wires through a single core, but Jim told me that two wires make a great transformer. [Wire pairs carrying dc currents should pass through ferrite cores as plus and minus pairs to avoid magnetizing the ferrites. — *Ed.*]

Although I've used inexpensive relays before for high power RF control — specifically for remote antenna switching — Jim's articles detail the testing and use of these relays for high RF power, including compensation for VHF and UHF installations. The relays, seen in **Figure 4**, are used for power switching for the input and output as well as for the LPF.

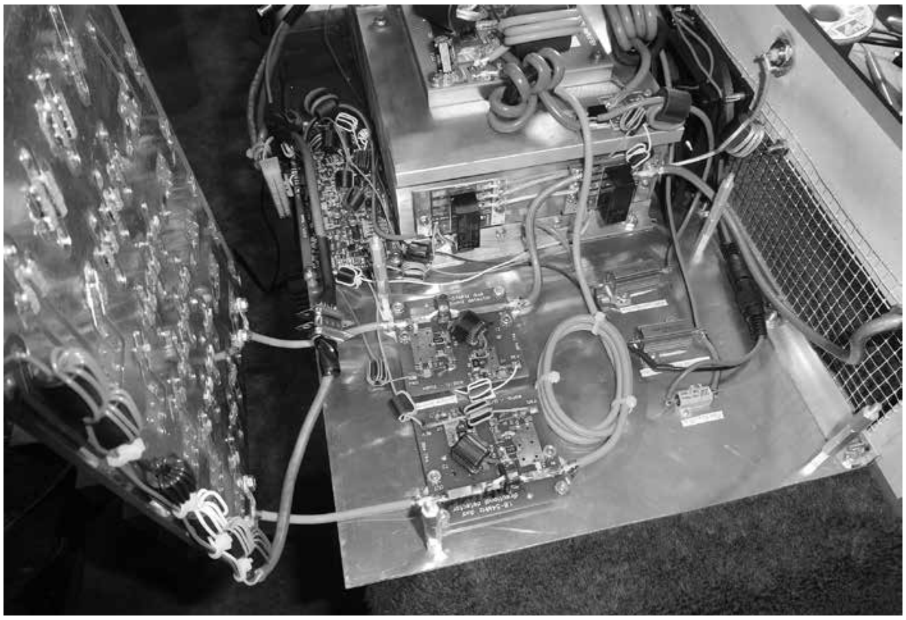


Figure 4 — Relay and SWR boards.

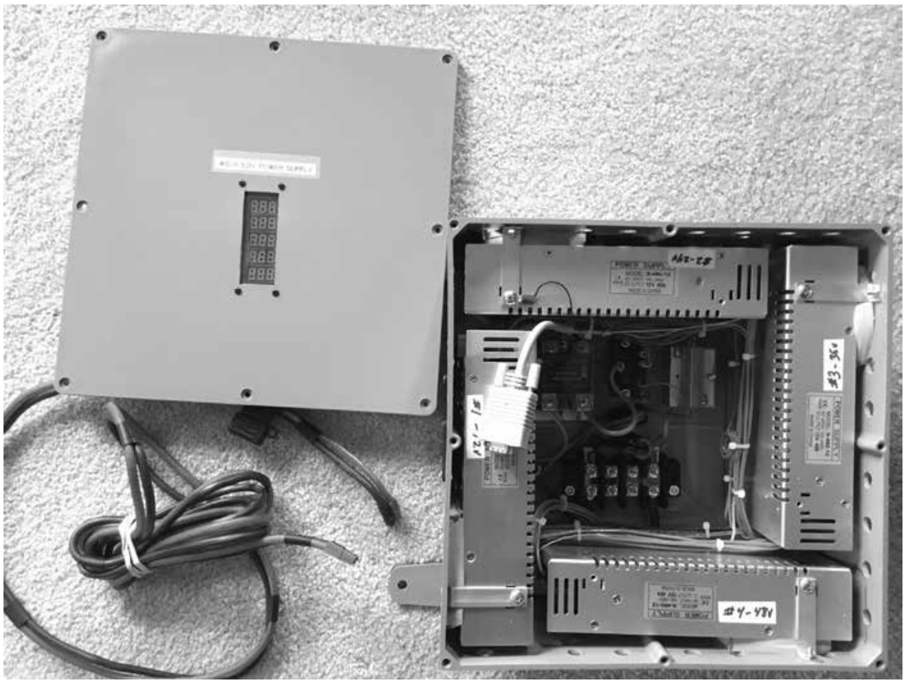


Figure 5 — Layout of the power supply.

Amplifier Power Supply

Many other builders of LDMOS amplifiers favor surplus server-type power supplies that can supply 60 A at 50 to 60 V. These can be very noisy — described as a jet taking off — but the noise can be subdued with modifications. I was aware of these, but decided to build my own supply. I found that four of the 12-V, 40-A supplies designed for LED lighting can be connected in series, and appeared to be a very good fit. The 40-A versions sell for $25 each on eBay and are available up to 60 A. I mounted some large ferrite cores in both the power supply and the amplifier end of the dc supply cable to reduce any possible RFI. I have had no noise issues.

I purchased a large 12" by 12" by 6" plastic electrical box from the local big-box hardware store for about $30. This allowed mounting the four power supplies in a square configuration (**Figure 5**). One was set slightly higher than the rest to allow mounting the ac entrance connector below it. I drilled holes to allow for air flow into the box, as well as for the supply voltage adjust screws and the 120/240 input voltage switches on the bottom. Additionally, I drilled four large 2" holes for the supply exhaust fans and used some thick foam to force the air flow outside the box. The solid state relay (SSR) for the ac switch was mounted in the center along with the soft-start resistor and bypass relay. Five digital voltmeters on top of the case display the voltage across each supply, plus the total voltage. Each supply was setup for 12.5 V to provide a total 50-V output.

The mounting holes were drilled to match the supply mounting pattern. The supplies were wired for ac and dc, then dropped into the box and screwed into place. I can run the power supplies on 120 V ac, but the system runs better on 240 V ac so the shack lights won't flicker.

Initial Amplifier Operation

I discovered in some of my early testing on one band that the amplifier would oscillate; the power output would cycle rapidly from zero to full power. In troubleshooting I found I had the cables behind the amplifier poorly dressed, and somewhat randomly looped around. The amplifier has a great deal of gain, and if the input cable gets too much feedback from the output coax, it can affect the amplifier in drastic ways. I now keep the input cable well separated from the output, and use double-shielded Teflon RG-142U coax for better isolation. I also added a heavy bolt with a wing nut to the back of the case to ground the heat sink and chassis internally for a connection to my station ground.

Nearing completion, I got anxious and started testing the power output of the amplifier into a dummy load, but I had not finished checking out the protection circuits. At one point, I transmitted into the amplifier with the band switch set incorrectly. I immediately noticed the power not coming up and shut down within a half second. This short burst, however, blew out the LDMOS chip. In discussing the loss with Jim, he suggested that to check out the protection circuit, simply set the band switch down one band, key the microphone and snap your fingers. The system should go into protection mode (SWR FAIL) instantly. I requested, and Jim added something to his protection circuit description to help others. He added a test using a 1.5 V battery to the Safety Sensor output line (no transmit required) that must put the amplifier control board into protection mode, shutting down the FET dc power switch and turning on the SWR Fail LED. In troubleshooting, I found I had the SWR Fail LED installed backwards, preventing the circuit from working.

The fan switch circuit control board is designed to turn the fans on during transmit. When the heat sink reaches about 105 °F, the fans are turned on constantly, even during the receive cycle, and turn off at about 100 °F. I found that during a contact, the temperature seemed to stay around 100 °F and didn't go below that, because the fans were off. The fans also seemed to come on and stay on quite frequently. I made a minor modification by adding a 470 Ω resistor across the fan switching FET to ground that turned on the fans all the time, but at a slow quiet speed. During the receiving cycle, the temperature of the heat sink will now continue to drop. During transmit, the added resistor is shorted out by the existing FET switch, and the fans come on at the normal speed.

Solid State Relay Circuit

Rather than run the ac power through the amplifier to the switch that turns on the power supply, I used a 240 V ac solid state relay (SSR) in the power supply. This can be enabled by a 5 V dc signal sourced from the Arduino. Most SSRs are basically ac SPST switches that can be enabled with a dc voltage between 3 and 12 V dc at a few milliamps. I've used SSRs fairly liberally around my shack.

Input Power Attenuator

In experimenting with the communication commands, I discovered I could change the output power from either the K3S or KX3 radio in 1-W increments only, even though manual control allowed the power to be incremented in tenths of a watt. Considering the amplifier required around 2-W of drive, this wasn't enough resolution. A 6 dB attenuator would allow the system to increment the power by 0.25 W and 10 W would provide 2.5 W into the amplifier. The 6-meter band typically requires 5 to 6 W of drive. I therefore set up the 6 dB attenuator with a relay input so when energized from the 6-m LPF switch, it would bypass the attenuator. The attenuator also ensured that I would not overdrive the amplifier input from the KX3. For the K3S, I found the commands to put the internal KPA3 — the 100 W amplifier built into the rig — in bypass mode, ensuring that it could not overdrive the amplifier input.

A note of caution: I have learned that some radios when set for lower power, have been reported to transmit 100 W or more on the first "dit" or transmitted syllable. Transmitting like this, even a short signal can be devastating to a solid state amplifier. Before testing with your specific radio, you might want to investigate and perhaps even test your radio at low power, by using a peak reading wattmeter or an oscilloscope. In investigating this phenomenon, I found an article **[2]** by Phil Salas, AD5X, on using a gas discharge tube to limit the overshoot to the amplifier.

Arduino Development

There are quite a number of different Arduino boards available, each with different capabilities. Because my plans were to use more than one communication port, I settled on the 4-port Arduino MEGA. Initially I am using two communication ports, either of which can connect to the radio. I also started with a 2-line, 16-character display from **Adafruit.com** that includes five push buttons. This shield — a shield is a board connected to an Arduino — requires only two wires using the I2C bus for communicating with the display. There are other similar displays and keyboard combinations available but they require the use of many more pins. I discovered I could mount the Arduino very close to the amplifier and use a longer 4-conductor shielded cable for 5 V power, ground, and SCL and SDA for the I2C wires, to remotely mount the display and keys. I used an old PS2 mouse cable to allow disconnecting the display box. In researching the I2C communications bus, I found that the distance between the Arduino and the display could be lengthened by adding 2.2 kΩ pull-up resistors to the SCL and SDA output lines for the I2C bus.

Power for the Arduino is provided by a small 9-V wall wart supply. I chose 9 V to limit the voltage drop across the Arduino internal voltage regulator. The current draw of about 110 mA does not create a temperature problem. Do not to use the small variable switching power regulators since they generate a considerable amount of RF noise.

Arduino Mounting Cases

Figure 6 shows the case for my display. I cut an opening for the display, and drilled holes centered above the 6 key buttons, and mounted some rubber push buttons cut from an old TV remote control. This gives the keys a very comfortable, tactile feel.

I found a metal case to mount the Arduino MEGA. The case was a bit tight (**Figure 7**) once I mounted the board with the circuitry, beeper, serial interface and internal wiring, but it was quite manageable. The boards are removable for maintenance.

If you decide to use a non-conductive case, make sure you line the inside with copper or aluminum tape and ground it to the system. This helps prevent noise from the Arduino getting into your receiver.

I was concerned about heat buildup in the box that houses the Arduino, so I mounted a very small — about 1" diameter — 5 V fan on the back of the case, and drilled exhaust holes in the top and bottom of the front. I also added a 10 kΩ thermistor to measure the internal case temperature, see **Figure 8**. An analog input measures the voltage across the thermistor for a temperature measurement. I set a pulse width modulation (PWM) output to drive the fan at different speeds. The fan, it turns out, was unnecessary. The only time it has come on was during my testing phase, when I gently warmed up the case with a heat gun.

I also added a line from the thermistor inside the amplifier to another analog input on the Arduino, and put a 20 kΩ resistor in series with the line to the Arduino analog input to keep from loading the thermistor circuit. The added resistor would not affect the readings appreciably, since the input impedance on the Arduino analog input is quite high. I was able to use the same software routines to read the thermistor for the Arduino temperature and the amplifier heat sink because both were setup with the thermistor connected to ground. I used an external digital thermometer to calibrate the thermistor readings.

Arduino I/O Circuits

Never put more than 5 V on any of the Arduino input or output pins. I used an opto-isolator for each of the LPF band switch relays, along with an NPN transistor to ensure I had enough current to drive the pair of relays on the LPF board, see **Figure 9**. Similarly, I used opto-isolators to drive the 12 V outputs from the amplifier to the Arduino inputs for the *Transmit*, *SWR Failure*, and *Over-Temperature* indications. During my design phase, this seemed like a good safety measure, but after re-evaluating the system, I didn't end up with full isolation between the amplifier and the Arduino. This led me to believe many of these opto-isolators were not necessary. The only exception would be turning on the 12 V to the *Bypass* switch, where the pull-up resistor to the PNP resistor would put 12 V on the Arduino pin when not enabled.

The actual amplifier front panel switches are left in the unpowered positions, the band-switch in the *Auto/160 m*, power switch to *Off* and Operate/Bypass switch in *Bypass* position. This allows the Arduino to control these functions, otherwise the system can also be run manually if the Arduino is disconnected.

There are essentially four different types of circuits used to interface between the amplifier and the Arduino. All of these circuits must limit the voltage to 5 V on any of the Arduino pins.

The LPF circuit board has five 12 V relays to enable the six band filter segments. The 160-meter segment is enabled whenever none of the other segments is enabled. The common side for all relays is tied to +12 V and the negative side of each relay must be switched to ground to enable the band segments. The relay must have a snubber

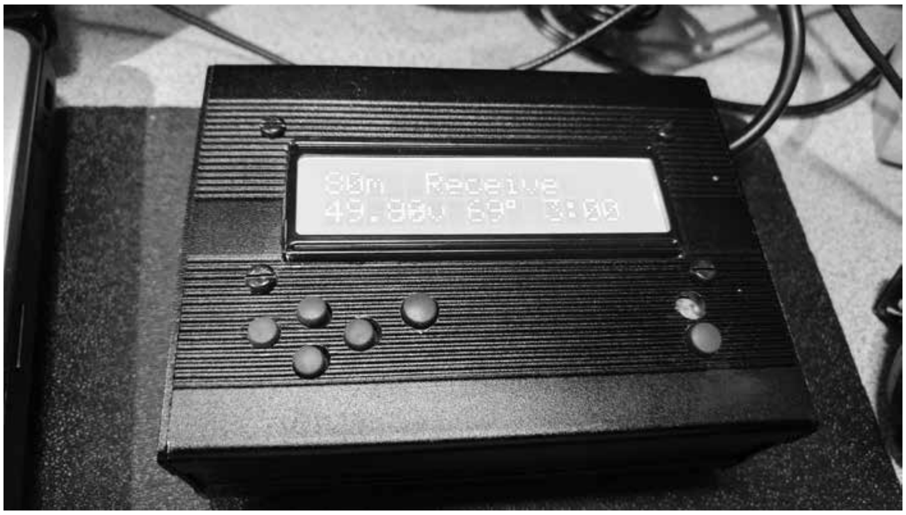


Figure 6 — The display housing.

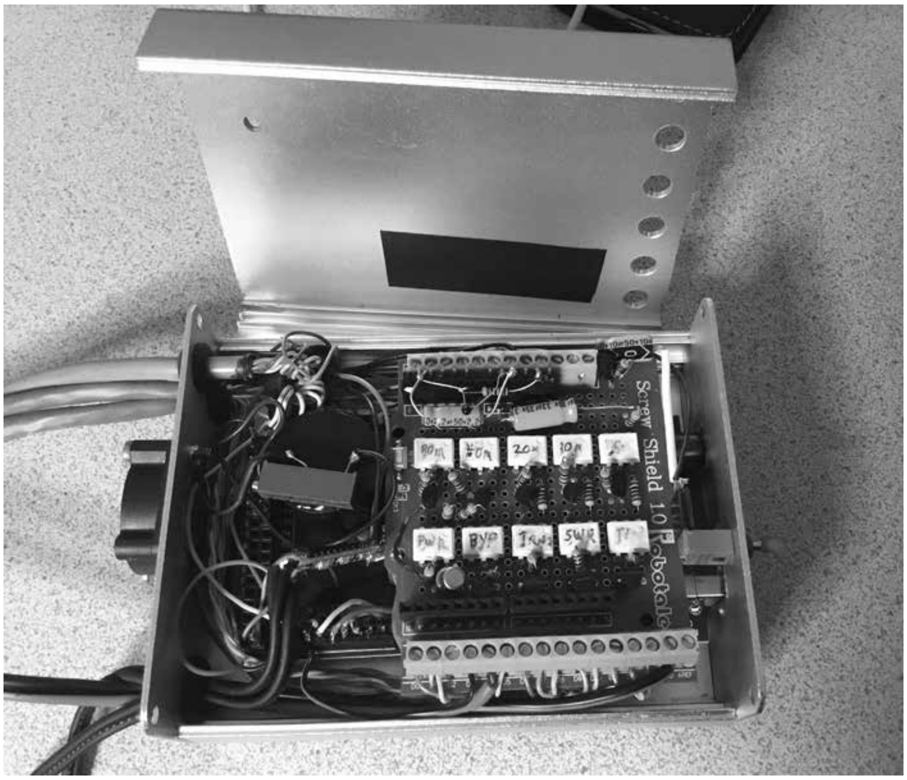


Figure 7 — Internal view of the Arduino housing.

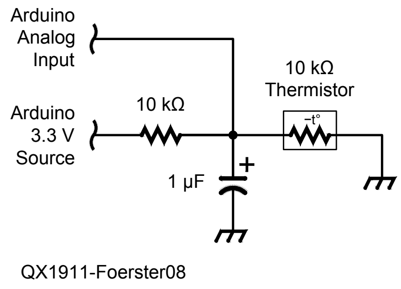


Figure 8 — Thermistor circuit.

diode across it (seen in Figure 9) to prevent damage to the transistor. This is a normal part of the relay circuit on the LPF board for each of the five relays. Because only one relay will be enabled at a time, all five relay driver circuits can share a single pull-up resistor.

Any digital inputs (**Figure 10**) from the amplifier — *Transmit*, *SWR Fail*, and *Over-Temperature* — can be sent to the Arduino through a pair of resistors as a voltage divider to limit the input to less than about 4 V to be safe. The minimum input voltage for an Arduino is about 2.4 V.

A 5 V source (**Figure 11**) to enable the SSR, or enable the *Operate* mode to the amplifier can be accomplished by providing a low signal from the Arduino pin to the base of a PNP transistor, through a 1.5 kΩ

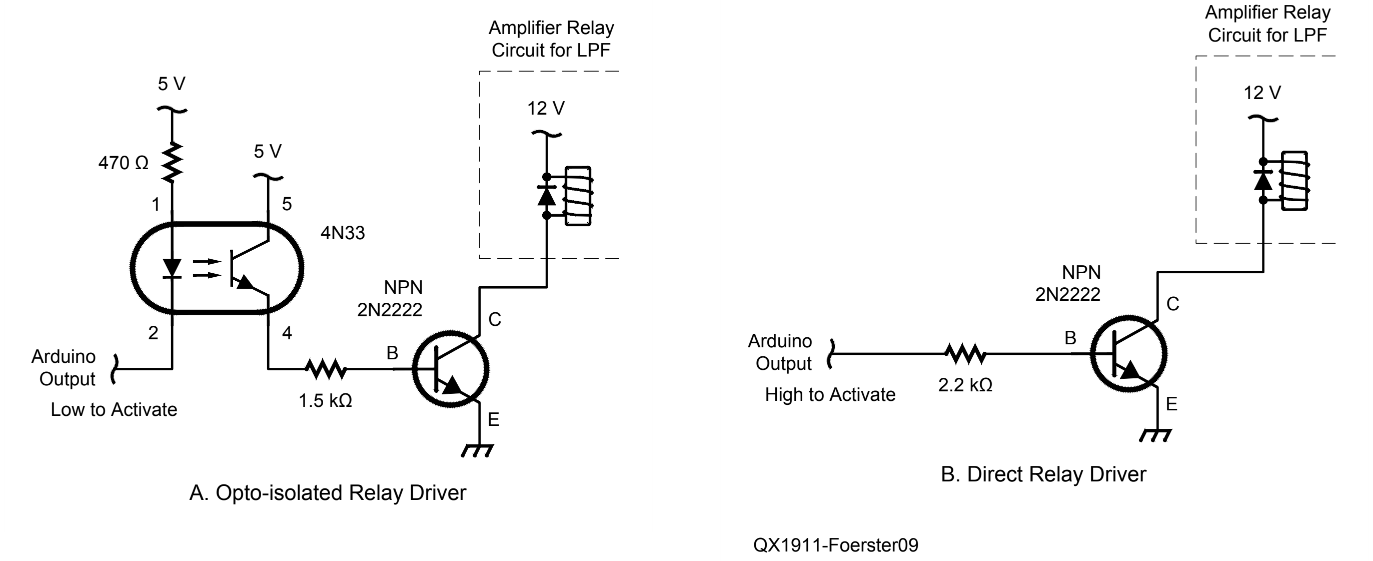


Figure 9 — Connections between the Arduino and relay driver: (A) with opto-isolator, and (B) direct.

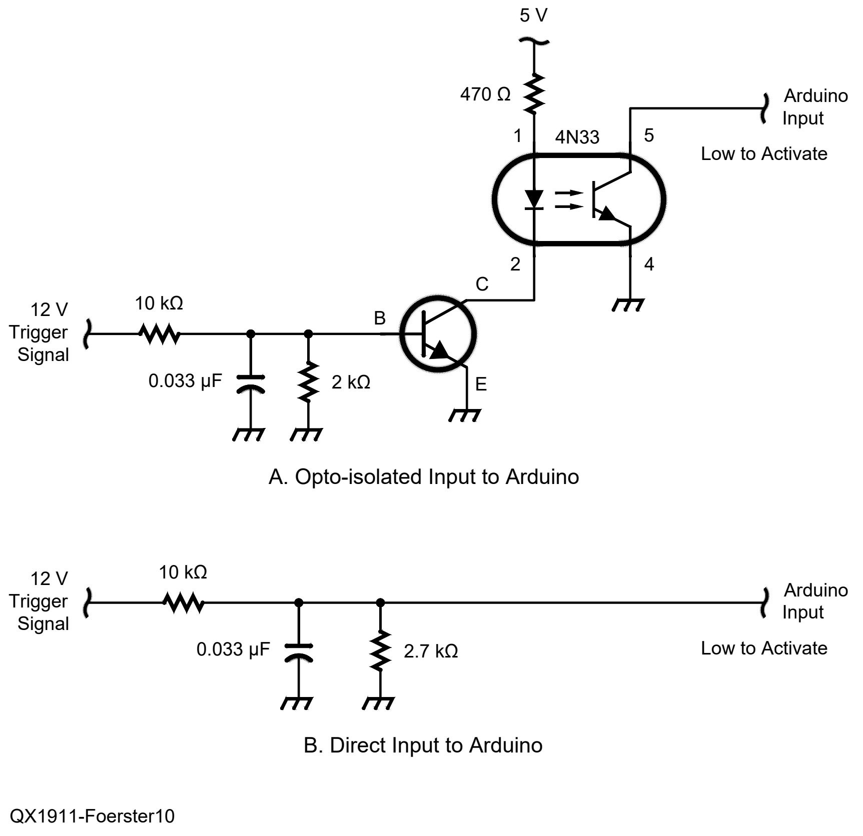


Figure 10 — Arduino digital input circuit: (A) with opto-isolator, and (B) direct connection.

resistor. The emitter of the PNP is tied to the positive voltage, either 5 or 12 V, and brought low to enable a positive voltage to the collector to turn on the output of the circuit. A special circuit may be required to ensure that switching 12 V does not reach the Arduino input.

Figure 12 shows how the 50 V supply voltage is connected from the amplifier to the Arduino to be measured using the analog inputs. To limit this voltage to no more than 5 V, it is prudent to add a Zener diode to limit the voltage to less than 5 V. I used 4.7-V Zener diodes across each input to make sure the voltage cannot go above 5 V. As for the 50 V input, the voltage from the supply should be limited to around 4 V for the 50 V reading. This allows headroom for the voltage to rise above that value, indicating a high voltage failure. Each analog input should also have a small 100 µH choke to limit any RF signal from affecting the inputs.

Serial Communications

The biggest challenge I faced was getting the communication to the radios to work reliably. The lesson I learned was that the inexpensive MAX3232 boards available on eBay are very prone to oscillation, drawing in excessive of 200 mA and with very intermittent behavior. I finally ordered real Maxtor MAX3232 devices from a US online provider and replaced the chips on the small boards. Communications have since worked flawlessly.

While trouble shooting this problem, I used an old trick to measure the current without having to break the circuit. Place a diode capable of supporting the circuit current in series with the positive power input (**Figure 13**). You can then measure the current through the circuit at any time by placing your DVM current meter across the diode. The current meter has a much lower voltage drop than the diode, allowing you to accurately measure the current to the circuit without actually interrupting the current flow to the circuit.

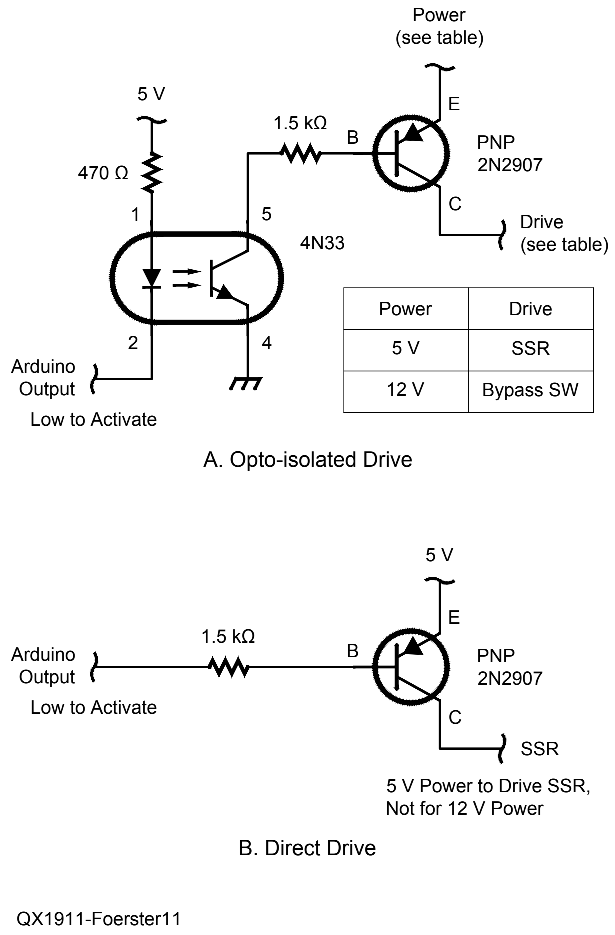
Forward and Reflected Power Indications

I opted for a cross-needle power indication for the forward and reflected power. In retrospect I should have purchased LED metering. Typical SWR boards have negative voltage outputs, but metering and input to the Arduino requires a positive voltage. As it turns out, the LED meters that Jim had available have an op-amp that converts the negative voltage to a positive voltage with no negative supply required. I got boards from Jim with the op-amps but with out the LEDs. Considering that the metering isn't intended to be exact, the LED meters would have been just fine for the amplifier. When using the op-amp driver, make sure that the voltage to the Arduino analog inputs can never exceed 5 V. As with other analog inputs, I added a 4.7 V Zener diode across the op-amp output.

Arduino Code

I won't go into a great deal of detail of coding the Arduino, but there are a few concepts worth pointing out. My project was designed for my Elecraft radios, but the commands for other radios will be very similar. You need just a few basic commands such as reading the operating frequency, reading and setting the radio power output, and turning the radio off.

The Arduino first has a *setup()* routine, which defines all of the input and output pins and executes only once during startup. This is followed by the *loop()* code, which is executed from the top to the bottom, and then



Power	Drive
5 V	SSR
12 V	Bypass SW

Figure 11 — The 5 V drive to the solid state relay: (A) with opto-isolator, and (B) direct connection.

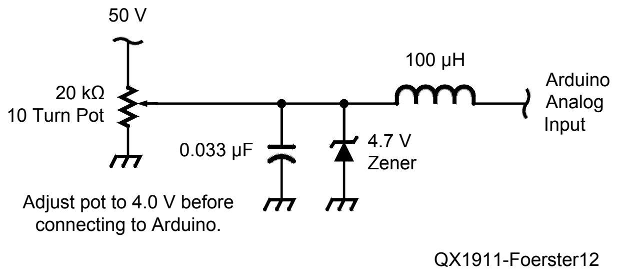


Figure 12 — Reading the 50 V signal without exceeding the Arduino 5 V limit.

repeated constantly. On this project I learned to use the tabs in the Arduino development environment to divide up all of the code into different subject pages. Creating a tab is quite simple. Click on the down arrow button on the upper right side of the Arduino development environment, and click "New Tab". This allowed me to split up the code into different logical files, which makes developing and debugging much easier to understand. All of the declarations, *setup()* and *loop()* are in one file. The other files are: *Analog, Band, Buttons (keys), Display, Eeprom, Morse, PowerUpDown, RigComms, Timeout, Subs* (State Subroutines from main loop).

If you want to view the code I have written, download the Arduino Integrated Development Environment (IDE) **[3]**. Then download my code files from the **www.arrl.org/QEXfiles** web page into a directory named "*LDMOS-Amp-IF*". Double click any of the files in that directory. All of the files should open at once, the main "*LDMOS-Amp-IF*" file starts on the left. The rest are in alphabetical order.

You will also need the Adafruit display driver to compile the project. Download the *Sketch/Include Library/Add.ZIP Libarary* **[4]** and install it in your Arduino environment. You will also need to set the "*Tools/Board*" menu to "*Arduino Mega*".

State Machine Flow

I set up the code flow as a "state machine" (see **www.arrl.org/QEXfiles**), which defines a number of different states, or modes, that are named using easy to understand *NAMES* within the code. Each state is a very clearly defined name. The states change when the user presses the pre-defined keys or from the digital inputs from the amplifier such as "*Transmit*", "*SWR Fail*" or "*High-Temp*". This makes it very easy for the programmer developing the code to keep track of the operations within each state for not only the main loop but also for the key button or the display routines.

The Arduino will execute a loop roughly every 200 ms when running in most modes, other than *ModeOff* and some of the start-up and shut-down routines. As long as you know and understand the modes, it's very easy to figure out the key (button) actions.

Key Actions:

- From ModeOff, Press Select key, proceeds to ModePowerTurnedOn
- From ModeReceive, press of Select key cycles to ModeSetupBandPower (Start of Setup Mode)
  – Up Key increases power by 1 W to a 12 W maximum
  – Down Key decreases power by 1 W to a 1 W minimum
  – Right Key changes band up (80 m to 40 m, etc.)
  – Left Key changes band down
- From ModeSetupBandPower, press Select key cycles to ModeSetupTimeout
  – Up Key increases time by 1 hour to a 9 hour maximum
  – Down Key decreases time by 1 hour to a 1 hour minimum
- From the ModeSetupTimeout, press Select key cycles to ModeSetupBypOper
  – Up key sets mode after band change to Operate mode
  – Down key sets mode after band change to Bypass mode
- From ModeSetupBypOper, press Select key cycles back to ModeReceive
- From ModeReceive, pressing the right key will reset the timeout timer
- From ModeReceive, pressing the Left key will repeat the last Morse error code
- From ModeReceive, a 3 second press of the Select Key turns system off and change to ModeOff
- During Power down routine, pressing the Left key will prevent the radio from being powered off.

The display actions are setup similarly, see **Figure 14**. Knowing what state the code is in makes it easy to define what is shown on the display for each mode.

In some of my early code testing, I found that the system would occasionally attempt to change bands while I was transmitting. It didn't take long to figure out that I needed to inhibit any band changes during transmit, when the RF signal could affect the circuits. The routine to check the radio frequency is executed about every two seconds. The fix was to simply keep resetting the timer for this routine constantly during transmit. Therefore, the check for band change was never executed until at least two seconds after the transmit cycle was completed. Anytime a band change is detected, it's always wise to wait a short period of time (100 ms) and then recheck the change, to be certain there was not an invalid change momentarily detected. I felt it was also important to put the amplifier in *Bypass* mode during the band changeover period.

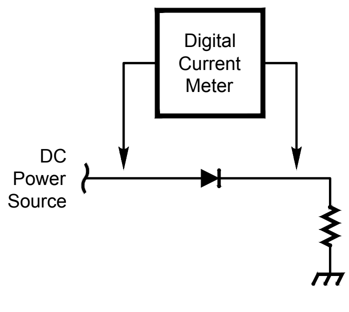
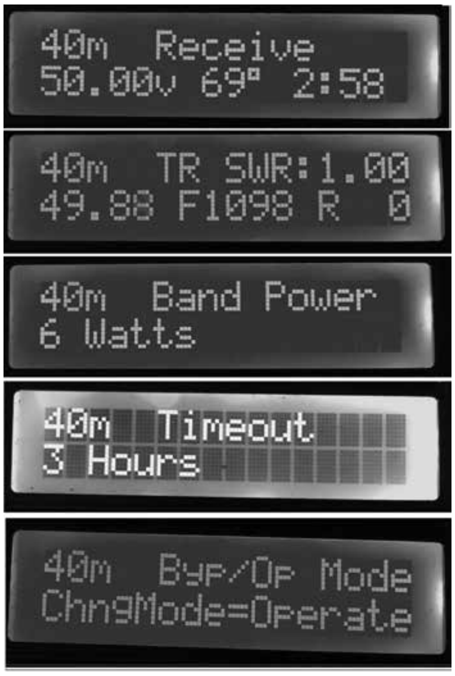


Figure 13 — Current measurement solution that doesn't require breaking a connection.



```
40m  Receive
50.00v 69° 2:58

40m  TR SWR:1.00
49.88 F1098 R  0

40m  Band Power
6 Watts

40m  Timeout
3 Hours

40m  Byp/Op Mode
ChngMode=Operate
```

Figure 14 — Display progression.

System Test Procedure

Before I retired I was a software test engineer. I wrote formal test procedures and executed them for the projects on which I worked. Back then, most of the testing was done using test automation software. True to my past experience, I wrote and executed a test procedure, but executed it manually. I've actually executed the tests a number of times as I made changes to the code. I did find bugs in my code and problems with the hardware as a result of the testing, and even found some new enhancements to add to the system. I discovered I wasn't sending an adequate error message, should the system not preset the power in the radio, or disable the K3 internal power amplifier. At this point I also added commands to turn off the radio, as well as the amplifier. The test procedure, see the **QEXfiles** web page, is also nice to execute after making a number of code changes to make sure everything is still operating as intended.

You need to pay very close attention to any failure modes that are possible with the system, things like if the frequency reading is invalid, or if the writes to the rig power or tune output fail. After each write to change the rig power, you should verify that the setting was successfully initiated. Any failure of these functions must cause the system to go into Bypass mode, preventing possible amplifier action that could damage the amplifier by overdriving the input. It is wise to check the loop time for the different modes often. You may find some of your routines take too long to execute, and timing may need to be adjusted to compensate.

Code Reviews

One of the other habits I retain from my professional days is to review my code. Through the reviews, I've added many comments and modifications to the code to make it more readable and cleaner, besides fixing a number of bugs. I have even asked my son, Tony Foerster, KEØPXK, to review my code. It helps to have another eye on the code to keep you from developing bad habits, and spotting things you may take for granted.

Simplified Design Considerations

In this project I've added a lot of functionality that may not be necessary for everyone, such as controlling radio power output, system power timeout, mode change to bypass, turning the rig off, and so on. Other designers may choose to include a 13 dB attenuator to allow a 100 W input, and therefore controlling the power may not be necessary to prevent overdriving the amplifier.

A much simplified version of the controller could be built to read the band data either by the Comms or BCD band outputs (Elecraft or Yaesu) or even reading the analog pin for an Icom radio to select the band for band switching. You could consider adding an Arduino NANO to simply follow your radio band changes to change the band on the amplifier. This could be done without a display.

Another thought would be to mount the display on the front panel of the amplifier itself. A good RF shield would be necessary to prevent the amplifier RF from affecting the Arduino, and to keep noise from the circuits from generating birdies in your receiver.

Future Plans

When I started this project, I had envisioned a touch-screen display to operate the amplifier interface. However, I wasn't comfortable learning to code the graphical display. Since I was quite familiar with the 2-line, 16-character displays, I decided to use it for the initial build. In thinking about this upgrade, I may try using an Arduino NANO internal to the amplifier with just a serial communication line coming out of the amplifier and going to another Arduino, which would drive the touch-screen display. The connection between these two could be a serial communications port to pass the information required between the two. The communications to the rig would be maintained out of the external Arduino device.

Another future project is to allow this amplifier to emulate the KPA500 serial communications commands from the Arduino to allow it to be used with remote control. The challenge is not with the amplifier commands, but rather allowing the radio to communicate with both the Arduino and the remote control access software through a single port on the radio.

ARRL member Michael Foerster, WØIH, holds an Amateur Extra Class license and has been continuously licensed since he received his first license, WNØVNH, in 1968, then several months later as WAØVNH. He has worked as an electronics technician, and moved into software testing about 25 years ago. Michael retired in 2015 and enjoys experimenting with ham radio, remote radio control, Arduinos and antennas. He also spends part of his summers on a 28-foot sailboat. His amateur station includes the Elecraft K3S, P3, KPA500, along with a KX3 portable radio and vintage Heathkit SB-101, HW-101 and SB-221 equipment.

Notes

[1] J. Klitzing, W6PQL, web page: **www.w6pql.com/**

[2] P. Salas, AD5X, "Amplifier Overshoot-Drive Protection", *QEX*, Sep./Oct., 2018, pp. 15-16.

[3] Arduino Development Environment: **https://www.arduino.cc/en/Main/Software**

[4] Adafruit Display/Keyboard Driver available at: **https://github.com/adafruit/Adafruit-RGB-LCD-Shield-Library**