Paul Young, K1XM

SO2RDuino — a USB SO2R Box

While most SO2R boxes connect with a computer via traditional parallel (LPT) ports, such ports these days are obsolete — replaced by USB ports. I decided to design a USB SO2R box that would be simple. My eventual goal is to have a box that's small enough to take along when I travel.

Features

Simple is nice, but an SO2R box should have a full set of features. This design:

- switches headphones, microphone, CW keyer and PTT (from the computer or a footswitch)
- uses relays for headphone and microphone switching.
- communicates through, and is powered by, a USB port
- includes override switches for transmit and receive and LEDs to display status
- incorporates a BLEND control that blends left and right channels in stereo mode
- Provides four-bit outputs, with band information for each radio. The outputs are compatible with Array Solutions, TopTen, Unified Microsystems or similar band decoders.
- includes a "latch" mode. If the computer transmits (ie, sets PTT), the headphones will be connected in mono to the *non*-transmitting radio, reverting to normal operation when the computer stops transmitting.
- can be set so it will *not* go into stereo — useful for people who have a hearing issue in one ear.
- is compatible with *N1MM Logger*, *WriteLog*, *Win-Test* and similar logging programs

Some Terminology

The *open two-radio switching protocol* (OTRSP) is a simple serial protocol that's supported by the most popular contest loggers. The protocol specification (see **www.k1xm.org/OTRSP/**) is available under a Creative Commons license, which allows anyone to use it.

The *Arduino*[1] is a single-board computer designed for teaching purposes. It includes the microprocessor, crystal and USB interface. It appears to the PC as a serial (COM) port. It can be programmed through the USB interface, and the software to do this is free. There are many clones of the Arduino in different shapes and sizes. An Arduino or clone is almost perfect for this project.

OTRSP uses the DSR modem line to control PTT. The Arduino has DSR available on a pin of the USB interface chip, but it is not connected to the microprocessor. This can be fixed by soldering a 1000 Ω resistor between the appropriate pins of two ICs.

Surrounding Circuitry

The circuitry to make an Arduino into an SO2R box is straightforward. Figure 1 is a block diagram. Two relays are used for the headphones. One relay switches between the two radios; the other switches to stereo. One relay switches the microphone between the two radios, and, to avoid hum, it switches both the hot and ground sides of the microphone.

Two transistors are used for PTT — one

Table 1
SO2Rduino commands

Command	Set
TX1	Set transmitter to Radio 1
TX2	Set transmitter to Radio 2
RX1	Set receiver to Radio 1 mono
RX2	Set receiver to Radio 2 mono
RX1S	Set receiver to Radio 1 (left) and Radio 2 (right) for stereo
?NAME	Print the name of the box (SO2Rduino)
AUX11	Set the value of aux port 1 to 1
AUX215	Set the value of aux port 2 to 15
VMONO1	Do not allow stereo receive
VMONO0	Allow stereo receive
VLATCH1	Turn on latch mode (on transmit listen to the other radio)
VLATCH0	Turn off latch mode

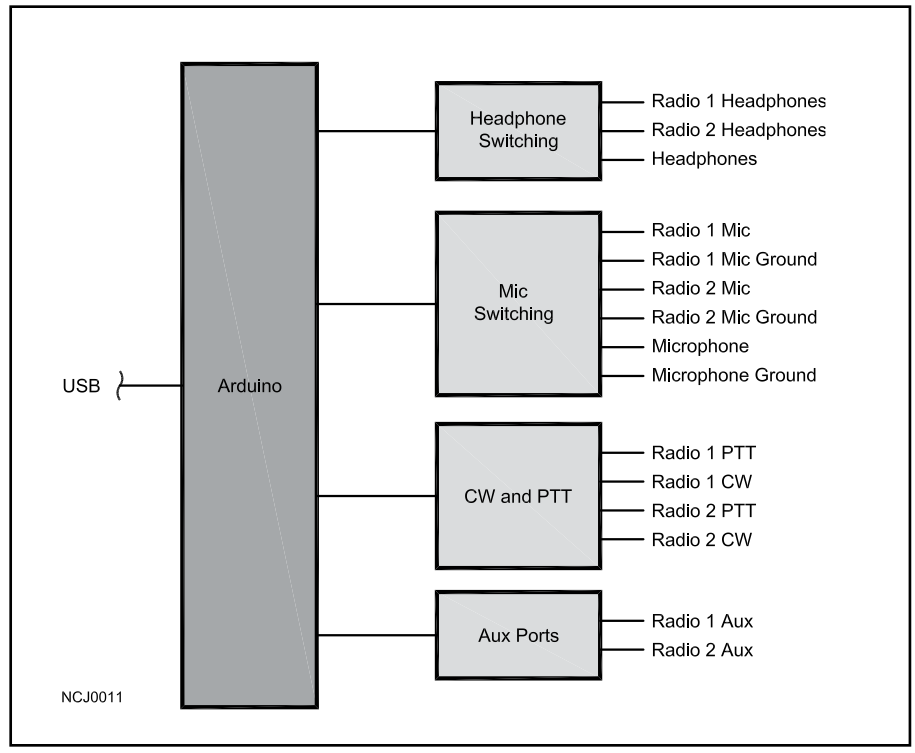

Figure 1 — Block diagram

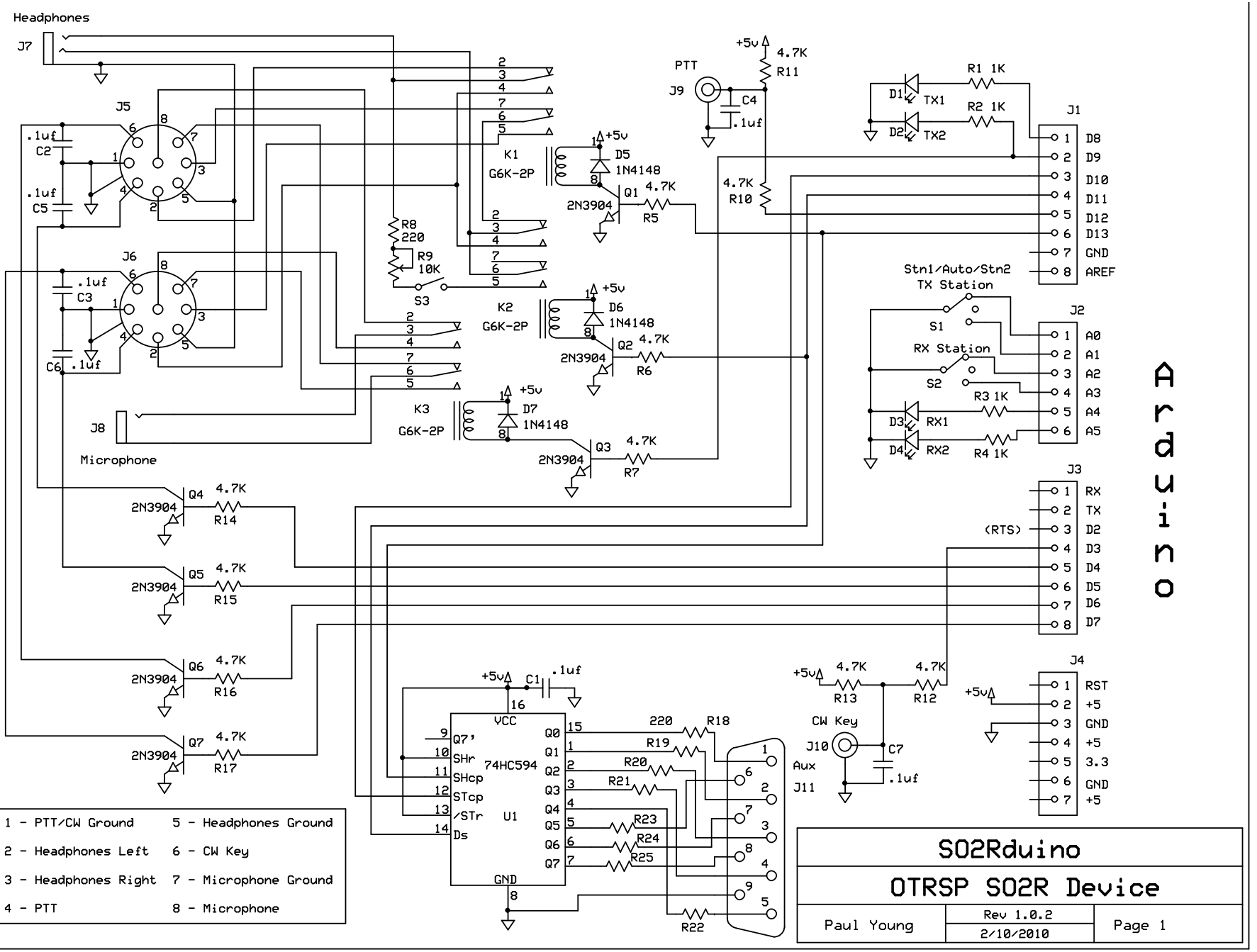

Figure 2 — Schematic diagram and parts list on next page

Parts List

C1 – C7	0.1uf 50 V	Mouser 80-C315C104M5U5CA
D1, D2	Green LED	Mouser 604-WP7113GD
D3, D4	Red LED	Mouser 604-WP7113ID
D5 – D7	1N4148	Mouser
J1 – J4	Stackable header kit for Arduino	NKC ARD-0021
J5, J6	8-pin DIN jack	Jameco 15861
J7	¼-inch stereo jack	Mouser 568-NYS234-3
J8	3.5-mm jack	Mouser 161-7300-EX
J9, J10	RCA phono jack	Mouser 161-2052
J11	DB-9	Mouser 523-G17S0900110EU
K1 – K3	DPDT 5V	Mouser 653-G6K-2P-DC5
Q1 – Q7	2N3904	Mouser
R1 – R4, R*	1 kΩ	Mouser 291-1K-RC
R5 – R7, R10 – R17	4.7 kΩ	Mouser 291-4.7K-RC
R8, R18 – 25	220 Ω	Mouser 291-220-RC
R9	10 kΩ pot with switch	Mouser 774-270X232A103B1B1
S1, S2	SPDT center off	Mouser 108-1MS3T1B1M1QE-EVX
S3	Part of R9	
U1	74HC594	Mouser
Enclosure		Ten-Tec TPB-45
LED holders		Mouser 696-SSH-LX5091
Arduino		NKC ARD-0002
Megashield PCB		NKC ARD-0046
16-pin IC socket		Mouser 535-16-3518-10

R* is the resistor soldered to the Freeduino board.

for each radio, and two more are used for CW keying. This could have been done using another relay, but transistors are cheaper [and don't make noise — *Ed*].

The AUX ports are driven by a shift register IC. The shift register shares two outputs with the headphone relays. The shift register loads within microseconds, so the relays are not affected by the sharing. The front panel has two switches, one for transmit and one for receive. They select Radio 1, Auto, or Radio 2. It also has four LEDs — two for transmit and two for receive. Figure 2 shows the schematic.

Prototype

I used a Freeduino for the prototype. It is available from NKC Electronics, **www.nkc-electronics.com**. It is designed to accept a plug-in daughter board, called a "shield" in Arduino terminology. The standard shield is too small for all of the SO2R box components. A larger version of the Arduino, the Mega, uses a larger shield and has more connectors. The connectors on the Arduino match some of those on the Mega shield, so that shield can be used with a regular Arduino. I purchased a Freeduino, a bare Mega shield and a set of Arduino shield connectors from NKC.

The Freeduino is a kit. When I assembled it, I omitted the power supply parts and the LEDs, because I would not need them for this project. You can install them if you prefer, however. I also soldered the reset button to the shield rather than to the Freeduino board, because the shield is installed on top of the Freeduino board.

I also connected a 1 kΩ resistor from pin 3 of the FT232RL chip to pin PD2 of the processor. If the microprocessor is a DIP, this is pin 4. If it is a surface mount device, it is pin 28 or 32 (the *highest* numbered pin on the device). The FT232 is a surface-mount chip, but I did not use special tools to solder the resistor. I simply put a tiny drop of solder on the end of the resistor lead, placed it over the pin and heated it with my soldering iron. It took me a couple tries to get it right (ie, connecting to pin 3 and *not* shorting to any other pins).

Almost all remaining parts were installed on the shield. I used the holes for the extra connectors as additional board space and put the PTT and CW transistors there. The IC was installed in the area near the center of the board, which is set up for a DIP. The relays were placed to the rear of the board.

I decided to use eight-pin DIN connectors for the radio connections. The pinout matches my existing YCCC SO2R Box.[2] I don't enjoy soldering DIN connectors, but the alternative is to put eight connectors on the box instead of two. The 0.1 µF capacitors were installed on the connectors.

Most parts came from Mouser Electronics, **www.mouser.com**. The DIN connectors came from Jameco Electronics, **www.jameco.com**. The parts I chose were not the only possibilities, nor were they even the best available. In some cases I picked them because they were the first ones I found when looking through the catalogs. The enclosure was chosen because I had one on hand. It is an inexpensive Ten-Tec case, **www.tentec.com**, and it is bigger than needed.

The parts list does not provide part numbers for all components because in some cases there are several manufacturers — the 2N3904 transistor is one example. My strategy is to type the item into the "search" box on the Mouser Web site and buy the least expensive item in stock.

There is nothing critical about the parts I used. The relays are 5 V DPDT units that do not draw much current. The microphone connector is of a type that does not connect to the chassis. I used a stereo connector and only wired two pins.

The shield board simply plugs into the Arduino. I wired parts of the board, plugged it in and tested them as I went. This allowed me to check the hardware and software one piece at a time.

Note that the DIN connector in the schematic is shown from the *plug side*. At first I forgot this and had them halfway wired before realizing that I had them backwards!

Pictures of the front, rear and inside of the SO2R Durino are available on the *NCJ* website, **www.ncjweb.com\bonus.php**.

Programming and Testing

One feature of the Arduino is the complete software, available on the Arduino Web site. Follow the "Getting Started" instructions to download and set up the software. Then go to the *NCJ* Web site, **www.ncjweb.com/bonus.php**, and download the *SO2Rduino* program. Unpack the ZIP file and load it, using the Arduino software. Once the software is loaded the Arduino will become an SO2R box. One transmit LED and one receive LED should be lit. You can change which LEDs are lit by using the two switches. Don't be surprised if the relays switch a few times when you connect the SO2Rduino to the computer.

The next test is to use a terminal program, such as *Hyperterminal*. Start the terminal program and connect to the COM port where the Arduino is connected. Set the speed to 9600 baud, 8 bits, no parity, 1 stop bit, no flow control.

Now you can type commands to the SO2Rduino. All commands are in upper case followed by the return character (the Enter key). Table 1 lists a few commands and their effects. If the RX1S command does not produce stereo, with both receive LEDs lit, then the box is probably in mono-only mode. Use the VMONO0 command to change this.

Connect the box to one or two radios, a microphone, headphones and a keyer. If you hear hum in the headphones make sure that the SO2R box and the radios are

properly grounded. Check that by using the switches and commands you can listen to either or both radios and transmit on either radio. The final check is that the computer can set the PTT connection. You will need to use a logging program for this.

Operation

How you set up the SO2Rduino depends on the logging program. The Arduino COM port is set to OTRSP, 9600 baud. RTS is set to PTT, and DTR is set to off. Some logging programs will automatically set the baud rate and other parameters for OTRSP.

If you had been using a different SO2R box, make sure you clear all of the old settings before setting up this box. If the program offers a choice, set the keyer to work with both radios. Once the logging program is set up properly, it should be able to switch transmit and receive radios on command.

Possible Modifications

The most obvious modifications would be to leave things out. For example, if you do not need band decoder outputs you could leave out U1, J11 and R18 – R25. If you don't want a BLEND control you could leave out R8 and R9. In addition you really don't need the switches and LEDs, because the computer can fully control the SO2Rduino, so you could leave out S1, S2, D1 – D4 and R1 – R4. One operator I know prefers to always hear Radio 1 mostly in the left ear and Radio 2 mostly in the right ear. This can be done by leaving out a few wires connecting K1 and K2 to the radios and rewiring the BLEND pot so it is always connected.

A different Arduino clone could be used. Several clones are designed to plug into breadboards. It might be easier to build an SO2Rduino using one of these. You could use 12 V relays. This may be convenient if you have an existing SO2R box that you want to upgrade. The 2N3904 will drive a 12 V relay. You will need to use an external power supply, and you can power the Arduino from that supply rather than from the USB port.

If you have the ability to deal with surface-mount parts, you could make a PC board for this project. If you do, you might want to consider using an FT2232 IC, which is a dual USB to serial device. Then you could add a K1EL Winkey 2 IC to the board (visit **http://k1el.tripod.com/WinKeyer2.html**).

Future Plans

I have started to build an SO2Rduino for my traveling station. It will use the Modified Pico, **www.modifiedelectronics.com**, a small Arduino clone. All features will be available except the AUX port, since I don't plan to take band decoders on the road. The result should fit nicely in a Ten-Tec TPB-17 box. The SO2Rduino, a K1EL WinKeyer and a multiport USB-to-serial converter will give me SO2R capabilities with a notebook computer.

Summary

Using an Arduino or clone makes a computerized SO2R box that's about as simple as one that works from the LPT port. In addition, it gains a few features, such as AUX ports for both radios, *and* it won't soon become obsolete.

Notes

[1] The official Arduino site is **www.arduino.cc**. More information about clones can be found at **www.freeduino.org/**.

[2] The YCCC SO2R Box (plus) is a more complex SO2R device with a built-in keyer. Information is available at **http://so2r.k1xm.org**.

NCJ