Yet Another Audio Interface for the Digital Modes (But With CAT Control) Version 2.0

John Price – WA2FZW

Introduction

A while back, my primary radio went belly up, and while it was out for repairs, I thought I might drag out my old Swan-250C 6 meter rig so I could at least get on the air. I had built a simple hand-wired audio interface for it, but couldn't find it, so I designed a new one!

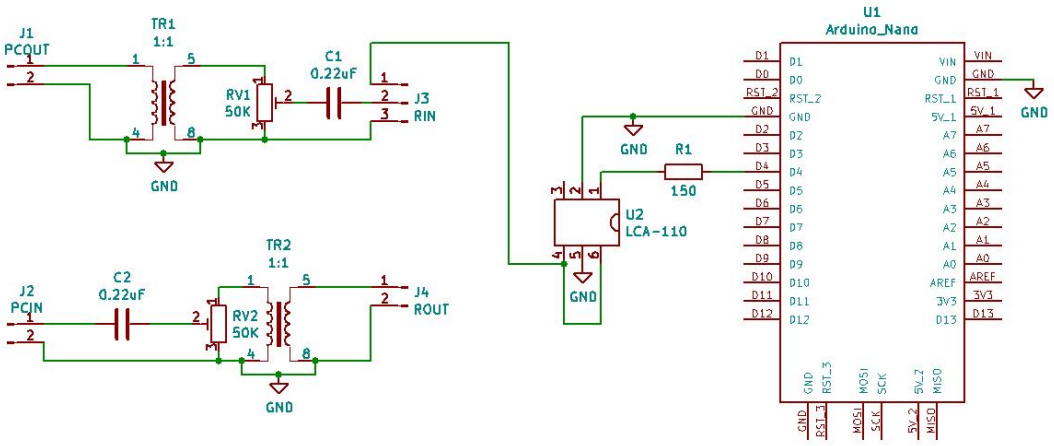
The audio portion of the device is pretty standard. It is a hybrid of the "Easy Digi" audio interface and a few other designs that are floating around. What's different about this one is that I added an Arduino Nano processor so that the transmitter could be keyed via CAT control instead of using the more common RS-232 hack.

Changes in Version 2.0

In Version 2.0 I replaced the entire message handling code that was in the original versions with a standard Arduino library which I had developed for another project. That library can be downloaded from GitHub and installed in your Arduino library directory.

Hardware

The audio portion of the hardware is similar to other audio interfaces commonly available. Here's the full schematic:



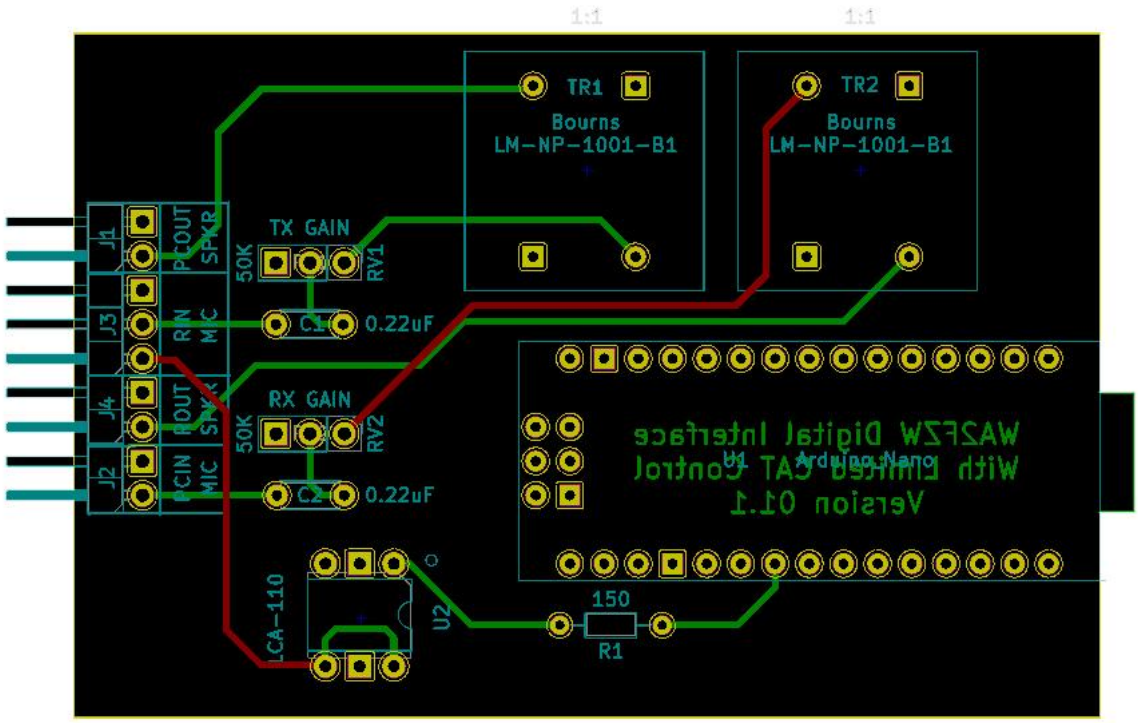
Transformer isolation is used for both the transmit and receive audio paths and C1 and C2 block any DC on the microphone connections; the PC microphone is set up for an electret type microphone and thus there is a DC bias present. The same is true for my BitX-40 radio as well.

RV1 & RV2 provide a means of setting the audio levels such that they can be more finely adjusted using the PC's control panel settings and/or by the program that is interfaced to the radio. The schematic shows these to both be 50KΩ trimpots (what I happened to have in the parts bin), but almost any value will probably work.

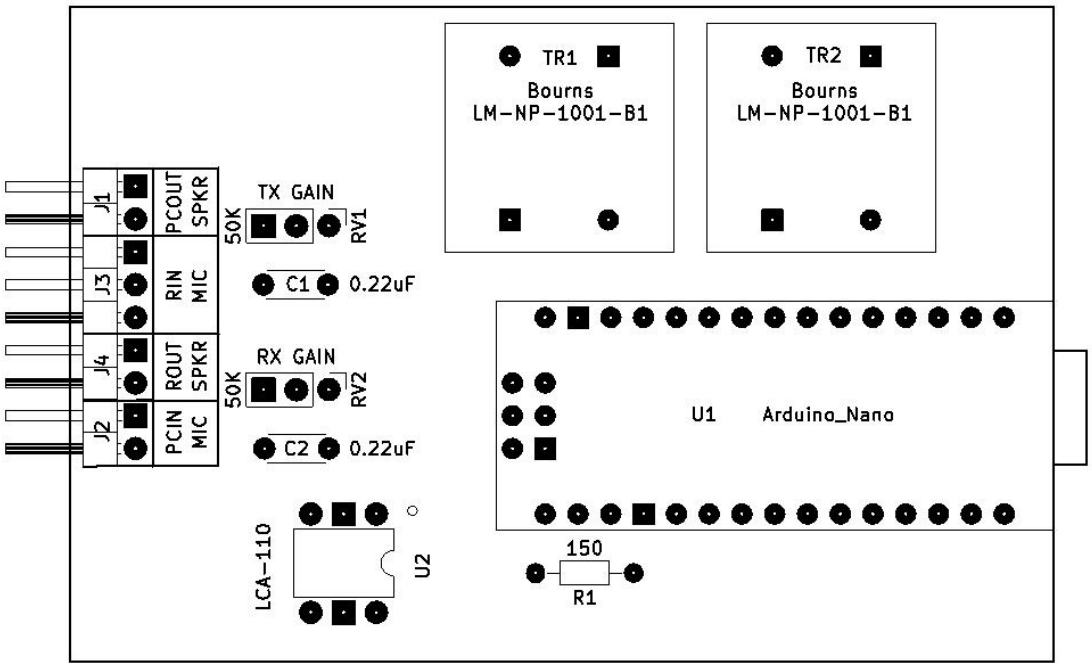
When you install RV1 and RV2 on the PCB, you want to orient them so that the resistance between the ground pin and the center pin increases as you turn the adjustment clockwise.

The Arduino operates U2, which is a common solid state relay to key the transceiver's PTT line when it receives a command to transmit or receive from the PC via its built-in USB port.

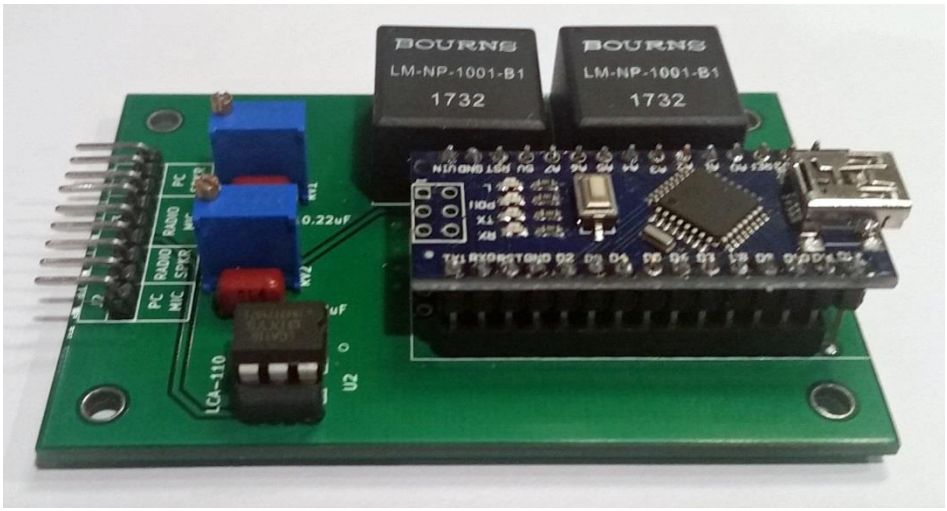
Here’s the circuit board (the Gerbers are included in the distribution package):



Here’s a clearer picture showing the parts layout:

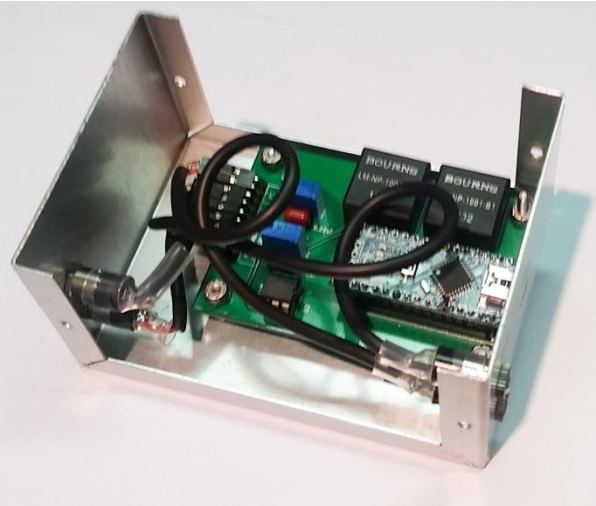


And the completed (Version 01.0 – No R1) circuit board:



I used a standard 28 pin IC socket for the Nano, even though it has 30 pins. As D12 and D13 are not used (the 2 end pins on the end of the board with the USB connector) they don't go in the socket. If you look closely at the picture above, you can see that.

Update: I finally put it in a box; here's what that looks like:



Software

The software is a pretty "dumb" CAT control program. It uses the Yaesu FT-891 command language. Why did I choose that instead of the more common Kenwood language? My primary radio is an FT-891 and my primary CAT control program is *DxCommander*. When I first got the radio, there were a number of issues with how the two communicated due to some things I consider bugs in the radio's software and some misconceptions that the author of *DxCommander* had about the radio's command language (he didn't have one to test with). The author and I spend several weeks working out the kinks, so I know this code will work with *DxCommander*. It should also work with other CAT controllers capable of communicating with the FT-891 such as *N1MM+*.

Program File

In Version 2.0, there is only one file; *Digital_Ifc_V2.0.ino*. As noted earlier, all of the actual CAT control message handling is now done in the *FT891 CAT* library available on GitHub.

Operational Overview

As I stated earlier, this is a pretty "dumb" program. It only responds intelligently to two messages; "TX" and "ID". When it receives an "ID" message, it responds with "ID0650;" indicating the CAT language in use is that for the FT-891. "TX1;" and "TX0;" messages tell the Arduino to turn the transmitter on or off.

Commands to change parameters in the "radio" simply update the values of various things such as frequencies and operating modes stored in the library. Status requests for these items are answered based on those stored values. This keeps the controlling PC program happy as not getting a response to a request might cause problems.

Program Functions

The following sections provide a high level description of what each function in the program does. More detailed information can be found in the comments in the files themselves.

The functions are listed in the order in which they appear in the program.

setup

This function assigns the Arduino pins used to control the transmitter and on-board LED, starts the serial communications channel and and initializes the *FT891_CAT* library.

loop

Once the *setup* function completes, *loop* runs forever.

It calls *CAT.CheckCAT()* in an if statement. If that function returns a *true* indication, it means that a new command was received and processed by the library and that we should go look to see what changed. If it returns *false*, then nothing changed.

If a new command was processed, we call *CAT.GetTX()* in an if statement to get the current transmit/receive status and turn the Arduino's on-board LED on or off as appropriate; note, the CAT library will turn the transmitter on or off, so this program doesn't need to handle that task.

Delay

This is a non-blocking time waster. The standard Arduino "delay" function blocks interrupts, which prevents I/O from happening.

Setting the Audio Levels

The two trimpots (RV1 & RV2) are used to adjust the audio levels between the real radio and the PC. There are several adjustments that can affect the levels:

- The trimpots
- The Windows (or other OS) audio level controls (in the Control Panel) under "Sound".
- The audio output level adjustment and mic gain adjustments for the radio itself.
- The "Power" adjustment on the *WSJT-X* main screen

As we're dealing with old-fashioned radios here, there may be no microphone gain adjustment, and the only adjustment for the output level may be the volume control (unless you go into the radio and pick off an audio signal before the volume control).

To the extent possible set the audio controls on the radio to mid-range values or for the audio output, a comfortable listening level. Similarly, set the PC level controls and *DxCommander* "Power" controls to mid-range values.

With the radio just receiving noise, adjust the “RX Gain” trimpot so that the receive level meter on *DxCommander* shows about 40dB. If you can’t adjust it to that level, you’ll need to diddle with the PC’s microphone level adjustment or the volume setting on the radio.

If your radio has a way of displaying the ALC level, with the radio transmitting on one of the digital modes, adjust the “TX Gain” trimpot so that it just starts to show the ALC kicking in then back it off a tad.

If you can’t monitor the ALC, adjust the “TX Gain” trim pot for full power output then back it off a tad.

If you can’t get the transmit level to adjust with the trimpot you’ll need to adjust the PC’s audio output level.

Bill of Materials

Here’s what you need to build it.

U1	Arduino Nano Processor (socket optional)
U2	SCA-110 Solid State Relay (socket optional)
TR1 & TR2	Bourns LM-NP-1001-B1 Transformer
C1 & C2	0.22uF capacitor
R1	150Ω ¼ watt resistor
J1 – J4	Male right angle 2.54mm pin headers (the board is designed so that a single 9 pin header can be used).
RV1 & RV2	Vertical style trimpots. The schematic calls for 50KΩ since I had a bunch on hand, but most likely almost any value greater than 2KΩ should work just fine
J1A, J2A & J4A	(Not shown on the schematic) 1/8” TS (mono type) audio jacks mounted on the enclosure and connected to J1, J2 and J4 respectively.

J3A	(Not shown on the schematic) 1/8” TRS (stereo type) audio jack mounted on the enclosure and connected to J3.
Enclosure	At least 4” long x 2.25” wide x 1.5” high. The one I used came from Jameco.
Cables	Whatever you need to connect between the device and your PC and radio.

Suggestion Box

I welcome any suggestions for further improvements. Please feel free to email me at WA2FZW@ARRL.net.