QEX

November/December 2020

www.arrl.org

A Forum for Communications Experimenters

Issue No. 323

W3YJ operates FT8, a WSJT-X mode, remotely from a coffee shop.

Harry Bloomberg, W3YJ

201 Delafield Rd., Pittsburgh, PA 15215; **w3yj@arrl.net**

Remote Operating with a Raspberry Pi, *Fldigi/Flrig*, *WSJT-X*, and *NoMachine*

This inexpensive configuration lets you operate an all-mode station located remotely.

My primary residence is on a small suburban lot where putting up a good HF antenna is a challenge. A few years ago I inherited five acres of woods and an old Amish farmhouse 85 miles away. I hung some antennas from tall walnut and pine trees on the property and spent weekends and holidays operating from there.

I started researching how to access the station located at my wooded location from my primary residence. Some solutions were quite expensive and complex. However, I discovered three key pieces of technology that allowed me to operate CW, digital modes, and SSB remotely and inexpensively.

The first piece was the Raspberry Pi, a small inexpensive single-board computer that is used in many embedded applications. It runs a distribution of the Raspberry Pi OS (Raspbian) Open Source Linux operating system. I purchased a Raspberry Pi 4 with 4 GB of RAM as part of a kit containing important accessories at a cost of US $99.

The next pieces were *Fldigi* and *Flrig*, part of the *NBEMS* suite developed by Dave Freese, W1HKJ. *Flrig* enables you to control a transceiver through a USB interface. You can change frequency, adjust power, and control other major parameters on a variety of transceivers. *NBEMS* runs on Windows, MacOS and Linux, including the Raspberry Pi. You can also use *Flrig* to act as rig control for *WSJT-X*.

The final piece was the *NoMachine* remote operating software. I looked into various ways of logging onto the Pi. Then a good friend told me about how *NoMachine* was used to log securely onto computers at a major government lab. *NoMachine* is free for personal use and will stream audio both to and from the Raspberry Pi. I joined a *NoMachine* support forum and soon learned how to interface *NoMachine* to the audio system on the Raspberry Pi. *NoMachine* makes clients

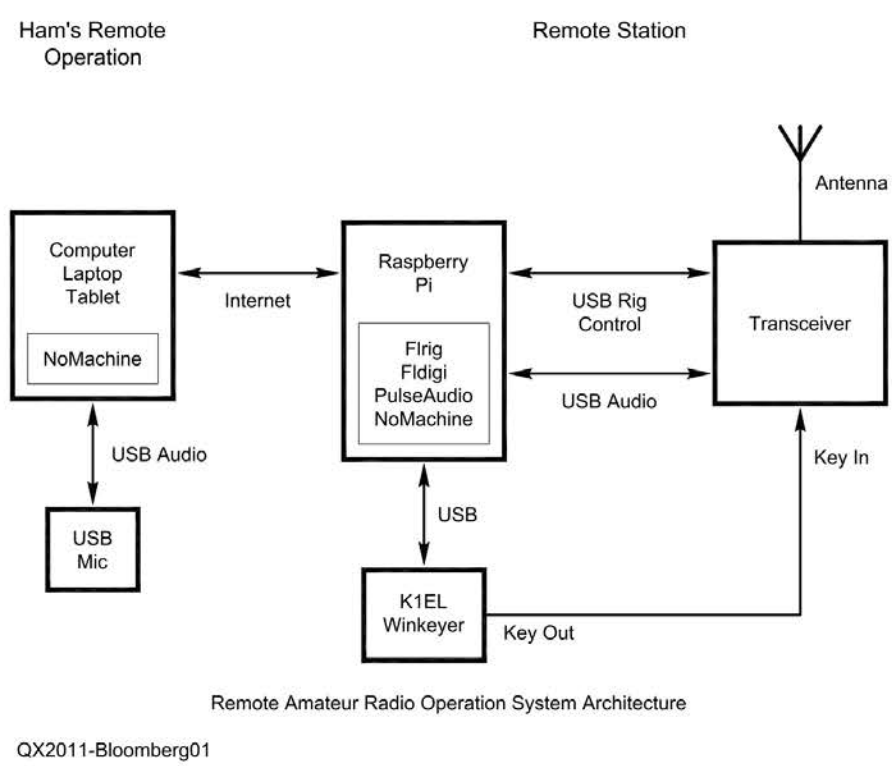


Figure 1 — Block diagram of the system architecture.

for all major platforms including Windows, MacOS, iOS and Android that can connect to the *NoMachine* server on the Pi. *NoMachine* for all platforms along with installation instructions, documentation, and support is available from **www.nomachine.com**.

I knew my remote system was fully operational when I connected to my station from a coffee shop and worked *FT8* stations from my iPad with *WSJT-X*, and then made a CW contact with a special events station using *Fldigi*. I also made several SSB contacts in the Pennsylvania QSO Party from a hotel room in Michigan.

The advantage of this method of remote operating is that it is very inexpensive. The only additional hardware you need to purchase is a Raspberry Pi. You must, however, learn Linux system administration. Most ham clubs have some members who are familiar with Linux who might be able to help you.

Integrating NoMachine with the Raspberry Pi

Figure 1 shows a block diagram of the system. At the center of everything is a Raspberry Pi Model 4. I initially made a prototype system with a Pi Model 3B+, but on occasion the IC-7200 would lock up and lose the audio connection to the Pi. This has never happened with the Pi 4. I suspect the Pi 4 works better because it is much more powerful than the Pi 3.

The IC-7200 is particularly well suited for connecting to a computer because both audio and rig control commands are carried over the USB cable. I have also operated an Elecraft KX3 remotely with the Pi, but an external USB soundcard and cables going to the microphone and headphone jacks are required.

Audio is processed on the Pi by a package named *PulseAudio*, which acts as an audio server, sending and receiving streams of audio much as a webserver sends and receives internet data. *NoMachine* interfaces with *PulseAudio*. A common commercial use for *NoMachine* is logging onto remote systems and streaming audio remotely to remote microphones and speakers for VoIP communications. Hams of course want to hear audio coming from the transceiver and send audio to a microphone input on the radio. Some Linux shell commands are run once after logging into the Pi to redirect these audio streams.

A detailed explanation of how to interface *NoMachine* with *PulseAudio* is "beyond the present scope. Please see **www.w1hkj.com/W3YJ/** for detailed tech notes. But, only a handful of Linux shell commands are required. I have written a Perl program that probes your Raspberry Pi for soundcards and writes the shell commands for you. This script is named write_script.pl and is also available for download from **www.w1hkj.com/W3YJ/**. *PulseAudio* refers to audio inputs like mics as "sources" and output devices like speakers as "sinks". Figure 2 shows sinks and sources when my IC-7200 is connected to the Pi and how the Linux shell commands cause sinks and sources to be reconfigured to interface the IC-7200 to NoMachine. The tech notes explain how to obtain the names of sources and sinks for your USB soundcard."

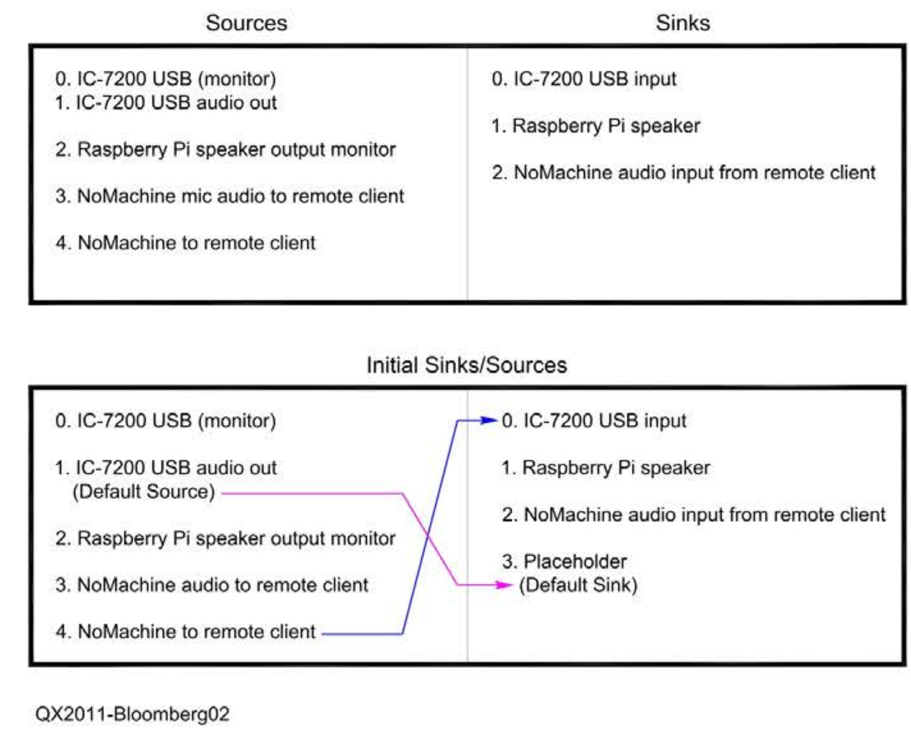


Figure 2 — Summary of sinks and sources when the IC-7200 is connected to the Raspberry Pi.

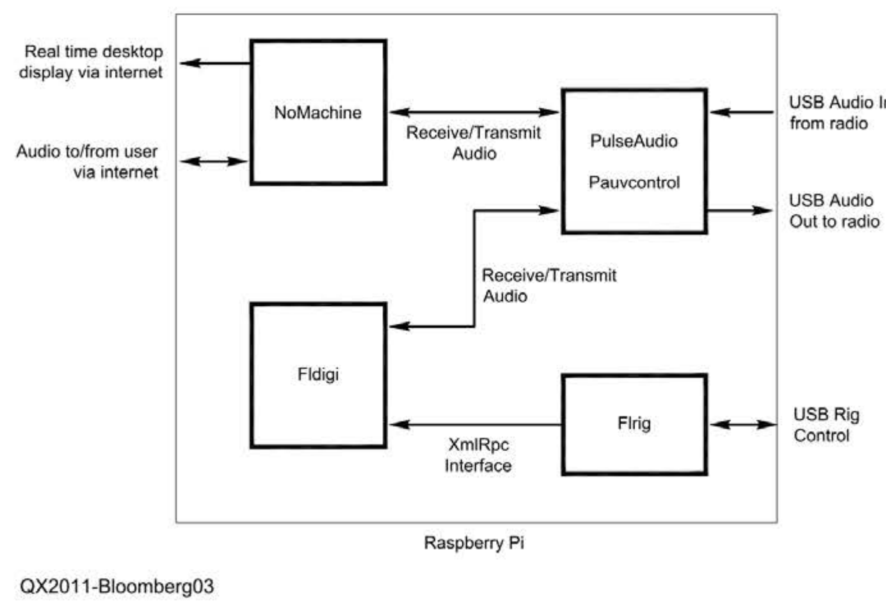


Figure 3 — PulseAudio sources and destinations controlled by pauvcontrol.

Adding *Fldigi* and *Flrig*

There are two ways to install *Fldigi* and *Flrig*, see **Figure 3**. The easiest way is to use the Pi *Add/Remove* software utility. Unfortunately, this will install a version of the software from a Raspberry Pi repository that is quite old, and you will not have the latest features or bug fixes. An alternative is to build the software from source code. Excellent instructions can be found at the W1HKJ web page **[1]**. You will need to open a terminal and enter some Linux shell commands.

Once *Fldigi* and *Flrig* have been installed, you must first configure *Flrig*. Go into the Configure menu and select your transceiver and USB communications parameters. You'll know *Flrig* is configured correctly when you see your transceiver's frequency correctly displayed in *Flrig*.

Next, configure *Fldigi* to use *Flrig* for Rig Control. Go to the Configure > Rig Control menu and click on the checkbox to use *Flrig*. Again, you'll know *Fldigi* is properly configured when you see the correct frequency displayed in *Fldigi*.

You are now ready to tell *Fldigi* to use *PulseAudio* for audio and to connect *Fldigi* to your USB soundcard devices. In *Fldigi*, go to the Configure > Soundcard menu. Select the Devices tab and click the checkbox for *PulseAudio*. Leave the box for Server String empty because you're accessing *PulseAudio* on this computer.

To connect *Fldigi* to the correct USB soundcard devices, open the *pavucontrol* utility. You can install *pavucontrol* using the Raspberry Pi *Add/Remove* software utility. Go to the Recording tab. You'll see one audio stream with the *Fldigi* icon next to it. This is the volume control for audio being directed into *Fldigi*. Select your USB soundcard device from the drop down. Then adjust your audio level and make sure you see signals on the waterfall. A good rule of thumb for adjusting the input level is that the waterfall is mostly blue with signals in yellow. If you see signals that are red, your level is too high and you won't get the full benefit of your soundcard dynamic range.

The process is similar for *Fldigi* transmit audio. Go to the Playback tab and find the stream that has a small *Fldigi* icon. Change the dropdown to your USB audio device. Adjust the output level so that your radio ALC just moves a little bit. Note that if you click on the *Flrig* SWR meter while transmitting, you'll see ALC displayed.

My favorite way to have *Fldigi* generate CW and key the transceiver is by installing a *WinKeyer* by K1EL. I have had great success with both the *WKUSB-SMT* and *WKmini*. The K1EL web page **[2]** has full instructions, as does the W1HKJ *NBEMS* web page **[3]**. *Fldigi* can also control a nanoKeyer **[4]**.

I made a few CW contacts in the Worked All Europe (WAE) contest while staying in a hotel room on a vacation. To configure *Fldigi* as a contest logger, go to Configure > Contest/Logging > Contest. The Contest drop-down menu allows you to set up *Fldigi* for a variety of popular contests including ARRL Field Day. *Fldigi* can also check for dupes and automatically generate serial numbers. Macro keys can be edited to automate contest exchanges and logging. Logs can be exported in a variety of formats including ADIF, CSV and Cabrillo. *Fldigi* has an excellent CW modem to help you copy CW.

Remotely Logging Onto Your Raspberry Pi

Perform the following steps to log onto your Raspberry Pi.

1) Right click on the *NoMachine* icon on the toolbar at the top of your Pi screen. Go to the Show the Service status menu. Note the *nx:* address that has a global IP address, which is an IP address that is not on you local network. Also write down the port number. This is the number that is separated from the IP address by a colon.

2) Enter this *nx:* address into *NoMachine* on your computer or tablet device.

You can now connect to your Raspberry Pi from your computer or tablet. To ease this in the future, you may want to subscribe to a service like NoIP so that you can connect to your Pi by the machine's name instead of IP address, which could be changed by your Internet provider.

Configuring *NoMachine* Audio

When you log onto your Raspberry Pi using *NoMachine*, run the shell script (described in the Tech Notes) to connect *PulseAudio* to *NoMachine*. You must do this while logged in using *NoMachine*. If you try to run the script while logged directly onto Pi, the *PulseAudio* sources and sinks for *NoMachine* won't be available and the script will fail.

To adjust the volume of the audio from your radio that is being sent to your device, open *pavucontrol*, go to the Recording tab, and adjust the level of the *NoMachine* slider. You may need to change the value of the drop down for this slider to "Monitor of Null Output."

I have found that often the default audio setting for the *NoMachine* client is to have audio muted. You may have to go into *NoMachine* settings, then go to the audio icon, and click on the left speaker icon to un-mute. This control also gives you another audio level for the volume of the audio coming from your radio.

Configuring for SSB Operation

Using *NoMachine* for remote operating works most simply if you want to operate CW or digital modes. But SSB operation is also possible with a USB microphone connected to your computer or using the microphone built into your computer. You must change the audio settings on your *NoMachine* client on your computer to unmute your microphone, **Figure 4(A)**. You can also adjust levels in your *NoMachine* microphone settings, **Figure 4(B)**. In addition you'll need to adjust levels on your computer and select PTT, **Figure 4(C)**. Finally, you must go to the Playback tab on the *pavucontrol* program and change the output for the *NoMachine* audio stream to your radio USB soundcard device. Note that you cannot do this unless your microphone is enabled in *NoMachine*.

Operating *WSJT-X* with *NoMachine*

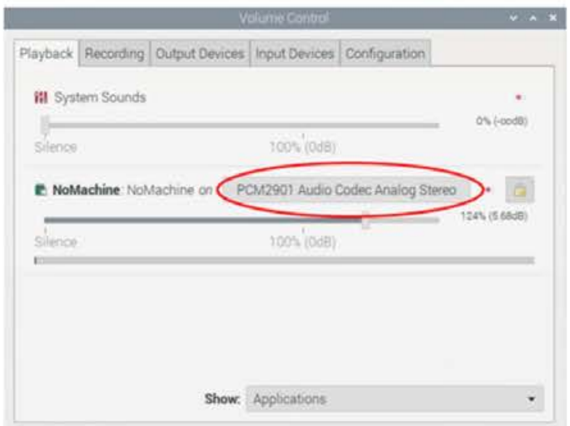
WSJT-X has a built-in radio interface to *Flrig* that will simplify radio configuration. In *WSJT-X* go to File > Settings > Radio and select "FLrig" as the radio and CAT as the PTT control. This allows *Flrig* to control *WSJT-X* and saves you having to configure your radio directly through *WSJT-X*.

For *WSJT-X* audio settings, go to the Audio tab and select your USB soundcard. Setting audio levels with *WSJT-X* and *pavucontrol* is a little tricky. A quirk of *WSJT-X* is that the audio input from *PulseAudio* is by default always set to 100%. You will need to change this every time you start *WSJT-X*. To reduce the level, go to the Recording tab in *pavucontrol* and find the stream with the *WSJT-X* icon labeled QtPulseAudio. Use this control to reduce the input audio level. I find a setting around 50% works for me.

You can only adjust the audio level for transmitting while actually transmitting. To set the audio level, first reduce the power level on your transmitter in *Flrig* to a very low value to minimize interference. Press the *WSJT-X* Tune button. While

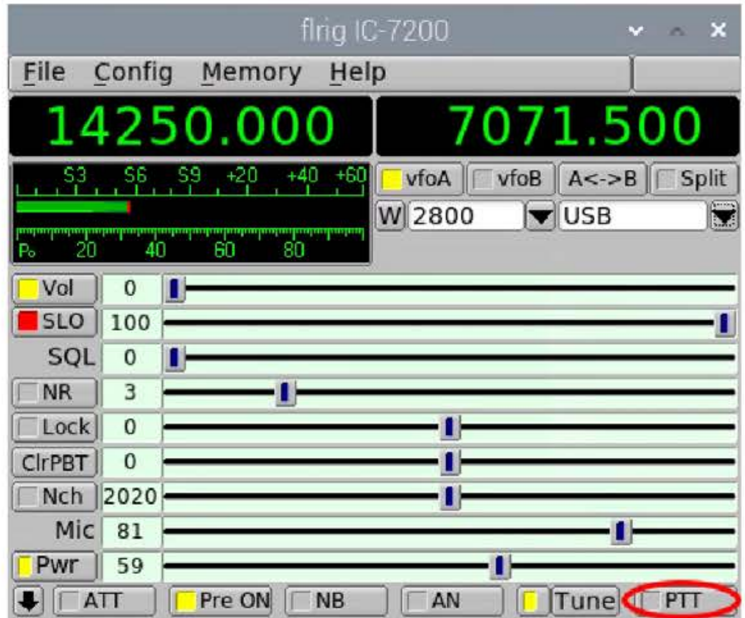
QX2011-Bloomberg04A

(A)



QX2011-Bloomberg04B

(B)



QX2011-Bloomberg04C

(C)

Figure 4 — Configuring for SSB, (A) unmute the audio at 'Mic in'; (B) set audio level for the NoMachine audio codec; (C) select PTT in Flrig IC-7200.

transmitting, go to *pavucontrol* on the Playback tab. Find the stream with the *WSJT-X* icon labeled QtPulseAudio. Adjust this level until you see a small amount of ALC on your transmitter. Fortunately, this needs to be adjusted once and will stay the same every time you execute *WSJT-X*.

Remote Station Considerations

Should something go wrong with your remote station, you cannot be there to pull the kill switch on your radio or cycle power to your Raspberry Pi. I have installed Wi-Fi controlled ac power sockets for both my IC-7200 and Pi. This allows me to cycle power to both devices using an app on my cell phone. To increase the reliability of my home network, the ac power for both my wireless router and the Raspberry Pi are connected to a UPS. I also have the Raspberry Pi directly plugged into the wireless router using an Ethernet cable.

Putting It All Together

Are there a lot of moving parts in this method for remote operation? Absolutely! You are almost guaranteed to have total failure if you install *NoMachine*, *Fldigi*, *Fldigi*, and run your PulseAudio scripts all at the same time. It is very important to add one feature at a time, make sure each new feature works, and then move on to the next step. You should add functions in the following order:

1) Build *Fldigi* and *Flrig* on your Pi from the source, or install from the repository.

2) Connect to your rig using *Flrig* and make sure all the major controls in *Flrig* work with your rig.

3) Set up *Fldigi*. First, make sure it works with *Flrig*. Next, make sure you can send and transmit audio using *PulseAudio*. Adjust audio levels with *pavucontrol*.

4) Connect your K1EL Winkeyer and make sure you can key your radio with *Fldigi* in CW mode.

5) Install *WSJT-X* and use *Flrig* as rig control. Adjust audio levels with *pavucontrol*.

6) At this point you can make digital and CW contacts using your Raspberry Pi. Spend some time on the air to become familiar with operating using your Pi.

7) Run the *write_script.pl* Perl program **[5]** to identify your USB audio device and to write the shell commands to reroute audio streams.

8) Install *NoMachine* on your Pi and your computer or tablet.

9) Log onto your Pi using *NoMachine* from your local network.

10) Run the scripts written by *write_script.pl* to redirect audio. Enable your speaker and microphone in *NoMachine*.

You are now ready to connect to your remote system. Try it out in your shack before you head to your favorite coffee shop (**Figure 5**).

EmComm Applications

Installing an HF antenna at a mobile or portable command post can be problematic due to the nature of HF antennas, especially for the 80-meter band. Yes, you can use a portable or mobile antenna, but even the best of these will not work as well as a full-size antenna. You might consider connecting to a remote HF station during a deployment. Think back to your most recent Field Day operation and remember how long it took to set up your HF station and how much planning was required. Think of all the little things that can go wrong like a bad piece of coax or a poorly soldered RF connector. Now imagine setting up an HF station under the duress of a drill or actual emergency deployment. You may be better off trying to connect to a well-maintained permanent station if a network is available, rather than hastily setting up an HF station with a poor-performing temporary antenna at the deployment site

Does this add a layer of complexity to a deployment? The same was said years ago about NBEMS, Winlink, and mesh networking. They're now standard parts of our EmComm toolbox. Also think back to your most recent ARRL Field Day operation and remember how long it took to set up your HF station and how much planning was required. Think of all the little things that can go wrong, like a bad piece of coax or a poorly soldered RF connector. Now imagine setting up an HF station under the duress of a drill or actual emergency deployment. You may be better off trying to connect to a well-maintained permanent station if a network is available rather than hastily setting up an HF station with a temporary antenna at the deployment site.

Future Considerations

With a price of $35-55, the Raspberry Pi is easily affordable. It also has many input/output ports that can be adopted for controlling equipment in a ham shack. The Linux operating system is an ideal platform for amateur radio operators because it is open source and therefore suitable for tinkering and experimenting. Vendors of commercial amateur radio equipment should consider porting their rig control and radio programming software to Linux so their software can run on a Raspberry Pi. Why write this article? One reason is fame and fortune. Another is to ask for help from other hams. I worked on this alone and your collective wisdom is welcomed. Please review my work! Is there another way to do this? I love NoMachine but I'd rather be using software that is Open Source. Is there something out there that works as well as NoMachine and supports so many different platforms? And, what can be done to simplify all this so that a ham with minimal Linux admin skills can make this work?

Acknowledgements

Thanks to Juan Manfredi, NAØB of the Panther Amateur Radio Club of the University of Pittsburgh; to *NoMachine* Tech Support for helping me configure the interface with *PulseAudio*; and to Dave Freese, W1HKJ, the leader of the NBEMS team and everyone else working on the project.

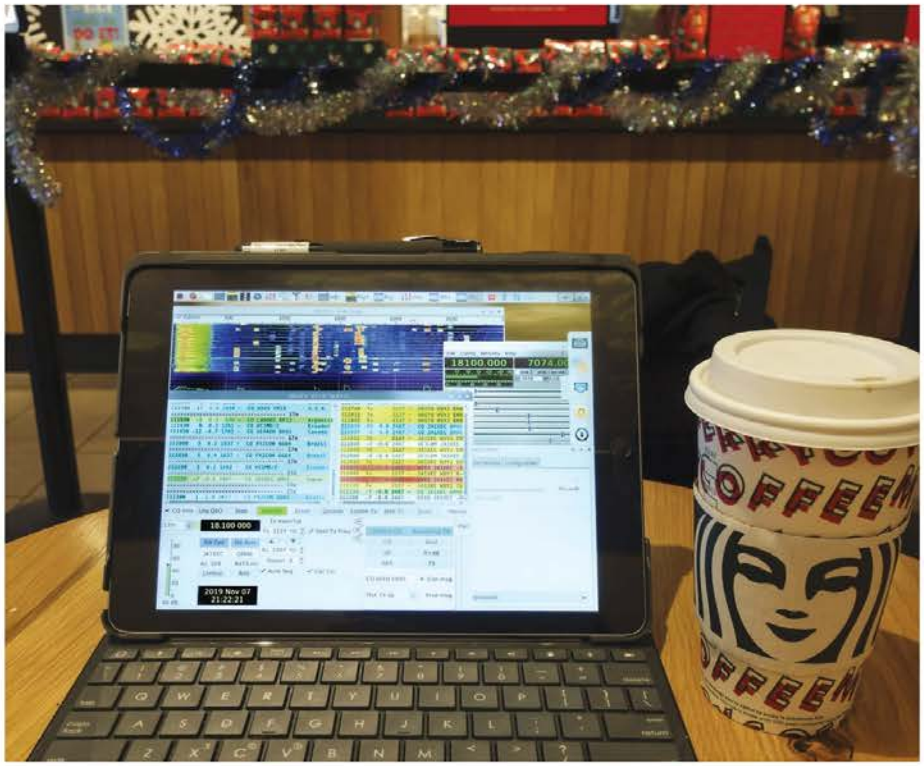


Figure 5 — Operating FT8 remotely from a coffee shop, before the current pandemic.

Harry Bloomberg, W3YJ, was first licensed in 1972 as WN3TBL. He recently retired after many years working as a software engineer. Harry graduated from the University of Pittsburgh in 1979 with a BSEE degree. He also holds a Masters of Mathematical Sciences degree from the University of Texas at Dallas. Harry is an alumni member of Panther Amateur Radio Club (PARC) at the University of Pittsburgh. He also belongs to Mercer County ARC and Skyview Radio Society. Harry enjoys CW contesting and working many digital modes. He has had three articles published in QST *about NBEMS and wrote a chapter in the* ARRL Public Service Communications Handbook *on NBEMS.*

Notes

[1] **https://sourceforge.net/p/fldigi/wiki/debian_howto/**
[2] **https://www.hamcrafters2.com/**.
[3] **www.w1hkj.com**.
[4] **https://nanokeyer.wordpress.com/**.
[5] Download the script from **www.w1hkj.com/W3YJ/write_script.tar.gz** or from **www.w1hkj.com/W3YJ/**.

Errata

In John E. Post, KA5GSQ, "Generation and Reception of Single-sideband Signals using GNU Radio Companion," *QEX* Sep./Oct. 2020:

In the last paragraph of the section, "Filter Methods of SSB Generation," the next to last sentence should read: "The result of passing the double-sided representation, **Figure 5**, through the sideband filters, **Figure 6**, is shown in **Figure 7(a)**…"

In the first paragraph of "Filter Method of SSB Reception", the last sentence should read: "The frequency xlating FIR block also includes a low pass filter whose width must be set to f_m to pass both sidebands shown in **Figure 18(a)**, or whose width must be set to $f_m/2$ to pass the sideband configuration shown in **Figure 18(b)**.

Equation **(A6)** should read:

$$\mathcal{F}^{-1}\left\{\operatorname{sgn} f\, M\left(f\right)\right\} = j\hat{m}\left(t\right)$$

In **Figure 8** the file name in the file source blocks should be "test_audio_36k". Delete "Cuttof Freq: 3k" in AGC2 block.

In **Figure 16** Set Gain = −1 for lower low pass filter. Arrow should enter File Sink Block.

In **Figure 20** sample rate for Band Pass Filter Block should be 36 KSPS.

In **Figure 26** Rcv_frequency WX GUI Slider Default Value should be 430.05M. Tx_attn WX GUI Chooser Choices should be 89.75, 50.