Lynn H. Hansen, KU7Q

10914 Westchester Dr., P.O. Box 301, Lava Hot Springs, ID 83246; **lynn@lynovation.com**

CTR2 – An HMI for All of Your Radios

The CTR2 Human-Machine-Interface is intended as a fixed station accessory providing control and audio management for the large variety of radios and components in your shack.

To be useful, machines usually have ways for humans to interact with them. From a simple ON/OFF switch to controlling a machine using a modern touch panel controller, the purpose of machine control is to allow interaction with humans. Machine control systems are commonly referred to as Human Machine Interfaces (HMIs). So it is in amateur radio. Every radio on the market has an HMI of sorts. For years, the radio front panel with its dials and switches served as its HMI. The latest radios feature touch screen controls with virtual dials and buttons replacing dials and switches. Spectrum displays are the norm rather than an expensive accessory.

Ever since the first PC hit the market, hams have been creating new HMIs in software for their radios. Even if you don't own the latest radio, your PC can give your old radio modern features. But what if you're like me with radios of various vintages scattered around your shack? Or maybe you've invested heavily into high-end keys, microphones and headsets, and primarily use one radio because it's such a hassle to disconnect and reconnect these accessories? What about the new digital modes? Wouldn't it be nice to use any radio in your shack using a single PC USB cable?

Figure 1 — The basic multi-radio CTR2 system consists of the HMI board, an RJ45 adapter board, a color remote display, a manual RJ45 switch, and a couple of radio I/O modules.

Some History

In 2001 I created a Pocket PC program I called Control The Radio (CTR). Along with that software, I designed a Bluetooth® serial interface named BlueLync that allowed you to control your radio wirelessly. CTR and BlueLync were published in the February 2007 issue of *QST*. The software could control of any number of BlueLync equipped radios by simply connecting to the radio of interest. After using CTR for a while I was struck by just how useful a common user interface was for all of my radios. Unfortunately, the Pocket PC, CTR, and BlueLync became dinosaurs after the iPhone® came out later that year.

I really liked the convenience of a common user interface with its shared frequency list. I thought about updating CTR

and BlueLync to work with the newer cell phone technologies but decided against it. By then there were already many radio control and logging programs available for these devices. While wireless connectivity is a neat party trick I ultimately decided that Bluetooth added an unnecessary layer of complexity. I wanted something modern, simple to expand, that took advantage of the latest technologies, managed audio — which the original CTR did not do — provided digital signal processing, had all the normal options like keyers and decoders, and wasn't integrated into a cell phone, tablet, or PC. I also wanted to use it on all of my radios from my 35 year old TS-680 to my new FTDX101.

The Radio Multiplexer

Over the years the thought of a common user interface still resonated with me so I set out to create a full-featured standalone HMI that could interface with all of my radios — a radio multiplexer if you will. Advancements in microprocessor technology offer interface and control options that I only dreamt of 20 years ago. I call my HMI *Control The Radio Too* (CTR2) because controlling the radio is just one of many functions it performs. It's a self-contained radio HMI that manages up to 16 different radios and allows me to use a single key, microphone, headphone, and computer (if needed) with all of my radios. Each radio looks the same as far as controlling its basic features (frequency, mode, Tx and Rx audio). **Figure 1** shows the basic multi-radio CTR2 system consisting of the HMI board, an RJ45 adapter board, a color remote display, a manual RJ45 switch, and a couple of radio I/O modules. **Figure 2** shows a functional drawing of this system.

CTR2 uses a modular design approach. You build only what you need now. Additional features can be added later as your needs expand. Option boards stack on top of the HMI board and interconnections are done using 10-wire ribbon cables. There are currently four option boards:

1) An RJ45 adapter board. This board adapts the 10-pin ribbon cable to an RJ45 jack so a manual RJ45 switch can connect to the HMI as seen in **Figure 1**.

2) A four-to-sixteen port RJ45 switch that automatically routes the radio I/O (audio, serial CAT control, PTT, and key) from the selected radio to the HMI.

3) A radio antenna switch controller that controls one or two 8-port commercial antenna switches (such as the DX Engineering RR8B-HP) to automatically route a common antenna to the selected radio.

4) An antenna switch controller that controls an 8-port commercial antenna switch, which automatically selects one or multiple antennas (for phased arrays) and routes them to the common antenna port. A different antenna can be selected for each band. A fully configured CTR2 system is shown in **Figure A** on the **www.arrl.org/QEXfiles** web page.

The CTR2 HMI

The CTR2 HMI is totally self-contained and does not require an external computer to operate. At its core is a Teensy 4.1 development board from **PJRC.com [1]**. This board contains a 600 MHz ARM Cortex-M7 processor, 1 GB RAM, 2 GB of Flash memory, a micro-USB interface, an SD memory slot, and host of serial ports and other interfaces. An optional ESP8266 can be installed in the HMI to support Wi-Fi connectivity. Wired Ethernet is also available via an expansion kit, but I have not yet exploited that feature.

Audio is managed by the Teensy Audio Adapter **[2]**. This little marvel includes the NXP SGTL5000 codec that converts audio to and from digital. Once in digital form the chip provides mixers, amplifiers, oscillators, and signal routing via virtual patch cords. **PJRC.com** provides the Arduino libraries to support the SGTL500 chip but they've gone one better. They provide an audio design tool **[3]** that allows you to easily configure the SGTL5000 functions without writing a single line of code. **Figure B** on the **QEXFiles** web page is a screenshot of the CTR2 audio interface as designed on the PJRC audio design tool. If you're interested in interfacing audio to microprocessors, you should check out the PJRC web site. Paul Stoffregen at **PJRC.com** and his online community have done an amazing job creating a microcontroller/audio ecosystem that just about anyone can use.

A Nextion enhanced serial display **[4]** connects to the HMI via CAT5 cable. This allows the display to be placed in the most convenient location on your operating style.

Two programs run concurrently to create the HMI. The main program runs on the

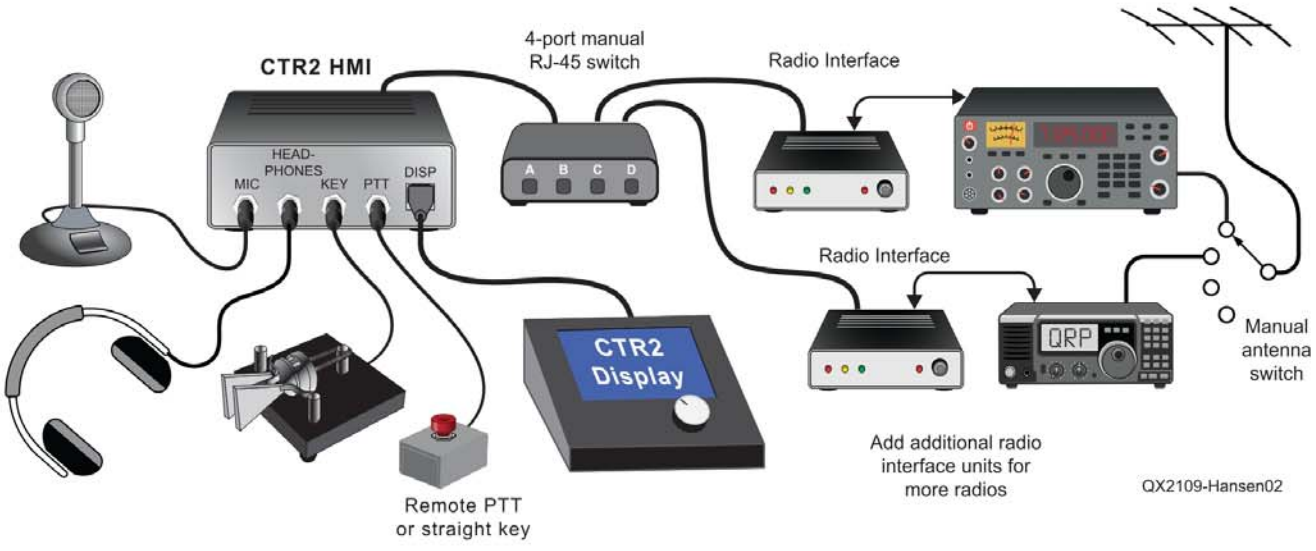

Figure 2 — A functional drawing of the basic CTR2 system.

Figure 3 — The Home page is where most of the radio interaction occurs.

Figure 4 — Screen shot of the Radio Select page.

Teensy processor. This program is written in Arduino code using the Teensyduino add-on available at **PJRC.com** and is open source. It handles the time sensitive tasks such as Morse encoding and decoding, file management, etc. The second program runs on the Nextion display and was created using the Nextion Editor **[4]**. It uses a unique C-like coding environment and is extremely powerful. The display program handles the user interface including page rendering and touch control. The two programs communicate continuously across the serial link to stay in sync with each other. Source code for both programs is available on the **QEXFiles** web and will be hosted on GitHub in the future.

The user interface is page-based and operates like a browser. The *Home* page (**Figure 3**) is where most of the radio interaction occurs. Pressing the [Back] button of the other pages returns you to the *Home* page.

The top line in **Figure 3** controls the transmit buffer system in CW and RTTY modes. Pressing the [Keyboard] button in the top left opens a virtual keyboard that can be used to compose messages that are transmitted when the [Enter] key is pressed. Touching the [Menu] button next to the [Keyboard] button opens the *Tx Buffer* page where you can select, edit, or delete the ten messages for each mode (including voice mode). Touch [Select] in the *Tx Buffer* page then touch a message to transmit it. In addition, there are ten buttons across the top of the *Home* page that instantly transmit that buffer when pressed. Once a message starts transmitting you can pause and restart it, or touch the [X] button to delete it. The Rx buffer is on the second line. This is where decoded CW or RTTY text is displayed. Touch the Rx message area to display the full 512 character buffer.

The third line starts with the [Radio Select] button (R2:A4 in the photo). R2:A4 means Radio 2 and Antenna 4 are selected and a red background indicates the radio is using the shared database. **Figure 4** is a screen shot of the **Radio Select** page. Each radio port can be named. Red text indicates that port is using the shared database. Back on the *Home* page the name of the radio is displayed at left-center, below the date and time. The rest of the third line contains the log buttons. The [Auto>] button parses the Rx buffer to find anything that looks like a call sign or RST report. Simply touch the desired data to add it to the log entry.

The audio spectrum display and gain controls are immediately below the third line. The spectrum displays the upper or lower sideband from 100 to 3200 Hz. In CW and RTTY modes, tuning cursors are displayed to assist in tuning the decoders. Filter cursors are also displayed here if either the low-pass or notch filters have been moved into the audio passband. The notch filter follows tuning changes so it remains on the interfering carrier. Touching a signal on the FFT display retunes the radio to place that signal approximately in the decoder's range. Selecting the low-pass or notch filter on the encoder, then touching the display places the selected filter at that location.

The [<] and [>] buttons on either side of the FFT display allow you to step through the 100 memory channels. The sliders on the right adjust the FFT gain, volume, and audio squelch level. More about the squelch control later.

The three buttons at left-center allow you to select the operating band (and radio model), radio mode, and operating mode. The frequency is displayed in the center. You can adjust frequency by clicking the top or bottom of any digit, rotating the encoder (changes by the selected step – the digit shown in red), or pressing the [Keypad] direct entry button on the next line down. Of course you can also turn the radio dial too! The [Lock] and [A/b] VFO, and [>0] at right-center allow you to lock the display, toggle A/B VFOs, and set all digits to the right of the selected digit to 0.

The second to last line contain VFO/Memory functions. The [V<P] sends the previous frequency and mode to the VFO. This is handy when you tune off frequency or switch radios and want to return to the frequency and mode you had selected previously. The [V>M] (VFO to Memory) button allows you to save the current frequency and mode to the favorites list. The [Keypad] button opens the frequency input page. The [V<M] button allows you to select a

Figure 5 — Screen shot of the favorites list page.

frequency from memory and mode from your favorites list. **Figure 5** shows a screen shot of the favorites list page.

The [Scan] button allows you to specify the scan type, frequency or memory. In frequency scan you can define the range, step, and speed. In memory scan you can choose the start and end memory location.

The options menu is on the bottom line. The bottom right button is the Transmit Enable. When set to [Tx Off] you can use the paddle to practice code. In **Figure 3** [Iambic-B] indicates the paddle mode selected and that Transmit is enabled. Pressing the key will send the code to the transmitter and turn this button red. The [Settings] button takes you to the setting page for the selected mode. You can also edit the settings of other modes while in this page. [Log] displays a summary of your log. Touching the [Freq] button cycles through the functions of the rotary encoder. As each function is selected its associated control on the display is highlighted in red. Turn the encoder to change the selected control's settings. Finally, [Help] opens a help page that contains useful program information, common Q-signals, and the phonetic alphabet.

What Else Does It Do?

The CTR2 HMI also includes the following.

• It can be powered from any voltage source from 9 to 36 V dc as designed. 18 to 72 V dc can be used if the 18 to 72 V dc/dc converter is installed. It needs at least 300 mA at 12 V. Polarity is normally negative ground but cutting JP2 allows positive grounded or floating battery systems to be used.

• It supports Icom, Kenwood, and most Yaesu radio protocols. It also supports the Icom PCR-1000 receiver and the Flex 6000 series radios. It only controls frequency and mode so most implementations of these protocols by other manufacturers work as well. Options like S-meters, power control, etc., vary widely among radios so I chose not to support them.

• The rotary encoder normally tunes the frequency by the selected frequency step. Pressing it steps through various options that can also be adjusted. In voice, RTTY, and digital modes the encoder allows you to change the Tx audio level during transmit by simply turning the encoder. This is handy in digital modes like FT8 where power levels change as you move around the pass band. In CW transmit mode, turning the encoder during transmit changes the keyer speed — a very handy feature.

• Touching the spectrum display moves the selected signal to the tune decoder cursor and automatically shifts the tune frequency step to 10 Hz for fine tuning CW and RTTY signals with the encoder.

• A USB-A host port supports a USB keyboard and mouse. The keyboard works with the CW and RTTY keyers and the mouse emulates the rotary encoder for tuning and option selection. A virtual keyboard is included so a physical keyboard is not required.

• Each radio port has its own settings unique to that radio. There is also a common database of favorite frequencies, band stacking registers, antenna configurations, and transmit buffers that all radios can share. Settings can be copied to other radio ports.

• Basic standalone logging is supported, including contest sequence numbering and exchange text. The log is saved in ADIF format on the SD card and can be downloaded via the built-in web server.

• The HMI can create a Wi-Fi access point or connect to your local Wi-Fi network. This feature is not fully developed yet but a simple web server is functional. A Wi-Fi network is required to control a Flex radio.

• CW mode supports straight key, pass-through, Iambic-A and B, Ultimatic, and Bug modes. Pass-through allows you to use the selected radio's keyer instead of the built-in keyer in CTR2. Ten transmit CW buffers are available per radio port plus ten more in the shared database. CW mode has its own adjustable low-pass and notch filters and receive audio is emphasized around the selected side tone frequency.

• In voice mode 5-band equalizers are available for Tx and Rx audio. Voice mode has its own adjustable Tx level, low-pass and notch filters. A ten buffer voice keyer is also included. Message length is limited only by the size of the SD card. The shared database includes ten more voice buffers.

• RTTY mode supports, 45.45, 50, 75, and 100 baud with 170, 225, 425, 450, and 850 Hz carrier shifts. Ten transmit RTTY buffers are available per radio port with ten more in the shared database. RTTY mode has its own adjustable Tx level, low pass and notch filters and receive audio is enhanced around the higher carrier frequencies.

• A computer is required for digital modes such as PSK31 and WSJT-X. In digital mode the HMI emulates a USB sound card and provides a USB serial interface. Configure your digital program to use the Teensy USB audio interface for Tx and Rx then set the Tx level. For third-party applications the HMI appears as a Kenwood TS2000 radio on the USB serial port. Your program controls the frequency and mode of the HMI using the Kenwood protocol. The HMI in turn will control whatever radio is selected on the radio interface side using that radio's control protocol. The HMI also allows you to monitor transmit and receive audio on the USB port, which is nice when you just want to listen to what's going on.

• Off-air recording is supported in all modes. Receive recordings are limited to the size of your SD card. The maximum SD card size is 32 gigabytes.

• When the optional antenna control board is installed, the HMI can automatically select a different antenna (or antennas in the case of phased arrays) for each band.

• In Beacon mode the HMI displays the currently active beacon from the International Beacon project **[5]** on the selected band. The internal real-time clock should be calibrated with WWV for this to work properly. Beacon frequencies are pre-programmed in memory slots 91 to 95.

• A WWV decode option allows the internal clock to automatically synchronize with the IRIG time codes broadcast by WWV/WWVH **[6]**. WWV frequencies are pre-programmed in memory slots 96 to 100. Due to fading and distortion, synchronization may take some time.

• CTR2 provides a VOX circuit with an adjustable threshold. This is great for voice, RTTY, and digital modes since no additional keying circuit is required for your radio. You can also set VOX to just pass Tx audio to your radio and use your radio's VOX circuit or an external PTT if you desire. ***CAUTION:*** *Windows may switch USB audio ports on its own. If it switches the microphone input to the Teensy board and HMI's VOX is active, audio from your PC will be transmitted over the air. You should disable transmit mode on the HMI when not using it.*

• Finally, CTR2 has an adjustable all mode audio squelch that allows you to mute Rx audio until the band gets busy. This helps alleviate the fatigue that sets in when listening to band noise all the time. This is similar to using squelch on FM but it works when the audio level increases.

Design Considerations

Circuit schematics and bill of materials for all boards are posted on the **QEXfiles** web page. While the design is quite straight-

forward, there are a few interesting features to note. First, I use CAT5 cable to connect the radio interfaces to the HMI. This allows use of readily available, inexpensive cables. However, this created two problems.

First, operators can get confused with what plugs into what. The HMI *does not* use an Ethernet interface to the radio I/O modules or the display. Never plug a CAT5 cable from the HMI into an Ethernet device. Wires 7 and 8 on the HMI interconnection are used for +5 V dc and ground. Therefore, if you use Power over Ethernet (PoE) devices and plug an HMI cable into your network you could run into trouble. *I suggest clearly labeling CTR2 cables to minimize this possibility.* Also note that the RJ45 jack for the remote display is on the front of the HMI, next to the 3.5 mm audio and keying jacks. Radio I/O RJ45 jacks exit the rear of the HMI.

Second, I wanted balanced audio (i.e. transformer isolation) between the radios and the HMI for noise immunity. To squeeze Line-In and Line-Out audio (4 wires), +5 V dc and ground (2 wires), transmit and receive serial (CAT) data (2 wires), PTT and Key signals (2 wires) into 8 wires I used a trick I learned years ago in the telecom industry. Using CT (center tapped) transformers on each end of the audio paths provides two extra wires because the normal audio paths are balanced through the transformers (i.e. have no ground reference). The CT of the transformer makes the audio pair look like a single wire. The two signals, audio and dc control, are independent of each other. **Figure 6** is a simplified diagram of how this is done.

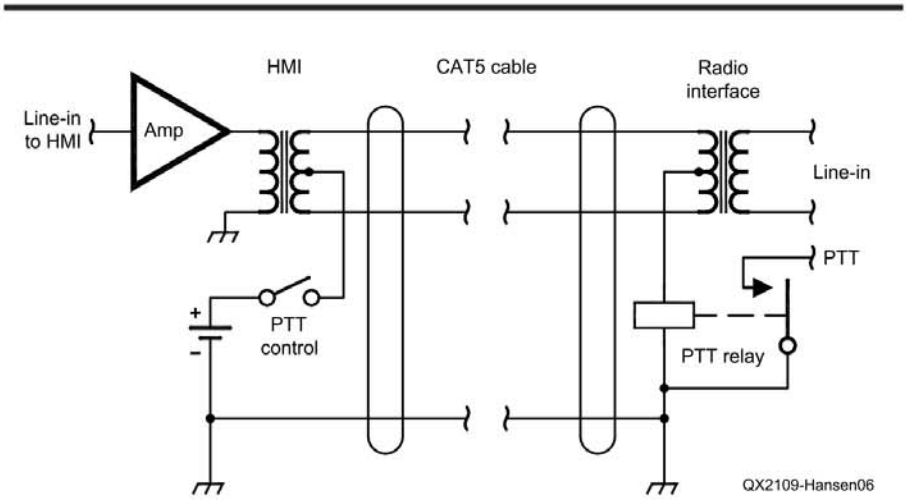

Figure 6 — Simplified interconnection diagram.

Radio I/O Modules

CAT serial port level conversion is done in the radio I/O module. This makes the HMI radio port agnostic to the electrical interface of the radio it's connected to. Early in the digital age the 'big three' radio manufacturers each developed their own unique serial interfaces. Over time, it's just become a headache for the owners. There are three standards. Icom has the CI-V interface, which is a two-wire bus scheme that allows multiple radios to share the same connection on a single pair of wires. Kenwood has their IF-10C interface that provides a non-inverted three-wire TTL interface, which interfaced to yet another add-on (IF-232C) to provide RS232 levels. The radio I/O module can interface directly to the IF-10C. Yaesu used a three-wire non-inverted TTL method similar to the Kenwood IF-10C. In time, most manufacturers just installed RS-232 ports on their radios. The design of the CTR2 radio I/O module accommodates all three methods through a switch matrix similar to what I used on CTR-BlueLync. Even though it is 5 V based, it will work on RS-232 lines using the inverted settings, as long as your RS-232 line receiver doesn't expect the voltage to go below zero volts.

ULN2803A Darlington transistor arrays are used as line drivers for all control signals on the radio interconnect cabling. This provides isolation for the microprocessor's I/O pins and allows signal inversion when needed in the level converter. They also drive the LEDs built into the RJ45 jacks. On the HMI end, the RJ45 LEDs indicate Tx and Rx data to and from the radio on the control port. Both flashing is a good sign. On the radio interface end, the RJ45's LEDs indicate PTT and Key output status (LED is ON when active).

Reed relays drive the PTT and Key output lines to the radio. The relays can switch up to 200 V at 0.5 A, any polarity. There is a 1 ms delay on the relay close and open operations. Jumpers can be installed to bypass the relays and use the output of the ULN2803A instead. It can switch up to 50 V dc, 0.5 A, positive polarity only. There is no delay in the ULN2803A output.

I used high-frequency bypass capacitors and ferrite beads liberally to eliminate RFI problems. If you experience RFI you may need to add additional ferrite chokes to all the interface lines.

Let's Build It

This project uses surface mount devices (SMDs) so the only viable construction method is with pre-populated PCBs. You can build a temporary system on a breadboard if you want to try it out or experiment with it. **Figure 7** shows the project in the early stages of development. Bread-boarding is a good way to prove a concept, but you'll find that audio and digital signals don't play nicely together on flying wires. In this configuration, simply moving a wire affects the received noise floor.

Perf-board construction is not recommended even if you use through-hole components because most of the connectors are not perf-board friendly.

I can supply PCBs with the SMD components already installed. All that's needed is to add the connectors and header pin jacks. Because board prices drop drastically in quantity, I'll queue orders until I can order 50 at a time. My web page will have ordering information. **Figure 8** shows a close-up of the HMI and the RJ45 port adapter PCBs.

Automated Bill of Material (BoM) files for the boards are available for download from the **QEXfiles** web page. These files run in your browser and make it easy to order the parts or build each board. **Figure C on QEXFiles** shows the front of the HMI board in the automated BoM. Click the [B] button to switch to the back side. There are columns to track sourcing parts and placing them. As you move the mouse cursor over each part on the list, the corresponding parts on the board are highlighted showing their location. You can also click on a device on the board and it will be highlighted on the part list.

On the HMI, install jumper JP3 on the

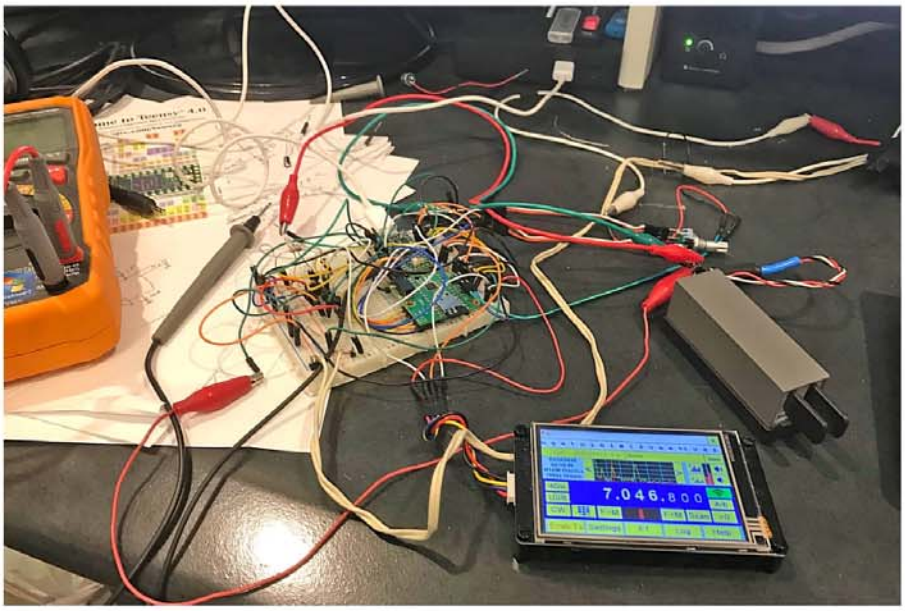

Figure 7 — The project in the early stages of development.

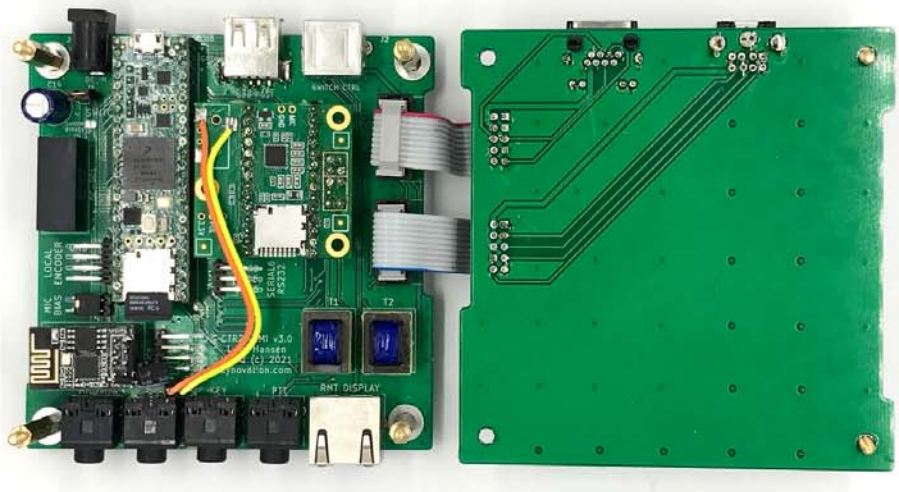

Figure 8 — A close-up of the HMI and the RJ45 port adapter boards.

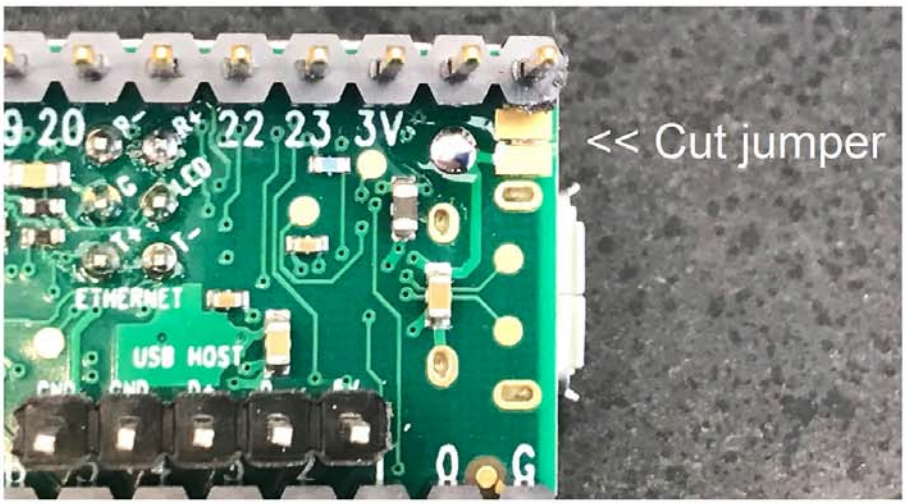

Figure 9 — Isolate the Teensy from USB power by cutting the indicated jumper.

Mic Bias header if you are using an electret microphone. Header pins J9 allow you to connect an external switch on the HMI. Solder across jumper JP1 to bypass the external switch.

You must build the RJ45 adapter board to use the HMI with a manual RJ45 switch. This board is not required if you install the automatic RJ45 switch option.

The display driver board just requires the RJ45 connector and pin headers to be installed. On the radio I/O module, leave JP1 open and install RV1 if you want an adjustable attenuator on Line-In. Insert a jumper on JP2 and install R4 (8 to 32 Ω, 2 W) to add a speaker termination resistor if you will be using the speaker jack on the radio for Line-In. Don't install the relays, and solder across JP3 and JP4 if you want to use solid-state keying. Header J1 allows you to select +5 V dc or ground on pin 8 of the terminal block. The +5 V dc source is current limited to about 40 mA by 120 Ω resistor R8.

A 12-pin pluggable terminal strip is used for the radio interface. This provides a universal interface but you'll need to build custom cables to fit the radios you are using. The Line-In and Line-Out lines are transformer isolated and dc blocked.

A Word about Power

CTR2 is intended to be a fixed station accessory providing control and audio management for the large variety of radios in your shack. It has a large input voltage range of 9 to 36 V and requires up to 500 mA at 12 V dc. It can even run on –24 V dc or –48 V dc if you cut JP2 to isolate the power input negative lead from ground. If –48 V dc operation is desired, replace the EC4SAW-24S05N dc-dc converter with an EC4SAW-48S05N, increase the voltage ratings of C16, C17, and C18 to 75 V dc, and cut JP2.

Nominally the system pulls around 240 mA on a 12 V supply (450 mA on the +5 V dc side of the converter). This can briefly spike to over 800 mA on the 5 V side when the ESP8266 Wi-Fi module is transmitting. While it is tempting to power the Teensy from the USB connection to your PC, the PC USB port is not able to supply the current required to power all of HMI options such the Wi-Fi, the 5-inch display, and the optional CTR2 switches. It is also not advisable to have the USB 5 V power connected when the Teensy is being powered externally. To isolate the Teensy from USB power turn the Teensy 4.1 board over and cut the jumper (**Figure 9**) between the pads next to the +5 V pin.

Packaging It

The CTR2 system was designed so that additional option boards such as the automated RJ45 switch and antenna controller boards stack on top of the HMI using 25 mm standoffs. This forms a self-supporting framework that doesn't require an enclosure. The display enclosure is a simple wedge shape cut from acrylic.

CNC files are available for the cube and display enclosures. They were created in the Shapeoko Carbide Create program **[7]**. Even if you don't have access to a CNC mill, these files will give you the dimensions needed to cut the cases.

I can supply a limited number of pre-cut display enclosures.

The radio I/O module is mounted in a PacTec 115129 CN-XRL housing (**Figure 10**). One end is cut to fit the RJ45 jack and the other end exposes the 12-pin pluggable terminal connector used to wire the radio interface. **Figure 11** shows a close-up of the radio I/O terminal block wiring. A #14 AWG copper wire is folded back and connected to two ground pins to form a strain relief.

Figure 10 — The radio I/O module is mounted in a PacTec 115129 CN-XRL housing.

Firmware

The firmware for this project is open source and is licensed for non-commercial use only. Contact me if you have any questions regarding this licensing. Understanding the code might be a little daunting at first since there are two separate systems interchanging information with each other. I've tried to document the functions in the code in my own style. Yours will undoubtedly be different than mine. Reviewing the code will give you an idea of what it takes to create a device like the HMI.

Source code files are available from he **QEXfiles** web page. I will eventually place these on GitHub repositories. There are three sets of source code files, one for the Nextion 3.5-inch display, one for the Nextion 5-inch display, and one for the Teensy 4.1 board.

The source files can be modified in any text editor. All options are free for individual developers and are available for Windows, Mac, and Linux (except Visual Studio). You must install the Arduino IDE to download the binary file to the Teensy board.

The Nextion display firmware requires the Nextion Editor to modify and compile programs for the displays. This is a free Windows program **[8]**, which provides drag and drop visual editing of the HMI screens. The binary file (.tft) from this program is then copied to an SD card that is then inserted into the SD card slot on the display. At power-up the display detects this file and updates the onboard firmware. This is an easy option for customizing any of the displays to your liking.

The project's firmware is not mature and not necessarily bug-free. This is a learning experience for me. It works as intended but there is a lot of room for improvement. Some of the features are experimental and not fully debugged. A forum user's tag line says it all; "Hardware eventually fails. Software eventually works." I plan to continue pushing my limits and welcome suggestions and help from others.

Final Comments

Future plans include developing the Wi-Fi and Ethernet interfaces. This would allow the HMI to be controlled remotely from anywhere. My desire was to design and build a custom expandable radio HMI. I now have a test bed to experiment with new ideas.

Much of my inspiration comes from the pages of *QST* and *QEX* magazines. The inspiration for this project came from an article by Rick Dubbs, WW9JD, in the April 2020 issue of *QST*. Rick described the keyer he designed using the Nextion display.

I would like to thank my son, Matt, KE7AQM, for his help and perseverance in converting me to KiCad 3D modeling and

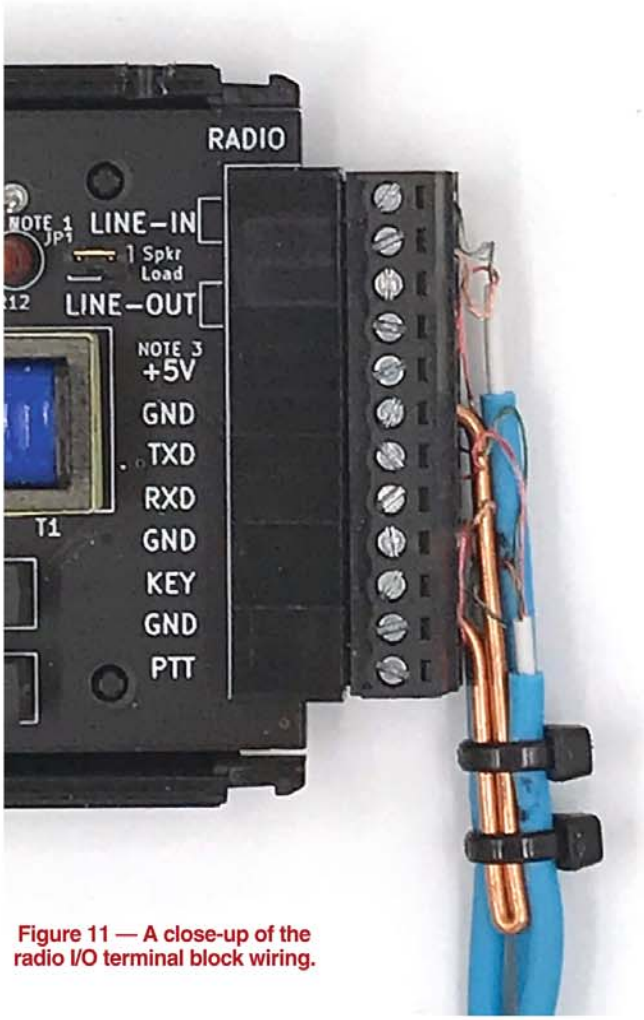

Figure 11 — A close-up of the radio I/O terminal block wiring.

the automated BOM.

I plan on offering board sets and I can cut a limited number of display and HMI enclosures on my CNC if there's enough interest. A user's manual and additional information can be found on my web page **[9]**.

ARRL member Lynn Hansen, KU7Q, was first licensed in 1971 as WN7QYG at age 14. Within a year he upgraded to WA7QYG, then achieved the Amateur Extra class license in 1981 while studying for his Commercial Class Radio Telephone license. His amateur radio experience and several Cleveland Institute of Electronics home-study courses enabled him to follow a life-long passion of working in electronics and communications as a communications technician, an engineering assistant, and finally as an operations manager over a large multi-state utility communications network. Now retired, he has the time to follow his passions, such as this project.

References

[1] **https://www.pjrc.com/store/teensy41.html**.

[2] **https://www.pjrc.com/store/teensy3_audio.html**.

[3] **https://www.pjrc.com/teensy/gui/**.

[4] **https://nextion.tech/enhanced-series-introduction/**.

[5] **https://www.ncdxf.org/beacon/**.

[6] **https://www.nist.gov/pml/time-and-frequency-division/radio-stations/wwv/wwv-and-wwvh-digital-time-code-and-broadcast**.

[7] Free download at; **https://carbide3d.com/carbidecreate/**.

[8] **https://nextion.tech/nextion-editor/**.

[9] **www.lynovation.com/**.

Letters

Analysis, Design and Verification of SA602A IC Hartley Oscillators, (May/June 2021)

Dear Editor,

John E. Post, KA5GSQ, shows a very interesting use of the *Spice* program for an *LC* oscillator at about 1 MHz. I will address some circuit improvements that are based on analysis in *Ansoft Harmonica*. I opted to take the RF power from the collector, which is not atypical. The phase noise and output power can both improve significantly. Referring to Post Figure 2 with component values of Figure 3, I reduced the emitter resistor R_E from 20,000 Ω to 2,000 Ω, reduced the coupling capacitors C_{bypass} to the tuned circuit from 1 µF down to 5 pF, and interchanged the inductor values of *L* and *nL* so that the smaller inductance is on the bottom. This reduces the loading on the base of transistor *Q2*, which according to my *Ansoft Harmonica* simulation not only gets the output power up by more than 10 dB, but improves the phase noise by more than 20 dB. Thus, an 800 kHz oscillator phase noise is –140 dBc/Hz at 1 kHz offset, and an 80 MHz oscillator phase noise is –130 dBc/Hz at 10 kHz offset. This kind of analysis can not be done in *Spice* — *Best regards*, *Ulrich L. Rohde, N1UL*; **dr.ulrich.l.rohde@gmail.com**.

From the Editor,

Thank you for your observation that different analysis programs can lead to different optimization opportunities for oscillator circuits. — *73, QEX Editor.*

Send your *QEX* Letter to the Editor, via email to qex@arrl.org. We reserve the right to edit your letter for clarity, and to fit in the available page space. *QEX* Letters may also appear in other ARRL media. The publishers of *QEX* assume no responsibilities for statements made by correspondents.