YaesuControl

A Python Software Object

Background

- Primary Goal: Create a software object that could be used:
  - By individuals wanting to create “Rig Control” software for use in Go-Kits
  - On the popular Raspberry Pi computers
  - In a way that allowed it to run if accessed remotely
    - In Amateur Radio, this is done via “Broadband-Hamnet”

Background

- Secondary Goal: Create an Interface program that could:
  - Demonstrate the features of the software object
  - Run on the popular Raspberry Pi computers
  - Simulate the basic operation of the Transceiver

Background

- Secondary Goal: Demonstrate the software on hardware that would most likely be used to impliment it.
  - Raspberry Pi
  - WRT54G Router running “Broadband-Hamnet”
  - All running on 12volts (to simulate operation on battery power)

Requirements

- Locking and unlocking the keypad on the radio's control head.
- Engaging and disengaging the radio's push-to-talk circuitry, effectively switching the radio between transmit and receive.
- Directly setting the operating frequency of the transceiver.
- Directly setting the operating mode of the transceiver.
- Engaging and disengaging the radio's clarifier function.
- Directly setting the clarifier offset direction and frequency.
- Toggling between the radio's two variable frequency oscillators, referred to as VFO-A and VFO-B.
- Engaging and disengaging the radio's split operating mode function, allowing transmitting and receiving on two different frequencies.
- Setting the radio's repeater offset direction (plus, minus, or simplex.)
- Directly setting the radio's repeater offset frequency.
- Engaging and disengaging the radios
- CTCSS and DCS encoders and decoders.
- Directly setting the radio's CTCSS tone.
- Directly setting the radio's DCS code.
- Reading the current operating frequency and mode from the radio.
- Increasing and decreasing the operating frequency by logical "step" amounts
- Setting up transmit and receive frequencies along with engaging split operation
- Setting up operation for a known repeater
- Switching between "bands" and setting their standard modes (i.e. choosing 20 meter SSB changes the frequency to 14.150 and sets the operating mode to USB)

Code Highlights

- Radio Commands sent in Hexadecimal.
- 5 byte blocks:

  0x00 0x00 0x00 0x00 0x00
- Latin Encoding

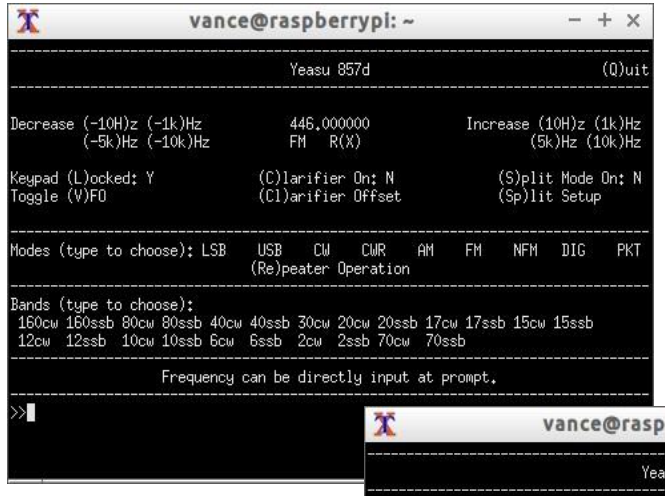
yaesuControl.py - /home/vance/yaesuControl.py

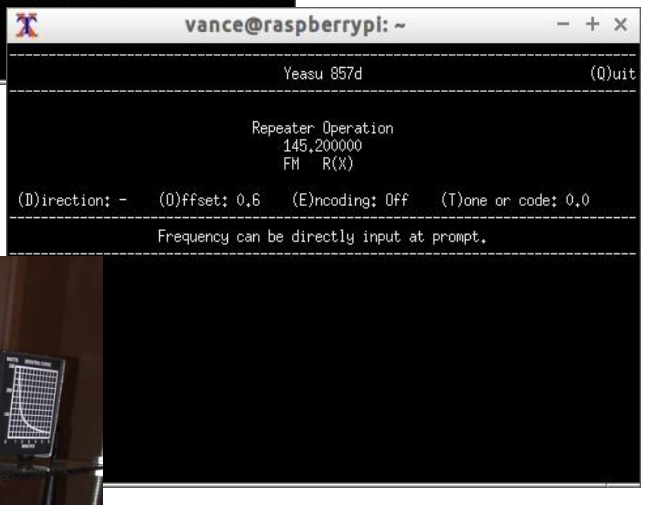
File Edit Format Run Options Windows Help

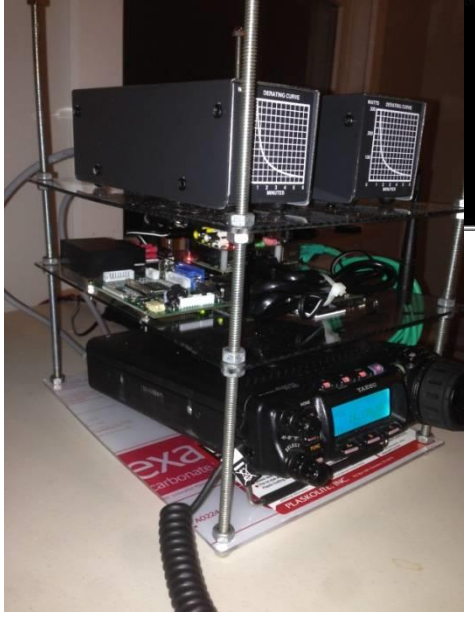
```
def setFrequency(self, frequency):
    #This function takes a float representing the frequency in MHz
    #(i.e. 14.225) and sets the radio to that frequency.  The frequency
    #must be in the Amateure allotment, from 1.8 - 450Mhz

    if frequency > 1.79999 and frequency < 450.00001:
        freq = '%09.05f' % frequency
            #formats the frequency float to the correct precision
            #and format
        freq = (freq[:3] + freq[4:])
            #takes out the '.' in preperation for creading the hex string

        commandbytes = []
        for i in range (0, len(freq), 2):
            #steps through freq and makes the hex string that the
            #radio needs
            commandbytes.append(chr(int(freq[i:i+2], 16)))

        command = ''.join(commandbytes).encode('latin-1')
            #creates the hex command, minus the final control character
            #latin encoding used here because UTF incorrectly codes some
            #some of the radio commnds, resulting in extra characters

        self.radioConnection.write(command)
            #writes the command to the radio
        self.radioConnection.write(b'\x01')
            #writes the control command for change frequency

        response = self.radioConnection.read()
```

Final Product(s)

- Project Proposal
- Project Plan
- Status Report
- yeasuControl.py
- radioInterface3.py
- yeasuControl class documentation
- radioInterface3 documentation







Demo &
Q&A