```
# Please do not change the configuration file
quisk_conf_defaults.py.
# Instead copy one of the other quisk_conf_*.py files to
your own
# .quisk_conf.py and make changes there. For a
normal sound card
# configuration, copy quisk_conf_model.py to
your .quisk_conf.py.
#
# Quisk imports quisk_conf_defaults to set its
configuration.
# If you have a configuration file, it then overwrites the
defaults
# with your parameters. Your configuration file must be
named
# ~/.quisk_conf.py, where "~" means your home
directory. Or
# you may specify a different name with the -c or –
config command
# line option. Try –help. Check the config screen to
make sure that
# the correct configuration file is in use.
#
# The Quisk receiver can use a high quality sound card
for capture and playback,
# or it can use the SDR-IQ by RfSpace for capture and
a lower quality
# sound card for playback. Quisk can also be used as a
```

```
panadapter.
# Quisk can control some rigs. See quisk_hardware_*.py. If you have a rig
# to control, copy one of the quisk_hardware_*.py files to your own file named
# quisk_hardware.py, and edit that file. If there is no quisk_hardware.py, then
# quisk_hardware_model.py is used instead.
import sys
# Import the default Hardware module. You can import a different module in
# your .quisk_conf.py.
#import quisk_hardware_model as quisk_hardware
from softrock import hardware_usb as quisk_hardware
usb_vendor_id = 0x16c0
usb_product_id = 0x05dc
softrock_model = "RxEnsemble2"
# Module for additional widgets (advanced usage).
quisk_widgets = None
# Select the default screen when Quisk starts:
#default_screen = 'Graph'
default_screen = 'WFall'
#default_screen = 'Config'
# The width of the graph data as a fraction of the total screen size. This
# will be adjusted by Quisk to accommodate preferred FFT sizes. It can
# not be changed once Quisk starts. It can not be made
```

too small because
# of the space needed for all the buttons.
graph_width = 0.8
 # This controls the speed of the graph peak hold. Lower numbers give a longer time constant.
graph_peak_hold_1 = 0.25
graph_peak_hold_2 = 0.10
 # Select the default mode when Quisk starts (overruled by persistent_state):
 # default_mode = 'FM'
default_mode = 'LSB'
 # Select the way the waterfall screen scrolls:
 # waterfall_scroll_mode = 0 # scroll at a constant rate.
waterfall_scroll_mode = 0 # scroll faster at the top so that a new signal appears sooner.
 # Select the initial size in pixels (minimum 1) of the graph at the top of the waterfall.
waterfall_graph_size = 135
 # These are the initial values for the Y-scale and Y-zero sliders for each screen.
 # The sliders go from zero to 160.
graph_y_scale = 100
graph_y_zero = 0
waterfall_y_scale = 20
waterfall_y_zero = 0
waterfall_graph_y_scale = 100
waterfall_graph_y_zero = 60
scope_y_scale = 80

scope_y_zero = 0 # Currently doesn't do anything
filter_y_scale = 90
filter_y_zero = 0
 # Quisk can save its current state in a file on exit, and restore it when you restart.
 # State includes band, frequency and mode, but not every item of state (not screen).
 # The file is .quisk_init.pkl in the same directory as your config file. If this file
 # becomes corrupted, just delete it and it will be reconstructed.
#persistent_state = False
persistent_state = True
 # This converts from dB to S-units for the S-meter (it is in S-units).
correct_smeter = 15.5
 # This is the fraction of spectrum to display from zero to one. It is needed if
 # the passband edges are not valid. Use 0.85 for the SDR-IQ.
display_fraction = 1.00
 # Quisk has record and playback buttons to save radio sound. If there is no more room for
 # sound, the old sound is discarded and the most recent sound is retained. This controls
 # the maximum length of sound storage in minutes. If you want to transmit recorded sound,
 # then mic_sample_rate must equal playback_rate and

both must be 48000.
max_record_minutes = 0.25
 # Thanks to Steve Murphy, KB8RWQ for the patch adding additional color control.
 # Define colors used by all widgets in wxPython colour format:
#color_bg = 'light steel blue' # Lower screen background
#color_graph = 'lemonchiffon1' # Graph background
#color_config2 = 'lemonchiffon3' # color in tab row of config screen
#color_gl = 'grey' # Lines on the graph
#color_graphticks = 'black' # Graph ticks
#color_graphline = '#005500' # graph data line color
#color_graphlabels = '#555555' # graph label color
#color_btn = 'steelblue2' # button color
#color_check_btn = 'yellow2' # color of a check button when it is checked
#color_cycle_btn = 'goldenrod3' # color of a cycle button when it is checked
#color_adjust_btn = 'orange3' # color of an adjustable button when it is checked
#color_test = 'hot pink' # color of a button used for test (turn off for tx)
#color_freq = 'lightcyan1' # background color of frequency and s-meter
#color_entry = color_freq # frequency entry box
#color_enable = 'black' # text color for an enabled

button
#color_disable = ‘white’ # text color for a disabled button
#color_bandwidth = ‘lemonchiffon2′ # color for bandwidth display; thanks to WB4JFI
#color_bandwidth = ‘lemonchiffon3′
#color_txline = ‘red’ # vertical line color for tx in graph
#color_rxline = ‘green’ # vertical line color for rx in graph
 # This is a dark color scheme designed by Steve Murphy, KB8RWQ.
#color_bg = ‘#111111′
#color_graph = ‘#111111′
#color_config2 = color_bg
#color_gl = ‘#555555′
#color_graphticks = ‘#DDDDDD’
#color_graphline = ‘#00AA00′
#color_graphlabels = ‘#FFFFFF’
#color_btn = ‘#666666′
#color_check_btn = ‘#996699′
#color_cycle_btn = ‘#666699′
#color_adjust_btn = ‘#669999′
#color_test = ‘hot pink’
#color_freq = ‘#333333′
#color_entry = color_freq
#color_enable = ‘white’
#color_disable = ‘black’
#color_bandwidth = ‘#333333′

#color_txline = ‘red’
#color_rxline = ‘green’
 # This is a dark colour scheme modified by VK5FSCK based on designed by Steve Murphy, KB8RWQ.
color_bg = ‘#111111′
color_graph = ‘#111111′
color_config2 = color_bg
color_gl = ‘#555555′
color_graphticks = ‘#DDDDDD’
color_graphline = ‘#00AA00′
color_graphlabels = ‘#FFFFFF’
color_btn = ‘#4D4D4D’
color_check_btn = ‘#0000FF’
color_cycle_btn = ‘#666699′
color_adjust_btn = ‘#669999′
color_test = ‘red’
color_freq = ‘#333333′
color_entry = color_freq
color_enable = ‘white’
color_disable = ‘black’
color_bandwidth = ‘#333333′
color_txline = ‘red’
color_rxline = ‘green’
filter_display = 1 # Display the filter bandwidth on the graph screen; 0 or 1; thanks to WB4JFI
 # These are the palettes for the waterfall. The one used is named waterfallPallette,
 # so to use a different one, overwrite this name in

your .quisk_conf.py.
waterfallPalette = (
(0, 0, 0, 0),
(36, 85, 0, 255),
(73, 153, 0, 255),
(109, 255, 0, 128),
(146, 255, 119, 0),
(182, 85, 255, 100),
(219, 255, 255, 0),
(255, 255, 255, 255)
)
digipanWaterfallPalette = (
(0, 0, 0, 0),
(32, 0, 0, 62),
(64, 0, 0, 126),
(96, 145, 142, 96),
(128, 181, 184, 48),
(160, 223, 226, 105),
(192, 254, 254, 4),
(255, 255, 58, 0)
)
 # Quisk can access your sound card through PortAudio or through ALSA drivers.
 # In PortAudio, soundcards have an index number 0, 1, 2, … and a name.
 # The name can be something like “HDA NVidia: AD198x Analog (hw:0,0)” or
 # “surround41″. In Quisk, all PortAudio device names

start with “portaudio”.
 # A device name like “portaudio#6″ directly specifies the index. A name like
 # “portaudio:text” means to search for “text” in all available devices. And
 # there is a default device “portaudiodefault”. So these portaudio names are useful:
#name_of_sound_capt = “portaudio:(hw:0,0)” # First sound card
#name_of_sound_capt = “portaudio:(hw:1,0)” # Second sound card, etc.
#name_of_sound_capt = “portaudio#1″ # Directly specified index
#name_of_sound_capt = “portaudiodefault” # May give poor performance on capture
 # In ALSA, soundcards have these names. The “hw” devices are the raw
 # hardware devices, and should be used for soundcard capture.
#name_of_sound_capt = “hw:0″ # First sound card
#name_of_sound_capt = “hw:1″ # Second sound card, etc.
#name_of_sound_capt = “plughw”
#name_of_sound_capt = “plughw:1″
#name_of_sound_capt = “default”
 # Normally you would capture and play on the same soundcard to avoid problems with the
 # two clocks running at slightly different rates. But you

can define name_of_sound_play
 # to play back on a different device. Define this as the empty string “” to turn off
 # play (for a panadapter).
#
 # For the SDR-IQ the soundcard is not used for capture; it only plays back audio.
 # Quisk has a custom decimation scheme for each sample rate. The allowable sample rates
 # are the four SDR-IQ rates plus 24, 48, 96, 192, 240, 384, 480, and 960 ksps. Other rates
 # can be added.
 # Configuration for soundcard capture and playback
use_sdriq = 0 # Get ADC samples from SDR-IQ is not used
use_rx_udp = 0 # Get ADC samples from UDP is not used
sample_rate = 48000 # ADC hardware sample rate in Hertz
if sys.platform == “win32″:
name_of_sound_capt = “Primary”
else:
name_of_sound_capt = “hw:0″ # Name of soundcard capture hardware device.
name_of_sound_play = “hw:0″ # Use the same device for play back
#name_of_sound_play = “” # Panadapter: Do not play
channel_i = 0 # Soundcard index of in-phase channel:

0, 1, 2, …
channel_q = 1 # Soundcard index of quadrature channel: 0, 1, 2, …
 # Thanks to Franco Spinelli for this fix:
 # The H101 hardware using the PCM2904 chip has a one-sample delay between
 # channels, which must be fixed in software. If you have this problem,
 # change channel_delay to either channel_i or channel_q. Use -1 for no delay.
channel_delay = -1
 # This is for mic playback (SoftRock transmit):
tx_channel_delay = -1
 # If you use a soundcard with Ethernet control of the VFO, set these parameters:
rx_ip = “” # Receiver IP address for VFO control
 # If you use an SDR-IQ for capture, see the sample config file quisk_conf_sdriq.py.
 # For the N2ADR 2010 transceiver described in QEX, and for the improved version HiQSDR,
 # see the sample config file in the hiqsdr package directory, and set these:
 # tx_level sets the transmit level 0 to 255 for each band. The None band is the default.
tx_level = {None:120, ’60′:110}
 # Mode DGTL uses this instead of tx_level, so you can reduce the power.
digital_tx_level = 20

 # If you use the HiQSDR hardware, set these:
 # The HiQSDR_BandDict sets the preselect (4 bits) on the X1 connector.
HiQSDR_BandDict = {‘160′:1, ’80′:2, ’40′:3, ’30′:4, ’20′:5, ’15′:6, ’17′:7,
’12′:8, ’10′:9, ‘6’:10, ‘500k':11, ‘137k':12 }
 # For the original N2ADR hardware set this:
 # use_rx_udp = 1
 # For the newer HiQSDR hardware set this:
 # use_rx_udp = 2
 # Thanks to Ethan Blanton, KB8OJH, for this patch for the Si570 (many SoftRock’s):
 # If you are using a DG8SAQ interface to set a Si570 clock directly, set
 # this to True. Complex controllers which have their own internal
 # crystal calibration do not require this.
si570_direct_control = True
 # This is the Si570 startup frequency in Hz. 114.285MHz is the typical
 # value from the data sheet; you can use ‘usbsoftrock calibrate’ to find
 # the value for your device.
si570_xtal_freq = 114213790
 # This is the received radio sound playback rate. The default will
 # be 48 kHz for the SDR-IQ and UDP port samples, and sample_rate for sound

 # card capture. Set it yourself for other rates or hardware.
 # The playback_rate must be 24000, 48000, 96000 or 192000.
 # The preferred rate is 48000 for use with DGTL and transmit of recorded audio.
 # playback_rate = 48000
 # If you use quisk_hardware_fixed.py, this is the fixed VFO frequency in Hertz
fixed_vfo_freq = 7056000
 # Some hardware must be polled to get the key up/down state. This is the time
 # between polls in milliseconds. Use zero to turn off the poll.
key_poll_msec = 0
 # This determines what happens when you tune by dragging the mouse. The correct
 # choice depends on how your hardware performs tuning. You may want to use a
 # custom hardware file with a custom ChangeFrequency() method too.
mouse_tune_method = 0 # The Quisk tune frequency changes and the VFO frequency is unchanged.
#mouse_tune_method = 1 # The Quisk tune frequency is unchanged and the VFO changes.
 # If freq_spacing is not zero, frequencies are rounded to the freq_base plus the
*# freq_spacing; frequency = freq_base + N **

freq_spacing. This is useful at
 # VHF and higher when Quisk is used with a transverter.
freq_spacing = 0
freq_base = 0
 # This is the CW tone frequency in Hertz
cwTone = 600
 # The “DGTL” mode mostly acts like USB, but it sends received audio to an external
 # program that can decode digital modes, and receives audio to transmit.
 # The only program currently supported is Fldigi. Set Fldigi to USB, XML-RPC control.
digital_xmlrpc_url = “http://localhost:7362″ # URL for control by XML-RPC
 # Input audio from an external program for use with mode DGTL. The input must be
 # stereo at 48000 sps, and you must set mic_sample_rate to 48000 also.
digital_input_name = “” # device name for transmit audio
 # digital_input_name = ‘hw:Loopback,0′
 # Output audio to an external program for use with mode DGTL. The output is
 # stereo at the same sample rate as the radio sound playback.
digital_output_name = “” # device name for received audio

 # digital_output_name = digital_input_name
 # You can control Quisk from Hamlib. Set the Hamlib rig to 2 and the device for rig 2 to
 # localhost:4575, or other hamlib_port as used by Quisk.
hamlib_port = 4575 # Standard port for Quisk control. Set the port in Hamlib to 4575 too.
#hamlib_port = 4532 # Default port for rig 2. Use this if you can not set the Hamlib port.
#hamlib_port = 0 # Turn off Hamlib control.
 # If you use the microphone feature, the mic_channel_I and Q are the two capture
 # microphone channels. Quisk uses a monophonic mic, so audio is taken from the I
 # channel, and the Q channel is (currently) ignored. It is OK to set the same
 # channel number for both, and this is necessary for a USB mono mic. The mic sample rate
 # should be 48000 to enable DGTL and the sound recorder to work, but 8000 can be used.
 # Mic samples can be sent to an Ethernet device (use tx_ip and name_of_mic_play = "")
 # or to a sound card (use name_of_mic_play="hw:1" or other device).
 # If mic samples are sent to a sound card for Tx, the samples are tuned to the audio
 # transmit frequency, and are set to zero unless the key is down.

 # If there is no mic (microphone_name = “”), it is still possible to transmit CW,
 # and you should set mic_playback_rate to the I/Q receive capture rate.
 # Microphone capture:
microphone_name = “” # Name of microphone capture device (or “hw:1″)
mic_sample_rate = 48000 # Microphone capture sample rate in Hertz, should be 48000, can be 8000
mic_channel_I = 0 # Soundcard index of mic capture audio channel
mic_channel_Q = 0 # Soundcard index of ignored capture channel
 # Microphone samples sent to soundcard:
name_of_mic_play = “” # Name of play device if mic I/Q is sent to a sound card
mic_playback_rate = 48000 # Playback rate must be a multiple 1, 2, … of mic_sample_rate
mic_play_chan_I = 0 # Soundcard index of mic I play channel
mic_play_chan_Q = 1 # Soundcard index of mic Q play channel
mic_out_volume = 0.7 # Microphone output volume (after all processing) as a fraction 0.0 to 0.7
 # Microphone samples sent to UDP:
tx_ip = “” # Transmit IP address for mic sent to UDP (or “192.168.2.195”)
tx_audio_port = 0 # UDP port for mic samples (or

0x553B)
 # Microphone audio processing:
 # The original audio processing used mic_clip = 4.0; mic_preemphasis = -1.0
 # For no mic audio processing, use mic_clip = 1.0; mic_preemphasis = 0.0
mic_clip = 3.0 # Mic amplitude clipping; larger numbers give more clipping
mic_preemphasis = 0.6 # Mic pre-emphasis 0.0 (none) to 1.0; or -1.0 for a Butterworth filter
 # If your mixing scheme inverts the RF spectrum, set this option to un-invert it
invertSpectrum = 0
 # Use “amixer -c 1 contents” to get a list of mixer controls and their numid’s for
 # card 1 (or “-c 0″ for card 0). Then make a list of (device_name, numid, value)
 # for each control you need to set. The sample settings are for my USB microphone.
#mixer_settings = [
 # ("hw:1", 2, 0.80), # numid of microphone volume control, volume 0.0 to 1.0;
 # ("hw:1", 1, 1.0) # numid of capture on/off control, turn on with 1.0;
 #]
 # If you want Quisk to add a button to generate a 2-tone IMD test signal,
 # set this to 1. This feature requires the microphone to

work.
add_imd_button = 0
 # If you want Quisk to add a full duplex button (transmit and receive at the
 # same time), set this to 1.
add_fdx_button = 0
 # If you want to write your own I/Q filter and demodulation module, set
 # this to the name of the button to add, and change extdemod.c.
 # add_extern_demod = “WFM”
add_extern_demod = “”
 # These are the suppressed carrier frequencies for 60 meters
freq60 = (5330500, 5346500, 5357000, 5371500, 5403500)
 # These are the filter bandwidths for each mode. Quisk has built-in optimized filters
 # for these values, but you can change them if you want.
FilterBwCW = (200, 400, 600, 1000, 1500, 3000)
FilterBwSSB = (2000, 2200, 2500, 2800, 3000, 3300)
FilterBwAM = (4000, 5000, 6000, 8000, 10000, 9000)
FilterBwFM = (8000, 10000, 12000, 15000, 17000, 20000)
FilterBwIMD = FilterBwSSB
FilterBwEXT = (8000, 10000, 12000, 15000, 17000, 20000)

 # This is the data used to draw colored lines on the frequency X axis to
 # indicate CW and Phone sub-bands. You can make it anything you want.
 # These are the colors used for sub-bands:
CW = '#FF4444' # General class CW
eCW = '#FF8888' # Extra class CW
Phone = '#4444FF' # General class phone
ePhone = '#8888FF' # Extra class phone
 # ARRL band plan special frequencies
Data = '#FF9900'
DxData = '#CC6600'
RTTY = '#FF9900'
SSTV = '#FFFF00'
AM = '#00FF00'
Packet = '#00FFFF'
Beacons = '#66FF66'
Satellite = '#22AA88'
Repeater = '#AA00FF' # Repeater outputs
RepInput = '#AA88FF' # Repeater inputs
Simplex = '#00FF44'
Special = 'hot pink'
Other = '#888888'
 # Colors start at the indicated frequency and continue until the
 # next frequency. The special color "None" turns off color.
#

```
# To change BandPlan in your config file, first remove
any frequencies in the range
# you want to change; then add your frequencies; and
then sort the list. Or you could just
# replace the whole list.
BandPlan = [
# Test display of colors
#[ 0, CW], [ 50000, eCW], [ 100000, Phone], [ 150000,
ePhone], [ 200000, Data], [ 250000, DxData], [ 300000,
RTTY], [ 350000, SSTV],
#[ 400000, AM], [ 450000, Packet], [ 500000, Beacons],
[ 550000, Satellite], [ 600000, Repeater], [ 650000,
RepInput], [ 700000, Simplex],
#[ 750000, Other], [ 800000, Special], [ 850000, None],
# 160 meters
[ 1800000, Data],
[ 1809000, Other],
[ 1811000, CW],
[ 1843000, Phone],
[ 1908000, Other],
[ 1912000, Phone],
[ 1995000, Other],
[ 2000000, None],
# 80 meters
[ 3500000, eCW],
[ 3525000, CW],
[ 3570000, Data],
[ 3589000, DxData],
```

```
[ 3591000, Data],
[ 3600000, ePhone],
[ 3790000, Other],
[ 3800000, Phone],
[ 3844000, SSTV],
[ 3846000, Phone],
[ 3880000, AM],
[ 3890000, Phone],
[ 4000000, None],
# 60 meters
[ freq60[0], Phone],
[ freq60[0] + 2800, None],
[ freq60[1], Phone],
[ freq60[1] + 2800, None],
[ freq60[2], Phone],
[ freq60[2] + 2800, None],
[ freq60[3], Phone],
[ freq60[3] + 2800, None],
[ freq60[4], Phone],
[ freq60[4] + 2800, None],
# 40 meters
[ 7000000, eCW],
[ 7025000, CW],
[ 7039000, DxData],
[ 7041000, CW],
[ 7080000, Data],
[ 7125000, ePhone],
[ 7170000, SSTV],
```

[7172000, ePhone],
[7175000, Phone],
[7285000, AM],
[7295000, Phone],
[7300000, None],
 # 30 meters
[10100000, CW],
[10130000, RTTY],
[10140000, Packet],
[10150000, None],
 # 20 meters
[14000000, eCW],
[14025000, CW],
[14070000, RTTY],
[14095000, Packet],
[14099500, Other],
[14100500, Packet],
[14112000, CW],
[14150000, ePhone],
[14225000, Phone],
[14229000, SSTV],
[14231000, Phone],
[14281000, AM],
[14291000, Phone],
[14350000, None],
 # 17 meters
[18068000, CW],
[18100000, RTTY],

[18105000, Packet],
[18110000, Phone],
[18168000, None],
 # 15 meters
[21000000, eCW],
[21025000, CW],
[21070000, RTTY],
[21110000, CW],
[21200000, ePhone],
[21275000, Phone],
[21339000, SSTV],
[21341000, Phone],
[21450000, None],
 # 12 meters
[24890000, CW],
[24920000, RTTY],
[24925000, Packet],
[24930000, Phone],
[24990000, None],
 # 10 meters
[28000000, CW],
[28070000, RTTY],
[28150000, CW],
[28200000, Beacons],
[28300000, Phone],
[28679000, SSTV],
[28681000, Phone],
[29000000, AM],

```
[29200000, Phone],
[29300000, Satellite],
[29520000, Repeater],
[29590000, Simplex],
[29610000, Repeater],
[29700000, None],
# 6 meters
[50000000, Beacons],
[50100000, Phone],
[54000000, None],
# 2 meters
[144000000, CW],
[144200000, Phone],
[144275000, Beacons],
[144300000, Satellite],
[144380000, Special],
[144400000, Satellite],
[144500000, RepInput],
[144900000, Other],
[145100000, Repeater],
[145500000, Other],
[145800000, Satellite],
[146010000, RepInput],
[146400000, Simplex],
[146510000, Special], # Simplex calling frequency
[146530000, Simplex],
[146610000, Repeater],
[147420000, Simplex],
```

[147600000, RepInput],
[148000000, None],
 # 1.25 meters
[222000000, Phone],
[222250000, RepInput],
[223400000, Simplex],
[223520000, Data],
[223640000, Repeater],
[225000000, None],
#70 centimeters
[420000000, SSTV],
[432000000, Satellite],
[432070000, Phone],
[432300000, Beacons],
[432400000, Phone],
[433000000, Repeater],
[435000000, Satellite],
[438000000, Repeater],
[445900000, Simplex],
[445990000, Special], # Simplex calling frequency
[446010000, Simplex],
[446100000, Repeater],
[450000000, None],
 # 33 centimeters
[902000000, Other],
[928000000, None],
 # 23 centimeters
[1240000000, Other],

[1300000000, None],
]
 # For each band, this dictionary gives the lower and upper band edges. Frequencies
 # outside these limits will not be remembered as the last frequency in the band.
BandEdge = {
‘160’:(1800000, 2000000), ’80’ 3500000, 4000000), ’60’ 5300000, 5430000), ’40’ 7000000, 7300000), ’30’ :(10100000, 10150000), ’20’ :(14000000, 14350000),
’17’ :(18068000, 18168000), ’15’ :(21000000, 21450000),
’12’ :(24890000, 24990000), ’10’ :(28000000, 29700000),
‘6’ :(50000000, 54000000),
‘2’ :(144000000, 148000000),
}
 # For each band, this dictionary gives the initial center frequency, tuning
 # frequency as an offset from the center frequency, and the mode. This is
 # no longer too useful because the persistent_state feature saves and then
 # overwrites these values anyway.
bandState = {‘Audio':(0, 0, ‘LSB’),
‘160’:(1890000, -10000, ‘LSB’), ’80’ 3660000, -10000, ‘LSB’),

’60’ 5370000, 1500, ‘USB’), ’40’ 7180000, -5000, ‘LSB’), ’30′:(10120000, -10000, ‘CWL’),
’20’ :(14200000, -10000, ‘USB’), ’17’ :(18120000, 10000, ‘USB’), ’15′:(21250000, -10000, ‘USB’),
’12’ :(24940000, 10000, ‘USB’), ’10’ :(28400000, -10000, ‘USB’),
‘Time':(5000000, 0, ‘AM’), ‘6’ :(50040000, 10000, ‘USB’),
}
 # For the Time band, this is the center frequency, tuning frequency and mode:
bandTime = [
(2500000-10000, 10000, 'AM'),
(3330000-10000, 10000, 'AM'),
(5000000-10000, 10000, 'AM'),
(7335000-10000, 10000, 'AM'),
(10000000-10000, 10000, 'AM'),
(14670000-10000, 10000, 'AM'),
(15000000-10000, 10000, 'AM'),
(20000000-10000, 10000, 'AM'),
]
 # This is the list of band buttons that Quisk displays, and it should have
 # a length of 12. Empty buttons can have a null string “” label.
 # Note that the 60 meter band and the Time band have buttons that support
 # multiple presses.

```
bandLabels = ['Audio', '160', '80', ('60',) * 5, '40', '30',
'20', '17',
'15', '12', '10', ('Time',) * len(bandTime)]
# If you get your I/Q samples from a sound card, you
will need to correct the
# amplitude and phase for inaccuracies in the analog
hardware. The data is
# entered using the controls from the “Rx Phase” button
on the config screen.
# The corrections are saved by the persistent_state
feature.
#
# The available range of the amplitude and phase
controls for receive:
rx_max_amplitude_correct = 0.2 # Correction relative
to 1.000000 (ideally 0.0000)
rx_max_phase_correct = 10.0 # Correction in degrees
of phase (ideally 0.0000)
#
# The bandAmplPhase dictionary gives the amplitude
and phase corrections for
# sound card data. The format is a dictionary with key
“band”, giving a dictionary
# with key “rx” or “tx”, giving a list of tuples (VFO, tune,
amplitude, phase).
#
# If you use Quisk as a panadapter, the corrections will
not depend on the band.
```

```
# In that case create a band “panadapter” in your config file, and all corrections
# will be read/written to that band.
bandAmplPhase = {} # Empty dictionary to start
#bandAmplPhase = {‘panadapter':{}} # Create “panadapter” band for all corrections
# The program polls the soundcard or SDR-IQ for data every data_poll_usec microseconds.
# A lower time reduces latency; a higher time is less taxing on the hardware.
if sys.platform == “win32″:
data_poll_usec = 20000 # poll time in microseconds
else:
data_poll_usec = 5000 # poll time in microseconds
# The fft_size is the width of the data on the screen (about 800 to
# 1200 pixels) times the fft_size_multiplier. Multiple FFTs are averaged
# together to achieve your graph refresh rate. If fft_size_multiplier is
# too small you will get many fft errors. You can specify fft_size_multiplier,
# or enter a large number (use 9999) to maximize it, or enter zero to let
# quisk calculate it for you. Look for fft_size_multiplier in quisk.py.
# If your hardware can change the decimation, there are further compilcations.
```

 # The FFT size is fixed, and only the average count can change to adjust the
 # refresh rate.
fft_size_multiplier = 0
 # The graph_refresh is the frequency at which the graph is updated,
 # and should be about 5 to 10 Hertz. Higher rates require more processor power.
graph_refresh = 7 # update the graph at this rate in Hertz
 # latency_millisecs determines how many samples are in the soundcard play buffer.
 # A larger number makes it less likely that you will run out of samples to play,
 # but increases latency. It is OK to suffer a certain number of play buffer
 # underruns in order to get lower latency.
latency_millisecs = 150 # latency time in milliseconds
 # Select the method to test the state of the key; see is_key_down.c
key_method = “” # No keying, or internal method
 # key_method = “/dev/parport0″ # Use the named parallel port
 # key_method = “/dev/ttyS0″ # Use the named serial port
 # key_method = “192.168.1.44” # Use UDP from this address
 # If you are using keying, key-down throws away the

current capture buffer
 # and starts a sidetone with a rise time of 5 milliseconds. For
 # key-up, the sidetone is ended with a fall time of 5 milliseconds, then
 # a silent period starts, then normal audio starts with a rise time of
 # 5 milliseconds. The length of the silent period is given by keyupDelay,
 # but will be at least the time necessary to collect enough samples to
 # refill the filters. A larger keyupDelay may be needed to accomodate
 # antenna switching or other requirement of your hardware.
keyupDelay = 23 # extra milliseconds silence on key up
 # This is a tuning parameter for the AGC. It controls the maximum AGC gain and thus
 # the scale of the AGC slider control. It depends on your sample magnitudes.
agc_max_gain = 15000.0
 # This controls the audio gain when AGC is off. It depends on your sample magnitudes.
 # Reduce it if turning off the AGC makes the sound too loud.
agc_off_gain = 1400.0
 # For FM transmit, this is the modulation index.
modulation_index = 1.67

Download quisk_conf.py
The only real thing you'll have to change in the config file is:
si570_xtal_freq = 114213790
This needs to be set to the frequency you have calibrated your receiver for.
Only other things I have modified in the config file is layout and colour scheme and to start in WFall, you can see that from the screen shot above.