
API for RF receivers including ThinkRF WSA4000

Release 0.3.0-dev

ThinkRF Corporation

February 15, 2016

1	Reference	3
1.1	pyrf.devices.thinkrf	3
1.2	pyrf.config	3
1.3	pyrf.vrt	4
1.4	pyrf.util	5
1.5	pyrf.numpy_util	5
2	Basic Examples	7
2.1	show_i_q.py	7
2.2	plot_fft.py	8
3	GUI Example: wsa4000demo	11
3.1	wsa4000demo.py	11
3.2	gui.py	11
3.3	spectrum.py	16
3.4	util.py	20
4	Indices and tables	21
	Python Module Index	23

Contents:

1.1 pyrf.devices.thinkrf

1.2 pyrf.config

```
class pyrf.config.SweepEntry (fstart=2400000000, fstop=2400000000, fstep=100000000, fshift=0,
    decimation=0, antenna=1, gain='vlow', ifgain=0, spp=1024, ppb=1,
    trigtype='none', level_fstart=50000000, level_fstop=1000000000,
    level_amplitude=-100)
```

Sweep entry for `pyrf.devices.thinkrf.WSA4000.sweep_add()`

Parameters

- **fstart** – starting frequency in Hz
- **fstop** – ending frequency in Hz
- **shift** – the frequency shift in Hz
- **decimation** – the decimation value (0 or 4 - 1023)
- **antenna** – the antenna (1 or 2)
- **gain** – the RF gain value ('high', 'medium', 'low' or 'vlow')
- **ifgain** – the IF gain in dB (-10 - 34)
- **spp** – samples per packet
- **ppb** – packets per block
- **trigtype** – trigger type ('none' or 'level')
- **level_fstart** – level trigger starting frequency in Hz
- **level_fstop** – level trigger ending frequency in Hz
- **level_amplitude** – level trigger minimum in dBm

```
class pyrf.config.TriggerSettings (trigtype='NONE', fstart=None, fstop=None, ampli-  
tude=None)
```

Trigger settings for `pyrf.devices.thinkrf.WSA4000.trigger()`.

Parameters

- **trigtype** – “LEVEL” or “NONE” to disable
- **fstart** – starting frequency in Hz

- **fstop** – ending frequency in Hz
- **amplitude** – minimum level for trigger in dBm

exception `pyrf.config.TriggerSettingsError`

1.3 `pyrf.vrt`

class `pyrf.vrt.ContextPacket` (*pkt_type, word, socket*)

A Context Packet received from `pyrf.vrt.Stream.read_packet()`

fields

a dict containing field names and values from the packet

is_context_packet (*p_type=None*)

Parameters *p_type* – “Receiver”, “Digitizer” or None for any packet type

Returns True if this packet matches the type passed

is_data_packet ()

Returns False

class `pyrf.vrt.DataPacket` (*word, socket*)

A Data Packet received from `pyrf.vrt.Stream.read_packet()`

data

a `pyrf.vrt.IQData` object containing the packet data

is_context_packet (*p_type=None*)

Returns False

is_data_packet ()

Returns True

class `pyrf.vrt.IQData` (*binary_data*)

Data Packet values as a lazy collection of (I, Q) tuples read from *binary_data*.

This object behaves as an immutable python sequence, e.g. you may do any of the following:

```
points = len(iq_data)

i_and_q = iq_data[5]

for i, q in iq_data:
    print i, q
```

numpy_array ()

Return a numpy array of I, Q values for this data similar to:

```
array([[ -44,    8],
       [ -40,   60],
       [ -12,   92],
       ...,
       [-132,   -8],
       [-124,   56],
       [ -44,   80]], dtype=int16)
```

exception `pyrf.vrt.InvalidDataReceived`

class `pyrf.vrt.Stream`(*socket*)

A VRT Packet Stream interface wrapping *socket*.

has_data()

Returns True if there is data waiting on *socket*.

read_packet()

Read a complete packet from *socket* and return either a `pyrf.vrt.ContextPacket` or a `pyrf.vrt.DataPacket`.

1.4 pyrf.util

`pyrf.util.read_data_and_reflevel`(*dut, points=1024*)

Wait for and capture a data packet and a reference level packet.

Returns (*data_pkt, reflevel_pkt*)

`pyrf.util.socketread`(*socket, count, flags=None*)

Retry socket read until count data received, like reading from a file.

1.5 pyrf.numpy_util

`pyrf.numpy_util.compute_fft`(*dut, data_pkt, reflevel_pkt*)

Return an array of dBm values by computing the FFT of the passed data and reference level.

Parameters

- **dut** (`pyrf.devices.thinkrf.WSA4000`) – WSA device
- **data_pkt** (`pyrf.vrt.DataPacket`) – packet containing samples
- **reflevel_pkt** (`pyrf.vrt.ContextPacket`) – packet containing ‘reflevel’ value

This function uses only *dut.ADC_DYNAMIC_RANGE*, *data_pkt.data* and *reflevel_pkt[‘reflevel’]*.

Returns numpy array of dBm values as floats

Basic Examples

2.1 show_i_q.py

This example connects to a device specified on the command line, tunes it to a center frequency of 2.450 MHz then reads and displays one capture of 1024 i, q values.

```
#!/usr/bin/env python

import sys
from pyrf.devices.thinkrf import WSA4000

# connect to wsa
dut = WSA4000()
dut.connect(sys.argv[1])

# setup test conditions
dut.reset()
dut.request_read_perm()
dut.ifgain(0)
dut.freq(2450e6)
dut.gain('low')
dut.fshift(0)
dut.decimation(0)

# capture 1 packet
dut.capture(1024, 1)

# read until I get 1 data packet
while not dut.eof():
    pkt = dut.read()

    if pkt.is_data_packet():
        break

# print I/Q data into i and q
for i, q in pkt.data:
    print "%d,%d" % (i, q)
```

Example output (truncated):

```
0,-20
-8,-16
0,-24
```

```
-8,-12
0,-32
24,-24
32,-16
-12,-24
-20,0
12,-32
32,-4
0,12
-20,-16
-48,16
-12,12
0,-36
4,-12
```

2.2 plot_fft.py

This example connects to a device specified on the command line, tunes it to a center frequency of 2.450 MHz and sets a trigger for a signal with an amplitude of -70 dBm or greater between 2.400 MHz and 2.480 MHz.

When the trigger is satisfied the data is captured and rendered as a spectrum display using NumPy and matplotlib.

```
#!/usr/bin/env python

from pyrf.devices.thinkrf import WSA4000
from pyrf.config import TriggerSettings
from pyrf.util import read_data_and_reflevel
from pyrf.numpy_util import compute_fft

import sys
import time
import math

from matplotlib.pyplot import plot, figure, axis, xlabel, ylabel, show

# connect to wsa
dut = WSA4000()
dut.connect(sys.argv[1])

# setup test conditions
dut.reset()
dut.request_read_perm()
dut.ifgain(0)
dut.freq(2450e6)
dut.gain('high')
dut.fshift(0)
dut.decimation(0)
trigger = TriggerSettings(
    trigtype="LEVEL",
    fstart=2400e6,
    fstop=2480e6,
    amplitude=-70)
dut.trigger(trigger)

# capture 1 packet
data, reflevel = read_data_and_reflevel(dut, 1024)
```

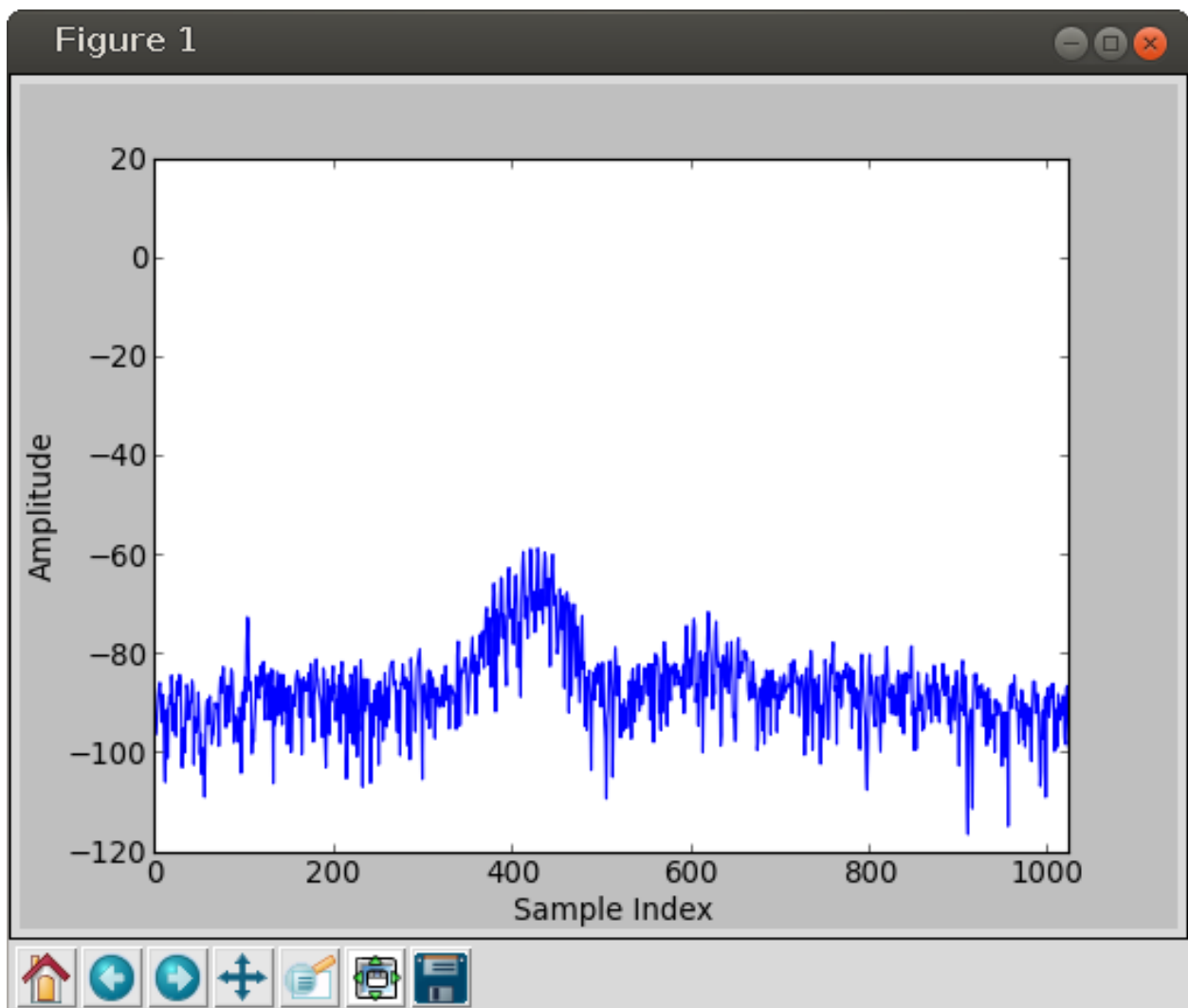
```
# compute the fft of the complex data
powdata = compute_fft(dut, data, refllevel)

# setup my graph
fig = figure(1)
axis([0, 1024, -120, 20])

xlabel("Sample Index")
ylabel("Amplitude")

# plot something
plot(powdata, color='blue')

# show graph
show()
```

Fig. 2.1: Example output of `plot_fft.py`

GUI Example: wsa4000demo

wsa4000demo is a cross-platform GUI application built with the `Qt` toolkit and `PySide` bindings for Python.

You may run application by launching the `wsa4000demo.py` script in the `examples/gui` directory.

You may specify a device on the command line or open a device after the GUI has launched. Adding `--reset` to the command line parameters will cause the device to be reset to defaults after connecting.

3.1 wsa4000demo.py

This is the script that launches the application.

```
#!/usr/bin/env python

import sys
from PySide import QtGui
from gui import MainWindow

app = QtGui.QApplication(sys.argv)
ex = MainWindow()
sys.exit(app.exec_())
```

3.2 gui.py

The main application window and GUI controls are created here.

`MainWindow` creates and handles the `File | Open Device` menu and wraps the `MainPanel` widget responsible for most of the interface.

All the buttons and controls and their callback functions are built in `MainPanel` and arranged on a grid. A `SpectrumView` is created and placed to left of the controls.

Note: This version calls `MainPanel.update_screen()` in a timer loop 20 times a second. This method makes a blocking call to capture the next packet and render it all in the same thread as the application. This can make the interface slow or completely unresponsive if you lose connection to the device.

The next release will move the blocking call and data processing into a separate process.

```
import sys
import socket

from PySide import QtGui, QtCore
from spectrum import SpectrumView
from util import frequency_text

from pyrf.devices.thinkrf import WSA4000
from pyrf.util import read_data_and_reflevel
from pyrf.numpy_util import compute_fft

DEVICE_FULL_SPAN = 125e6
REFRESH_CHARTS = 0.05

class MainWindow(QtGui.QMainWindow):
    def __init__(self, name=None):
        super(MainWindow, self).__init__()
        self.initUI()

        self.dut = None
        if len(sys.argv) > 1:
            self.open_device(sys.argv[1])
        else:
            self.open_device_dialog()
        self.show()

        timer = QtCore.QTimer(self)
        timer.timeout.connect(self.update_charts)
        timer.start(REFRESH_CHARTS)

    def initUI(self):
        openAction = QtGui.QAction('&Open Device', self)
        openAction.triggered.connect(self.open_device_dialog)
        exitAction = QtGui.QAction('&Exit', self)
        exitAction.setShortcut('Ctrl+Q')
        exitAction.triggered.connect(self.close)
        menubar = self.menuBar()
        fileMenu = menubar.addMenu('&File')
        fileMenu.addAction(openAction)
        fileMenu.addAction(exitAction)

        self.setWindowTitle('ThinkRF WSA4000')

    def open_device_dialog(self):
        name, ok = QtGui.QInputDialog.getText(self, 'Open Device',
            'Enter a hostname or IP address:')
        while True:
            if not ok:
                return

            try:
                self.open_device(name)
                break
            except socket.error:
                name, ok = QtGui.QInputDialog.getText(self, 'Open Device',
```



```

        'Connection Failed, please try again\n\n'
        'Enter a hostname or IP address:')

def open_device(self, name):
    dut = WSA4000()
    dut.connect(name)
    dut.request_read_perm()
    if '--reset' in sys.argv:
        dut.reset()

    self.dut = dut
    self.setCentralWidget(MainPanel(dut))
    self.setWindowTitle('ThinkRF WSA4000: %s' % name)

def update_charts(self):
    if self.dut is None:
        return
    self.centralWidget().update_screen()

class MainPanel(QtGui.QWidget):

    def __init__(self, dut):
        super(MainPanel, self).__init__()
        self.dut = dut
        self.get_freq_mhz()
        self.get_decimation()
        self.decimation_points = None
        data, refllevel = read_data_and_refllevel(dut)
        self.screen = SpectrumView(
            compute_fft(dut, data, refllevel),
            self.center_freq,
            self.decimation_factor)
        self.initUI()

    def initUI(self):
        grid = QtGui.QGridLayout()
        grid.setSpacing(10)
        grid.addWidget(self.screen, 0, 0, 8, 1)
        grid.setColumnMinimumWidth(0, 400)

        y = 0
        grid.addWidget(self._antenna_control(), y, 1, 1, 2)
        grid.addWidget(self._bpf_control(), y, 3, 1, 2)
        y += 1
        grid.addWidget(self._gain_control(), y, 1, 1, 2)
        grid.addWidget(QtGui.QLabel('IF Gain:'), y, 3, 1, 1)
        grid.addWidget(self._ifgain_control(), y, 4, 1, 1)
        y += 1
        freq, steps, freq_plus, freq_minus = self._freq_controls()
        grid.addWidget(QtGui.QLabel('Center Freq:'), y, 1, 1, 1)
        grid.addWidget(freq, y, 2, 1, 2)
        grid.addWidget(QtGui.QLabel('MHz'), y, 4, 1, 1)
        y += 1
        grid.addWidget(steps, y, 2, 1, 2)
        grid.addWidget(freq_minus, y, 1, 1, 1)
        grid.addWidget(freq_plus, y, 4, 1, 1)
        y += 1

```

```

span, rbw = self._span_rbw_controls()
grid.addWidget(span, y, 1, 1, 2)
grid.addWidget(rbw, y, 3, 1, 2)

self.setLayout(grid)
self.show()

def _antenna_control(self):
    antenna = QtGui.QComboBox(self)
    antenna.addItem("Antenna 1")
    antenna.addItem("Antenna 2")
    antenna.setCurrentIndex(self.dut.antenna() - 1)
    def new_antenna():
        self.dut.antenna(int(antenna.currentText().split()[-1]))
    antenna.currentIndexChanged.connect(new_antenna)
    return antenna

def _bpf_control(self):
    bpf = QtGui.QComboBox(self)
    bpf.addItem("BPF On")
    bpf.addItem("BPF Off")
    bpf.setCurrentIndex(0 if self.dut.preselect_filter() else 1)
    def new_bpf():
        self.dut.preselect_filter("On" in bpf.currentText())
    bpf.currentIndexChanged.connect(new_bpf)
    return bpf

def _gain_control(self):
    gain = QtGui.QComboBox(self)
    gain_values = ['High', 'Med', 'Low', 'VLow']
    for g in gain_values:
        gain.addItem("RF Gain: %s" % g)
    gain_index = [g.lower() for g in gain_values].index(self.dut.gain())
    gain.setCurrentIndex(gain_index)
    def new_gain():
        self.dut.gain(gain.currentText().split()[-1].lower())
    gain.currentIndexChanged.connect(new_gain)
    return gain

def _ifgain_control(self):
    ifgain = QtGui.QSpinBox(self)
    ifgain.setRange(-10, 34)
    ifgain.setSuffix(" dB")
    ifgain.setValue(int(self.dut.ifgain()))
    def new_ifgain():
        self.dut.ifgain(ifgain.value())
    ifgain.valueChanged.connect(new_ifgain)
    return ifgain

def _freq_controls(self):
    freq = QtGui.QLineEdit("")
    def read_freq():
        freq.setText("%0.1f" % self.get_freq_mhz())
    read_freq()
    def write_freq():
        try:
            f = float(freq.text())
        except ValueError:

```

```

        return
        self.set_freq_mhz(f)
freq.editingFinished.connect(write_freq)

steps = QtGui.QComboBox(self)
steps.addItem("Adjust: 1 MHz")
steps.addItem("Adjust: 2.5 MHz")
steps.addItem("Adjust: 10 MHz")
steps.addItem("Adjust: 25 MHz")
steps.addItem("Adjust: 100 MHz")
steps.setCurrentIndex(2)
def freq_step(factor):
    try:
        f = float(freq.text())
    except ValueError:
        return read_freq()
    delta = float(steps.currentText().split()[1]) * factor
    freq.setText("%0.1f" % (f + delta))
    write_freq()
freq_minus = QtGui.QPushButton('-')
freq_minus.clicked.connect(lambda: freq_step(-1))
freq_plus = QtGui.QPushButton('+')
freq_plus.clicked.connect(lambda: freq_step(1))

return freq, steps, freq_plus, freq_minus

def _span_rbw_controls(self):
span = QtGui.QComboBox(self)
decimation_values = [1] + [2 ** x for x in range(2, 10)]
for d in decimation_values:
    span.addItem("Span: %s" % frequency_text(DEVICE_FULL_SPAN / d))
span.setCurrentIndex(decimation_values.index(self.dut.decimation()))
def new_span():
    self.set_decimation(decimation_values[span.currentIndex()])
    build_rbw()
span.currentIndexChanged.connect(new_span)

rbw = QtGui.QComboBox(self)
points_values = [2 ** x for x in range(8, 16)]
rbw.addItems([str(p) for p in points_values])
def build_rbw():
    d = self.decimation_factor
    for i, p in enumerate(points_values):
        r = DEVICE_FULL_SPAN / d / p
        rbw.setItemText(i, "RBW: %s" % frequency_text(r))
        if self.decimation_points and self.decimation_points == d * p:
            rbw.setCurrentIndex(i)
    self.points = points_values[rbw.currentIndex()]
build_rbw()
def new_rbw():
    self.points = points_values[rbw.currentIndex()]
    self.decimation_points = self.decimation_factor * self.points
rbw.setCurrentIndex(points_values.index(1024))
new_rbw()
rbw.currentIndexChanged.connect(new_rbw)

return span, rbw

```

```

def update_screen(self):
    data, refllevel = read_data_and_refllevel(
        self.dut,
        self.points)
    self.screen.update_data(
        compute_fft(self.dut, data, refllevel),
        self.center_freq,
        self.decimation_factor)

def get_freq_mhz(self):
    self.center_freq = self.dut.freq()
    return self.center_freq / 1e6

def set_freq_mhz(self, f):
    self.center_freq = f * 1e6
    self.dut.freq(self.center_freq)

def get_decimation(self):
    d = self.dut.decimation()
    self.decimation_factor = 1 if d == 0 else d

def set_decimation(self, d):
    self.decimation_factor = 1 if d == 0 else d
    self.dut.decimation(d)

```

3.3 spectrum.py

The SpectrumView widget is divided into three parts:

- SpectrumViewPlot contains the plotted spectrum data.
- SpectrumViewLeftAxis displays the dBm axis along the left.
- SpectrumViewBottomAxis displays the MHz axis across the bottom.

The utility functions dBm_labels and MHz_labels compute the positions and labels for each axis.

```

import numpy
import itertools
from PySide import QtGui, QtCore

TOP_MARGIN = 20
RIGHT_MARGIN = 20
LEFT_AXIS_WIDTH = 70
BOTTOM_AXIS_HEIGHT = 40
AXIS_THICKNESS = 1

DBM_TOP = 20
DBM_BOTTOM = -140
DBM_STEPS = 9

class SpectrumView(QtGui.QWidget):
    """
    A complete spectrum view with left/bottom axis and plot
    """

```

```

def __init__(self, powdata, center_freq, decimation_factor):
    super(SpectrumView, self).__init__()
    self.plot = SpectrumViewPlot(powdata, center_freq, decimation_factor)
    self.left = SpectrumViewLeftAxis()
    self.bottom = SpectrumViewBottomAxis()
    self.bottom.update_params(center_freq, decimation_factor)
    self.initUI()

def initUI(self):
    grid = QtGui.QGridLayout()
    grid.setSpacing(0)
    grid.addWidget(self.left, 0, 0, 2, 1)
    grid.addWidget(self.plot, 0, 1, 1, 1)
    grid.addWidget(self.bottom, 1, 1, 1, 1)
    grid.setRowStretch(0, 1)
    grid.setColumnStretch(1, 1)
    grid.setColumnMinimumWidth(0, LEFT_AXIS_WIDTH)
    grid.setRowMinimumHeight(1, BOTTOM_AXIS_HEIGHT)

    grid.setContentsMargins(0, 0, 0, 0)
    self.setLayout(grid)

def update_data(self, powdata, center_freq, decimation_factor):
    if (self.plot.center_freq, self.plot.decimation_factor) != (
        center_freq, decimation_factor):
        self.bottom.update_params(center_freq, decimation_factor)
        self.plot.update_data(powdata, center_freq, decimation_factor)

def dBm_labels(height):
    """
    return a list of (position, label_text) tuples where position
    is a value between 0 (top) and height (bottom).
    """
    # simple, fixed implementation for now
    dBm_labels = [str(d) for d in
        numpy.linspace(DBM_TOP, DBM_BOTTOM, DBM_STEPS)]
    y_values = numpy.linspace(0, height, DBM_STEPS)
    return zip(y_values, dBm_labels)

class SpectrumViewLeftAxis(QtGui.QWidget):
    """
    The left axis of a spectrum view showing dBm range

    This widget includes the space to the left of the bottom axis
    """
    def paintEvent(self, e):
        qp = QtGui.QPainter()
        qp.begin(self)
        size = self.size()
        self.drawAxis(qp, size.width(), size.height())
        qp.end()

    def drawAxis(self, qp, width, height):
        qp.fillRect(0, 0, width, height, QtCore.Qt.black)
        qp.setPen(QtCore.Qt.gray)
        qp.fillRect(
            width - AXIS_THICKNESS,

```

```

        TOP_MARGIN,
        AXIS_THICKNESS,
        height - BOTTOM_AXIS_HEIGHT + AXIS_THICKNESS - TOP_MARGIN,
        QtCore.Qt.gray)

    for y, txt in dBm_labels(height - BOTTOM_AXIS_HEIGHT - TOP_MARGIN):
        qp.drawText(
            0,
            y + TOP_MARGIN - 10,
            LEFT_AXIS_WIDTH - 5,
            20,
            QtCore.Qt.AlignRight | QtCore.Qt.AlignVCenter,
            txt)

def MHz_labels(width, center_freq, decimation_factor):
    """
    return a list of (position, label_text) tuples where position
    is a value between 0 (left) and width (right).
    """
    df = float(decimation_factor)
    # simple, fixed implementation for now
    offsets = (-50, -25, 0, 25, 50)
    freq_labels = [str(center_freq / 1e6 + d/df) for d in offsets]
    x_values = [(d + 62.5) * width / 125 for d in offsets]
    return zip(x_values, freq_labels)

class SpectrumViewBottomAxis(QtGui.QWidget):
    """
    The bottom axis of a spectrum view showing frequencies
    """
    def update_params(self, center_freq, decimation_factor):
        self.center_freq = center_freq
        self.decimation_factor = decimation_factor
        self.update()

    def paintEvent(self, e):
        qp = QtGui.QPainter()
        qp.begin(self)
        size = self.size()
        self.drawAxis(qp, size.width(), size.height())
        qp.end()

    def drawAxis(self, qp, width, height):
        qp.fillRect(0, 0, width, height, QtCore.Qt.black)
        qp.setPen(QtCore.Qt.gray)
        qp.fillRect(
            0,
            0,
            width - RIGHT_MARGIN,
            AXIS_THICKNESS,
            QtCore.Qt.gray)

        for x, txt in MHz_labels(
            width - RIGHT_MARGIN,
            self.center_freq,
            self.decimation_factor):
            qp.drawText(
                x - 40,

```

```

        5,
        80,
        BOTTOM_AXIS_HEIGHT - 10,
        QtCore.Qt.AlignTop | QtCore.Qt.AlignHCenter,
        txt)

class SpectrumViewPlot(QtGui.QWidget):
    """
    The data plot of a spectrum view
    """
    def __init__(self, powdata, center_freq, decimation_factor):
        super(SpectrumViewPlot, self).__init__()
        self.powdata = powdata
        self.center_freq = center_freq
        self.decimation_factor = decimation_factor

    def update_data(self, powdata, center_freq, decimation_factor):
        self.powdata = powdata
        self.center_freq = center_freq
        self.decimation_factor = decimation_factor
        self.update()

    def paintEvent(self, e):
        qp = QtGui.QPainter()
        qp.begin(self)
        self.drawLines(qp)
        qp.end()

    def drawLines(self, qp):
        size = self.size()
        width = size.width()
        height = size.height()
        qp.fillRect(0, 0, width, height, QtCore.Qt.black)

        qp.setPen(QtGui.QPen(QtCore.Qt.gray, 1, QtCore.Qt.DotLine))
        for y, txt in dBm_labels(height - TOP_MARGIN):
            qp.drawLine(
                0,
                y + TOP_MARGIN,
                width - RIGHT_MARGIN - 1,
                y + TOP_MARGIN)
            for x, txt in MHz_labels(
                width - RIGHT_MARGIN,
                self.center_freq,
                self.decimation_factor):
                qp.drawLine(
                    x,
                    TOP_MARGIN,
                    x,
                    height - 1)

        qp.setPen(QtCore.Qt.green)

        y_values = height - 1 - (self.powdata - DBM_BOTTOM) * (
            float(height - TOP_MARGIN) / (DBM_TOP - DBM_BOTTOM))
        x_values = numpy.linspace(0, width - 1 - RIGHT_MARGIN,

```

```

len(self.powdata))

path = QtGui.QPainterPath()
points = itertools.izip(x_values, y_values)
path.moveTo(*next(points))
for x,y in points:
    path.lineTo(x, y)
qp.drawPath(path)

```

3.4 util.py

Pretty-print frequency values

```

def frequency_text (hz) :
    """
    return hz as readable text in Hz, kHz, MHz or GHz
    """
    if hz < 1e3:
        return "%.3f Hz" % hz
    elif hz < 1e6:
        return "%.3f kHz" % (hz / 1e3)
    elif hz < 1e9:
        return "%.3f MHz" % (hz / 1e6)
    return "%.3f GHz" % (hz / 1e9)

```

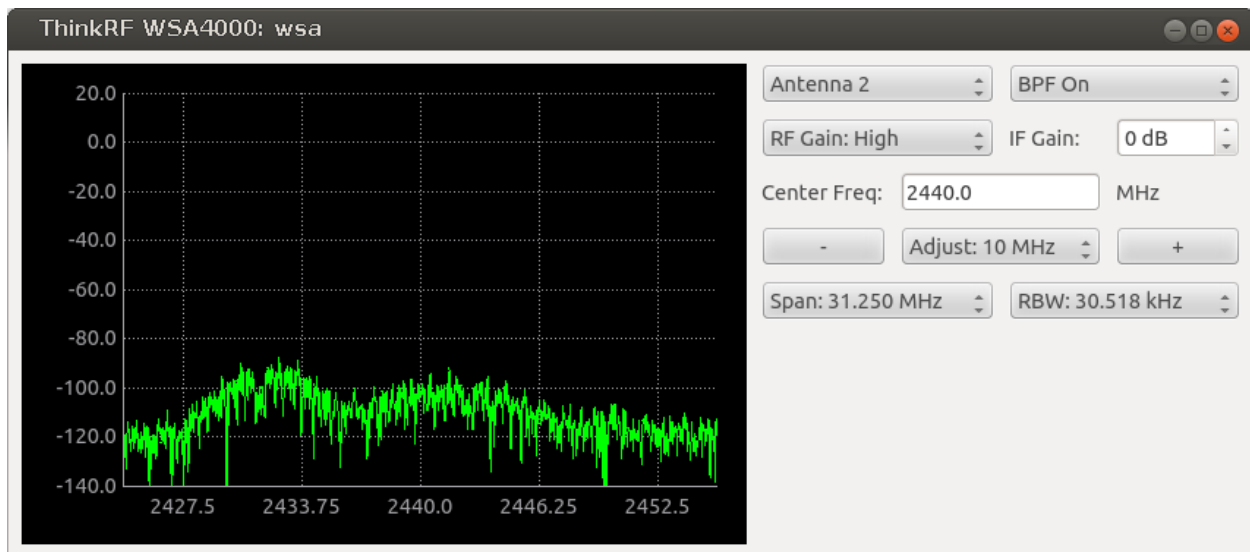


Fig. 3.1: The *wsa4000demo* GUI application

Indices and tables

- `genindex`
- `search`

p

`pyrf.config`, 3
`pyrf.numpy_util`, 5
`pyrf.util`, 5
`pyrf.vrt`, 4

C

compute_fft() (in module pyrf.numpy_util), 5
ContextPacket (class in pyrf.vrt), 4

D

data (pyrf.vrt.DataPacket attribute), 4
DataPacket (class in pyrf.vrt), 4

F

fields (pyrf.vrt.ContextPacket attribute), 4

H

has_data() (pyrf.vrt.Stream method), 5

I

InvalidDataReceived, 4
IQData (class in pyrf.vrt), 4
is_context_packet() (pyrf.vrt.ContextPacket method), 4
is_context_packet() (pyrf.vrt.DataPacket method), 4
is_data_packet() (pyrf.vrt.ContextPacket method), 4
is_data_packet() (pyrf.vrt.DataPacket method), 4

N

numpy_array() (pyrf.vrt.IQData method), 4

P

pyrf.config (module), 3
pyrf.numpy_util (module), 5
pyrf.util (module), 5
pyrf.vrt (module), 4

R

read_data_and_reflevel() (in module pyrf.util), 5
read_packet() (pyrf.vrt.Stream method), 5

S

socketread() (in module pyrf.util), 5
Stream (class in pyrf.vrt), 4
SweepEntry (class in pyrf.config), 3

T

TriggerSettings (class in pyrf.config), 3
TriggerSettingsError, 4