138MHz-4.4GHz USB Simple Spectrum Analyzer - BG7TBL_Reader - SDR

Autor:
Data de publicació: 19-07-2016

Description
100% Brand New And High Quality
Output frequency range 138MHz-4.4GHz,
Stepping: 1kHz
Output power (for reference): 150M-3.2DBM; 250M-3.4DBM, 500M,-3.4DBM; 750M-2.8DBM, 1000M-3.4DBM; 1500M-3.6DBM; 2000M-4.0DBM; 2500M-2.8DBM; 3000M-1.8DBM; 3500M-3.6DBM
Spectrum of the input frequency range of 138M-4.4GHz
Frequency Accuracy (after calibration): +-1k @ 1GHz
Low-noise input signal amplitude:-70DBM @ 0.5G
Power: USB2.0, USB3.0 (+5 V/0.3A, it is best to pick USB3.0)
Interface: USB2.0/ USB3.0, SMA
Recommended tracking source (purchased separately) on our ebay shop.
Package includes:
1 x 138MHz-4.4GHz Analyzer
1 x USB cable
2 x SMA cable

---

http://www.rtk-service.de/shop/index.htm

Firmware für den FA-NWT und HFM9
Beschreibung der Anschaltung des FA-Dämpfungsgliedes (Hochgeladen 23.01.08)

Variante 1
This variant is for the FA-NWT direct DDS-clock supply of 400MHz. In the earlier HW to recognize the built-helix filter. The new FA-NWT are all direct DDS-clock supply of 400MHz and also covers all of the Variante 1. All other NWT as the HFM9 direct DDS-clock supply of 400MHz also fall under this variant.

Diese Variante ist für den FA-NWT mit direkter DDS-Takteinspeisung von 400MHz. Bei der älteren HW zu erkennen am eingebauten Helixfilter. Die neuen FA-NWT sind alle mit direkter DDS-Takteinspeisung von 400MHz und fallen auch alle unter der Variante 1. Alle anderen NWT wie der HFM9 mit direkter DDS-Takteinspeisung von 400MHz fallen auch unter diese Variante.

Version 1.20 Variante 1 als FW-Update (Hochgeladen 24.04.12)
Version 1.19 Variante 1 als FW-Update (Hochgeladen 28.04.09)

Software zum Netzwerktester FA-NWT NWT7 NWT500 HFM9
Dokumentationen zur Software V4.xx
Starthilfe LinNWT/WinNWT V4.xx (PDF) von der Zeitschrift "FUNKAMATEUR". Diese Starthilfe für ist nicht nur für Neueinsteiger. (Hochgeladen 30.04.2009)

LinNWT/WinNWT V4.xx (PDF) (Hochgeladen 30.07.2009; 700 kByte) Gesamtdokumentation in deutsch.

LinNWT/WinNWT_en (PDF) (Hochgeladen 06.08.2014; etwa 700 kByte) Gesamtdokumentation in englisch. Vielen Dank an Paolo Spina aus Italien. Eventuell sind noch Fehler im Dokument. Mein Englisch ist leider nicht ausreichend.

Eine kleine Anleitung zur Installation unter UBUNTU Eventuell auch für andere Distributionen verwendbar. Danke an Christof Bodner. Er hat diese Anleitung geschrieben.

Software History
Versionsverlauf (Hochgeladen 14.02.2013)

Download der PC-Software

Wichtiger Hinweis: Die Software unterliegt der GNU LESSER GENERAL PUBLIC LICENSE. Die Software funktioniert nur einwandfrei im Zusammenspiel mit der von mir entwickelten Firmware in den verschiedenen NWTs. Ich übernehmen keinerlei Haftung über auftretende Mängel oder Schäden!

Die Software wird nicht weiter entwickelt!!!

Erweiterung zum Audio-NF-Wobbeln und Pegelmessungen. Das funktioniert nur mit der neuen FW 1.20. Ein Beitrag dazu erscheint ist in der Zeitschrift Funkamateur. erschienen.

Das Wattmeter funktioniert jetzt mit FW1.19 und FW1.20. Wird die FW 1.20 benutzt muss das Wattmeter neu kalibriert werden, da beim Wattmeter eine Messung aus dem Mittelwert von 32 Einzelmessungen berechnet wird.

LinNWT V4.11.10 Quellen für Linux. Bitte wie gehabt selbst kompilieren. (Hochgeladen 28.03.2016; 562 kByte)

WinNWT V4.11.09 Setup für Windows. (Hochgeladen 30.09.2013; 5,7 MByte)

WinNWT V4.11.09 Setup Windows-hoz magyarul.

NWT V4.11.07 Software für Mac OS X (10.6.8/Snow Leopard) (Hochgeladen 14.02.2013; 26 MByte)Diese Datei wurde mir von Matt Scholz W6ZBA zur Verfügung gestellt.

Fehlerbeseitigung siehe Software History

WinNWT ICON (Hochgeladen 02.06.07; 40 Byte)

Firmware der NWTs
Dieser Bereich wurde in eine neue Seite ausgelagert. Zur Firmware

The BG7TBL USB RF Signal Generator is a PC-based function generator. It has no external controls, requiring a USB connection to a computer.

Software to run this hardware can be found here http://www.dl4jal.eu/.

This device can be bought on ebay for ca $65. Search for "138MHz-4.4GHz". There is a version with the ADF4351 instead that will give you more range (35MHz-4.4GHz).

Contents

1 Hardware
2 Photos
3 Protocol

3.1 Query firmware
3.2 Signal generator

3.2.1 Setting frequency

3.3 Spectrum analyzer

Hardware

Microcontroller: Atmel Atmega 8L
Wideband Synthesizer: ADF4350
USB to Serial: FTDI FT232RL
Logarithmic Amplifier AD8307
Mixer: IAM 81008
LDO: AMS1117

Photos

Protocol
Baud 57600, 8n1

The protocol is binary serial based.

Query firmware

Send 0x8f+"v"

The answer should be in hex:

77

Where the returned byte is the firmware version, 0x77 = 119.

Signal generator
Setting frequency
To set a frequency send:

0x8f

Then f and then the frequency divided by 10 with leading zeroes. For example this is the payload for 400MHz.

400 000 000 / 10
f040000000

Spectrum analyzer
The protocol is briefly described in the source code here:

https://github.com/DoYouKnow/BG7TBL_Reader

BG7TBL_Reader
Read data from a BG7TBL Simple spectrum analyzer device

This program will read data from a BG7TBL and plot its fft and a scrolling spectrogram. In Linux Mint, you can open the fig.jpg file while it is running, so that the environment gets notified of the changes. Other linuxes probably have similar behavior.

The default configuration is to plot the wifi band (2.4 GHz). You can change the start frequency and the other fields in the source file, but keep in mind the length of each field in ascii digits after the comment symbol.

All frequencies are multiplied by 10 behind the device. So, if you want to enter 1 GHz, enter 100,000,000 (without the commas) - which basically means 100 MHz. This is due to the internal frequency scaler.

If you want to enter 200 MHz, enter 020,000,000 (without the commas), or "020000000" in the freq field.

I designed this based on the BG7TBL Simple spectrum analyzer with range 138MHz-4.4 GHz, but devices based on the same software should work.

The step is configurable, and when finding the maximum frequency, the multiplication of the step by the number of samples is done. This must be multiplied by 10 (all frequencies are basically 10 times higher than what you input on the device), and added to the initial frequency int(freq) to get the maximum frequency.

xvfb-run is needed if you don't have access to an x-server, since I presently using some pylab routines in the source code, which is part of matplotlib.

--include doesn't work yet (panning feature)

BG7TBL_Reader/bg7.py

```
import serial

import time

import numpy as np

import Image

from pylab import *
```

```
hold(False)

ser = serial.Serial('/dev/ttyUSB0',57600,timeout=1)

freq = "240000000" # 9

stepsize = "00011000" # 8

samples = "1000" # 4

print stepsize

arr = np.zeros( (1, int(samples)), np.uint8 )
```

```
import matplotlib.pyplot as plt

arr = []

bw = int(stepsize)*int(samples)*10

print "Bandwidth = " + str(bw) + " Hz"

print "Min freq = " + str(int(freq)*10)

print "Max freq = " + str(int(freq)*10 + int(bw))

X = np.linspace(int(freq)*10, int(freq)*10 + int(bw), int(samples))
```

```
for x in range(0, 1080):




time.sleep(5)




ser.write("x8fx78" + freq + stepsize + samples)




#print "please wait..."




time.sleep(30)




s = ser.readline()




#print s
```

byte_list = map(ord,s)

c = byte_list

sub_list = filter(lambda a: a != 0, c)[::2]

arr.append(sub_list)

ax=subplot(211)

arr2 = np.mean(arr,axis=0)

ax.autoscale(True)

```
ax.plot(X,arr2)

ax1=subplot(212)

ax1.imshow(arr,cmap=plt.cm.spectral) #Needs to be in row,col order

plt.savefig('./fig.jpg')
```

```
#print arr

#q = np.array(arr)

#im = Image.fromarray(q)

#im.save("/home/michael/specgram.jpeg")
```

```
#sub_list2 = [75*(float(x) / 255) for x in sub_list]




#print sub_list




#print len(sub_list)




#print sub_list2




#import matplotlib.pyplot as plt




#plt.plot(X,sub_list2)




#plt.show()
```

#plt.ylabel('RSSI')

#plt.show()

BG7TBL_Reader/bg_power.py

---

import sys

import os

import serial

import time

```
import numpy as np

import Image

from pylab import *

import time

from datetime import date
```

```
date = datetime.datetime.now()

csvfile = date.strftime("%Y%m%d-%H%M%S")

file = open(csvfile + '.csv', 'w')

#today = date.fromtimestamp(time.time())

hold(False)
```

```
ser = serial.Serial('/dev/ttyUSB0',57600,timeout=1)




#freq = "340000000" # 9




#stepsize = "00011000" # 8




freq = "013800000" # 9




stepsize = "00426220" # 8









samples = "1000" # 4
```

```
#print stepsize



arr = np.zeros( (1, int(samples)), np.uint8 )



arr = []



bw = int(stepsize)*int(samples)*10



#print "Bandwidth = " + str(bw) + " Hz"



#print "Min freq = " + str(int(freq)*10)
```

```
#print "Max freq = " + str(int(freq)*10 + int(bw))

X = np.linspace(int(freq)*10, int(freq)*10 + int(bw), int(samples))

for x in range(0, 1080):

time.sleep(5)

ser.write("x8fx78" + freq + stepsize + samples)

#print "please wait..."

time.sleep(30)
```

```
s = ser.readline()



#print s



byte_list = map(ord,s)



c = byte_list



sub_list = filter(lambda a: a != 0, c)[::2]



# sub_list = [((byte*0.191187)+-87.139634) for byte in sub_list]



chunks=[sub_list[x:x+10] for x in xrange(0, len(sub_list), 10)]
```

```
arr.append(sub_list)

date = datetime.datetime.now()

freq1 = int(freq)*10

freq2 = int(freq)*10

stepsizeline = int(stepsize)*10

stepsizesingle = int(stepsize)
```

```
for i in range (0, len(chunks), 1):

sub_list_str = map(str,chunks[i])

powerstring = '', ''.join(sub_list_str)

sample_max = int(samples)

sample_min = 1

freq1 = (int(freq)*10 + i*(stepsizeline*10))

freq2 = ((int(freq)*10) + (i+1)*(stepsizeline*10) - (stepsizesingle*10))
```

```
string1 = date.strftime("%Y-%m-%d") + ", " + date.strftime('%H:%M:%S') + ", " + str(freq1) + ", " + str(freq2) + ", " +
str(int(stepsize)*10) + ", " + str(int(samples)) + ", " + powerstring

print string1

sys.stdout.flush()

file.close()

#       ax=subplot(211)

#       arr2 = np.mean(arr,axis=0)
```

```
#       ax.autoscale(True)

#       ax.plot(X, arr2)

# ax1=subplot(212)

# r = int(freq)*10

# s = int(freq)*10 + int(bw)

# ax1.imshow(arr, cmap=plt.cm.spectral) #Needs to be in row,col order

#       plt.savefig('./fig.jpg')
```

```
#print arr

#q = np.array(arr)

#im = Image.fromarray(q)

#im.save("/home/michael/specgram.jpeg")
```

```
#sub_list2 = [75*(float(x) / 255) for x in sub_list]




#print sub_list




#print len(sub_list)




#print sub_list2




#import matplotlib.pyplot as plt




#plt.plot(X,sub_list2)




#plt.show()
```

```
#plt.ylabel('RSSI')

#plt.show()
```

BG7TBL_Reader/fullscan.py

```
import sys
```

```
import os

import serial

import time

import numpy as np

import Image

import argparse

from pylab import *
```

```
import time

from datetime import date

# Maximum and minimum device frequencies

maximum_frequency=4400000000

minimum_frequency=138000000

parser = argparse.ArgumentParser(description='Output heatmap.py-compatible power files for BG7TBL Spec Analyzer')
```

```
parser.add_argument('output_path', metavar='TITLE', type=str,

help='One-word-title of your job.')

slicegroup = parser.add_argument_group('Slicing',

'Efficiently render a portion of the data. (optional)  Frequencies can take G/M/k suffixes.  Timestamps look like "YYYY-MM-DD HH:MM:SS"  Durations take d/h/m/s suffixes.')

slicegroup.add_argument('--low', dest='low_freq', default=None,

help='Minimum frequency for scanning.')

slicegroup.add_argument('--high', dest='high_freq', default=None,
```

```
help='Maximum frequency for scanning.')

slicegroup.add_argument('--include', dest='include_freq', default=None,

help='Align to frequency for scanning.')

#slicegroup.add_argument('--res', dest='res_factor', default=None,

#    help='Resolution factor. Greater than 1 results in n-range scan')

#slicegroup.add_argument('--samples', dest='samples', default=None,

#    help='Number of samples')
```

```
#slicegroup.add_argument('--stepsize', dest='stepsize', default=None,

#    help='Duration to use, stopping at the end.')

for i, arg in enumerate(sys.argv):

if (arg[0] == '-') and arg[1].isdigit():

sys.argv[i] = ' ' + arg

args = parser.parse_args()

#print args.include_freq
```

```
#From keenard's code

def freq_parse(s):

suffix = 1

if s.lower().endswith('k'):

suffix = 1e3

if s.lower().endswith('m'):
```

```
suffix = 1e6

if s.lower().endswith('g'):

suffix = 1e9

if suffix != 1:

s = s[:-1]

return float(s) * suffix
```

```
def gcd(a, b):

    """Return greatest common divisor using Euclid's Algorithm."""

    while b:

        a, b = b, a % b

    return a


def lcm(a, b):
```

```
return (a * b // gcd(a, b))
```

```
low_freq = freq_parse(args.low_freq)

high_freq = freq_parse(args.high_freq)

samples = 1000

stepsize=int(((int(high_freq)-int(low_freq))/samples))/10

if (stepsize < 11000):

print "Stepsize too small. Increasing frequency interval."
```

```
stepsize=11000

high_freq=int(low_freq)+stepsize*samples

if (high_freq > maximum_frequency):

print "High frequency bound exceeded, changing high freq"

high_freq=maximum_frequency

samples=((int(high_freq)-int(low_freq))/(stepsize*10))
```

```
#print args.include_freq

if args.include_freq is None:

print "No include freq specified. Continuing..."

else:

stepsize = int((high_freq-low_freq)/(samples*10))

shift=((low_freq-(low_freq%stepsize)))
```

```
gcd_12=gcd(shift,stepsize)

# print str(gcd_12)

# print str(float(freq_parse(args.include_freq))-(float(freq_parse(args.include_freq))/gcd_12))

low_freq = float(freq_parse(args.include_freq))-(float(freq_parse(args.include_freq))/gcd_12)

# print shift

while ((low_freq - stepsize*10) > minimum_frequency):
```

```
low_freq = low_freq - (stepsize*10*1)

print "New low_freq: " + str(low_freq)

print "New high freq: " + str(low_freq+(samples*stepsize*10))

high_freq = low_freq+(samples*stepsize*10)

while (high_freq > maximum_frequency):

samples = (samples-1)
```

high_freq = low_freq+(samples*stepsize*10)

freq_int = int(int(low_freq)/10)

bw = stepsize*samples*10

```
date = datetime.datetime.now()

csvfile = date.strftime("%Y_%j__%H_%M_%S_%f")

file = open(args.output_path + '--' + csvfile + '.csv', 'w')

#today = date.fromtimestamp(time.time())

hold(False)
```

```
ser = serial.Serial('/dev/ttyUSB0',57600,timeout=1)




freq_int = int(int(low_freq)/10)




freq_str = "{0:0>9}".format(freq_int) # 9




#freq2_str = "{0:0>9}".format(freq2) # 9




#stepsize = "00011000" # 8
```

```
#freq = "013800000" # 9

stepsize_str = "{0:0>8}".format(stepsize) # 9

#stepsize2 = "00100000" # 8

samples_str = "{0:0>4}".format(samples) # 9

#samples = "1000" # 4

#print stepsize
```

```
arr = np.zeros( (1, samples), np.uint8 )
```

```
arr = []
```

```
bw = stepsize*samples*10
```

```
print "Bandwidth = " + str(bw) + " Hz"
```

```
print "Min freq = " + str(freq_int*10)
```

```
print "Max freq = " + str(freq_int*10 + int(bw))
```

```
print "Samples = " + str(samples)

print "Stepsize = " + str(stepsize*10)

X = np.linspace(freq_int*10, freq_int*10 + int(bw), samples)

print args.output_path + '--' + csvfile + '.csv'

sys.stdout.flush()

for x in range(0, 1000000):

time.sleep(5)
```

```
ser.write("x8fx78" + freq_str + stepsize_str + samples_str)




time.sleep(12)




t = ser.readline()




#print "please wait..."




#print s




byte_list = map(ord,t)
```

```
c = byte_list

sub_list = filter(lambda a: a != 0, c)[::2]

# sub_list = [((byte*0.191187)+-87.139634) for byte in sub_list]

chunks=[sub_list[x:x+10] for x in xrange(0, len(sub_list), 10)]

arr.append(sub_list)
```

```
date = datetime.datetime.now()

freq1 = freq_int*10

freq2 = freq_int*10

stepsizeline = stepsize*10

stepsizesingle = stepsize

for i in range (0, len(chunks), 1):

sub_list_str = map(str,chunks[i])
```

```
powerstring = ", ".join(sub_list_str)

sample_max = int(samples)

sample_min = 1

freq1 = (freq_int*10 + i*(stepsizeline*10))

freq2 = ((freq_int*10) + (i+1)*(stepsizeline*10) - (stepsizesingle*10))

#               freq2 = ((freq_int*10) + (i+1)*(stepsizeline*10) - 1)

string1 = date.strftime("%Y-%m-%d") + ", " + date.strftime('%H:%M:%S') + ", " + str(freq1) + ", " + str(freq2) + ", " +
str(stepsize*10) + ", " + str(samples) + ", " + powerstring + "n"
```

```
file.write(string1)

file.flush()

file.close()

#	ax=subplot(211)

#	arr2 = np.mean(arr,axis=0)
```

```
#       ax.autoscale(True)

#       ax.plot(X, arr2)

# ax1=subplot(212)

# r = int(freq)*10

# s = int(freq)*10 + int(bw)

# ax1.imshow(arr, cmap=plt.cm.spectral) #Needs to be in row,col order

#       plt.savefig('./fig.jpg')
```

```
#print arr

#q = np.array(arr)

#im = Image.fromarray(q)

#im.save("/home/michael/specgram.jpeg")
```

```
#sub_list2 = [75*(float(x) / 255) for x in sub_list]




#print sub_list




#print len(sub_list)




#print sub_list2




#import matplotlib.pyplot as plt




#plt.plot(X,sub_list2)




#plt.show()
```

#plt.ylabel('RSSI')

#plt.show()

https://github.com/DoYouKnow/BG7TBL_Reader
Contact GitHub
API
Training
Shop
Blog
About

© 2016 GitHub, Inc.
Terms
Privacy
Security