

ソフトウェア無線機

第3回目

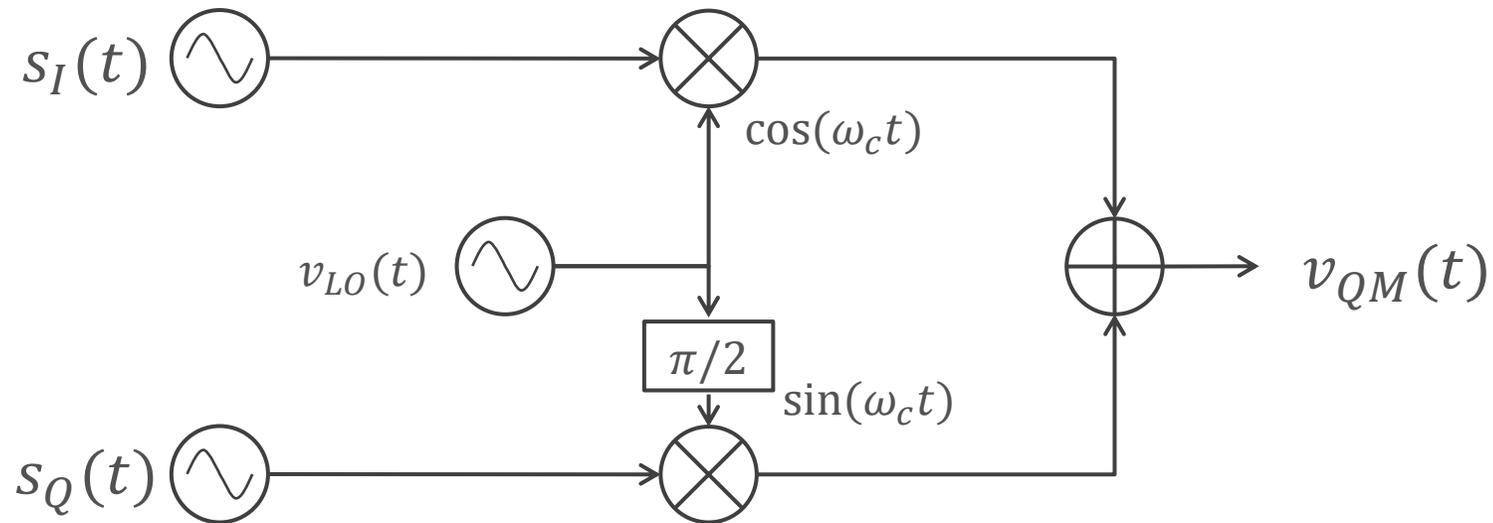
電子磁気応用資料

内容

- 全4回程度を想定（演習の進み具合で変化）
 1. 変調について
 2. 無線機のハードウェア
 3. ソフトウェアによる無線機の構成
 - ✓ 直交変復調方式について
 4. ソフトウェア無線の実習(GNU Radio使用)
 1. AMの受信
 2. FMの受信

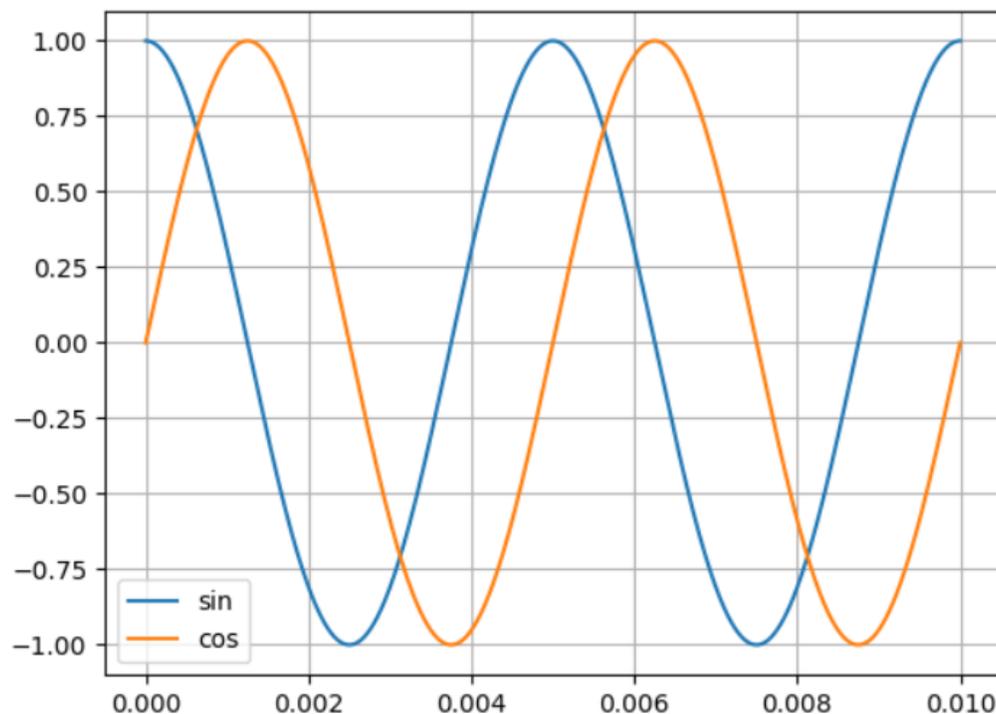
直交変調器による変調

直交変調器(IQ変調器)



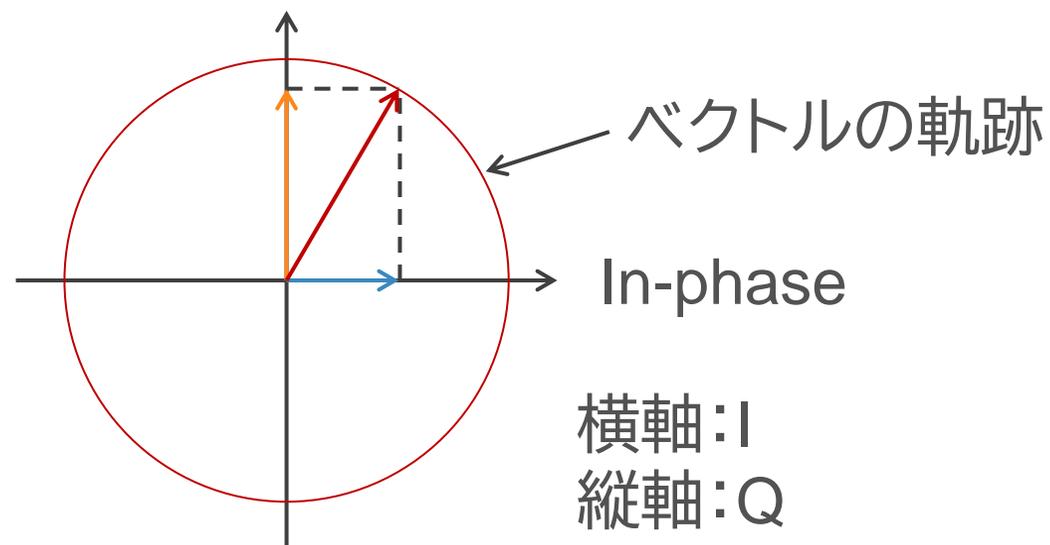
$$v_{QM}(t) = s_I(t) \cos(\omega_c t) + s_Q(t) \sin(\omega_c t)$$

sinとcosとIとQの関係



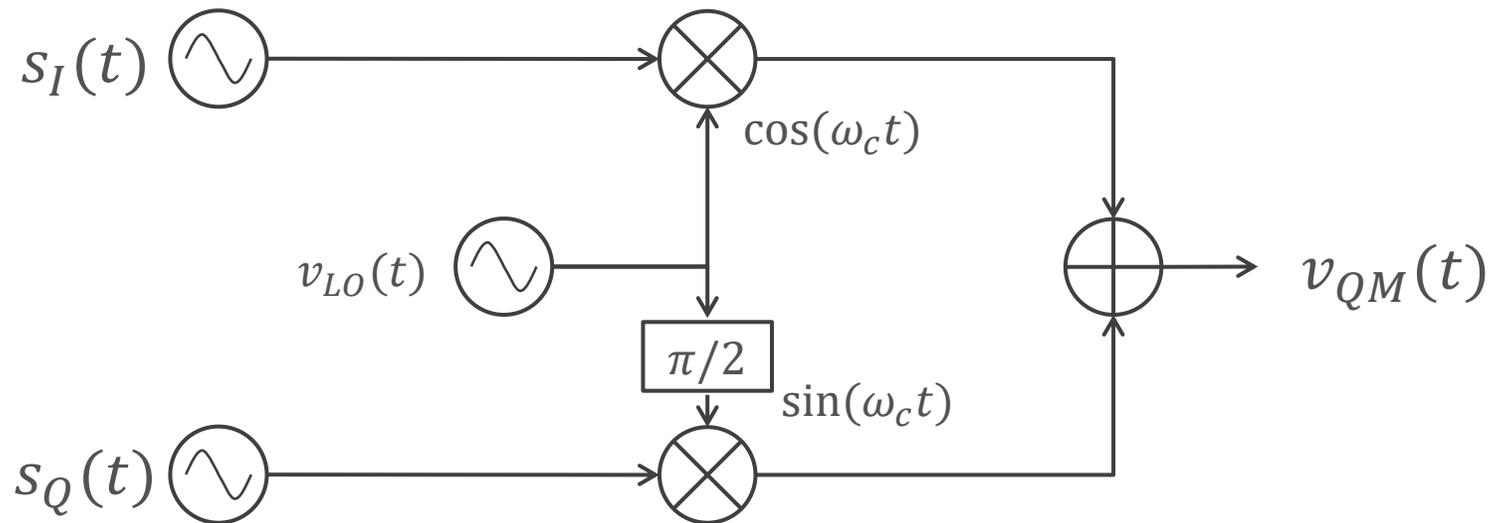
sinとcosは直交関係にある(内積がゼロ)

Quadrature-phase



Iに対応するcosとQに対応するsinの波形を合わせることで、任意の振幅、位相の波形を作り出すことができる。

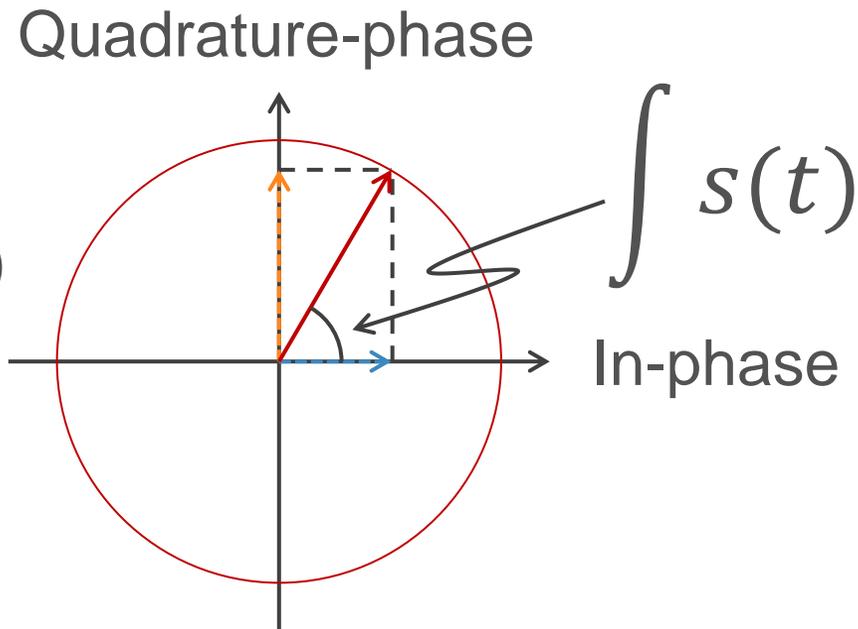
直交変調器を用いたFM



$$v_{QM}(t) = s_I(t) \cos(\omega_c t) + s_Q(t) \sin(\omega_c t)$$

$$s_I(t) = \cos \left(\omega_m \int \phi s(t) \right)$$

$$s_Q(t) = \sin \left(\omega_m \int \phi s(t) \right)$$



IとQのそれぞれの振幅に分解して入力する
入力する前に積分する

直交変調器によるFMのシミュレーションコード

```
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 0.1, 1000)
fc = 200
vci = np.cos(2*fc*np.pi*t)
vcq = -np.sin(2*fc*np.pi*t)

delta_omega = 0.05
fs = 20
vs = delta_omega * np.cos(2*fs*np.pi*t)
phi_s = np.cumsum(vs) ← 積分処理

vsi = np.cos(phi_s) ← Iの処理
vsq = np.sin(phi_s) ← Qの処理

vfm = np.cos(2*fc*np.pi*t + phi_s)
vqm = vcq*vsq + vci*vsi
```

```
fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vfm)
az = fig.add_subplot(413)
az.grid()
az.plot(t, vsi)
az.plot(t, vsq)
aa = fig.add_subplot(414)
aa.grid()
aa.plot(t, vqm)

axes = fig.get_axes()
for n in axes:
    n.tick_params(axis='x', which='both',
bottom=False, labelbottom=False)
n.tick_params(axis='x', which='both',
bottom=True, labelbottom=True)
```

Colabを用いたシミュレーション

```

import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 0.1, 1000)
fc = 200
vci = np.cos(2*fc*np.pi*t)
vcq = -np.sin(2*fc*np.pi*t)

delta_omega = 0.05
fs = 20
vs = delta_omega * np.cos(2*fs*np.pi*t)
phi_s = np.cumsum(vs)

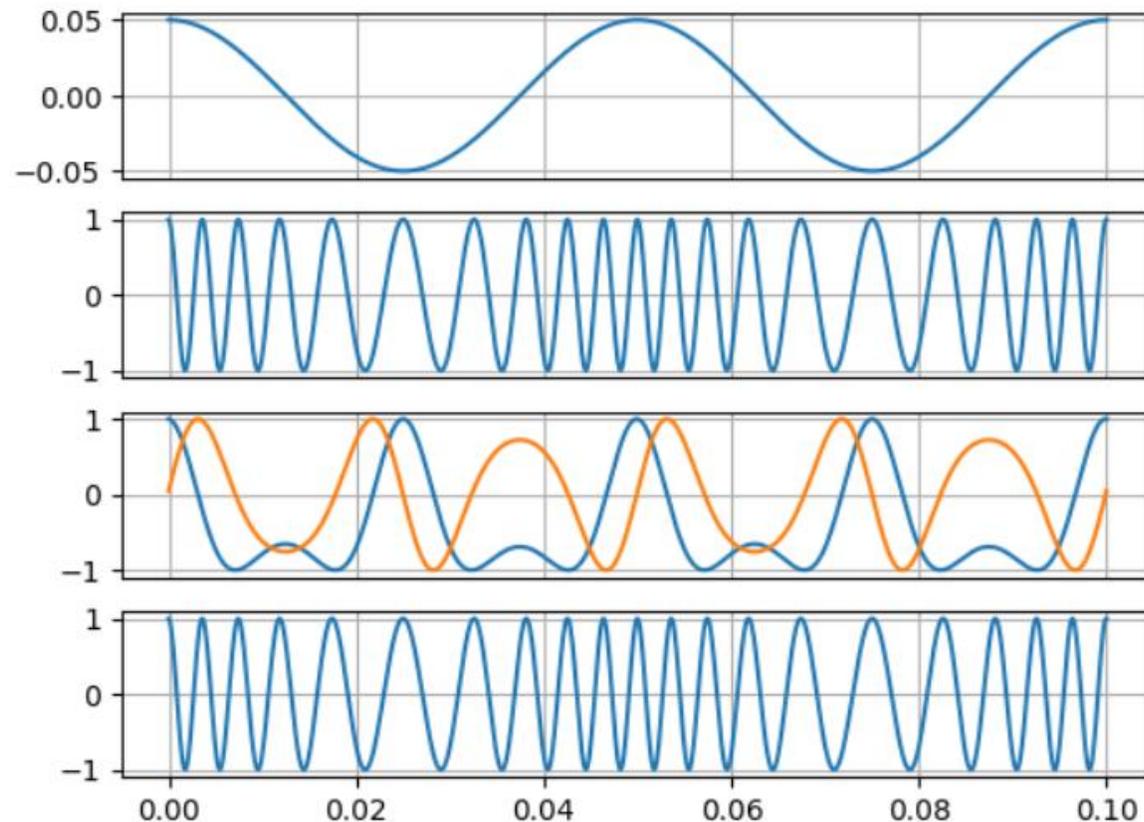
vsi = np.cos(phi_s)
vsq = np.sin(phi_s)

vfm = np.cos(2*fc*np.pi*t + phi_s)
vqm = vcq*vsq + vci*vsi

fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vfm)
az = fig.add_subplot(413)
az.grid()
az.plot(t, vsi)
az.plot(t, vsq)
aa = fig.add_subplot(414)
aa.grid()
aa.plot(t, vqm)

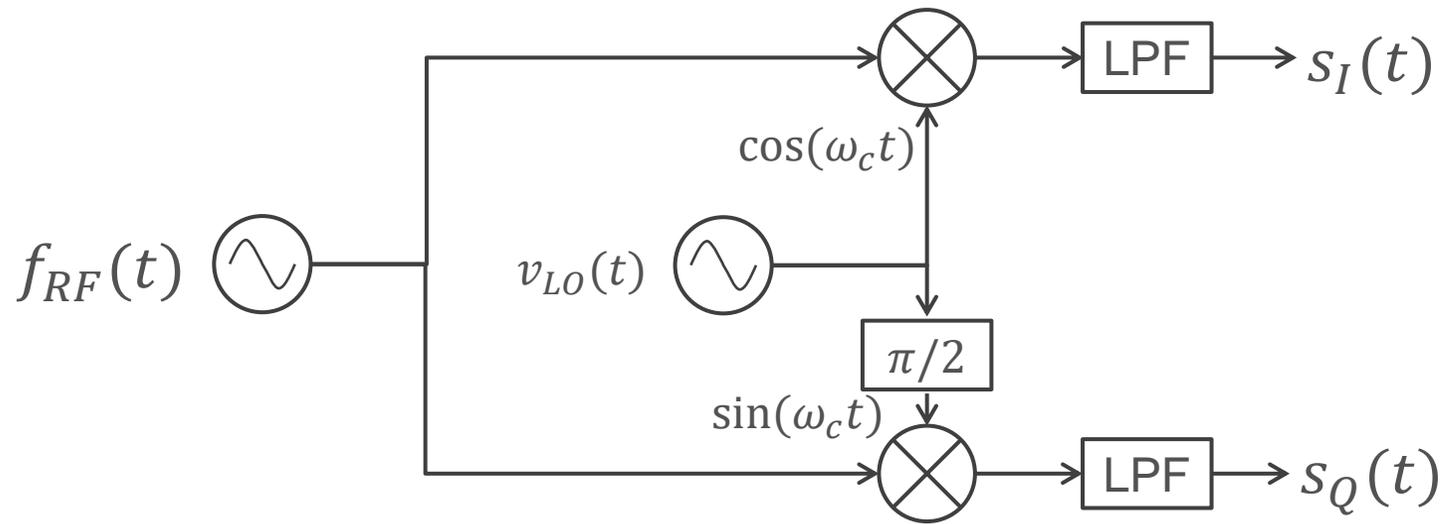
axes = fig.get_axes()
for n in axes:
    n.tick_params(axis='x', which='both', bottom=False, labelbottom=False)
    n.tick_params(axis='x', which='both', bottom=True, labelbottom=True)

```

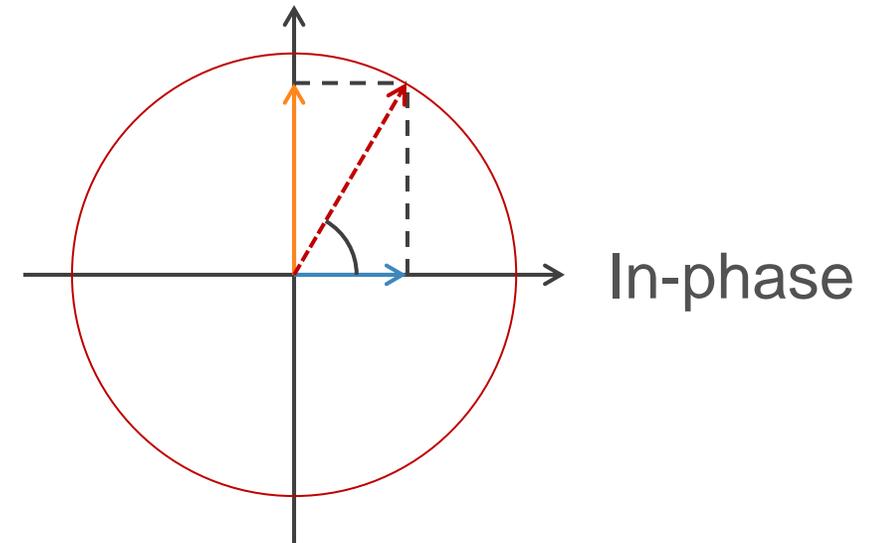


直交復調器による復調

直交復調器(IQ復調器)



Quadrature-phase



直交復調器を用いるとLOと

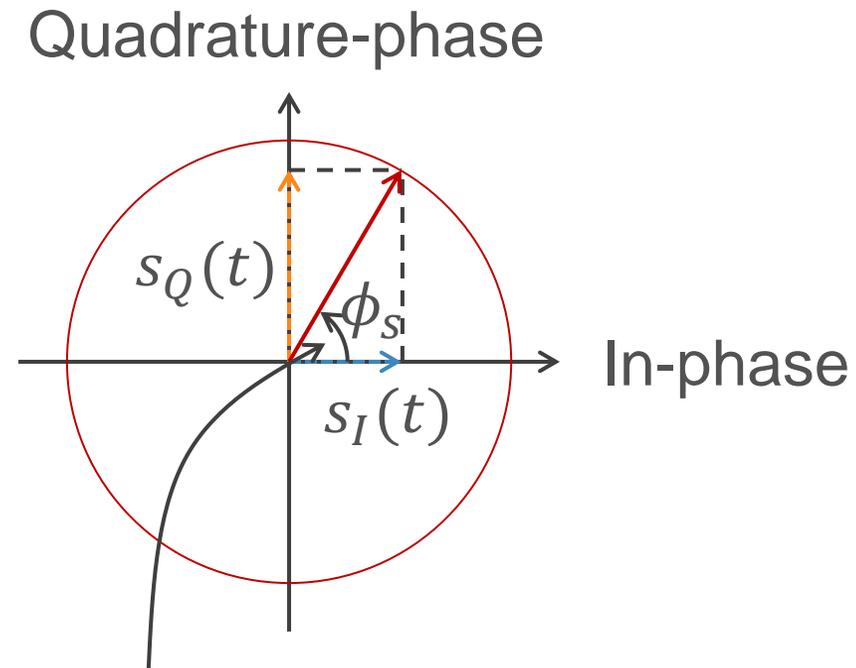
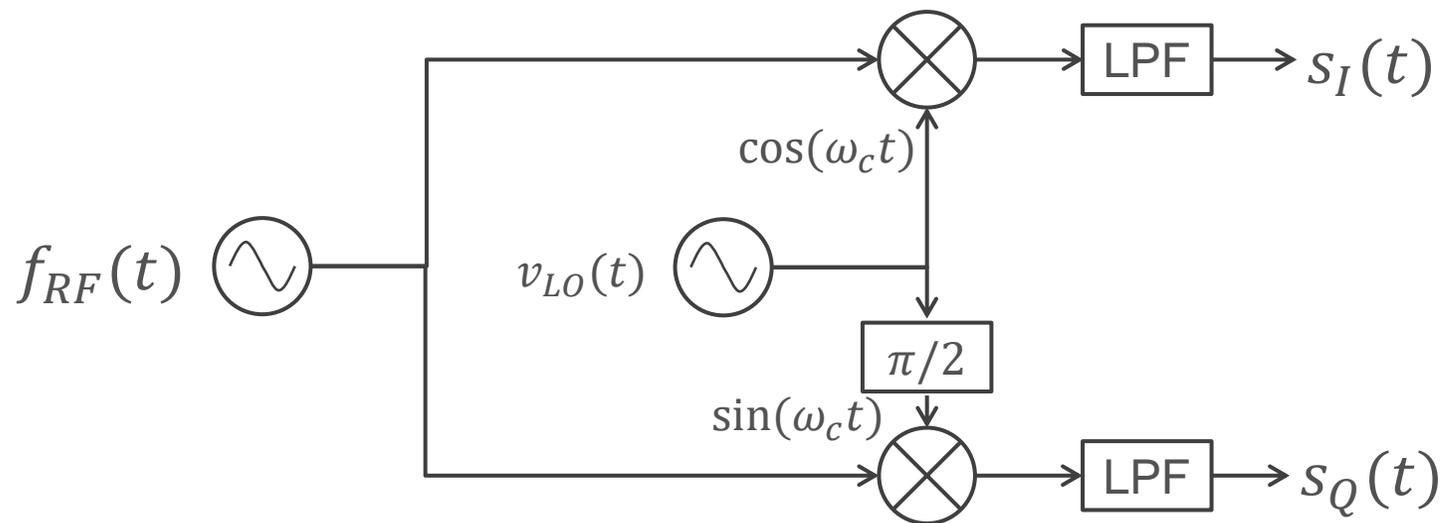
- 同相の信号成分(I)
- 直交する信号成分(Q)

の2つを取り出すことができる。

IとQの2つの信号を処理することで、
変調前の信号を取り出すことができる。

➡ ソフトウェアで無線機を実現できる

直交復調器を用いたPM復調



元の信号を取り出すには角度成分を取り出す。

角度成分はアークタンジェントを取ればいいので、2つのなす角 ϕ_s が元の信号の成分

$$s(t) = \arctan \frac{S_Q(t)}{S_I(t)}$$

と書くことができる。

直交復調器によるPM復調のシミュレーションコード

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

t = np.linspace(0, 0.1, 1000)
fc = 500
vci = np.cos(2*fc*np.pi*t)
vcq = -np.sin(2*fc*np.pi*t)

delta_phi = 60/180*np.pi
fs = 20
vs = delta_phi * np.sin(2*fs*np.pi*t)

vpm = np.sin(2*fc*np.pi*t + vs)

vifi = vpm*vci
vifq = vpm*vcq

v_i_dec = signal.decimate(vifi, 5)
v_q_dec = signal.decimate(vifq, 5)

v_s_demod = np.angle((v_i_dec + v_q_dec * 1.j))
```

```
fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vqm)
az = fig.add_subplot(413)
az.grid()
az.plot(t[::5], v_i_dec)
az.plot(t[::5], v_q_dec)
aa = fig.add_subplot(414)
aa.plot(t[::5], v_s_demod)
aa.grid()
axes = fig.get_axes()
for n in axes:
    n.tick_params(axis='x', which='both',
bottom=False, labelbottom=False)
    n.tick_params(axis='x', which='both',
bottom=True, labelbottom=True)
```

角度を求める
(アークタンジェント)

複素数として扱う

Colabを用いたシミュレーション

```

import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

t = np.linspace(0, 0.1, 1000)
fc = 500
vci = np.cos(2*fc*np.pi*t)
vcq = -np.sin(2*fc*np.pi*t)

delta_phi = 60/180*np.pi
fs = 20
vs = delta_phi * np.sin(2*fs*np.pi*t)

vpm = np.cos(2*fc*np.pi*t + vs)

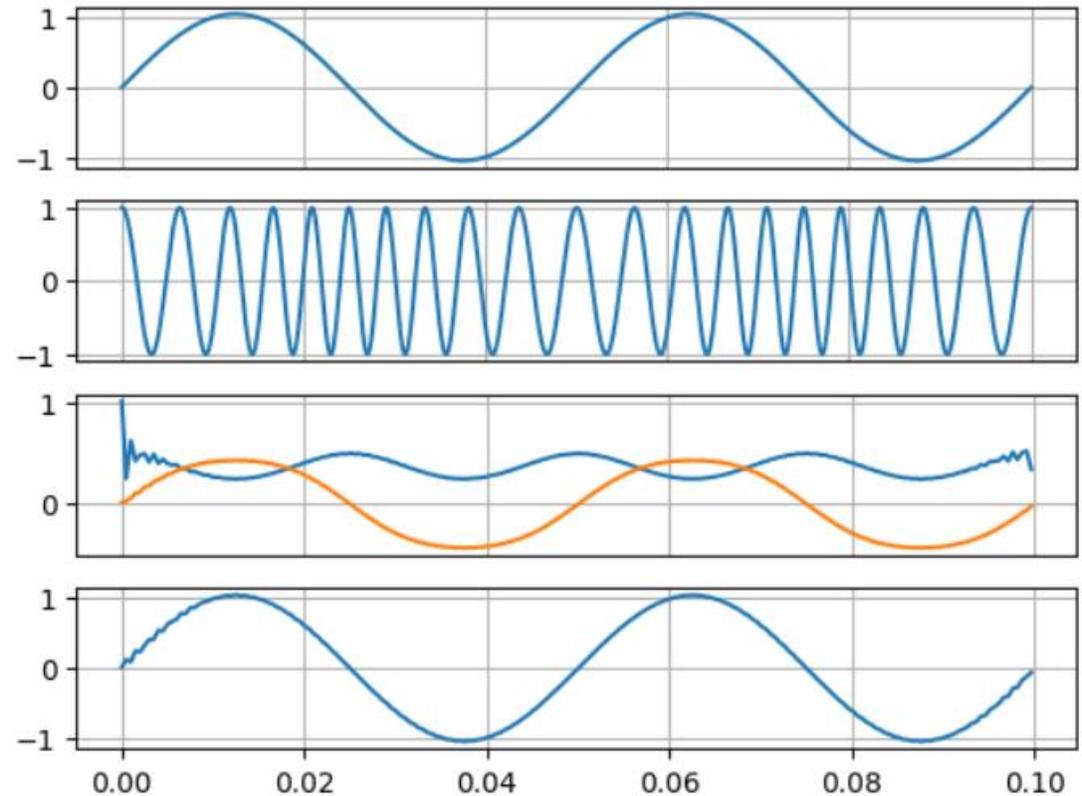
vifi = vpm*vci
vifq = vpm*vcq

v_i_dec = signal.decimate(vifi, 5)
v_q_dec = signal.decimate(vifq, 5)

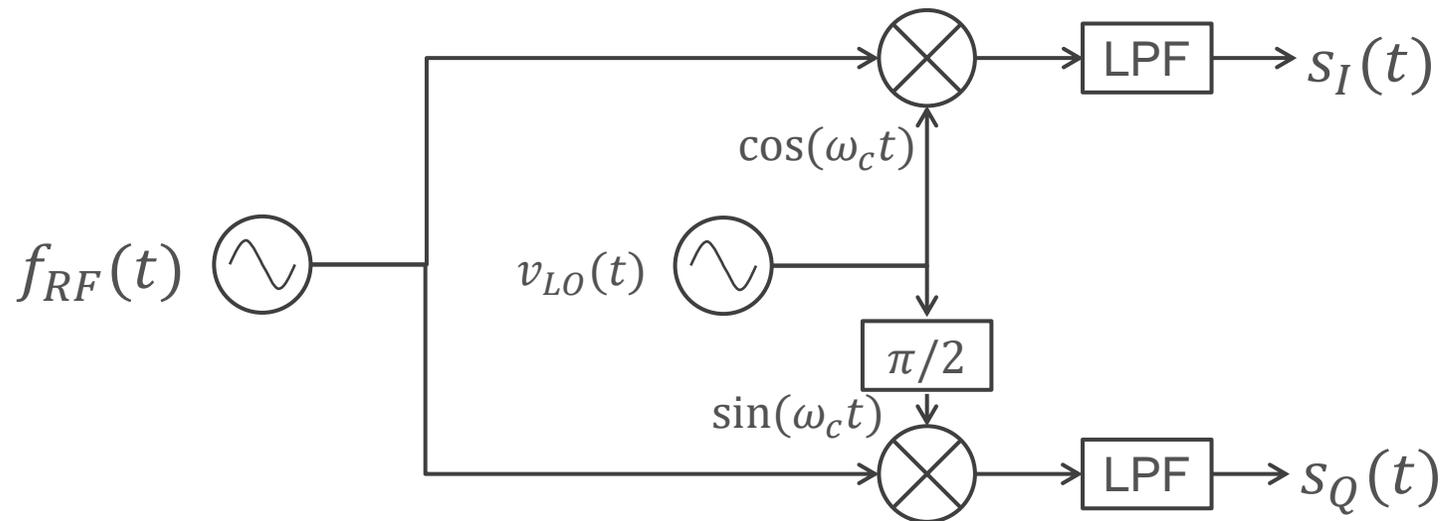
v_s_demod = np.angle((v_i_dec + v_q_dec * 1.j))

fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vqm)
az = fig.add_subplot(413)
az.grid()
az.plot(t[:5], v_i_dec)
az.plot(t[:5], v_q_dec)
aa = fig.add_subplot(414)
aa.grid()
aa.plot(t[:5], v_s_demod)
aa.grid()
axes = fig.get_axes()
for n in axes:
    n.tick_params(axis='x', which='both', bottom=False, labelbottom=False)
    n.tick_params(axis='x', which='both', bottom=True, labelbottom=True)

```



直交復調器を用いたFM復調



元の信号を取り出すには角度の差分を取り出す。

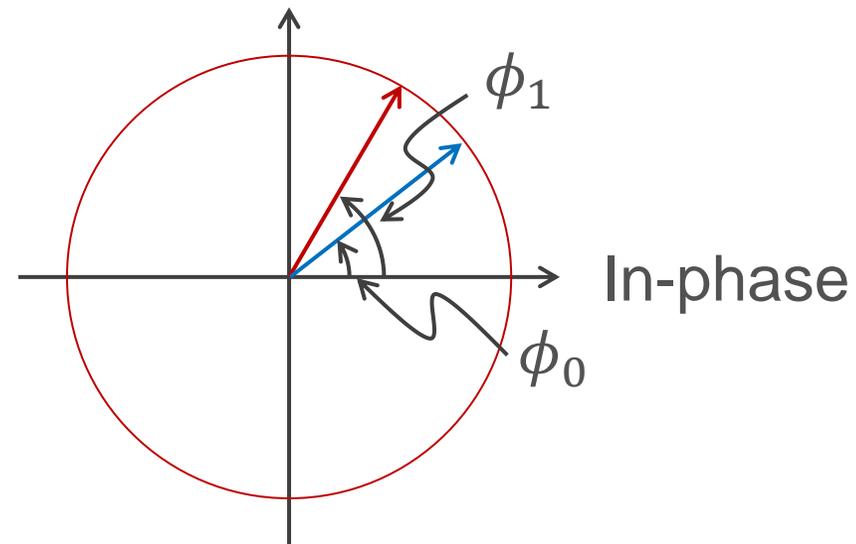
$$S_n = \phi_{n+1} - \phi_n$$

また角度はアークタンジェントを用いて、

$$\phi_n = \arctan \frac{S_{n,Q}}{S_{n,I}}$$

と書くことができる。

Quadrature-phase



信号の角度の変化量(微分値)を求める

直交変調器によるFM復調のシミュレーションコード

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

t = np.linspace(0, 0.1, 1000)
fc = 200
vci = np.cos(2*fc*np.pi*t)
vcq = -np.sin(2*fc*np.pi*t)

delta_omega = 0.015
fs = 20
vs = delta_omega * np.cos(2*fs*np.pi*t)
phi_s = np.cumsum(vs)

vfm = np.sin(2*fc*np.pi*t + phi_s)

vifi = vfm*vci
vifq = vfm*vcq

num_decm = 20

v_i_dec = signal.decimate(vifi, num_decm)
v_q_dec = signal.decimate(vifq, num_decm)

v_s_demod_phase = np.arctan2(v_q_dec, v_i_dec)
v_s_demod_diff = np.diff(v_s_demod_phase)
```

```
fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vfm)
az = fig.add_subplot(413)
az.grid()
az.plot(t[::num_decm], v_i_dec)
az.plot(t[::num_decm], v_q_dec)
aa = fig.add_subplot(414)
aa.plot(t[::num_decm][1:], v_s_demod_diff)
aa.set_ylim((-0.5, 0.5))
aa.grid()
axes = fig.get_axes()
for n in axes:
    n.tick_params(axis='x', which='both',
bottom=False, labelbottom=False)
n.tick_params(axis='x', which='both',
bottom=True, labelbottom=True)
```

} 角度を求める処理と差分を求める処理

Colabを用いたシミュレーション

```

import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

t = np.linspace(0, 0.1, 1000)
fc = 200
vci = np.cos(2*fc*np.pi*t)
vcq = -np.sin(2*fc*np.pi*t)

delta_omega = 0.015
fs = 20
vs = delta_omega * np.cos(2*fs*np.pi*t)
phi_s = np.cumsum(vs)

vfm = np.sin(2*fc*np.pi*t + phi_s)

vifi = vfm*vci
vifq = vfm*vcq

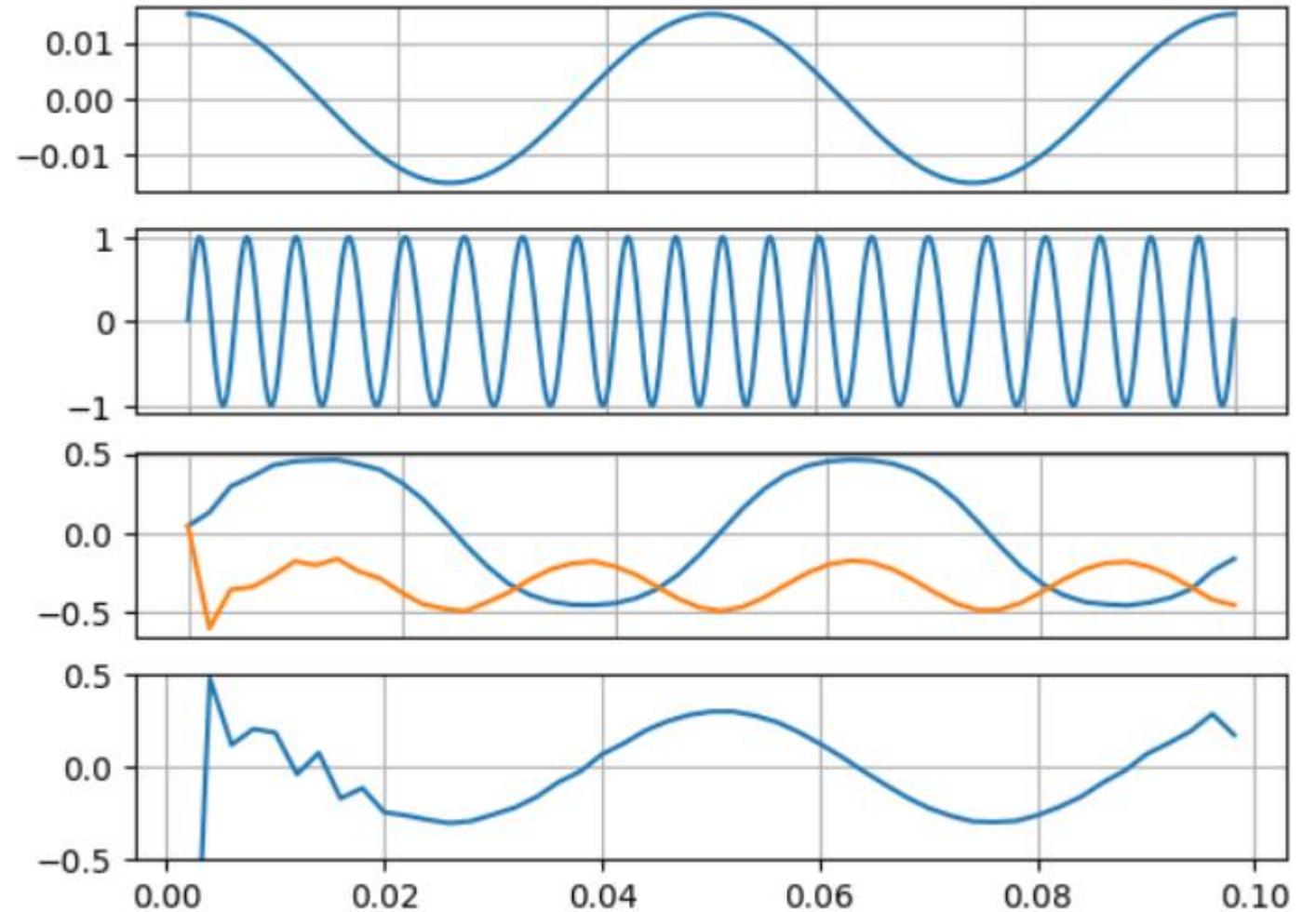
num_decm = 20

v_i_dec = signal.decimate(vifi, num_decm)
v_q_dec = signal.decimate(vifq, num_decm)

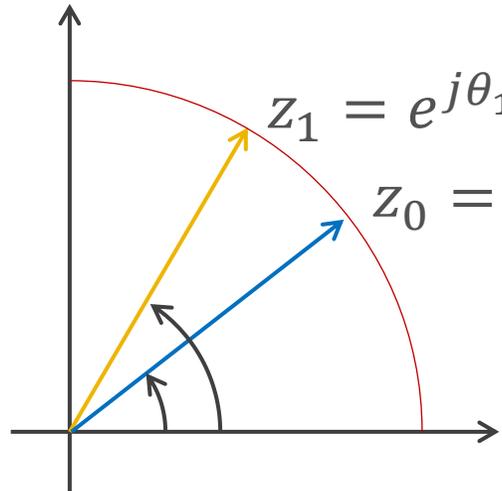
v_s_demod_phase = np.arctan2(v_q_dec, v_i_dec)
v_s_demod_diff = np.diff(v_s_demod_phase)

fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vfm)
az = fig.add_subplot(413)
az.grid()
az.plot(t[::num_decm], v_i_dec)
az.plot(t[::num_decm], v_q_dec)
aa = fig.add_subplot(414)
aa.grid()
aa.plot(t[::num_decm][1:], v_s_demod_diff)
aa.set_ylim((-0.5,0.5))
axes = fig.get_axes()
for n in axes:
    n.tick_params(axis='x', which='both', bottom=False, labelbottom=False)
    n.tick_params(axis='x', which='both', bottom=True, labelbottom=True)

```



ちょっとスマートなFM復調



$$z_1 = e^{j\theta_1}$$

$$z_0 = e^{j\theta_0}$$

それぞれのベクトルは複素数を用いて表現もできる。
複素数を用いる $e^{j\theta}$ と表現できる。

このとき2つの角度の差分は $e^{j(\theta_1 - \theta_0)}$ と表現できる。
これは分解すると、

$$e^{j(\theta_1 - \theta_0)} = e^{j\theta_1} e^{-j\theta_0} = z_1 \hat{z}_0$$

となる。ここで \hat{z}_0 は z_0 の複素共役を示す。

このように、角度の差分を表す複素数は積を用いて表現することもできる。
角度に戻すために \arctan を取ると、元の信号を再生することができる。

ちょっとスマートな直交変調器によるFM復調のシミュレーションコード

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

t = np.linspace(0, 0.1, 1000)
fc = 200
vci = np.cos(2*fc*np.pi*t)
vcq = -np.sin(2*fc*np.pi*t)

delta_omega = 0.015
fs = 20
vs = delta_omega * np.cos(2*fs*np.pi*t)
phi_s = np.cumsum(vs)

vfm = np.sin(2*fc*np.pi*t + phi_s)

vifi = vfm*vci
vifq = vfm*vcq

v_i_dec = signal.decimate(vifi, 20)
v_q_dec = signal.decimate(vifq, 20)

v_s_demod = v_i_dec + v_q_dec*1.j ← 複素数に変換する処理
v_s_demod_diff = v_s_demod[1:]*np.conj(v_s_demod[:-1]) ← 1つ前の値の複素共役を取って積を取る処理
v_s_demod_phase = np.angle(v_s_demod_diff) ← 角度に変換する処理
```

```
fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vfm)
az = fig.add_subplot(413)
az.grid()
az.plot(t[::20], v_i_dec)
az.plot(t[::20], v_q_dec)
aa = fig.add_subplot(414)
aa.plot(t[::20][1:], v_s_demod_phase)
aa.set_ylim((-0.5, 0.5))
aa.grid()
axes = fig.get_axes()
for n in axes:
    n.tick_params(axis='x', which='both',
bottom=False, labelbottom=False)
n.tick_params(axis='x', which='both',
bottom=True, labelbottom=True)
```

FMラジオの実装

今回作成するブロックダイアグラム

my_fm.grc - C:\Users\Masahiro\Desktop\gnuradio_test

File Edit View Run Tools Help

test_fm x my_fm x

Options
Title: Not titled yet
Author: Masahiro
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Values: 2.4M

QT GUI Range
ID: center_freq
Label: center_freq
Default Value: 80M
Start: 70M
Stop: 100M
Step: 10k

Soapy RTLSDR Source
Sample Rate: 2.4M
Center Freq (Hz): 80M

Low Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 2.4M
Cutoff Freq: 100k
Transition Width: 30k
Window: Hamming
Beta: 6.76

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2.4M
Autoscales: Yes

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 2.4M

Delay
Delay: 1

Complex conjugate

Multiply

Complex To Mag Phase

NULL Sink

Band Pass Filter
Decimation: 10
Gain: 1
Sample Rate: 480k
Low Cutoff Freq: 30
High Cutoff Freq: 5k
Transition Width: 10k
Window: Hamming
Beta: 6.76

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 480k
Autoscales: Yes

Audio Sink
Sample Rate: 48 kHz

> ADS-B
> Advanced File
> Core
 > ADALM-2000
 > Audio
 > Boolean Operators
 > Byte Operators
 > Channel Models
 > Channelizers
 > Coding
 > Control Port
 > Debug Tools
 > Deprecated
 > Digital Television
 > Equalizers
 > Error Coding
 > File Operators
 > Filters
 > Fourier Analysis
 > GUI Widgets
 > Impairment Models
 > Industrial I/O
 > Instrumentation
 > IQ Balance
 > IQ Correction
 > Level Controllers
 > Math Operators
 > Measurement Tools
 > Message Tools
 > Misc
 > Modulators
 > Networking Tools
 > OFDM
 > Packet Operators
 > PDU Tools
 > Peak Detectors
 > Resamplers
 > Soapy
 > Stream Operators
 > Stream Tag Tools
 > Symbol Coding
 > Synchronizers
 > Trellis Coding
 > Type Converters
 > UHD
 > Variables
 > Video

<<< Welcome to GNU Radio Companion 3.10.10.0 >>>

Block paths:
C:\ProgramData\radioconda\Library\share\gnuradio\grc\blocks

Loading: "C:\Users\Masahiro\Desktop\gnuradio_test\test_fm.grc"
>>> Done

Loading: "C:\Users\Masahiro\Desktop\gnuradio_test\my_fm.grc"
>>> Done

ID	Value
Imports	
Variables	
center_freq	80000000.0
samp_rate	2400000.0

変数の追加と設定

The screenshot shows the GNU Radio Companion (GRC) interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with various icons for file operations and execution. The main workspace contains three panels: 'Options' (Title: Not titled yet, Author: user, Output Language: Python, Generate Options: QT GUI), 'Variable' (ID: samp_rate, Value: 2.4M), and another 'Variable' (ID: freq, Value: 80M). A right-hand sidebar displays a tree view of GRC blocks, including ADS-B, Advanced File, Core, ADALM-2000, Audio, Boolean Operators, Byte Operators, Channel Models, Channelizers, Coding, Control Port, Debug Tools, Deprecated, Digital Television, Equalizers, Error Coding, File Operators, Filters, Fourier Analysis, GUI Widgets, instrumentation, IQ Balance, IQ Correction, measurement tools, and Message Tools.

The terminal window at the bottom left shows the following output:

```
C:\Users\user\radioconda\Library\share\gnuradio\grc\blocks
Loading: "C:\Users\user\Desktop\sdr\fm_only_TokyoFM.grc"
>>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJTT_ATIS.grc"
>>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJTT_ATIS_copy.grc"
>>> Done
```

The 'Variables' table in the center shows the following data:

ID	Value
freq	8000000.0
samp_rate	2400000.0

Yellow arrows point from the terminal output to the variable table. One arrow points from the 'freq' row to the terminal output, and another points from the 'samp_rate' row to the terminal output. The text 'freqを作成して、80e6を入力' (Create freq and input 80e6) is positioned above the 'freq' row, and 'samp_rateに2.4e6を入力' (Input 2.4e6 to samp_rate) is positioned below the 'samp_rate' row.

RTL SDRを追加

虫眼鏡(検索)を左クリック

Soapyを入力

枠内にドラッグ
アンドドロップ

Options
Title: Not titled yet
Author: user
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 2.4M

Variable
ID: freq
Value: 80M

Soapy RTLSDR Source
Sample Rates: 2.4M
Center Freq (Hz): 80M

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	24000000.0

```

C:\Users\user\radioconda\Library\share\gnuradio\grc\blocks
Loading: "C:\Users\user\Desktop\sd\fm_only_TokyoFM.grc"
>>> Done
Loading: "C:\Users\user\Desktop\sd\am_only_RJTJ_ATIS.grc"
>>> Done
Loading: "C:\Users\user\Desktop\sd\am_only_RJTJ_ATIS_copy.grc"
>>> Done
  
```

Low pass filterの追加

*untitled - GNU Radio Companion

File Edit View Run Tools Help

fm_only_TokyoFM × am_only_RJTT_ATIS × am_only_RJTT_ATIS_copy × untitled ×

Options
 Title: Not titled yet
 Author: user
 Output Language: Python
 Generate Options: QT GUI

Variable
 ID: samp_rate
 Value: 2.4M

Variable
 ID: freq
 Value: 80M

Low Pass Filter
 Decimation: 1
 Gain: 1
 Sample Rate: 2.4M
 Cutoff Freq: 80M
 Transition Width:
 Window: Hamming
 Beta: 6.76

Soapy RTLSDR Source
 Sample Rate: 2.4M
 Center Freq (Hz): 80M

cmd

out

in

out

「low pass」を検索

Low Pass Filter

Low-pass Filter Taps

枠内にドラッグ
アンドドロップ

OutからInに配線をつなぐ

C:

```

¥Users¥user¥radioconda¥Library¥share¥gnuradio¥grc¥blocks
Loading: "C:¥Users¥user¥Desktop¥sdr¥fm_only_TokyoFM.grc"
>>> Done

Loading: "C:¥Users¥user¥Desktop¥sdr¥am_only_RJTT_ATIS.grc"
>>> Done

Loading: "C:
¥Users¥user¥Desktop¥sdr¥am_only_RJTT_ATIS_copy.grc"
>>> Done
  
```

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	2400000.0

Low pass filter の設定

Low pass filterブロックをダブルクリックして設定画面を開く

Properties: Low Pass Filter

General Advanced Documentation

FIR Type Complex->Complex (Decimating)

Decimation 5 [int]

Gain 1 [real]

Sample Rate samp_rate [real]

Cutoff Freq 100e3 [real]

Transition Width 30e3 [real]

Window Hamming

Beta 6.76 [real]

Param - Transition Width(width):
Expression None is invalid for type 'real'.

OK Cancel Apply

5を入力

100e3を入力

30e3を入力

入力が終わったらOKをクリック

ut

Value

Delayの追加

*untitled - GNU Radio Companion

File Edit View Run Tools Help

fm_only_TokyoFM x am_only_RJTT_ATIS x am_only_RJTT_ATIS_copy x untitled x

Options
Title: Not titled yet
Author: user
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 2.4M

Variable
ID: freq
Value: 80M

「Delay」を検索

Q Dela

Core
Misc
Delay
Message Tools
Message Strobe Random-Delay
Filters
Filter Delay

枠内にドラッグ
アンドドロップ

OutからInに配線をつなぐ

cmd Soapy RTLSDR Source
Sample Rate: 2.4M
Center Freq (Hz): 80M

Low Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 2.4M
Cutoff Freq: 100k
Transition Width: 30k
Window: Hamming
Beta: 6.76

Delay
Delay: 0

C:
¥Users¥user¥radioconda¥Library¥share¥gnuradio¥grc¥blocks
Loading: "C:¥Users¥user¥Desktop¥sdr¥fm_only_TokyoFM.grc"
>>> Done
Loading: "C:¥Users¥user¥Desktop¥sdr¥am_only_RJTT_ATIS.grc"
>>> Done
Loading: "C:
¥Users¥user¥Desktop¥sdr¥am_only_RJTT_ATIS_copy.grc"
>>> Done

ID	Value	
Imports		+
Variables		+
freq	80000000.0	x
samp_rate	24000000.0	x

Delayの設定

The image shows a software dialog box titled "Properties: Delay" with three tabs: "General", "Advanced", and "Documentation". The "General" tab is selected. It contains several input fields: "Type" (a dropdown menu set to "complex"), "Delay" (a text box containing "1", highlighted with a yellow box and an arrow pointing to it from the text "1を入力"), "Num Ports" (a text box containing "1"), "Vector Length" (a text box containing "1"), and "Show Msg Ports" (a dropdown menu set to "No"). At the bottom, there is a status area with the text "Source - out(0): Port is not connected." and three buttons: "OK", "Cancel", and "Apply". The "OK" button is highlighted with a yellow box and an arrow pointing to it from the text "入力が終わったらOKをクリック".

Property	Value	Type
Type	complex	[enum]
Delay	1	[int]
Num Ports	1	[int]
Vector Length	1	[int]
Show Msg Ports	No	[bool]

Source - out(0):
Port is not connected.

Value

1を入力

入力が終わったらOKをクリック

Complex conjugateの追加

「Complex」を検索

The screenshot shows the GNU Radio Companion interface. On the left, there are three panels: 'Options' (Title: Not titled yet, Author: user, Output Language: Python, Generate Options: QT GUI), 'Variable' (ID: samp_rate, Value: 2.4M), and another 'Variable' (ID: freq, Value: 80M). The main workspace contains a block diagram with the following components and connections:

- Soapy RTLSDR Source** (cmd): Sample Rate: 2.4M, Center Freq (Hz): 80M. Its 'out' is connected to the 'in' of the **Low Pass Filter**.
- Low Pass Filter**: Decimation: 5, Gain: 1, Sample Rate: 2.4M, Cutoff Freq: 100k, Transition Width: 30k, Window: Hamming, Beta: 6.76. Its 'out' is connected to the 'in' of the **Delay** block.
- Delay**: Delay: 1. Its 'out' is connected to the 'in' of the **Complex Conjugate** block.
- Complex Conjugate** block is highlighted in red in the diagram.

OutからInに配線をつなぐ

枠内にドラッグ
アンドドロップ

Search results for 'Complex':

- Core
 - Type Converters
 - Complex to Arg
 - Complex To Float
 - Complex to Imag
 - Complex To IChar
 - Complex To IShort
 - Complex to Mag
 - Complex to Mag^2
 - Complex To Mag Phase
 - Complex To Real
 - IChar To Complex
 - IShort To Complex
 - Magnitude and Phase To Complex
 - Math Operators
 - Complex Conjugate** (highlighted)
 - Measurement Tools
 - Ctrlport Complex Probe
 - Waveform Generators
 - VCO (complex)
 - Iridium
 - iuchar_to_complex

```

C:
¥Users¥user¥radioconda¥Library¥share¥gnuradio¥grc¥blocks
Loading: "C:¥Users¥user¥Desktop¥sdr¥fm_only_TokyoFM.grc"
>>> Done
Loading: "C:¥Users¥user¥Desktop¥sdr¥am_only_RJTT_ATIS.grc"
>>> Done
Loading: "C:
¥Users¥user¥Desktop¥sdr¥am_only_RJTT_ATIS_copy.grc"
>>> Done
  
```

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	2400000.0

Multiplyを追加

*untitled - GNU Radio Companion

File Edit View Run Tools Help

fm_only_TokyoFM x am_only_RJTJ_ATIS x am_only_RJTJ_ATIS_copy x untitled x

Options
Title: Not titled yet
Author: user
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 2.4M

Variable
ID: freq
Value: 80M

「Multiply」を検索

Core
Math Operators
Multiply by Tag Value
Multiply Conjugate
Multiply Const
Fast Multiply Const
Multiply by Matrix
Multiply

Message Tools
Tagged Stream Multiply Length Tag

IEEE802.15.4
Multiuser Chirp Detector

RADAR
RADAR
Static Target Simulator

Satellites
CCSDS
Path ID Demultiplexer
Virtual Channel Demultiplexer

枠内にドラッグ
アンドドロップ

OutからInに配線をつなぐ

Soapy RTLSDR Source
Sample Rate: 2.4M
Center Freq (Hz): 80M

Low Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 2.4M
Cutoff Freq: 100k
Transition Width: 30k
Window: Hamming
Beta: 6.76

Delay
Delay: 1

Complex Conjugate

Multiply

```
C:\>
C:\Users\user\radioconda\Library\share\gnuradio\grc\blocks
Loading: "C:\Users\user\Desktop\sdr\fm_only_TokyoFM.grc"
>>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJTJ_ATIS.grc"
>>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJTJ_ATIS_copy.grc"
>>> Done
```

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	2400000.0

Complex to Mag Phaseを追加

GNU Radio Companion interface showing a flow graph and a block search menu.

「Complex」を検索 (Search for "Complex")

枠内にドラッグアンドドロップ (Drag and drop into the frame)

OutからInに配線をつなぐ (Connect wiring from Out to In)

The flow graph consists of the following blocks and connections:

- Soapy RTLSDR Source** (Sample Rate: 2.4M, Center Freq: 80M) connects to **Low Pass Filter**.
- Low Pass Filter** (Decimation: 5, Gain: 1, Sample Rate: 2.4M, Cutoff Freq: 100k, Transition Width: 30k, Window: Hamming, Beta: 6.76) connects to **Multiply**.
- Delay** (Delay: 1) connects to **Complex Conjugate**.
- Complex Conjugate** connects to **Multiply**.
- Multiply** connects to **Complex To Mag Phase**.

The search menu on the right shows the following items:

- Core
 - Type Converters
 - Complex to Arg
 - Complex To Float
 - Complex to Imag
 - Complex To IChar
 - Complex To IShort
 - Complex to Mag
 - Complex to Mag^2
 - Complex To Mag Phase** (highlighted)
 - Complex To Real
 - Float To Complex
 - IChar To Complex
 - IShort To Complex
 - Magnitude and Phase To Compl
 - Math Operators
 - Complex Conjugate
 - Measurement Tools
 - Ctrlport Complex Probe
 - Waveform Generators
 - VCO (complex)
 - GUI Widgets
 - QT
 - QT GUI Compass
 - Iridium
 - iuchar_to_complex
 - Satellites
 - Deframers

Block paths:

```
C:\Users\user\radioconda\Library\share\gnuradio\grc\blocks
```

Loading: "C:\Users\user\Desktop\sdr\fm_only_TokyoFM.grc"
>>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJTT_ATIS.grc"
>>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJTT_ATIS_copy.grc"

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	2400000.0

Null Sinkの追加

GNU Radio Companion (GRC) のスクリーンショット。画面には「untitled - GNU Radio Companion」と表示されています。

右側のコンポーネントパネルで「Null Sink」が選択されています。黄色い矢印が「Null」を検索した結果を示しています。

中央のワークスペースには、以下のブロックが接続されています:

- Soapy RTLSDR Source (Sample Rate: 2.4M, Center Freq (Hz): 80M)
- Low Pass Filter (Decimation: 5, Gain: 1, Sample Rate: 2.4M, Cutoff Freq: 100k, Transition Width: 30k, Window: Hamming, Beta: 6.76)
- Delay (Delay: 1)
- Complex Conjugate
- Multiply (in0, in1, out)
- Complex To Mag Phase (mag, phase)
- Null Sink

黄色い矢印が「OutからInに配線をつなぐ」ことを示しています。また、「Null Sink」をワークスペースにドラッグアンドドロップしたことを示す矢印も含まれています。

下部には、Block paths、Loading status、および Variables table が表示されています。

```

Block paths:
  C:\Users\user\radioconda\Library\share\gnuradio\grc\blocks

Loading: "C:\Users\user\Desktop\sdr\fm_only_TokyoFM.grc"
>>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJTT_ATIS.grc"
>>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJTT_ATIS_copy.grc"
  
```

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	2400000.0

Null Sinkの設定

Properties: Null Sink

General Advanced Documentation

Input Type float

Vector Length 1 [int]

Num Inputs 1 [int]

Bus Connections [[0,],]

OK Cancel Apply

floatに変更

入力が終わったらOKをクリック

Band pass filterの追加

*untitled - GNU Radio Companion

File Edit View Run Tools Help

fm_only_TokyoFM x am_only_RJT_ATIS x am_only_RJT_ATIS_copy x untitled x

Python
QT GUI

Variable
ID: samp_rate
Value: 2.4M

Variable
ID: freq
Value: 80M

「Band」を検索

枠内にドラッグ
アンドドロップ

OutからInに配線をつなぐ

Block paths:

```
C:\Users\user\radioconda\Library\share\gnuradio\grc\blocks
```

Loading: "C:\Users\user\Desktop\sdr\fm_only_TokyoFM.grc"
>>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJT_ATIS.grc"
>>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJT_ATIS_copy.grc"

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	2400000.0

Band pass filter の設定

The image shows a software dialog box titled "Properties: Band Pass Filter" with three tabs: "General", "Advanced", and "Documentation". The "General" tab is active. The dialog contains several input fields and dropdown menus, each with a yellow box around it and a yellow arrow pointing to a Japanese instruction. The instructions are as follows:

- FIR Type:** Float -> Float (Real Taps)(Decim)に変更 (Change to Float -> Float (Real Taps)(Decim))
- Decimation:** 10を入力 (Enter 10)
- Gain:** 3を入力 (Enter 3)
- Sample Rate:** 480e3を入力 (Enter 480e3)
- Low Cutoff Freq:** 10を入力 (Enter 10)
- High Cutoff Freq:** 5e3を入力 (Enter 5e3)
- Transition Width:** 10e3を入力 (Enter 10e3)
- Window:** Hamming (selected)
- Beta:** 6.76

At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Apply". A yellow box highlights the "OK" button, with a yellow arrow pointing to the instruction: "入力が終わったらOKをクリック" (Click OK when input is finished).

Audio Sinkの追加

GNU Radio Companion interface showing the process of adding an Audio Sink block to a signal flow graph.

The main window title is `*untitled - GNU Radio Companion`. The menu bar includes `File Edit View Run Tools Help`. The toolbar contains various icons for file operations and signal processing.

The search bar at the top right contains the text `Audio`, with a yellow box around it and the text `「Audio」を検索` (Search for "Audio").

The block palette on the right shows the `Audio Sink` block highlighted in blue. Other blocks listed include `Audio Source`, `Alaw Audio Decoder`, `g711 Alaw Audio Encoder`, `CODEC2 Audio Decoder`, `CODEC2 Audio Encoder`, `CVSD Audio Decoder (Raw Bit-L`, `CVSD Audio Encoder (Raw Bit-L`, `g721 Audio Decoder`, `g721 Audio Encoder`, `g723_24 Audio Decoder`, `g723_24 Audio Encoder`, `g723_40 Audio Decoder`, `g723_40 Audio Encoder`, `ulaw Audio Decoder`, and `ulaw Audio Encoder`.

The signal flow graph (SFG) shows the following blocks and connections:

- Filter** (Sample Rate: 2.4M, Cutoff: 10k, Transition Width: 30k) connected to **Delay** (Delay: 1).
- Delay** connected to **Complex Conjugate**.
- Complex Conjugate** connected to **Multiply** (in1).
- Multiply** connected to **Complex To Mag Phase** (in).
- Complex To Mag Phase** has two outputs: **mag** (connected to **Null Sink**) and **phase** (connected to **Band Pass Filter**).
- Band Pass Filter** (Decimation: 10, Gain: 1, Sample Rate: 480k, Low Cutoff Freq: 10, High Cutoff Freq: 10k, Transition Width: 10k, Window: Hamming, Beta: 6.76) connected to **Audio Sink** (Sample Rate: 48 kHz).

Yellow arrows indicate the search process and the drag-and-drop action of the `Audio Sink` block into the SFG.

Text annotations in Japanese:

- `枠内にドラッグアンドドロップ` (Drag and drop into the frame)
- `OutからInに配線をつなぐ` (Connect wiring from Out to In)

The bottom panel shows the `Block paths:` and `Block variables:` sections.

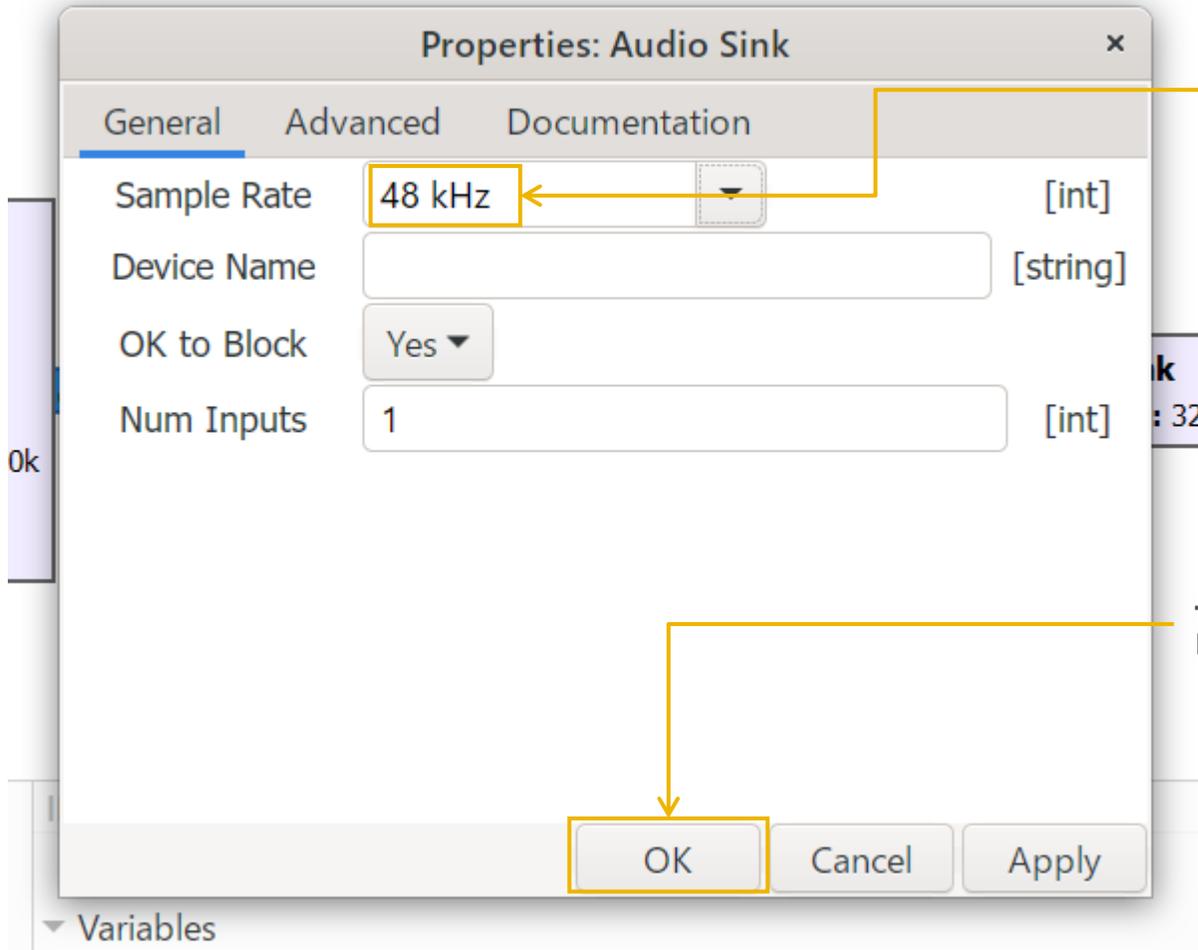
`Block paths:`

```
C:\Users\user\radiosconda\Library\share\gnuradio\grc\blocks
```

`Block variables:`

ID	Value
Imports	
Variables	
freq	8000000.0
samp_rate	2400000.0

Audio Sinkの設定



プルダウンメニューから「48kHz」を選択

設定したらOKをクリック

グラフの追加

GNU Radio Companion interface showing a signal processing flow graph and a component palette.

qtを入力 (Input qt)

枠内にドラッグアンドドロップ (Drag and drop into the frame)

The flow graph includes the following blocks:

- Source: File Source (2.4M, 30k, 100k)
- Filter: Filter (2.4M, 30k, 100k)
- QT GUI Time Sink (Number of Points: 1.024k, Sample Rate: 2.4M, Autoscale: No)
- QT GUI Frequency Sink (FFT Size: 1024, Center Frequency (Hz): 0, Bandwidth (Hz): 2.4M)
- Delay (Delay: 1)
- Complex Conjugate
- Multiply
- Complex To Mag Phase (mag, phase)
- Null Sink
- Band Pass Filter (Decimation: 10, Gain: 1, Sample Rate: 480k, Low Cutoff Freq: 10, High Cutoff Freq: 10k, Transition Width: 10k, Window: Hamming, Beta: 6.76)
- Audio Sink (Sample Rate: 48 kHz)
- QT GUI Time Sink (Number of Points: 1.024k, Sample Rate: 2.4M, Autoscale: No)

The component palette on the right shows the following categories and items:

- Core
- Instrumentation
 - QT
 - fosphor sink (Qt)
 - QT GUI Bercurve Sink
 - QT GUI Constellation Sink
 - QT GUI Eye Sink
 - QT GUI Frequency Sink
 - QT GUI Histogram Sink
 - QT GUI Number Sink
 - QT GUI Sink
 - QT GUI Time Raster Sink
 - QT GUI Time Sink
 - QT GUI Vector Sink
 - QT GUI Waterfall Sink
- UI Widgets
 - QT
 - QT GUI App Background
 - QT GUI Fast Auto-Correlator S
 - QT GUI Az-El Plot
 - QT GUI Check Box
 - QT GUI Chooser
 - QT GUI Compass
 - QT GUI Dial
 - QT GUI Dial Gauge
 - QT GUI Distance Radar
 - QT GUI Message Edit Box
 - QT GUI Entry

Block paths:

```
C:\Users\user\radioconda\Library\share\gnuradio\grc\blocks
```

Loading: "C:\Users\user\Desktop\sdr\fm_only_TokyoFM.grc" >>> Done

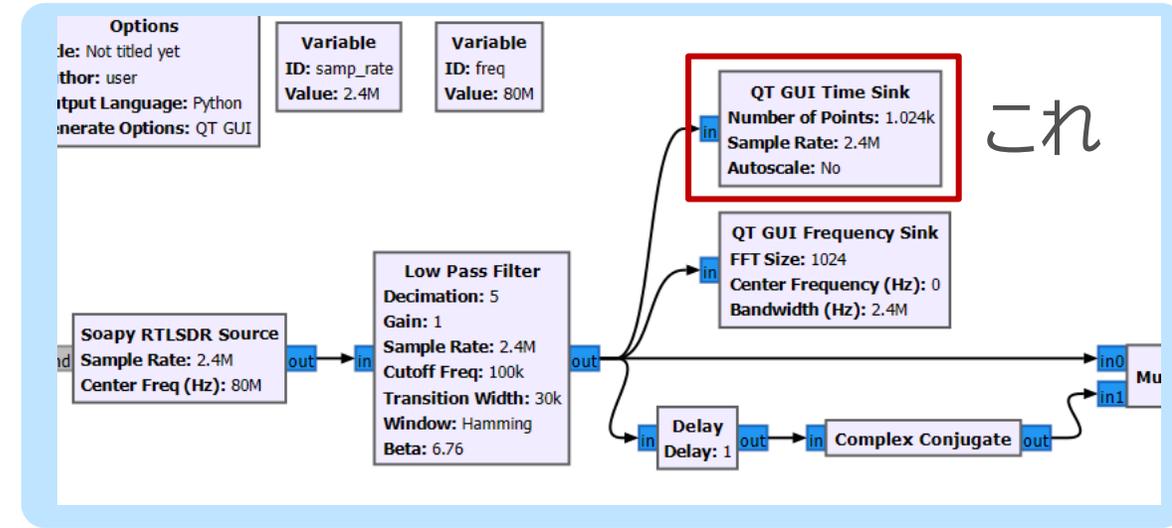
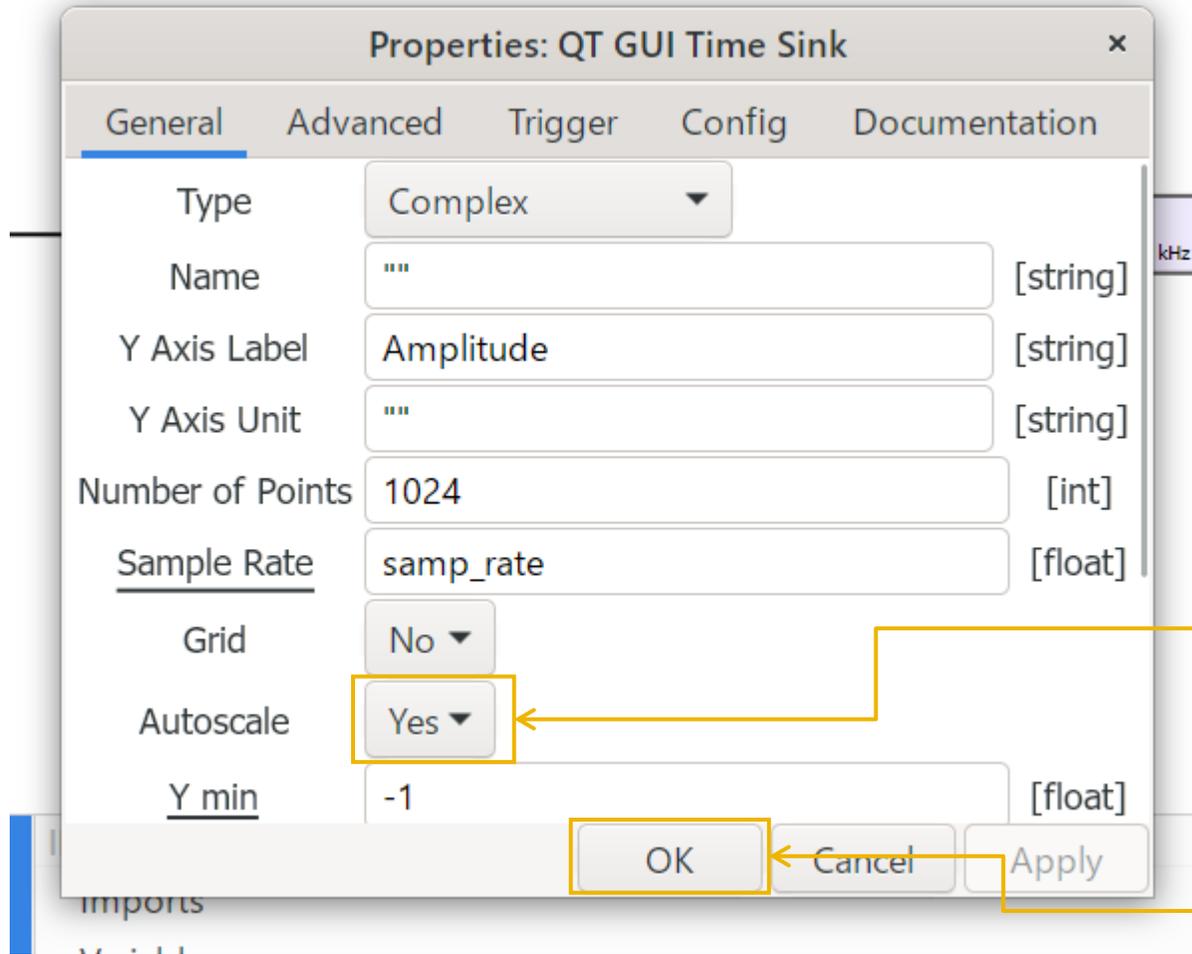
Loading: "C:\Users\user\Desktop\sdr\am_only_RJTT_ATIS.grc" >>> Done

Loading: "C:\Users\user\Desktop\sdr\am_only_RJTT_ATIS_copy.grc" >>> Done

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	2400000.0

1つ目のQT GUI Time sinkの設定

QT GUI Time sinkブロックをダブルクリックして設定を開く

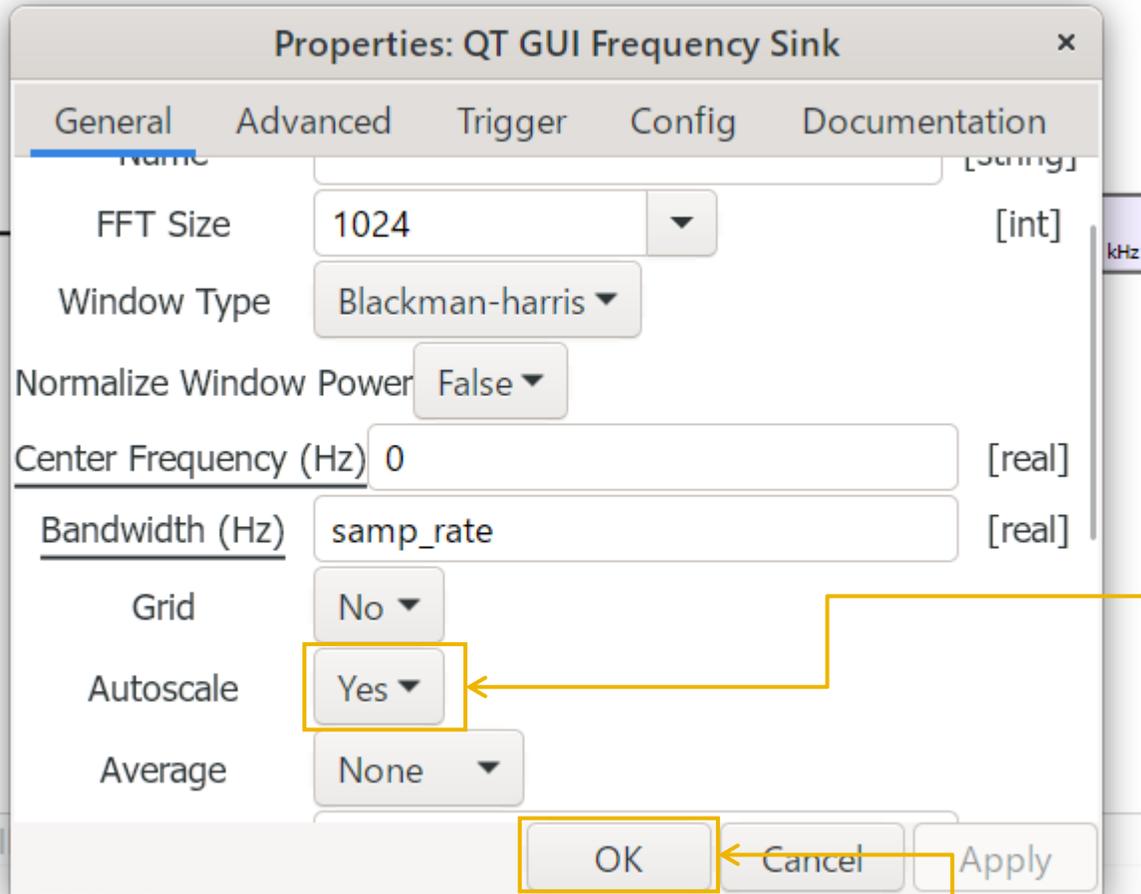


AutoscaleをYesに変更する

設定したらOKをクリック

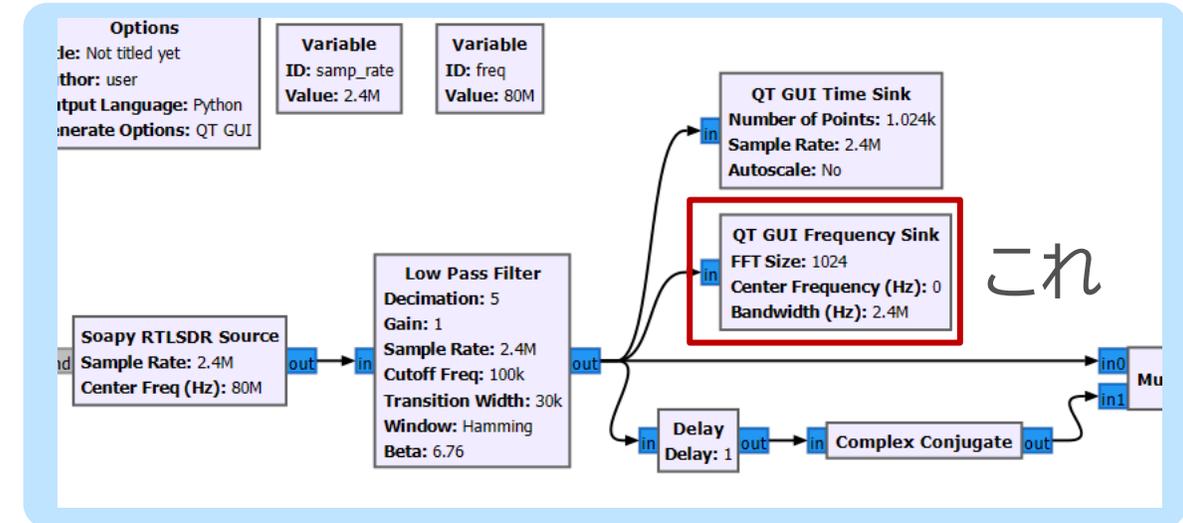
QT GUI Frequency sinkの設定

QT GUI Frequency sinkブロックをダブルクリックして設定を開く



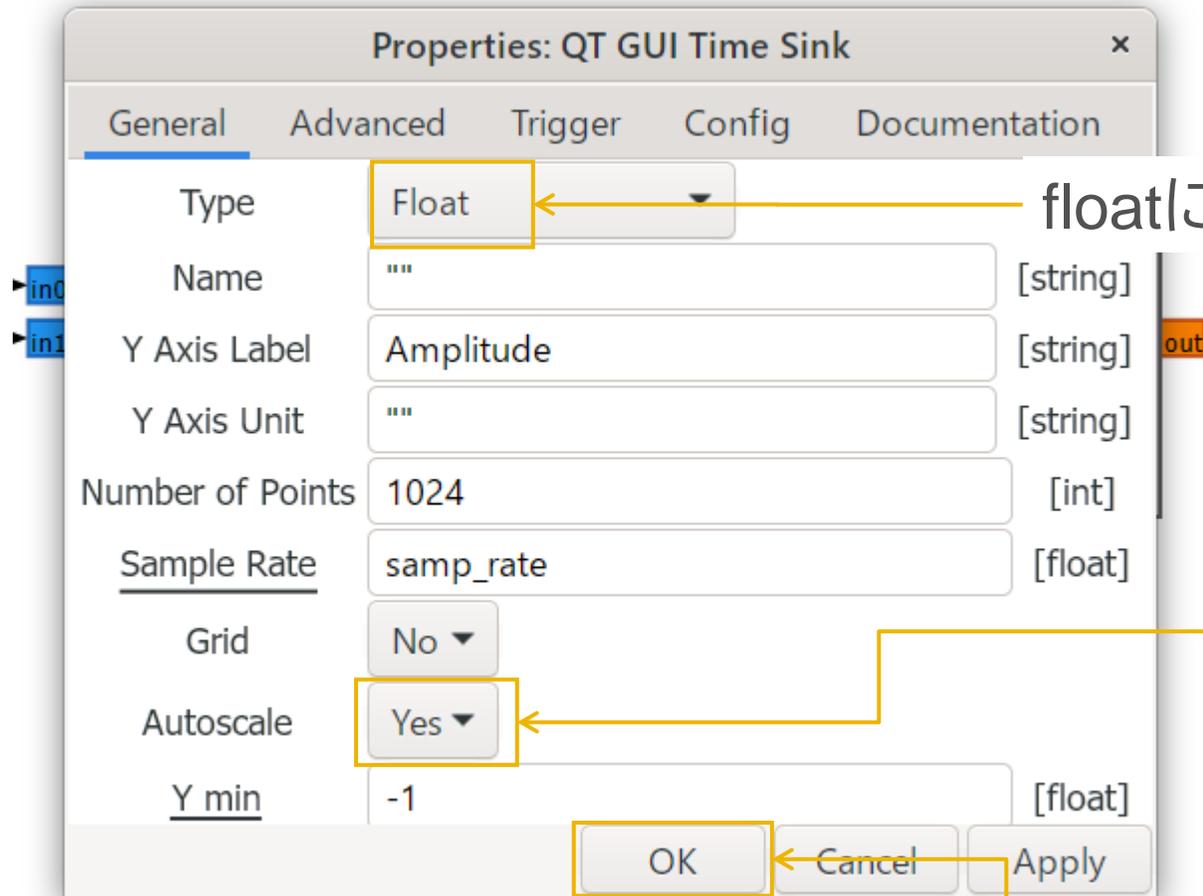
AutoscaleをYesに変更する

設定したらOKをクリック



2つ目のQT GUI Time sinkの設定

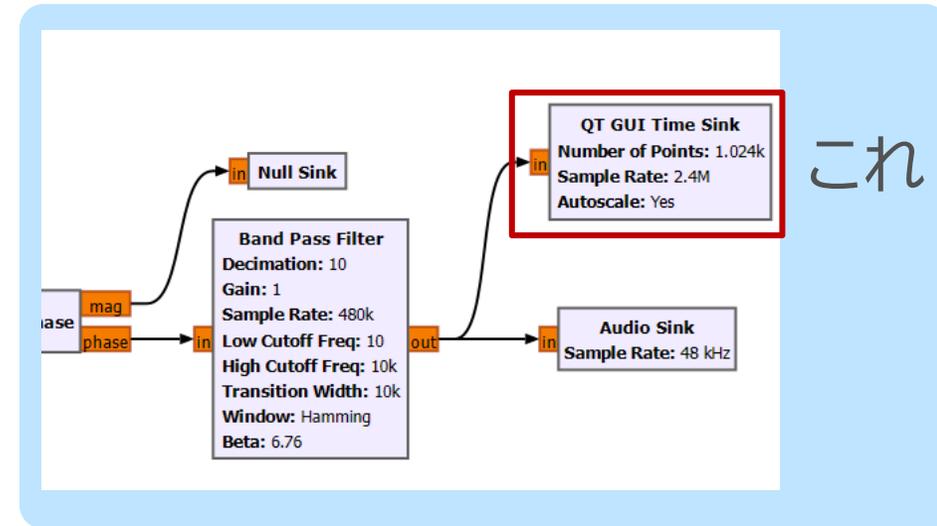
QT GUI Time sinkブロックをダブルクリックして設定を開く



floatに変更する

AutoscaleをYesに変更する

設定したらOKをクリック



ラジオの起動

my_fm_only_TokyoFM.grc - C:\Users\user\Desktop\sdr

File Edit View Run Tools Help

fm_only_TokyoFM x am_only_RJTT_ATIS x am_only_RJTT_ATIS_copy x my_fm_only_TokyoFM

左クリックするとラジオが起動する

Options
Title: Not titled yet
Author: user
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 2.4M

Variable
ID: freq
Value: 80M

Soapy RTLSDR Source
Sample Rate: 2.4M
Center Freq (Hz): 80M

Low Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 2.4M
Cutoff Freq: 100k
Transition Width: 30k
Window: Hamming
Beta: 6.76

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2.4M
Autoscale: Yes

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 2.4M

Delay
Delay: 1

Complex Conjugate

Multiply

Complex To Mag Phase
mag
phase

Band Pass Filter
Decimation: 10
Gain: 1
Sample Rate: 480k
Low Cutoff Freq: 10
High Cutoff Freq: 10k
Transition Width: 10k
Window: Hamming
Beta: 6.76

Null Sink

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2.4M
Autoscale: Yes

Audio Sink
Sample Rate: 48 kHz

mentation

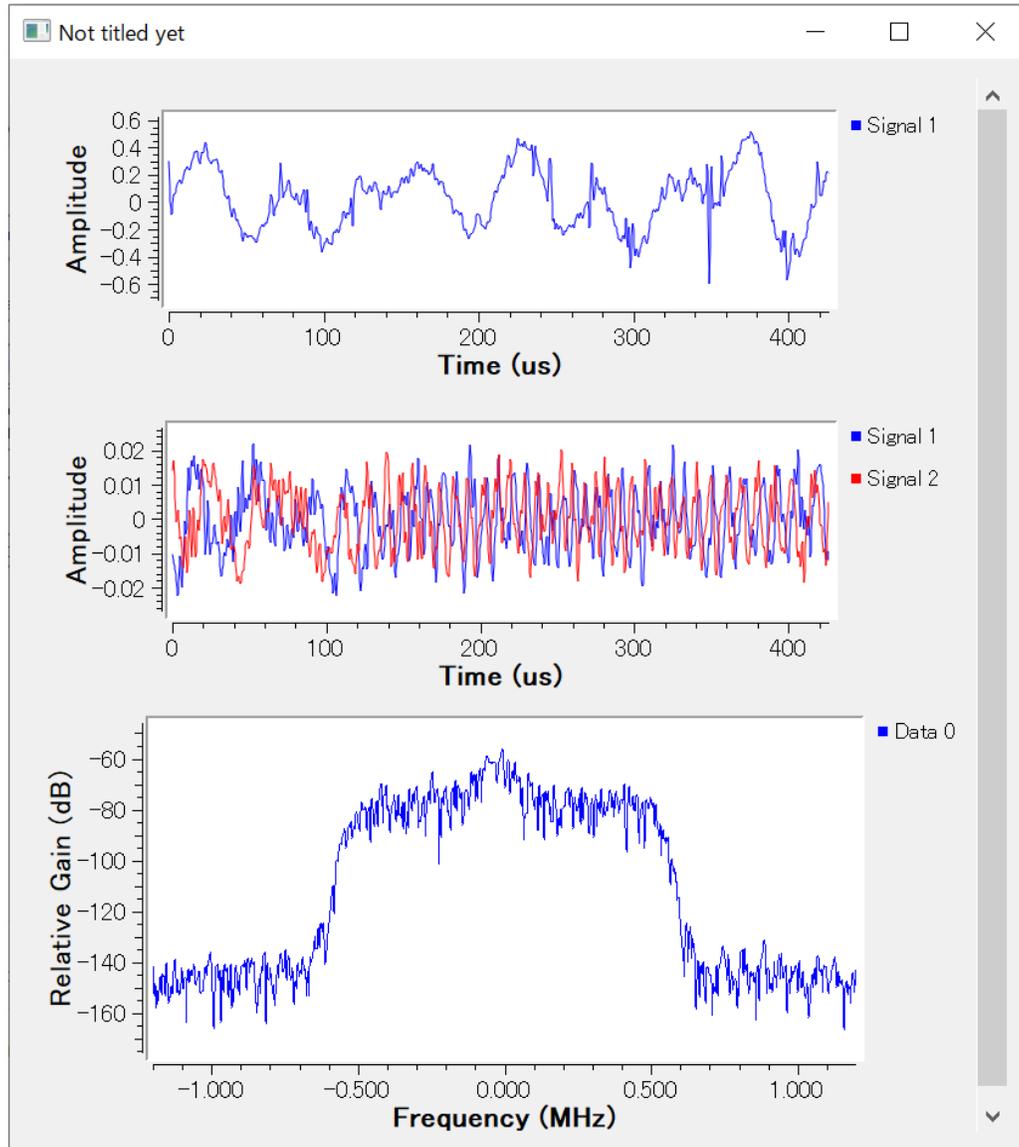
- QT
 - fosphor sink (Qt)
 - QT GUI Bercurve Sink
 - QT GUI Constellation Sink
 - QT GUI Eye Sink
 - QT GUI Frequency Sink
 - QT GUI Histogram Sink
 - QT GUI Number Sink
 - QT GUI Sink
 - QT GUI Time Raster Sink
 - QT GUI Time Sink**
 - QT GUI Vector Sink
 - QT GUI Waterfall Sink
- GUI Widgets
 - QT
 - QT GUI App Background
 - QT GUI Fast Auto-Correlator S
 - QT GUI Az-El Plot
 - QT GUI Check Box
 - QT GUI Chooser
 - QT GUI Compass
 - QT GUI Dial
 - QT GUI Dial Gauge
 - QT GUI Distance Radar
 - QT GUI Message Edit Box
 - QT GUI Entry

my_fm_only_TokyoFM.py

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	2400000.0

[1m][33m[WARNING] SoapyVOLKConverters: no VOLK config file found. Run volk_profile for best performance.[0m
Found Fitipower FC0013 tuner
[INFO] Opening Generic RTL2832U :: 77771111153705700...
Found Fitipower FC0013 tuner
[INFO] Using format CF32.

こんな画面が見られるとOK



TokyoFMが再生されるはず