ソフトウェア無線機

第3回目

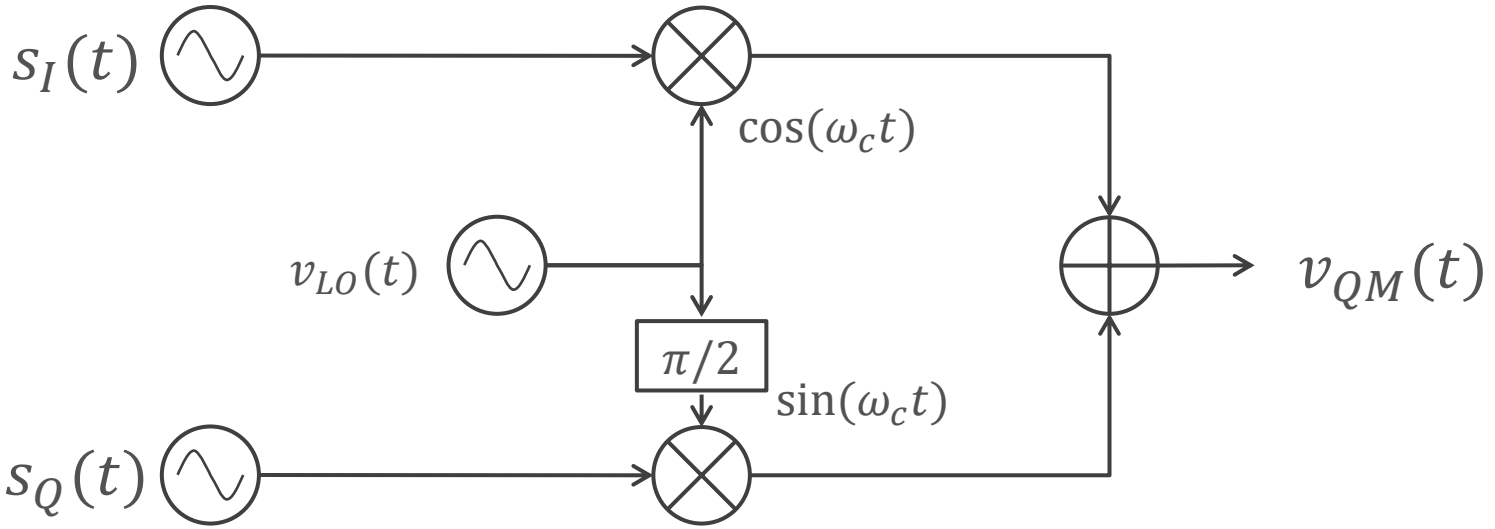
電子磁気応用資料

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

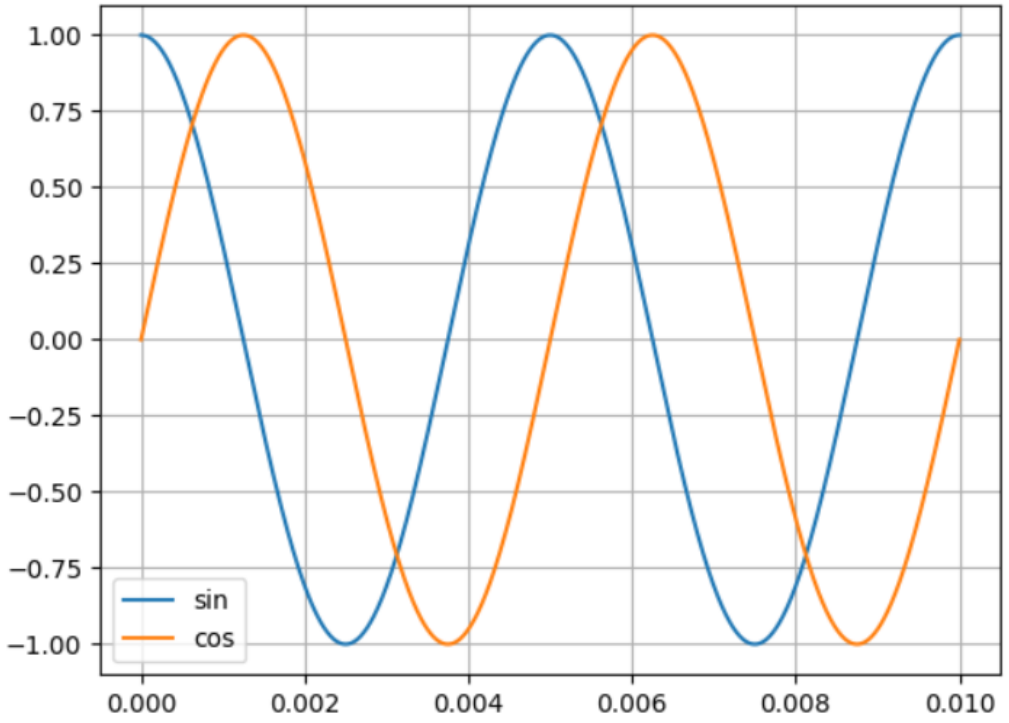
直交変調器による変調

直交変調器（IQ変調器）

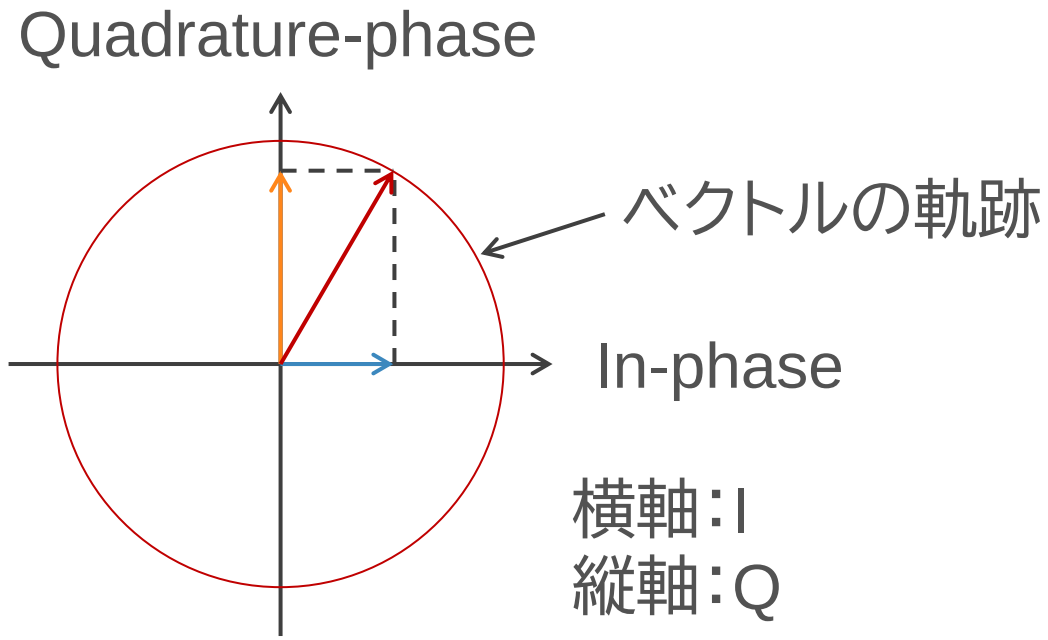


$$v_{QM}(t) = s_I(t)\cos(\omega_c t) + s_Q(t)\sin(\omega_c t)$$

sinとcosとIとQの関係

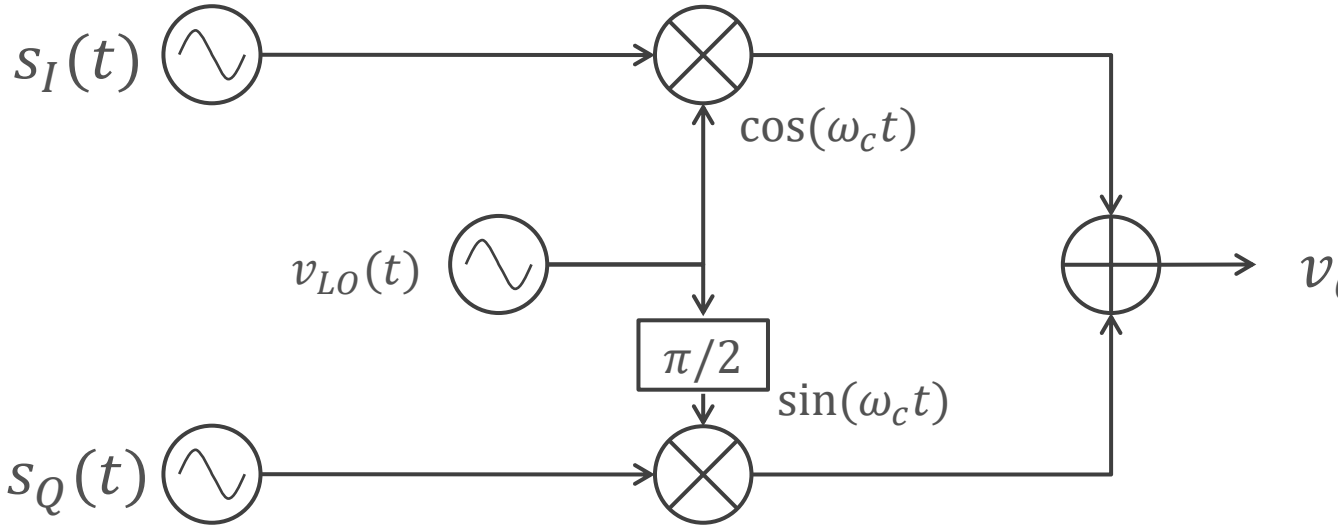


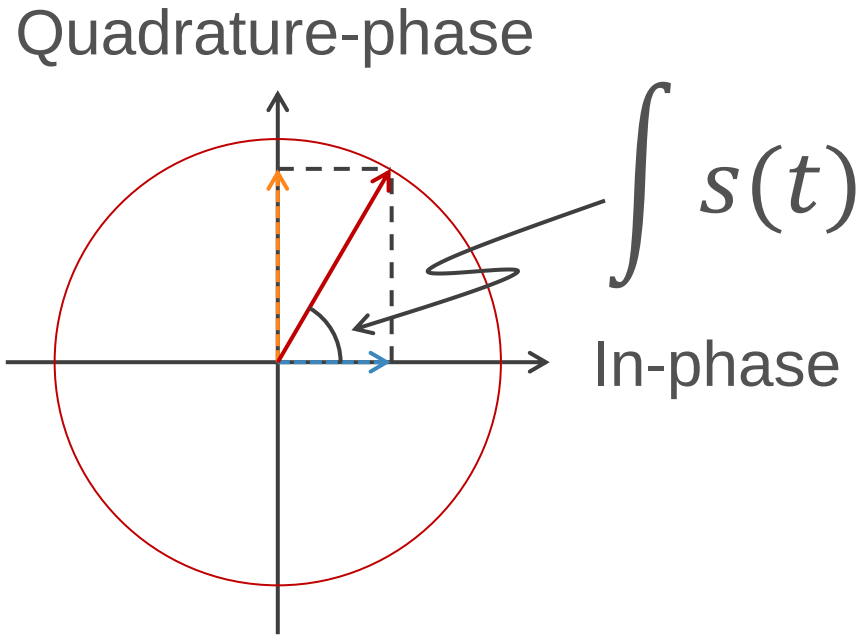
sinとcosは直交関係にある（内積がゼロ）



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

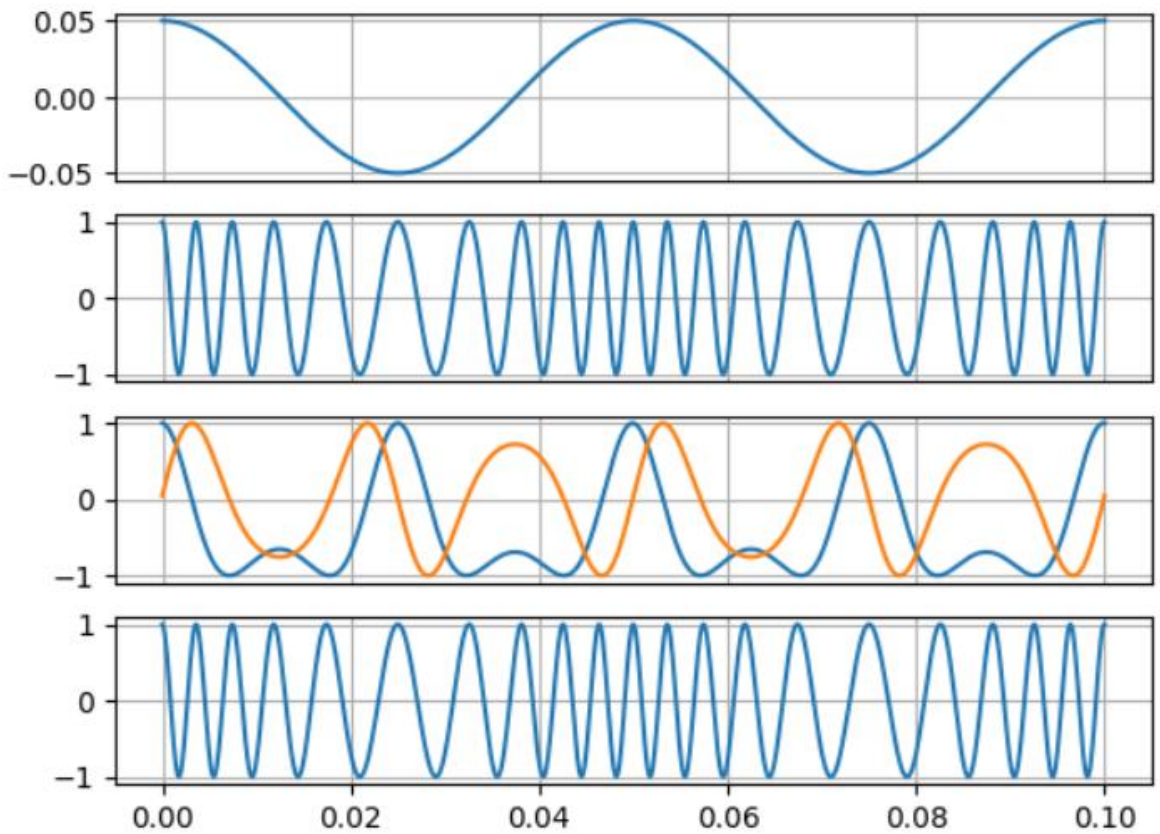
直交変調器によるFMのシミュレーションコード

```
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)
```

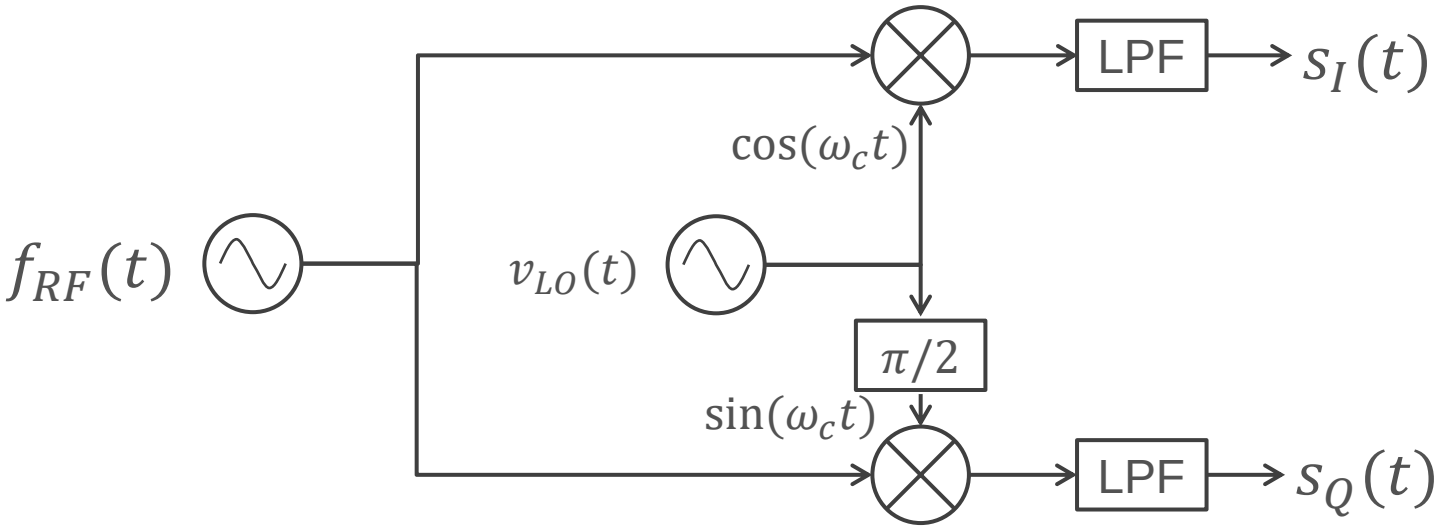
Colabを用いたシミュレーション

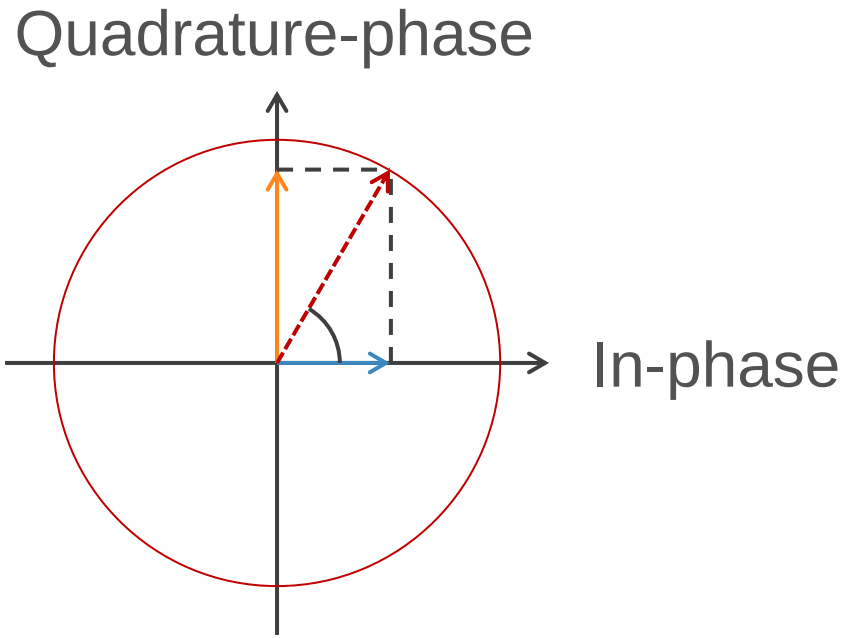
```
import numpy as np
import matplotlib.pyplot as plt

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)





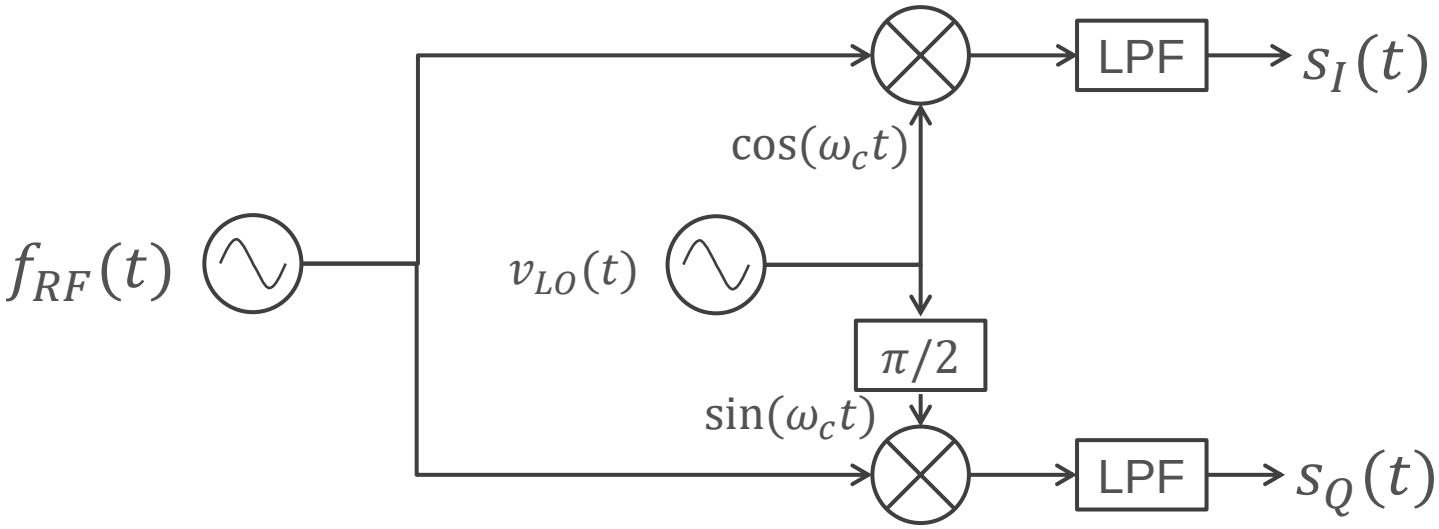
直交復調器を用いるとLOと
- **同相の信号成分(I)**
- **直交する信号成分(Q)**

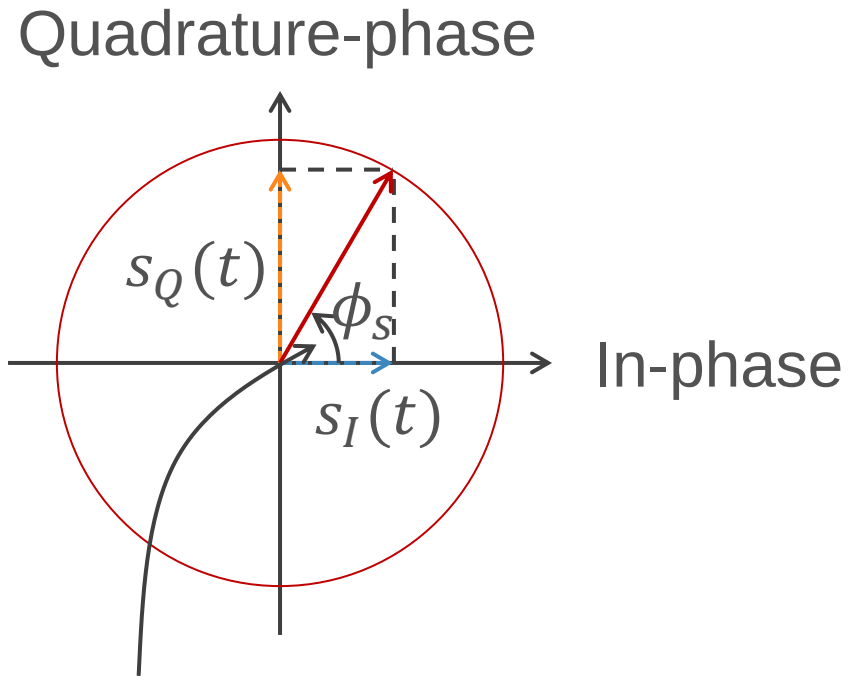
の2つを取り出すことができる。

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))
```

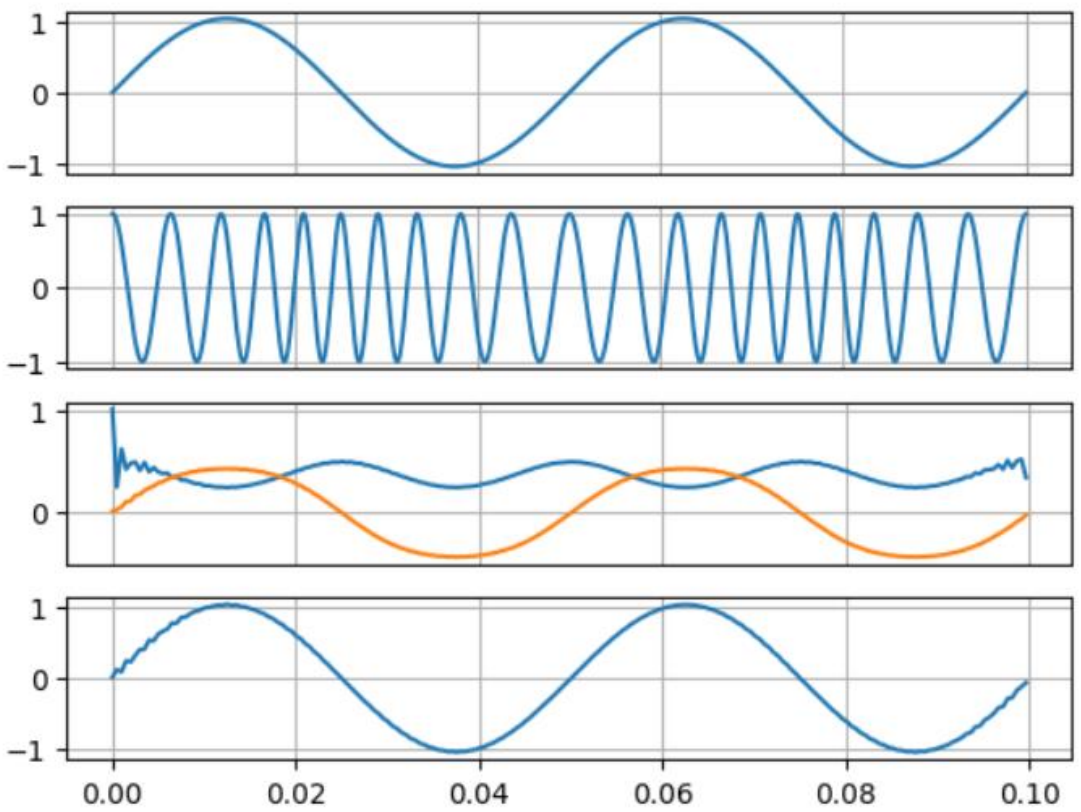
角度を求める
（アークタンジェント）

複素数として扱う

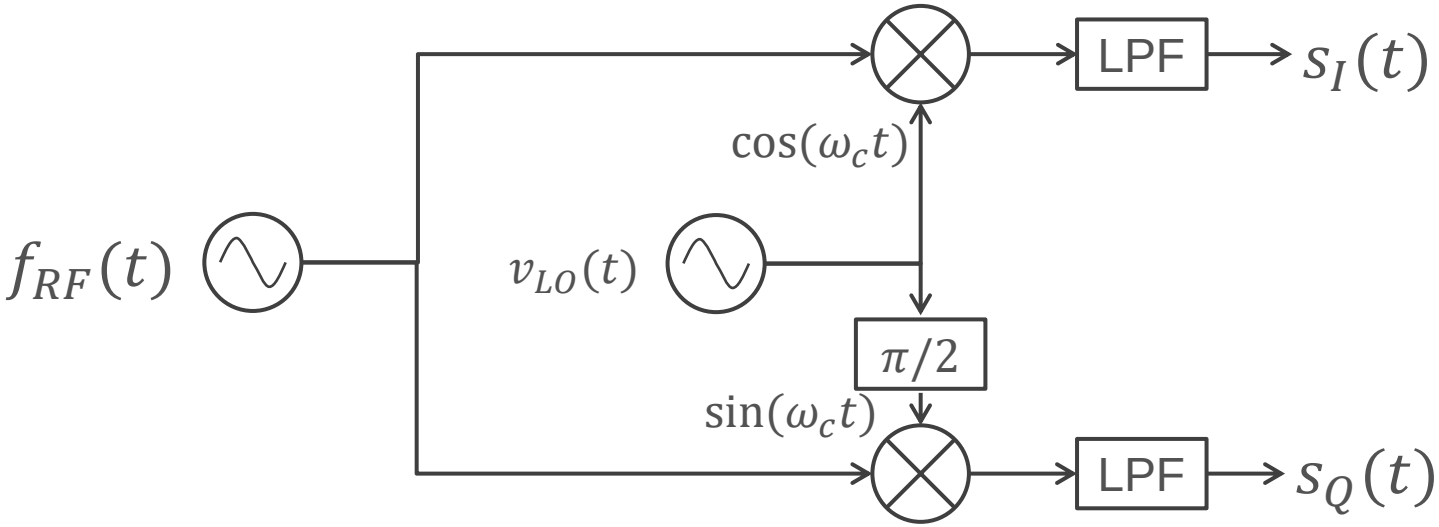
```
fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vqm)
az = fig.add_subplot(413)
az.grid()
az.plot(t[::5], v_i_dec)
az.plot(t[::5], v_q_dec)
aa = fig.add_subplot(414)
aa.plot(t[::5], v_s_demod)
aa.grid()
axes = fig.get_axes()
for 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)
```

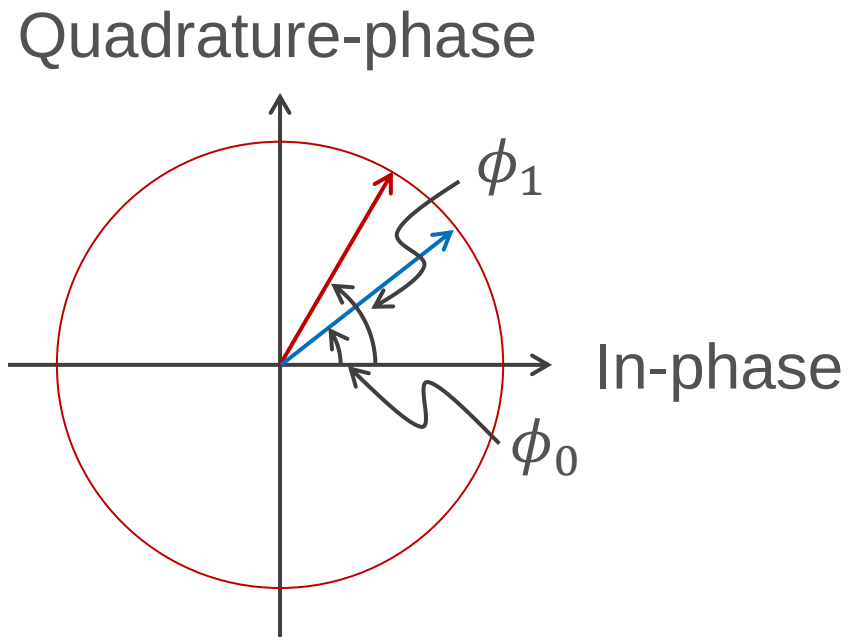
Colabを用いたシミュレーション

```
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.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。

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

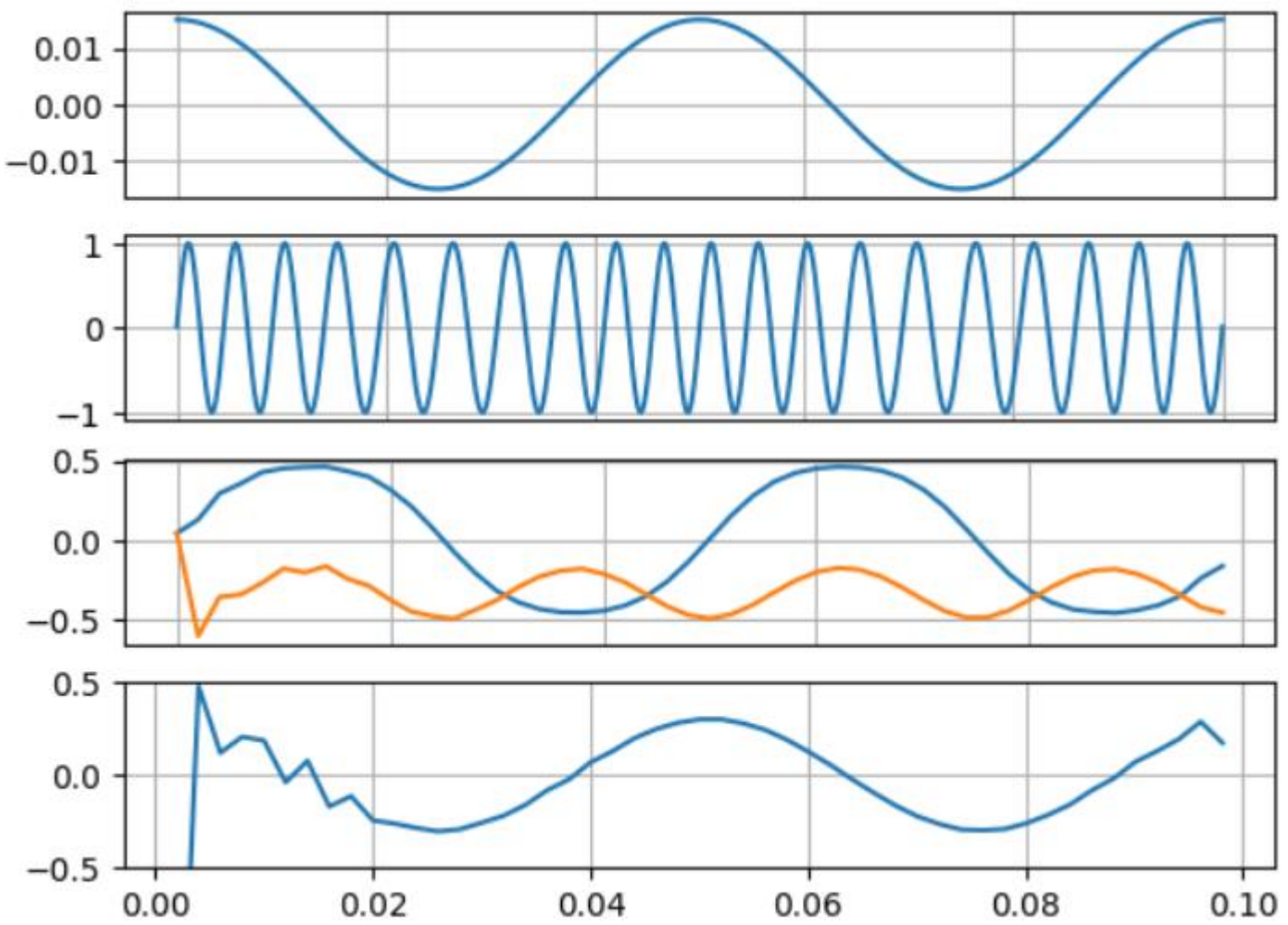
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

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)
```

角度を求める処理と差分を求める処理

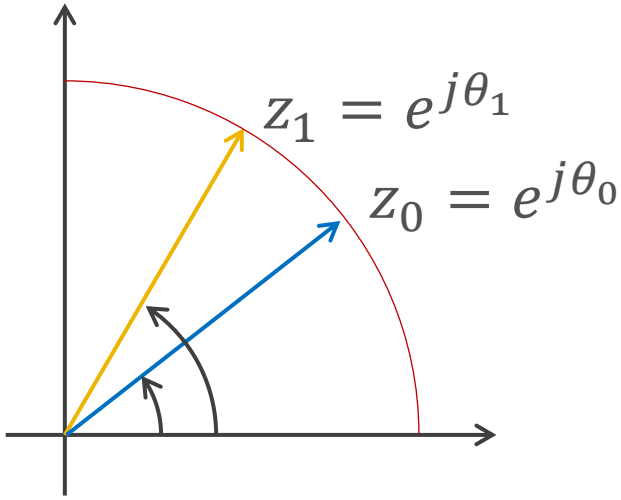
```
fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vfm)
az = fig.add_subplot(413)
az.grid()
az.plot(t[::num_decm], v_i_dec)
az.plot(t[::num_decm], v_q_dec)
aa = fig.add_subplot(414)
aa.plot(t[::num_decm][1:], v_s_demod_diff)
aa.set_ylim((-0.5,0.5))
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)
```

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.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)
```



ちょっとスマートなFM復調



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
v_s_demod_diff = v_s_demod[1:]*np.conj(v_s_demod[:-1])
v_s_demod_phase = np.angle(v_s_demod_diff)
```

← 複素数に変換する処理

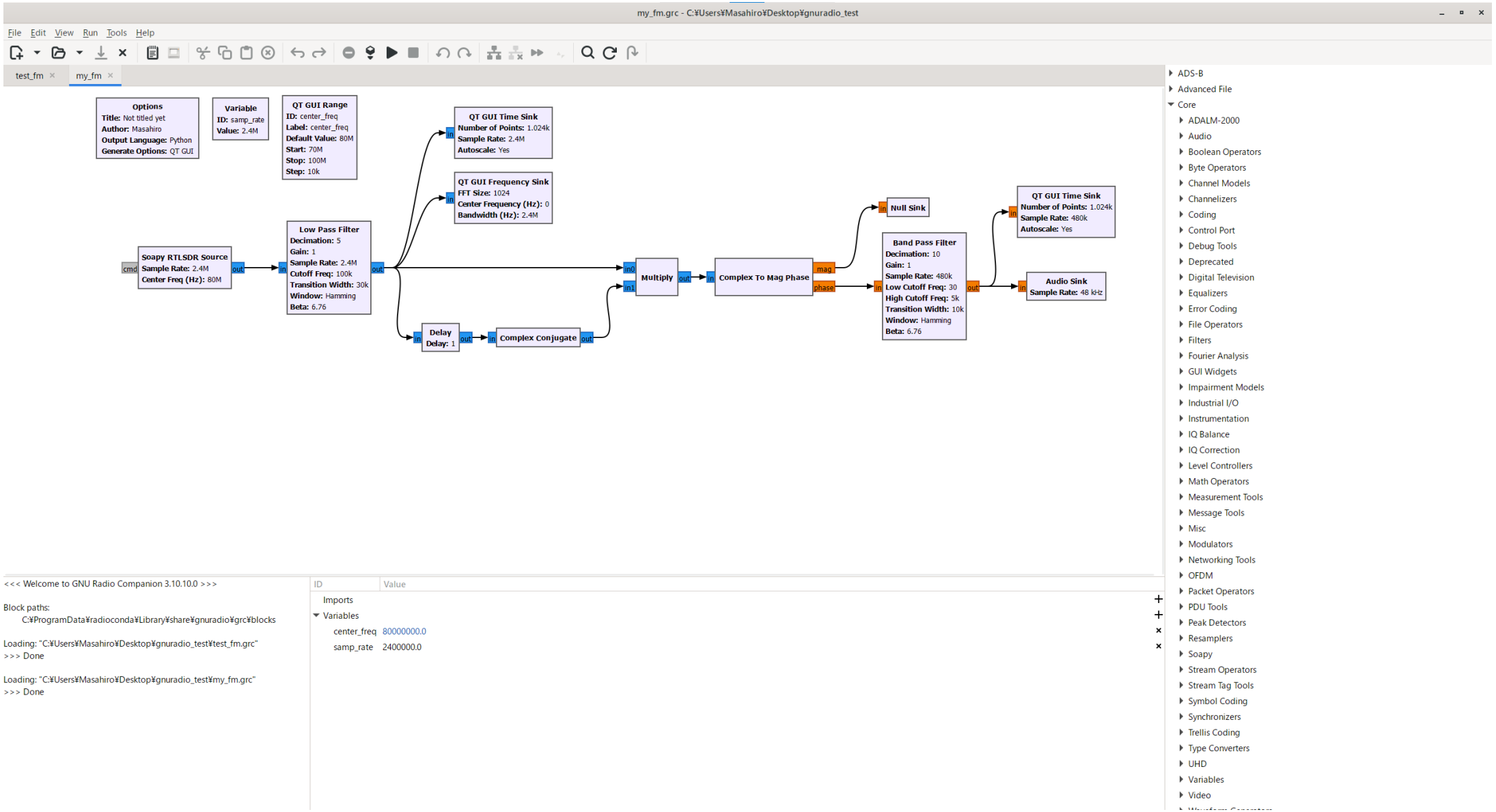
← 1つ前の値の複素共役を取って積を取る処理

← 角度に変換する処理

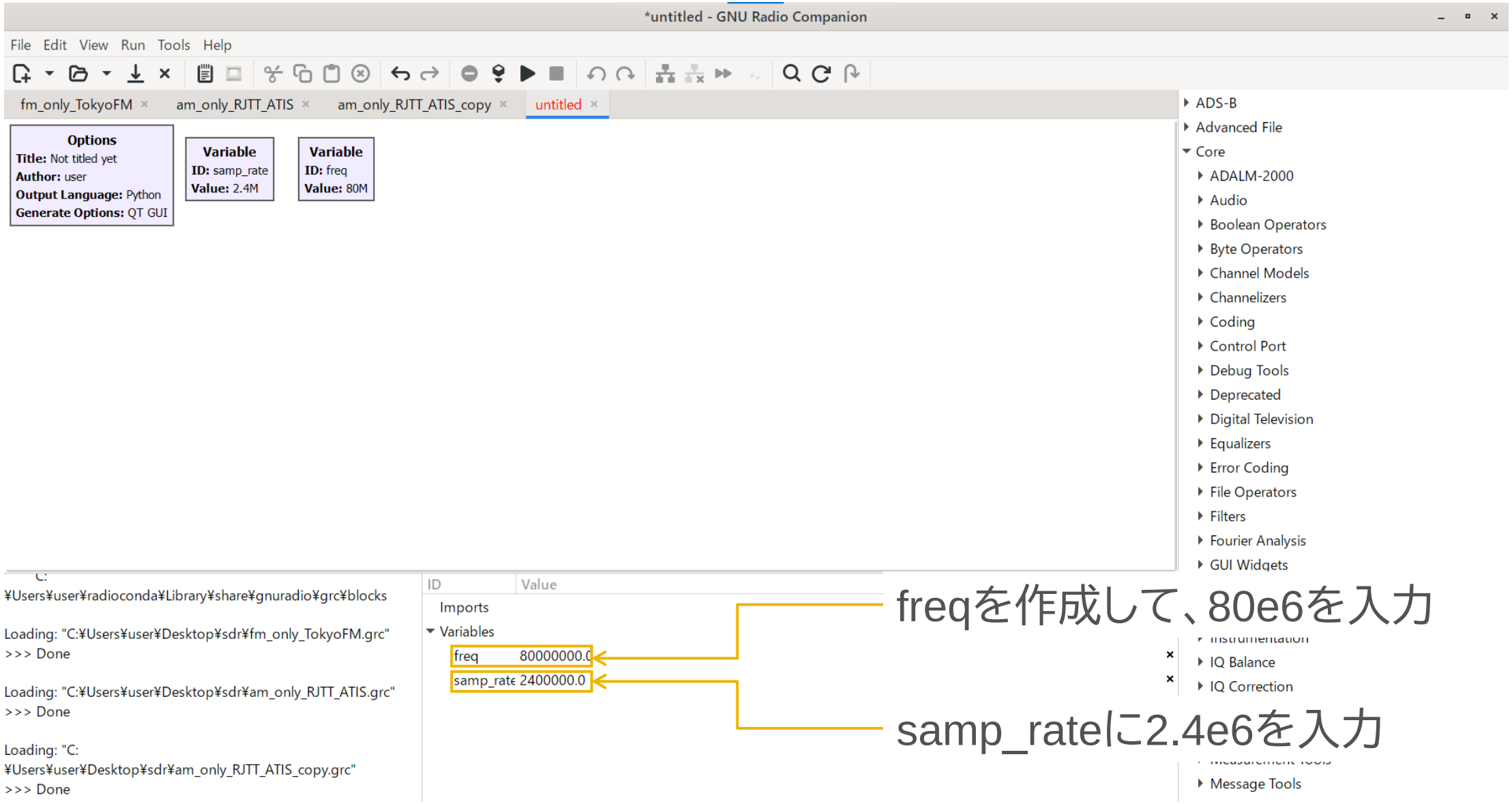
```
fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vfm)
az = fig.add_subplot(413)
az.grid()
az.plot(t[::20], v_i_dec)
az.plot(t[::20], v_q_dec)
aa = fig.add_subplot(414)
aa.plot(t[::20][1:], v_s_demod_phase)
aa.set_ylim((-0.5,0.5))
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



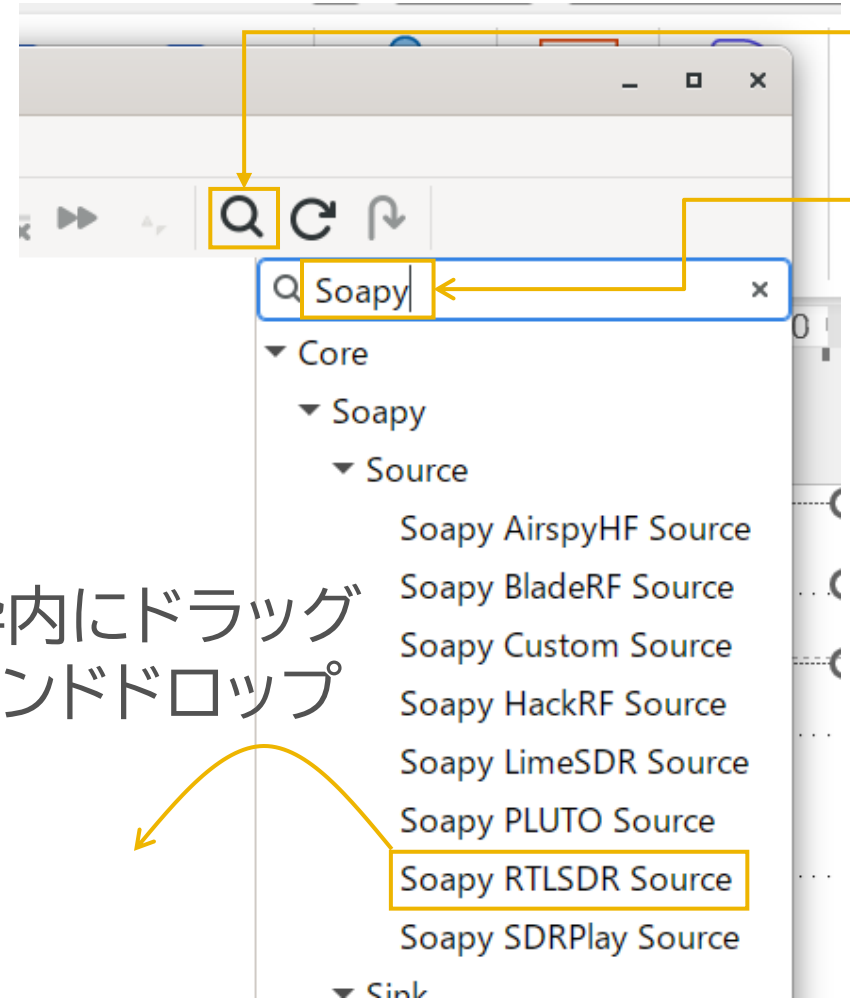
変数の追加と設定


*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
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
IQ Balance
IQ Correction
Message Tools
¥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
freqを作成して、80e6を入力
samp_rateに2.4e6を入力

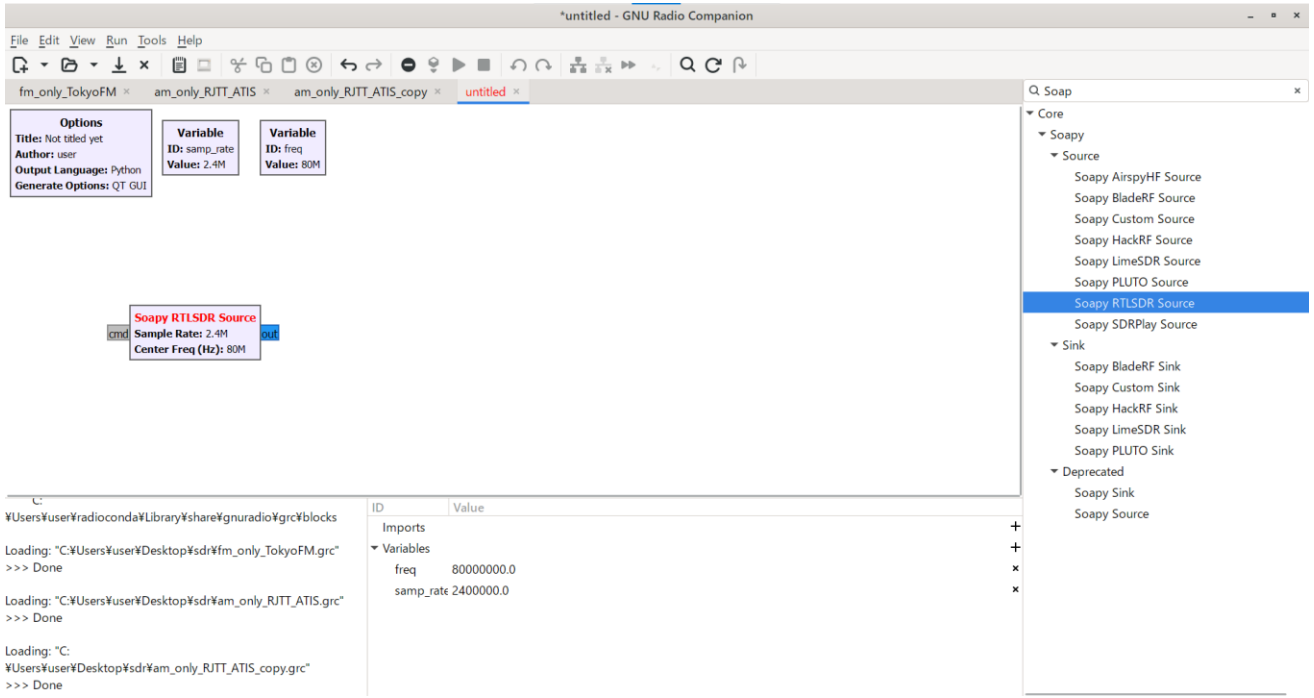
RTL SDRを追加

虫眼鏡(検索)を左クリック

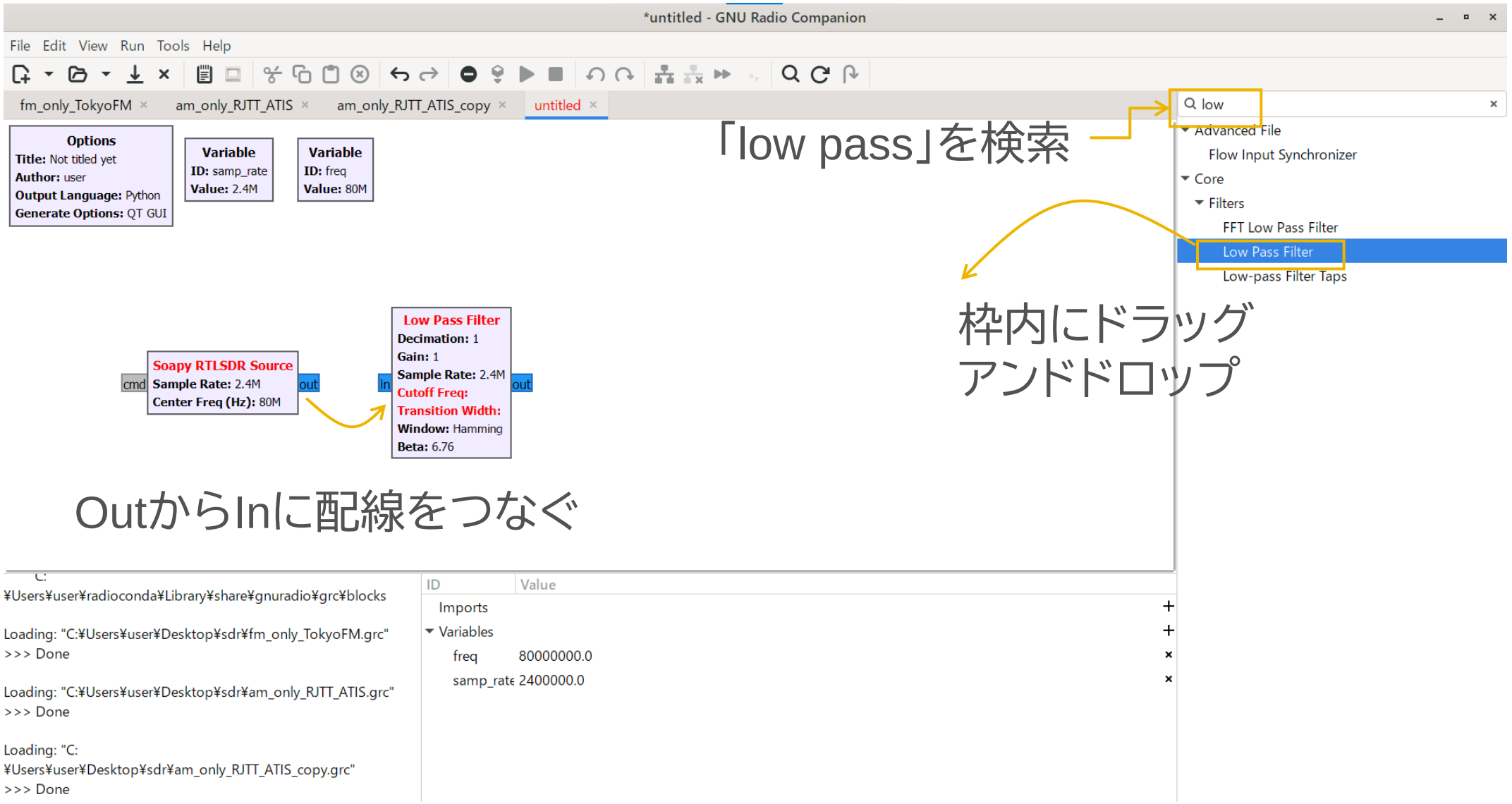
Soapyを入力



枠内にドラッグ
アンドドロップ

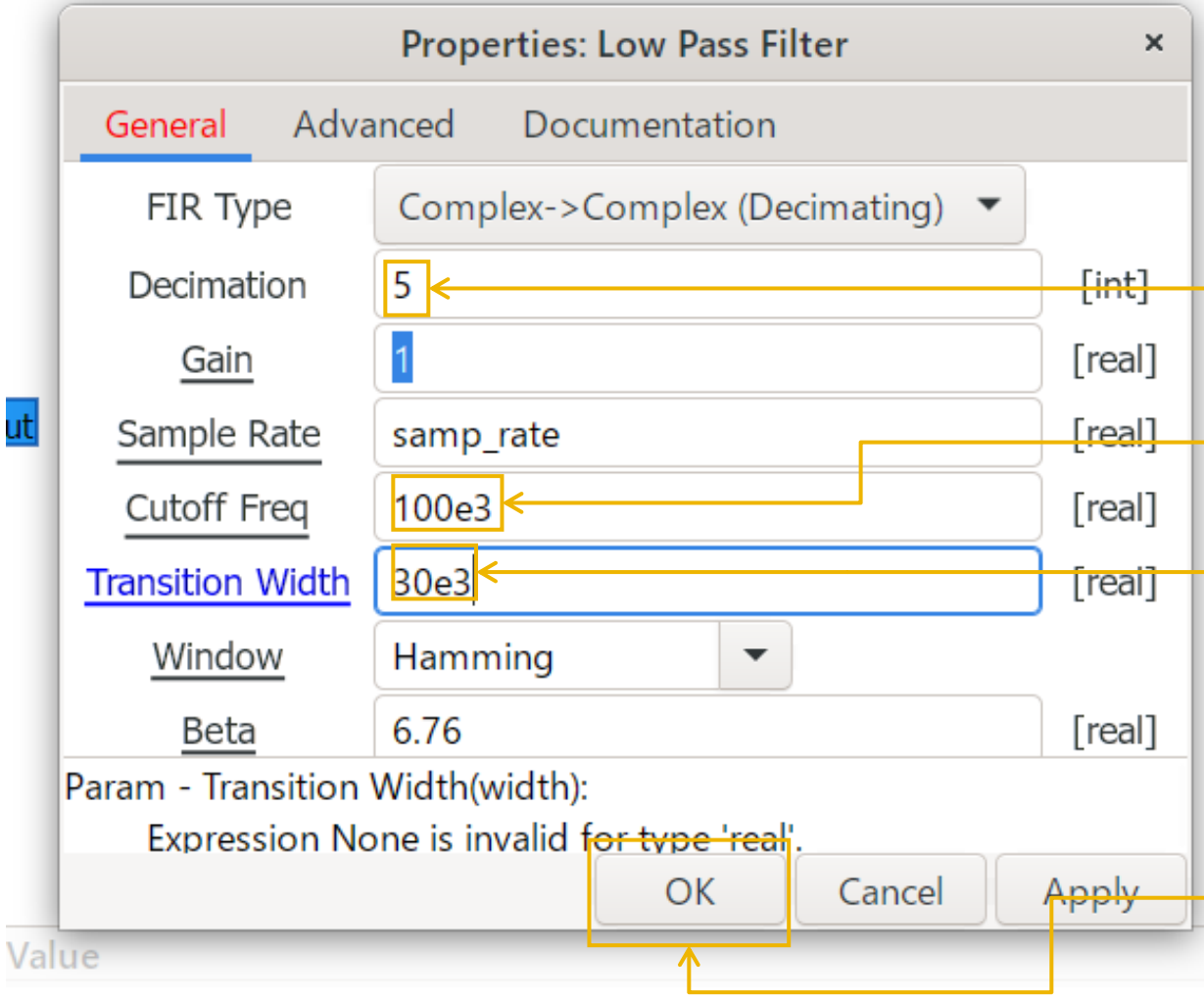


Low pass filterの追加



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



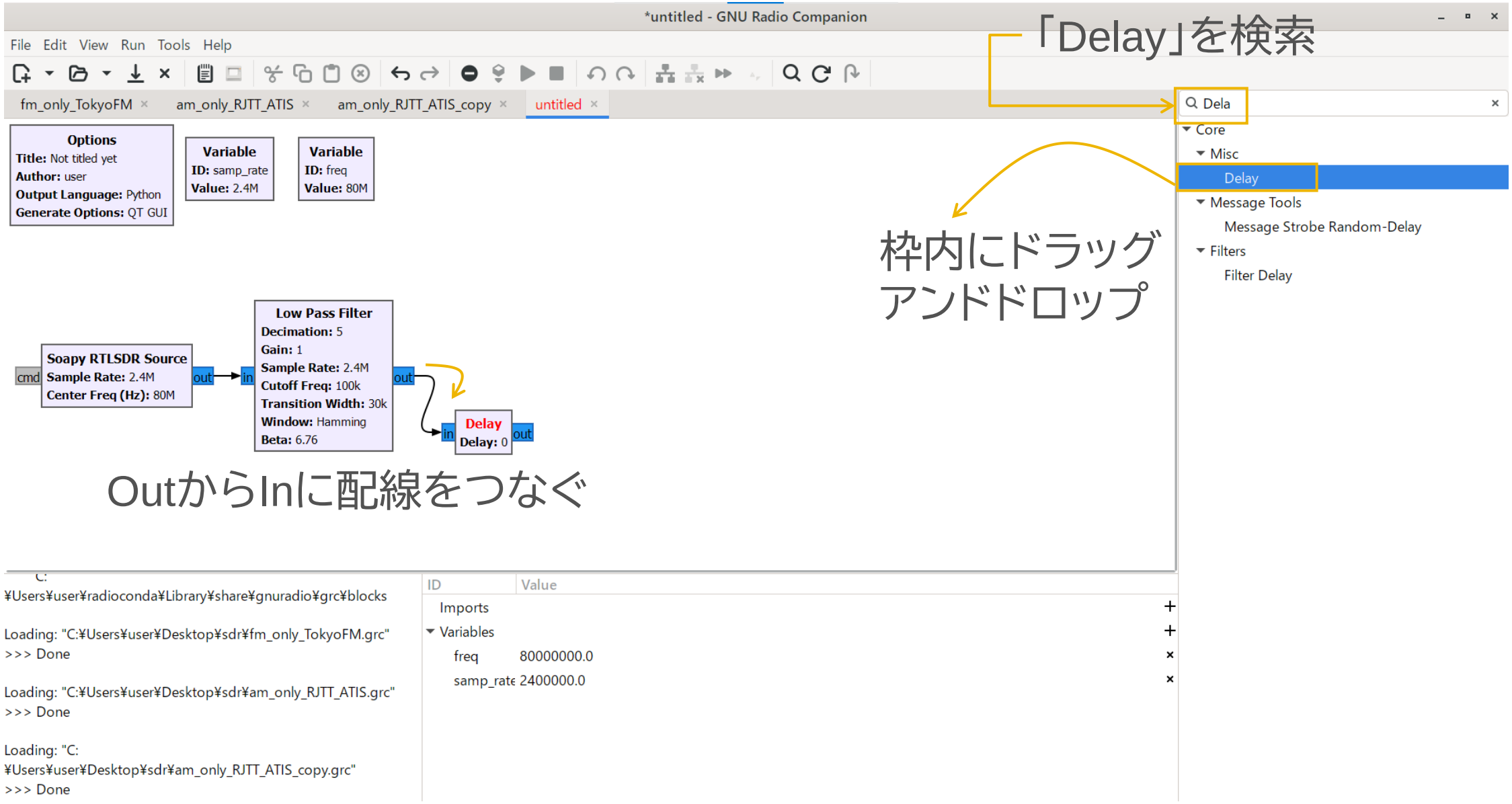
5を入力

100e3を入力

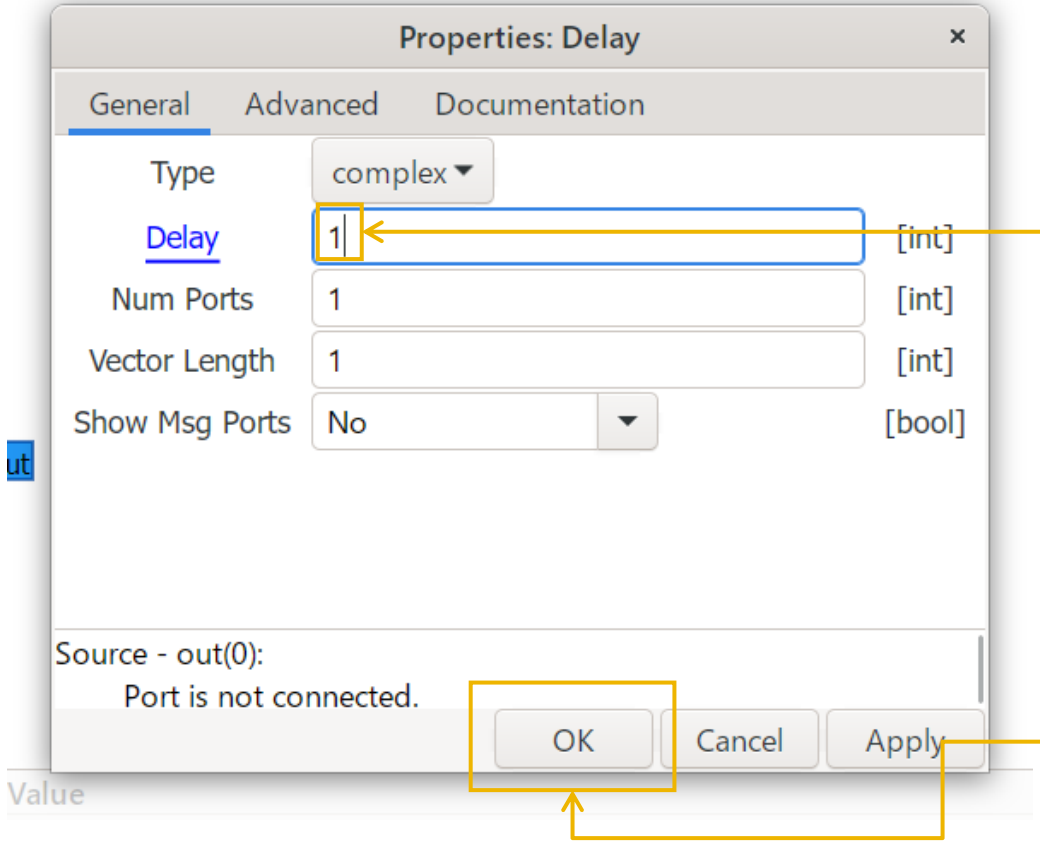
30e3を入力

入力が終わったらOKをクリック

Delay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
「Delay」を検索
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
Soapy RTLSDR Source
cmd
Sample Rate: 2.4M
Center Freq (Hz): 80M
out
in
Low Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 2.4M
Cutoff Freq: 100k
Transition Width: 30k
Window: Hamming
Beta: 6.76
out
in
Delay
Delay: 0
out
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

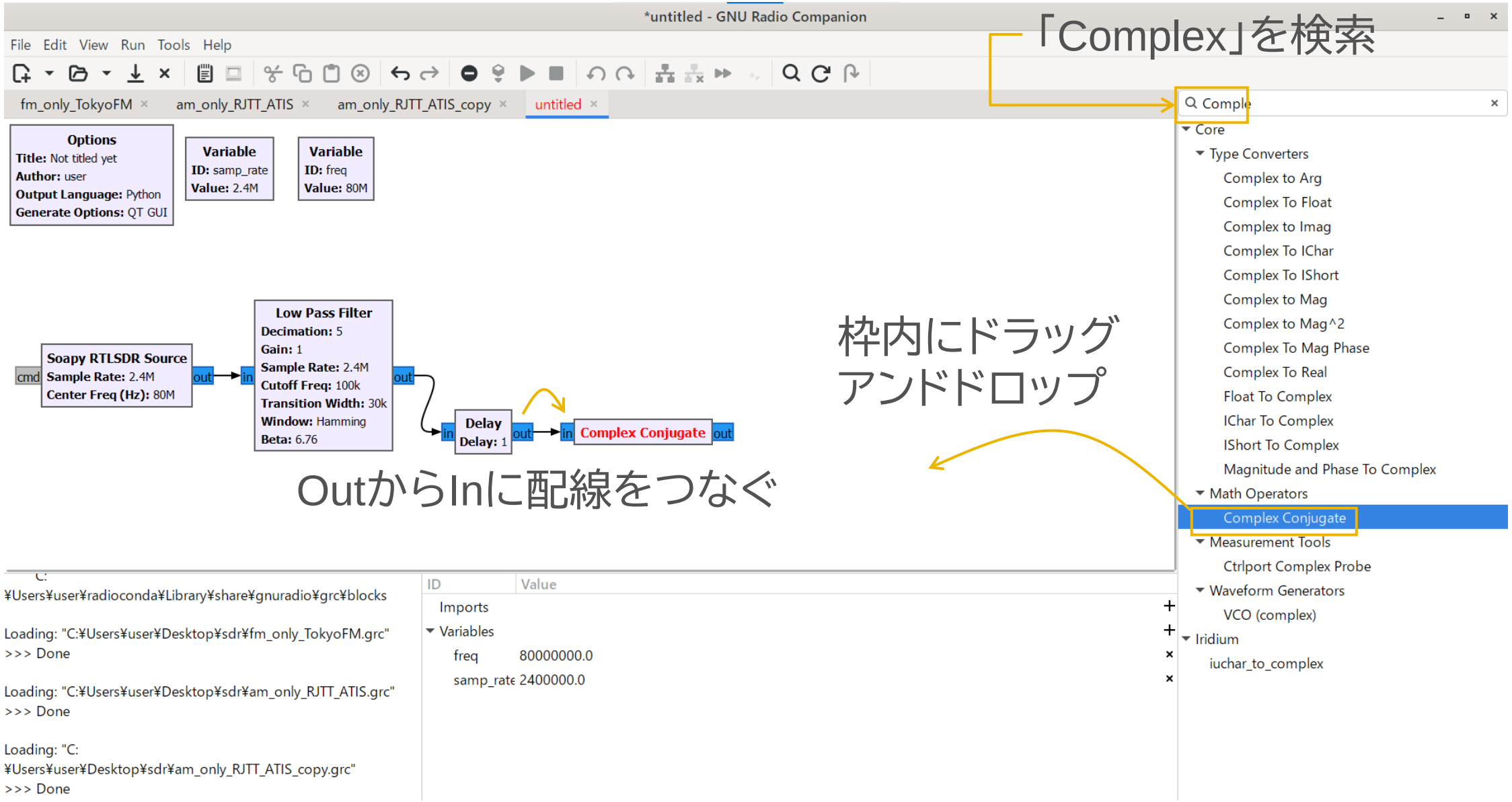
Delayの設定



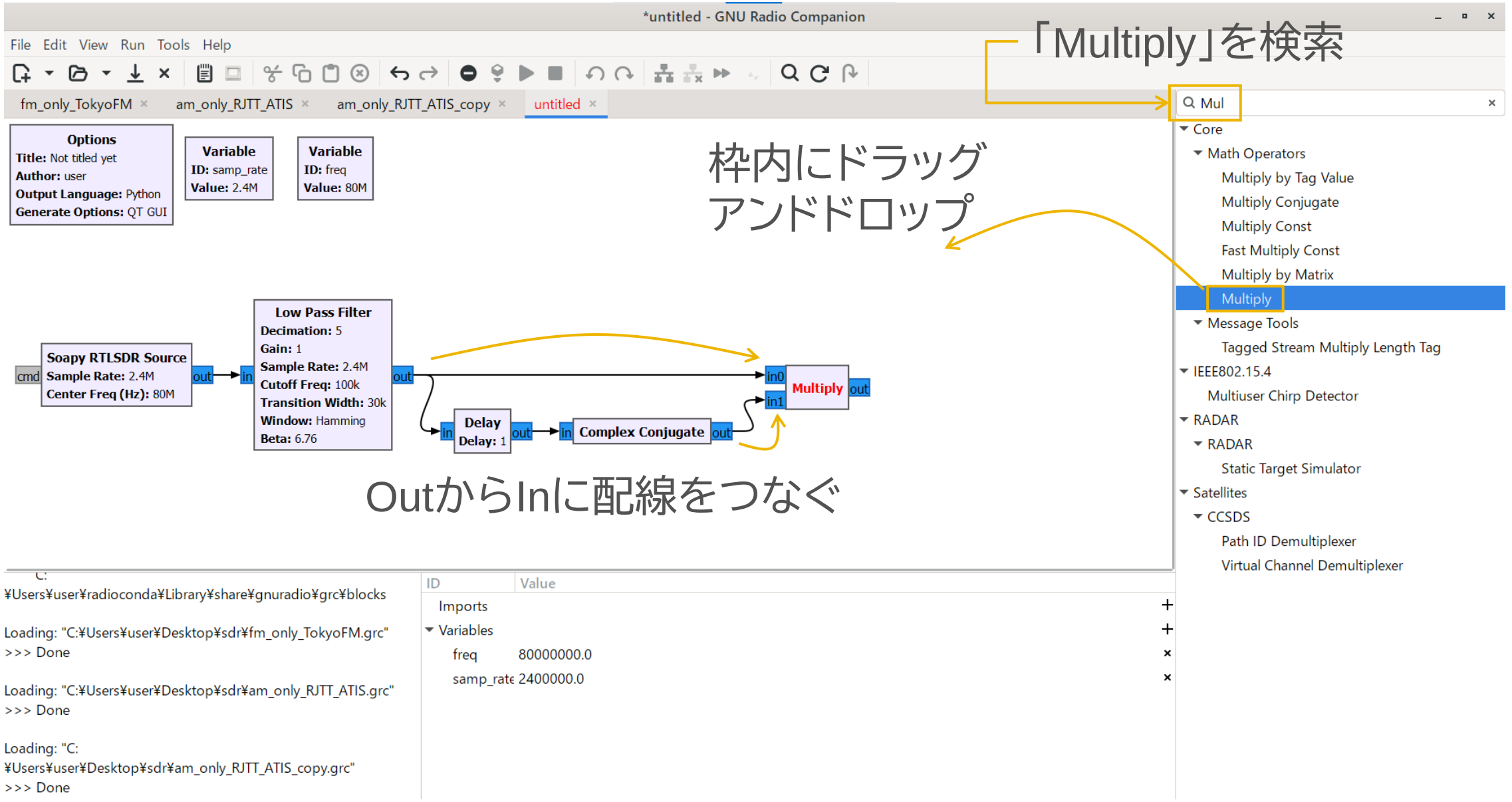
1を入力

入力が終わったらOKをクリック

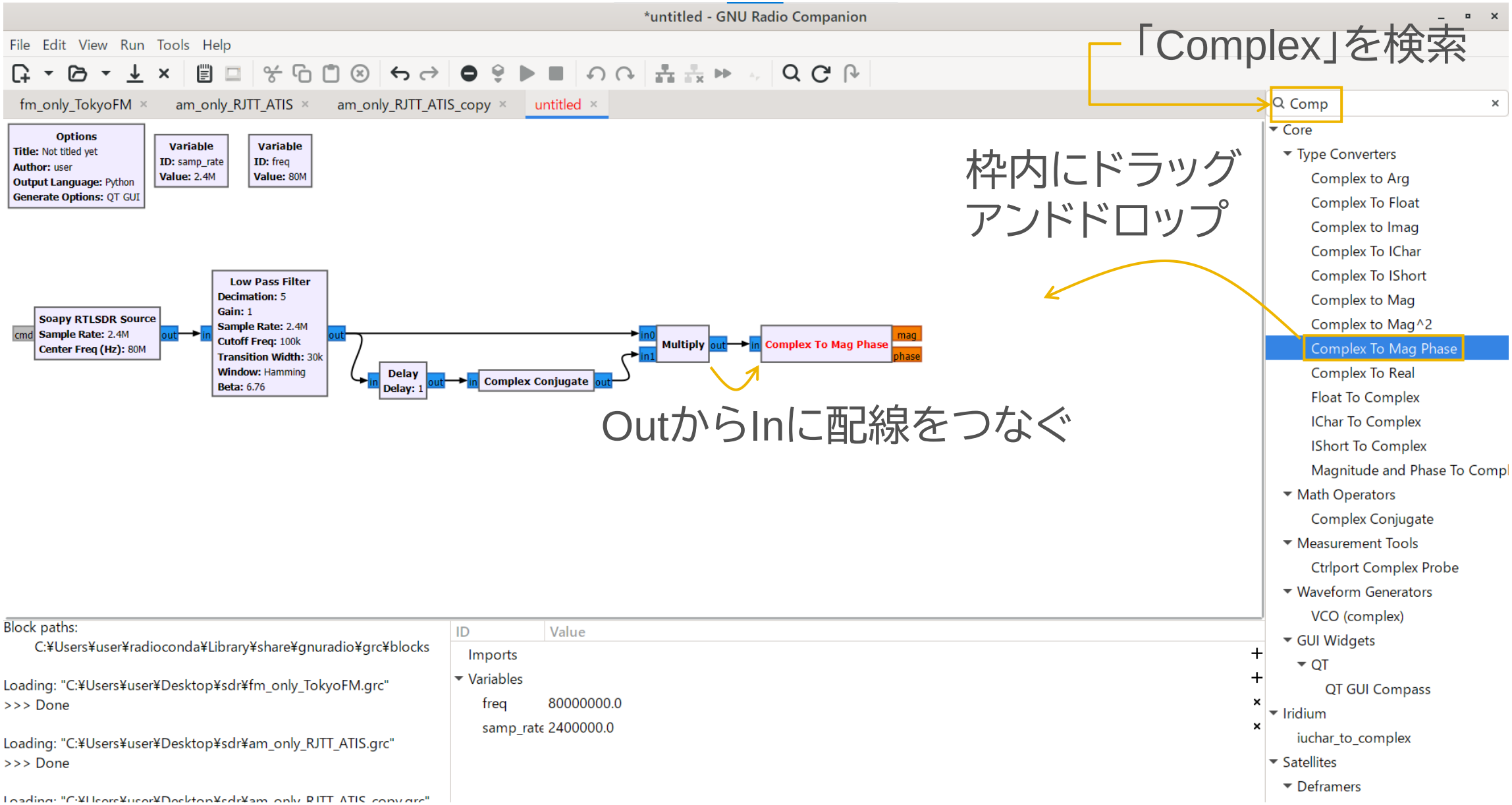
Complex conjugateの追加



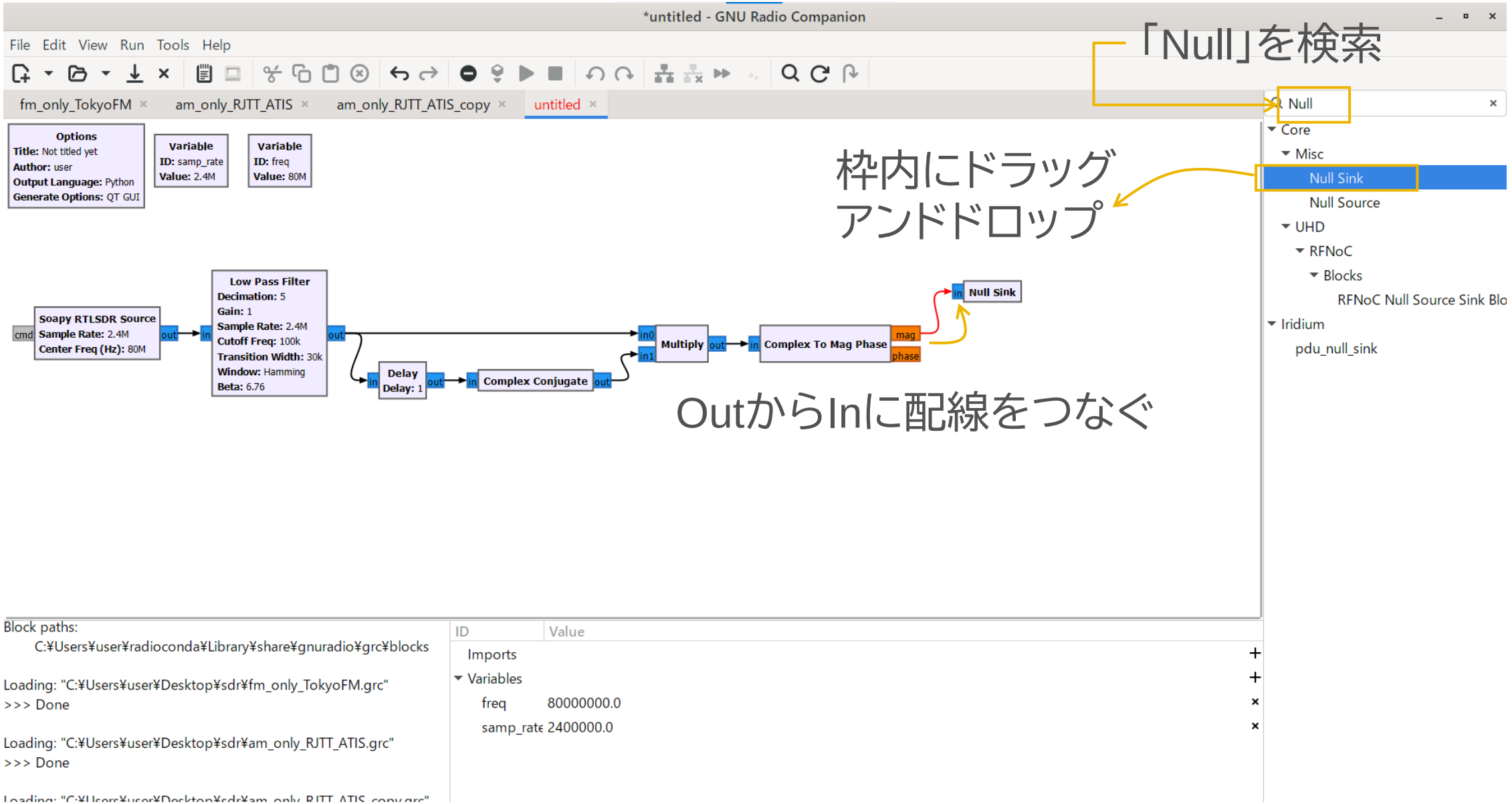
Multiplyを追加



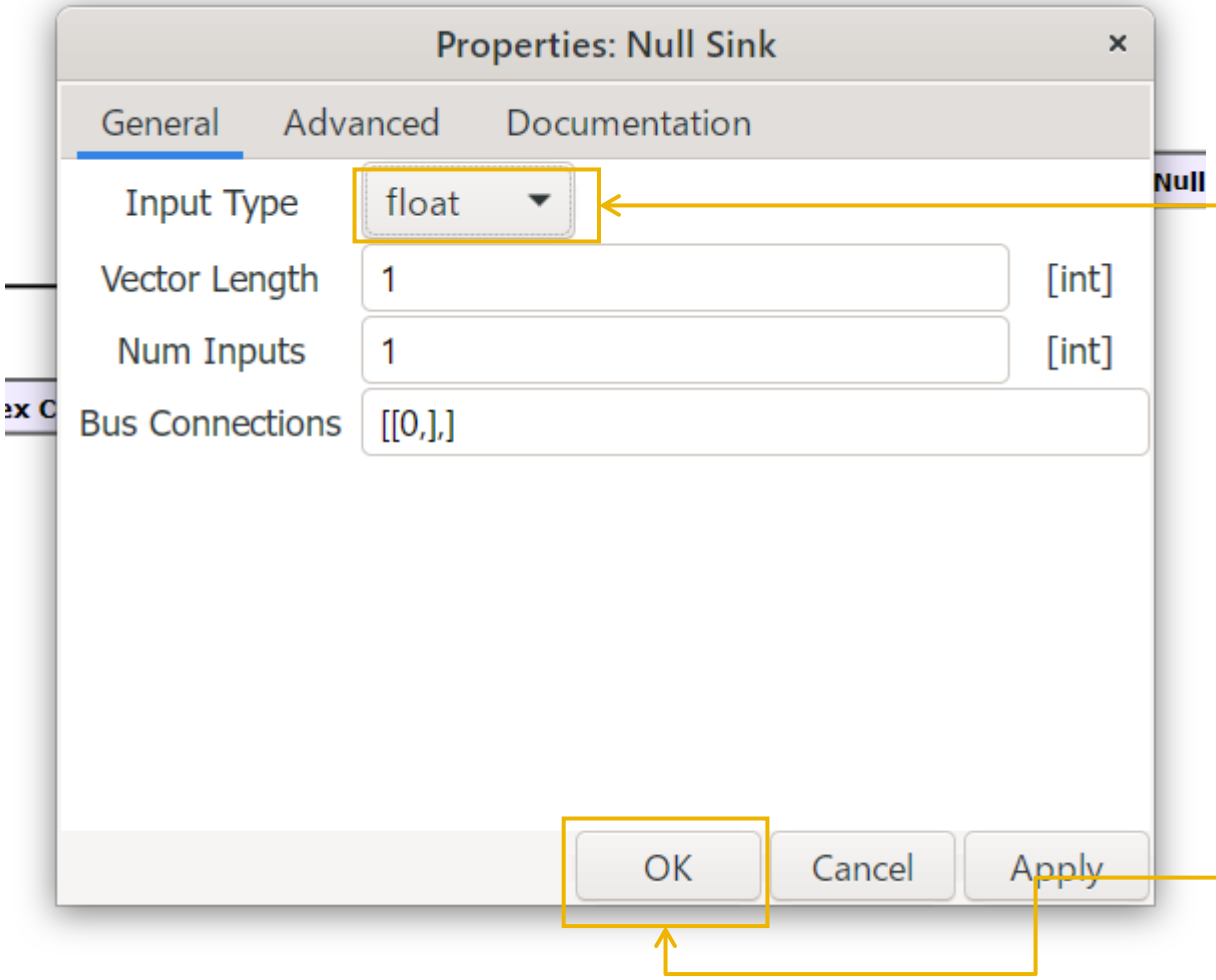
Complex to Mag Phaseを追加



Null Sinkの追加



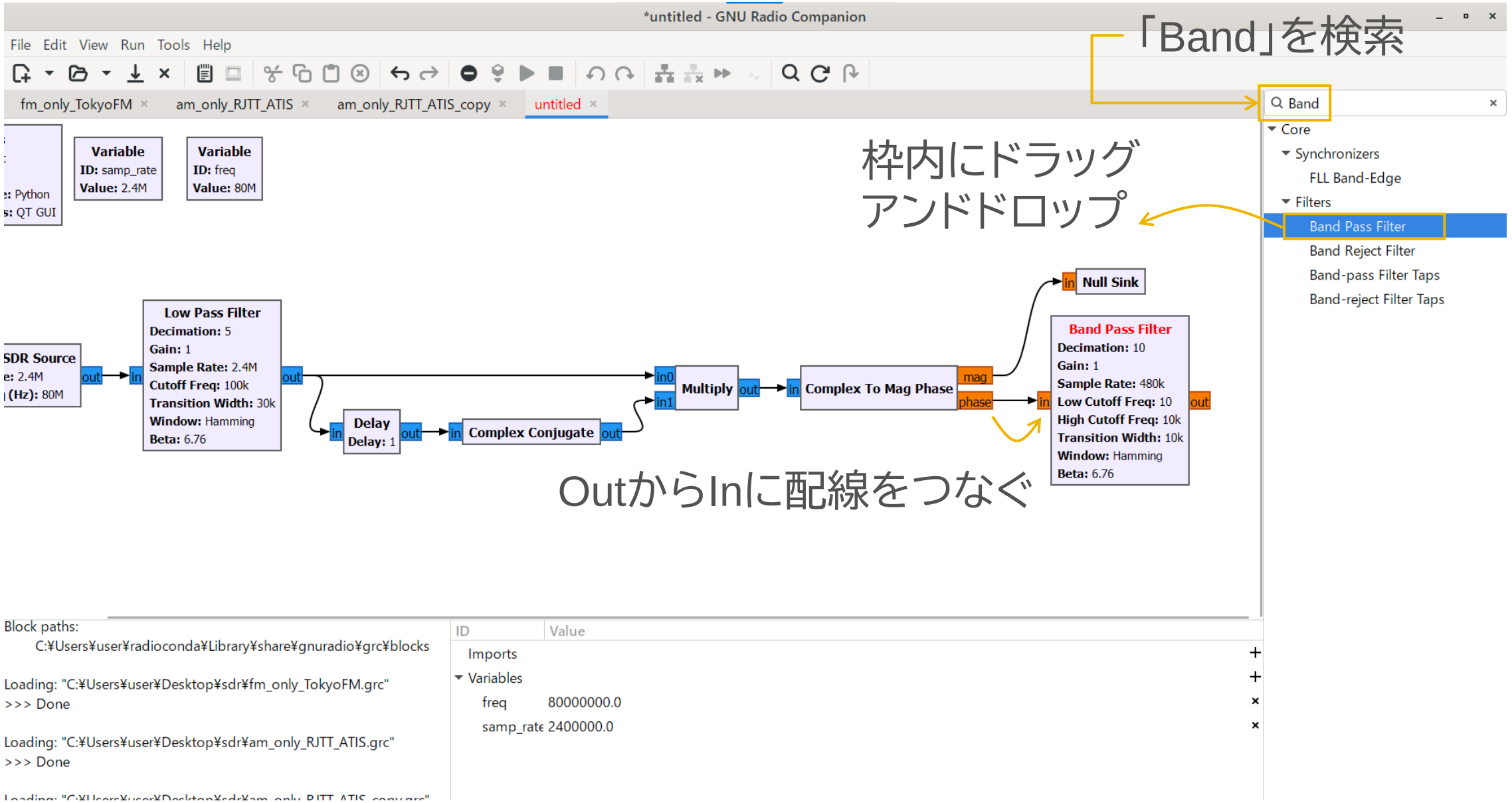
Null Sinkの設定



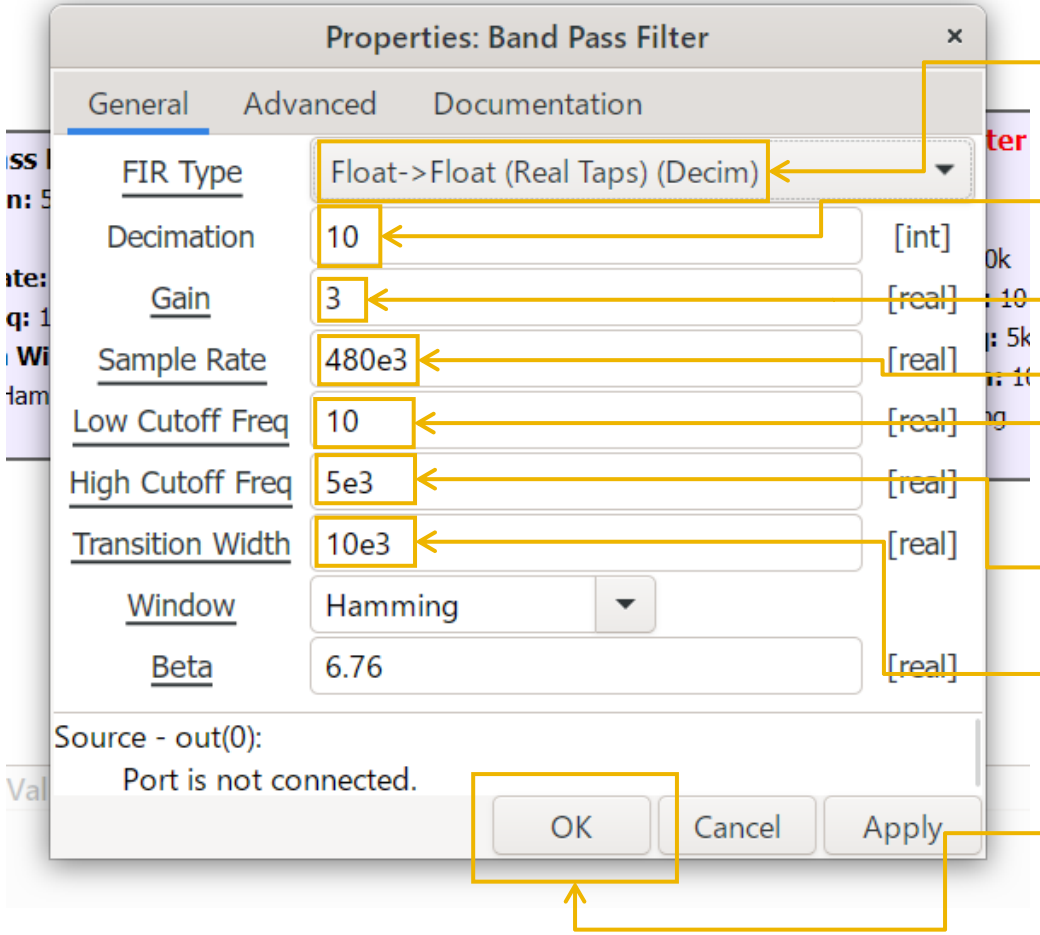
floatに変更

入力が終わったらOKをクリック

Band pass filterの追加



Band pass filter の設定



Float -> Float (Real Taps)(Decim)に変更

10を入力

3を入力

480e3を入力

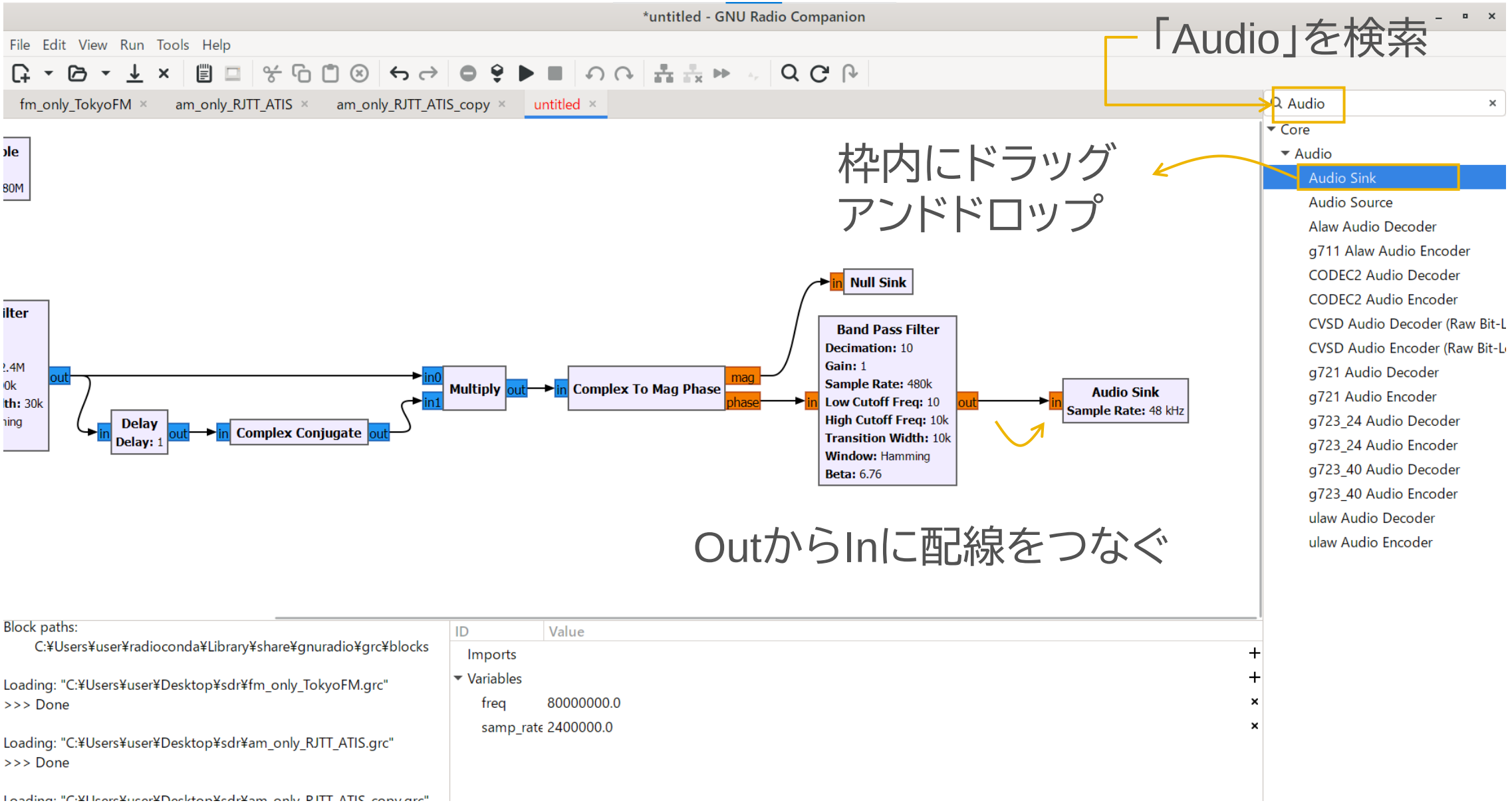
10を入力

5e3を入力

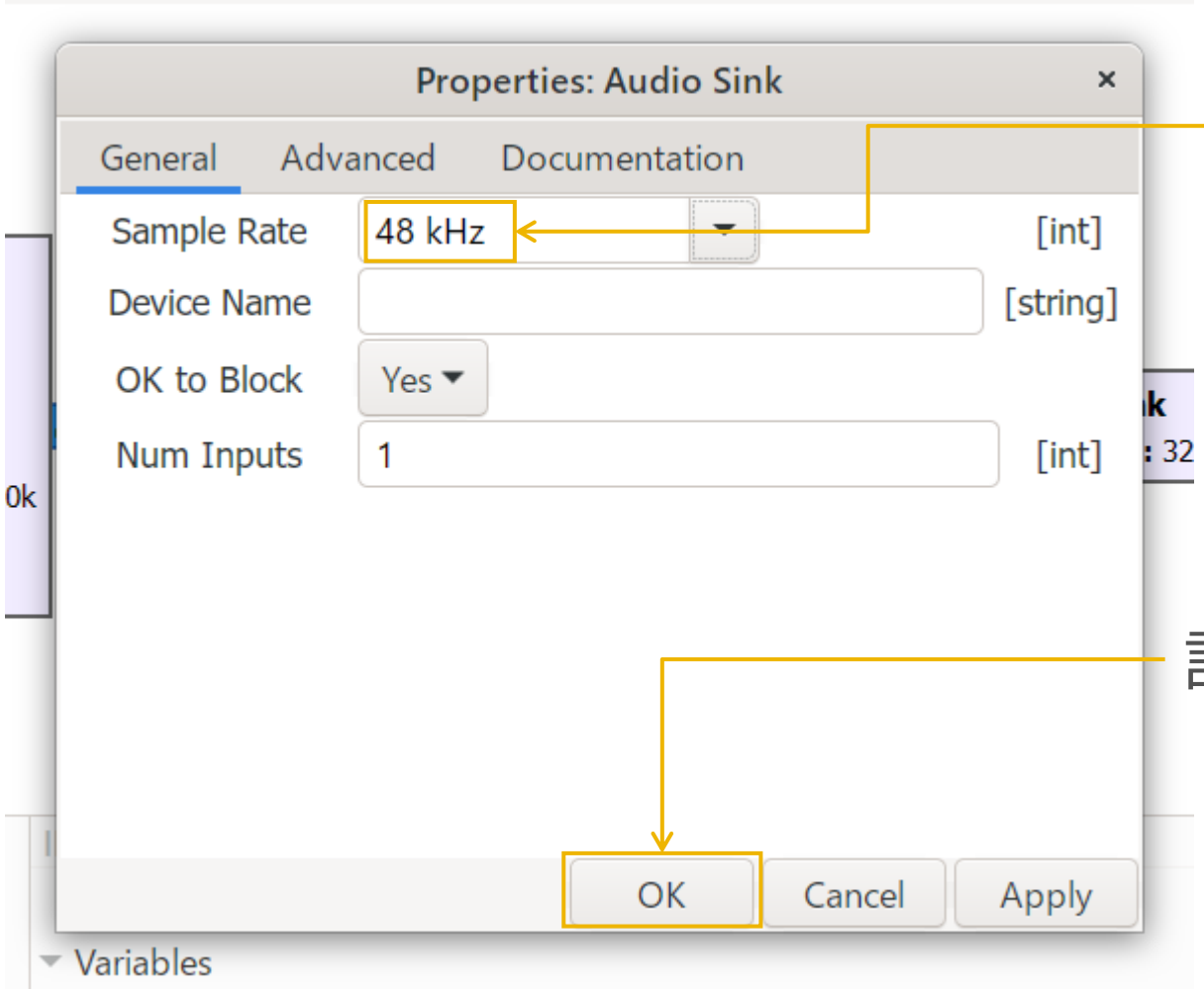
10e3を入力

入力が終わったらOKをクリック

Audio Sinkの追加



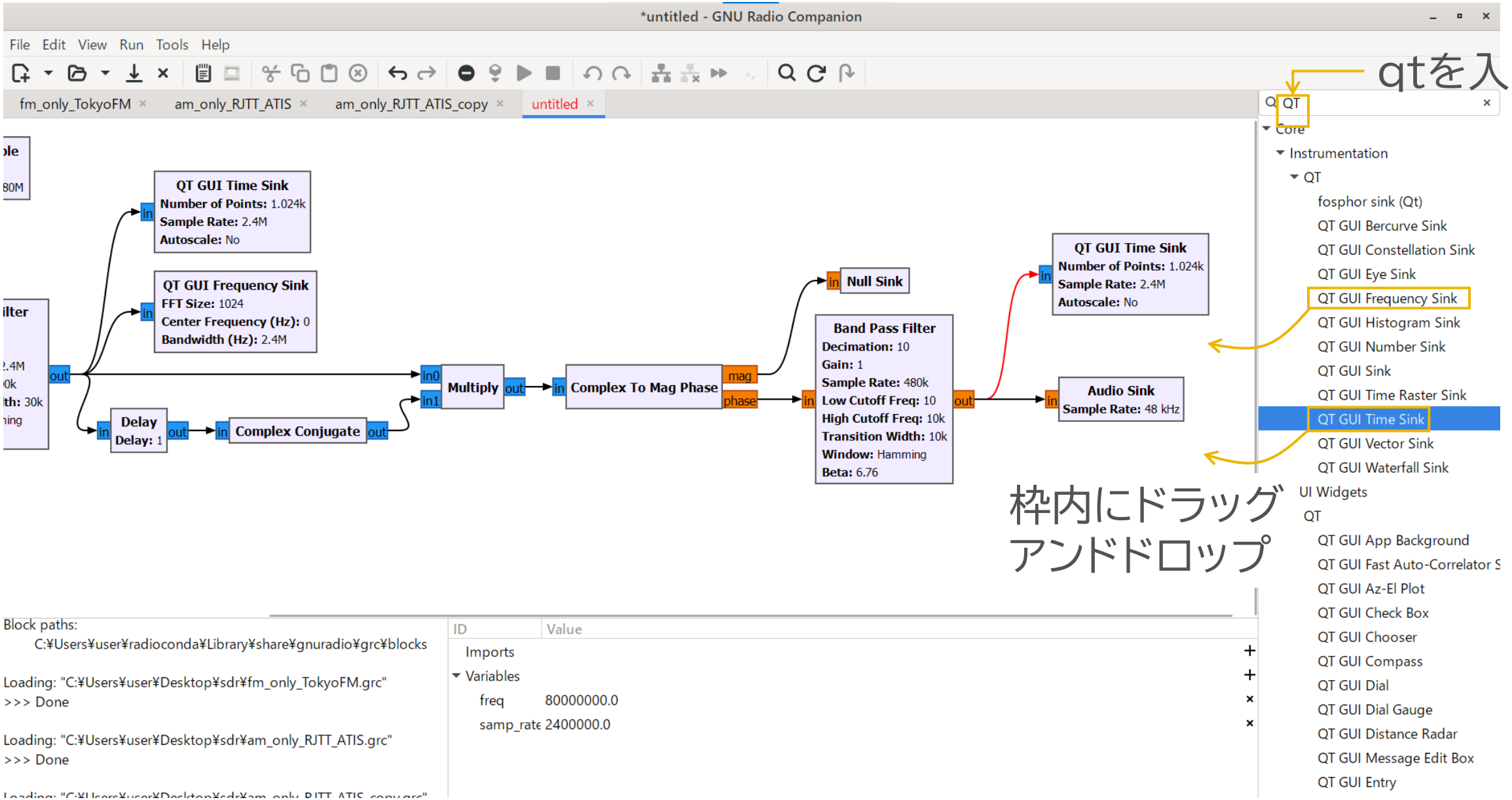
Audio Sinkの設定



プルダウンメニューから「48kHz」を選択

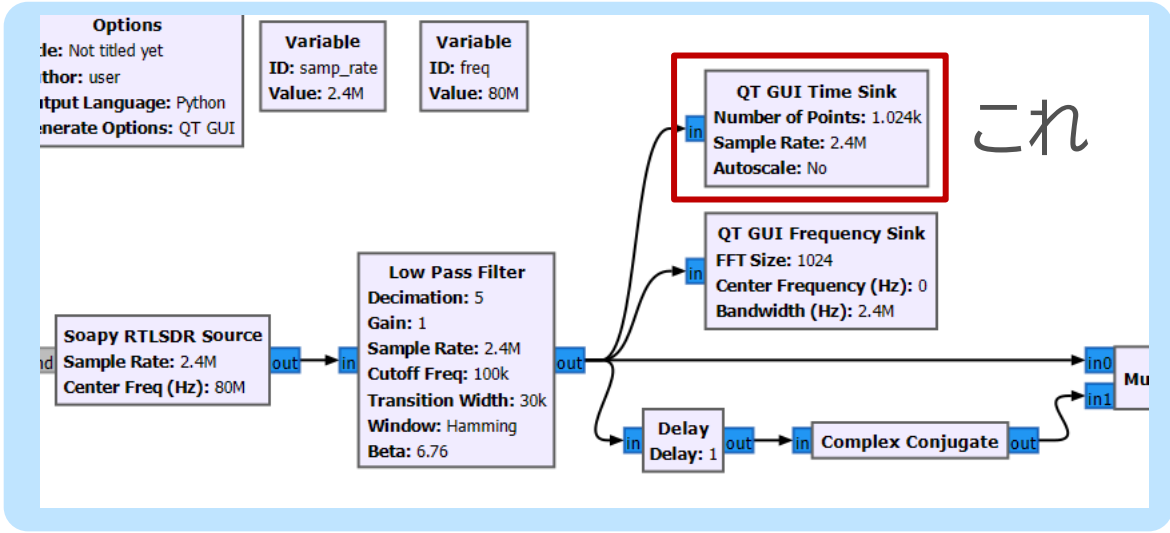
設定したらOKをクリック

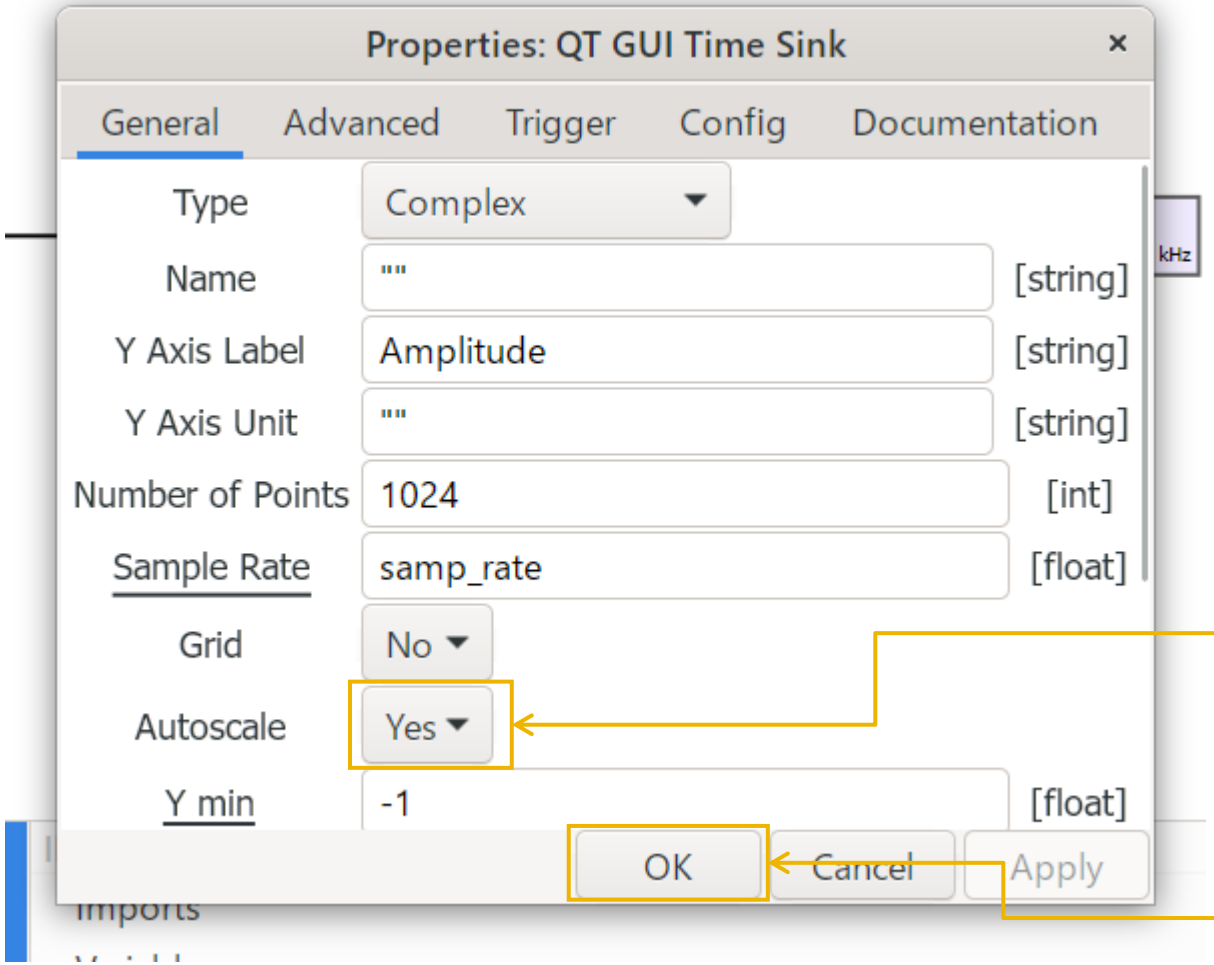
グラフの追加


*untitled - GNU Radio Companion
File Edit View Run Tools Help
fm_only_TokyoFM
am_only_RJTT_ATIS
am_only_RJTT_ATIS_copy
untitled
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
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
QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2.4M
Autoscale: No
Audio Sink
Sample Rate: 48 kHz
qtを入力
枠内にドラッグ
アンドドロップ
QT
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
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



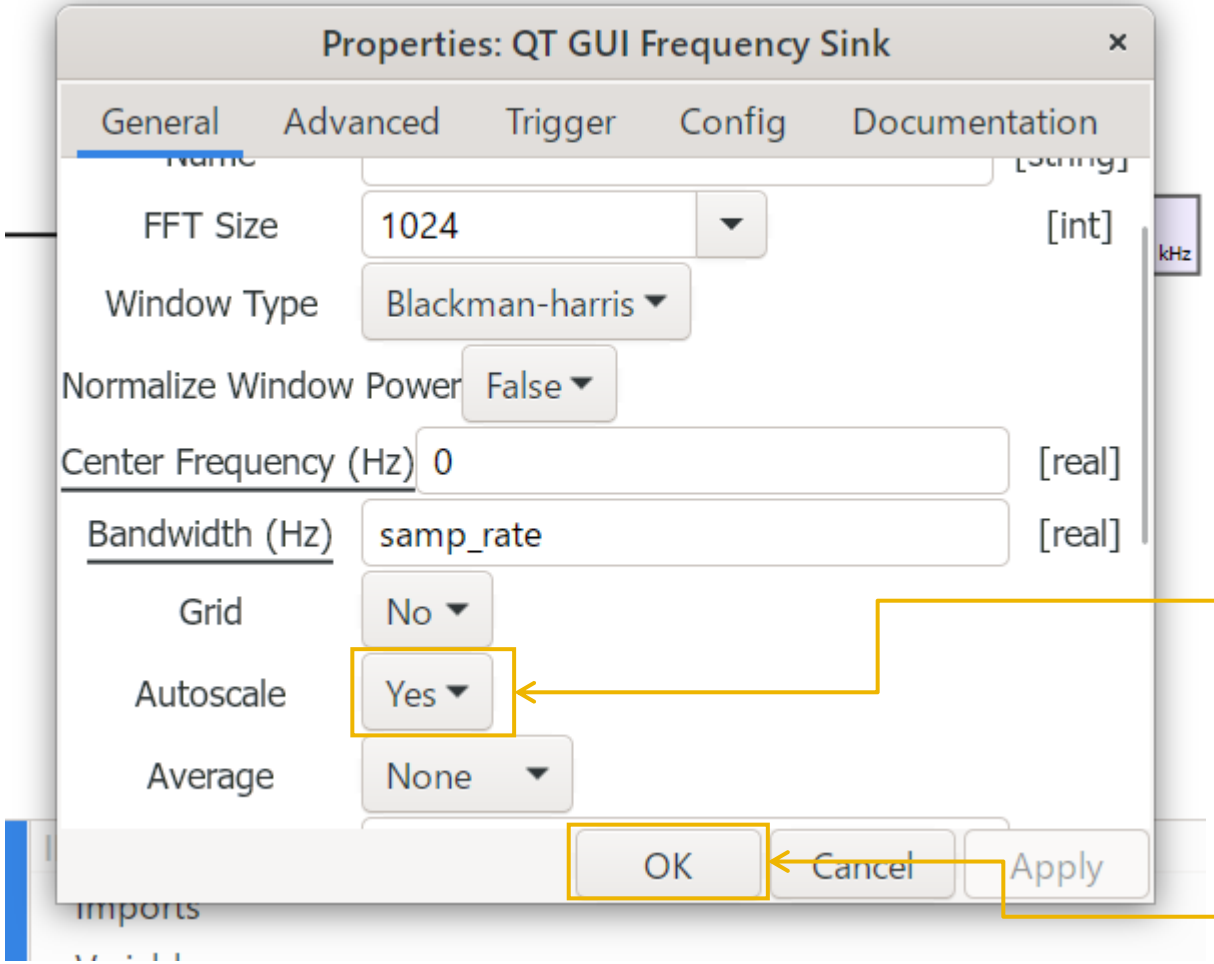


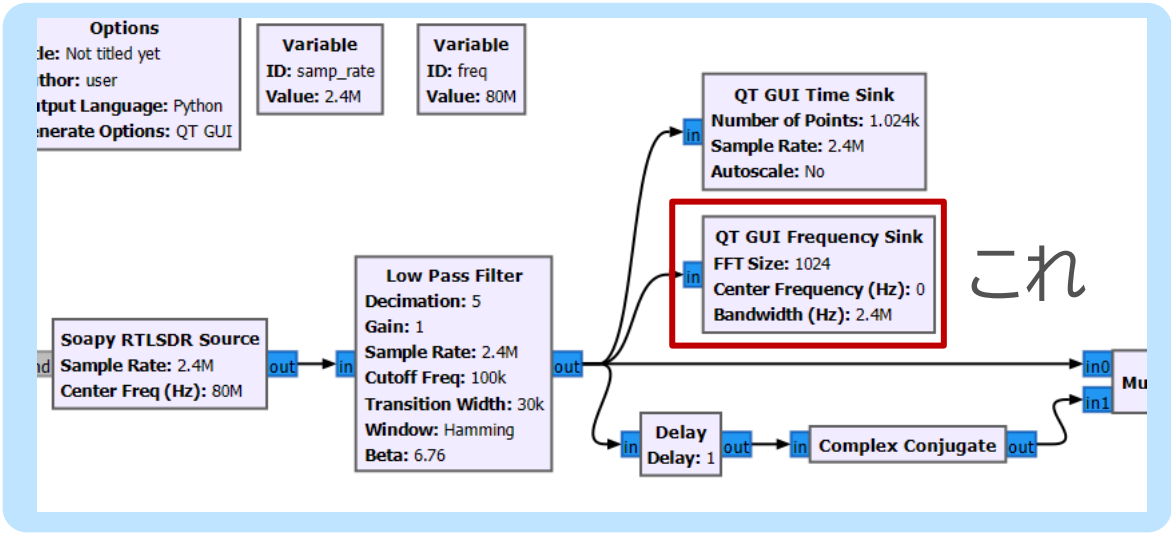
AutoscaleをYesに変更する

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



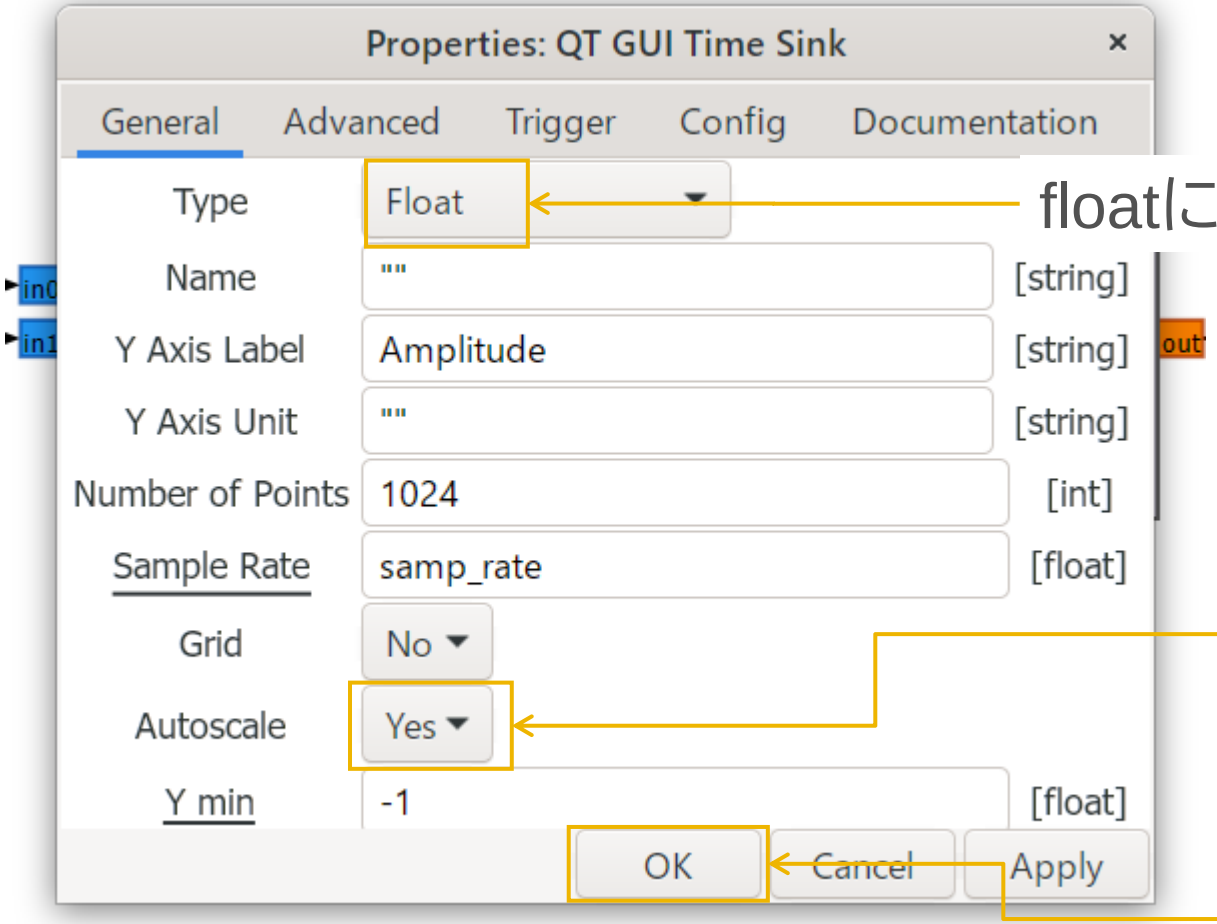


AutoscaleをYesに変更する

設定したらOKをクリック

2つ目のQT GUI Time sinkの設定

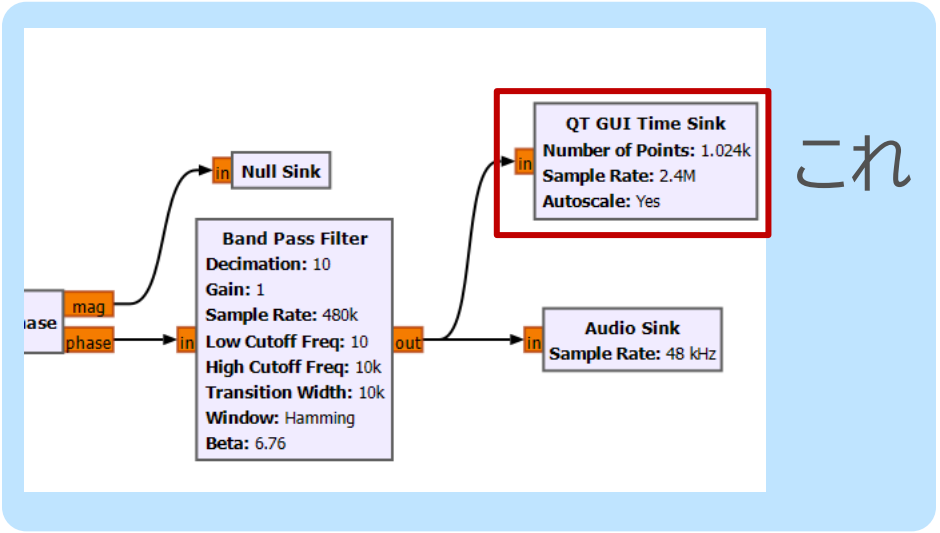
QT GUI Time sinkブロックをダブルクリックして設定を開く



floatに変更する

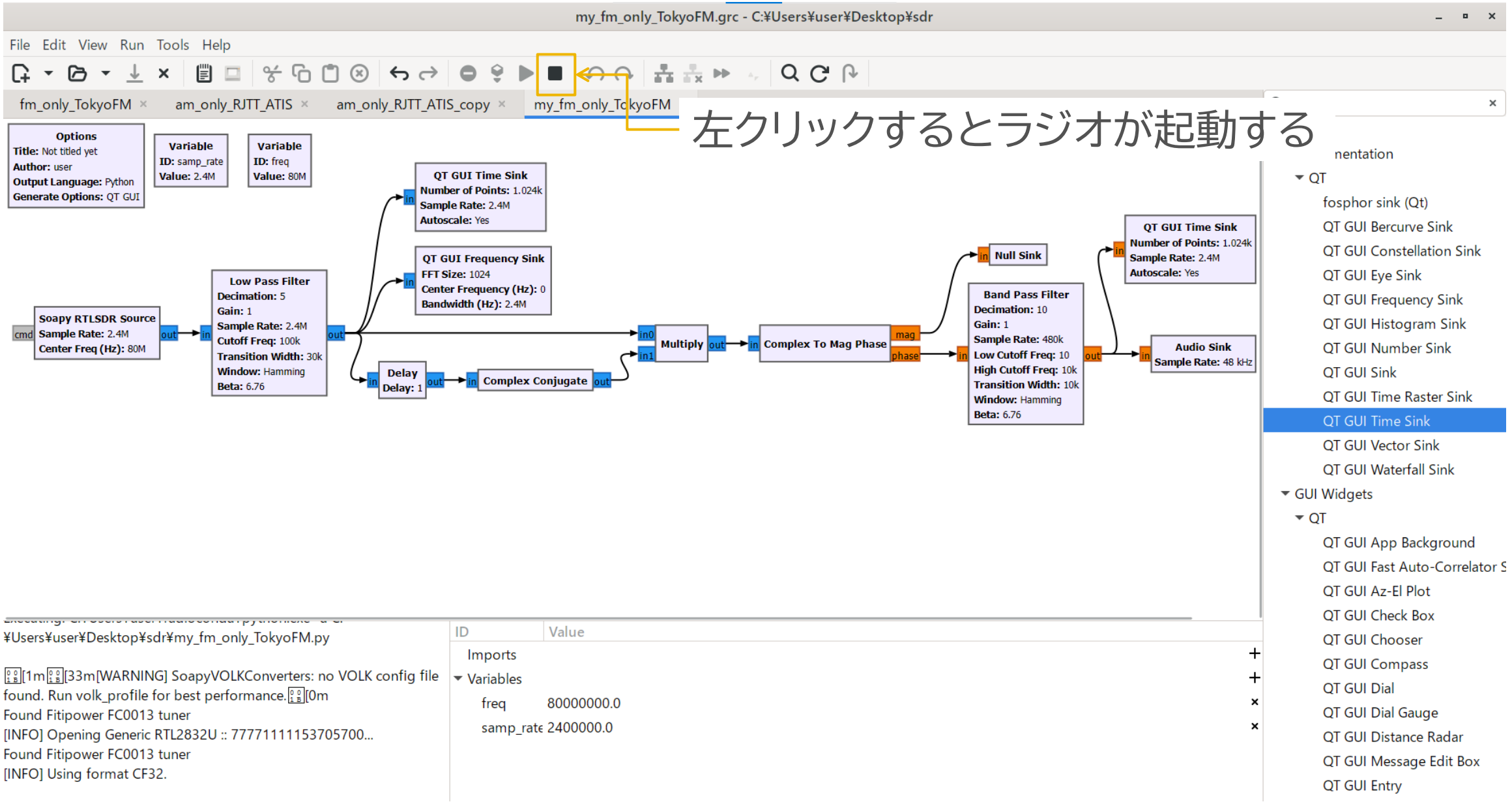
AutoscaleをYesに変更する

設定したらOKをクリック

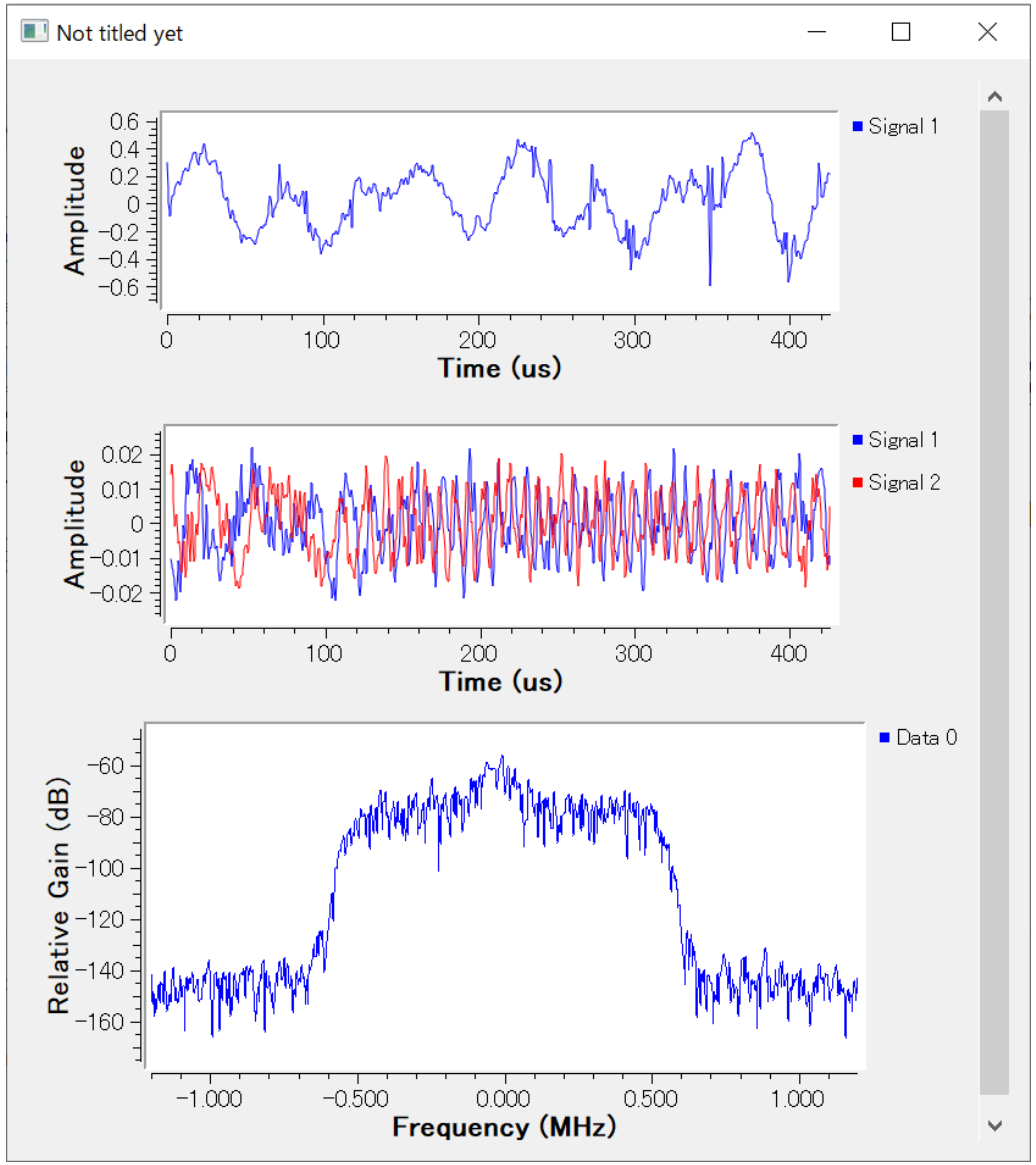


これ

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
am_only_RJTT_ATIS
am_only_RJTT_ATIS_copy
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
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
QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2.4M
Autoscale: Yes
Audio Sink
Sample Rate: 48 kHz
¥Users¥user¥Desktop¥sdr¥my_fm_only_TokyoFM.py
[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.
ID
Value
Imports
Variables
freq 80000000.0
samp_rate 2400000.0
nentation
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

こんな画面が見られるとOK



TokyoFMが再生されるはず