ソフトウェア無線機

第2回目

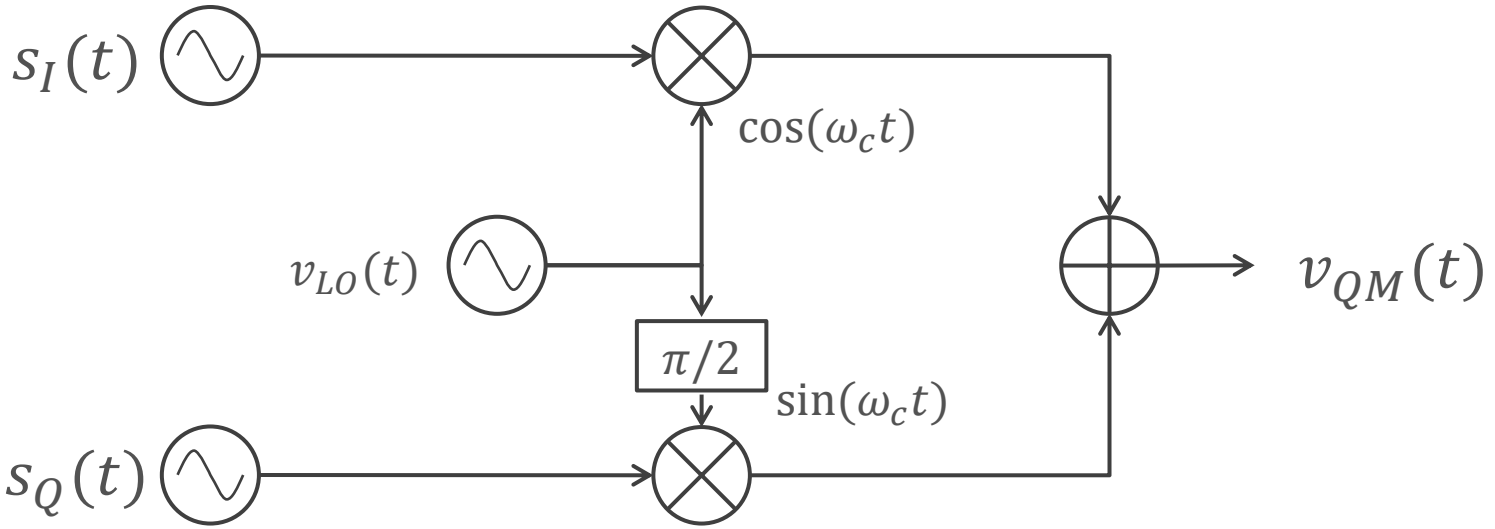
電子磁気応用資料

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

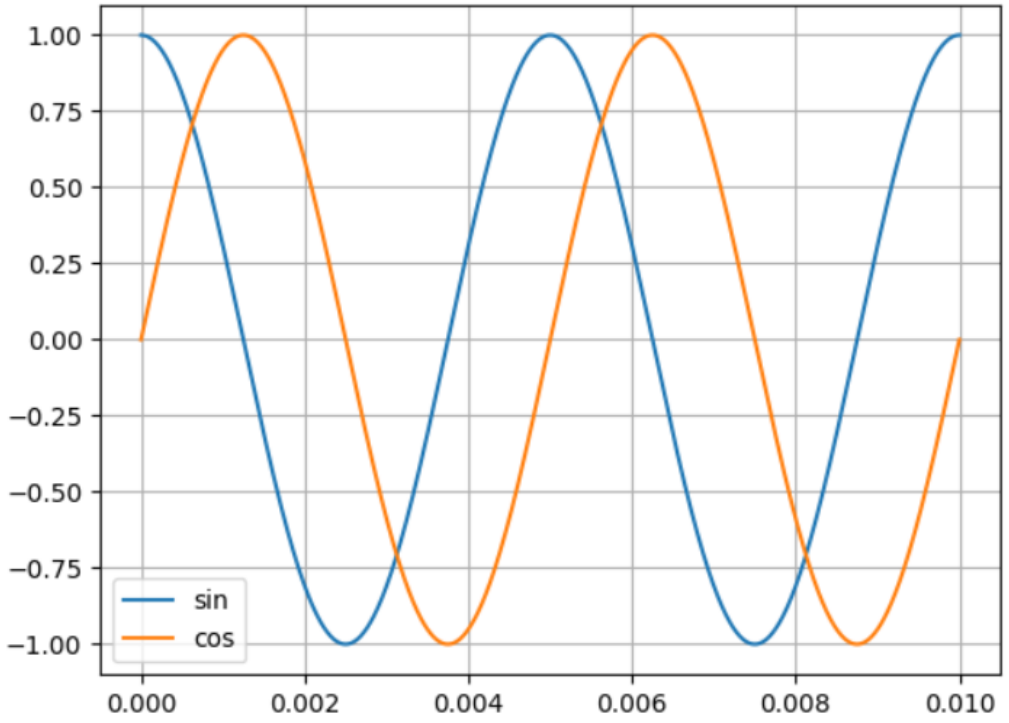
直交変調器による変調

直交変調器（IQ変調器）

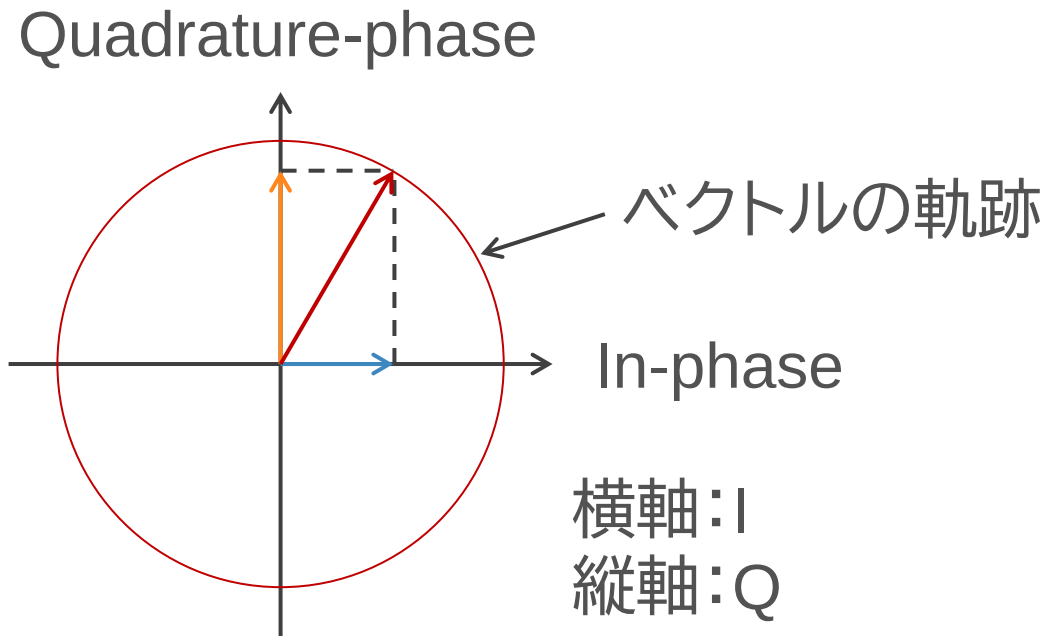


$$v_{QM}(t) = s_I(t)\cos(\omega_c t) + s_Q(t)\sin(\omega_c t)$$

sinとcosとIとQの関係

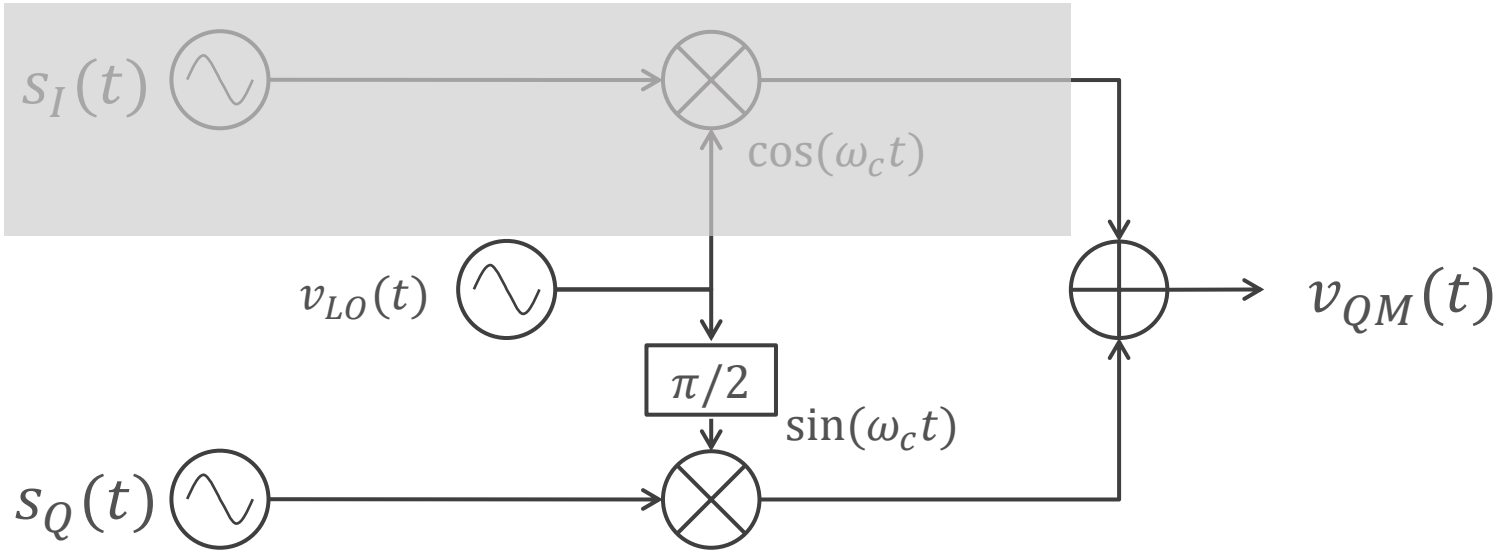


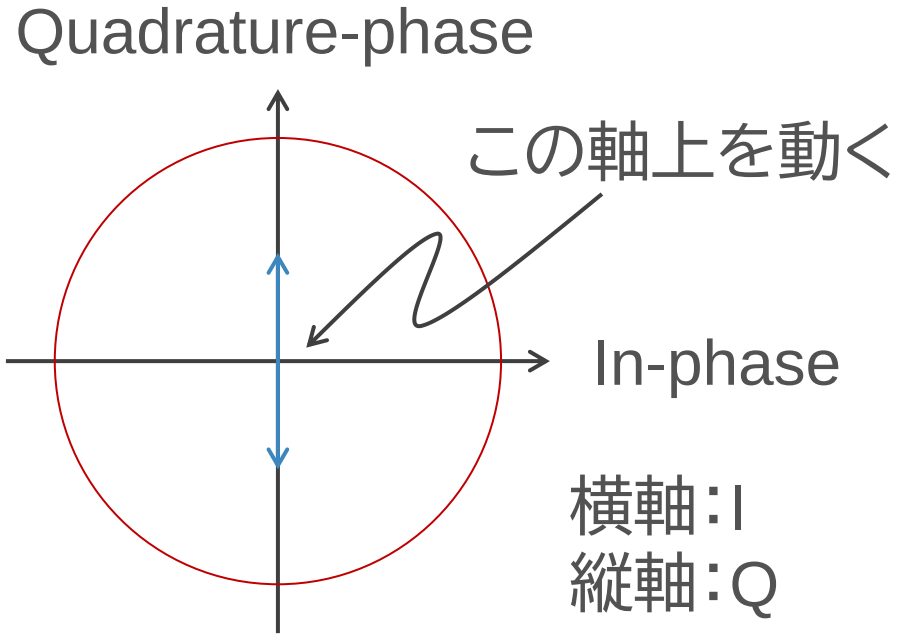
sinとcosは直交関係にある（内積がゼロ）



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AM





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

直交変調器を用いてAMを実現するにはどちらか一方のみに信号を入れれば良い。

直交変調器によるAMのシミュレーションコード

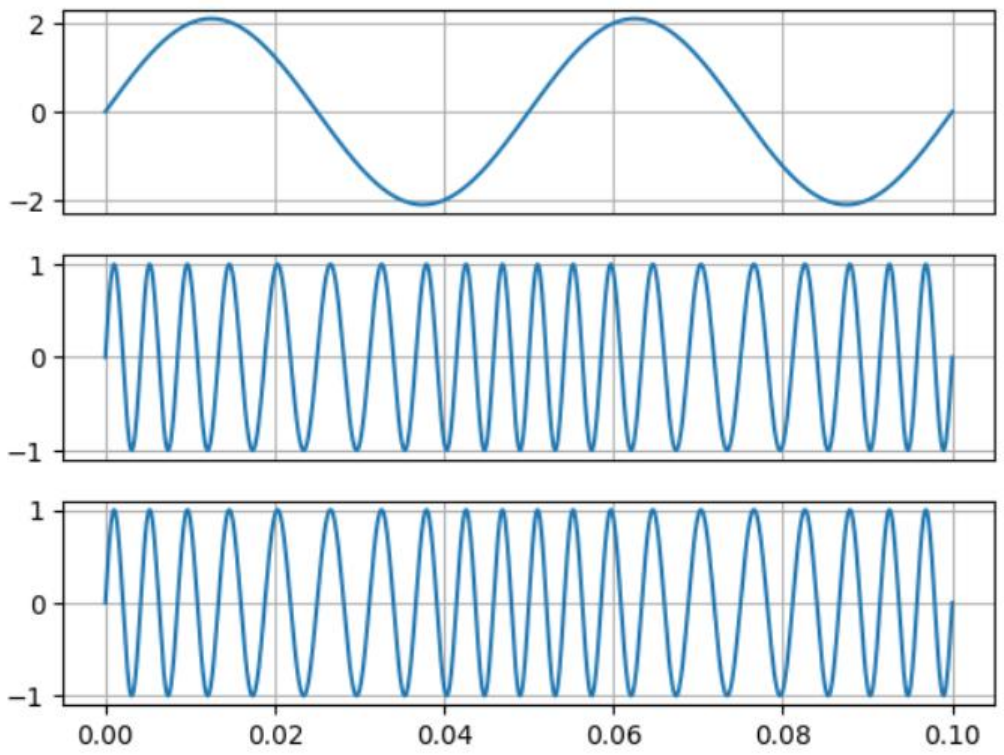
```
import numpy as np
import matplotlib.pyplot as plt

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

m = 0.2
fs = 20
vs = 1 + m*np.sin(2*fs*np.pi*t)

vam = vs*vcq
vqm = 0*vci + vs*vcq

fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vc)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vs)
az = fig.add_subplot(413)
az.grid()
az.plot(t, vam)
```

キャリアの信号
90度シフトしたキャリアの信号
信号波形
従来のAM
直交変調器の処理

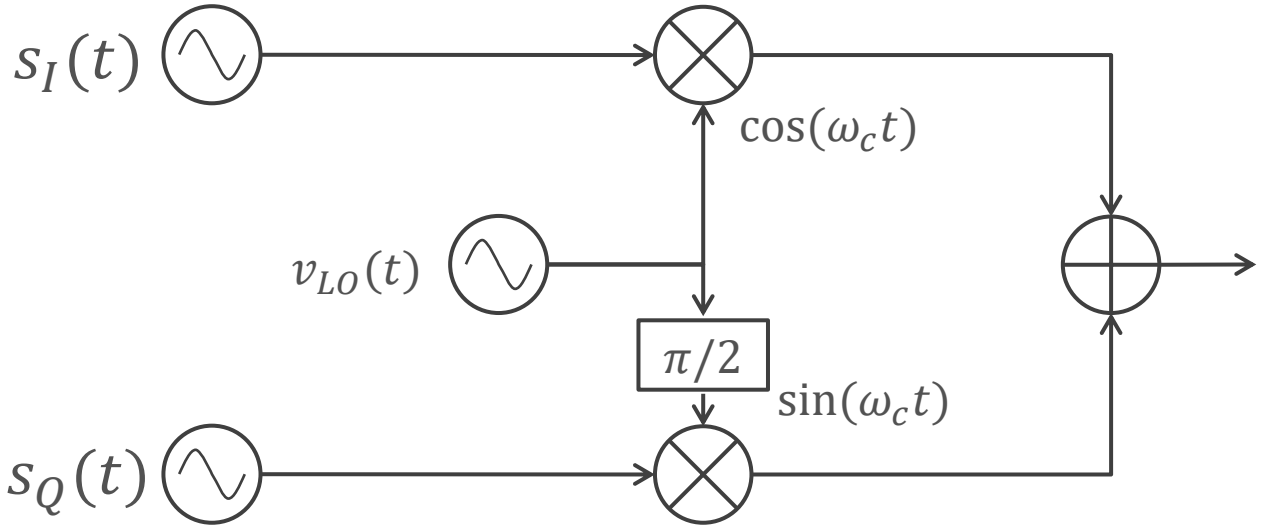
```
aa = fig.add_subplot(414)
aa.plot(t, vqm)
aa.grid()
axes = fig.get_axes()
for n in axes:
  n.tick_params(axis='x', which='both',
bottom=False, labelbottom=False)
n.tick_params(axis='x', which='both',
bottom=True, labelbottom=True)
```

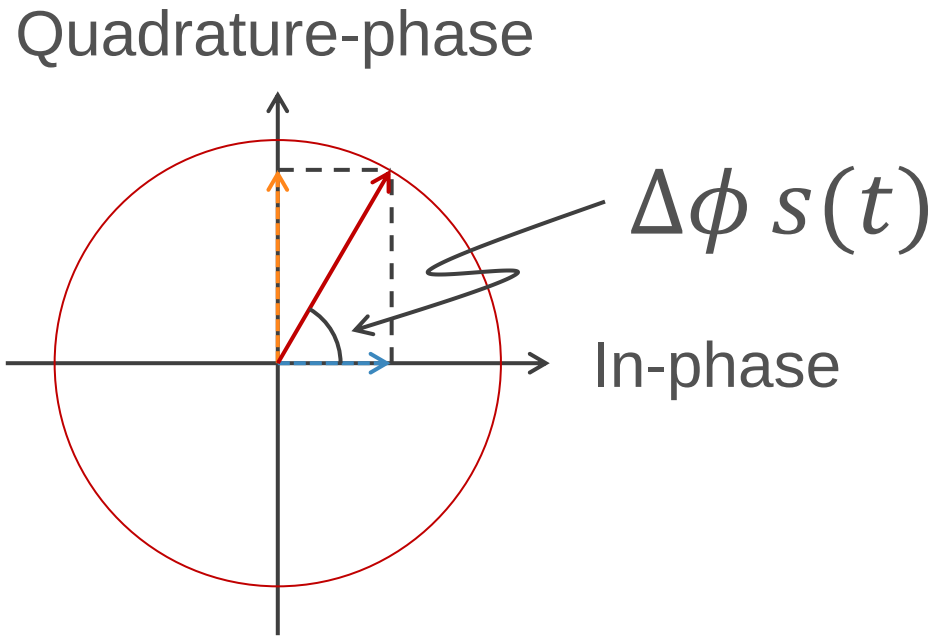
Colabを用いたシミュレーション

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

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_phi = 120/180*np.pi
fs = 20
vs = delta_phi * np.sin(2*fs*np.pi*t)

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

vpm = np.sin(2*fc*np.pi*t + vs)
vqm = vci*vsq + vcq*vsi

fig = plt.figure()
ax = fig.add_subplot(311)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(312)
ay.grid()
ay.plot(t, vpm)
az = fig.add_subplot(313)
az.grid()
az.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)
```



直交変調器を用いたPM





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

$$s_I(t) = \cos(\Delta\phi s(t))$$
$$s_Q(t) = \sin(\Delta\phi s(t))$$

IとQのそれぞれの振幅に分解して入力する

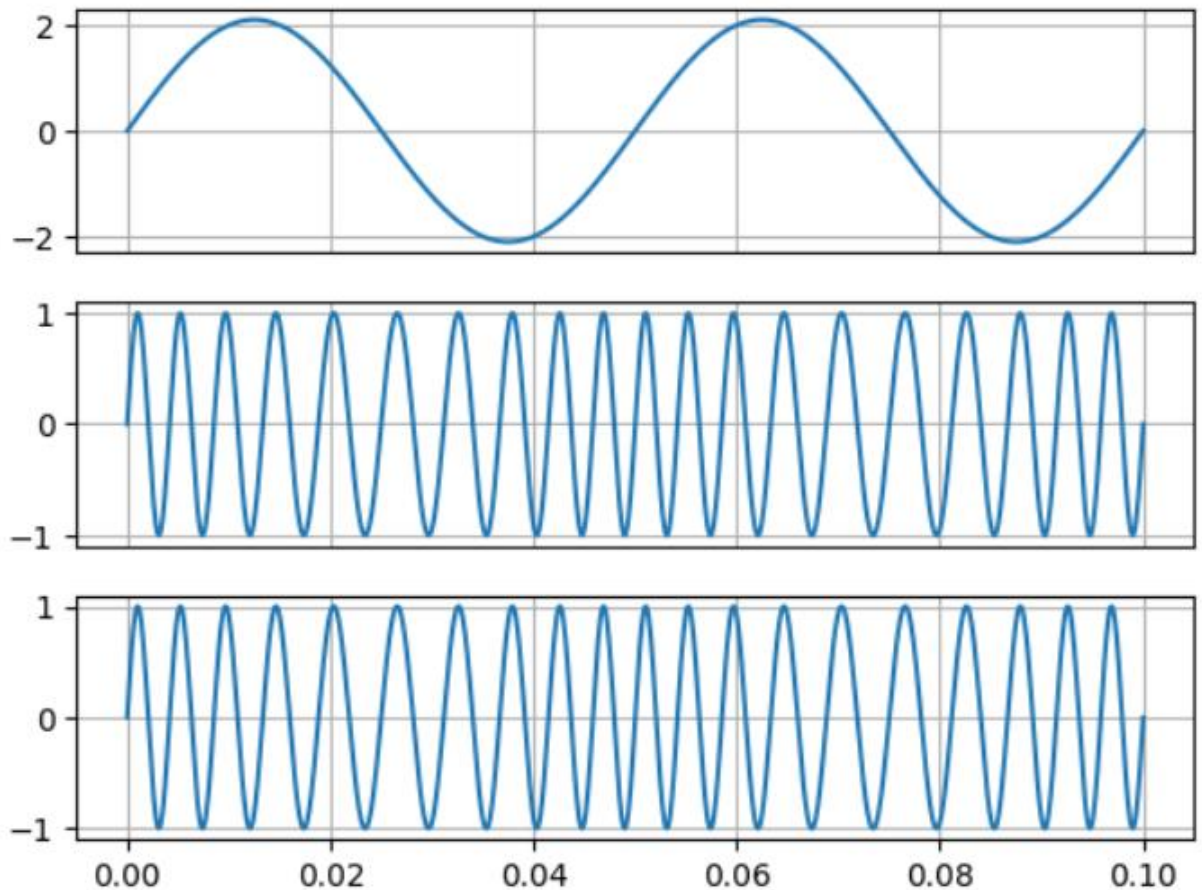
直交変調器によるPMのシミュレーションコード

```
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_phi = 120/180*np.pi
fs = 20
vs = delta_phi * np.sin(2*fs*np.pi*t)

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

vpm = np.sin(2*fc*np.pi*t + vs)
vqm = vci*vsq + vcq*vsi

fig = plt.figure()
ax = fig.add_subplot(311)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(312)
ay.grid()
ay.plot(t, vpm)

az = fig.add_subplot(313)
az.grid()
az.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)
```

キャリアの信号
90度シフトしたキャリアの信号
PMの信号波形
I入力の処理
Q入力の処理
従来のPM
直交変調器の処理

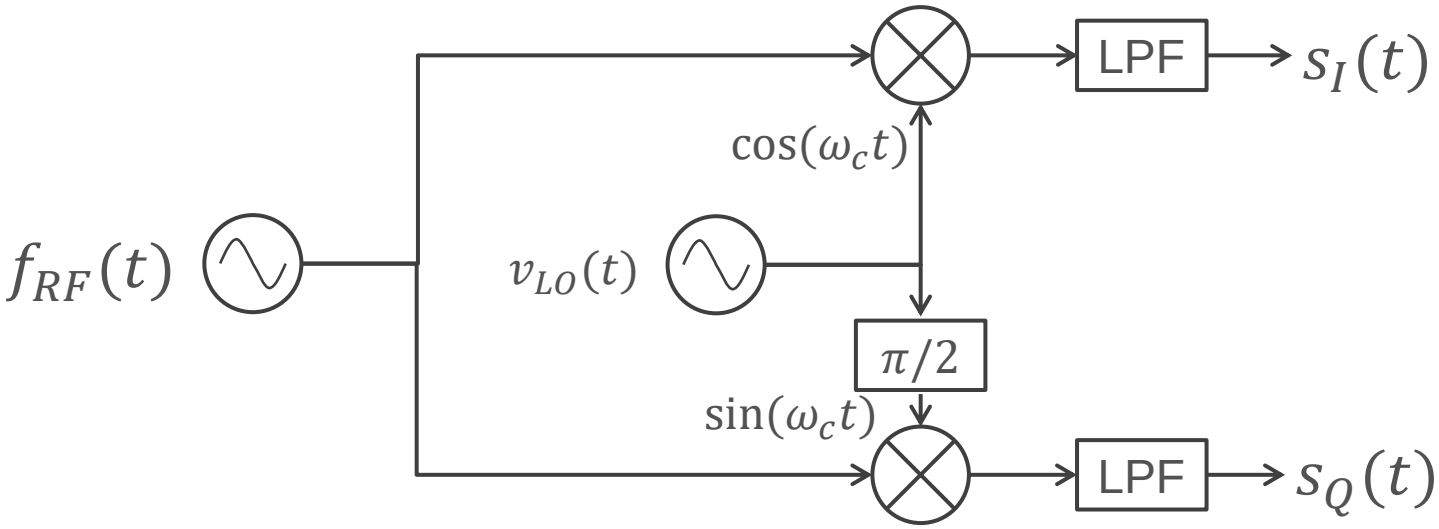
Colabを用いたシミュレーション

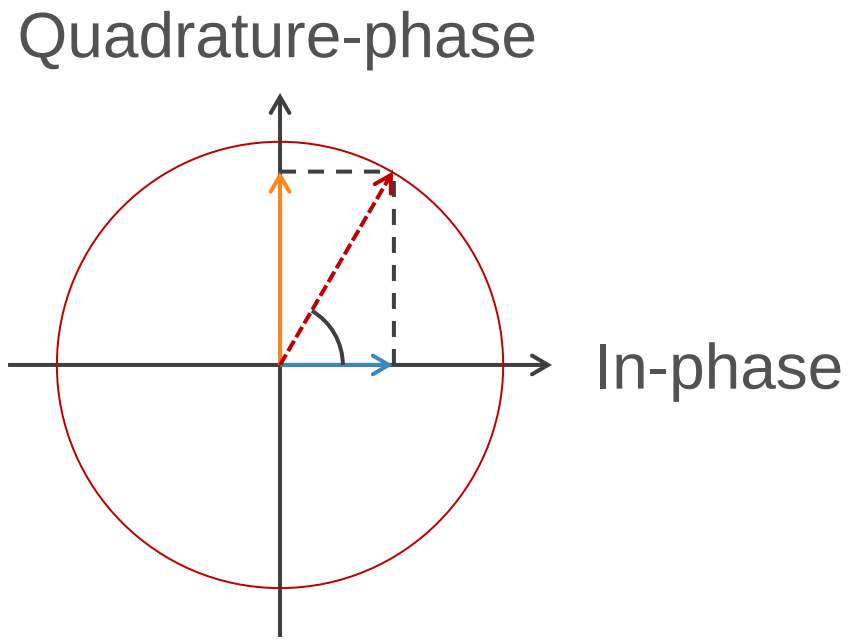
```
import numpy as np
import matplotlib.pyplot as plt

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_phi = 120/180*np.pi
fs = 20
vs = delta_phi * np.sin(2*fs*np.pi*t)

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

vpm = np.sin(2*fc*np.pi*t + vs)
vqm = vci*vsq + vcq*vsi

fig = plt.figure()
ax = fig.add_subplot(311)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(312)
ay.grid()
ay.plot(t, vpm)
az = fig.add_subplot(313)
az.grid()
az.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)





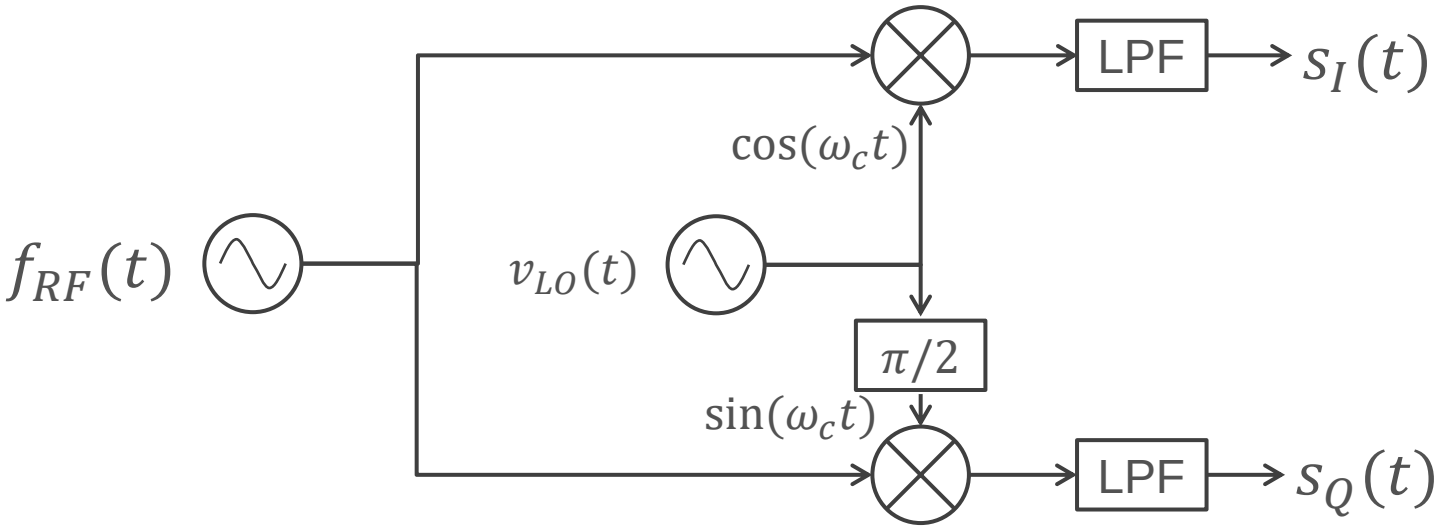
直交復調器を用いるとLOと
- 同相の信号成分(I)
- 直交する信号成分(Q)

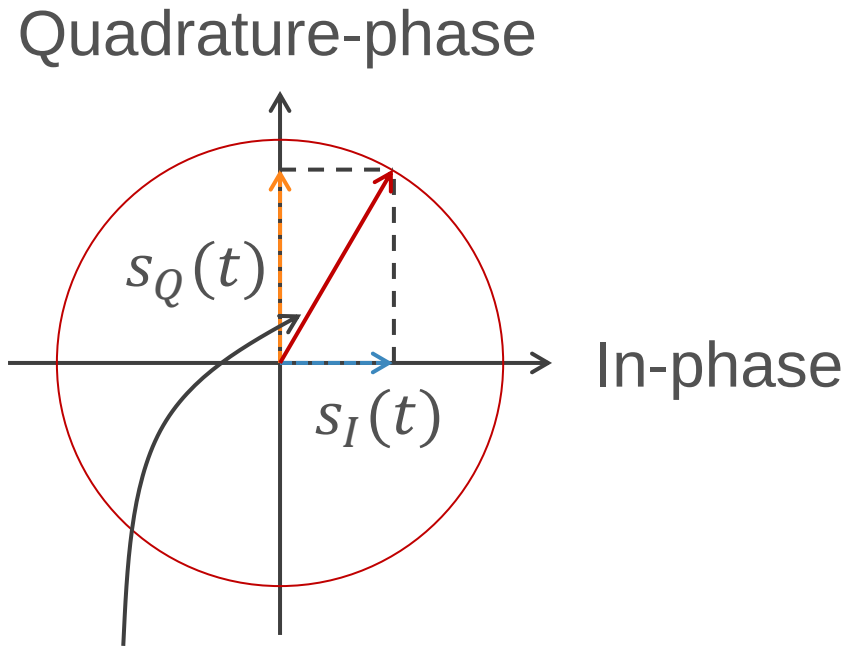
の2つを取り出すことができる。

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AM復調





元の信号を取り出すには振幅成分を取り出す。
振幅成分は長さを求めれば良いので、

振幅が元の信号の成分

$$s(t) = \sqrt{s_I(t)^2 + s_Q(t)^2}$$

と書くことができる。

直交変調器によるAM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)

m = 0.2
fs = 20
vs = 1 + m*np.sin(2*fs*np.pi*t)
vam = vs*vcq

vifi = vam*vci
vifq = vam*vcq

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

v_s_demod = np.sqrt(v_i_dec*v_i_dec + v_q_dec*v_q_dec)
```

← キャリアの信号

← 90度シフトしたキャリアの信号

AM信号生成

LO信号とRF信号の掛け算

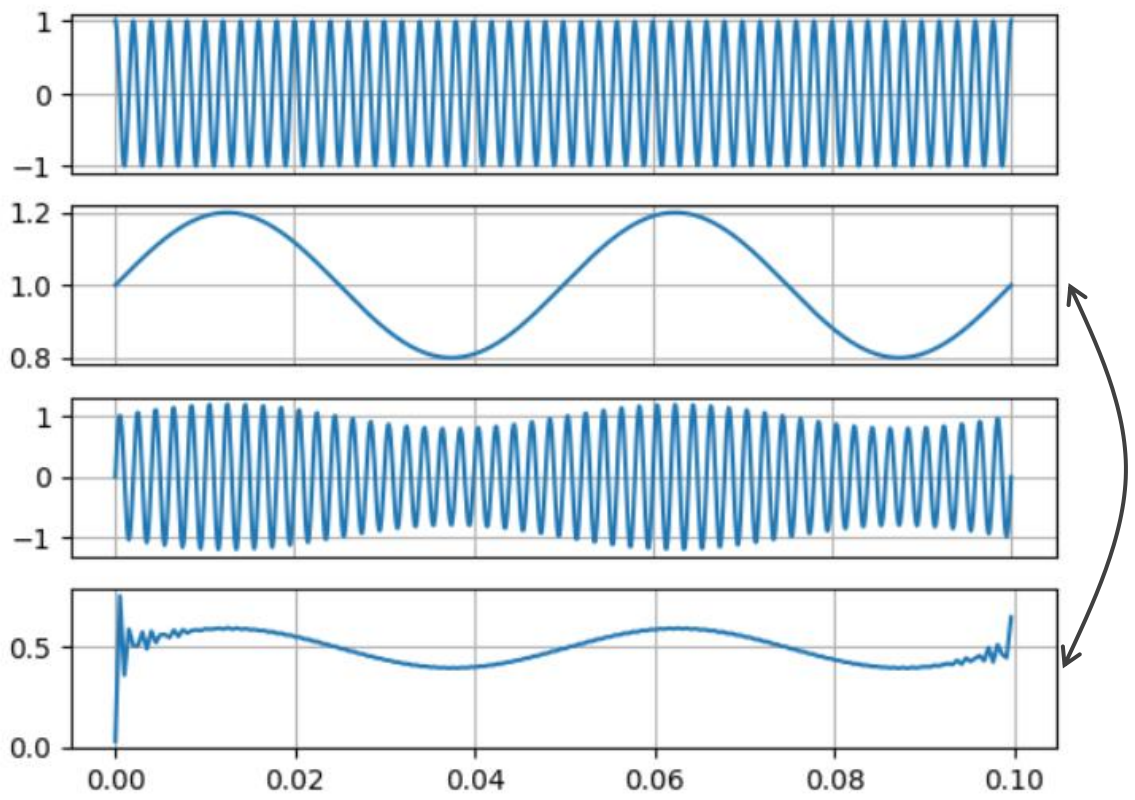
LPF処理

IとQの長さを求める処理

```
fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vci)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vs)
az = fig.add_subplot(413)
az.grid()
az.plot(t, vam)
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)

m = 0.2
fs = 20
vs = 1 + m*np.sin(2*fs*np.pi*t)
vam = vs*vcq

vifi = vam*vci
vifq = vam*vcq

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

v_s_demod = np.sqrt(v_i_dec*v_i_dec + v_q_dec*v_q_dec)

fig = plt.figure()
ax = fig.add_subplot(411)
ax.grid()
ax.plot(t, vci)
ay = fig.add_subplot(412)
ay.grid()
ay.plot(t, vs)
az = fig.add_subplot(413)
az.grid()
az.plot(t, vam)
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)
```



復調した波形は変調前の波形と同じ波形が得られている。

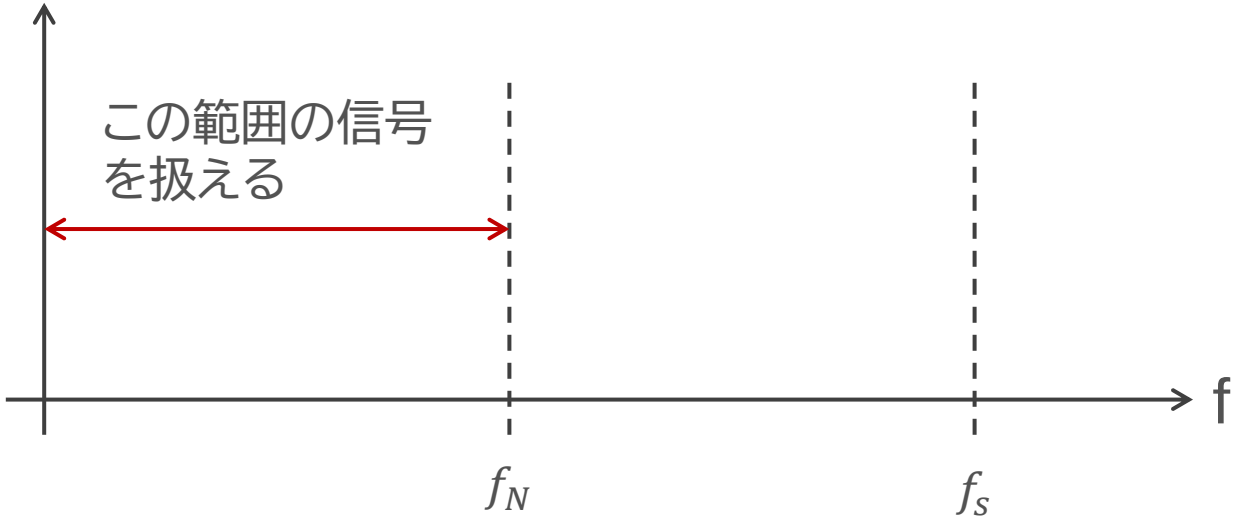
ナイキスト周波数

サンプリング周波数と扱える周波数には次の関係がある

$$f_N = \frac{f_s}{2}$$

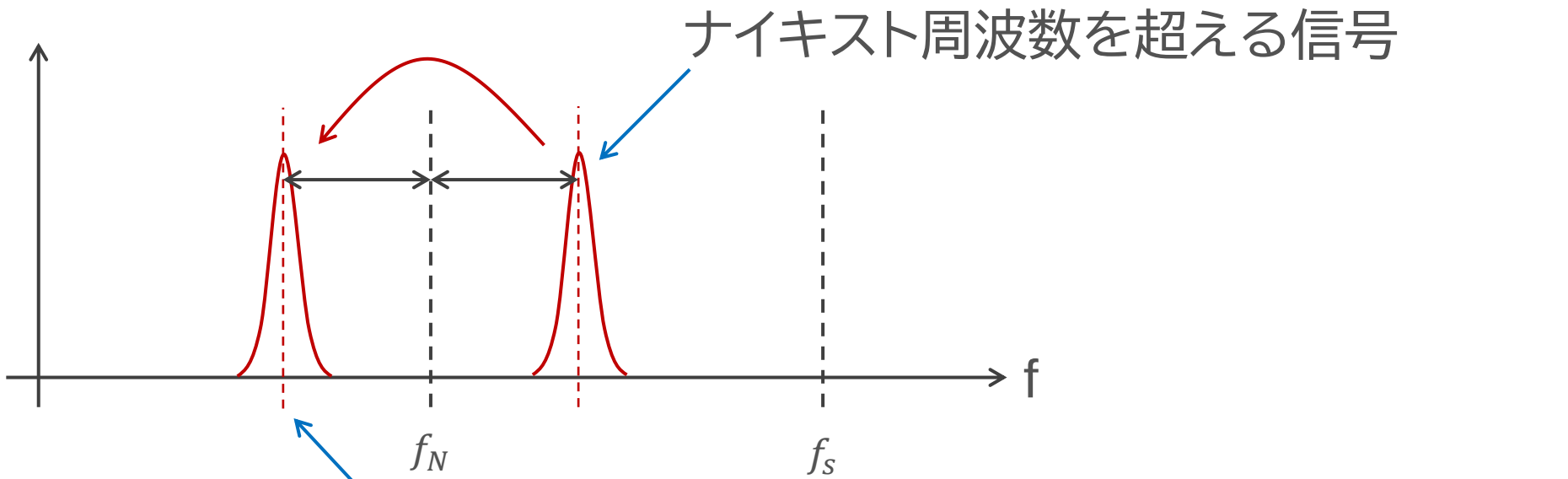
f_s :サンプリング周波数
f_N :ナイキスト周波数

サンプリング周波数の半分の周波数を**ナイキスト周波数**といい、
これ以上の周波数は扱うことができない。



折返し(エイリアシング)

ナイキスト周波数を超える信号はどうなるのか?

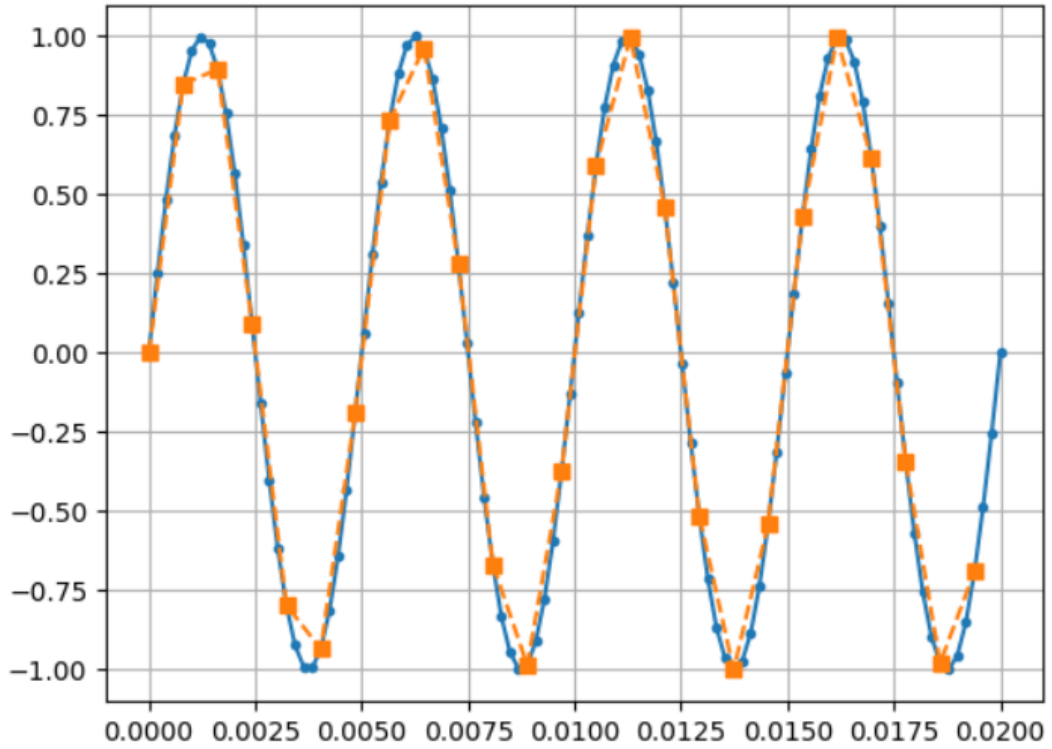


デシメーション

信号を間引いてサンプリング周波数を下げる処理をデシメーションという

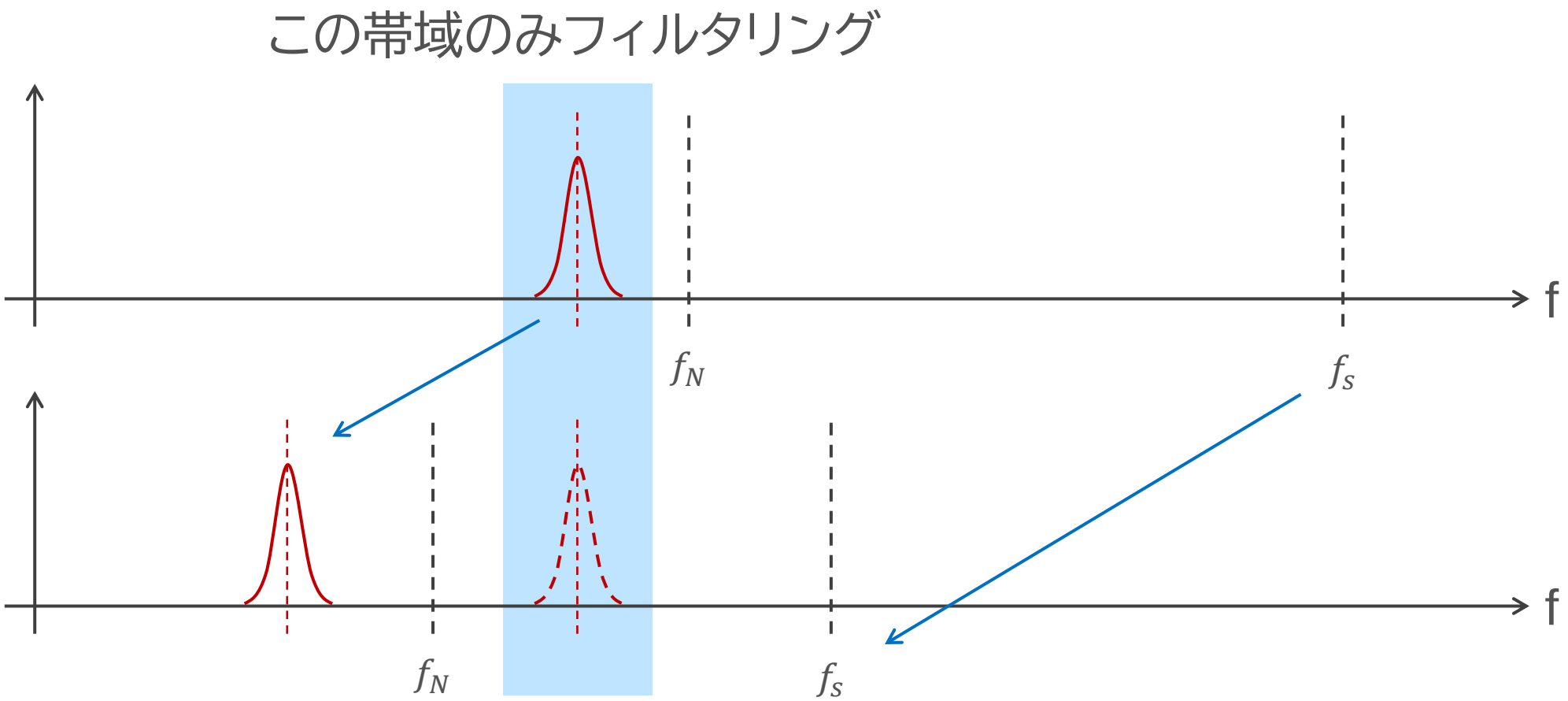
必要とする信号の周波数の2倍のサンプリング周波数で元の信号を再現できる

- サンプリング周波数を下げることで計算負荷を下げられる。
- ナイキスト周波数が変わるので、周波数変換ができる



デシメーションによる周波数変換

デシメーションを用いると周波数変換ができる

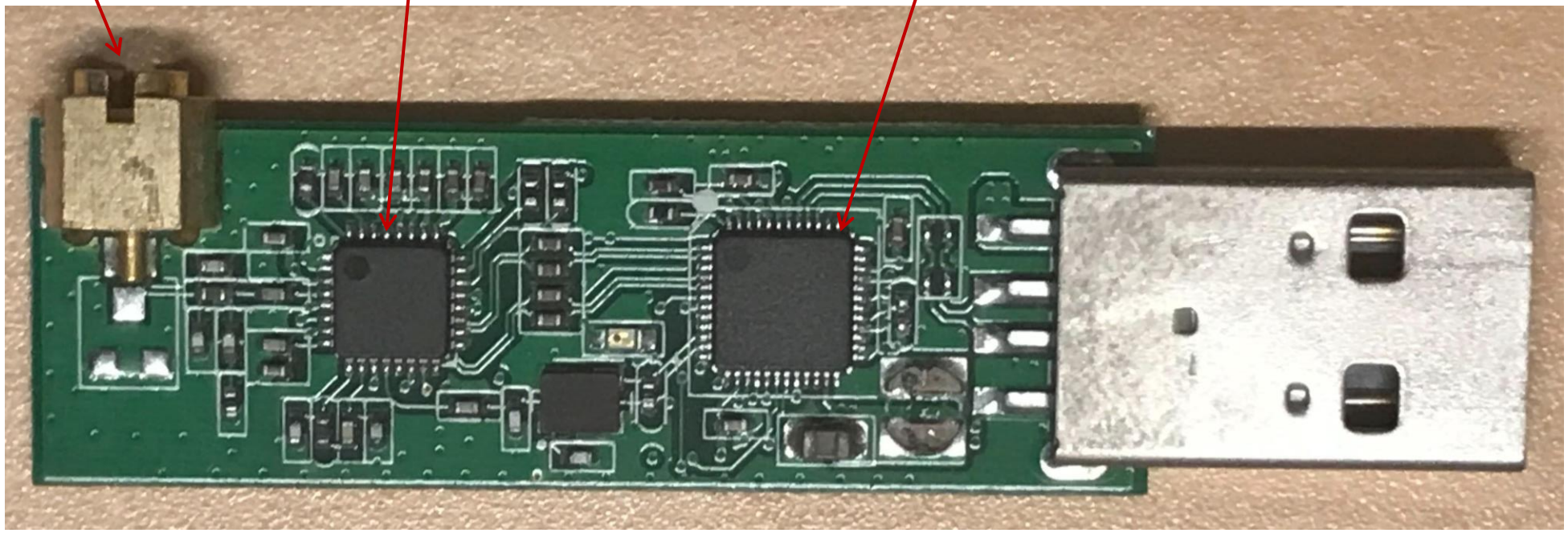


このように折り返しを用いて周波数を変換することができる。

ソフトウェア無線機の構成

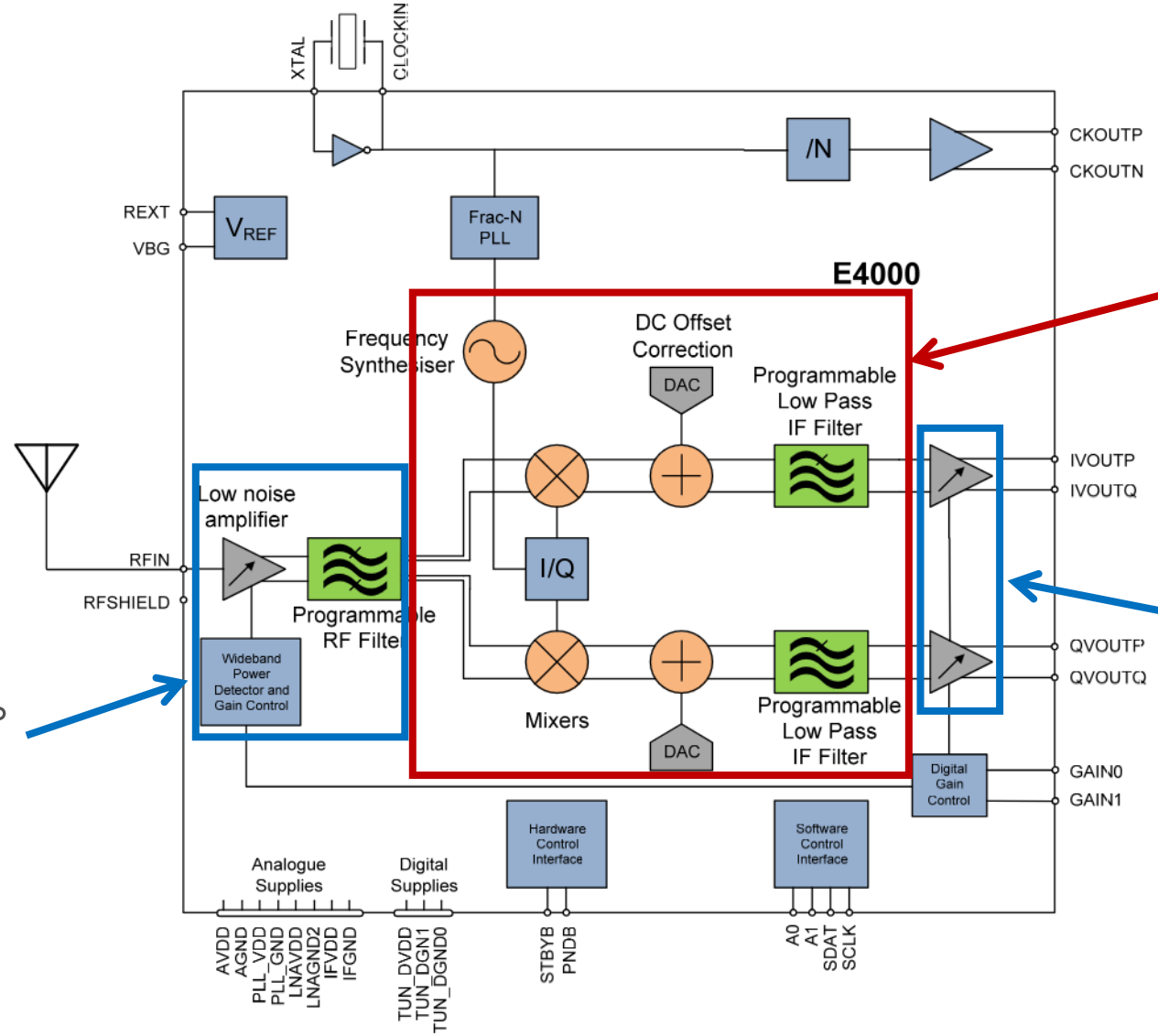
RTL2832U+FC0013

RF入力　　FC0013（RF tuner）　　RTL2832U（AD コンバータ、USB phy）



RFチューナーICの構成

今回のICとはちょっと異なりますが…



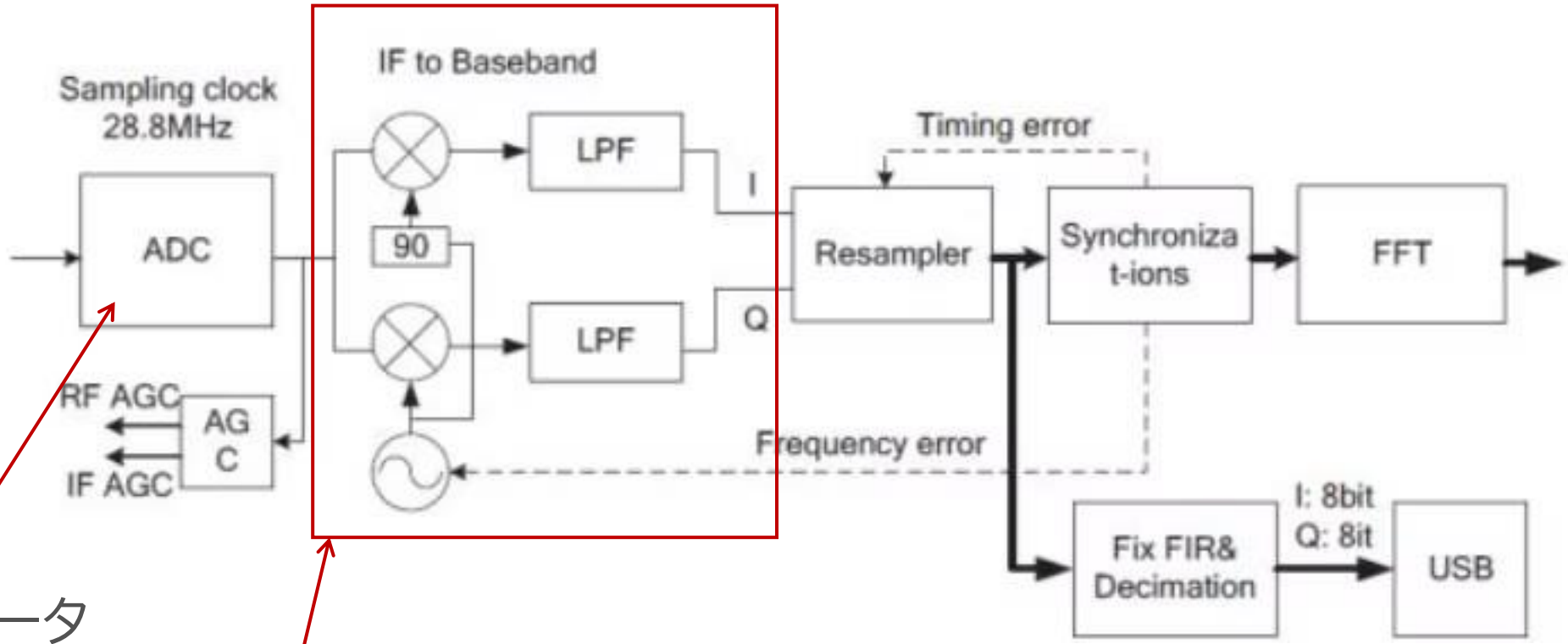
直交復調器

IFアンプ

RFアンプ

MULTI-STANDARD CMOS TERRESTRIAL RF TUNER, Elonics E4000 datasheets

RTL2832U



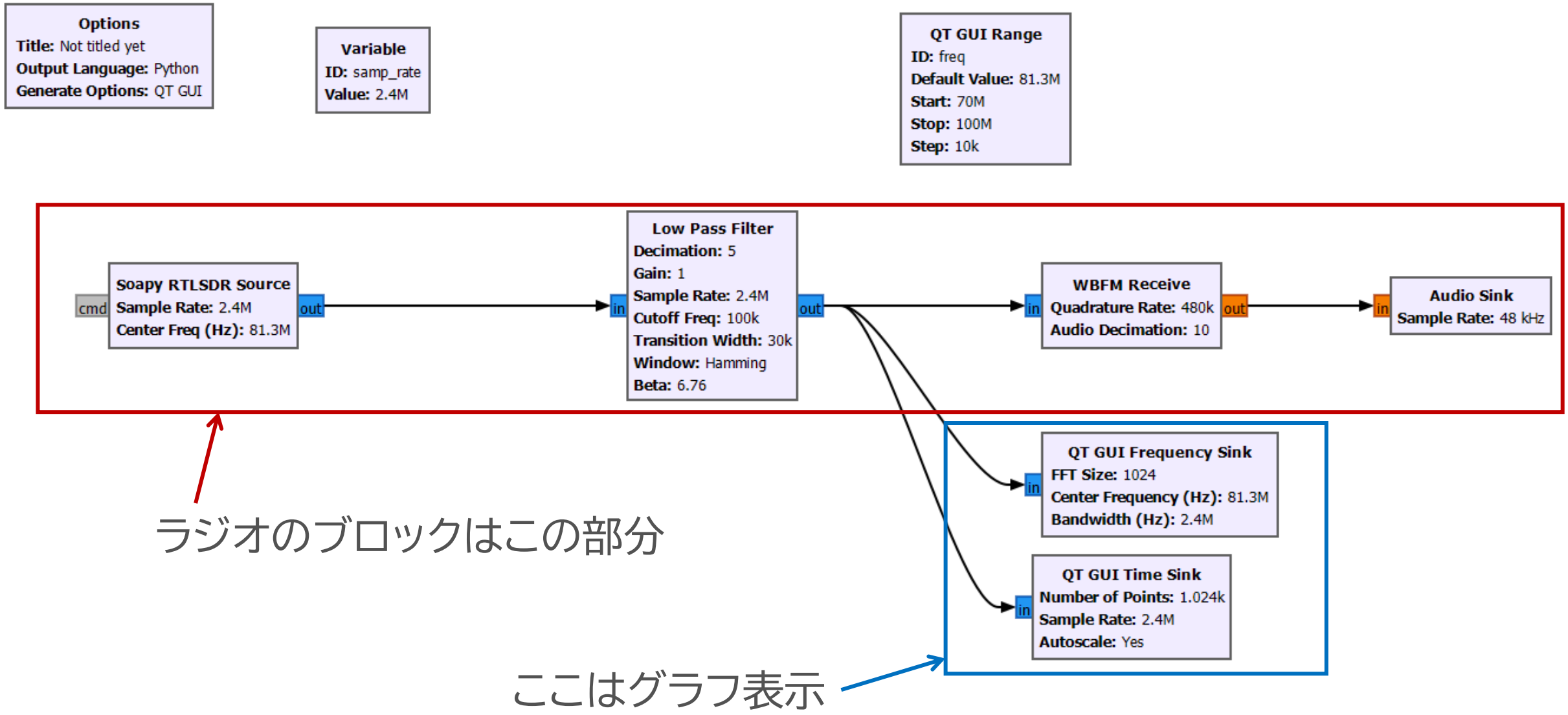
ADコンバータ

デジタル直交変調器

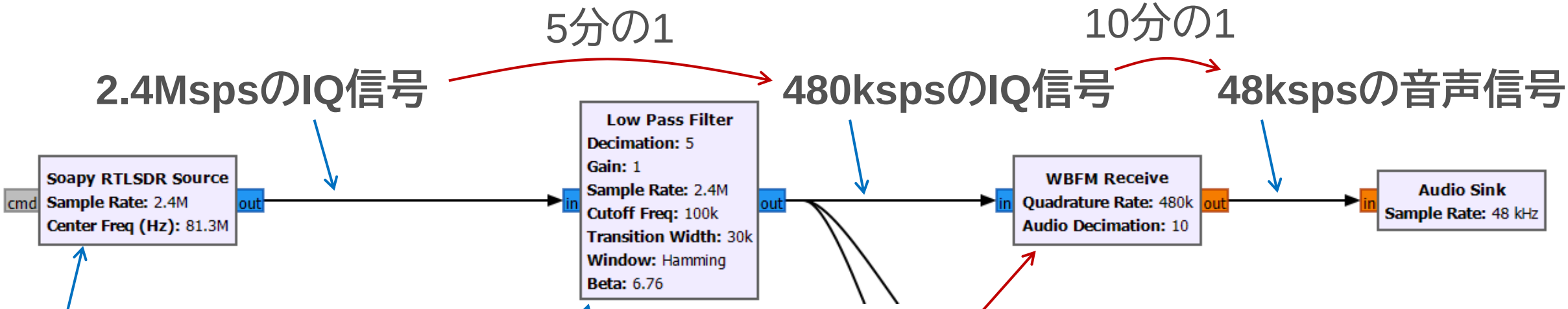
※実はこのチップ単体でもSDRとして振る舞う事ができる
元はISDB-T(テレビ放送)受信用のIC

FMラジオブロックを読み解く

前回のブロック図


Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI
Variable
ID: samp_rate
Value: 2.4M
QT GUI Range
ID: freq
Default Value: 81.3M
Start: 70M
Stop: 100M
Step: 10k
Soapy RTLSDR Source
Sample Rate: 2.4M
Center Freq (Hz): 81.3M
cmd
out
Low Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 2.4M
Cutoff Freq: 100k
Transition Width: 30k
Window: Hamming
Beta: 6.76
in
out
WBFM Receive
Quadrature Rate: 480k
Audio Decimation: 10
Audio Sink
Sample Rate: 48 kHz
QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 81.3M
Bandwidth (Hz): 2.4M
QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2.4M
Autoscale: Yes
ラジオのブロックはこの部分
ここはグラフ表示

ラジオ部分を読み解く



RTL2832Uから信号を取得
サンプリングレート：　2.4MHz
LOの中心周波数：　81.3MHz

FM受信処理
デシメーション：　10
サンプルレート：　480ksps

ローパスフィルタとデシメーション処理
デシメーション：　5
サンプルレート：　2.4Msps
カットオフ周波数：100kHz
遷移幅：　30kHz

次週はこの部分の自作を目指す
今週はAMを自作

AMラジオの実装

今回受信する放送局

RJTT ATIS

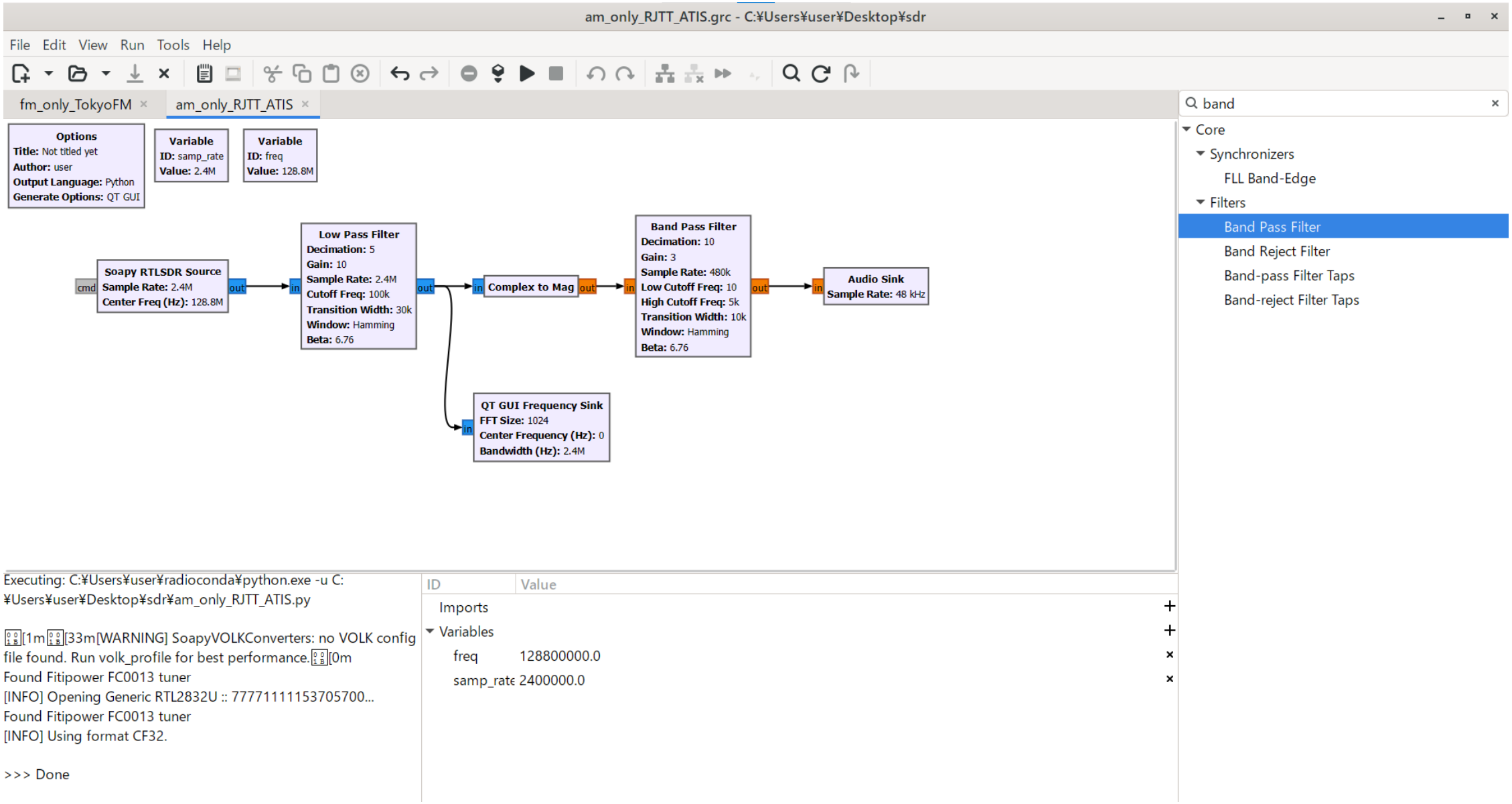
- 128.8MHzのAMで放送中
- RJTTは羽田空港の空港コード（ICAOコード）
- ATISはAutomated Terminal Information Serviceの略で空港周辺の気象情報や、飛行場の状態、航空保安施設の運用状況などを放送する仕組み
- 英語で放送されている

周辺の周波数ではグラウンドコントロール、タワー、アプローチなどの飛行機を誘導するための通信やカンパニー無線もあるので、余裕があれば聞いてみよう

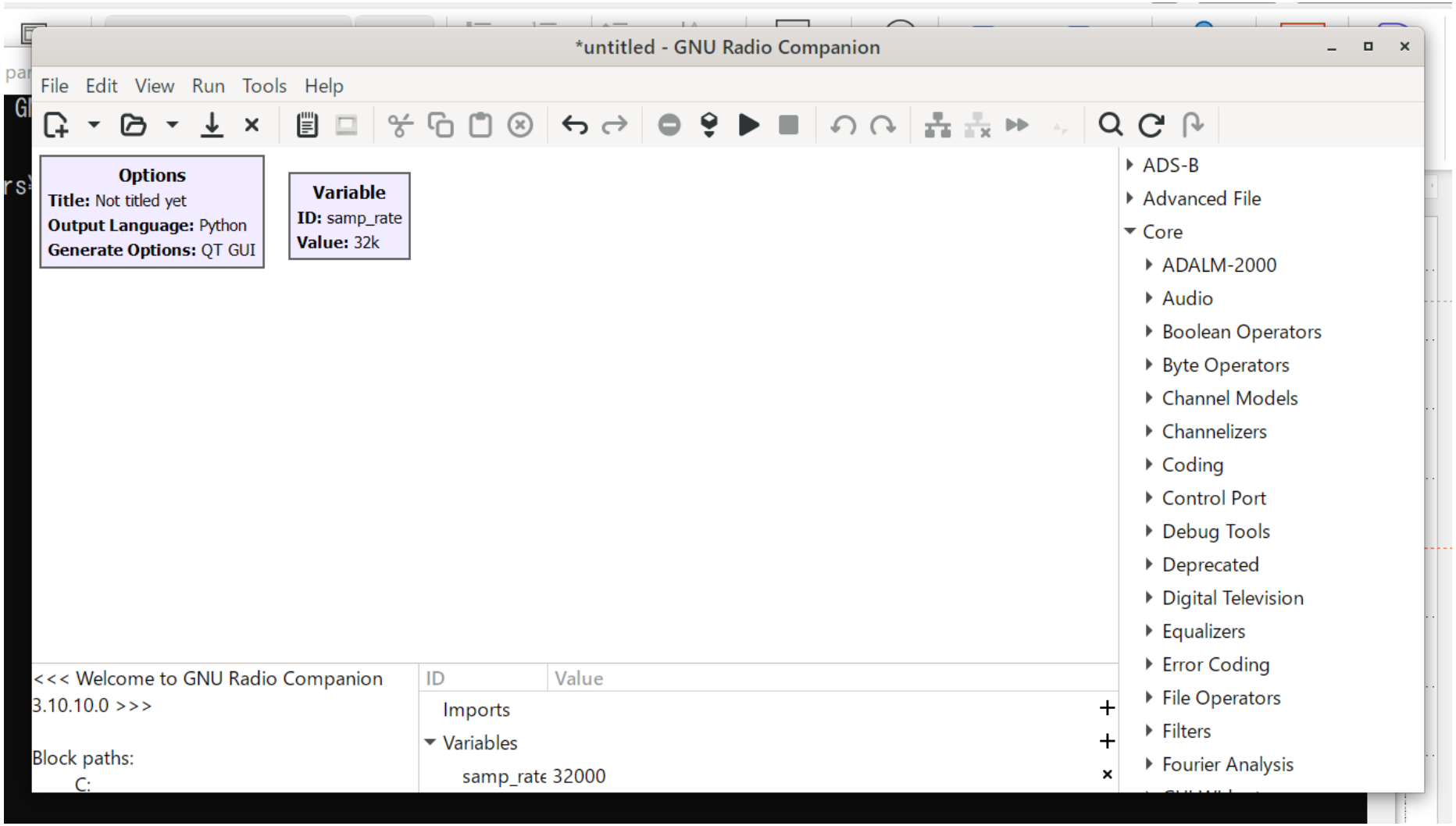
通信を傍受する際には関係法令に抵触しないように気をつける。

- 電波法第59条
- 電気通信事業法第4条1項

今回作成するブロックダイアグラム

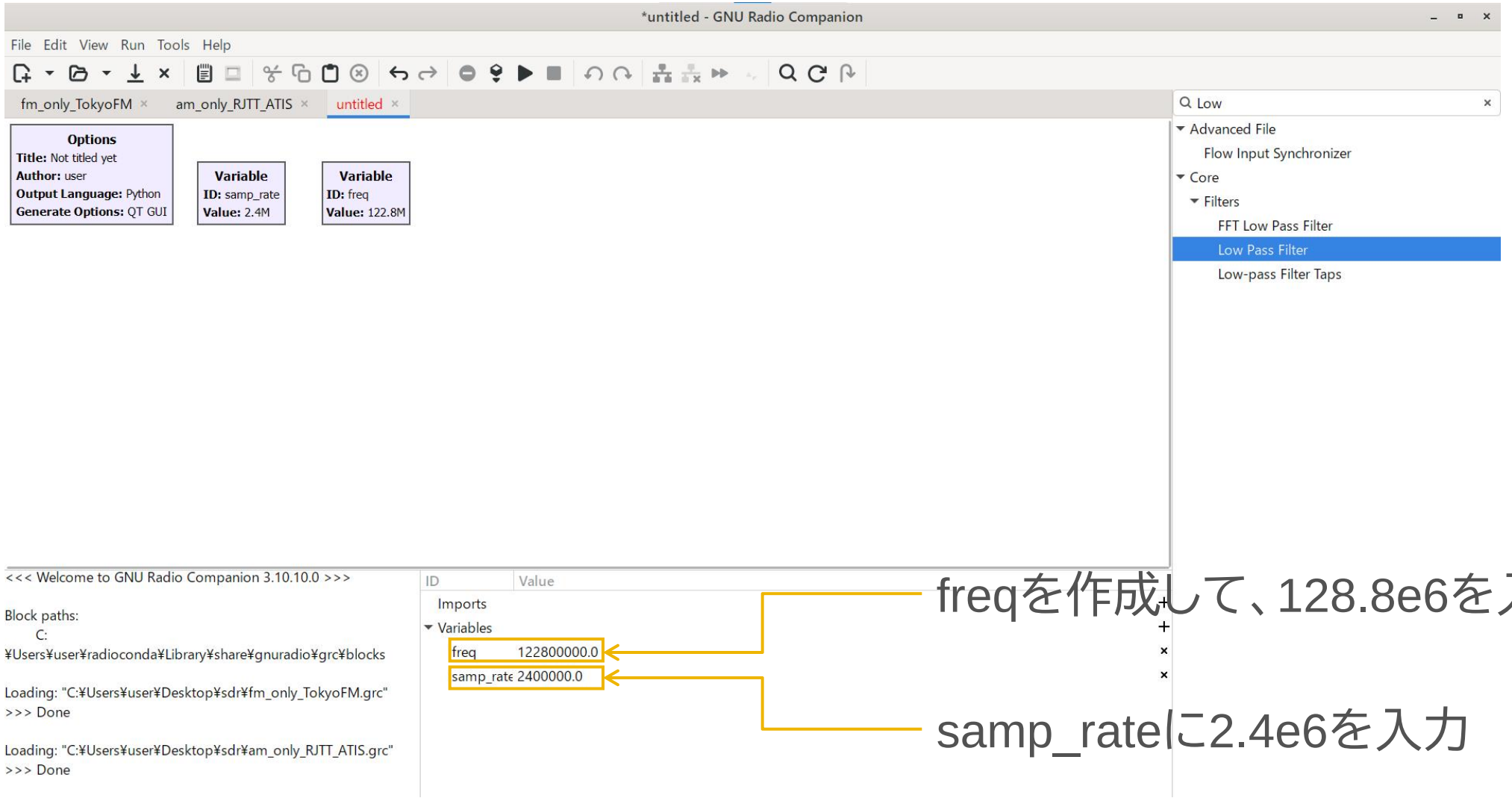


GNU Radioを起動



起動画面はこんな感じ

変数の追加と設定



freqを作成して、128.8e6を入力

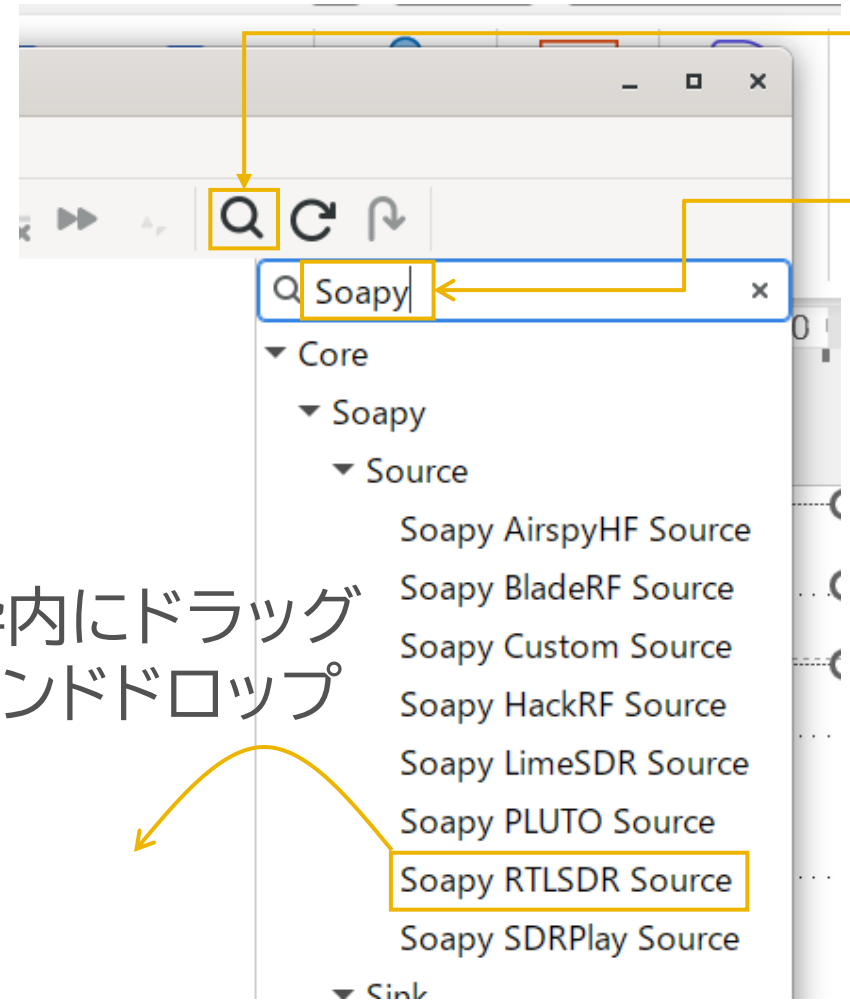
samp_rateに2.4e6を入力

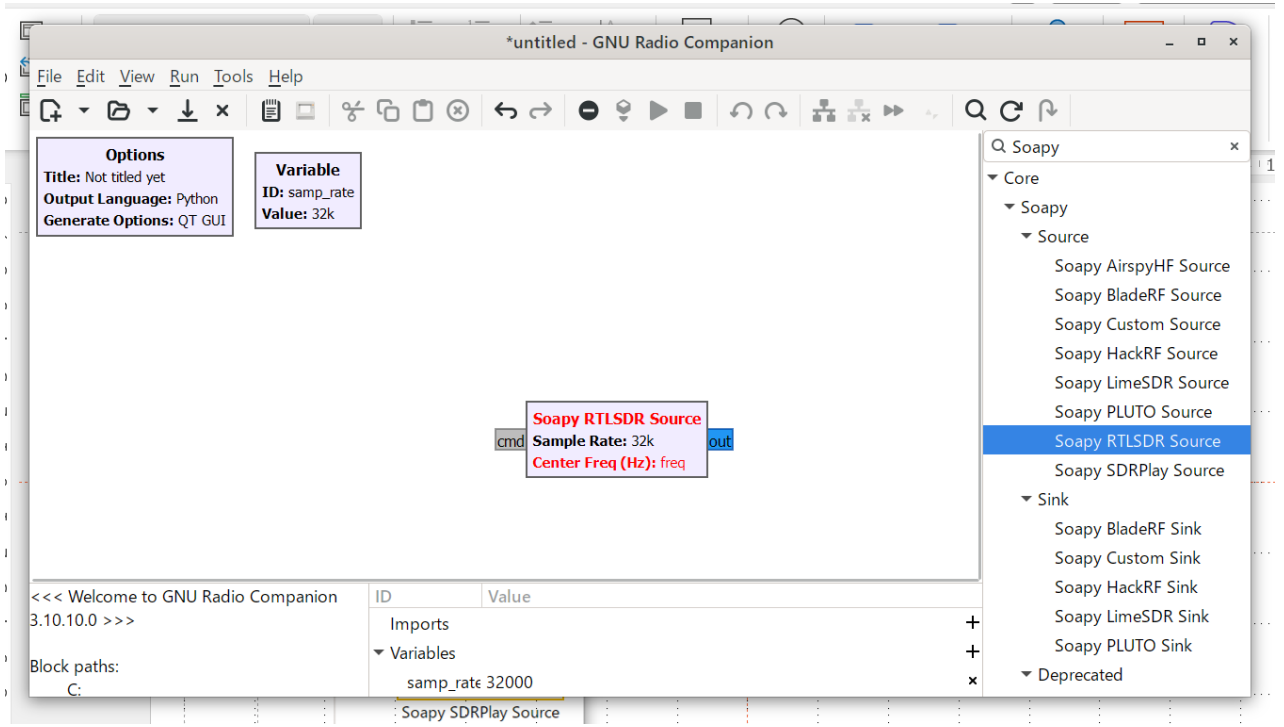
RTL SDRを追加

虫眼鏡(検索)を左クリック

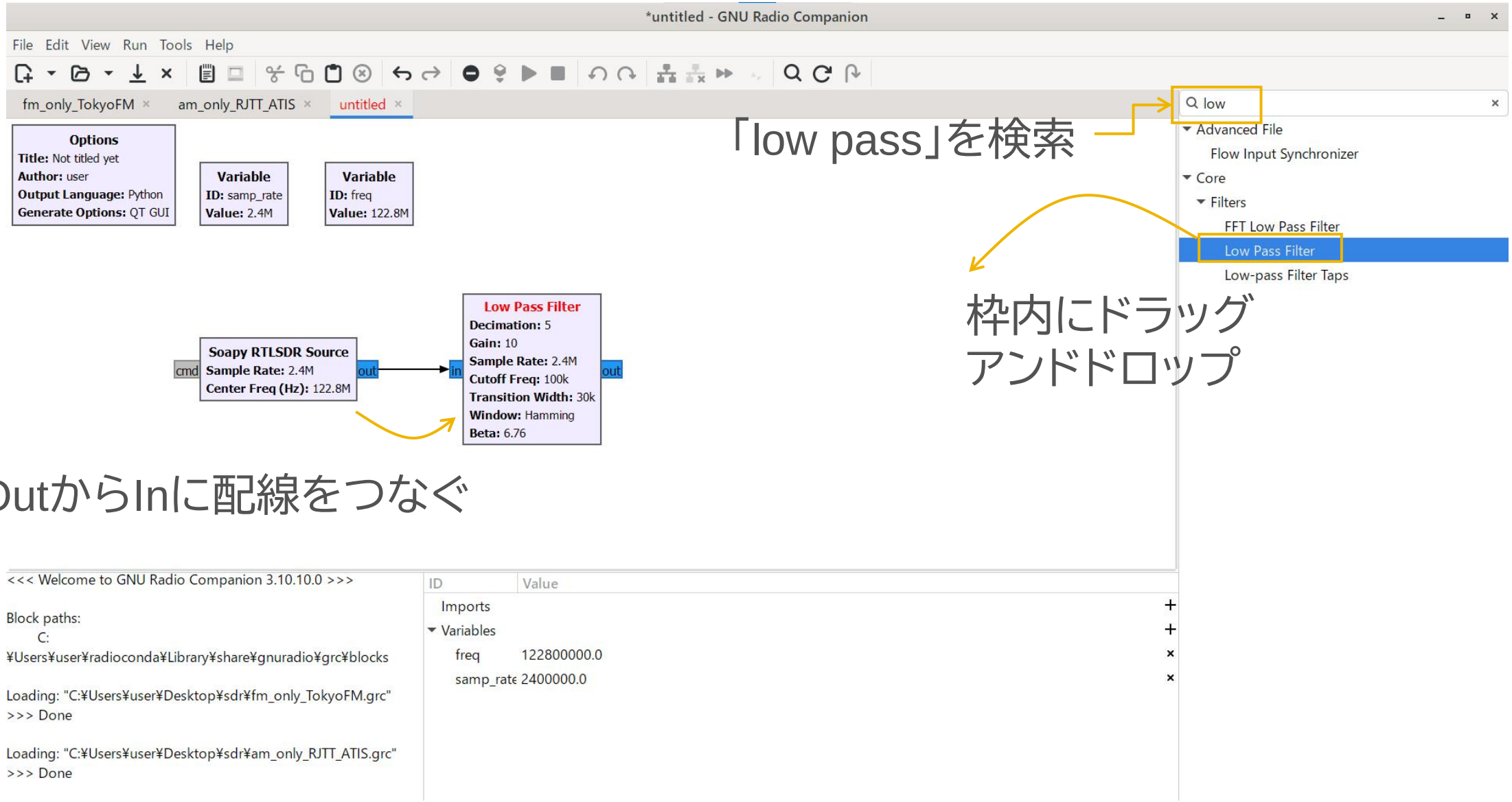
Soapyを入力

枠内にドラッグ
アンドドロップ





Low pass filterを追加



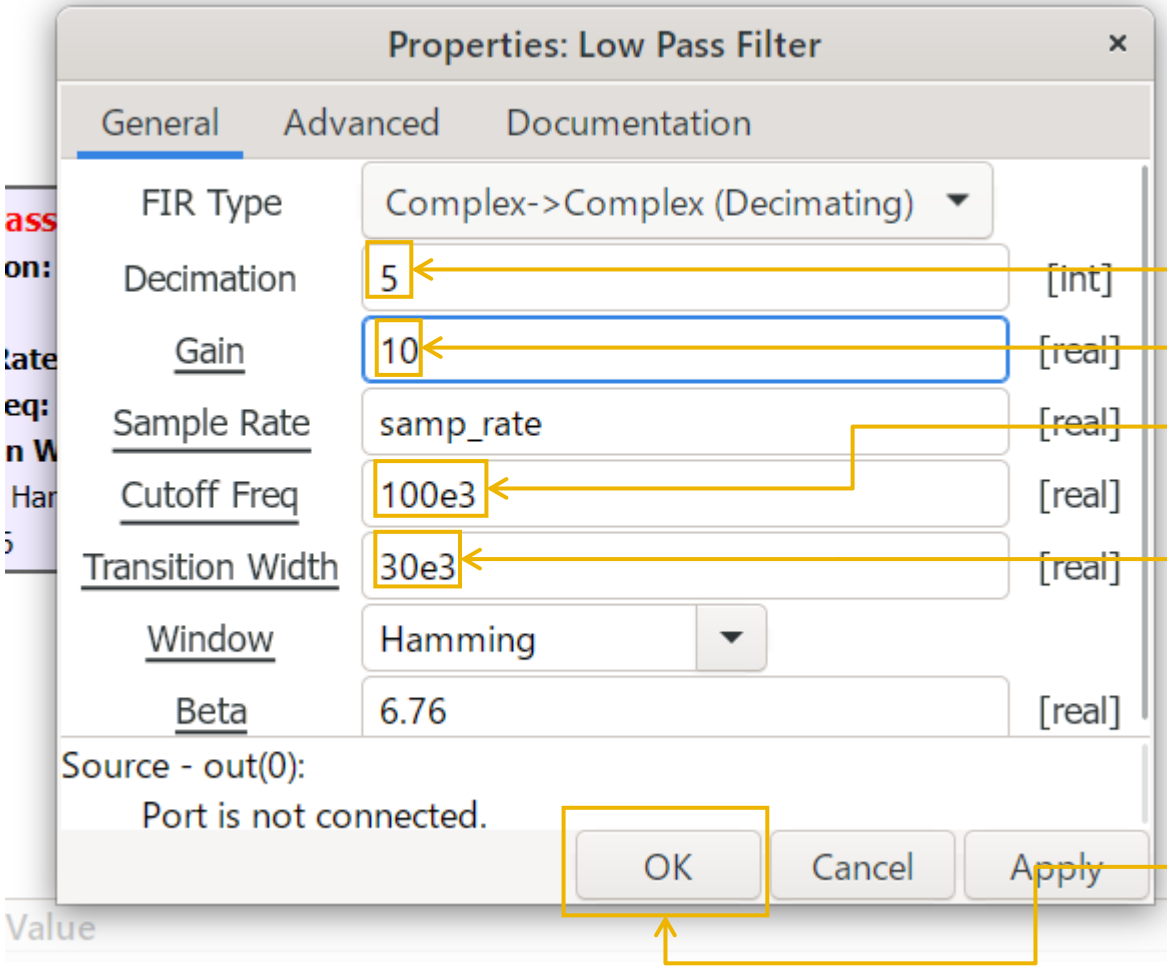
「low pass」を検索

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

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

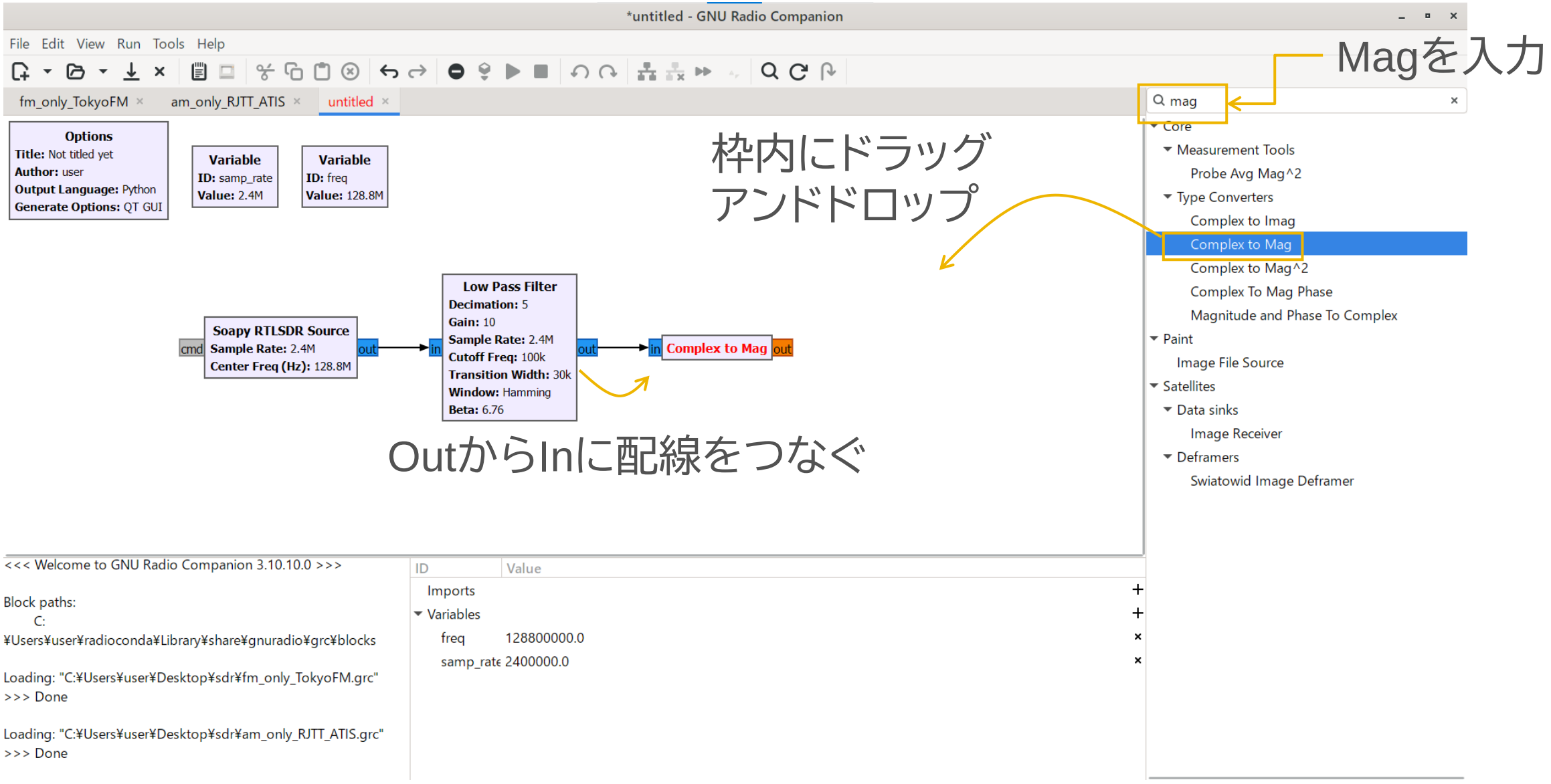
10を入力

100e3を入力

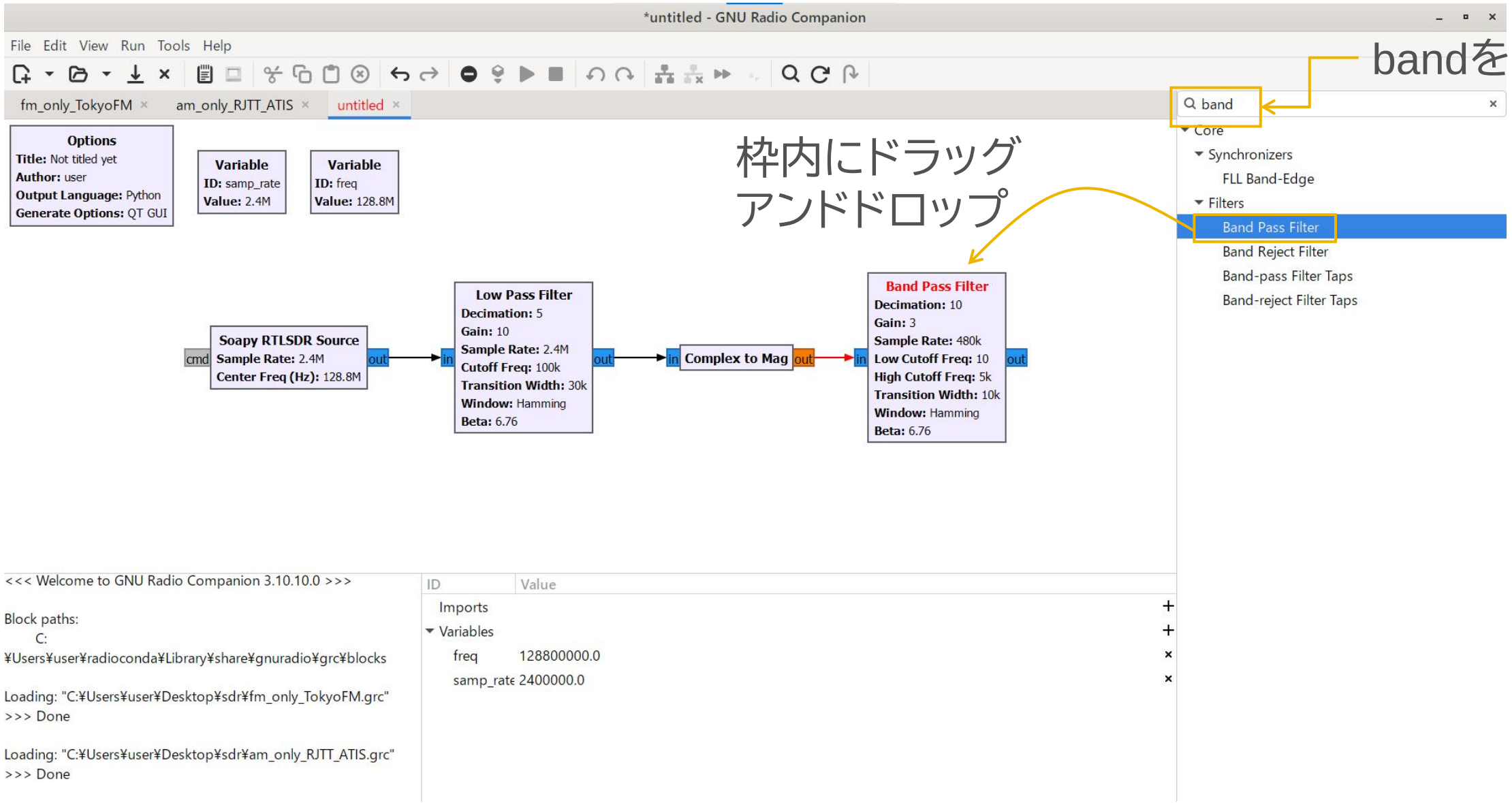
30e3を入力

入力が終わったらOKをクリック

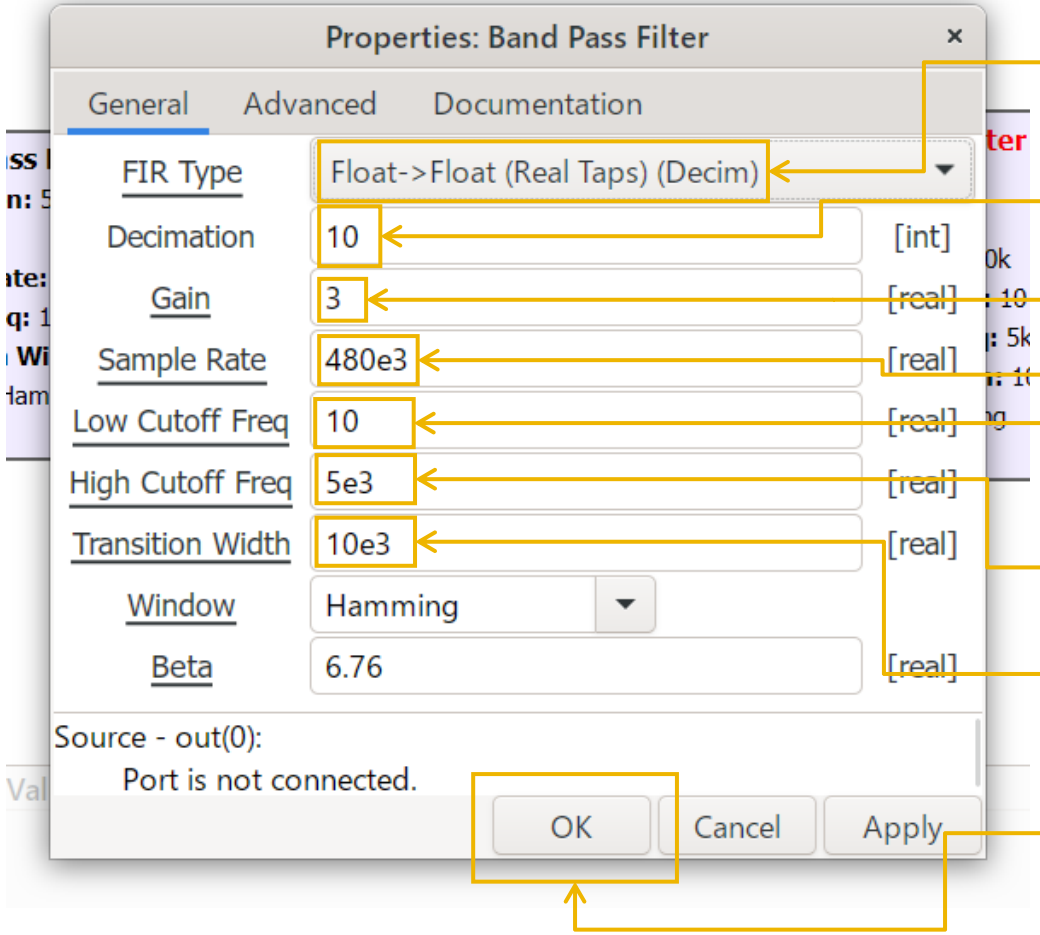
Complex to magの追加



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

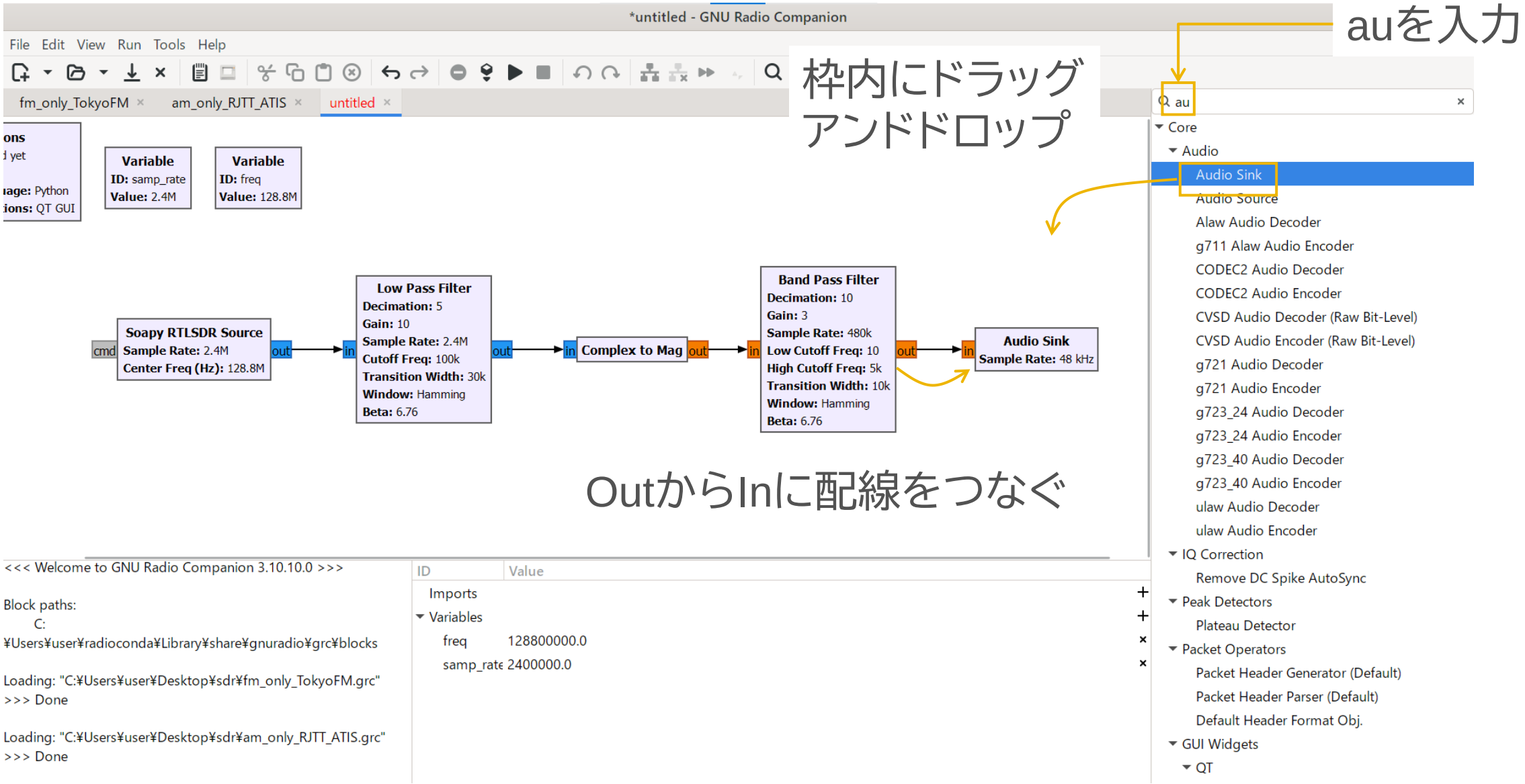
10を入力

5e3を入力

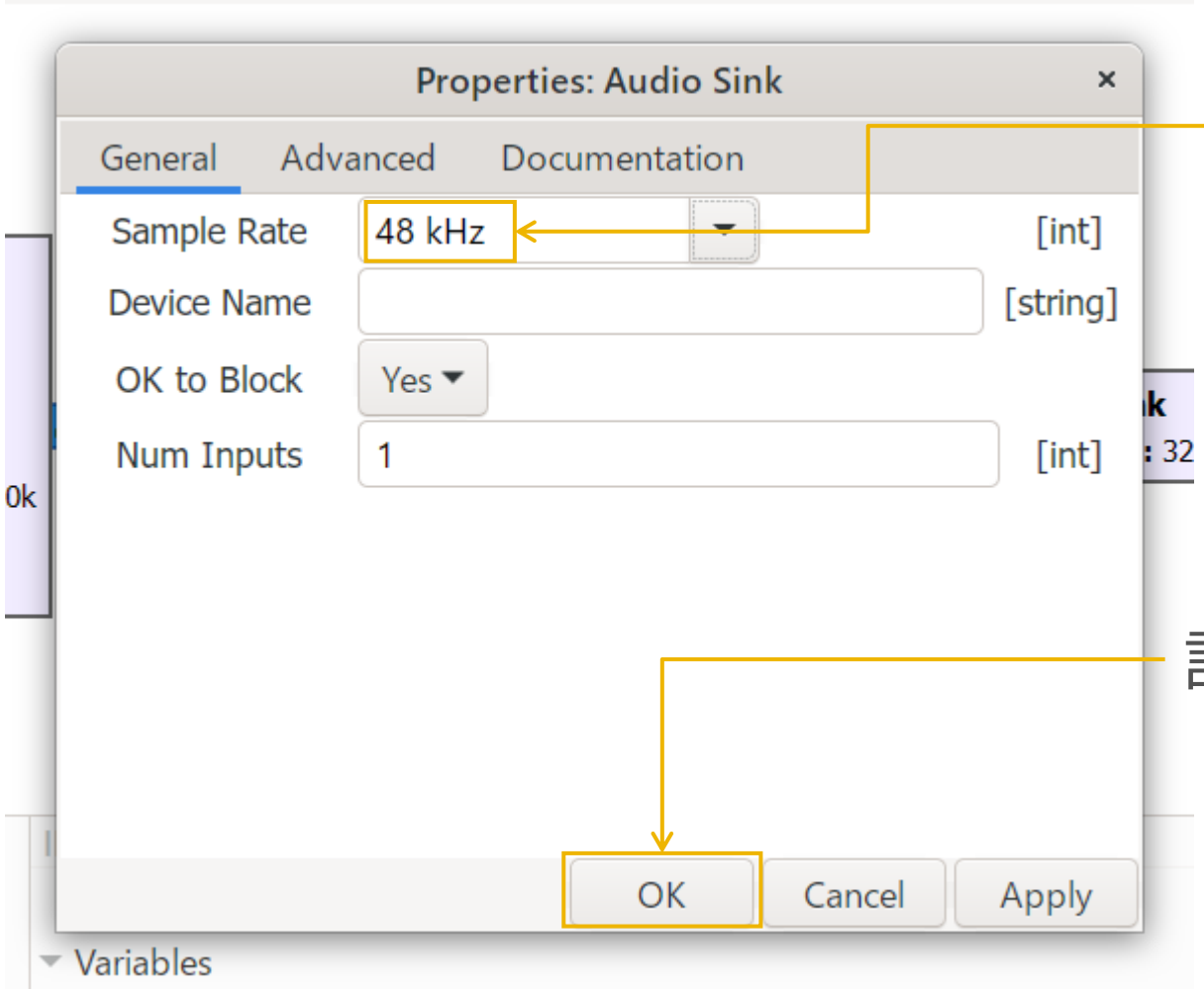
10e3を入力

入力が終わったらOKをクリック

Audio sinkの追加



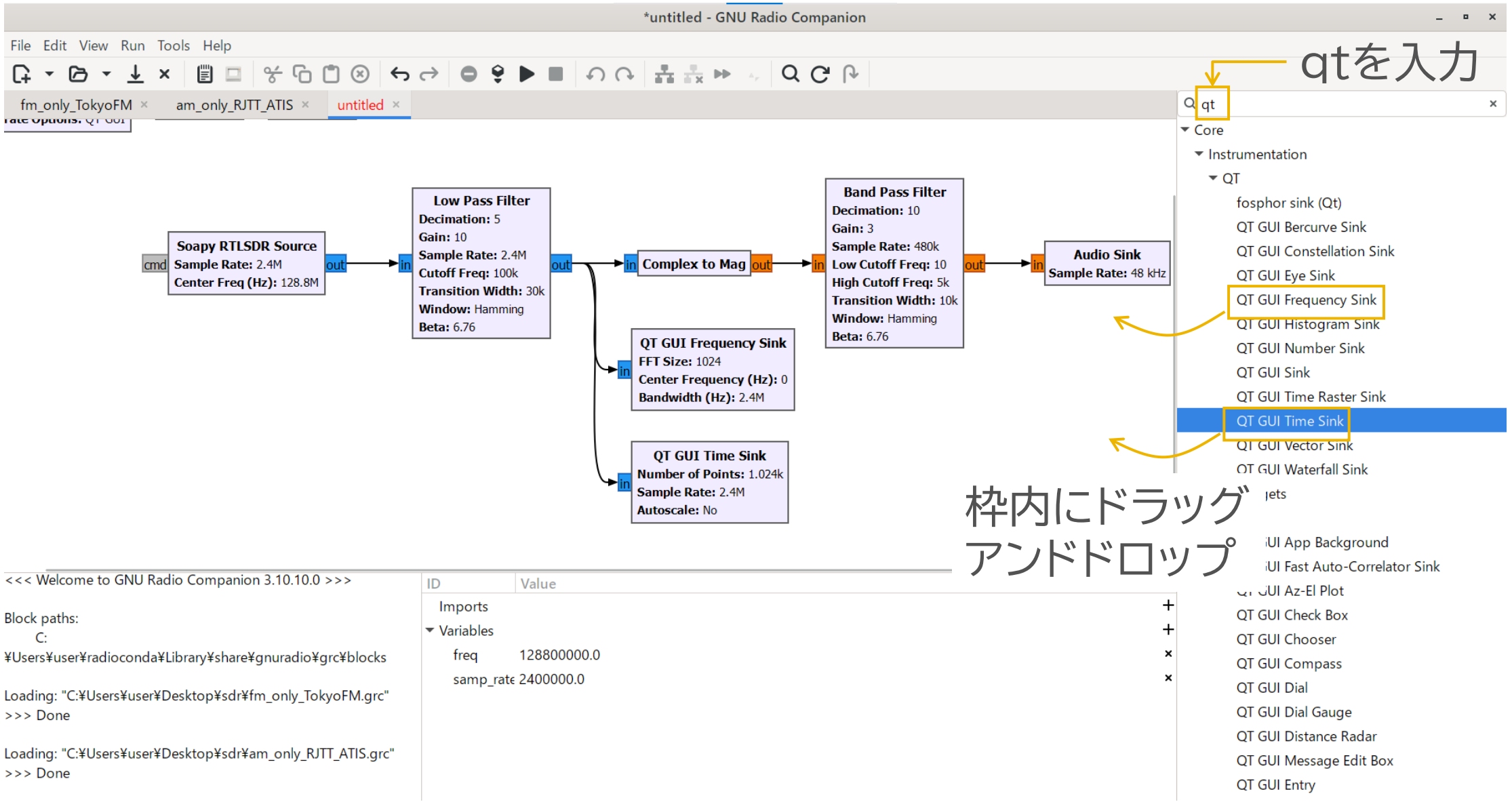
Audio Sinkの設定



プルダウンメニューから「48kHz」を選択

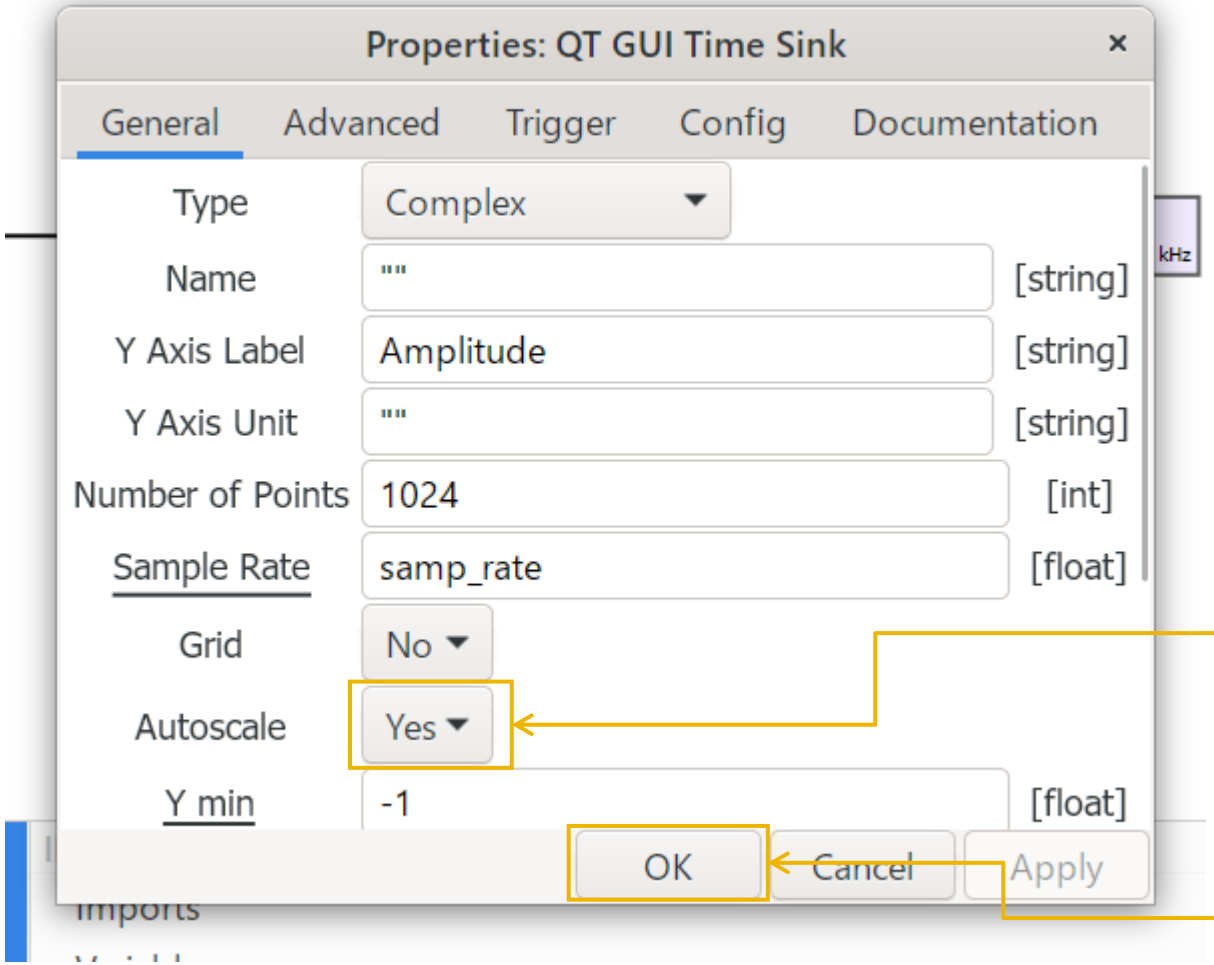
設定したらOKをクリック

グラフの追加


*untitled - GNU Radio Companion
File Edit View Run Tools Help
fm_only_TokyoFM
am_only_RJTT_ATIS
untitled
Soapy RTLSDR Source
Sample Rate: 2.4M
Center Freq (Hz): 128.8M
Low Pass Filter
Decimation: 5
Gain: 10
Sample Rate: 2.4M
Cutoff Freq: 100k
Transition Width: 30k
Window: Hamming
Beta: 6.76
Complex to Mag
Band Pass Filter
Decimation: 10
Gain: 3
Sample Rate: 480k
Low Cutoff Freq: 10
High Cutoff Freq: 5k
Transition Width: 10k
Window: Hamming
Beta: 6.76
Audio Sink
Sample Rate: 48 kHz
QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 2.4M
QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2.4M
Autoscale: No
qtを入力
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
枠内にドラッグ
アンドドロップ
<<< Welcome to GNU Radio Companion 3.10.10.0 >>>
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 128800000.0
samp_rate 2400000.0
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

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

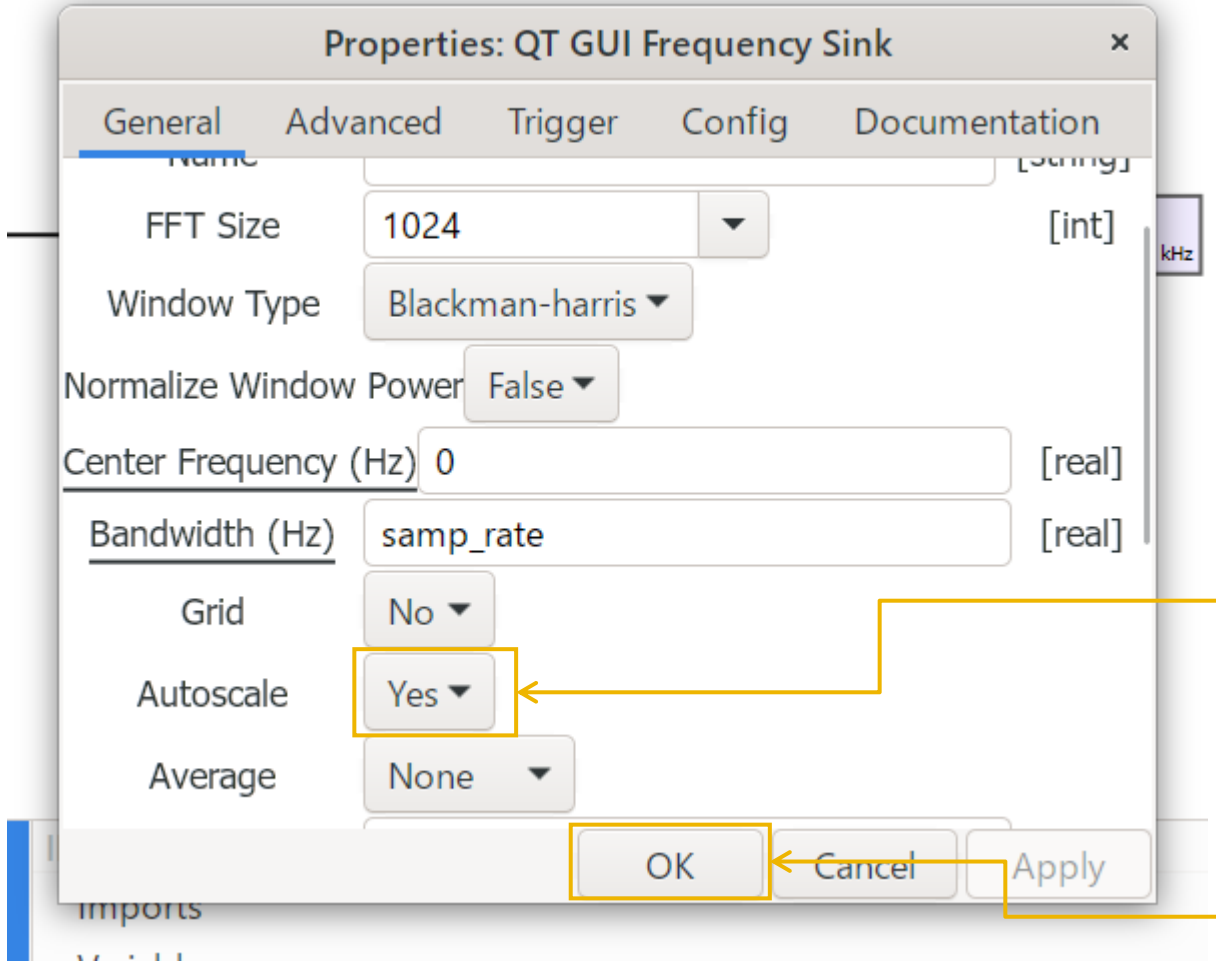


AutoscaleをYesに変更する

設定したらOKをクリック

QT GUI Frequency sinkの設定

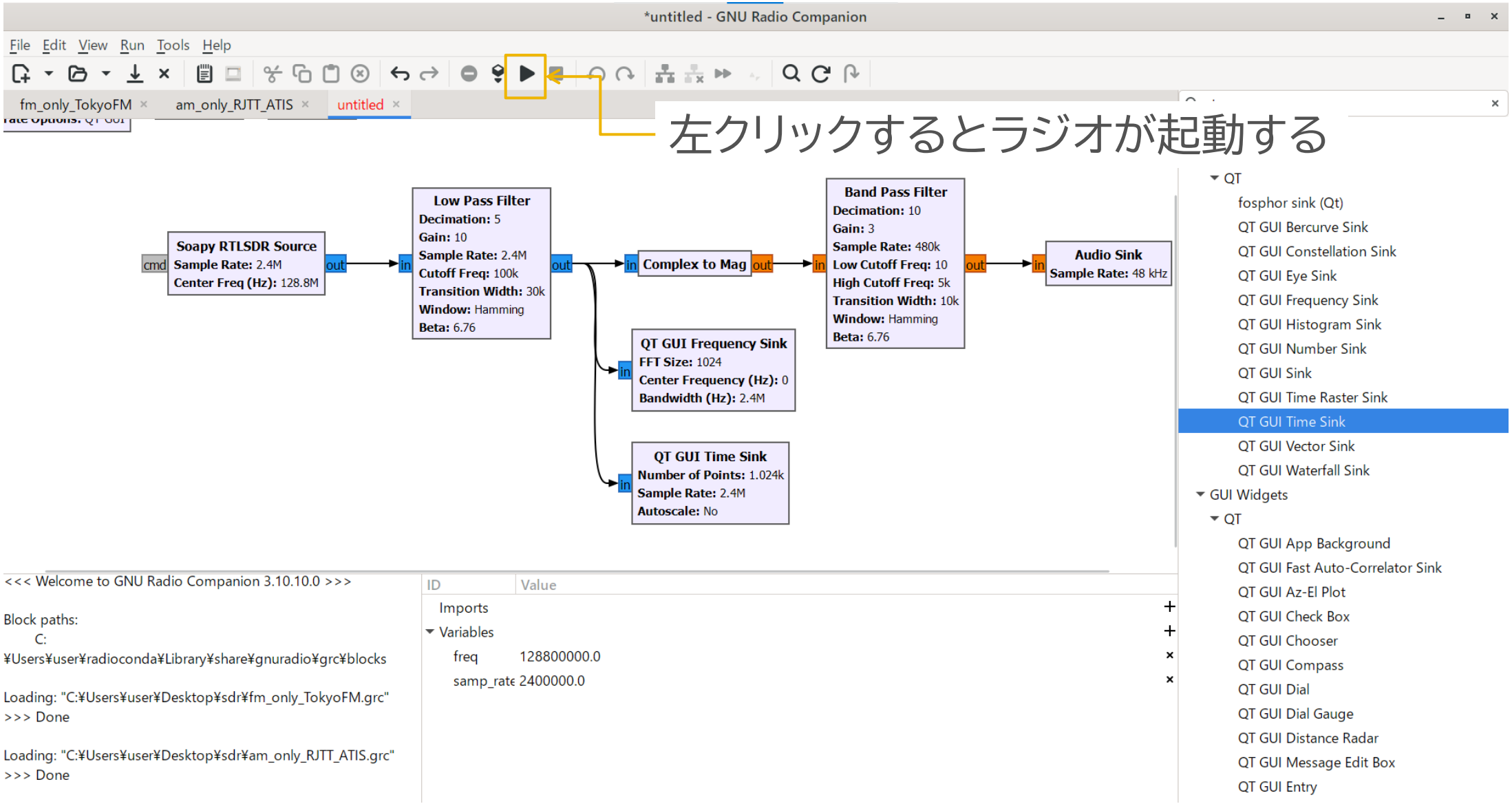
QT GUI Frequency sinkブロックをダブルクリックして設定を開く



AutoscaleをYesに変更する

設定したらOKをクリック

ラジオの起動


*untitled - GNU Radio Companion
File Edit View Run Tools Help
fm_only_TokyoFM
am_only_RJTT_ATIS
untitled
左クリックするとラジオが起動する
Soapy RTLSDR Source
Sample Rate: 2.4M
Center Freq (Hz): 128.8M
Low Pass Filter
Decimation: 5
Gain: 10
Sample Rate: 2.4M
Cutoff Freq: 100k
Transition Width: 30k
Window: Hamming
Beta: 6.76
Complex to Mag
Band Pass Filter
Decimation: 10
Gain: 3
Sample Rate: 480k
Low Cutoff Freq: 10
High Cutoff Freq: 5k
Transition Width: 10k
Window: Hamming
Beta: 6.76
Audio Sink
Sample Rate: 48 kHz
QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 2.4M
QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2.4M
Autoscale: No
<<< Welcome to GNU Radio Companion 3.10.10.0 >>>
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 128800000.0
samp_rate 2400000.0
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 Sink
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