

ソフトウェア無線機

第2回目

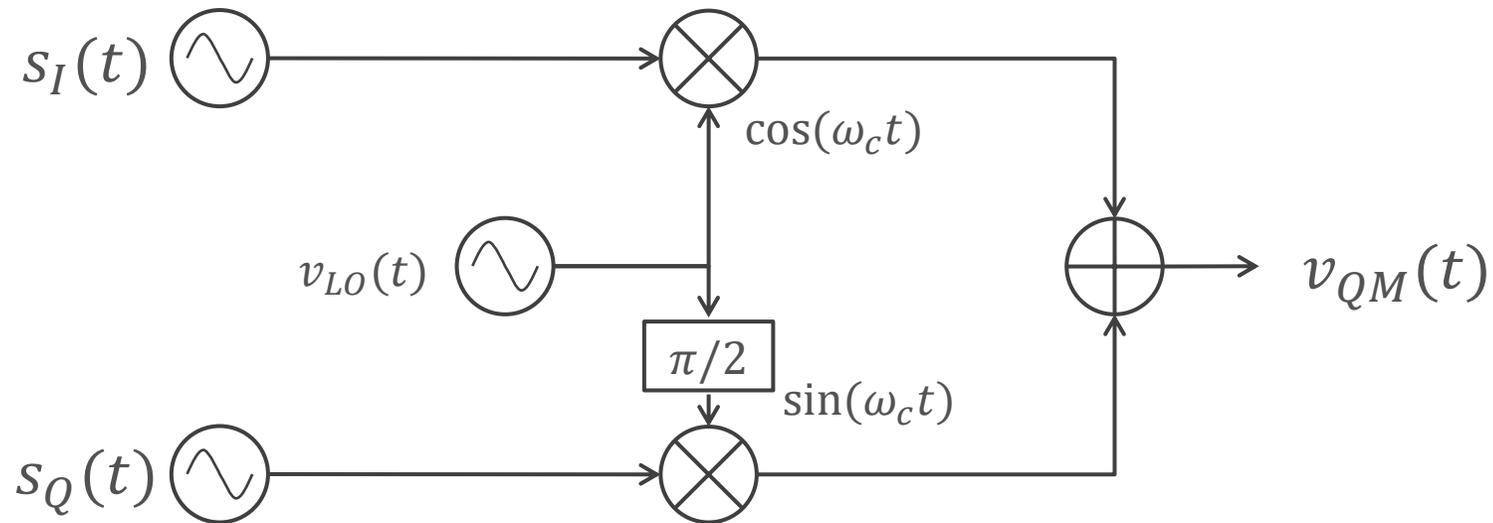
電子磁気応用資料

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

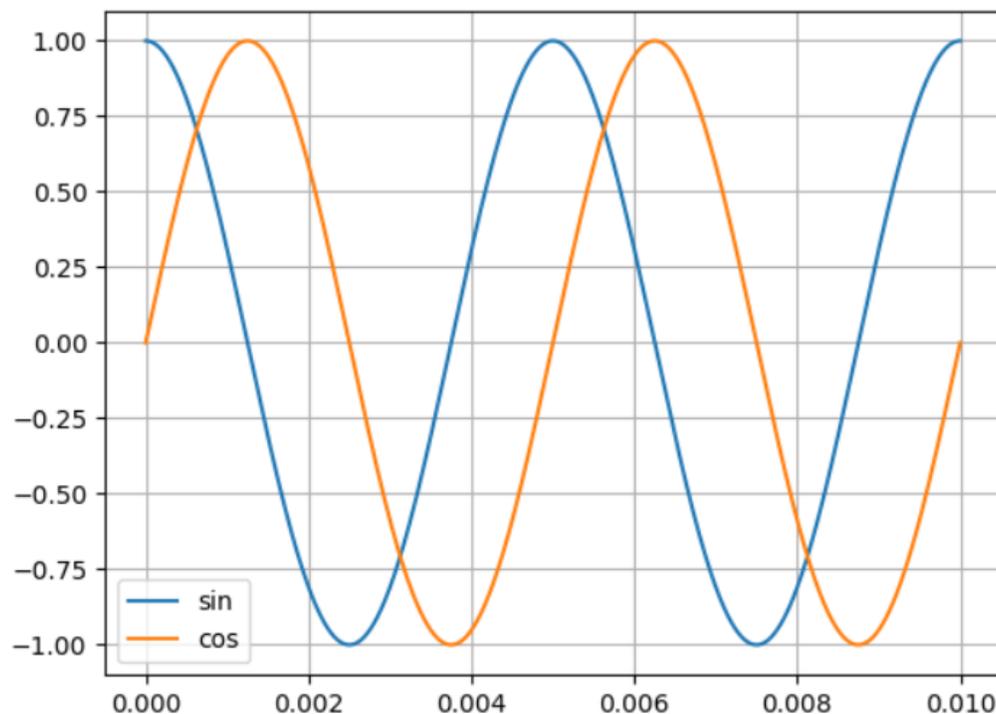
直交変調器による変調

直交変調器(IQ変調器)



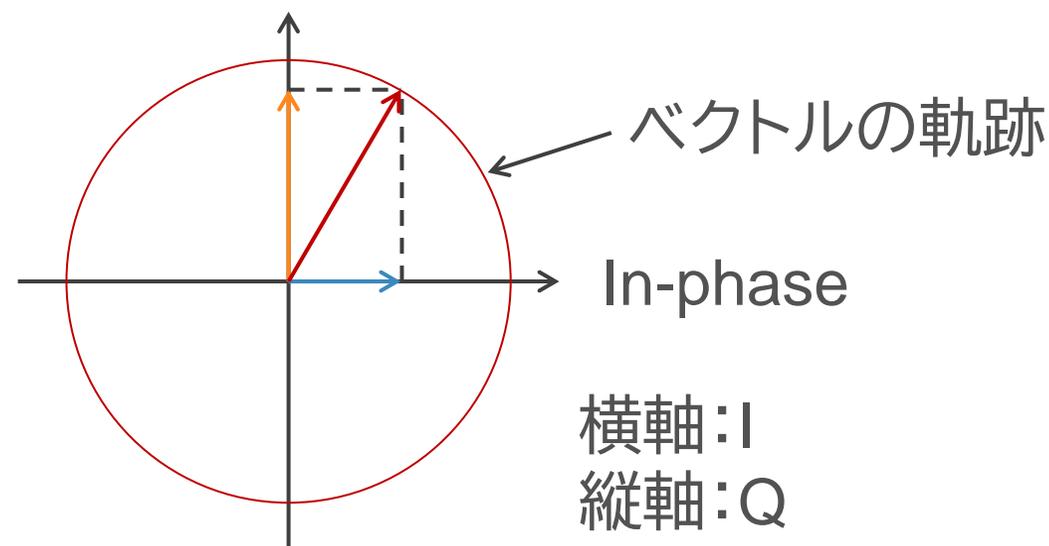
$$v_{QM}(t) = s_I(t) \cos(\omega_c t) + s_Q(t) \sin(\omega_c t)$$

sinとcosとIとQの関係



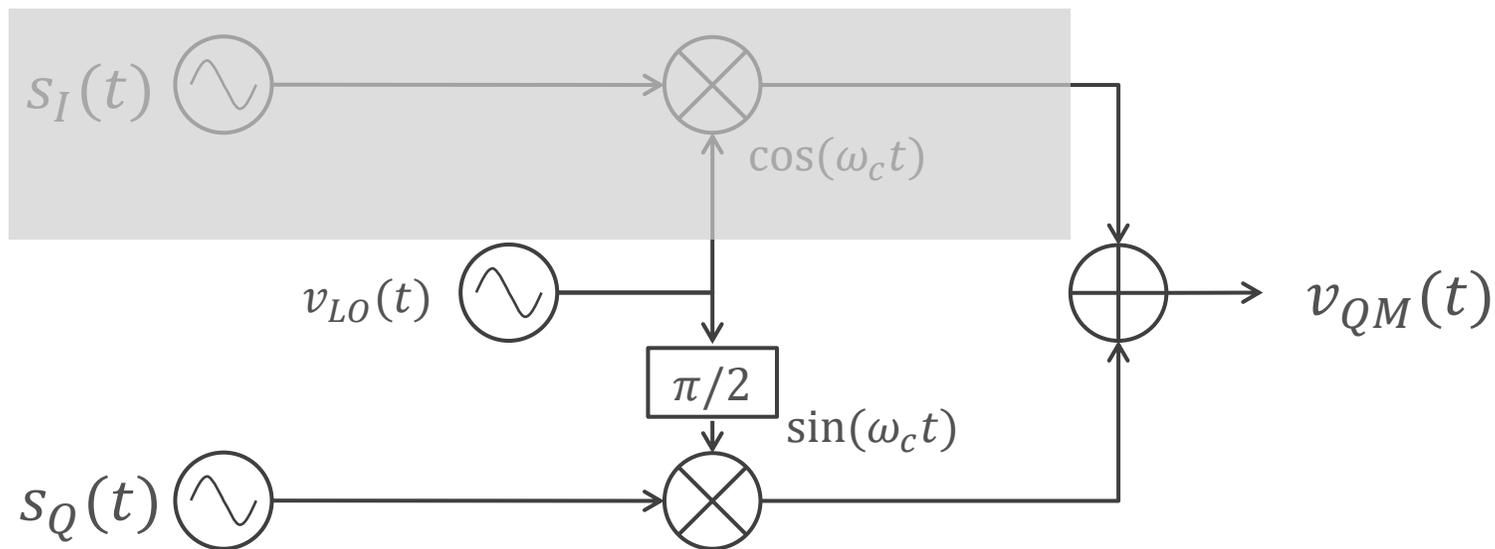
sinとcosは直交関係にある(内積がゼロ)

Quadrature-phase



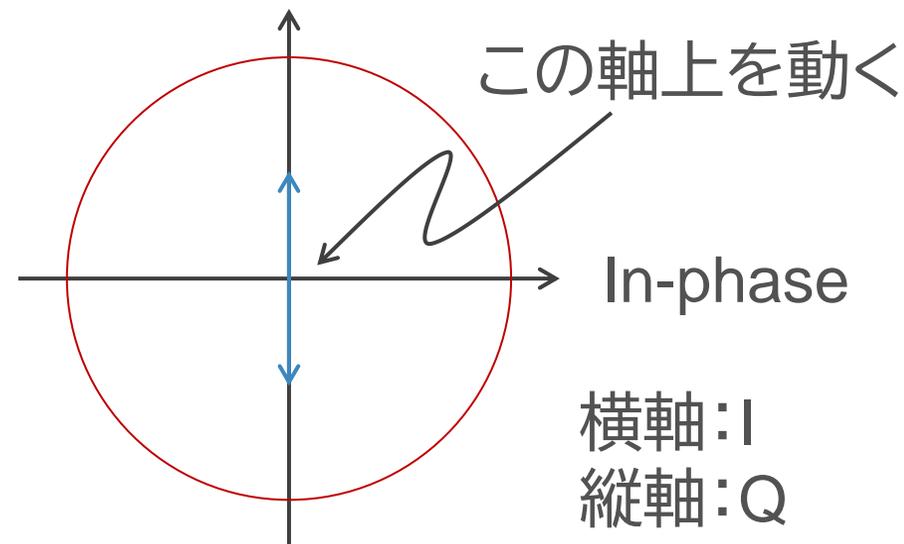
Iに対応するcosとQに対応するsinの波形を合わせることで、任意の振幅、位相の波形を作り出すことができる。

直交変調器を用いたAM



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

Quadrature-phase



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) ← キャリアの信号
vcq = np.sin(2*fc*np.pi*t) ← 90度シフトしたキャリアの信号

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

vam = vs*vcq ← 従来のAM
vqm = 0*vci + vs*vcq ← 直交変調器の処理

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.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)

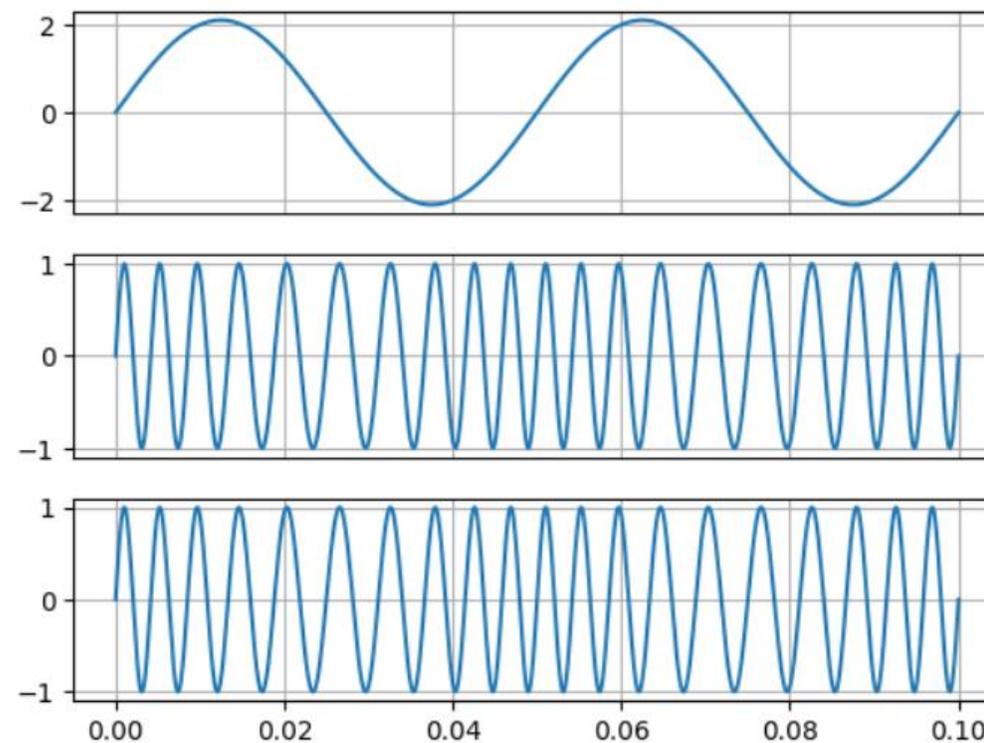
delta_phi = 120/180*np.pi
fs = 20
vs = delta_phi * np.sin(2*fs*np.pi*t)

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

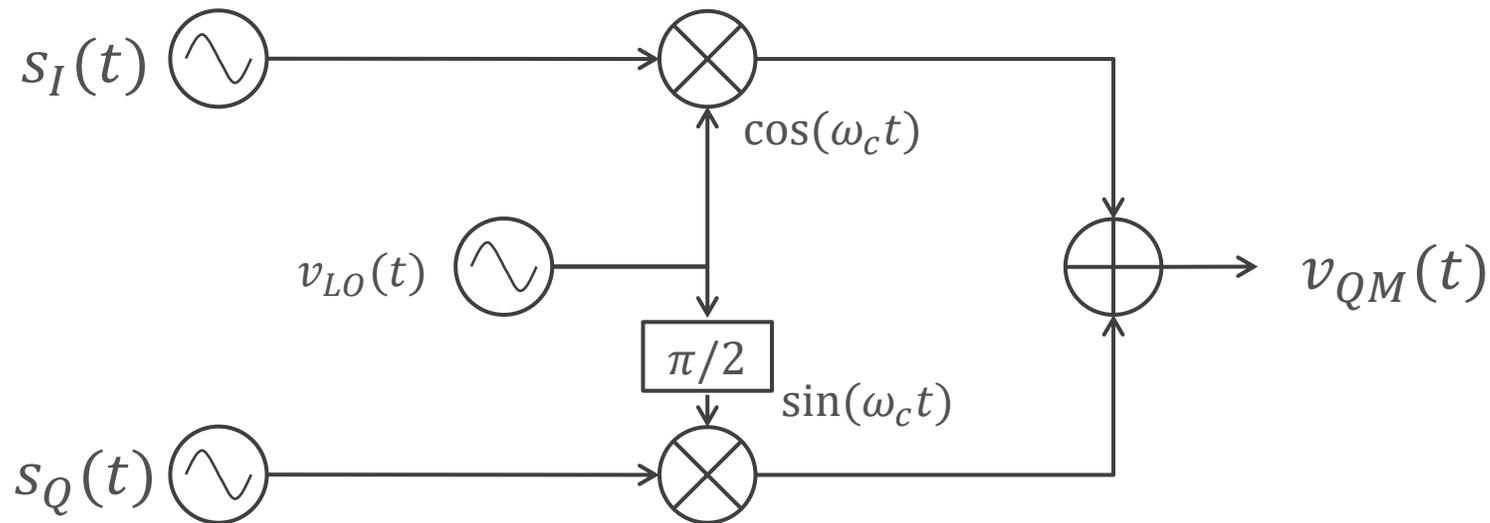
vpm = np.sin(2*fc*np.pi*t + vs)
vqm = vci*vsq + vcq*vsi

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)
```



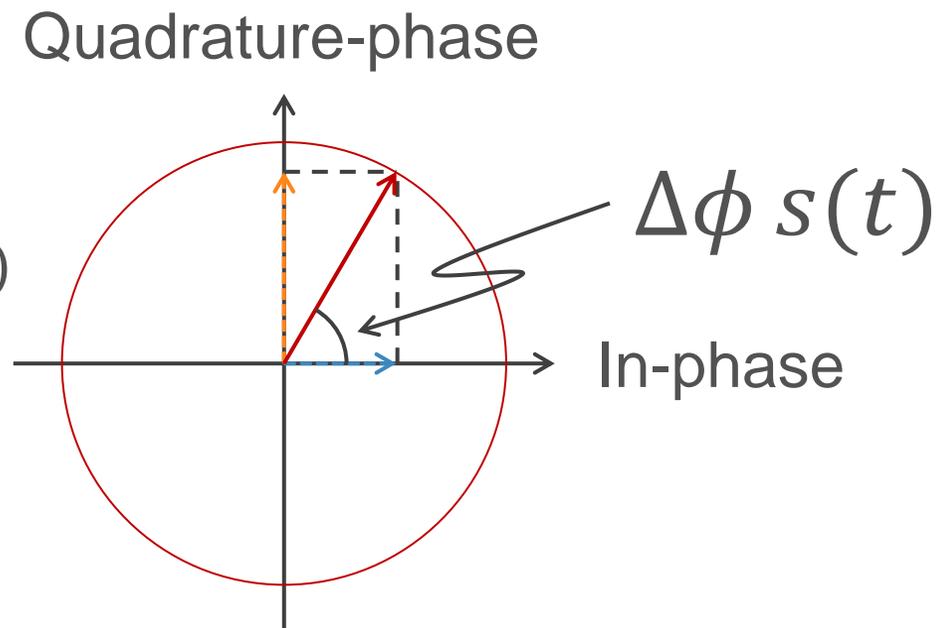
直交変調器を用いたPM



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

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

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



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) ← キャリアの信号
vcq = np.sin(2*fc*np.pi*t) ← 90度シフトしたキャリアの信号
```

```
delta_phi = 120/180*np.pi
fs = 20
vs = delta_phi * np.sin(2*fs*np.pi*t) } PMの信号波形
```

```
vsi = np.cos(vs) ← 入力の処理
vsq = np.sin(vs) ← Q入力の処理
```

```
vpm = np.sin(2*fc*np.pi*t + vs) ← 従来のPM
vqm = vci*vsq + vcq*vsi ← 直交変調器の処理
```

```
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)
```

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)

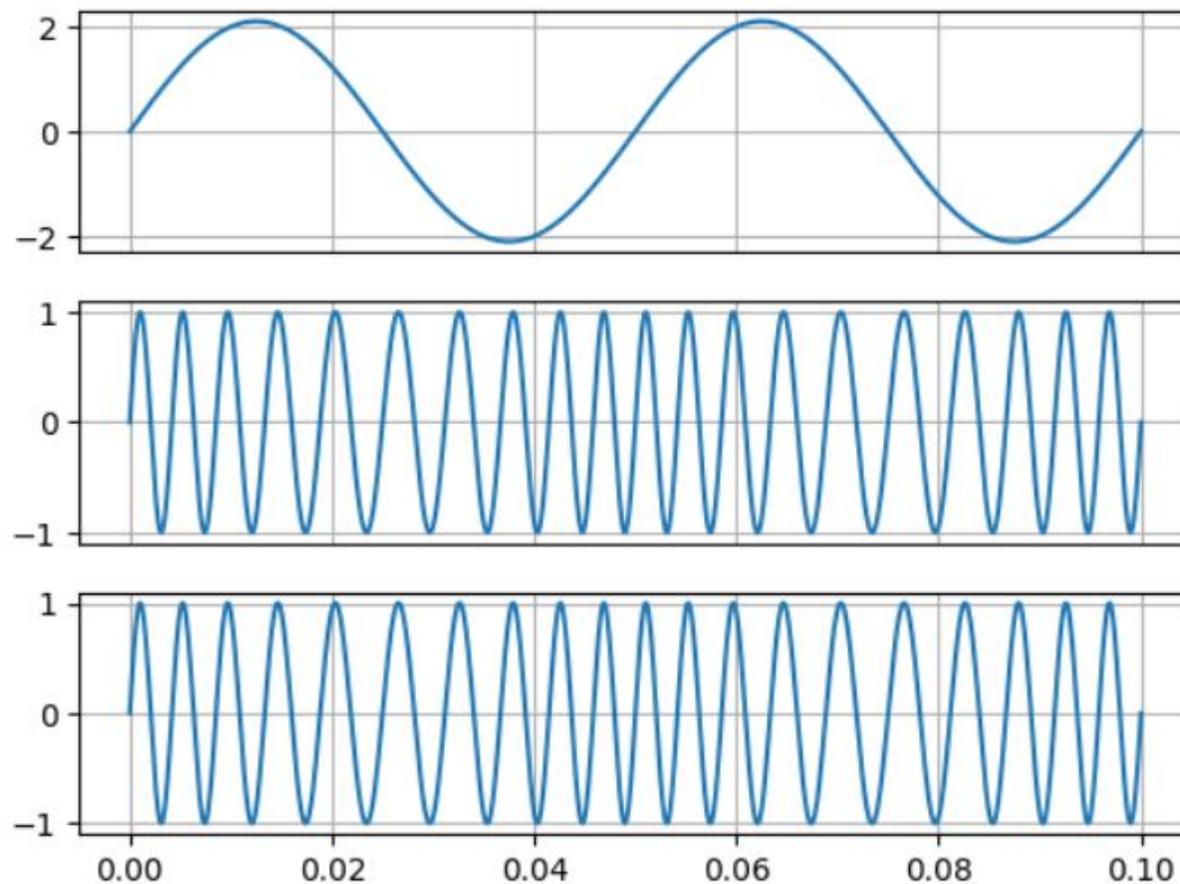
delta_phi = 120/180*np.pi
fs = 20
vs = delta_phi * np.sin(2*fs*np.pi*t)

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

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

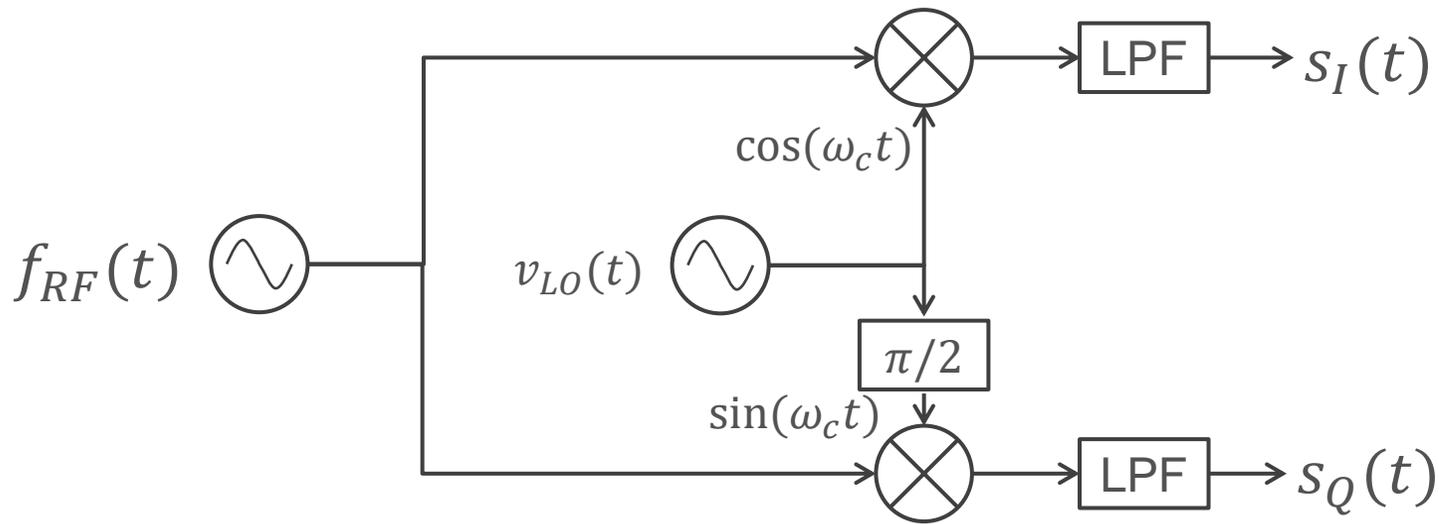
fig = plt.figure()
ax = fig.add_subplot(311)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(312)
ay.grid()
ay.plot(t, vpm)
az = fig.add_subplot(313)
az.grid()
az.plot(t, vqm)

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)
```

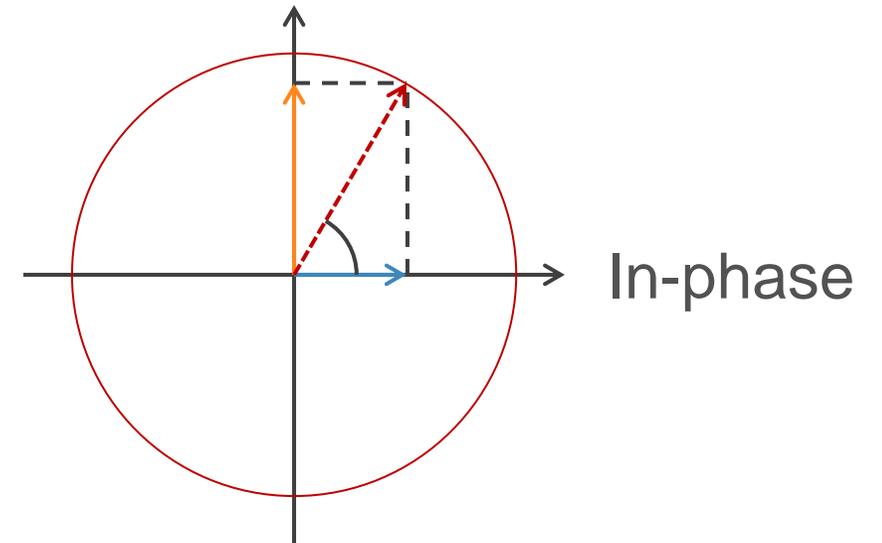


直交復調器による復調

直交復調器(IQ復調器)



Quadrature-phase



直交復調器を用いるとLOと

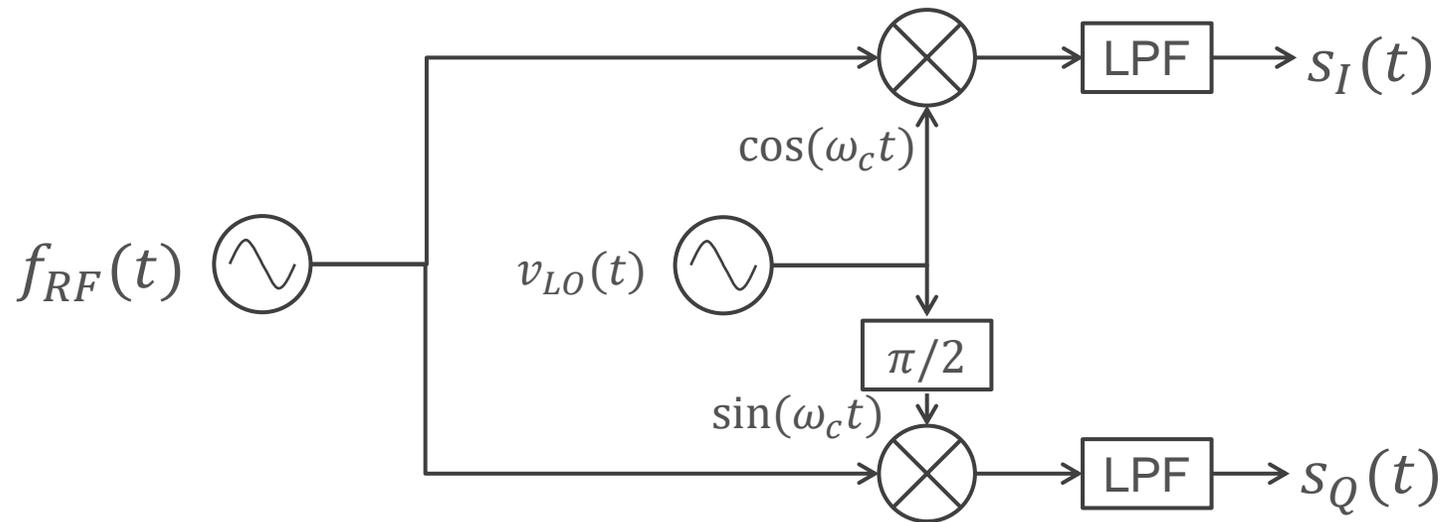
- 同相の信号成分(I)
- 直交する信号成分(Q)

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

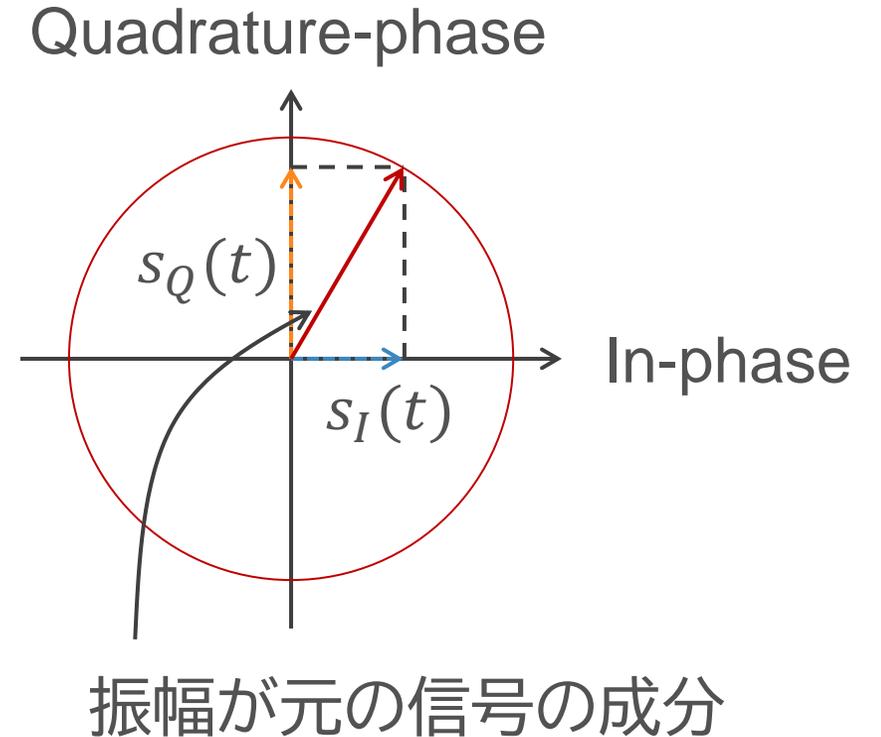
直交復調器を用いたAM復調



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、

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

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) ← キャリアの信号
vcq = np.sin(2*fc*np.pi*t) ← 90度シフトしたキャリアの信号

m = 0.2
fs = 20
vs = 1 + m*np.sin(2*fs*np.pi*t)
vam = vs*vcq } AM信号生成

vifi = vam*vci
vifq = vam*vcq } LO信号とRF信号の掛け算

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

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

↑ 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

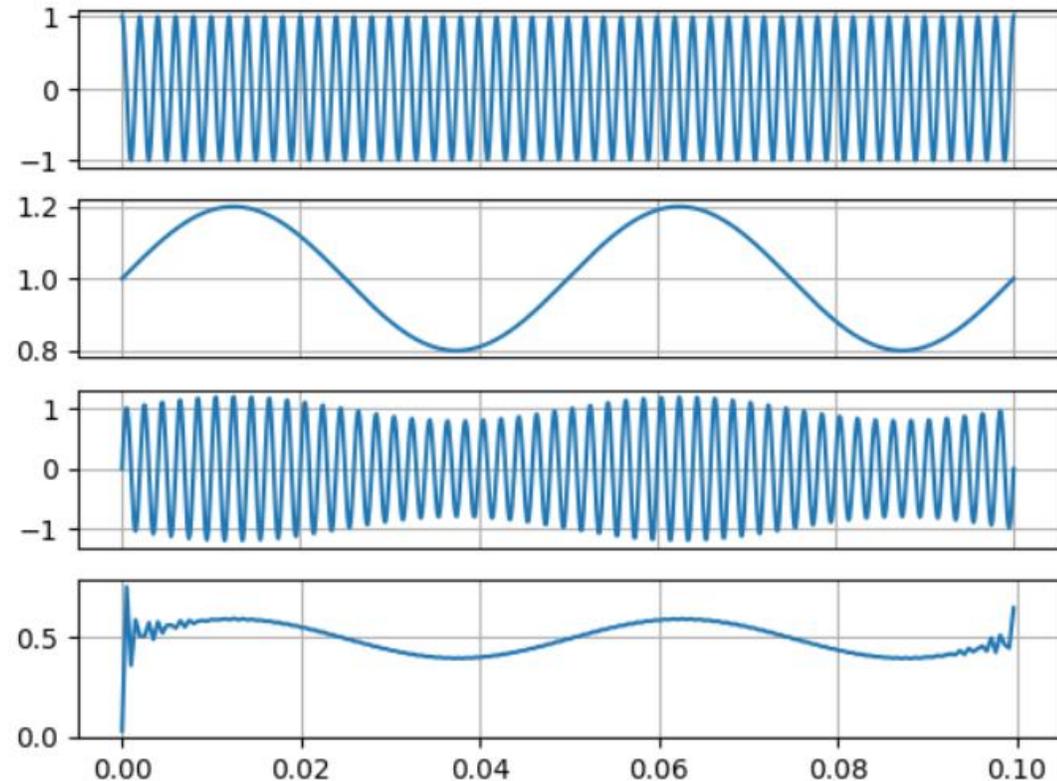
vifi = vam*vci
vifq = vam*vcq

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。

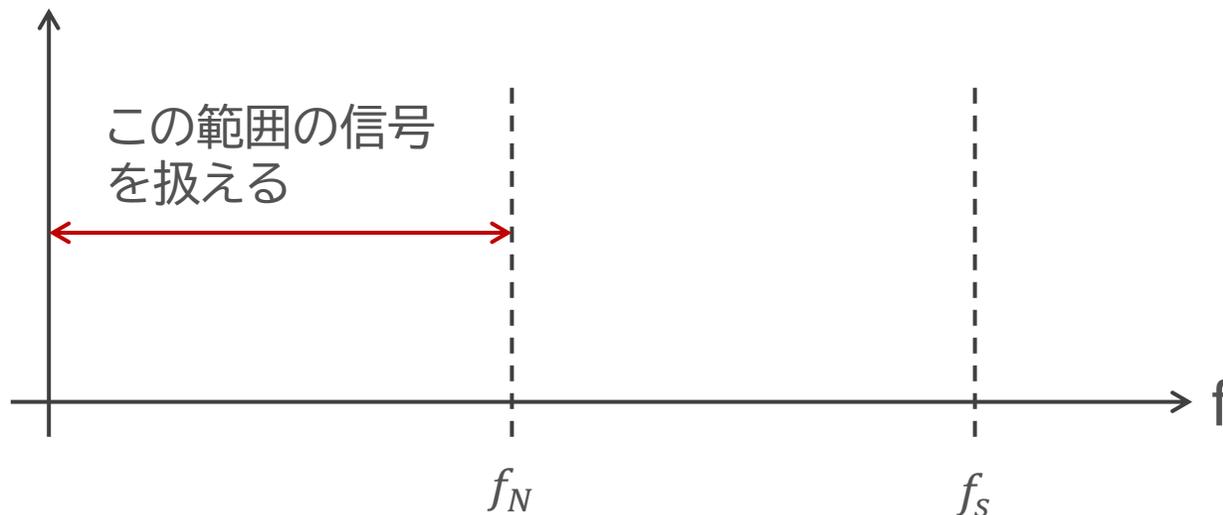
ナイキスト周波数

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

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

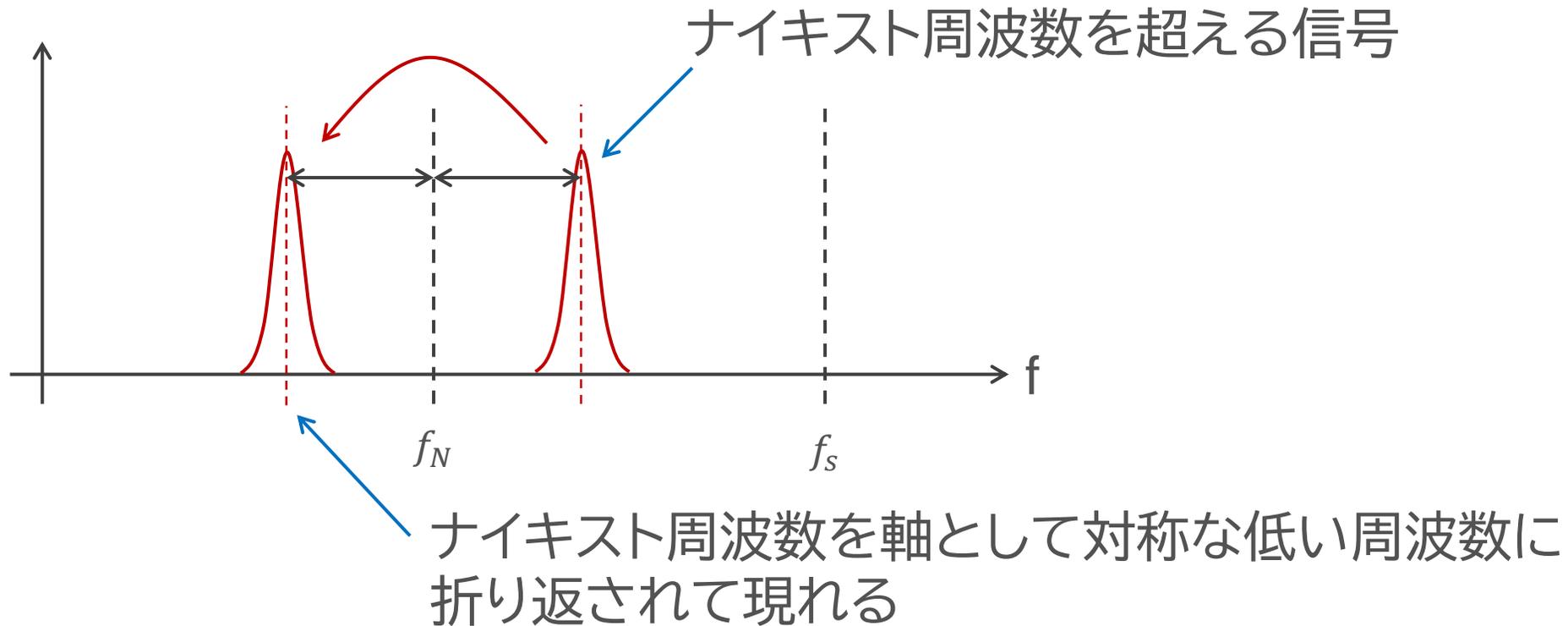
f_s : サンプルング周波数
 f_N : ナイキスト周波数

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)

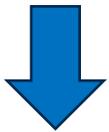
ナイキスト周波数を超える信号はどうなるのか？



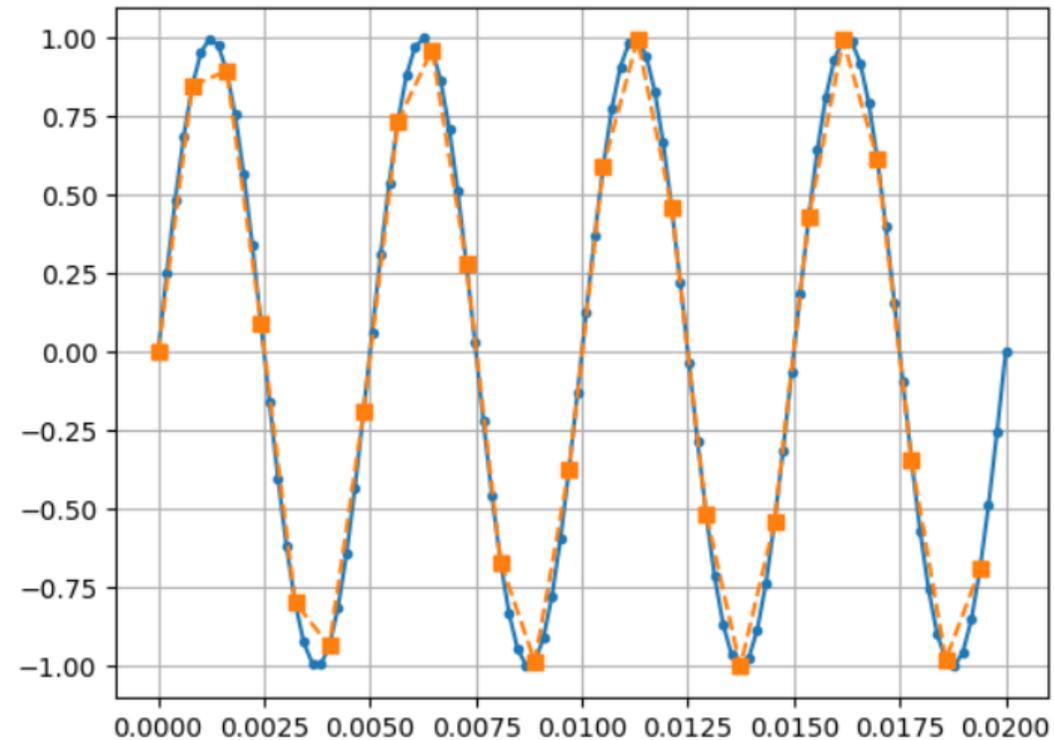
デシメーション

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



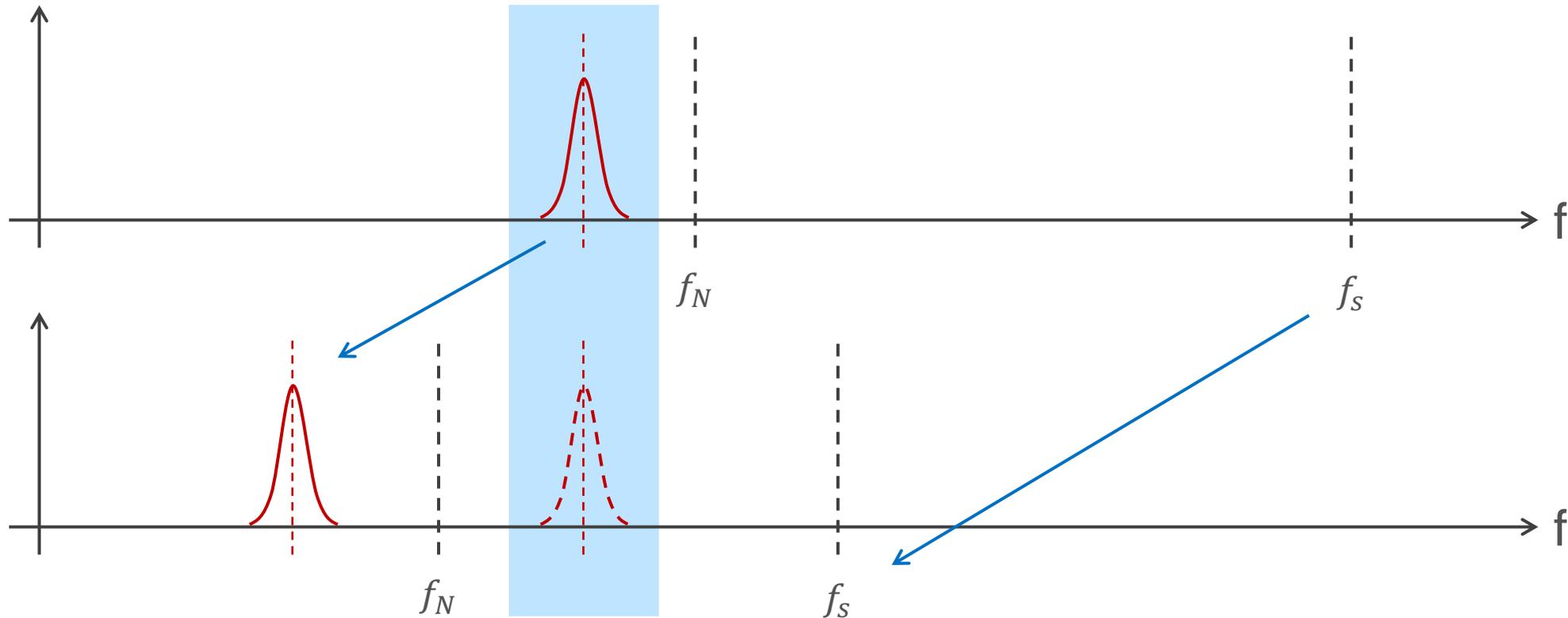
- サンプリング周波数を下げることで計算負荷を下げられる。
- ナイキスト周波数が変わるので、周波数変換ができる



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

この帯域のみフィルタリング



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。

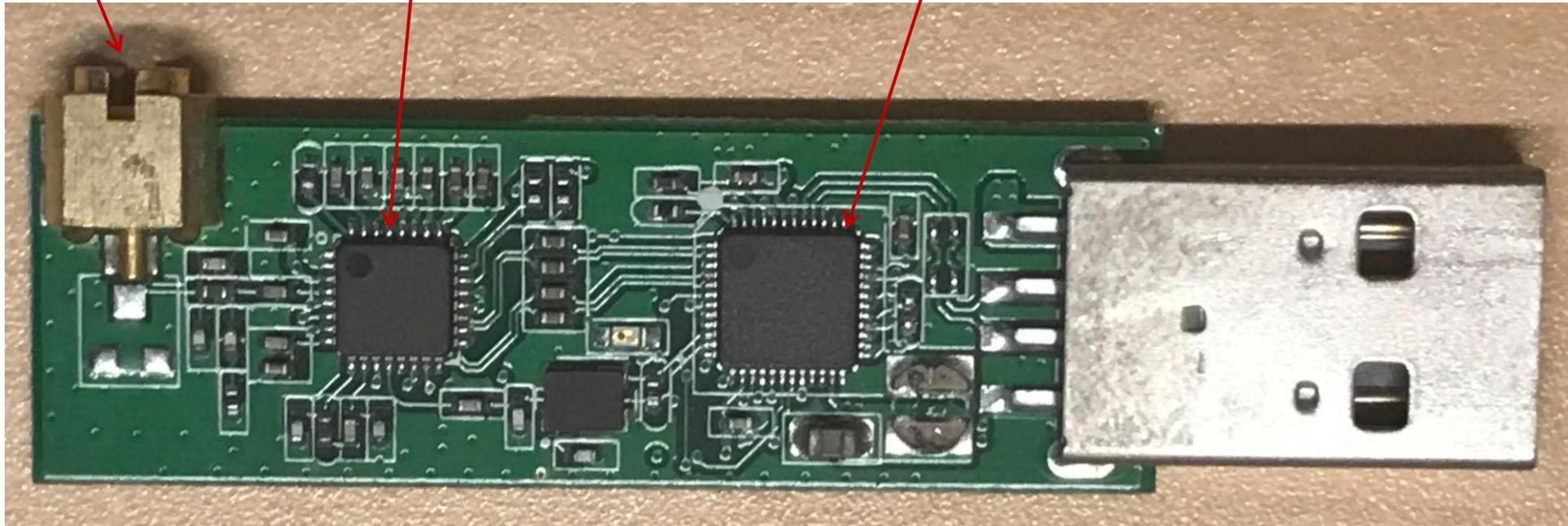
ソフトウェア無線機の構成

RTL2832U + FC0013

RF入力

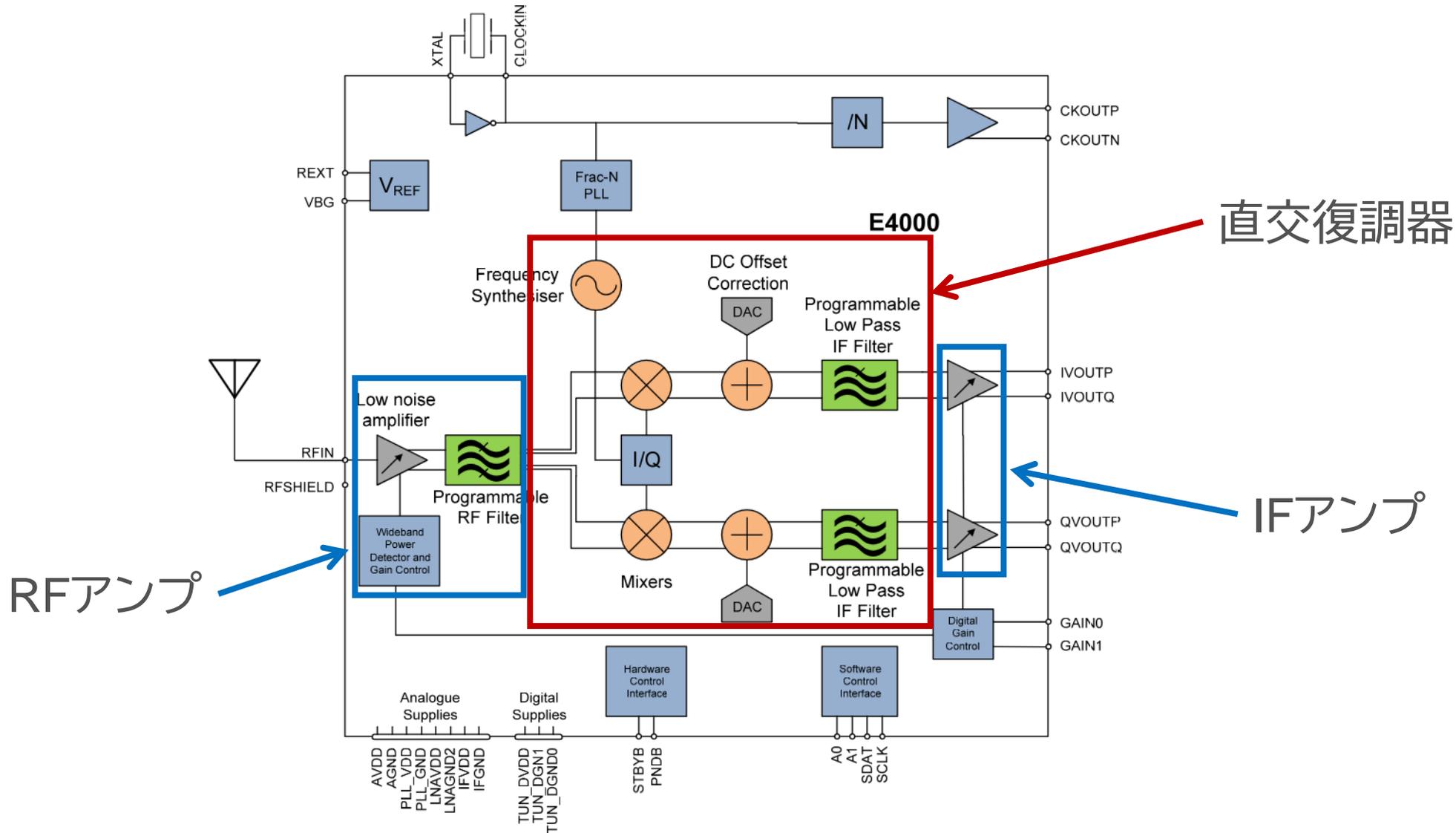
FC0013(RF tuner)

RTL2832U (AD コンバータ、USB phy)

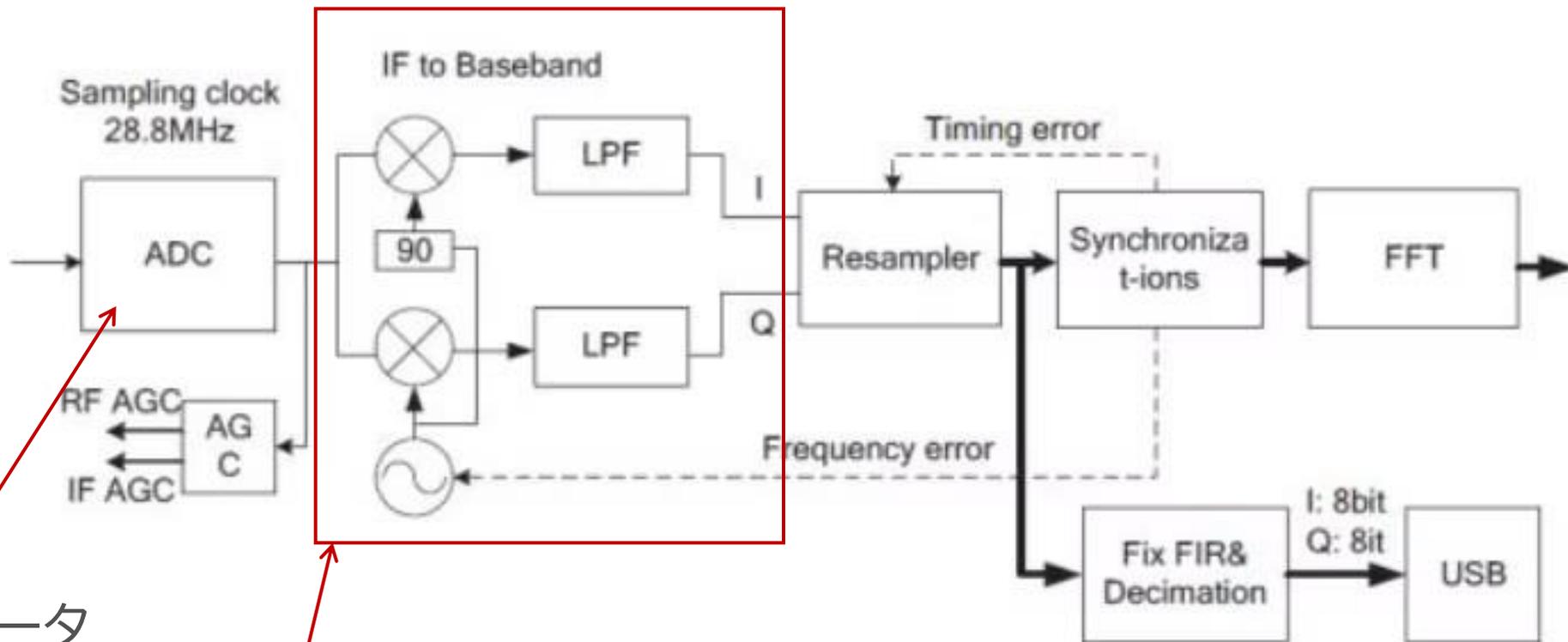


RFチューナーICの構成

今回のICとはちょっと異なりますが...



RTL2832U



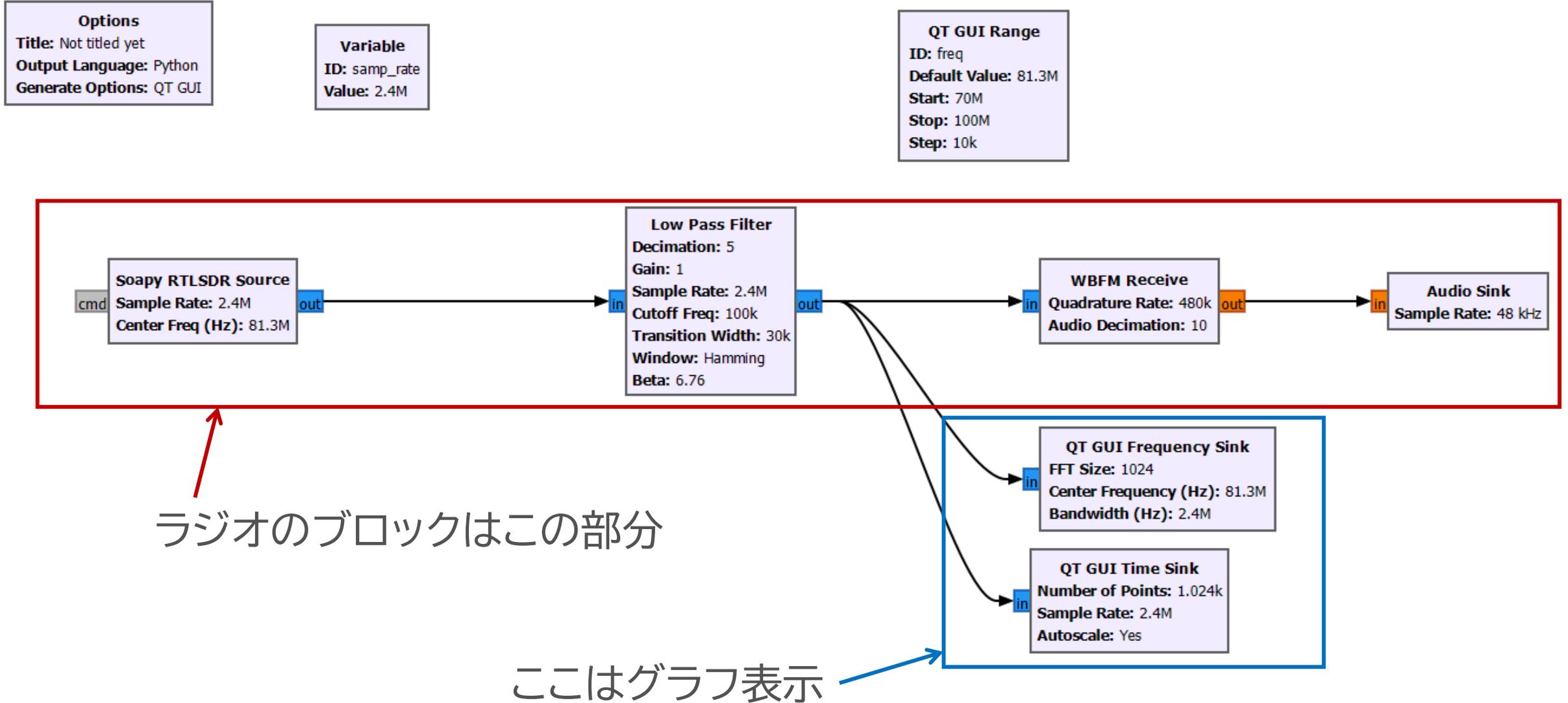
ADコンバータ

デジタル直交変調器

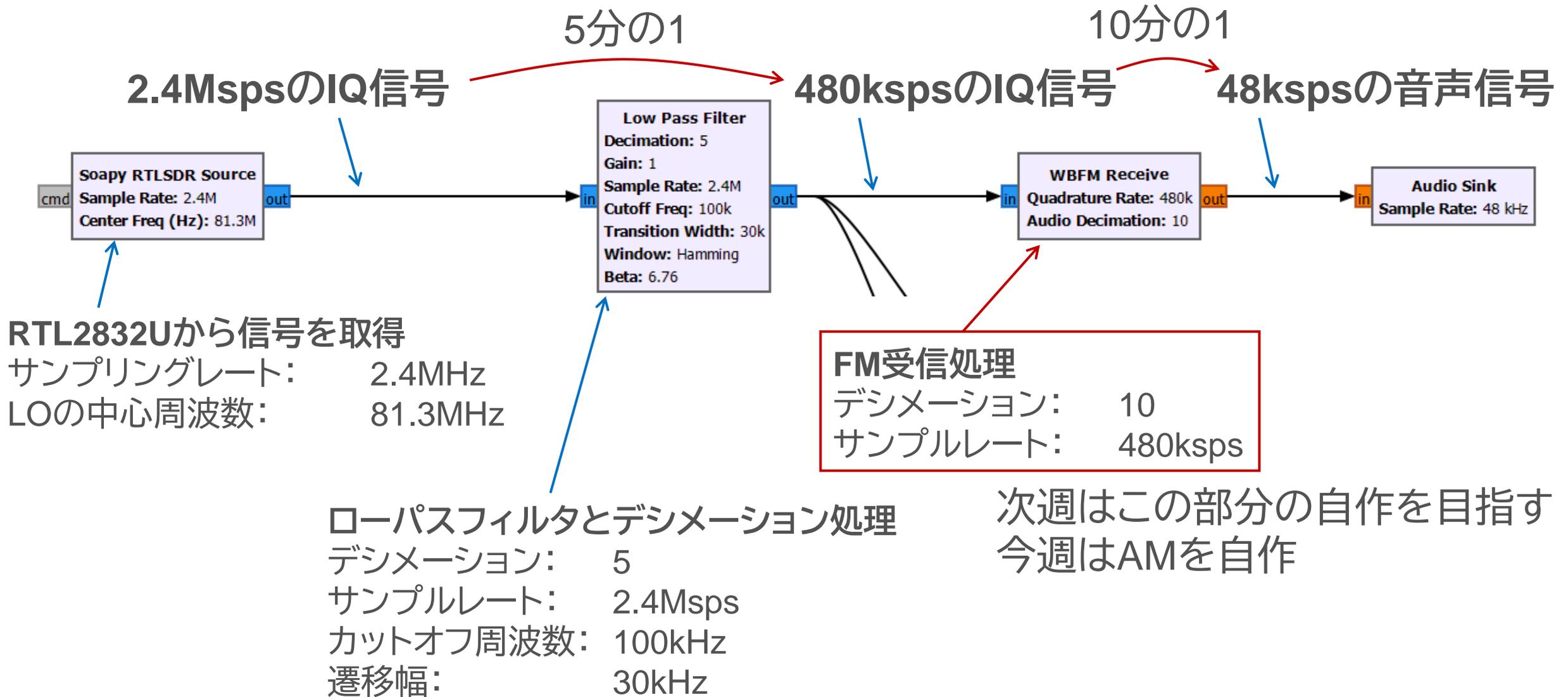
※実はこのチップ単体でもSDRとして振る舞う事ができる
元はISDB-T(テレビ放送)受信用のIC

FMラジオブロックを読み解く

前回のブロック図



ラジオ部分を読み解く



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

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

File Edit View Run Tools Help

fm_only_TokyoFM x am_only_RJTT_ATIS 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: 128.8M

Q band

- Core
 - Synchronizers
 - FLL Band-Edge
 - Filters
 - Band Pass Filter
 - Band Reject Filter
 - Band-pass Filter Taps
 - Band-reject Filter Taps

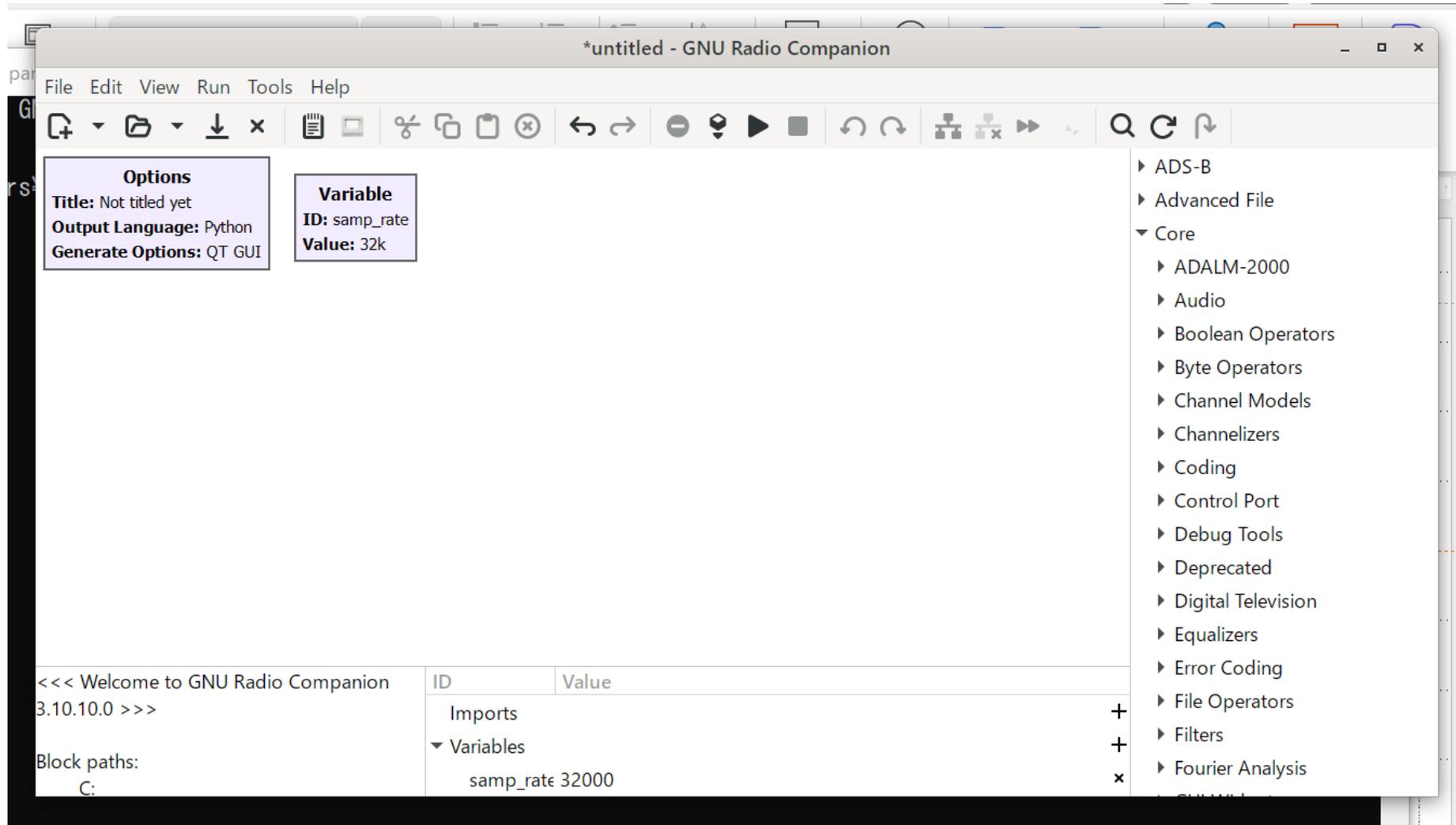
Executing: C:\Users\user\radioconda\python.exe -u C:\Users\user\Desktop\sdr\am_only_RJTT_ATIS.py

```
[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.

>>> Done
```

ID	Value	
Imports		
+ Variables		
freq	128800000.0	x
samp_rate	2400000.0	x

GNU Radioを起動



起動画面はこんな感じ

変数の追加と設定

The screenshot shows the GNU Radio Companion interface with the following components:

- Options Panel:**
 - Title: Not titled yet
 - Author: user
 - Output Language: Python
 - Generate Options: QT GUI
- Variable Configuration:**
 - Variable ID: samp_rate, Value: 2.4M
 - Variable ID: freq, Value: 122.8M
- Block Selection Panel:** Search for "Low" shows "Low Pass Filter" selected.
- Terminal:**

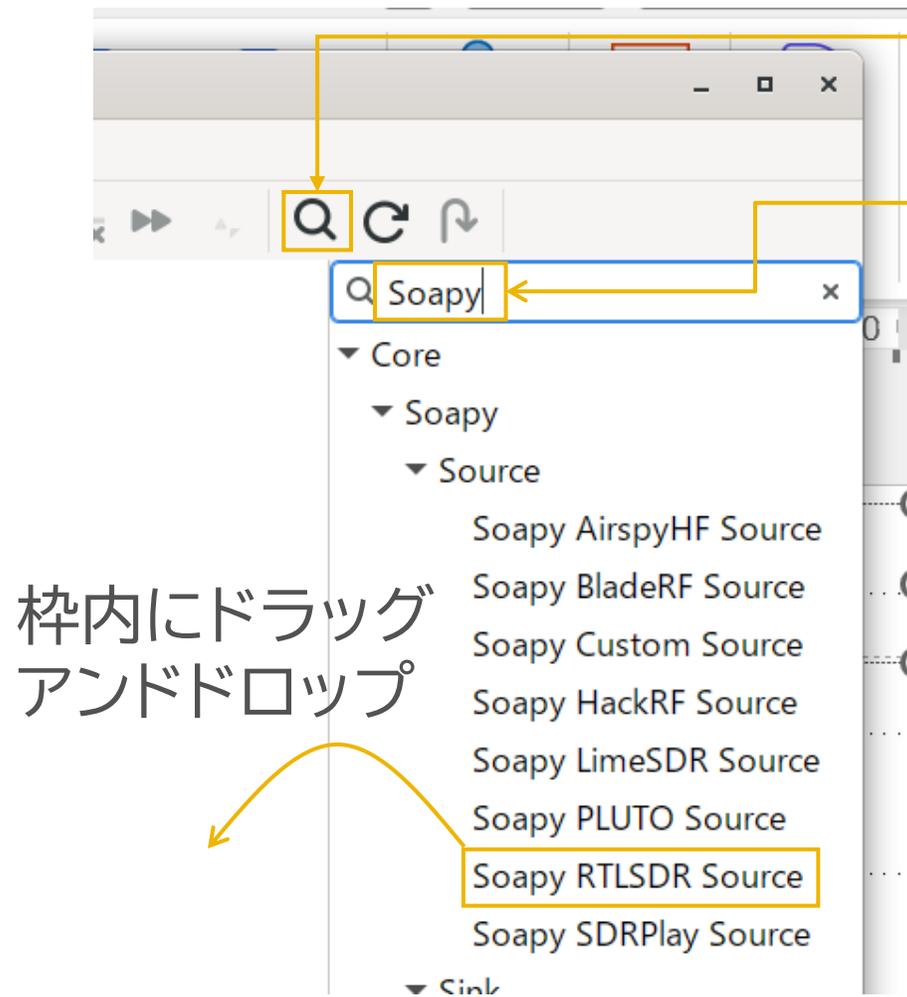
```
<<< Welcome to GNU Radio Companion 3.10.10.0 >>>
Block paths:
C:
¥Users¥user¥radioconda¥Library¥share¥gnuradio¥grc¥blocks
Loading: "C:¥Users¥user¥Desktop¥sdr¥fm_only_TokyoFM.grc"
>>> Done
Loading: "C:¥Users¥user¥Desktop¥sdr¥am_only_RJTT_ATIS.grc"
>>> Done
```
- Variables Table:**

ID	Value
freq	122800000.0
samp_rate	2400000.0

Annotations with yellow arrows point to the values in the Variables table:

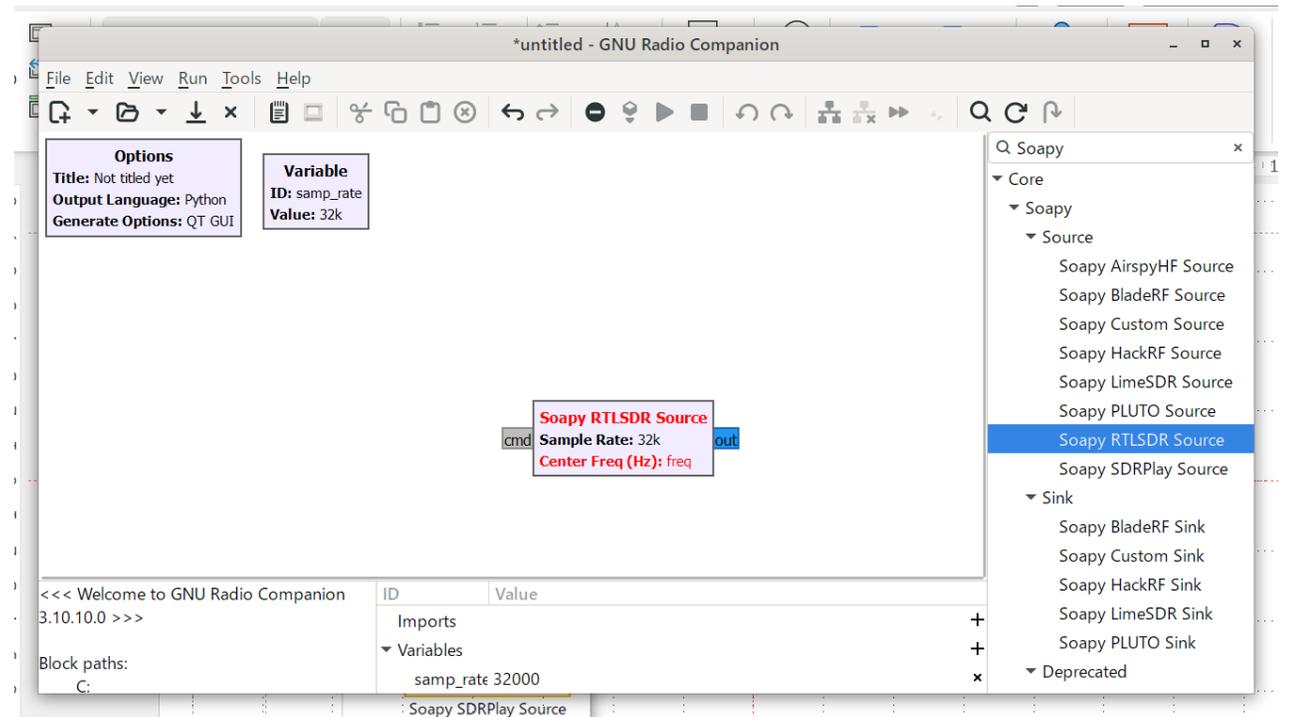
- freqを作成して、128.8e6を入力 (Create freq and input 128.8e6)
- samp_rateに2.4e6を入力 (Input 2.4e6 to samp_rate)

RTL SDRを追加



虫眼鏡(検索)を左クリック

Soapyを入力



Low pass filterを追加

GNU Radio Companion interface showing the process of adding a Low Pass Filter.

「low pass」を検索 (Search for "low pass")

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

OutからInに配線をつなぐ (Connect the output of the source to the input of the filter)

The screenshot shows the GNU Radio Companion interface with the following components and settings:

- Options:** Title: Not titled yet, Author: user, Output Language: Python, Generate Options: QT GUI
- Variable 1:** ID: samp_rate, Value: 2.4M
- Variable 2:** ID: freq, Value: 122.8M
- Soapy RTLSDR Source:** Sample Rate: 2.4M, Center Freq (Hz): 122.8M
- Low Pass Filter:** Decimation: 5, Gain: 10, Sample Rate: 2.4M, Cutoff Freq: 100k, Transition Width: 30k, Window: Hamming, Beta: 6.76

The search bar on the right contains the text "low", and the "Low Pass Filter" block is highlighted in the component palette. The workspace shows the Soapy RTLSDR Source block connected to the Low Pass Filter block.

The terminal output at the bottom shows the loading of the GNU Radio Companion 3.10.10.0 interface and the successful loading of the block paths for the Soapy RTLSDR Source and the Low Pass Filter.

```
<<< 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	122800000.0
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

The screenshot shows the 'Properties: Low Pass Filter' dialog box with the following settings and annotations:

- FIR Type:** Complex->Complex (Decimating)
- Decimation:** 5 (Annotation: 5を入力)
- Gain:** 10 (Annotation: 10を入力)
- Sample Rate:** samp_rate (Annotation: 100e3を入力)
- Cutoff Freq:** 100e3 (Annotation: 100e3を入力)
- Transition Width:** 30e3 (Annotation: 30e3を入力)
- Window:** Hamming
- Beta:** 6.76

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

Buttons: OK, Cancel, Apply (Annotation: 入力が終わったらOKをクリック)

Complex to magの追加

GNU Radio Companion interface showing the process of adding a 'Complex to Mag' block to a signal flow graph.

Magを入力 (Input Mag)

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 interface shows the following components and connections:

- Options:** Title: Not titled yet, Author: user, Output Language: Python, Generate Options: QT GUI
- Variables:**
 - Variable ID: samp_rate, Value: 2.4M
 - Variable ID: freq, Value: 128.8M
- Signal Flow Graph:**
 - 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:** (Newly added block)

The terminal output shows the loading of the blocks:

```
<<< 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

Band pass filterの追加

GNU Radio Companion interface showing the addition of a Band Pass Filter to a signal flow graph.

bandを入力 (Enter band)

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

The signal flow graph consists of the following blocks:

- 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

The component palette on the right shows the following filters:

- Core
- Synchronizers
- FLL Band-Edge
- Filters
 - Band Pass Filter** (highlighted)
 - Band Reject Filter
 - Band-pass Filter Taps
 - Band-reject Filter Taps

The terminal output shows the loading of the signal flow graph:

```
<<< 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

Band pass filter の設定

The image shows a screenshot of the 'Properties: Band Pass Filter' dialog box. The dialog has three tabs: 'General', 'Advanced', and 'Documentation'. The 'General' tab is selected. The settings are as follows:

Parameter	Value	Unit/Type
FIR Type	Float->Float (Real Taps) (Decim)	Dropdown
Decimation	10	[int]
Gain	3	[real]
Sample Rate	480e3	[real]
Low Cutoff Freq	10	[real]
High Cutoff Freq	5e3	[real]
Transition Width	10e3	[real]
Window	Hamming	Dropdown
Beta	6.76	[real]

Annotations with arrows point to the following elements:

- Float -> Float (Real Taps)(Decim)に変更 (Change to Float -> Float (Real Taps)(Decim))
- 10を入力 (Enter 10)
- 3を入力 (Enter 3)
- 480e3を入力 (Enter 480e3)
- 10を入力 (Enter 10)
- 5e3を入力 (Enter 5e3)
- 10e3を入力 (Enter 10e3)
- 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

*untitled - GNU Radio Companion

File Edit View Run Tools Help

fm_only_TokyoFM × am_only_RJTT_ATIS × untitled ×

auを入力

枠内にドラッグ
アンドドロップ

Variable
ID: samp_rate
Value: 2.4M

Variable
ID: freq
Value: 128.8M

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

OutからInに配線をつなぐ

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

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

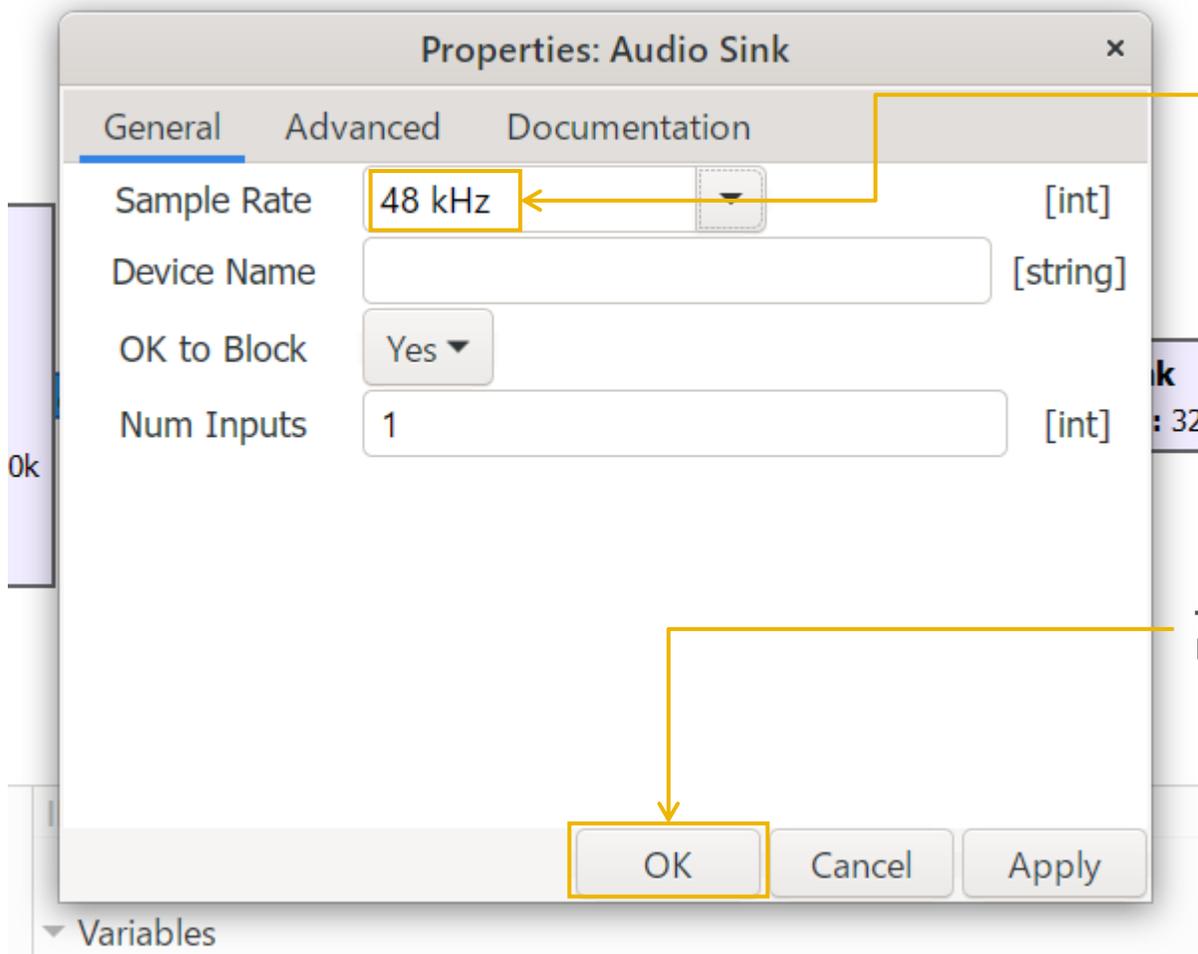
Loading: "C:%Users%user%Desktop%sdr%fm_only_TokyoFM.grc"
>>> Done

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

ID	Value
Imports	
Variables	
freq	128800000.0
samp_rate	2400000.0

- Core
 - Audio
 - Audio Sink
 - Audio Source
 - Alaw Audio Decoder
 - g711 Alaw Audio Encoder
 - CODEC2 Audio Decoder
 - CODEC2 Audio Encoder
 - CVSD Audio Decoder (Raw Bit-Level)
 - CVSD Audio Encoder (Raw Bit-Level)
 - 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
 - ulaw Audio Encoder
 - IQ Correction
 - Remove DC Spike AutoSync
 - Peak Detectors
 - Plateau Detector
 - Packet Operators
 - Packet Header Generator (Default)
 - Packet Header Parser (Default)
 - Default Header Format Obj.
 - GUI Widgets
 - QT

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

qtを入力

The screenshot shows the GNU Radio Companion interface. The main window displays a signal flow graph with the following blocks and connections:

- Soapy RTLSDR Source** (Sample Rate: 2.4M, Center Freq (Hz): 128.8M) connects to the **in** port of the **Low Pass Filter**.
- Low Pass Filter** (Decimation: 5, Gain: 10, Sample Rate: 2.4M, Cutoff Freq: 100k, Transition Width: 30k, Window: Hamming, Beta: 6.76) connects to the **in** port of the **Complex to Mag** block.
- Complex to Mag** block connects to the **in** port of the **Band Pass Filter**.
- 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) connects to the **in** port of the **Audio Sink**.
- Audio Sink** (Sample Rate: 48 kHz) is the final output block.
- There are also two **QT GUI** blocks connected to the output of the **Complex to Mag** block:
 - 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)

The block search panel on the right shows the search results for "qt". The **QT GUI Time Sink** block is highlighted in blue, and a yellow box is drawn around it. A yellow arrow points from the text "qtを入力" to the search input field, and another yellow arrow points from the highlighted block to the text "枠内にドラッグアンドドロップ".

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
- 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
- jets
- UI App Background
- UI Fast Auto-Correlator Sink
- UI 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

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

Properties: QT GUI Time Sink

General Advanced Trigger Config Documentation

Type Complex

Name "" [string]

Y Axis Label Amplitude [string]

Y Axis Unit "" [string]

Number of Points 1024 [int]

Sample Rate samp_rate [float]

Grid No

Autoscale Yes

Y min -1 [float]

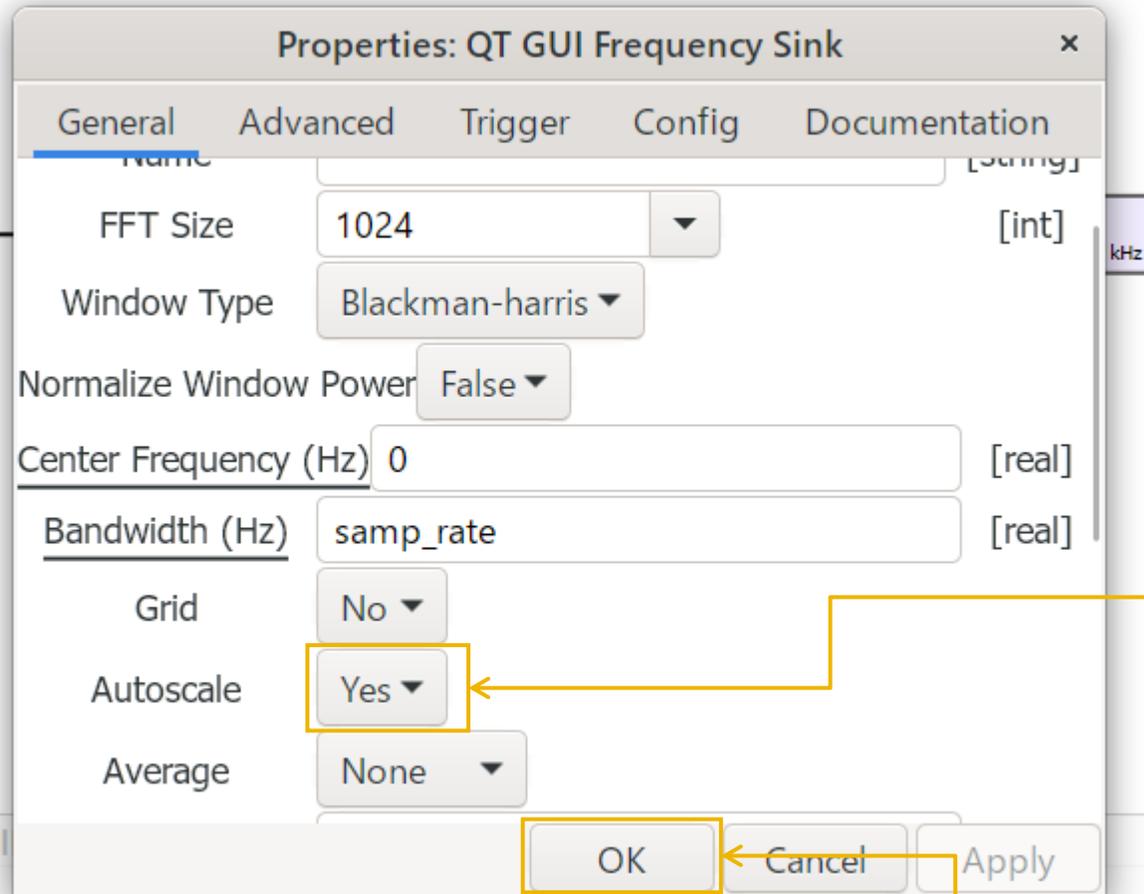
OK Cancel Apply

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

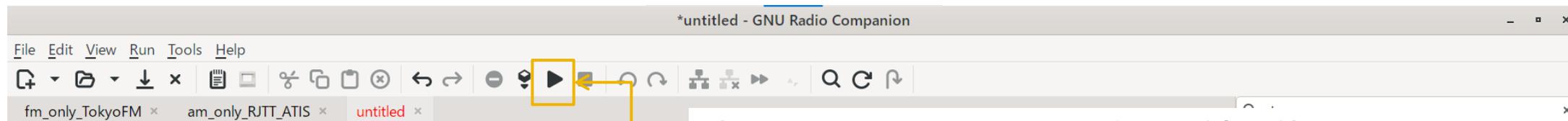
QT GUI Frequency sinkブロックをダブルクリックして設定を開く



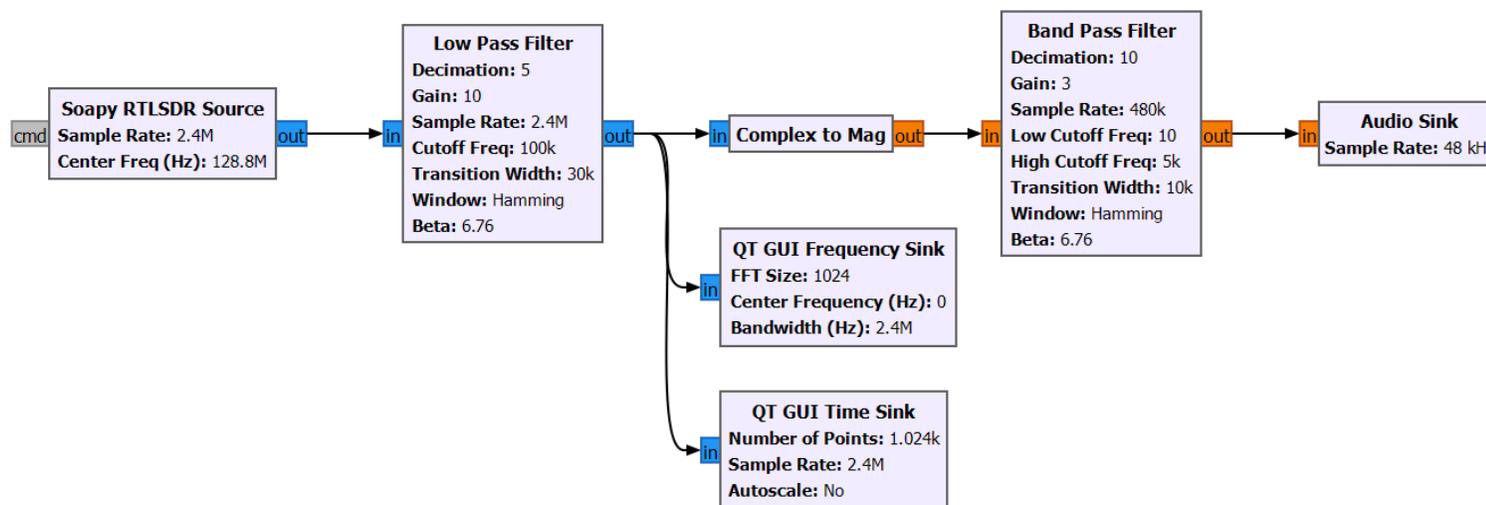
AutoscaleをYesに変更する

設定したらOKをクリック

ラジオの起動



左クリックするとラジオが起動する



▼ 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

<<< 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