

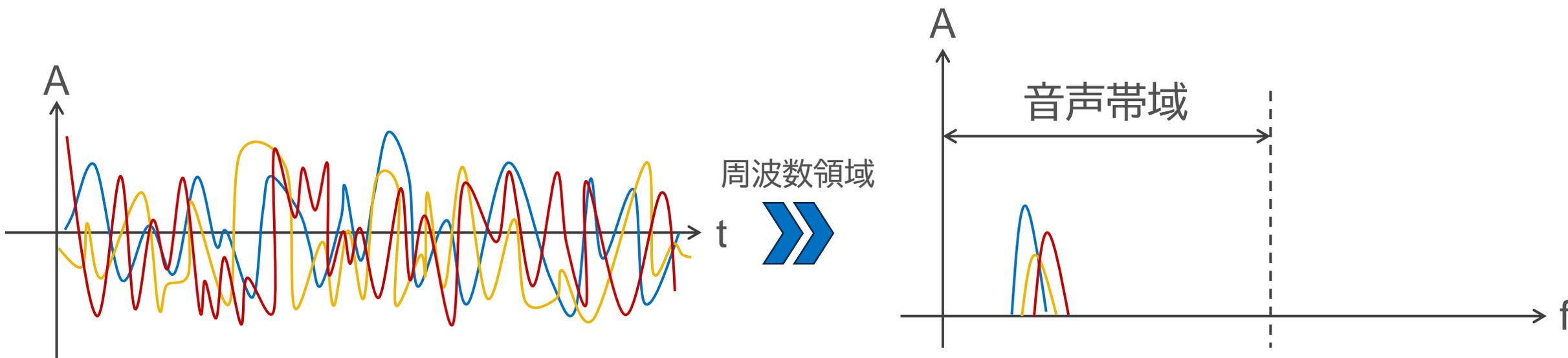
ソフトウェア無線機

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

音声を伝送する

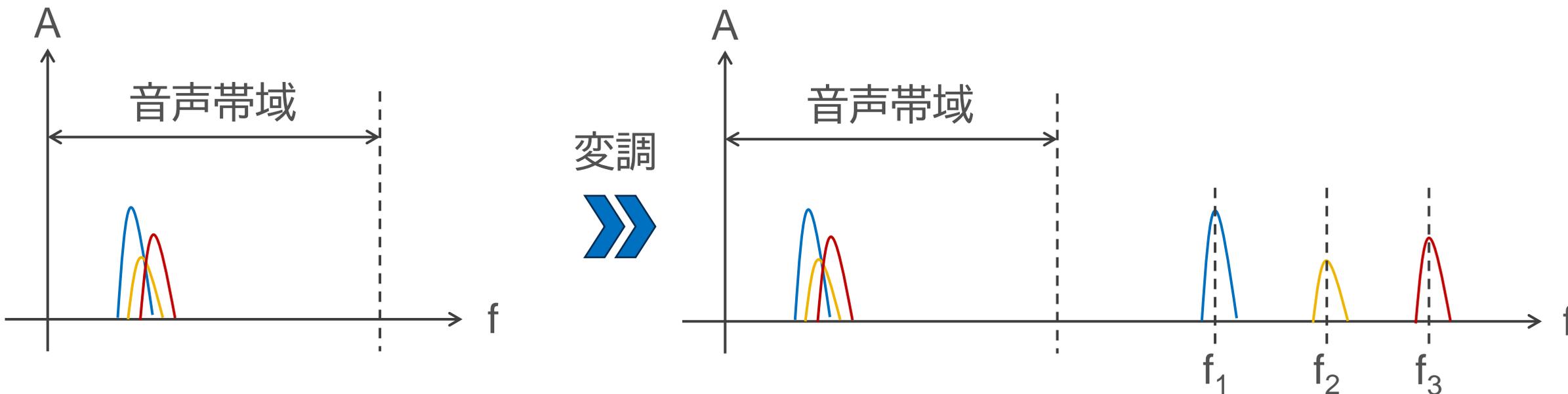


3つの音声を同時に混ぜて送っても、受け手側で分離することは難しい。

➡ 異なる周波数に情報を載せ直してから伝送する

→ 変調という

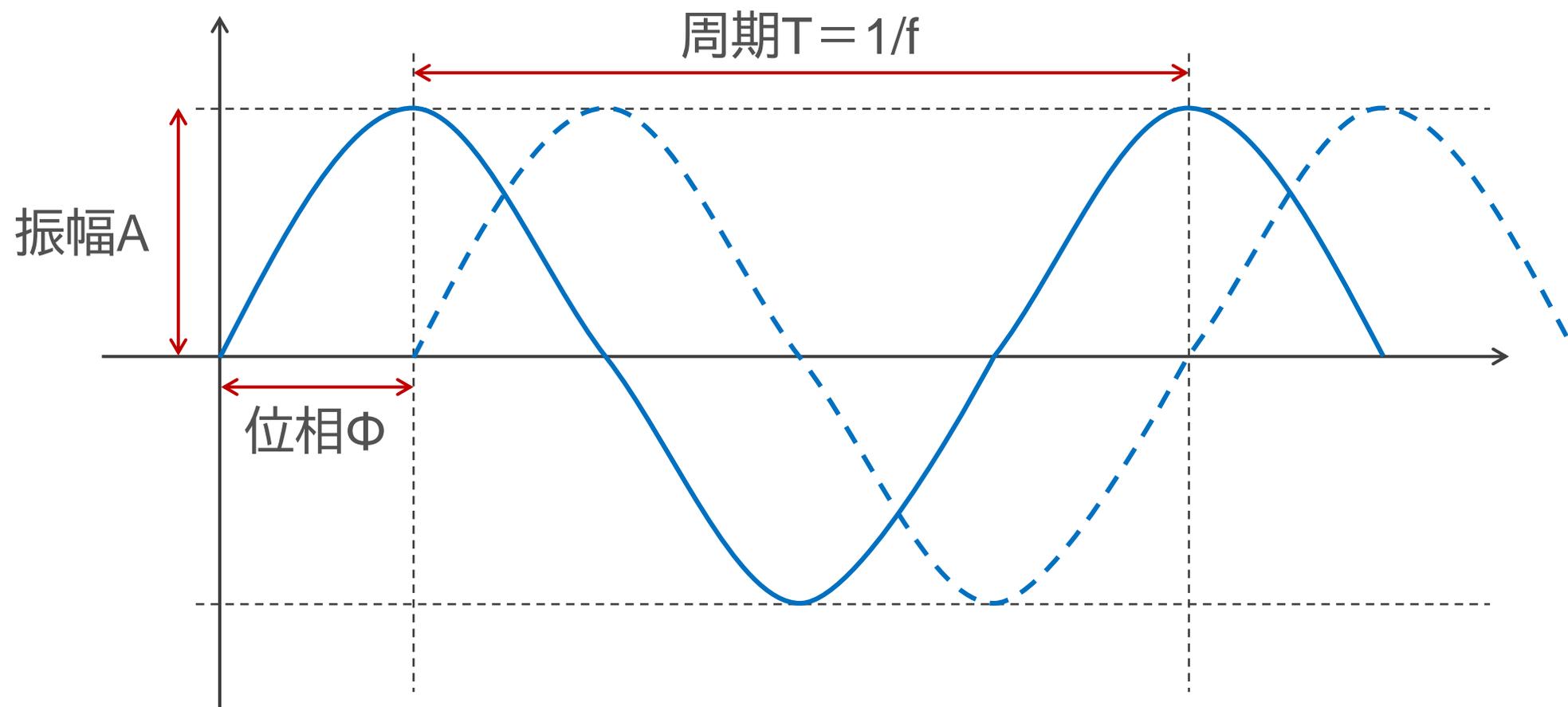
変調とは



そのままでは混ぜてしまう信号を別の周波数帯域に分離して伝送、受信側で元の信号に戻す方法

波を特徴づける3要素のいずれかに目的の信号を載せる事ができる。

波を特徴づける3要素



$$v(t) = A \sin(2\omega f t + \phi)$$

波の3要素を変調する

- 振幅を変調
 - AM: amplitude modulation
- 周波数を変調
 - FM: frequency modulation
- 位相を変調
 - PM: phase modulation

振幅変調

AMの式

$$v_{AM} = (V_C + V_S \cos(2\pi f_s t)) \sin(2\pi f_c t)$$

変調信号

キャリア信号

キャリアの振幅成分に信号を乗せる

一般的には変調度 $m = V_m/V_C$ を用いて、

$$v_{AM} = V_C (1 + m \cos(2\pi f_s t)) \sin(2\pi f_c t)$$

と書く。

振幅変調のシミュレーション

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

t = np.linspace(0, 0.1, 1000) ← 0~0.1秒の時間軸を作る
fc = 500 ← 500Hzのキャリア周波数
vc = np.sin(2*fc*np.pi*t) ← キャリアを生成
m = 0.2 ← 変調指数m
fs = 20 ← 信号の周波数
vs = 1 + m*np.sin(2*fs*np.pi*t) ← 信号の生成

v = vs*vc ← キャリアと信号の積 (AM処理)

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

この辺はグラフ処理

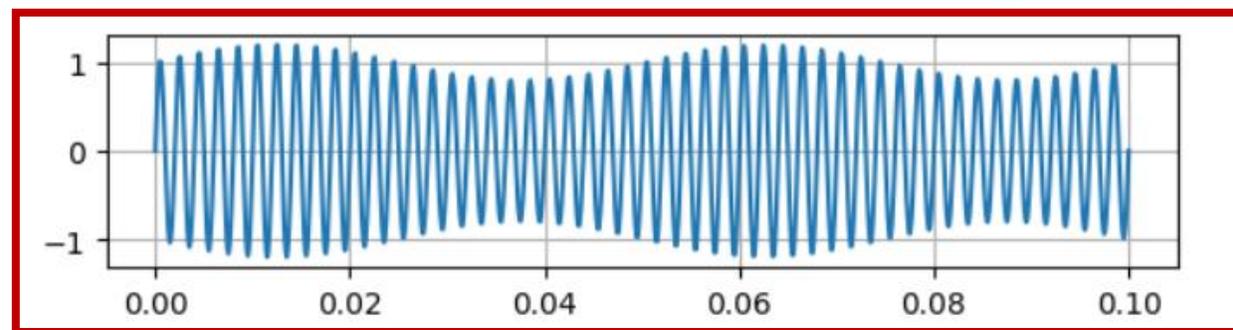
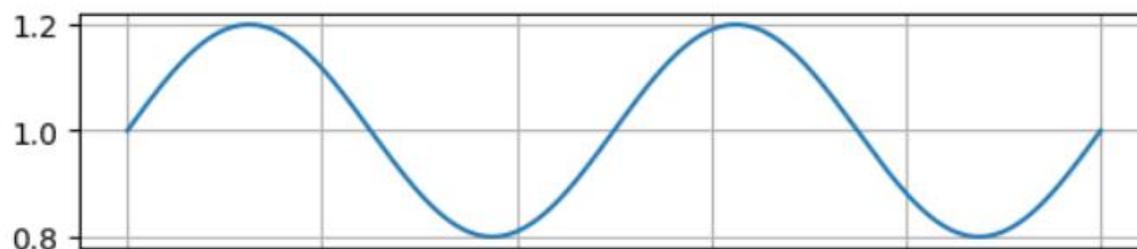
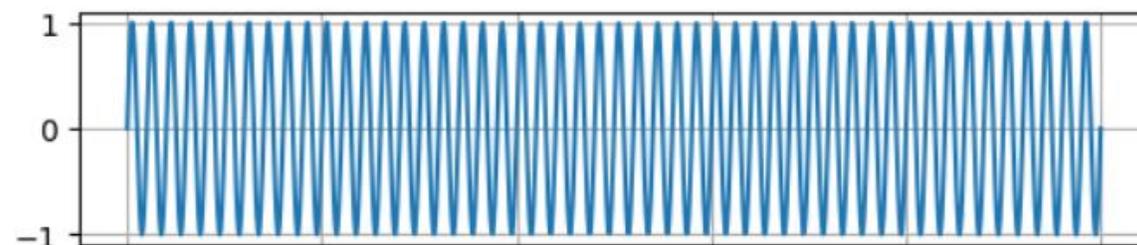
Colabでのシミュレーション結果

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

t = np.linspace(0, 0.1, 1000)
fc = 500
vc = np.sin(2*fc*np.pi*t)
m = 0.2
fs = 20
vs = 1 + m*np.sin(2*fs*np.pi*t)

v = vs*vc

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



授業ではこのグラフだけ作成

この部分をplt.plot(t, v)に置き換える

位相変調

PMの式

$$v_{PM} = V_c \sin(2\pi f_c t + \Delta\phi \sin(2\pi f_s t))$$

信号

キャリアの位相成分に信号を乗せる

$\Delta\phi$ はどの程度位相を偏移させるのかを表すパラメータで最大位相偏移という。正弦波の値域は $-1 \sim +1$ なので、 $-\Delta\phi \sim +\Delta\phi$ の範囲で位相が変化する。

位相変調のシミュレーション

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

```
t = np.linspace(0, 0.1, 1000) ← 0~0.1秒の時間軸を作る
fc = 200 ← 200Hzのキャリア周波数を設定
```

```
delta_phi = 120/180*np.pi ← 最大位相偏移 $\Delta\Phi$ を指定
fs = 20 ← 信号の周波数を設定
vs = delta_phi * np.sin(2*fs*np.pi*t) ← 信号の生成
```

```
v = np.sin(2*fc*np.pi*t + vs) ← PM処理
```

```
fig = plt.figure()
ax = fig.add_subplot(211)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(212)
ay.grid()
ay.plot(t, v)
```

```
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)
```

この辺はグラフ処理

Colabでのシミュレーション

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

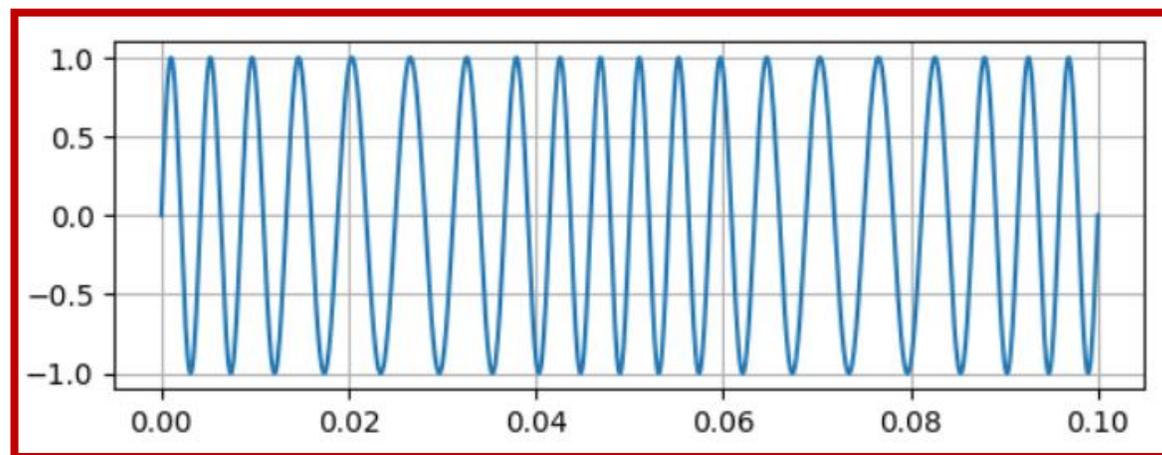
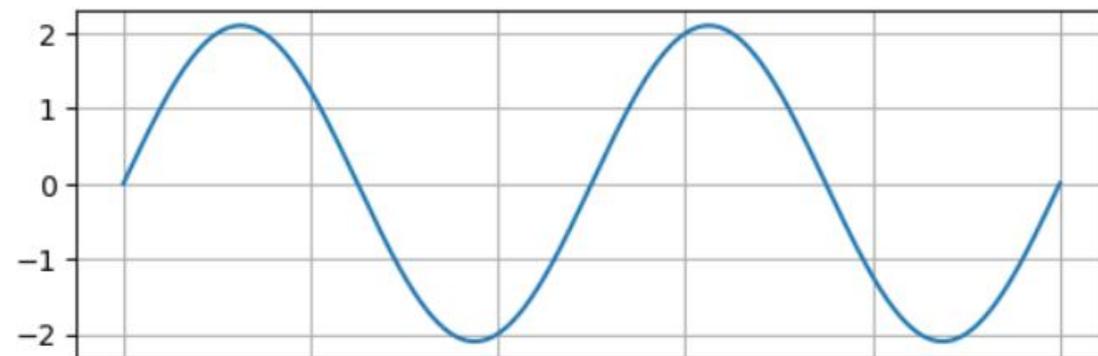
t = np.linspace(0, 0.1, 1000)
fc = 200

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

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

fig = plt.figure()
ax = fig.add_subplot(211)
ax.grid()
ax.plot(t, vs)
ay = fig.add_subplot(212)
ay.grid()
ay.plot(t, v)

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)
```



授業ではこのグラフだけ作成
この部分を`plt.plot(t, v)`に置き換える

周波数変調

FMの式

$$v_{FM} = V_c \sin(2\pi(f_c + \Delta\omega \sin(2\pi f_s t)) t)$$

信号

キャリアの周波数成分に信号を乗せる

- と、書きたいけれど、実際にはこの定義を実現する回路が無い。
(現実の回路は位相または位相の変化量を変える回路のみ)

FMの定義

$$v_{FM} = V_c \sin(2\pi f_c t + \Delta\omega \int_0^t \sin(2\pi f_s t) dt)$$

信号

PMとほとんど同じ式となる。

周波数変調と位相変調の関係1

周波数とはなにか？

位相の時間変化を示す値(位相の時間微分)

$$\frac{d\Phi_s}{dt} = \omega_s \quad \gg \quad \text{位相を使って表す事のできる値}$$

↑
角周波数(2πで割ると周波数)

周波数変調における位相を考える

信号で周波数を変える $\gg \frac{d\Phi_s}{dt} = s(t) \leftarrow$ 位相の時間変化が信号s(t)になるようにする

したがって、ある時間の位相 $\Phi_s(t)$ は積分することで、

$$\Phi_s(t) = \int_0^t \sin(2\pi f_s t) dt$$

となる。

周波数変調と位相変調の関係2

周波数変調の位相はキャリアの位相との合成になる。

$$\Phi(t) = 2\pi f_c t + \Delta\omega \int_0^t \sin(2\pi f_s t) dt$$

← FMの位相項

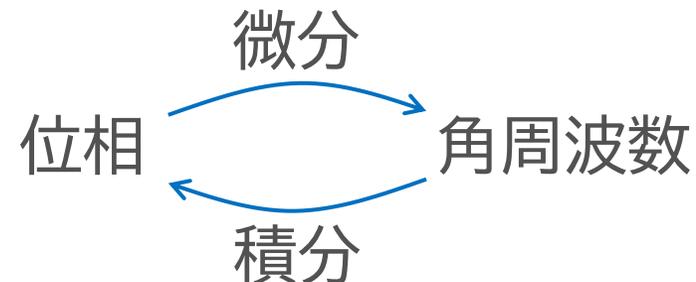
↑
キャリアの位相項

したがってFMの式は、

$$v_{FM} = V_c \sin\left(2\pi f_c t + \Delta\omega \int_0^t \sin(2\pi f_s t) dt\right)$$

となる。

位相と周波数は次の関係がある。



周波数変調のシミュレーション

```

import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 0.1, 1000) ← 0~0.1秒の時間軸を作る
fc = 200 ← 200Hzのキャリア周波数を設定

delta_omega = 0.1 ← 最大周波数偏移 $\Delta\omega$ を設定
fs = 20 ← 信号の周波数を設定
vs = delta_omega * np.sin(2*fs*np.pi*t) ← 信号の生成
phi_s = np.cumsum(vs) ← 積分処理(離散信号では累積和( $\Sigma$ )で代替)

v = np.sin(2*fc*np.pi*t + phi_s) ← FM処理

fig = plt.figure()
ax = fig.add_subplot(211)
ax.grid()
ax.plot(t, phi_s)
ay = fig.add_subplot(212)
ay.grid()
ay.plot(t, v)

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)

```

この辺はグラフ処理

Colabでのシミュレーション

```
[18] import numpy as np
import matplotlib.pyplot as plt

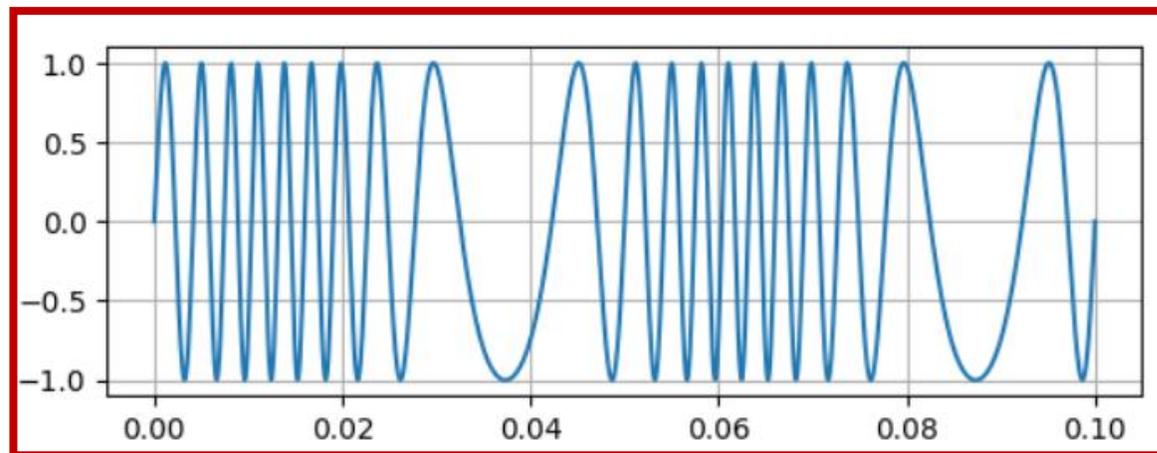
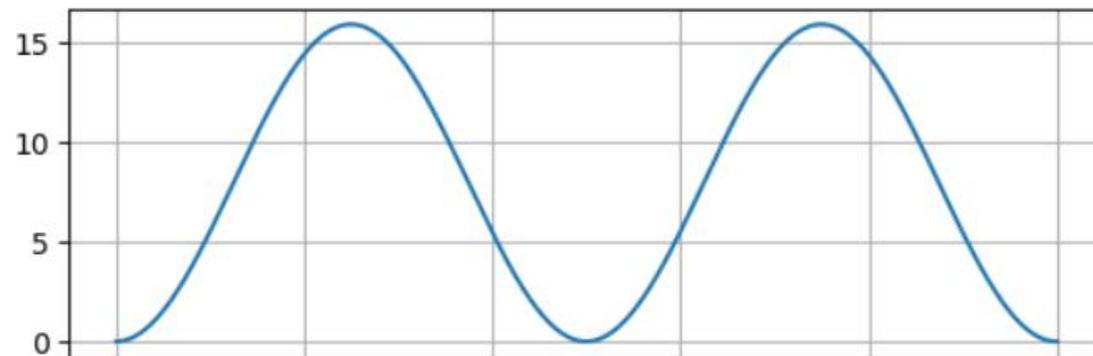
t = np.linspace(0, 0.1, 1000)
fc = 200

delta_omega = 0.1
fs = 20
vs = delta_omega * np.sin(2*fs*np.pi*t)
phi_s = np.cumsum(vs)

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

fig = plt.figure()
ax = fig.add_subplot(211)
ax.grid()
ax.plot(t, phi_s)
ay = fig.add_subplot(212)
ay.grid()
ay.plot(t, v)

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)
```

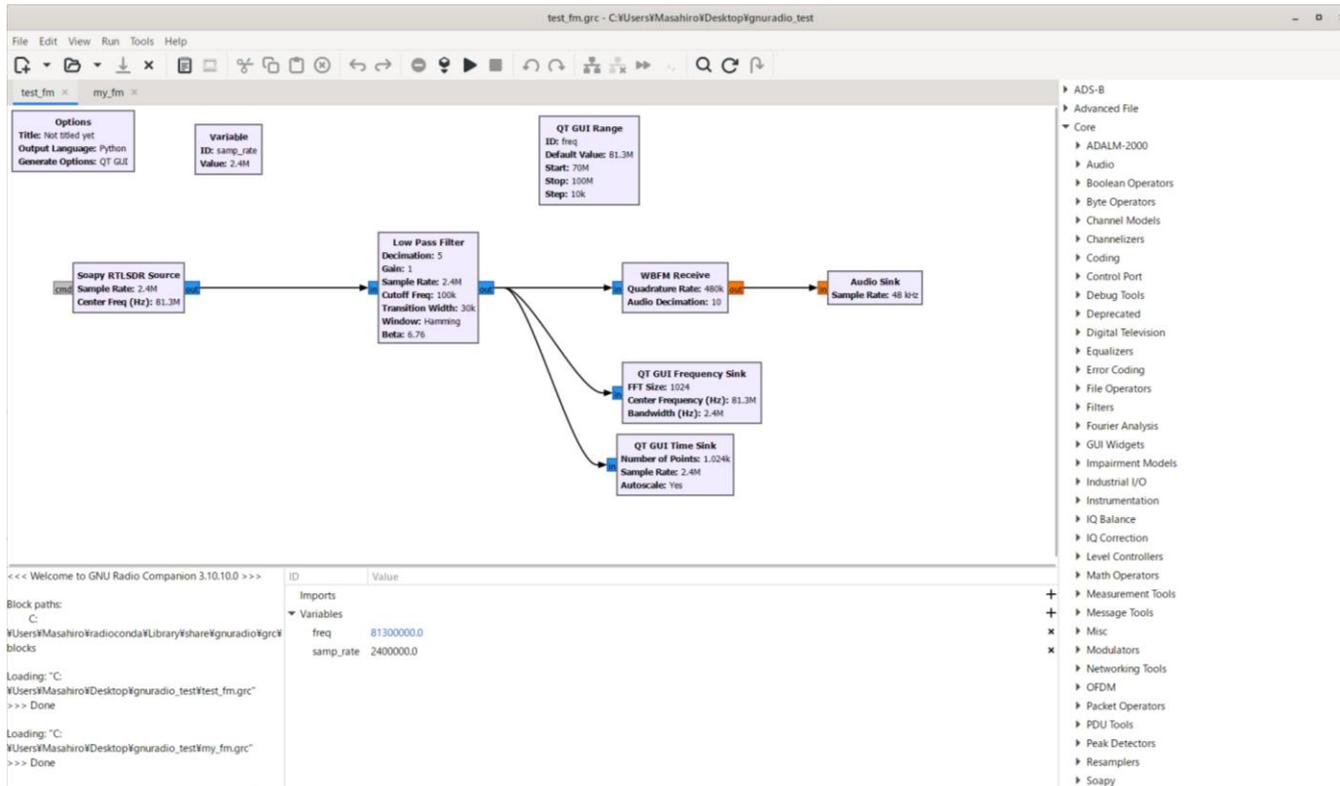


授業ではこのグラフだけ作成

この部分をplt.plot(t,v)に置き換える

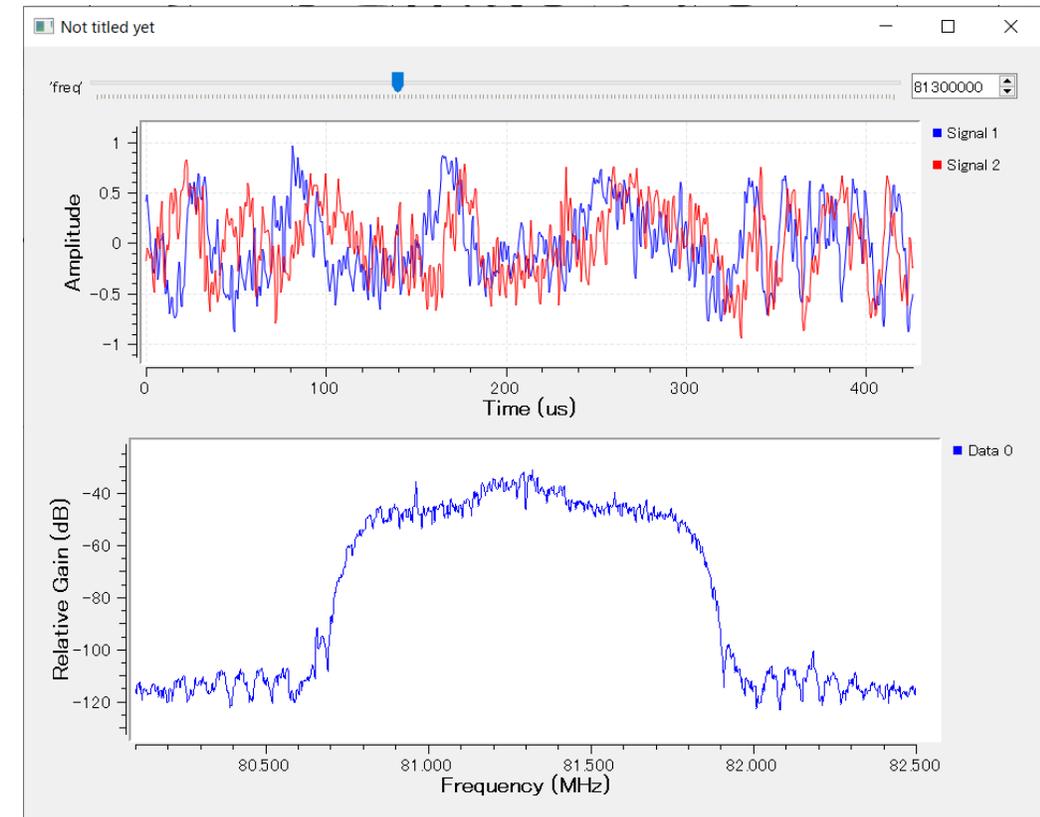
SDRの体験

ここではFMラジオを作成してみる



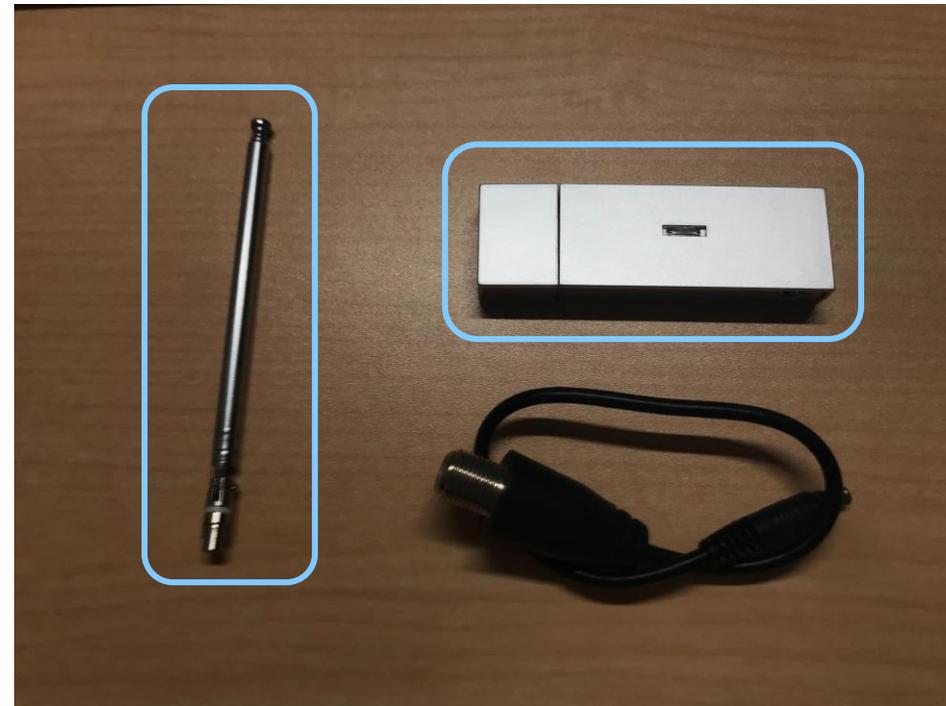
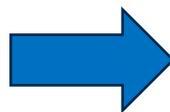
こんなブロック図で

こんな画面が出る



受信機の準備

内容物



今回使うのはアンテナと銀色の本体

アンテナの接続



つめ

切り欠きとつめを
合わせて差し込む

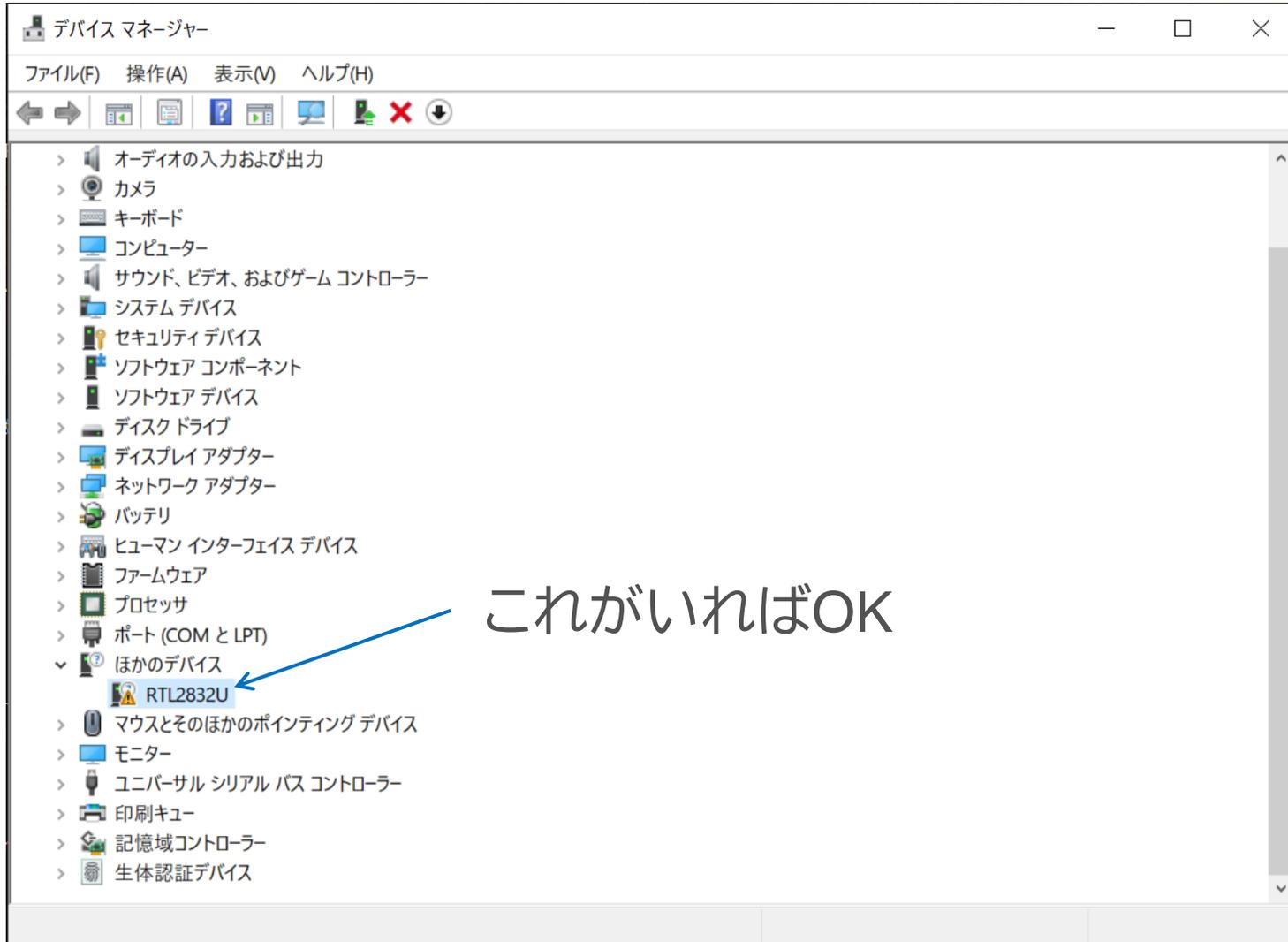
切り欠き

本体とアンテナの縁が同じに
なるくらいまで押し込む



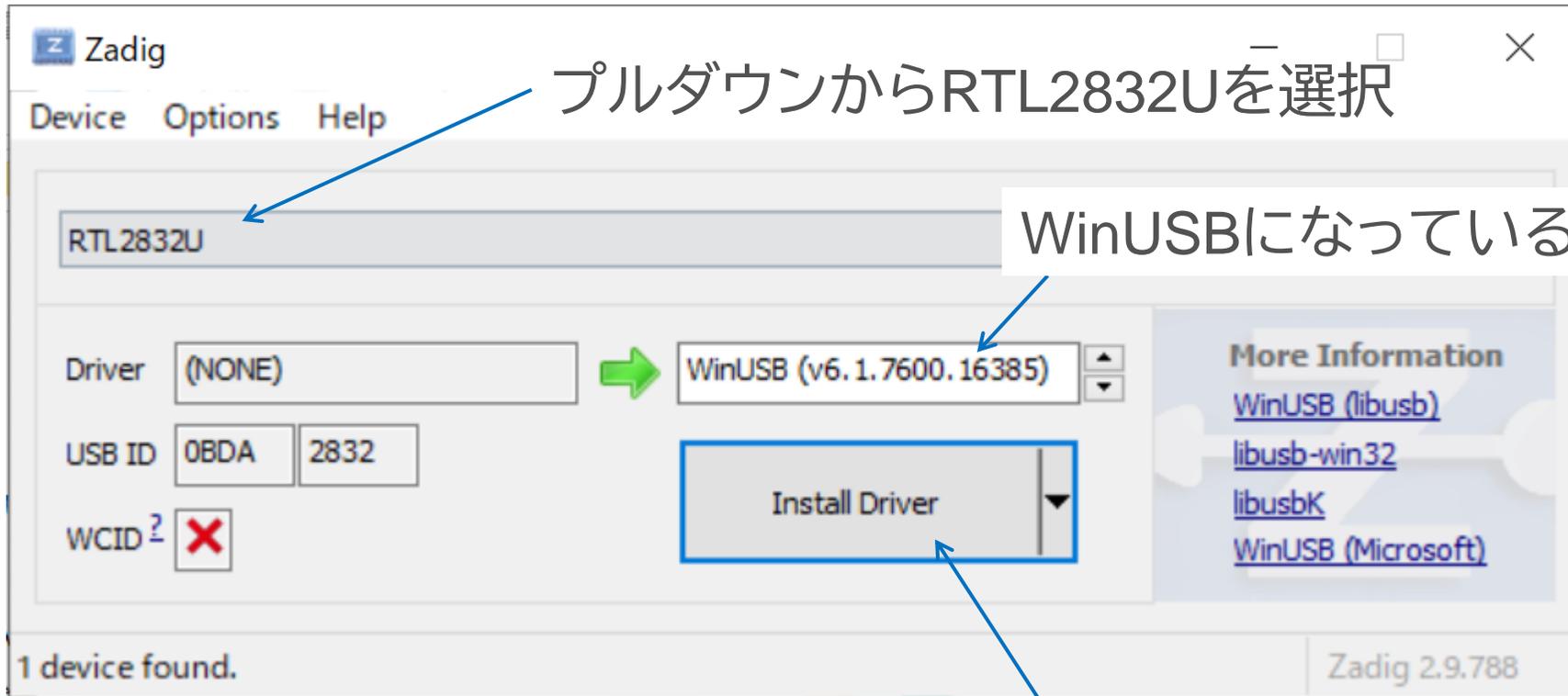
ドライバの入れ替え

デバイスが認識しているか確認



Zadigの起動

事前にダウンロードしたZadigを起動する。

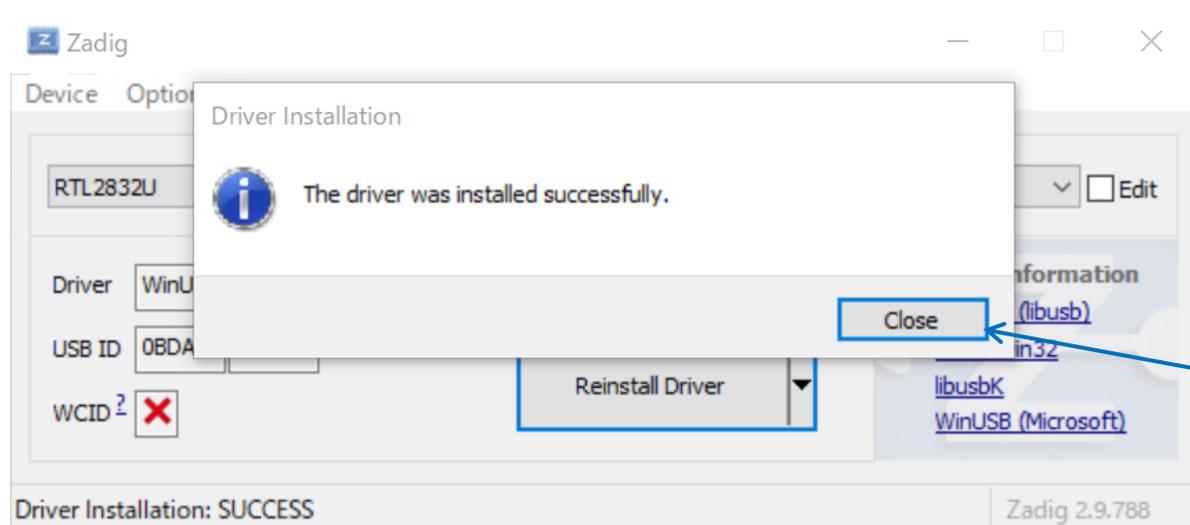
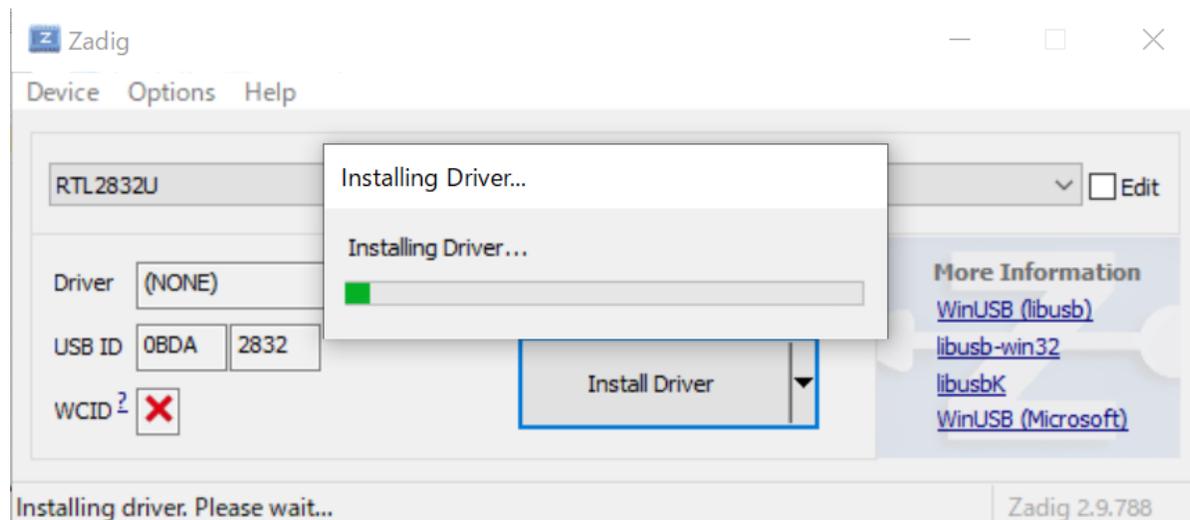


プルダウンからRTL2832Uを選択

WinUSBになっていることを確認

対象とドライバを確認したら左クリック

インストール中

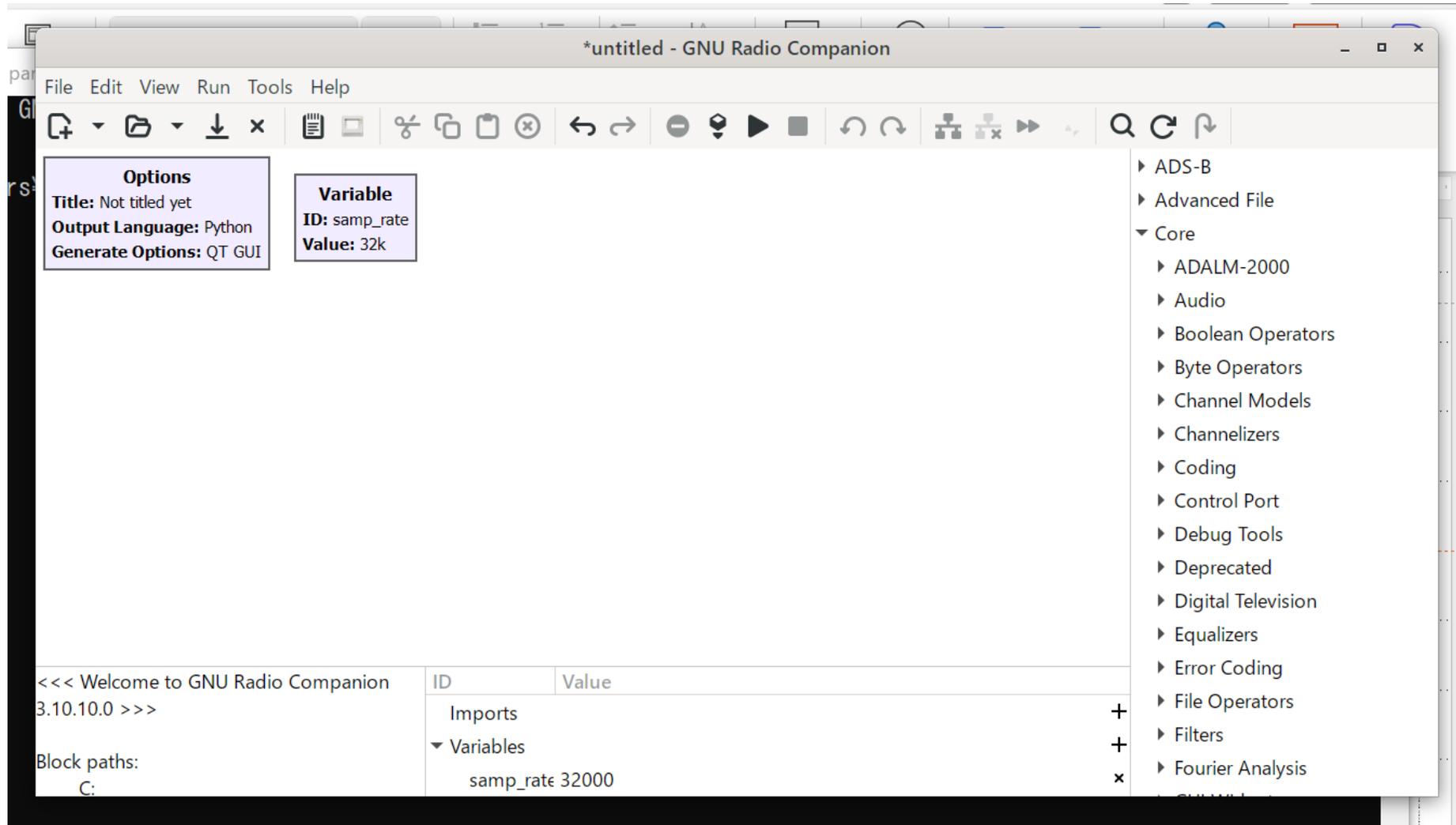


インストール完了

左クリックで閉じる

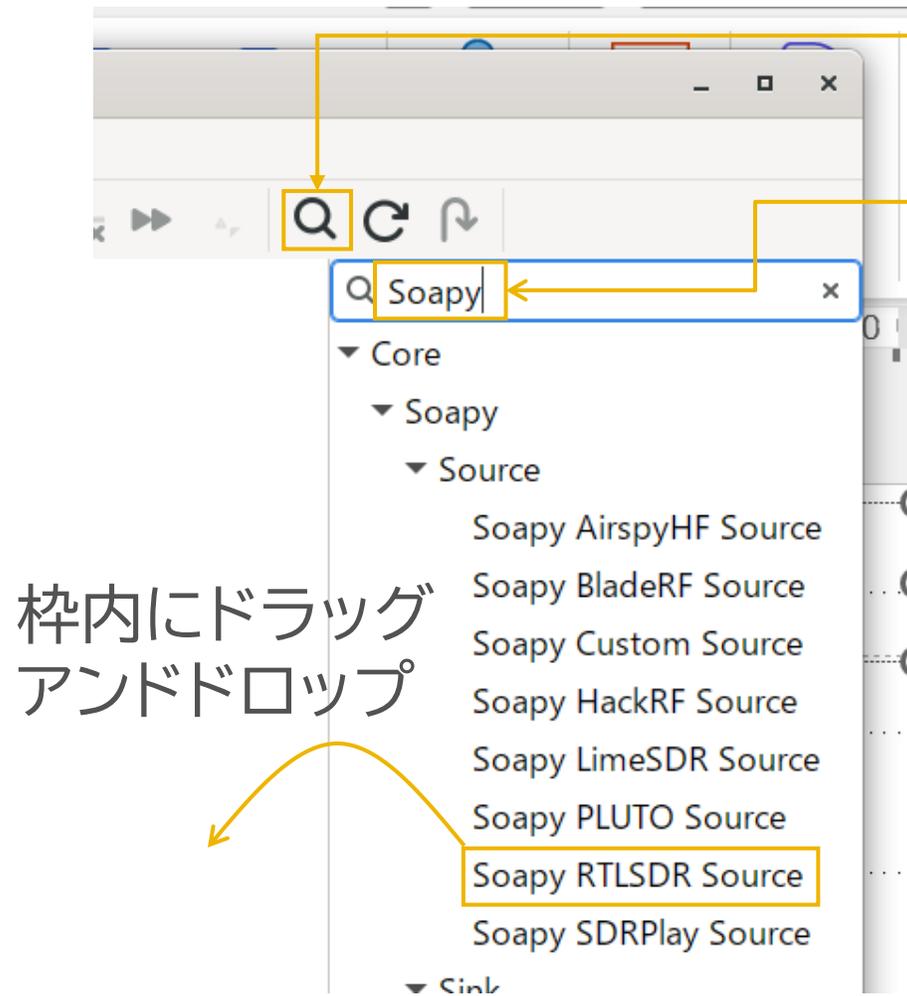
FMラジオの実装

GNU Radioを起動



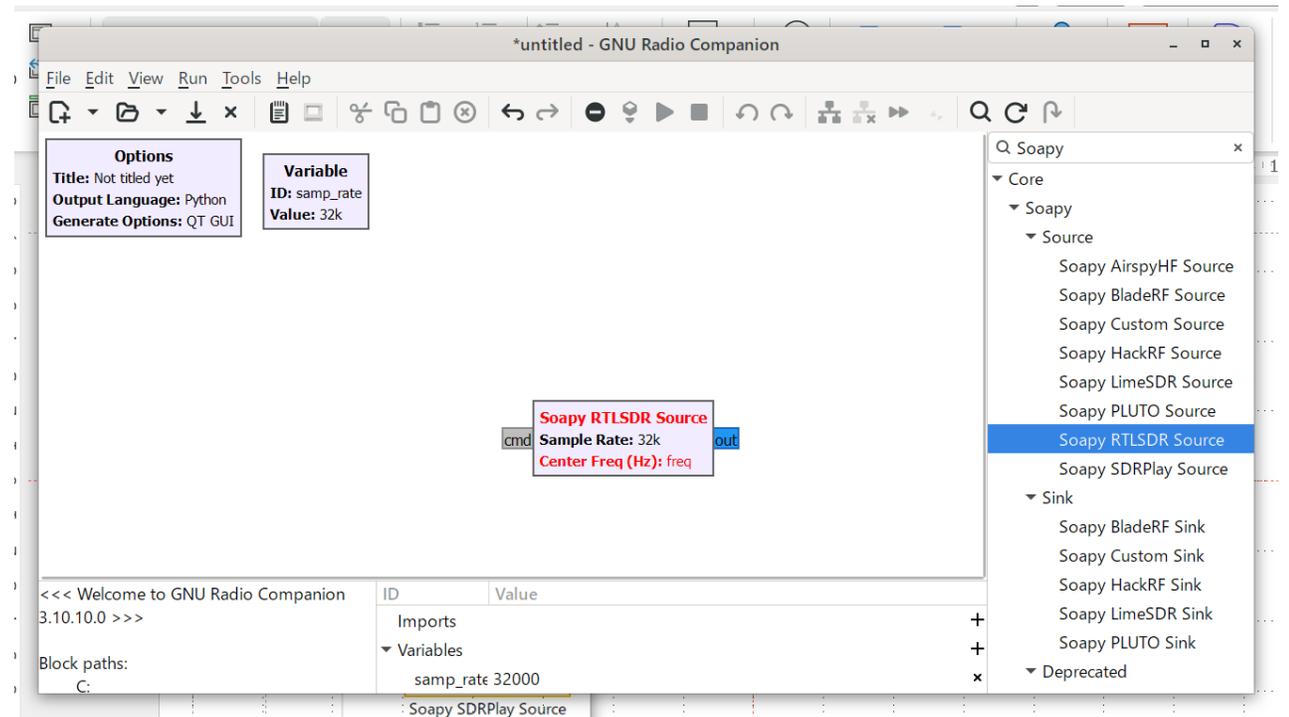
起動画面はこんな感じ

RTL SDRを追加



虫眼鏡(検索)を左クリック

Soapyを入力



変数の追加

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

- Options Panel:**
 - Title: Not titled yet
 - Output Language: Python
 - Generate Options: QT GUI
- Variable Panels:**
 - Variable ID: samp_rate, Value: 32k
 - Variable ID: freq, Value: 80M
- Block Diagram:** A 'Soapy RTLSDR Source' block is shown with 'Sample Rate: 32k' and 'Center Freq (Hz): 80M'. A yellow arrow points from the 'freq' variable to the 'Center Freq' field.
- Search Window:** A search window titled 'Q Soapy' is open, showing a tree view of source blocks. 'Soapy RTLSDR Source' is highlighted in blue. A yellow arrow points from the search window to the 'freq' variable in the table below.
- Table:**

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	32000
- Terminal:** Shows the command path and execution of 'test_fm.grc'.

Annotations in Japanese:

- ダブルクリックして「freq」に変更 (Double-click to change to 'freq')
- 左クリックして変数を追加 (Left-click to add variable)
- ダブルクリックして「80e6」を入力 (自動的に80000000.0になる) (Double-click to enter '80e6' (automatically becomes 80000000.0))

Low pass filterの追加

The screenshot shows the GNU Radio Companion (GRC) interface with a signal flow graph containing a **Soapy RTLSDR Source** block and a **Low Pass Filter** block. The **Low Pass Filter** block is highlighted in blue. A search bar on the right contains the text "low pass", and the search results show "Low Pass Filter" selected under the "Filters" category. A yellow arrow points from the search bar to the selected block, and another yellow arrow points from the selected block to the text "枠内にドラッグアンドドロップ".

「low pass」を検索

枠内にドラッグ
アンドドロップ

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

Variable
ID: samp_rate
Value: 32k

Variable
ID: freq
Value: 80M

Soapy RTLSDR Source
Sample Rate: 32k
Center Freq (Hz): 80M

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 32k
Cutoff Freq:
Transition Width:
Window: Hamming
Beta: 6.76

C:
¥Users¥user¥radioconda¥Library¥share¥g -
nuradio¥grc¥blocks
Loading: "C:
¥Users¥user¥Desktop¥sdr¥test_fm.grc"
>>> Done

ID	Value	
Imports		+
Variables		+
freq	80000000.0	x
samp_rate	32000	x

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:** 1
- 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

At the bottom, there is a message: "Param - Transition Width(width): Expression None is invalid for type 'real!'".

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

SourceとLow pass filterの接続

*untitled - GNU Radio Companion

File Edit View Run Tools Help

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

Variable
ID: samp_rate
Value: 32k

Variable
ID: freq
Value: 80M

Inまでドラッグする

Outで左クリック

Soapy RTLSDR Source
Sample Rate: 32k
Center Freq (Hz): 80M

Low Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 32k
Cutoff Freq: 100k
Transition Width: 30k
Window: Hamming
Beta: 6.76

Search: low pass

- Core
 - Filters
 - FFT Low Pass Filter
 - Low Pass Filter

C:
%Users%user%radioconda%Library%share%g -
nuradio%grc%blocks

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

ID	Value	
Imports		+
Variables		+
freq	80000000.0	x
samp_rate	32000	x

WBFM receiverブロックの配置

*untitled - GNU Radio Companion

File Edit View Run Tools Help

Language: Python
Options: QT GUI

ID: samp_rate
Value: 32k

ID: freq
Value: 80M

FMを入力

OutからInに配線をつなぐ

Low Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 32k
Cutoff Freq: 100k
Transition Width: 30k
Window: Hamming
Beta: 6.76

WBFM Receive
Quadrature Rate:
Audio Decimation:

FM

Core

- Modulators
 - FM Deemphasis
 - FM Demod
 - FM Preemphasis
 - NBFM Receive
 - NBFM Transmit
 - WBFM Receive**
 - WBFM Receive PLL
 - WBFM Transmit

枠内にドラッグ
アンドドロップ

C:

```

%Users%user%radioconda%Library%share%g -
nradio%grc%blocks
Loading: "C:
%Users%user%Desktop%sdr%test_fm.grc"
>>> Done

```

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	32000

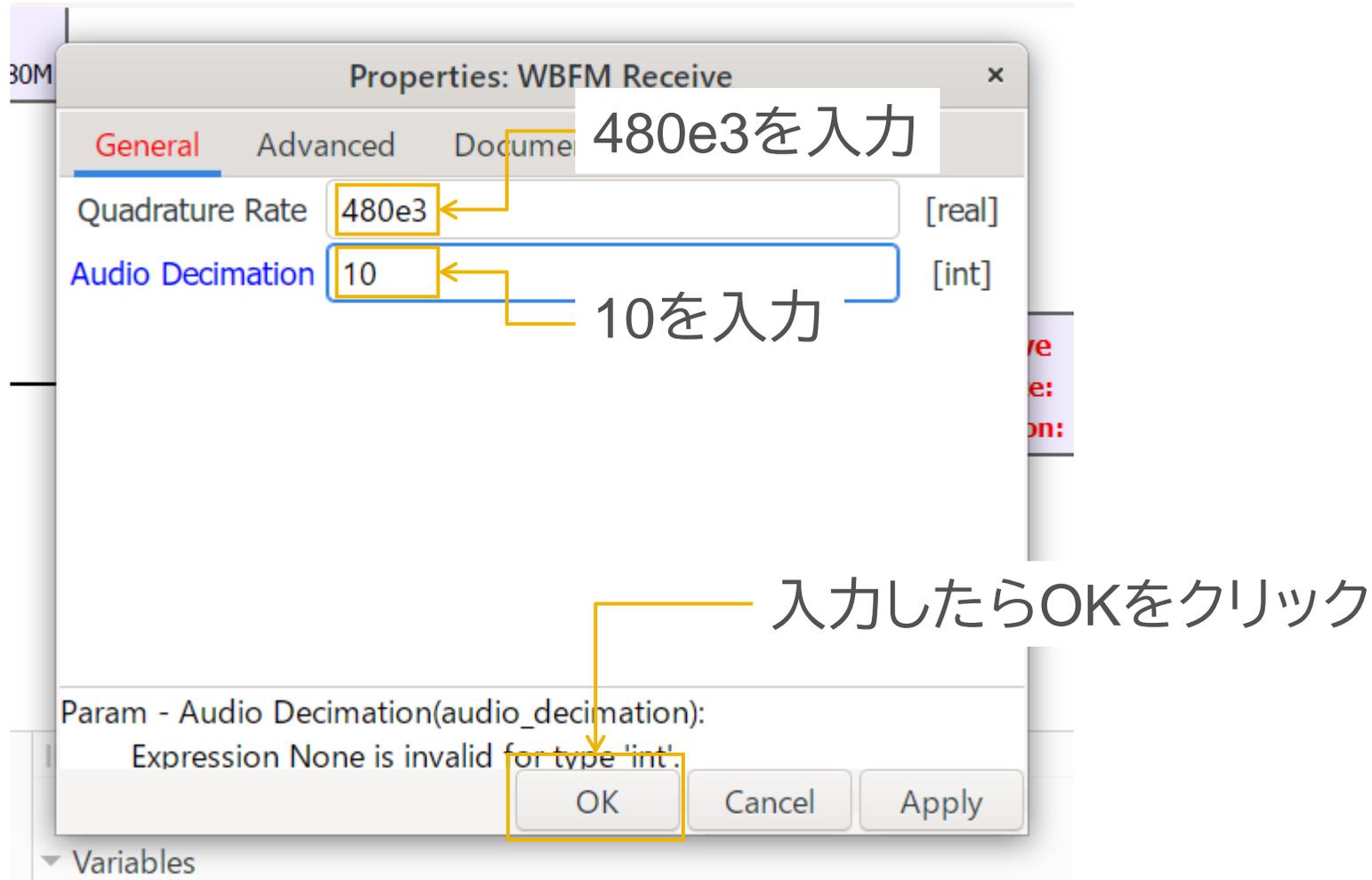
ns2/3/4 Sink

ns2/3/4 Source

- + FM Detector
- x RADAR
- x Estimators
 - Estimator FMCW
- Math Operators

WBFM receiverブロックの設定

WBFM receiverをダブルクリックして設定を開く



Audio Sinkの接続

GNU Radio Companion (GRC) のスクリーンショット。音声シンクの接続方法を説明しています。

auを入力

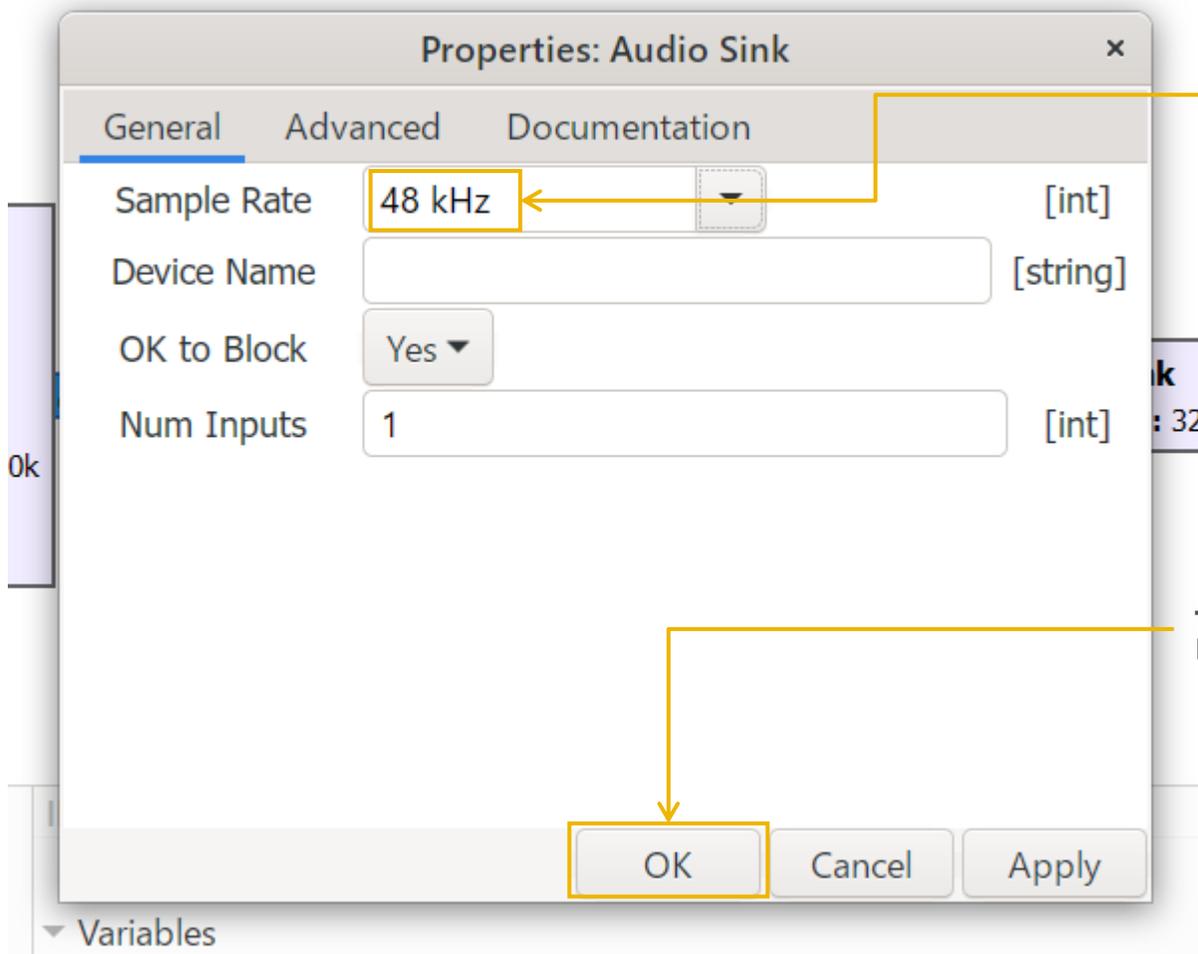
枠内にドラッグ
アンドドロップ

OutからInに配線をつなぐ

図は、GNU Radio Companion (GRC) のインターフェースを示しています。中央には、音声処理のブロックチェーンが描かれています。左から「Low Pass Filter」ブロックがあり、その出力（out）は「WBFM Receive」ブロックの入力（in）に接続されています。「WBFM Receive」ブロックの出力（out）は、「Audio Sink」ブロックの入力（in）に接続されています。右側のコンポーネントパネルには、「au」と検索し、「Audio Sink」が選択されています。下部には、変数設定のウィンドウがあり、「freq」が 80000000.0、「samp_rate」が 32000 に設定されています。

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	32000

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

QT GUI Time sinkの接続

QT GUI Time sinkの接続方法を示すスクリーンショット。GNU Radio Companionのインターフェースが示されています。

ブロック図には、以下のブロックが接続されています:

- Low Pass Filter**: Decimation: 5, Gain: 1, Sample Rate: 32k, Cutoff Freq: 100k, Transition Width: 30k, Window: Hamming, Beta: 6.76
- WBFM Receive**: Quadrature Rate: 480k, Audio Decimation: 10
- Audio Sink**: Sample Rate: 48 kHz
- QT GUI Time Sink**: Number of Points: 1.024k, Sample Rate: 32k, Autoscale: No

接続の流れは、入力からLow Pass Filterのinポートへ、そのoutポートからWBFM Receiveのinポートへ、そしてWBFM ReceiveのoutポートからAudio Sinkのinポートへと続きます。また、Low Pass FilterのoutポートからQT GUI Time Sinkのinポートへ接続されています。

検索メニューには「QT」が入力されており、検索結果として「QT GUI Time Sink」が選択されています。

下部のコンソールには、ブロックのロード状況が示されています:

```
Block paths:
  C:
  %Users%user%radioconda%Library%share%g -
  nuradio%grc%blocks
Loading: "C:
%Users%user%Desktop%sdr%test_fm.grc"
>>> Done
```

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	32000

注釈: OutからInに配線をつなぐ

注釈: 枠内にドラッグアンドドロップ

注釈: QTを入力

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の接続

The screenshot shows the GNU Radio Companion interface with a signal flow graph and a search menu. The signal flow graph consists of the following blocks:

- Soapy RTLSDR Source**: Sample Rate: 32k, Center Freq (Hz): 80M
- Low Pass Filter**: Decimation: 5, Gain: 1, Sample Rate: 32k, Cutoff Freq: 100k, Transition Width: 30k, Window: Hamming, Beta: 6.76
- WBFM Receive**: Quadrature Rate: 480k, Audio Decimation: 10
- Audio Sink**: Sample Rate: 48 kHz
- QT GUI Time Sink**: Number of Points: 1.024k, Sample Rate: 32k, Autoscale: No
- QT GUI Frequency Sink**: FFT Size: 1024, Center Frequency (Hz): 0, Bandwidth (Hz): 32k

The search menu on the right shows the following items:

- QT
- Core
- Instrumentation
- QT
 - fosphor sink (Qt)
 - QT GUI Bercurve Sink
 - QT GUI Constellation Sink
 - QT GUI Eye Sink
 - QT GUI Frequency Sink** (highlighted)
 - QT GUI Histogram Sink
 - QT GUI Lumber Sink
 - ink
 - ime Raster Sink
 - ime Sink
 - ector Sink
 - QT GUI Waterfall Sink
- GUI Widgets
 - QT
 - QT GUI App Background
 - QT GUI Fast Auto-Correlate
 - QT GUI Az-El Plot
 - QT GUI Check Box

A yellow arrow points from the highlighted 'QT GUI Frequency Sink' in the search menu to the corresponding block in the signal flow graph.

Block paths:

```
C:
¥Users¥user¥radioconda¥Library¥share¥g -
nuradio¥grc¥blocks

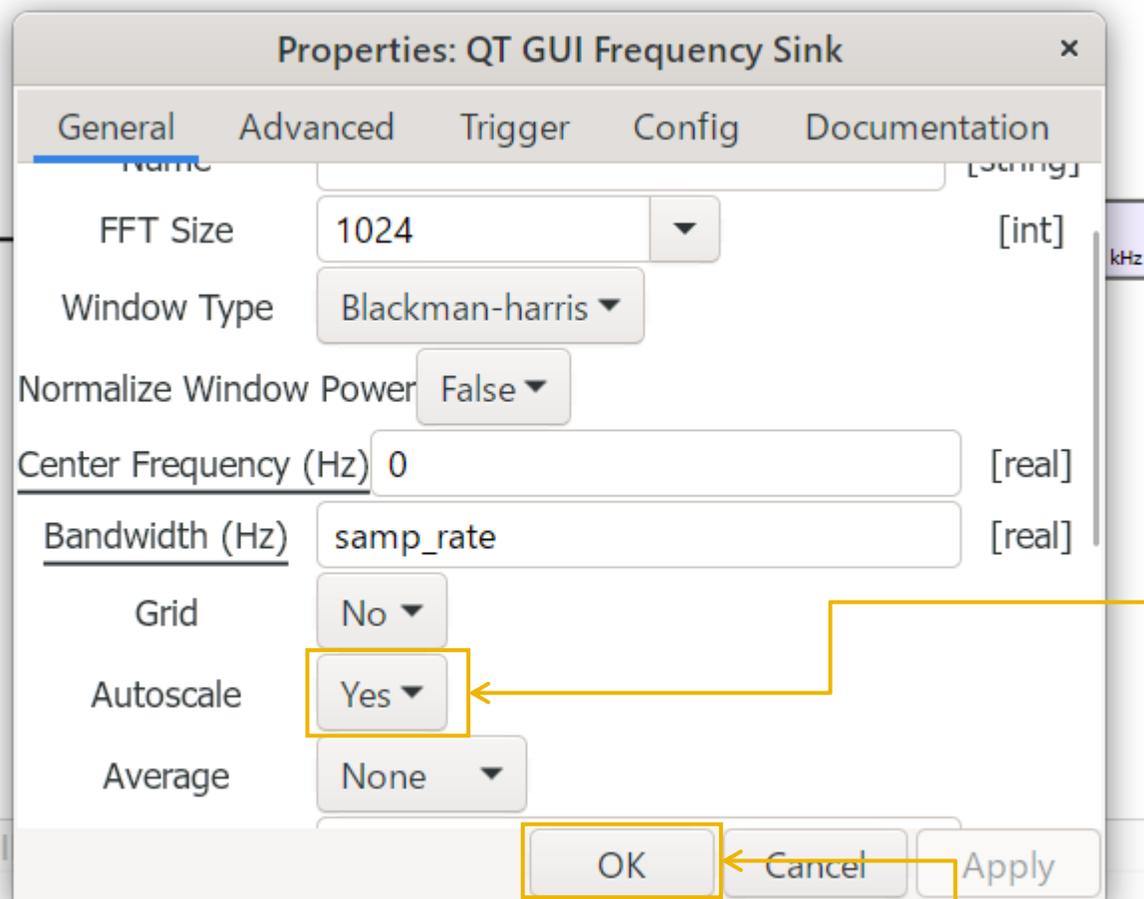
Loading: "C:
¥Users¥user¥Desktop¥sdr¥test_fm.grc"
>>> Done
```

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	32000.0

枠内にドラッグ
アンドドロップ

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

samp_rateの設定

ID	Value	
Imports		+
▼ Variables		+
freq	80000000.0	x
samp_rate	32000.0	x

この欄を2.4e6と入力する
(自動的に2400000.0と変換される)



ID	Value	
Imports		+
▼ Variables		+
freq	80000000.0	x
samp_rate	2400000.0	x

ラジオの起動

*fm_only_TokyoFM.grc - C:\Users\user\Desktop\sdr

File Edit View Run Tools Help

Options
Title: Not titled yet
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

WBFM Receive
Quadrature Rate: 480k
Audio Decimation: 10

Audio Sink
Sample Rate: 48 kHz

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

Core

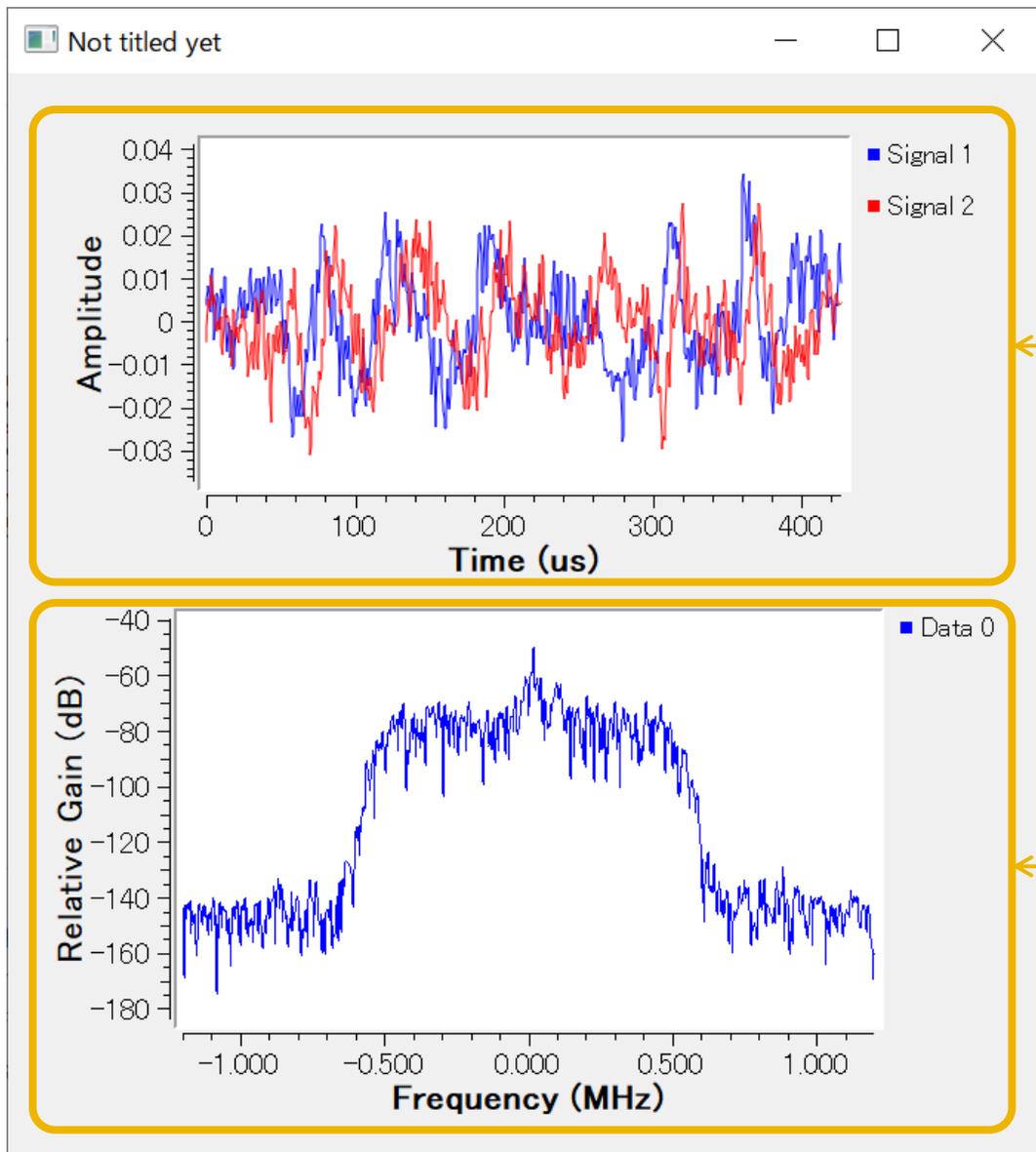
- ADALM-2000
- Audio
- Boolean Operators
- Byte Operators
- Channel Models
- Channelizers
- Coding
- Control Port
- Debug Tools
- Deprecated
- Digital Television
- Equalizers
- Error Coding
- File Operators
- Filters
- Fourier Analysis
- GUI Widgets
- Impairment Models

performance. [0]m
Found Fitipower FC0013 tuner
G [INFO] Opening Generic RTL2832U ::
7777111153705700...
Found Fitipower FC0013 tuner
[INFO] Using format CF32.
>>> Done

ID	Value
Imports	
Variables	
freq	80000000.0
samp_rate	2400000.0

左クリックするとラジオが起動する

画面



IQの時系列信号

IFのスペクトラム

この状態でスピーカーからは
Tokyo FM (80.0MHz)が聞こえる...はず。

次回以降では…

- ブロック線図それぞれがどのような役割をしているのか？
- それぞれのブロックの中身は何をしているのか？
- 現代の無線機を構成する基本技術について
- WBFM receiverブロックを使わずにプリミティブなブロックのみでAM、FM受信機を作成