

ものづくりやろう!

## 第四回 音声周波数帯信号のバンドパスフィルタの製作(2)

JH3RGD 葭谷安正

---

### ■はじめに

前回に引き続き今回も GNURadio について解説していきます。多少前回解説と重複しますがご容赦ください。

GNURadio は非常にたくさんの処理機能をもっていますが、この解説で GNURadio のすべてを解説することはできません(わたしの力量もすべてを解説できるほどありません)。またどんなことができるのかを一から説明しては学習効率が非常に悪くなってしまいます。そこで、この解説記事では GNURadio のチュートリアルに記載されている簡単なフローグラフ例を使い、その中で使われているブロックの解説を中心に説明していきます。これを発展させた例を示して GNURadio の一部の機能を紹介していきたいと思います。

GNURadio というのは、前回の記事でも記載しましたように「ソフトウェア無線の信号処理系」のことです。もっと具体的にいうと、「高周波信号の発生、フィルタ、変調や復調、増幅などの信号処理、処理信号のオシロスコープ状の表示や FFT 表示、ウォーターフォール表示などの信号表示機能をコンピュータ上で実現するソフトウェア」です。こんなにいろいろなことができるのですが無償で利用することができます。この GNURadio を使うためには Python 言語や C 言語の知識、無線の知識プログラミングの知識や経験が必要となりますが、このままでは非常にハードルが高いソフトウェアです。私も尻込みします。しかしこの GNURadio をそのまま使うのではなく、あるツールを使うことでプログラミングが簡単にできるようになります。そのツールというのが GNURadio の機能をグラフィックユーザーインターフェース(GUI)で提供し、プログラミングを補助してくれる[GNURadio Companion]、略して[GRC]です。GRC を使えば、GNURadio を使うのに必要な Python 言語や C 言語の知識は必要ありません。あらかじめ GNURadio で準備されている機能ブロックをマウスを使って並べ、このブロックの間を結線することで信号処理の様々な機能を実現することができます。この解説では以下で [GNURadio]と記載していますが、正確には GRC のことを指していると思ってください。

GRC を使って簡単な信号処理プログラムを作っていきます。ここで大切なことです。GNURadio を使えばさまざまな信号処理をパソコン上で行えますが、パソコンへのアナログ信号の読み込みや、GNURadio で処理された信号をパソコン外部に入出力するためには、GNURadio に対応したハードウェアが必要です。例えば、この解説記事の中でバンドパスフィルタを GNURadio で作成しますが、無線機から出力されたアナログ信号を GNURadio で処理するためにはデジタル信号に変換するためのハードウェアがパソコンに装備されていなければなりません。幸いほとんどのパソコンにはサウンドボードが実装されていますので、このサウンドボードを使って処理したいアナログ信号をデジタル信号に変換し、GNURadio で処理させることができます。サウンドボードは音声帯域周波数の信号を入出力することができますが、高周波信号の入出力はできません。GNURadio では AM や FM などの高周波信

号の変調やその復調などの処理もできます。しかしパソコン内部で GNURadio を使って変調処理を施した信号を電波として飛ばすためにはデジタル信号をアナログ信号に変換するハードウェア(D/A 変換装置)や電力増幅器などのハードウェアが必要になってきます。その代表例が[HackRF One]や[USRP]他何種類かあるようです。これらのハードウェアのことを SDR フロントエンドと呼んでいます。無銭家の私にはどちらも少し値が張りますし縁がありませんでした。また、これを使って電波を飛ばすとすると技適に適合しているのかも考慮する必要があるのかもしれない。幸い、電波を発射せず受信だけに絞れば安価な dongle タイプの広帯域受信機の RTL-SDR が GNURadio で使えます。GNURadio の存在を初めて知ったころに、“ジャングル”で RTL-SDR を購入し、それを GNURadio のフロントエンドとして使ってみました。ネット上に掲載されている情報を参考に HF(コンバータが必要でした)、VHF、UHF 帯の受信機を見よう見まねで作成しましたが、音が出てきたときには感激しました(中学生の頃、真空管の並四ラジオを作成したときほどは感激しませんでした Hi)。

## ■GNURadio の起動、開発環境

GNURadio の起動はパソコンの[スタートボタン]を押して、図 1 のよう[GNURadio Companion]を選択、クリックします。



図 1 [GNURadio companion]を選択

コンソール上を処理が流れ、その後図 2 のようにウィンドウが表示されます。

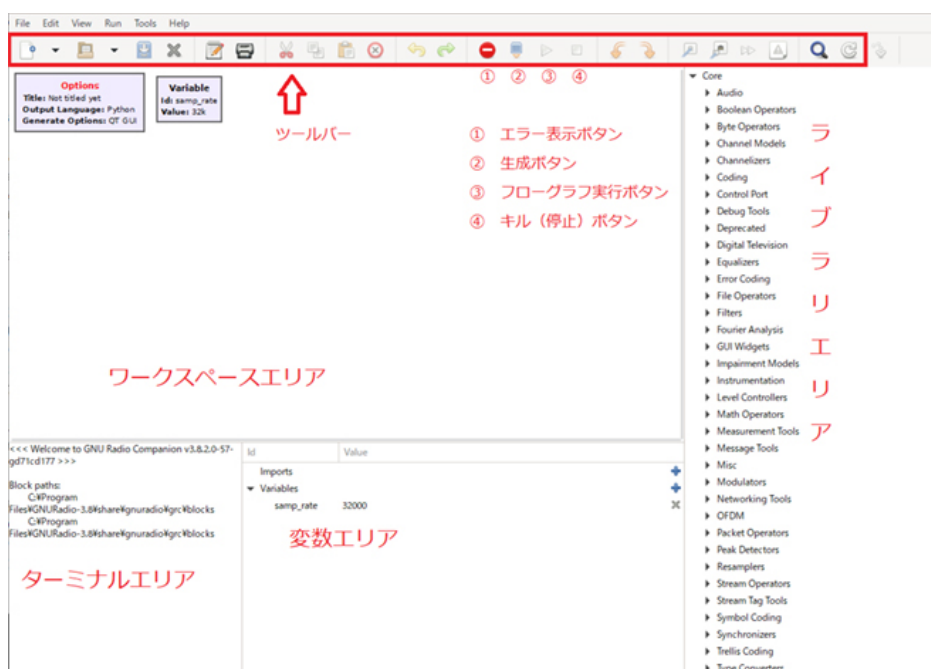


図 2 起動時のウィンドウ: オープニング画面

ウィンドウの各部の役割を記載します。

### (1) ツールバー

ツールバーは、ファイルのリードやセーブ、フローグラフから Python コードを出力(生成ボタン)したり、プログラム実行、停止などの操作を実行します。

### (2) ワークスペース

ワークスペースは、私たちが信号を処理するのに必要なソフトウェアを作成するために、ライブラリからブロックをもってきて並べ、それらのパラメータを変更してブロックを線で結び、処理の流れを表現するためのエリアです。このエリアに書いた図をフローグラフといいます。フローグラフを書いた後、ツールバーの生成ボタンを押すと、パラメータの不足や処理上のエラーがなければその処理を実行するための Python 言語のコードが自動生成されます。プログラムの実行するためには、ツールバー上のプログラム実行ボタンを押すことで処理プログラムが動きます。ライブラリ上のブロックをワークスペースに置くためには、ライブラリからブロックを探し、そのブロックの名前をクリックしてワークスペースにドラッグするか、または名前をダブルクリックしてワークスペースに自動的に配置します。

### (3) ライブラリ

ライブラリには、GRC にインストールされているたくさんの機能がブロックとして格納されています。GNURadio でフローグラフを書くためにはどんな部品がなんという名前でライブラリに入っているのかわかる必要があります。初めて触るときはまったくわかりませんが、GNURadio のチュートリアル の例を自分で入力するとだんだんとわかってきます。

しかしあまり使わないブロックや初めて使うブロックはどこにあるのかわかりません。このようなときはライブラリエリアの上にある虫眼鏡のアイコンをクリックするか Ctrl + f を入力すると、キーワードを入力エリアが開きますので、キーワードを入力することでそのキーワードに関するブロックが表示されます(図 3 ライブラリの検索)。

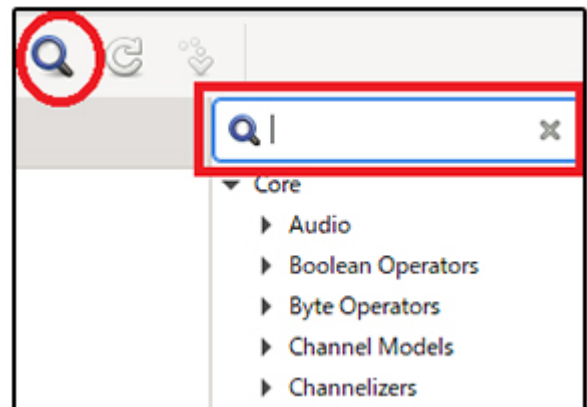


図 3 ライブラリの検索

部品を見つけるときに覚えておくと比較的早く部品を見つけることができる単語は[Source]と[Sink]です。信号発生器などの信号源は出力端子のみを持っていますが、このようなブロックは[Source]というカテゴリに属します。電気回路のブロック図で普通左端に書かれることが多い装置ですね。一方、オシロスコープなど出力装置や表示装置、外部端子への出力などは[Sink]と検索エリアに入力すると見つけることができます。電気回路のブロック図で普通右端に書かれることが多い計測器類が[Sink]に相当しますね。試しに、検索エリアに[Source]と入力しました。図 4 のように 20 件程見つかりました。この中に[Audio Source]というのが見えます。コンピュータに音声信号を入力するときに音声入力端子を使用しますが、この[Audio Source]をワークスペースエリアに配置し、使用するデバイス名などのパラメータ設定することで具体的な入力端子を指定することができます。

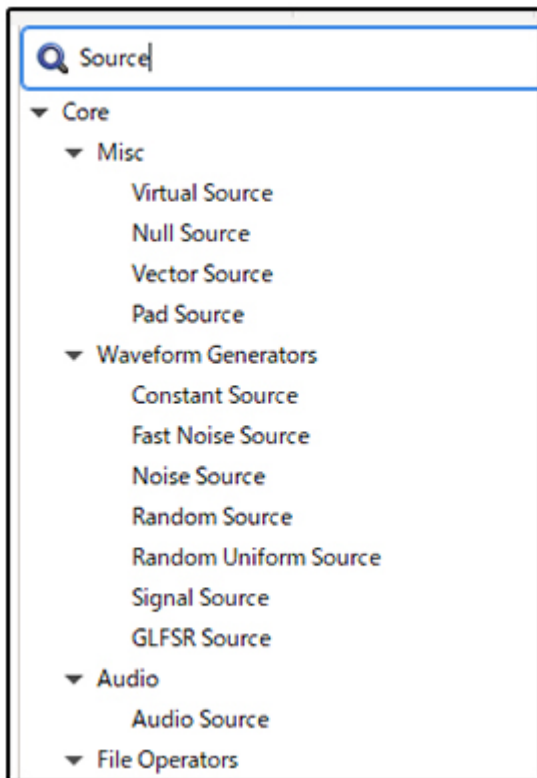


図 4 Source ブロックの検索結果

#### (4) ターミナルエリア

ターミナルエリアはファイルパス情報やエラー情報などが表示される部分です。この部分进行操作することはありません。

#### (5) 変数エリア

変数エリアでは、ワークスペースに置かれた変数ブロックの値や変数名の変更、削除などをこの場所で行うことができます。ワークスペース上に置いた変数ブロックをダブルクリックして変更することもできます。

それでは以下の例をもとに GNURadio で信号処理のプログラムを作成しましょう。

## ■フローグラフ例 1

### GNURadio 内での信号発生とその表示

---

[処理概要]: GNURadio 内で 1kHz の正弦波を発生させ、発生させた信号をそのまま GNURadio 内のオシロスコープで表示させます。

(1) 外部からの入力: なし

(2) 外部への出力: なし

(3) ソフトウェアでの処理:

a. GNURadio で 1kHz の正弦波信号を発生

b. GNURadio で発生した a.の信号をオシロスコープのような表示ブロックを使用して表示

---

この例はパソコン外部からの入力や外部への出力のない、GNURadio 内で完結した処理です。

[作成手順]

(1) GNURadio を起動

図 5 のように[スタート]-[GNURadio Companion]を押して起動します。

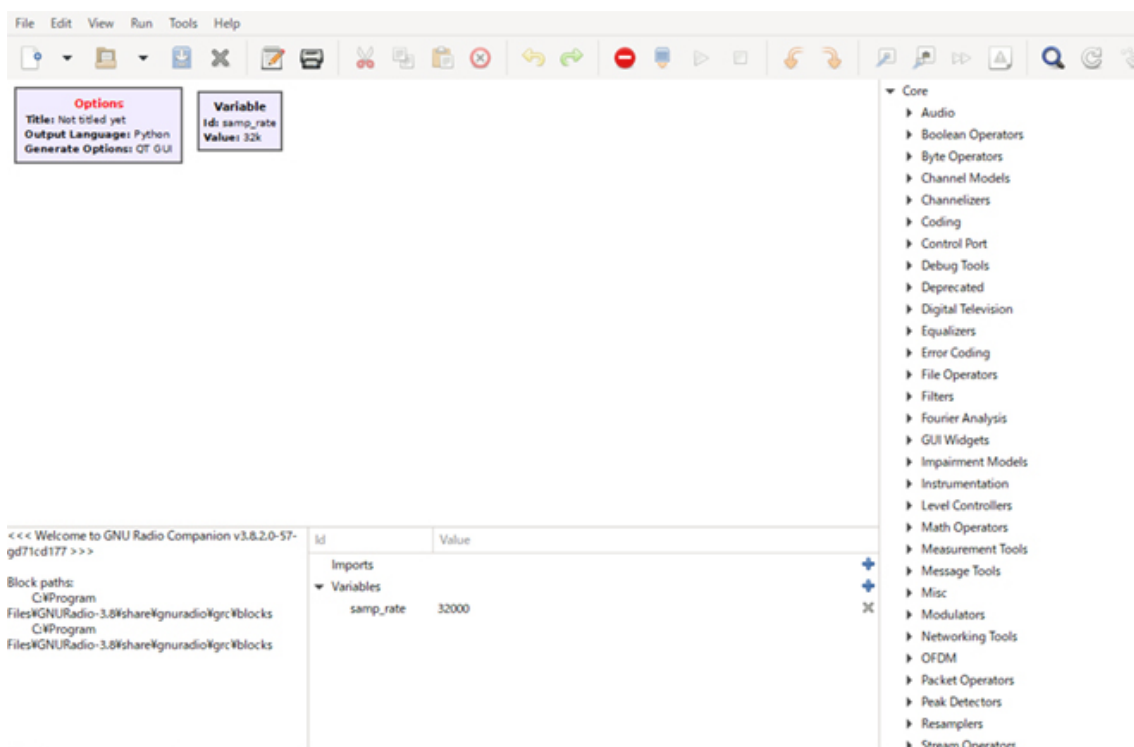


図 5 起動時画面

過去に GNURadio を起動したことがあれば図 5 のようには表示されず、最後に編集したフローグラフが表示されることがあります。

(2) ブロック配置: 起動時画面(図 5)のワークスペースに[Signal Source]と[QT GUI Time Sink]の 2 つのブロックを配置します。GNURadio を起動時に 2 個のブロックが自動的に設定されています。1 つは、[Options]ブロック、もう 1 つは[variable]ブロック(変数ブロック)です。[Options]ブロックのタイトルの色が赤色になっていることに注意してください。後ほど記載しますが、パラメータが不足(またはエラー)していることを表しています。具体的には[id]番号をいれなければなりません。

この例で作成したい内容は、GNURadio 内部で 1kHz の信号を発生させその信号をそのままオシロスコープに表示するわけです。信号発生に必要な 1 つめのブロックは信号発生器です。図 5 の右にあるライブラリエリアからシグナルジェネレーターを持ってきます。シグナルジェネレーターはどこにあるのかわからない場合は、ライブラリエリア上部にある虫眼鏡印をクリックして[Signal]と入れて検索します。この単語を含むブロックが出てきますのでその中から選びます。今回は[Waveform Generators]内の[Signal Source]ブロックを選択します。たくさんのブロックがありますので、どのブロックがどういう働きをするのかを分かるまでは一体どれを使えばいいのか分からなくなりますが、それは徐々に学んでください。

ブロックを並べて置いていきます。もし位置を変更したい場合にはそのブロックをクリックして移動したい位置までドラッグします。

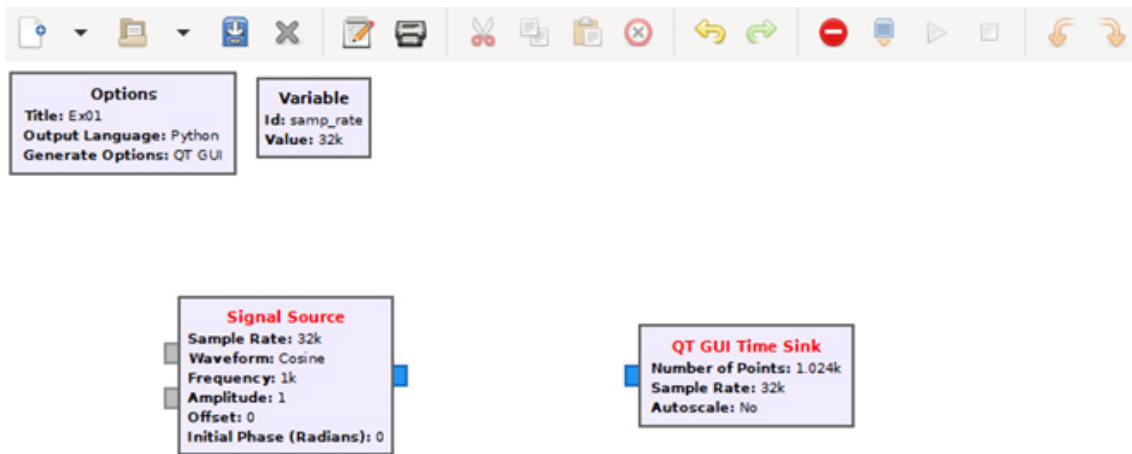


図 6 ブロックの配置

(3) 結線: [Signal Source]ブロックの右端 out 端子と[QT GUI Time Sink]ブロックの左端 in 端子を順番にクリックすると結線されます。結線を間違ったときは、削除したい結線をクリックすると青色になって編集状態になりますので、キーボードの[Delete]ボタンを押します。

(4) パラメータ設定: この例ではすでに設定されている値(default 値)をそのまま使用します。

出来上がったフローグラフを図 7 に示します。

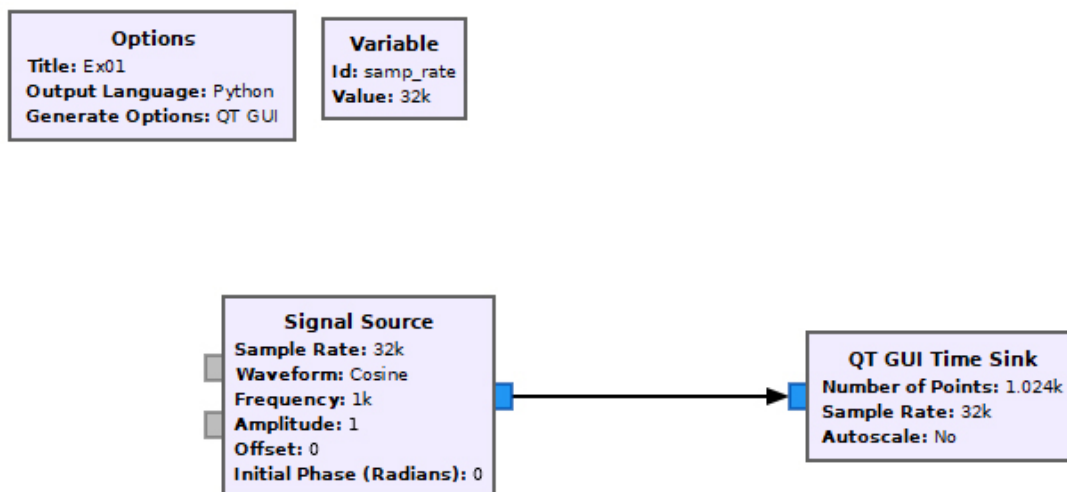


図 7 ブロック結線

(5) フローグラフの保存: 出来上がったフローグラフを保存します。

[File]-[Save As]を選び、ファイル名を入力します。ファイル名の拡張子は[grc]です。格納場所に[c:¥]などを選択すると左下の[ターミナル]エリアに“Error: Cannot save: C:¥Ex01.grc”とセーブできない旨のエラーメッセージが出て書き込めていないことがあります。確認しておいてください。

(6) 実行: [Generate]ボタンを押すとフローグラフから Python のコードを作成します。

図 7 のフローグラフを作成後[Generate]ボタンを押すと[ターミナル]エリアにエラーメッセージが発生しています。

エラーの内容を日本語に翻訳すると以下のような内容です。

「警告: このフローグラフにはフロー制御がない可能性があります。オーディオまたは RF ハードウェアブロックが見つかりません。CPU の過負荷を回避するために、フローグラフに Misc-> Throttle ブロックを追加します。」

[Throttle]ブロックを入れたほうがいいですよ、との警告です。GNURadio のマニュアルで[Throttle]を調べてみると、

「スロットルブロックは、フローグラフにレート制限ブロック(通常はハードウェア(SDR、スピーカ、マイクなど))が含まれていない場合にのみ使用する必要があります。サンプルのレートを正確に制御することは意図されておらず、効果的でもありません。これは、サンプルクロックに関連付けられたソースまたはシンクによって制御する必要があります。例えば、USRP またはオーディオカード。Throttle Block は通常は、ハードウェア以外のソースブロック(Signal Source など)の出力に直接接続され、そのソースブロックがサンプルを作成する速度を制限します。」

とのこと。要は、[Throttle]ブロックを入れて信号の流れが指定されたレート(32k: 1 秒間に 32k のデータを流す)で流れるようにすれば CPU に過負荷がかからないとのこと。警告メッセージにしたがって[Throttle]を入れたものが図 8 です。

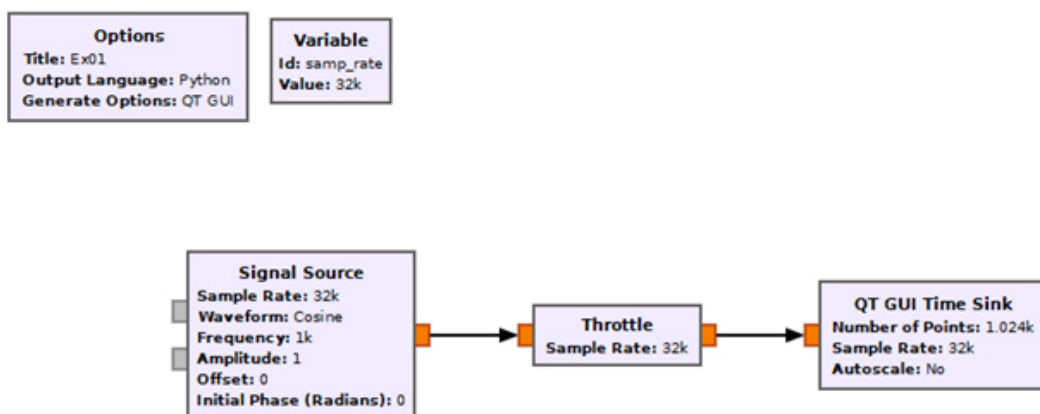


図 8 完成したフローグラフ

ファイルセーブ後、[Generate]ボタンを押すと Python のコードがセーブされ、[ターミナル]エリアには Python のコードの出力先情報が表示されます。

```
Generating: 'D:\GRC\Ex01.py'
```

さらに,[Execute]ボタンを押すと

```
Executing: C:\Program Files\GNURadio-3.8\tools\python3\python.exe -u
```

```
D:\GRC\Ex01.py
```

```
gr::pagesize: no info; setting pagesize = 4096
```

```
>>> Done (return code 1)
```

と実行した旨の表示がされました。なお、Python のコードが出力されるとそのコードを Python プログラムから実行することもできます。実行結果は図 9 のように、信号波形が表示されます。

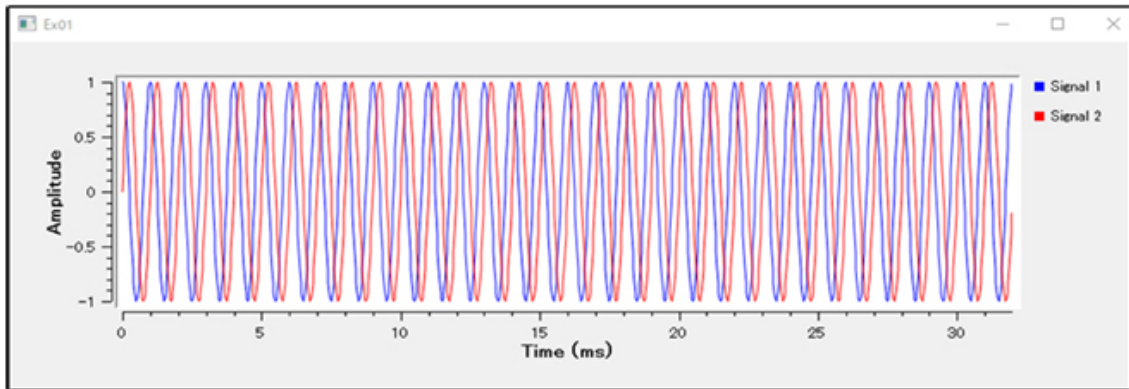


図 9 実行結果

実行の停止は、[Execute] ボタンの右にある [Kill] ボタンを押すことで実行しているプログラムを停止させることができます。

[実行結果について]

フローグラフ例 1 の実行結果(図 9)を見てみましょう。このブロックは、GNURadio 内で 1kHz の正弦波を発生させ、発生させた信号をそのまま GNURadio 内のオシロスコープで表示させる処理です。1kHz の正弦波を 1 つしか発生させていないのに図 9 では正弦波が 2 つ見えます。これは作成したブロックの信号をすべて複素数として処理したためです。フローグラフの [Signal Source] ブロックをダブルクリックしてパラメータがどのように設定されているのかを確認します。図 10 のように [Output Type] が [complex] になっています。これを [float] に変更します。ほかの [Throttle] ブロック、[QT GUI Time Sink] ブロックについても [Type] パラメータを [float] に変更します。

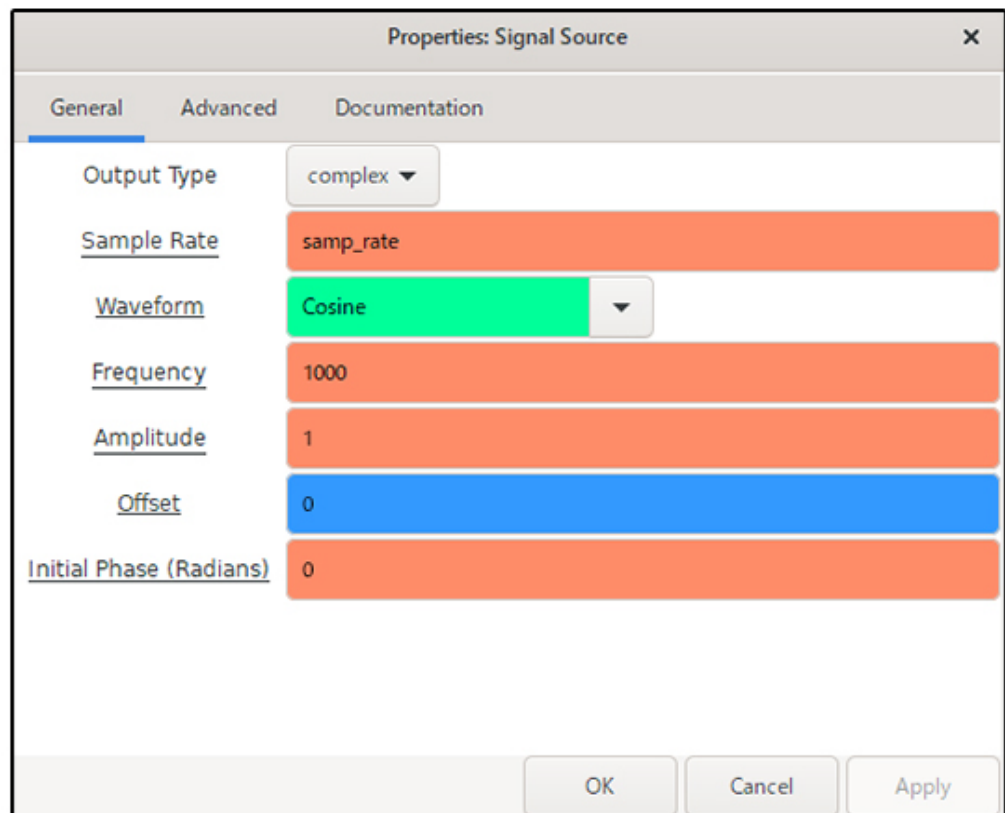


図 10 パラメータ設定内容



パラメータ変更後,再度実行すると図 11 のように正弦波が一つになります。

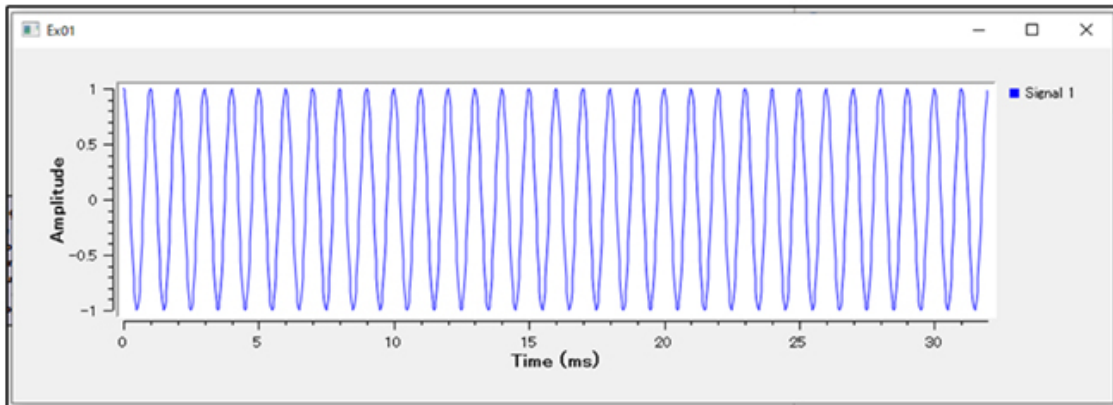


図 11 実行結果(データタイプ float の場合)

#### [エラーについて]

- ブロックタイトルの色によるエラー箇所の表示

結線前と結線後のブロックタイトルの色に変化があります。結線前はブロック内の文字がどちらも赤色になっています。一方、結線後は両ブロック名とも黒色に変わっています。これは、赤色のときはそのブロックに対してのパラメータ設定や結線などがまだ不足している時、エラーの時です。同時に[ツールバー]上の[エラー表示ボタン]も赤色になっています。しかし結線が行われるとブロックタイトルは黒色になりますが、[エラー表示ボタン]は赤色のままです。

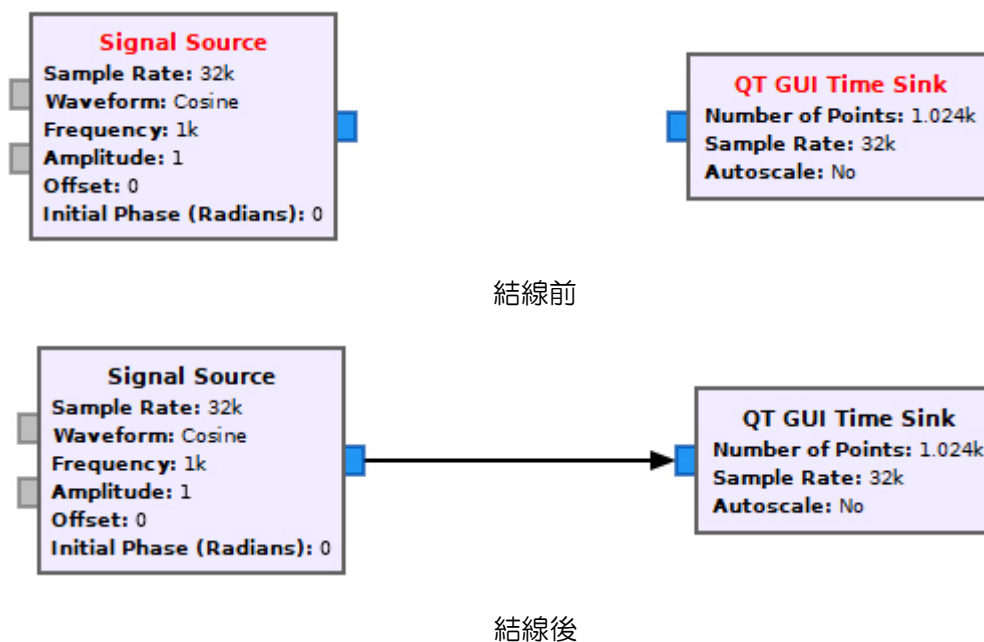


図 12 ブロックの結線の前後

どこにエラーがあるのかというと、図 13 のように左上部にある[Options]ブロックのブロックタイトル文字が赤字になっています。このブロックをダブルクリックして、その中の赤字になっているパラメータ[id]に(図 14 参照)、例えば[Ex01]と入力するとエラーが消えて赤字の[id]が黒字になります。このように単純なエラーはブロックのタイトル文字の色を見ることで判別できます。

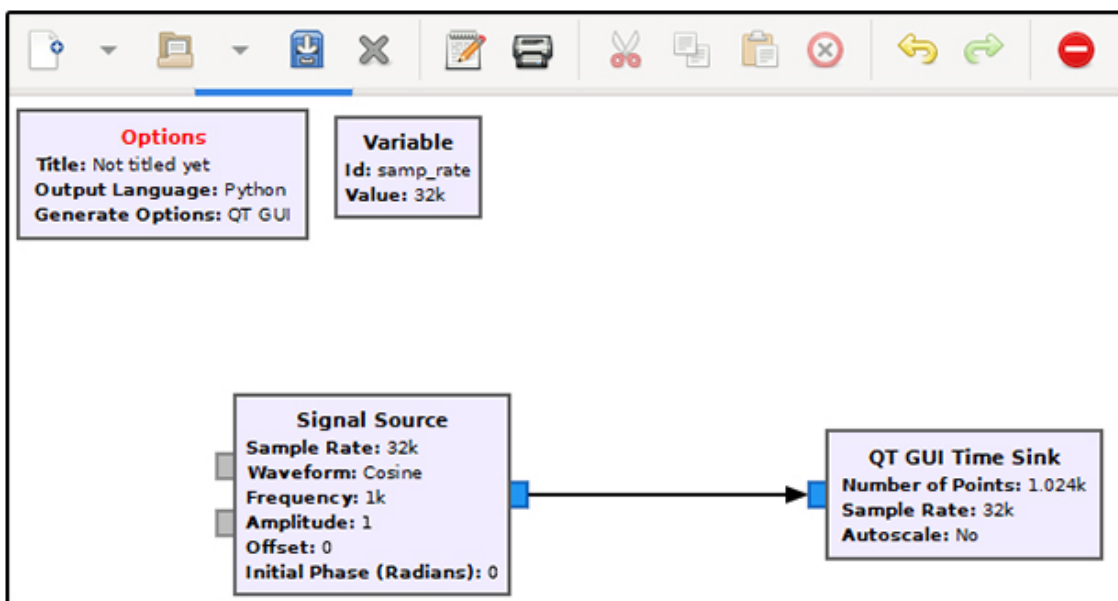


図 13 [Options]ブロックのエラー

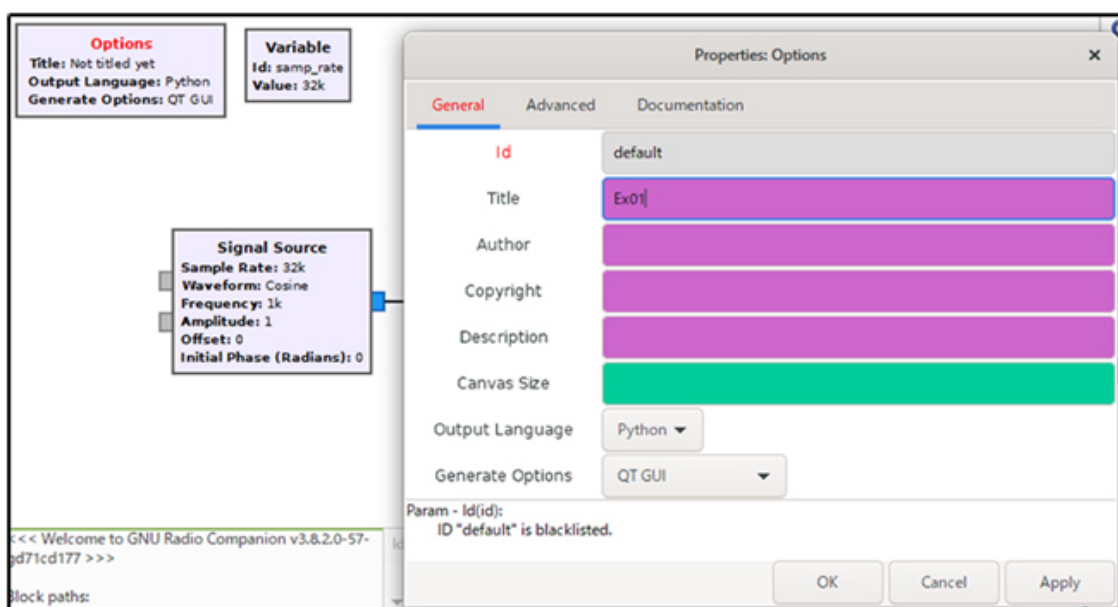


図 14 パラメータ[id]がエラー箇所

### [データタイプ不一致によるエラー]

エラーの原因として、データタイプの不一致もよく起こります。信号処理の分野では複素数[complex]や実数[float]が使われますが、GNURadio で使用されている処理ブロックの default 値が複素数や実数が混在しており、ブロックの入出力のデータタイプが異なることに気づかずに接続してエラーが発生することがあります。このような場合矢印の色が赤にかわっていますので、色を頼りにエラー箇所を見つけてください。図 15 は上記例の[Throttle]ブロックのデータタイプを[complex]タイプに変換した場合のエラー状況を示しています。矢印が赤色になっています。ツールバーにある[エラー表示ボタン]も色濃くなっているので、[エラー表示ボタン]をクリックすると下図のように[Errors and Warnings]ウィンドウが表示されエラー箇所が表示されます。[Throttle]ブロックを中心に入出力側にある 2 つのブロックでサイズが整合していないとのメッセージがでています。これらの情報を頼りにエラー修正を行っていきます。

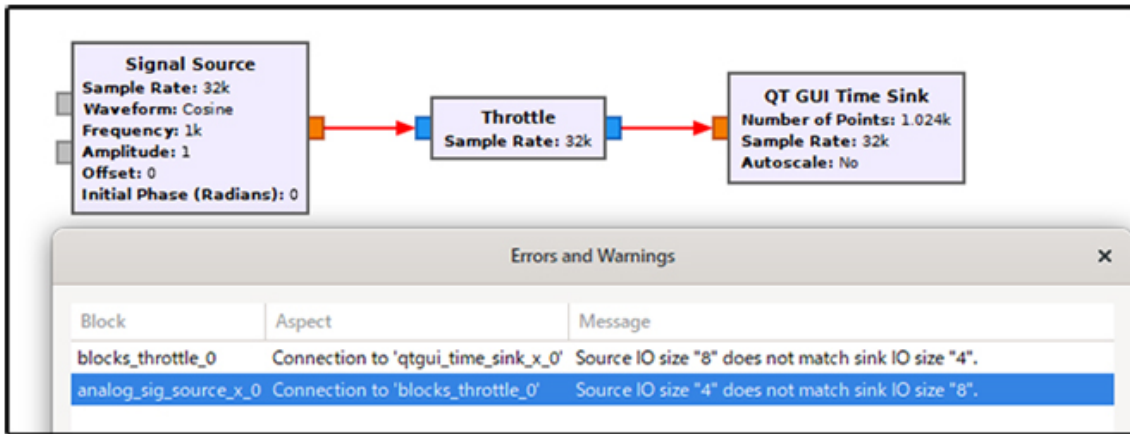


図 15 結線色によるエラー箇所の確認

それでは次の例に移りましょう。

## ■フローグラフ例 2

音声信号をマイクから入力し、その信号を GNURadio のオシロスコープ上に表示するとともにスピーカに出力する。また、信号の増幅機能を持たせる。

[処理概要]

- (1) 外部からの入力:マイク入力 → サウンドデバイス
- (2) 外部への出力:サウンドデバイス → スピーカ
- (3) ソフトウェアでの処理:
  - a.入力した信号を増幅
  - b.増幅した信号を GNURadio 上のオシロスコープに表示

前回の記事では、下図 16 のような処理フローを紹介しました。

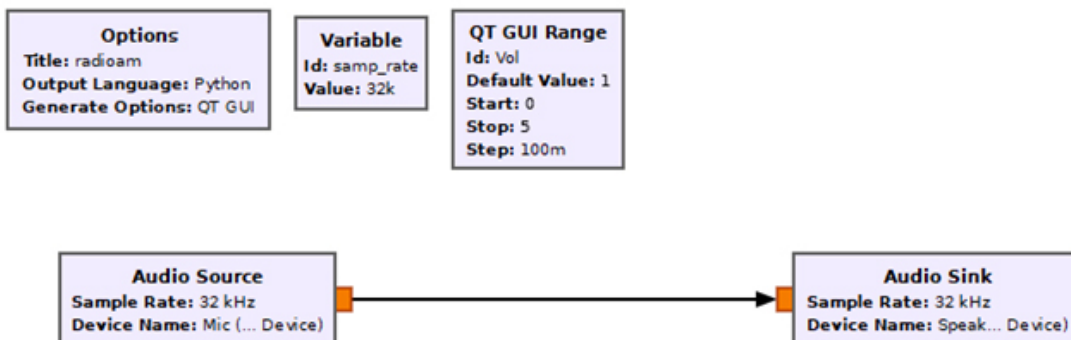


図 16 “Audio Source” と “Audio Sink” の結線

今から作成するフローグラフ例 2 は、図 16 の機能を少し拡張したフローグラフになります。

### [パラメータ設定]

必要なブロックを並べたフローが図 17 です。結線が赤色ですので、設定にエラーがあります。[Audio Source]と[Audio Sink]ブロックの端子色が橙色であるのに対して、[Multiply Const]と[QT GUI Time Sink]ブロックの端子色は青色です。これは[Audio Source]と[Audio Sink]のデータタイプが[`float`]であるのに対し、[Multiply Const]と[QT GUI Time Sink]ブロックのデータタイプが[`complex`]になっているためです。[Multiply Const]と[QT GUI Time Sink]ブロックをそれぞれダブルクリックして[IO Type]や[Type]を[`float`]に設定するとエラーが消えました。

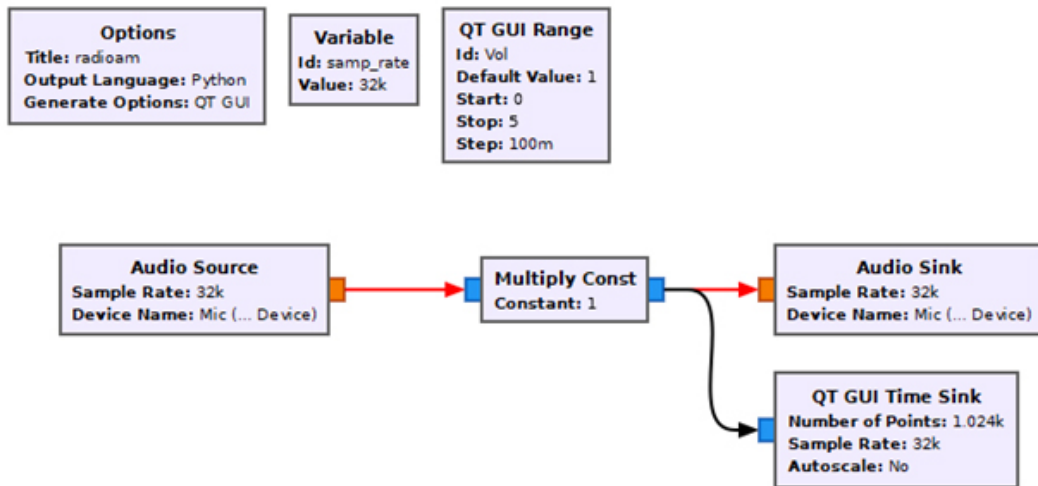


図 17 フローグラフ(結線まで)

次に[Audio Source]のパラメータ設定をみましょう。[Audio Source]や[Audio Sink]ボックスのパラメータ設定をすることでどのようなことがおきているのでしょうか。パソコンのサウンドボードにマイクを接続し、そのマイクからパソコン経由で GNURadio にデータを取り込む状態を想定してください。図 18 は GNURadio とパソコン、周辺装置の結線状況を表した概念図です。

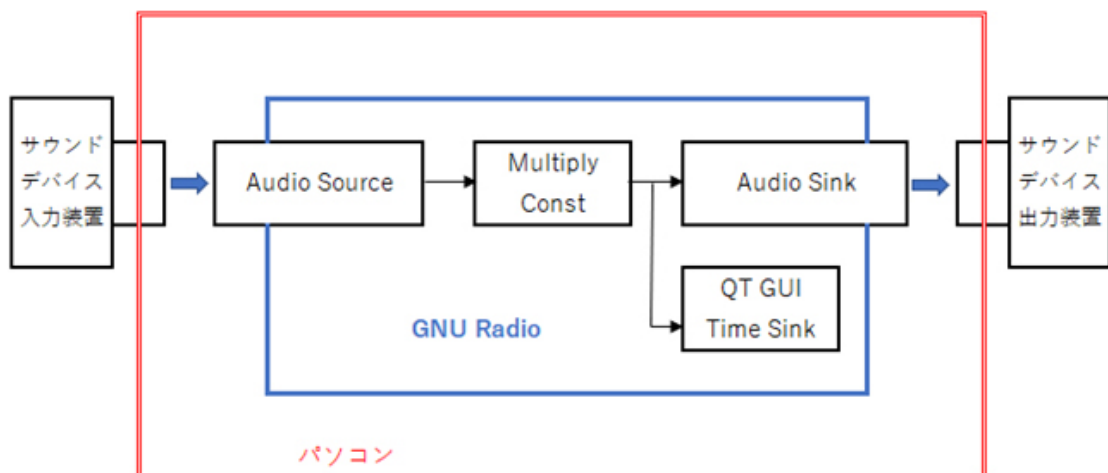
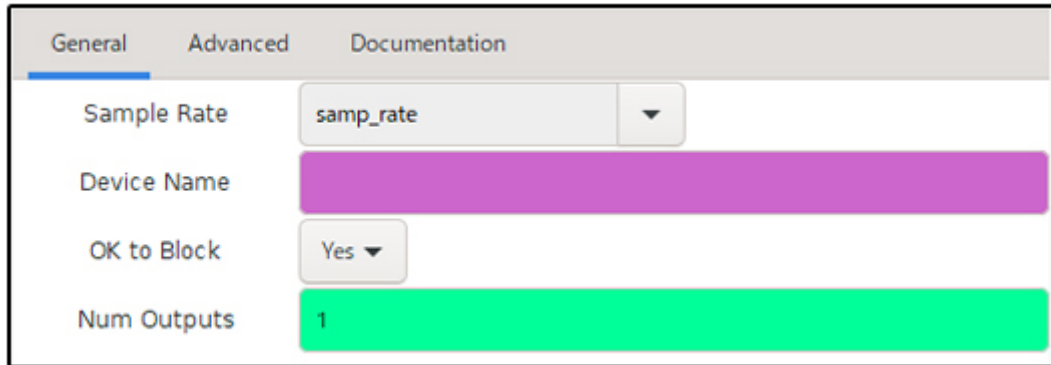


図 18 GNURadio と周辺装置との関係

周辺装置はパソコンの USB 端子などを通じて接続され、パソコンの OS がこれらの機器を認識し、必要に応じてドライバソフトを設定します。GNURadioの[Audio Source]や[Audio Sink]ボックスは、

図 18 のようにパソコンに接続された装置とのデータ受け渡しをします。どの装置とデータの受け渡しを行うのかを明確に指示する必要があります。そのために[Device name]などのパラメータを設定しています。サウンドボードと連携させるためのボックスが[Audio Source]や[Audio Sink]です。[Audio Source]のパラメータ設定を行うためには、ブロックをダブルクリックします。図 19 のようにパラメータ設定画面が開きます。



General	Advanced	Documentation
Sample Rate	samp_rate	▼
Device Name		
OK to Block	Yes	▼
Num Outputs	1	

図 19 Audio Source パラメータ設定画面

図 19 では、[Device Name]が空欄になっています。ここに何を入れればいいのでしょうか。図 19 の上部右端に見える[Documentation]タブを押すと[Audio Source]の説明文があり次のように記述があります。

Leave the device name blank to choose default audio device. ALSA users with audio trouble may try setting the device name to plughw:0,0(デフォルトのオーディオデバイスを選択するには、デバイス名を空白のままにします。オーディオに問題のある ALSA ユーザーは、デバイス名を plughw:0,0 に設定してみてください。)

デフォルトのデバイスを使用する場合はデバイス名を空白のままにするそうです。複数のオーディオデバイスがパソコンに接続されている場合は、接続したいデバイスをデフォルトデバイスに設定しておけば、あとは[Device Name]を空白にしておいてもよいのです。[Audio Source]ブロックを複数配置して、複数端子から音声入力を取り込みたい場合などは、デフォルトデバイスは1つしか指定できませんのでデバイス名で指定せざるを得ません。デバイス名で記述する場合は、[Documentation]に記載されている説明のように[plughw:0,0]などと入れてみましたが、Windows の GNURadio では認識してくれませんでした。いろいろ試した結果、日本語で書かれているデバイス名を Windows 上の設定で半角英数字に設定しなおし、その名前を[Device Name]に入れることで動作しました。デバイス名が“マイク”などの全角文字や漢字にするとだめでした。

筆者の環境下でサウンドデバイスを USB 端子に接続すると、入力デバイス名として[マイク(5- USB Audio Device)]という名前になっていました。図 20 のように、[デバイスのプロパティ]からデバイス名を、例えば[Mic (5- USB Audio Device)]と変更し、変更したデバイス名をダブルコーテーションで囲って[Device Name]に記載しました。なお、[Device Name]欄にデバイス名を記載するときは、スペースも含めて変更した名前を一字一句異なることなく記載してください(筆者はデバイス名の中にあった空白を省略して試した結果、日曜日の午後半日を無駄に過ごしてしまいました)。



図 20 デバイス名の変更

[ブロックのパラメータ]

[Audio Source]と[Audio Sink]のパラメータを下図 21、図 22 のように設定しました。(細かいことですが、文字[ Mic ]とカッコ[ ( )]の間に空白があります。この空白を省略するとマイクから音声を取り込むことができませんでした)

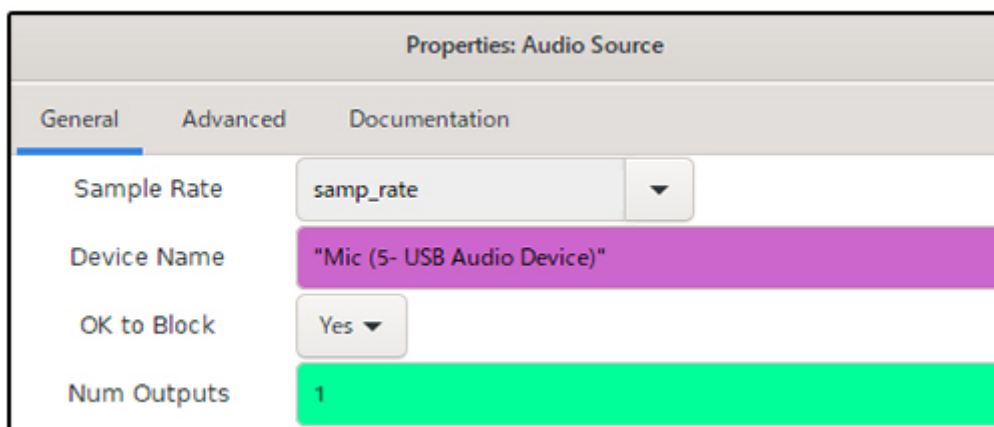


図 21 [Audio Source]の設定

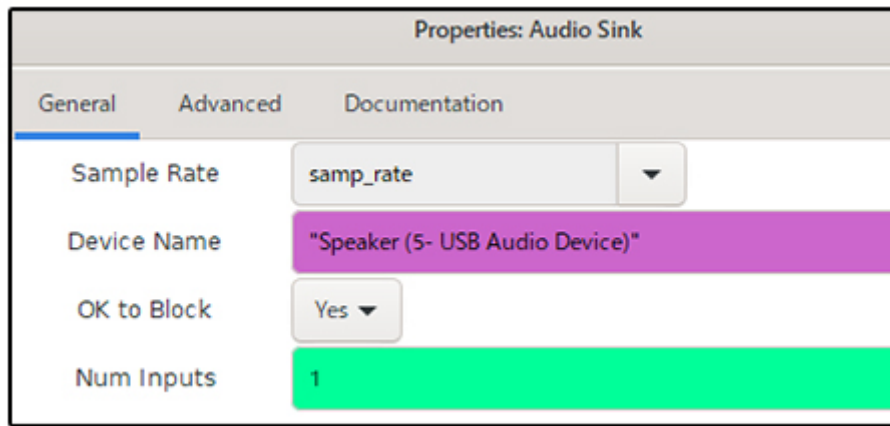


図 22 [Audio Sink]の設定

#### [実行結果]

サウンドデバイスの入力側にマイクを接続し、出力側にイヤホンを接続しました。その後、[Execute] ボタンを押してプログラムを実行し、マイクに向かって声を出すと、USB に挿したイヤホンから少し遅れて私の声が聞こえるとともに、パソコン画面上には図 23 のような波形が表示されます。また、図 23 の上部にある[Vol]をスライドすることで音量調節ができました。

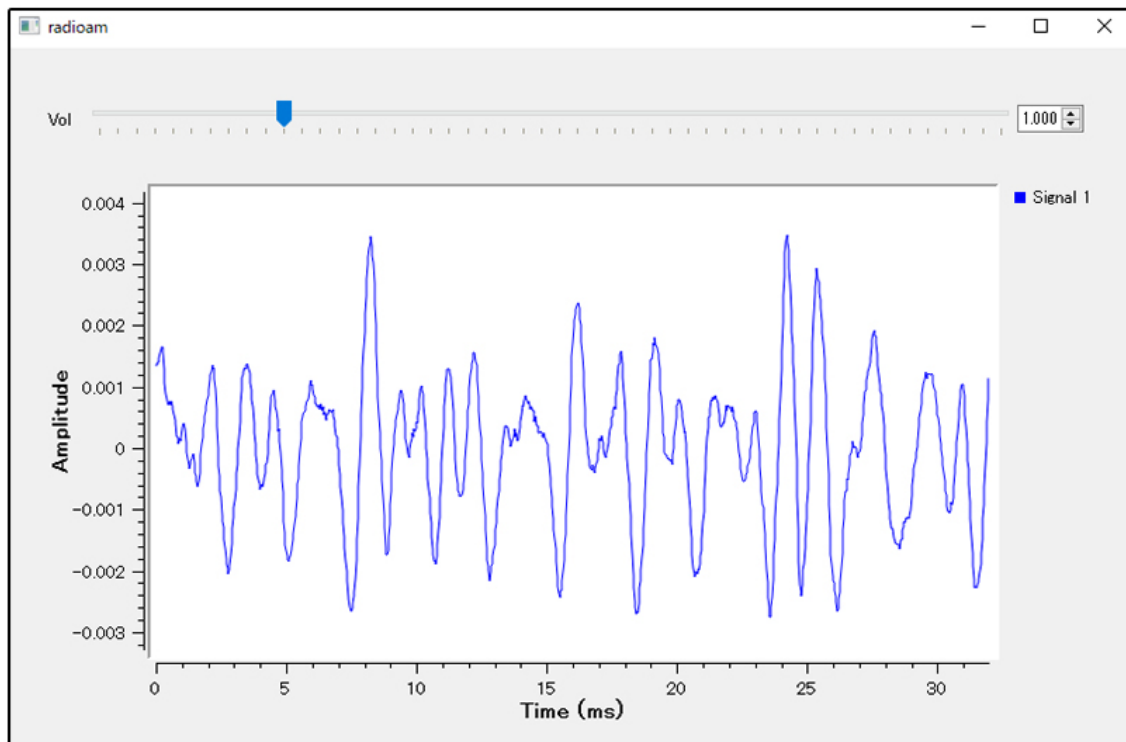


図 23 実行結果

#### [[Variable]ブロックについて]

[Variable]ブロックは変数を定義するために使います。プログラム言語などでは変数に値を代入した、代入された変数を使って参照したりします。例えば、 $i, k$  を変数とすると

$$i = 1$$

$$k = i + 3$$

と記述することで変数  $i$  には 1 が、変数  $k$  には 4 が入ります。GNU Radio では変数の定義はブロックで行います。変数が多くなるとワークスペースに変数ブロックがたくさん並んでいきますが仕方ありません。図 17 上の[Variable]ブロックをダブルクリックすると次のようにプロパティボックスが出てきます。

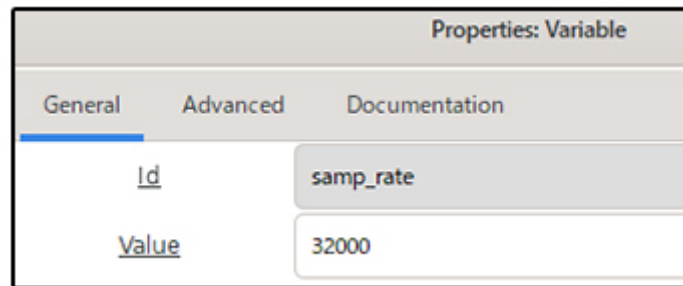


図 24 [Variable]ブロックのパラメータ設定値

これはプログラミング言語で言えば

```
samp_rate = 32000
```

という変数への代入と同じ意味です。参照の仕方ですが、図 17 の[QT GUI Time Sink]ブロックをダブルクリックすると、Type 欄に[Sample Rate]という文字があり、その右に[samp\_rate]の文字が入っています。

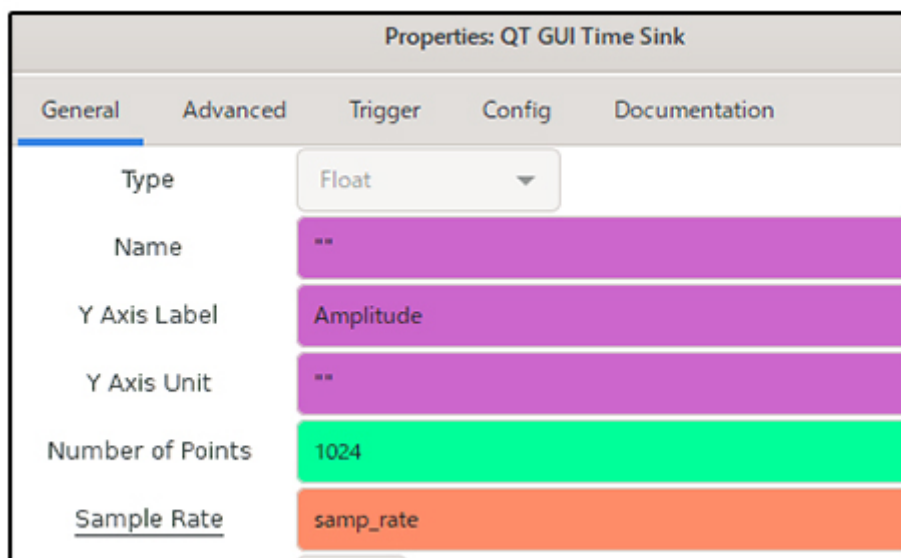


図 25 [QT GUI Time Sink]ブロックのパラメータ設定値

この[samp\_rate]は「Variable」ブロック内で定義されている変数に相当しており、[samp\_rate]と記載すると 32000 が Type[Sample Rate]に入ります。このように多くのブロックで共通の値を設定する必要がある場合は[Variable]ブロックに変数名と値を定義し、その使用は変数名を記載することで参照できます。[Variable]ブロックで定義された変数とその値は GRC ウィンドウ下部にある[変数]エリアに表示され、またこのエリアで変更することもできます。



[Sample Rate について]

サンプリングとは、アナログ信号をデジタル信号に変換(AD 変換)することで、日本語では標本化と呼ばれます。1 秒あたりのサンプル数をサンプルレート、サンプリングレート、サンプリング周波数などと言います。1 秒間に行う AD 変換を表すため単位には Hz(ヘルツ)が用いられます。ナイキストのサンプリング定理というのがありますが、これによるとサンプルレートは標本化対象信号の信号帯域幅の 2 倍以上の周波数が適切とされています。大部分の人は 20000Hz(20kHz)以上の周波数は聞こえませんが、その 2 倍の 40000Hz(40kHz)以上でサンプリングしてデジタル化すれば、その信号をアナログ信号に再変換しても元の信号が復元されます。音楽 CD のサンプルレートが 44.1kHz であるのもこのあたりに理由があります。

最近販売されているサウンドボードはサンプルレートが 32kHz や 44.1kHz,48kHz に可変できますが、[Audio Source]ブロックのサンプルレートには前記数値を含め、複数のサンプルレートを選ぶことができます。サンプルレートが高ければ高いほど高品質になりますが、データ量も比例して増えます。そのため、手持ちのパソコンが古い場合やストレージ容量によってはサンプルレートを調整する必要があります。

GNURadio では処理内容によってはブロックを経て信号が処理されると途中でサンプルレートを異なる値にしなければならないことがあります。このような場合信号処理の文献を紐解く必要があります。また、サンプルレートの設定がまちがっているためエラーが発生することもありますので、気をつけておいてください。

さて、前回の解説で予告していましたがバンドパスフィルタ(BPF)を作成していきます。最初に、「フローグラフ例 3」で BPF の動作性能を確認するため、外部信号をいきなり取り込むのではなく、GNURadio 内部で 2 つの周波数の信号を発生させ、その加算信号を BPF を通過させ調整をおこないます。その後、「フローグラフ例 4」で、「フローグラフ例 3」で作成したフローグラフの入力部分を[Audio Source]で置き換えて外部信号の BPF を実現します。

### ■フローグラフ例 3

内部発生信号からの特定周波数成分の抽出(バンドパスフィルタ)

---

[処理概要]

- (1) 外部からの入力信号: なし
  - (2) 外部への出力信号: サウンドボード経由スピーカやイヤホンへの出力
  - (3) ソフトウェアでの処理:
    - a. 内部生成合成信号の周波数成分表示(FFT 表示)
    - b. バンドパスフィルタ(BPF)による特定周波数信号の抽出
    - c. BPF 通過した信号の周波数成分表示(FFT 表示) BPF の通過周波数を次のようにします (CW 信号の分離を想定していますので、バンド幅を狭く設定できるようにしてあります) 中心周波数 : 700Hz(固定) 低域カットオフ周波数: 650Hz(可変:500Hz~690Hz) 広域カットオフ周波数: 750Hz(可変:710Hz~900Hz)
-

作成したバンドパスフィルタのフローグラフを図 26 に示します。

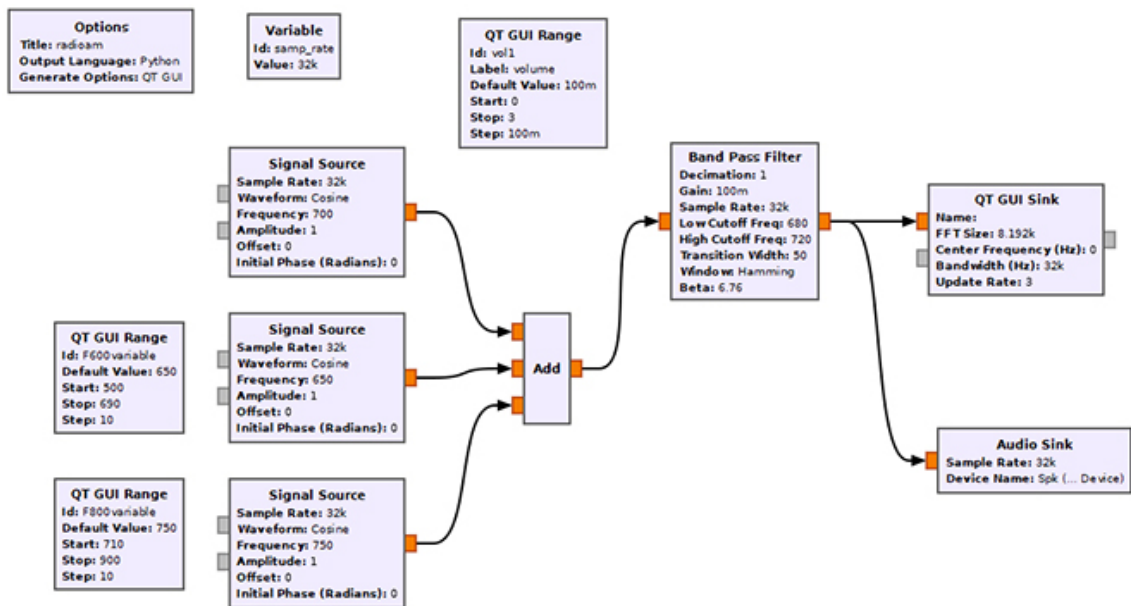


図 26 BPF(内部信号合成)

[フローグラフ説明]

・信号合成

左から 2 列目に 3 つの[Signal Source]を配置しました。それぞれの周波数は上から、700Hz(固定)、650Hz(可変)、750Hz(可変)です。これらはすべてデータタイプを[float]で指定しました。これら 3 つの信号を[Add]ブロックに入力することで合成信号(信号電圧の時間毎の加算値)を作成することができます。

・周波数を可変方法 ([QT GUI Range]ブロックの利用)

2 つのブロックの周波数を可変するため[Signal Source]の左側に配置した[QT GUI Range]で周波数を可変できるようにしました。

Properties: QT GUI Range		
General	Advanced	Documentation
<u>Id</u>	F600variable	
Label		
Type	float ▼	
<u>Default Value</u>	650	
Start	500	
Stop	690	
Step	10	
Widget	Counter + Slider ▼	

27 QT GUI Range のプロパティ

変数の値を[F600variable]とし、デフォルト値を 650Hz としました。可変範囲を 500Hz から 690Hz、増減数が 10Hz となるように設定しました。周波数を可変するために[Widget]でカウンタとスライダを選び、マウスで周波数を変化できるようにしました。650Hz の発信機(3 つある[Signal Source]の上から 2 番目のブロック)の周波数をスライダで変化させるためには、このブロックの[Frequency]プロパティに[F600variable] (“[“,”]” は入れません)と入力します。750Hz の発信機についても同様です。

- バンドパスフィルタの配置([Band Pass Filter]ブロックの利用)GNURadio には Low Pass Filter、High Pass Filter、Band Pass Filter などのブロックが準備されていますので、ライブラリから探してワークスペースに置きます。中心周波数を 700Hz としてカットオフ周波数を設定します。700Hz の  $\pm 50$ Hz をカットオフ周波数に設定しました(Low Cutoff Freq:650、High Cutoff Freq:750)。

[BPF 出力の FFT 表示]BPF の性能を確認するため、BPF 通過後の信号の周波数成分を確認するため FFT 表示用ブロックを挿入しました。図 26 の右端にある[QT GUI Sink]ブロックが挿入したブロックです。

[実行結果]

結果を FFT で表示してみました。650Hz、700Hz、750Hz の信号がみえています。

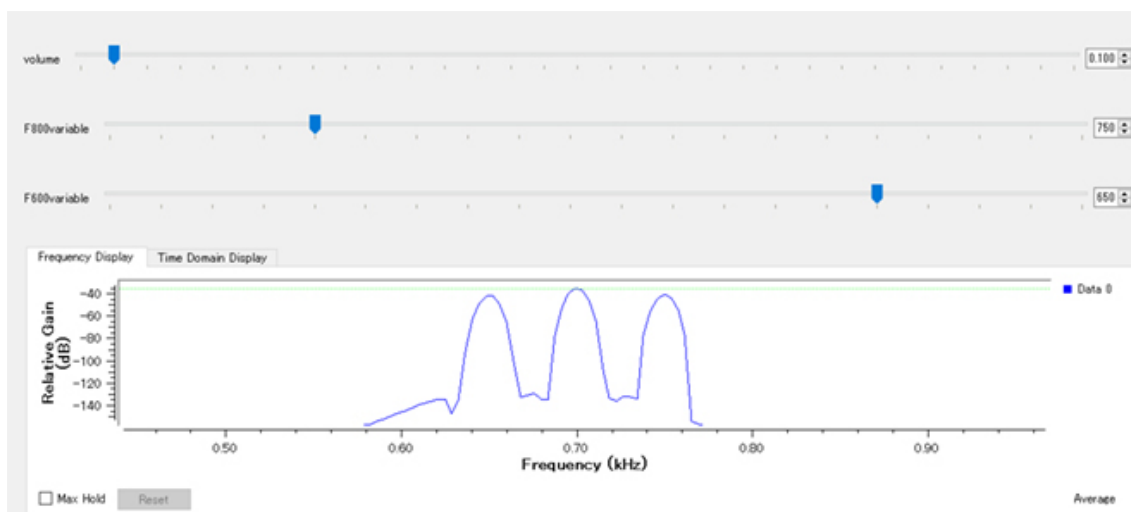


図 28 BPF 実行結果(Low Cutoff Freq:650Hz、High Cutoff Freq:750Hz)

700Hz の信号と、ほかの 650Hz、750Hz の信号とのレベル差は 6dB 程度です。またカットオフ周波数を 680Hz,720Hz に設定すると、650Hz、750Hz の信号とのレベル差は 68dB 程度に改善されました。

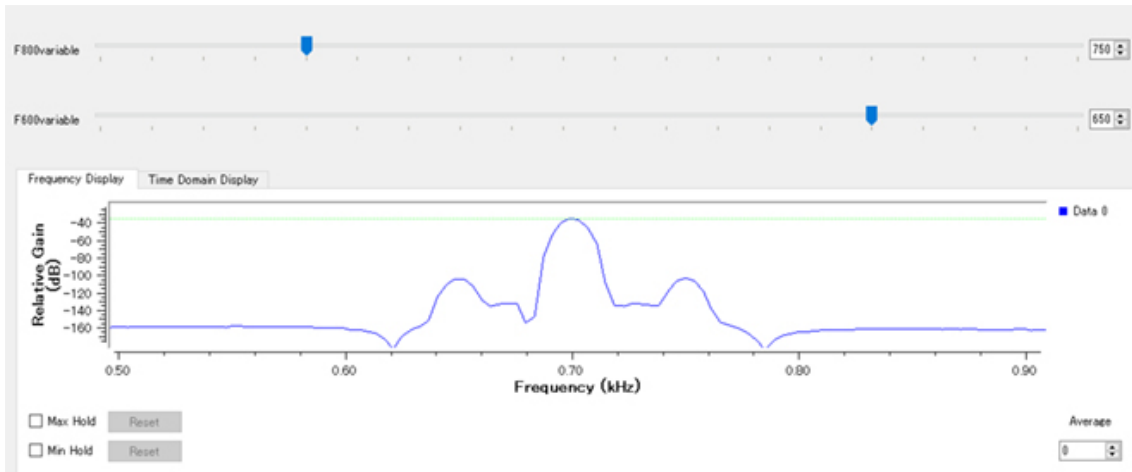


図 29 BPF 実行結果(Low Cutoff Freq:680Hz、High Cutoff Freq:720Hz)

次に、BPF への入力レベルと BPF 通過後の出力レベルを比較してみました。[Add]ブロックの出力に [QT GUI Sink]ブロックを接続し FFT 表示した結果、BPF 通過前と通過後で 700Hz 信号のレベルが 20dB 程度減衰していることが FFT 表示で確認できました(図 30)。

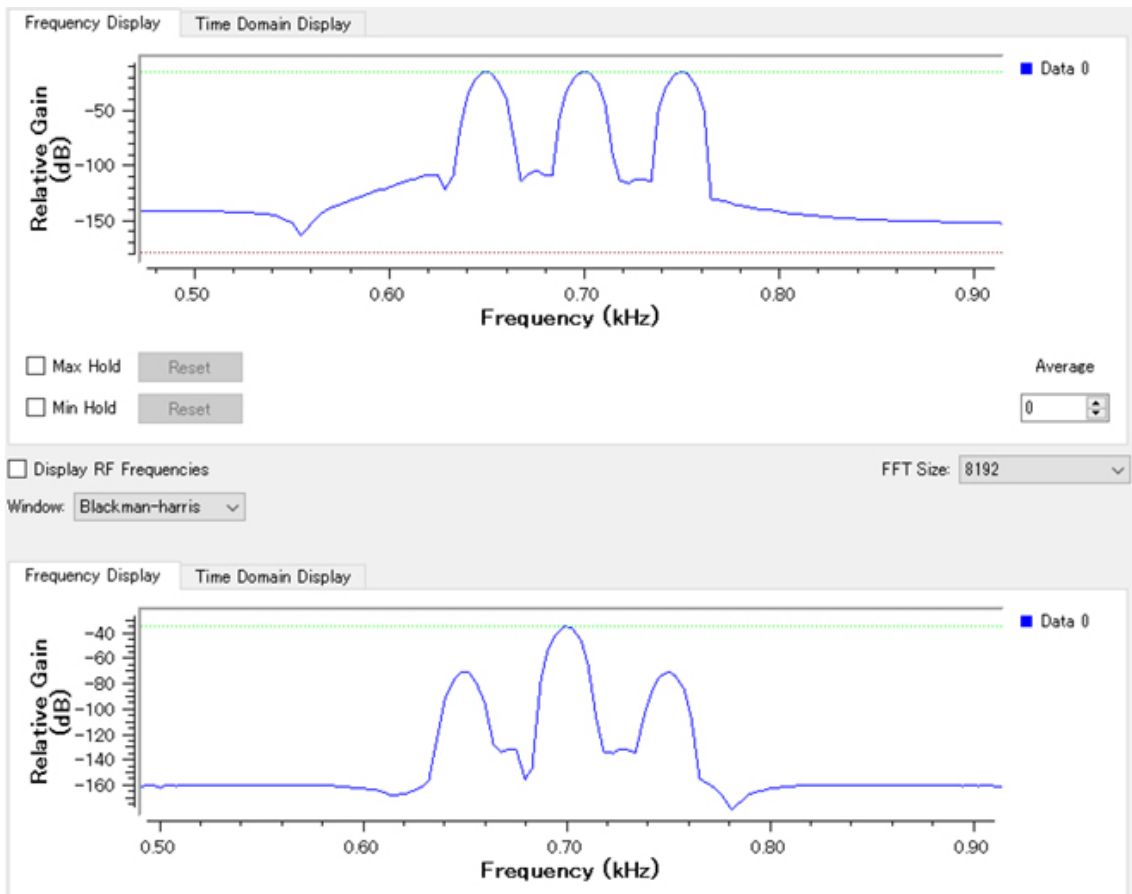


図 30 BPF 減衰特性(上図: BPF 入力信号、下図: BPF 出力信号)

フローグラフ例 3 を少し変形して外部入力信号の BPF を作成してみます。

## ■フローグラフ例 4

外部入力信号からの特定周波数成分の抽出(バンドパスフィルタ)

[処理概要]

- (1) 外部からの入力信号: サウンドボード経由、オーディオケーブルによる無線機の音声周波数帯域のアナログ電気信号
- (2) 外部への出力信号: サウンドボード経由スピーカやイヤホンへの出力
- (3) ソフトウェアでの処理:
  - a. 入力信号の周波数成分表示(FFT 表示)
  - b. バンドパスフィルタ(BPF)による特定周波数信号の抽出
  - c. BPF 通過した信号の周波数成分表示(FFT 表示)

この機能を持つフローグラフは、例 3 のフローグラフの入力部を[Audio Source]ブロックで置き換えることで実現できます。図 31 が外部信号を入力するフローグラフです。

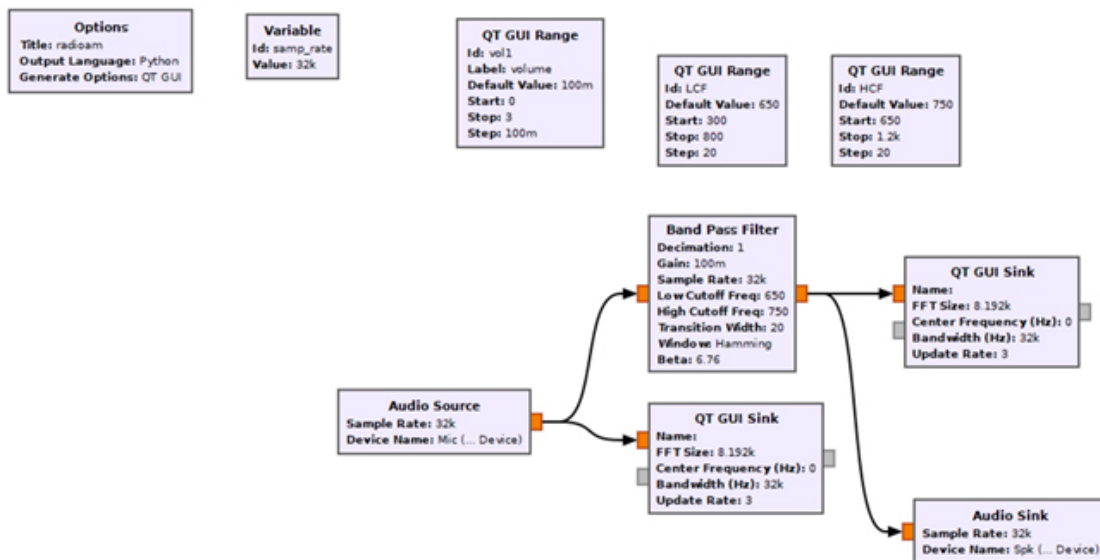


図 31 BPF(外部入力、外部出力)

無線機の PHONES 端子にオーディオケーブルを接続し、パソコンのオーディオデバイスに接続して CW 信号を取り込んでみました。耳計測では BPF の効果が実感できますが定量的計測をおこないませんでした。周波数の近接したモールス通信を見つけて効果の実機評価を行おうとしましたがなかなか見つけることができませんでした。実機評価は今後の課題としたいとおもいます。

## ■GNURadio の情報入手先(サイト)

GNURadio の情報は今ではネット上にたくさん掲載されています。自分の目的に合わせて検索結果からチョイスするのが早いかもしれません。次の URL は GNURadio の総本山です。英語で書かれているのでちょっと大変ですが、ブラウザで日本語に翻訳すれば少し訳がおかしいところもありますがニュースはつかめるとおもいます。

- GNURadio の情報サイトのトップ  
<https://www.gnuradio.org/>
- GNURadio のドキュメント  
[https://wiki.gnuradio.org/index.php/Main\\_Page](https://wiki.gnuradio.org/index.php/Main_Page)
- GNURadio のチュートリアル(図 32)  
(Tutorial:アプリケーションソフトなどの基本的な操作方法を覚えるための教材のこと)  
<https://wiki.gnuradio.org/index.php/Tutorials>

チュートリアルには GNURadio や SDR、DSP や GNURadio の使い方などがしめされています。英語版ですが、このチュートリアルを日本語に翻訳して参考にされることをおすすめします。チュートリアルの中に、「これらのチュートリアルは単にガイドとして意図されており、何かを学ぶための最良の方法はそれを試すことです。」と書かれています。もし GNURadio に興味が湧きましたらぜひ動かしてみてください。



図 32 GNURadio チュートリアル(Chrome 上で日本語に翻訳したもの)

今回はここまでで終わらせていただきます。

#### 【参考文献】

- (1) 高橋 知宏, GRC で広がる SDR の世界, CQ 出版, 東京, 2018.
- (2) rapidnack, GNURadio 電子工作 Vol.1 AM 送信・受信編 Kindle 版
- (3) rapidnack, GNURadio 電子工作 Vol.2 Embedded Python Block 編 Kindle 版  
3冊ともフローグラフを作成するのに非常に参考になりました。