卒業研究報告書

フレキシブル触覚呈示システムのための導電布上での マルチチャンネル FM 伝送の研究

報　告　者

学籍番号：1240175

氏名：義本 卓叶

指　導　教　員

野田 聡人 准教授

令和 6 年 2 月 16 日

高知工科大学 システム工学群

電子・光工学専攻

目次

第 1 章　序論

1.1　研究背景

バーチャルリアリティ (VR) やオーグメンテッドリアリティ (AR) における触覚の再現は，直感的な体験を実現する為に重要な課題である．現在実用化されている VR や AR においては，視覚と聴覚及び局所的な触覚（手持ちのデバイスやヘッドマウントディスプレイ周辺）の再現に留まっている．より臨場感のある体験を実現する為には，局所的ではなく全身に触覚を再現できることが望まれる．触覚を再現する為には，触覚刺激を与えられる振動デバイスを全身に配置した着用可能な触覚フィードバックシステムが必要である．全身にわたって触覚を再現することができれば，視覚や聴覚と同調してより再現度を高めることができるだけでなく，触覚を感知するセンシングシステムと併用して他者間で触覚を共有することも可能である．

全身に触覚刺激をフレキシブルに与えられるシステムの開発にあたり，複数の振動デバイスに対し個別に触覚刺激を与える方法やそれらの配線方法が課題である．全身に振動デバイスを，例えば数 cm 間隔で配置する場合少なくとも数十個以上となるが，これらのデバイスに対し個別にケーブルで接続するのは現実的ではない．配線の数が膨大になり，断線などの故障リスクが高い上，これらの配線によって身体の動きが妨げられると没入感の高い体験において障害となり得る．

1.2　研究目的

上記の課題に対しての解決策として，二次元通信を用いる方式がある．伝送路である二次元導電布上に複数のデバイスを配置し，ソフトウェア無線 (Software Defined Radio, SDR) で生成した多チャンネルの周波数変調 (FM) 波を用いて制御する手法が提案されている [1].

二次元伝送路上であれば任意の場所にデバイスを設置でき，比較的容易に再配置ができる為，同調するコンテンツや身長，体格に合わせた配置が可能である．また FM 波の受信には市販の FM ラジオ受信用 IC を用いることができ，回路の簡略化やコストの削減が可能である．SDR を用いることで複数の変調信号を加算した多チャンネルの被変調

波を容易に生成可能である．身体表面に装着して使用する場合，そのシステムによって身体の動作を妨げることなく，また動作による断線などの故障の可能性を低減する必要があり，二次元導電布はこれらの課題に対し有利である．無線通信と比較して接続安定性，遅延，同時に給電を行うことができる点でも優れている．また二次元伝送路上でのみ通信を行う為，三次元空間の周波数帯域を占有しない．すなわち絶縁された二次元伝送路同士や三次元通信は，同じ周波数を用いても干渉することなく通信を行うことができる．今後更に IoT 等の無線機器が普及すると考えられ，無線通信の混雑が予想される．電波資源の枯渇が心配される状況下において，周波数を占有せずに通信を行える点は大きな利点である．

SDR を用いて FM 波を送信するという原理的な実装の可能性は示されているものの，実用に向けた未解決の課題に次のものが挙げられる．

1. 多チャンネル送信時に無視することができない強度のスプリアスが発生していること
2. 任意の触覚刺激波形を任意のタイミングで送信できないこと
3. 受信デバイスの数 (チャンネルの数) を増やすことで処理負荷が増加すること

文献 [1] によると，多チャンネル送信における最小構成である 2ch の場合でも多数のスプリアスが発生することが報告されている．スプリアスが発生した周波数帯では干渉が発生する為，チャンネル数を増やすにあたってスプリアスの抑制は必須である．また文献 [1] で提案されているプログラムでは予め設定した触覚刺激波形を予め指定したチャンネルに対し送信する動作を繰り返す機能が実装されている．このままでは映像や操作と同調した触覚呈示に利用できず，VR 機器と組み合わせるには映像コンテンツとの同期やユーザ入力への応答として，任意のタイミングで任意の触覚刺激波形を送信する機能が必要である．チャンネル数を増やすにあたっては SDR のサンプリング周波数を増やす必要があり，それにより処理負荷が増大する．全身に触覚を再現できる数のチャンネルを設置する場合の処理負荷は，身体に装着できるサイズの小型コンピュータの処理能力を大幅に超えており，処理負荷を低減することも重要である．本論文ではこれらの課題についての解決手法を提案する．

1.3　概要

本論文は，2 章で実験や提案手法に関する概要を説明する．プログラムの概要や実験に用いた機器について，また二次元通信，SDR，FM 伝送が優れている要素について述べる．3 章でスプリアスの発生原因及び抑制方法についての調査，4 章で任意のタイミン

グで任意の触覚刺激波形を送信する為の調査及びそれを実現する手法の提案及びその考察，5 章で処理負荷を軽減して小型のコンピュータで動作させる手法の提案及びその考察，6 章で結論を述べる．提案手法では触覚刺激波形の出力レベルを下げることでスプリアスを大幅に抑制することができた．また，使用しているソフトウェアによって自動生成されるスクリプトを編集することで任意の触覚刺激波形を送信する機能を実装した．さらに処理負荷を抑える方法についても，小型のコンピュータで動作する程度まで処理負荷を抑える為の実装方法を提案する．

第 2 章　実験方法

本章では本論文で行う実験に係る機器や原理について説明する．2.1 節では実験に用いる機材及び実験方法について，2.2 節で多チャンネルの触覚刺激の原理について，2.3 節では二次元通信について，2.4 節では SDR について，2.5 節でプログラムの概要について説明する．

2.1　実験系の概要

本論文で用いた実験系について説明する．

SDR デバイスには USRP N210(Ettus Research) を使用した．また，実際に出力されているスペクトラムを確認する為にスペクトラムアナライザ (RSA5065N, RIGOL) を使用した．

実験系の全体写真を図 2.1 に，実験系の接続関係をブロック線図として図 2.2 に示す．コンピュータと N210 は LAN ケーブルで接続されており，N210 から同軸ケーブルで Bias Tee を介して二次元伝送路に接続している．コンピュータでプログラムを実行すると N210 が FM 変調された触覚刺激波形を出力する．出力された触覚刺激波形に Bias Tee で 3.3V を重畳して二次元伝送路に入力する．二次元伝送路上に配置された受信デバイスが触覚刺激波形を復調し，受信デバイスに接続されているボイスコイルアクチュエータ (AFT14A903A, Alps Alpine) を振動させる．

二次元伝送路を拡大したものを図 2.3 に示す．二次元伝送路として導電性の布 (導電布) を用いる．導電布上に受信デバイスを複数設置しており，これらの受信デバイスは並列接続されている為全てのデバイスが全ての信号を受信するが，それぞれ異なる周波数の信号のみを復調するようプログラムされている．

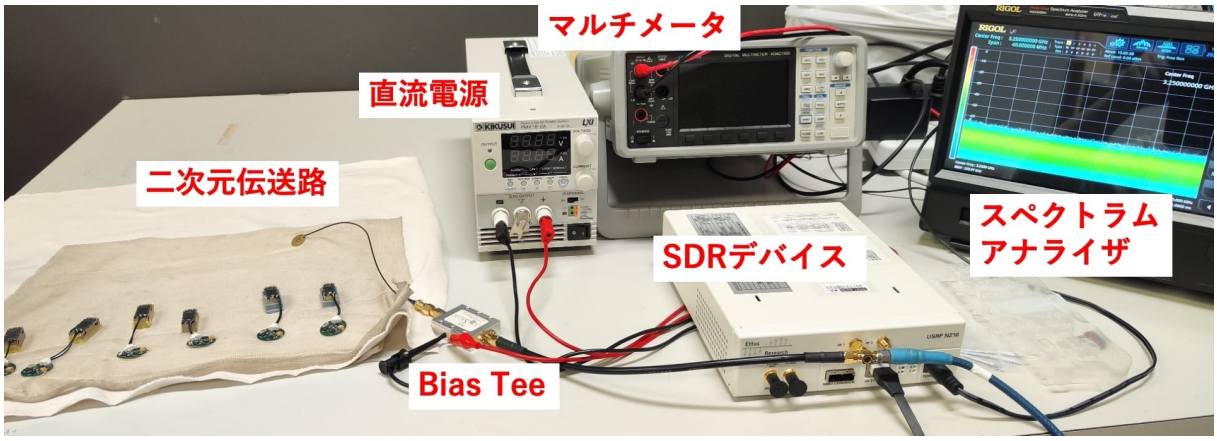

図 2.1 実験系の全体図

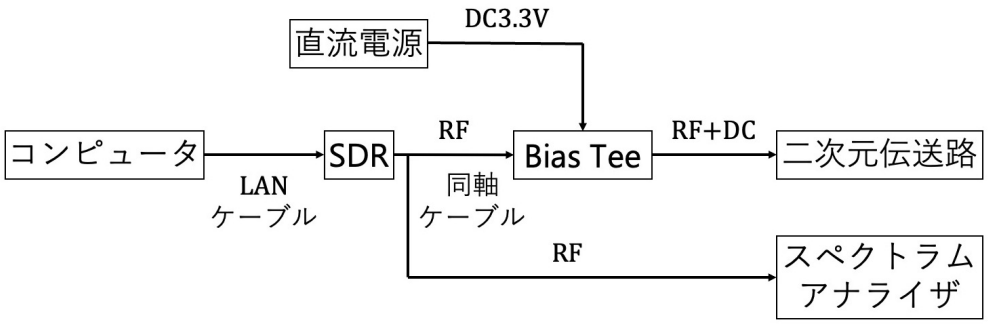

図 2.2 実験系のブロック線図

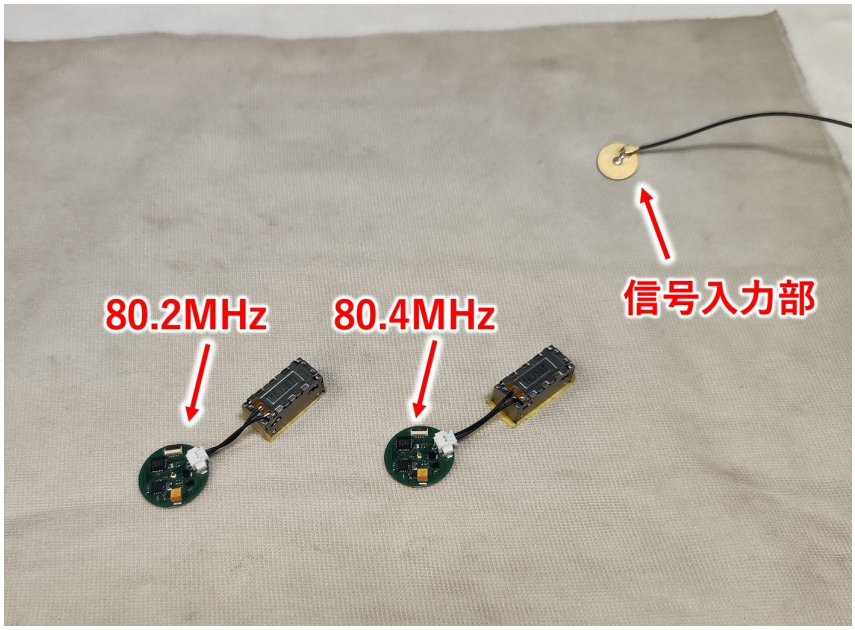

図 2.3 二次元伝送路上で並列接続された受信デバイス．受信デバイス毎に復調する周波数帯が異なる為，並列接続されている複数のデバイスに対し個別の制御が可能である．

2.2　多チャンネルの触覚刺激波形について

本節では多チャンネルの触覚刺激波形送信の原理について述べる．

一般的な単一キャリア (搬送波) の FM 変調は，周波数 f_0 のキャリアを変調信号 $v_0(t)$ によって変調するもので，変調信号の周波数と変調度 m の積 $mv_0(t)$ とキャリア周波数の和である $mv_0(t) + f_0$ が被変調波の周波数となる．多チャンネルの場合，それぞれの変調信号を周波数の異なるキャリアによって変調し，それらを足し合わせることで複数の周波数成分を含む被変調波を得る．本研究ではキャリア周波数はソフトウェア上で SDR デバイスの中心周波数 (Center Frequency) となる単一の周波数 $f = f_0$ で，予め変調信号に周波数オフセット Δf を掛けることで周波数の異なるキャリアを表現する．N 個のキャリアが Δf 間隔で配置されたとき，キャリア周波数は $f_0 + k\Delta f \ \ (k = 1, 2, \cdots, N)$ となり，変調信号 $v_k(t)$ を足した周波数成分 f_k は

$$f_k = (f_0 + k\Delta f) + mv_k(t) \ \ (k = 1, 2, \cdots, N)$$

となる．上記のフローのブロック線図を図 2.4 に示す．文献 [1] では変調信号の一つは Δf を含まないキャリア周波数 f で変調するが，本論文では 3 章で述べる理由により全ての変調信号に周波数オフセットを掛ける．

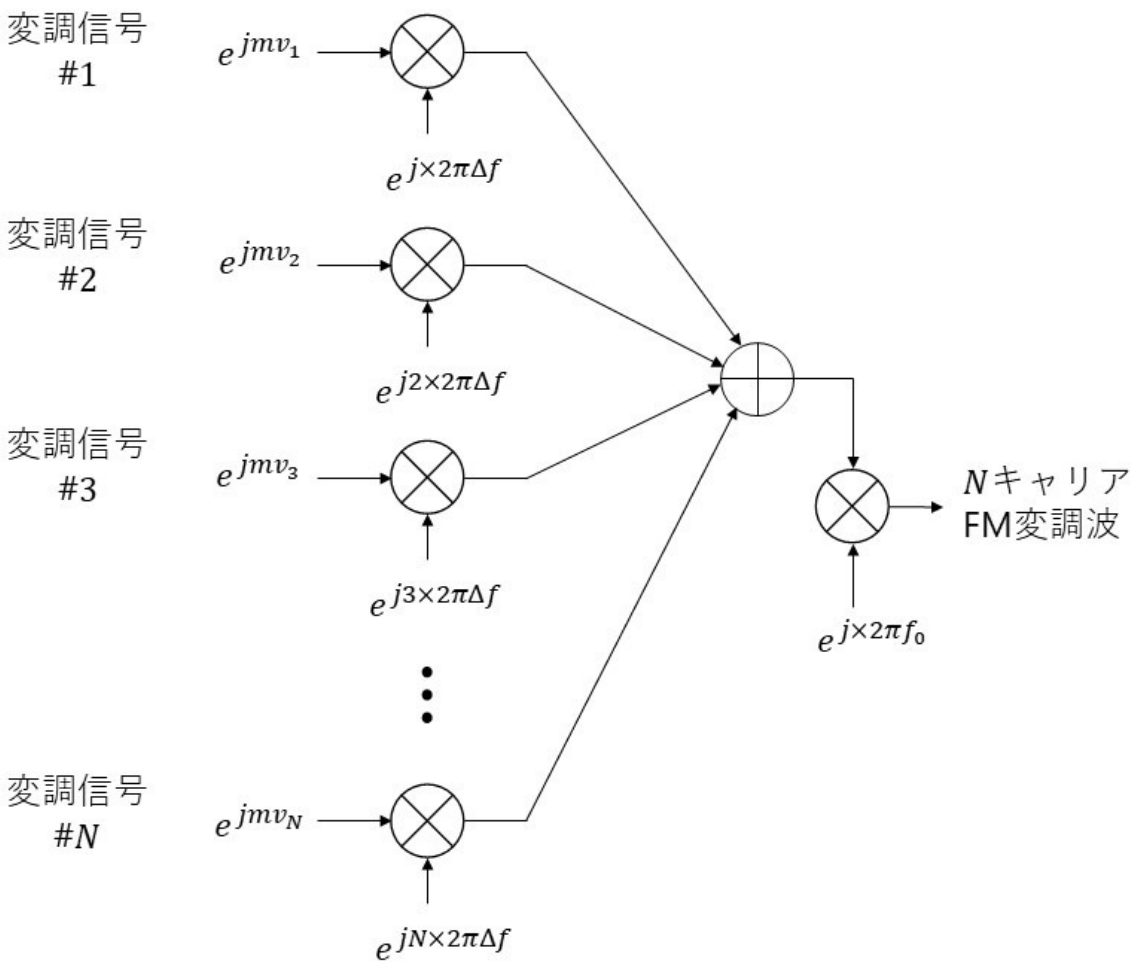

図 2.4 複素信号の乗算・加算による多チャンネル FM 変調波の生成

2.3 二次元伝送路及び受信デバイス

本節では，受信デバイスの仕様，及び SDR デバイスと受信デバイス間の接続に用いている二次元通信について説明する．

2.3.1 二次元通信について

二次元通信は，平面形状の伝送路を用いて通信するもので，二次元平面上であれば任意の位置同士で通信が可能である．二次元伝送路上に電波を閉じ込め近接した機器と通信を行うものと [2]，二次元伝送路と通信機器を電気的に導通させて通信を行うものがあり，本論文では後者を用いる．前者は近接するだけで通信が可能である為着脱が容易である．一方後者は + と − の二つの導体面に接続する必要があり着脱は前者と比較して手間がかかるが，信号に直流電圧を重畳して前者よりも大きな電源供給が可能である．

電波が二次元伝送路上にのみ伝搬する為，三次元通信 (一般的な無線通信) と異なり付近の同一周波数を用いた通信との干渉が起こらない．一次元通信 (ケーブルを用いた通信) と比較して配置が自由であり柔軟性が高く，製造や修理に係るコストを軽減できる．

2.3.2　二次元導電布について

二次元導電布は図 2.5，2.6 に示すように導電層，絶縁層，導電層の 3 層構造になっており，導電層同士は絶縁層によって絶縁されている．導電層は独立した導電布を絶縁布と重ねて用いるものや，絶縁布に導電性の糸で刺繍するもの [3] などがある．図 2.7 に文献 [4] より受信デバイスのコネクタ構造を示す．3 層を貫通するピン状のコネクタと，中央にコネクタ接続部がある受信デバイスを，3 層の布を挟み込むように取り付けることで受信デバイスは両方の導体面と接続される．二次元導電布上に複数のデバイスを取り付けた場合，それらは全て電気的に並列に接続される．実用化に向けては，文献 [5] に掲載されているような導電性の布で作成された衣服を利用したシステムを想定している．

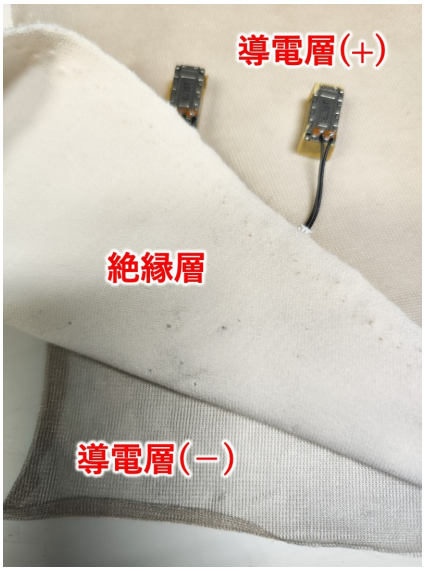

図 2.5　二次元導電布の構造

図 2.6　裏面の導電層．コネクタを介して受信デバイスは GND と接続されている．

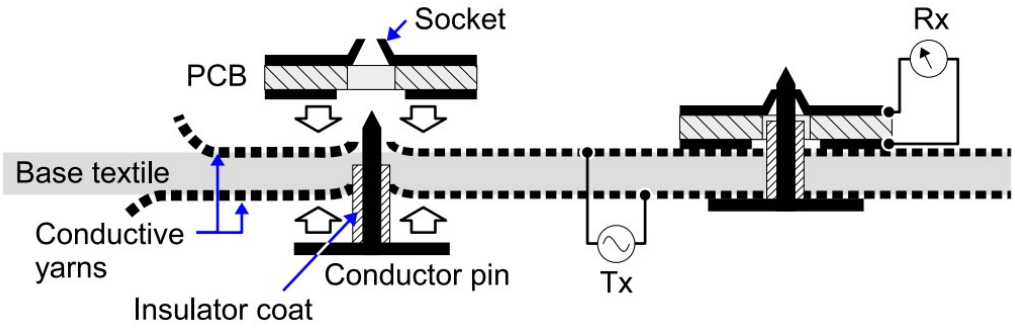

図 2.7　コネクタの構造を示す断面図 [4]

2.3.3 受信デバイスについて

受信デバイスとボイスコイルアクチュエータを図 2.8 に示す．この受信デバイスに搭載されている FM ラジオ受信用 IC(SI4721, Silicon Laboratories) の受信可能周波数帯域は 76.0MHz から 108.0MHz である．本論文では中心周波数を 80.0MHz，各チャンネルの間隔を 200kHz，振動周波数を 200Hz に設定している．

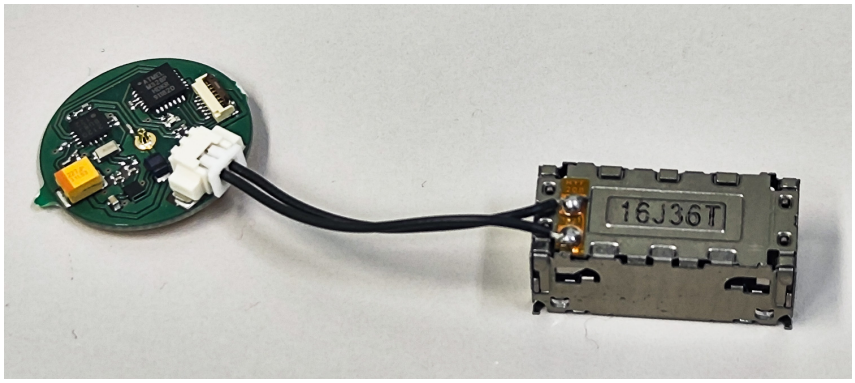

図 2.8 受信デバイスとボイスコイルアクチュエータ

2.4 アナログ波形送信機としての SDR デバイス

複数の受信デバイスを個別に制御する手法として FM 伝送を用いる．FM 波の受信には市販の FM ラジオ受信用 IC を利用できる為，受信デバイスの回路やソフトウェアの単純化，生産コストの削減，開発の負担軽減が可能である．

本研究では多チャンネルの FM 波の送信に SDR デバイスを用いている．SDR とは，アナログ部品 (変復調器，ミキサなど) を用いて行われていた変復調処理を，コンピュータのソフトウェアを用いて実現するものである．市販の FM 送信機器では複数のキャリアを同時に変調した被変調波を送信することが想定されておらず，受信デバイス 1 台に対し送信デバイスも 1 台必要となる．触覚呈示システムとしては数 cm 以下の間隔でボイスコイルアクチュエータを配置する必要がある．身体全体に触覚刺激を呈示すると想定した場合受信デバイスは少なくとも数十台以上であり，チャンネルと送信デバイスも受信デバイスと同じ数必要になる為非合理的である．SDR ではソフトウェア的には任意のチャンネル数を同時に変調，制御することが可能であり，送信デバイスを集約することができる．

2.5　プログラム

文献 [1] 及び本研究で用いた開発環境は GNU Radio Companion(GRC) [6] である．GRC では図 2.9 などに示すように，GUI で作成したシグナルフローグラフから GNU Radio ライブラリを利用する Python スクリプトを自動生成する．

フローグラフ例として図 2.9 の構造を説明する．Wav File Source ブロックで Wav ファイル形式の触覚刺激波形を読み込み，WBFM Transmit ブロックでそれを変調した変調信号を生成する．Wav ファイルのサンプリング周波数は 48kSps である為，Rational Resampler ブロックで SDR 側のサンプリング周波数 (7.2MSps) と同調させる．Frequency Shift ブロックで触覚刺激波形に予め設定したオフセット周波数を乗算し，UHD:USRP Sink ブロックで N210 から被変調波の送信を行う．複数のチャンネルの場合同様の処理を並列で行い，それを Add ブロックで加算することで複数の変調波を加算した被変調波を得る．4 章，5 章では GRC で実装できない機能を実装する為，自動生成されたスクリプトを編集して実装する．

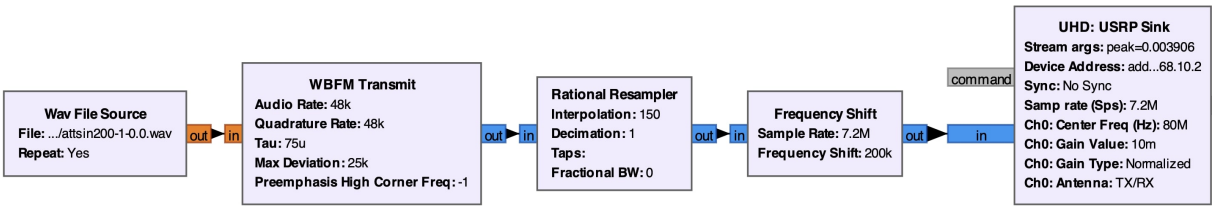

図 2.9　GRC で作成するフローグラフの例

第 3 章　スプリアスの抑制

本章では文献 [1] で提示されている課題であるスプリアスの抑制について実験を行う．

3.1　先行研究での実験

文献 [1] では，中心周波数 90.0MHz で，オフセット無し (90.0MHz)，オフセット周波数 +200kHz(90.2MHz) の 2ch のアナログ波形を送信する実験が行われている．実験時のスペクトラムアナライザの写真を図 3.1 に示す．

所望波以外の 89.6MHz，89.8MHz，90.4MHz 成分に無視できない強度のスプリアスが赤丸部分に生じていることがわかる．また表示範囲外にも同様のスプリアスが発生していると考えられる．スプリアスが発生した場合，発生した周波数帯に設置されているチャンネルと干渉することとなり，スプリアスによって指定した受信デバイス以外のデバイスが誤動作を起こす可能性がある．このままではチャンネル数を増やすことは難しく，広い周波数帯に多数のチャンネルを設ける為にはスプリアスを抑制する必要がある．

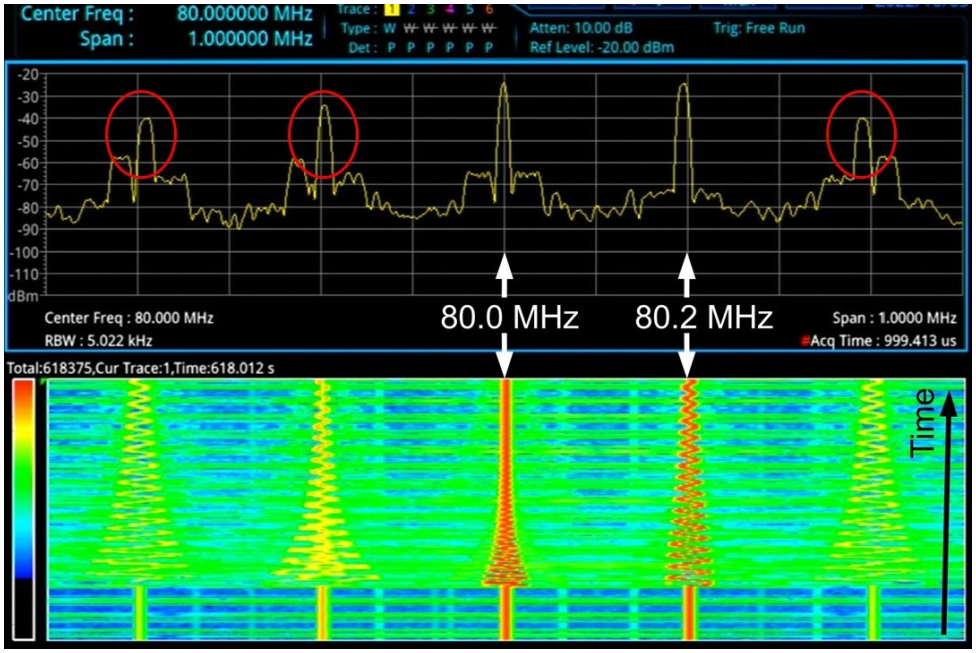

図 3.1　2ch 送信時のスペクトラム [1]．赤丸で示した部分に無視できない強度のスプリアスが生じていることがわかる．

3.2　スプリアスの原因の調査

3.2.1　スプリアスの発生状況

文献 [1] と同様の環境で実験を行った場合のスプリアスの発生状況を観測する．3.2.2 項ではまず 1ch の変調信号にオフセットを掛けて送信する場合について実験し，3.2.4 項で多チャンネルの場合について実験する．サンプリング周波数は 7.2MSps，中心周波数 80MHz で実験を行った．

3.2.2　1ch の場合

本項では 1ch の触覚刺激波形送信について実験した．オフセット周波数は +200kHz に設定した．実験に用いたフローグラフが図 3.2，スペクトラムが図 3.3 である．図 3.3 は 79.0MHz から 81.0MHz までのスペクトラムを表示している．

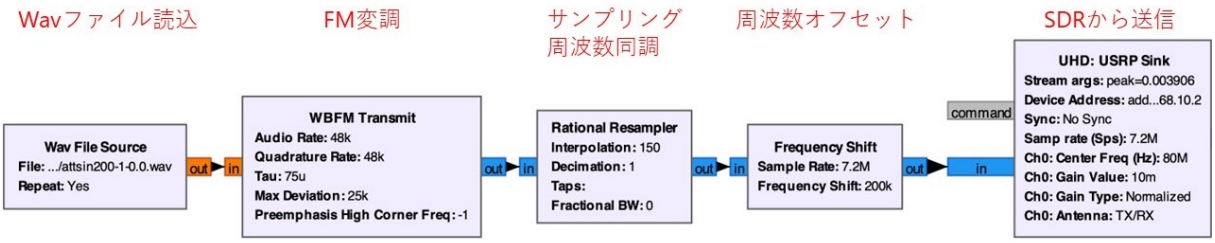

図 3.2　1ch のフローグラフ

図 3.3 において，比較的強度の大きいスプリアスに番号を振った．所望波のオフセット周波数を Δf_0 とする．番号ごとの原因を表 3.1 に示す．

表 3.1　それぞれのスプリアスについて推測される発生要因

①	$-3\Delta f_0$ 成分	系の非線形性に起因すると推測される −3 次高調波
②	$-\Delta f_0$ 成分	IQ Imbalance に起因すると推測される
③	DC 成分	LO Leakage に起因すると推測される [7]
④	$+5\Delta f_0$ 成分	系の非線形性に起因すると推測される 5 次高調波

これらのスプリアスはオフセット周波数を変更しても同様に発生した．DC 成分は中心周波数帯に発生する為，中心周波数の帯域にチャンネルを設置しないことで回避できるが，その他の成分はオフセット周波数に依存する為，オフセット周波数を変更するとそれに対応して周波数帯が変化する．これらのスプリアスは本論文で用いた実験系にお

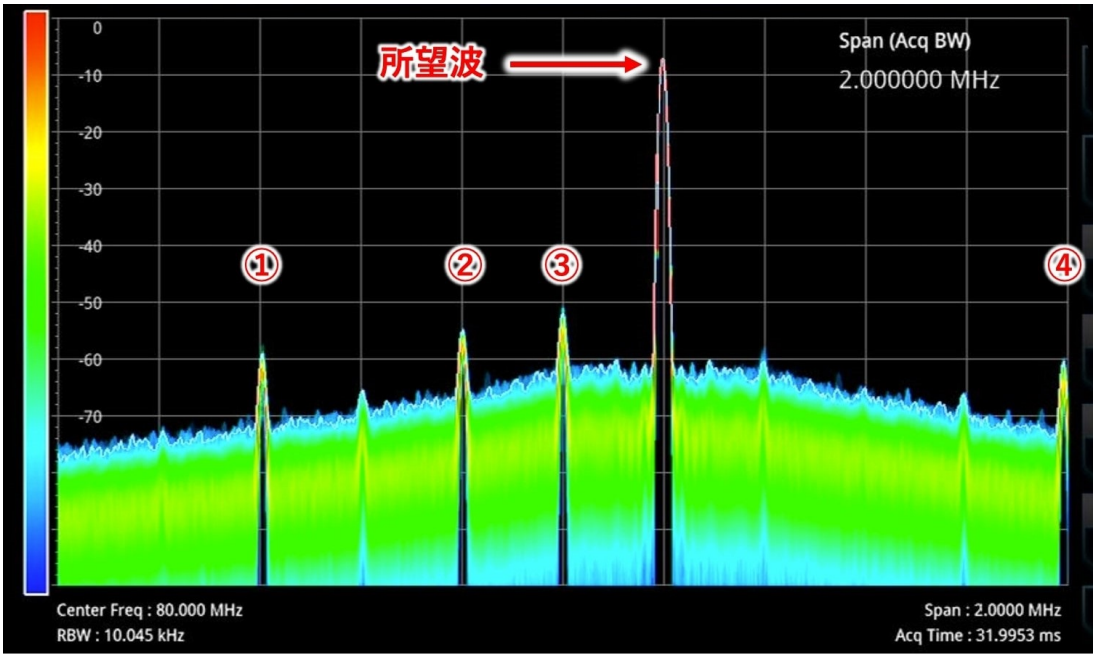

図 3.3 1ch の場合のスペクトラム

いては抑制や回避が難しいが，所望波と比較して十分に小さい為，無視しても問題ないと考えられる．

3.2.3　LO Leakage による DC 成分について

LO Leakage を原因とするスプリアスを図 3.4 に示す．変調信号はキャリアと比較して十分に小さい為，ここでは変調信号のないキャリアのみで考える．図 2.4 に示すように，本実験で使用するキャリアは中心周波数 f_0 とオフセット周波数 Δf を乗算して生成している．SDR デバイスからの出力は理想的には $f_0 + \Delta f$ 成分のみであるが，実際には f_0 成分や Δf 成分も僅かに出力される．Δf 成分は所望波から大きく離れており，バンドパスフィルタで除去することができるが，f_0 成分は所望波と近くフィルタで除去することができない．これに対し f_0 と同じ振幅で逆位相の信号を生成することでキャンセルする方法があるが [7]，本研究ではナイキスト周波数の制限により f_0 と同じ周波数の波形を生成できない為，中心周波数帯にチャンネルを設置しないことで回避する．

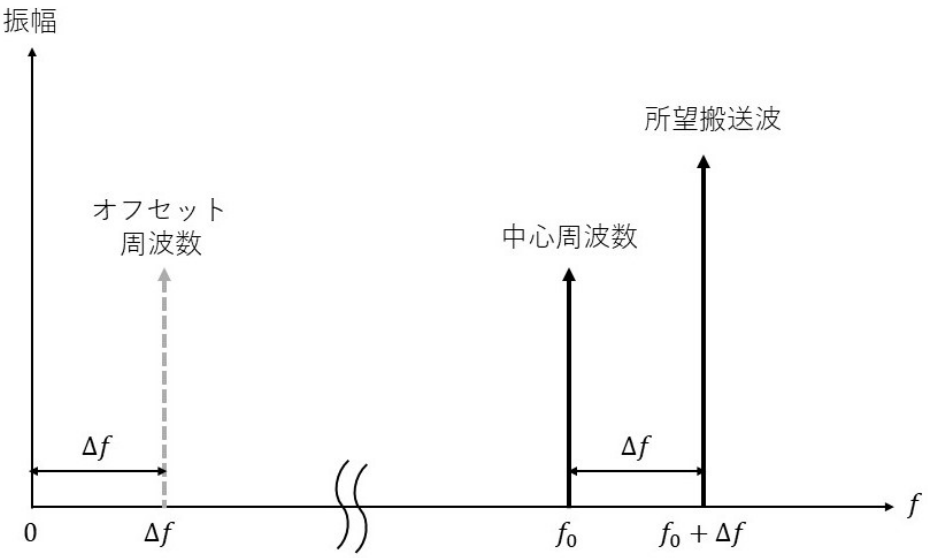

図 3.4　所望波と共に発生する LO Leakage 起因のスプリアス

3.2.4　2ch の場合

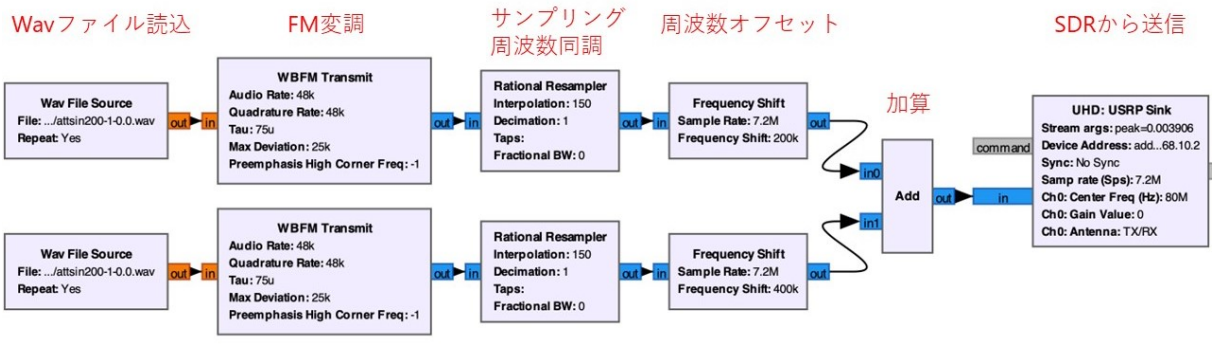

図 3.5　2ch のフローグラフ

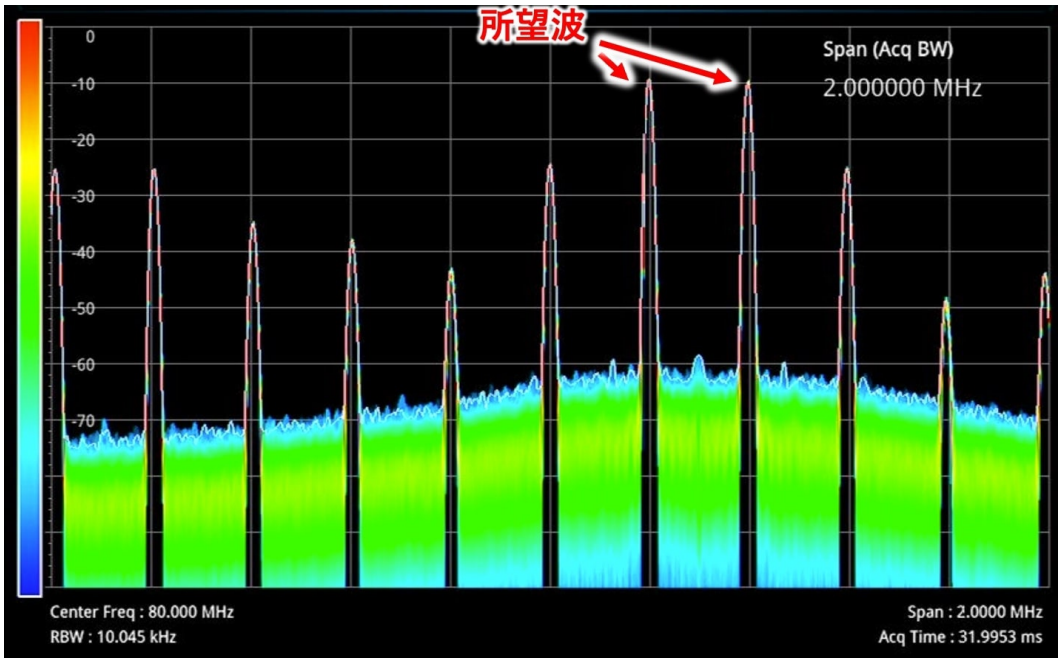

図 3.6 2ch の場合のスペクトラム

本項では 2ch の送信について実験した．文献 [1] では中心周波数 90.0MHz 帯と +200kHz の周波数オフセットを掛けた 90.2MHz 帯の 2ch としていたが，3.2.2 項で述べたように，周波数オフセットを掛けた場合中心周波数帯に DC 成分が生じる．DC 成分との干渉を避ける為，周波数オフセット無しの周波数帯にはチャンネルを配置せず，オフセット周波数 +200kHz，+400kHz の 2ch とした．

実験に用いたフローグラフを図 3.5 に，スペクトラムを図 3.6 に示す．図 3.6 は 79.0MHz から 81.0MHz までのスペクトラムを表示している．1ch の信号の場合はスプリアスの数が少なく強度も小さい一方，2ch の信号を送信した場合には多数のスプリアスが発生し，大きいものだと所望波と 15dB 程度しか強度の差がないことがわかる．

スプリアスの原因を推測するためには，所望波とスプリアスの対応関係を明確化する必要がある．前述の実験ではオフセット周波数を +200kHz，+400kHz としていたが，オフセット周波数を微小に変化させ，スプリアスの変化を観測した．

3.3　相互変調によるスプリアス

3.3.1　スプリアスの変化を観測する実験

一方のチャンネルを $f_2 = f_0 + \Delta f_2 = 80.4\text{MHz}$(オフセット $\Delta f_2 = +400\text{kHz}$) に固定し，もう一方のチャンネルを $f_1 = f_0 + \Delta f_1 = 80.0\text{MHz}$(オフセット $\Delta f_1 = 0\text{kHz}$) から $f_1 = 80.2\text{MHz}(\Delta f_1 = +200\text{kHz})$ まで変化させた際のスペクトラムを 50kHz 毎に記録したものを図 3.7 に示す．図中に青矢印と黄矢印で示すスプリアスについてその原因を推測する．青矢印で示すスプリアスは，$\Delta f_1 = 0\text{kHz}$ の時 79.6MHz 帯に発生し，Δf_1 の 2 倍の周波数分だけ周波数が減少していく．黄矢印で示すスプリアスは，$\Delta f_1 = 0\text{kHz}$ の時 79.6MHz 帯に発生し，Δf_1 の 2 倍の周波数分だけ周波数が増加していく．これらは，

$$79.6\text{MHz} \pm 2\Delta f_1$$

と表せる．$79.6\text{MHz} + 2\Delta f_1$ は，

$$2f_1 - f_2$$

と変換することができる．$79.6\text{MHz} - 2\Delta f_1$ は，

$$2(f_0 - \Delta f_1) - f_2$$

と変換することができる．

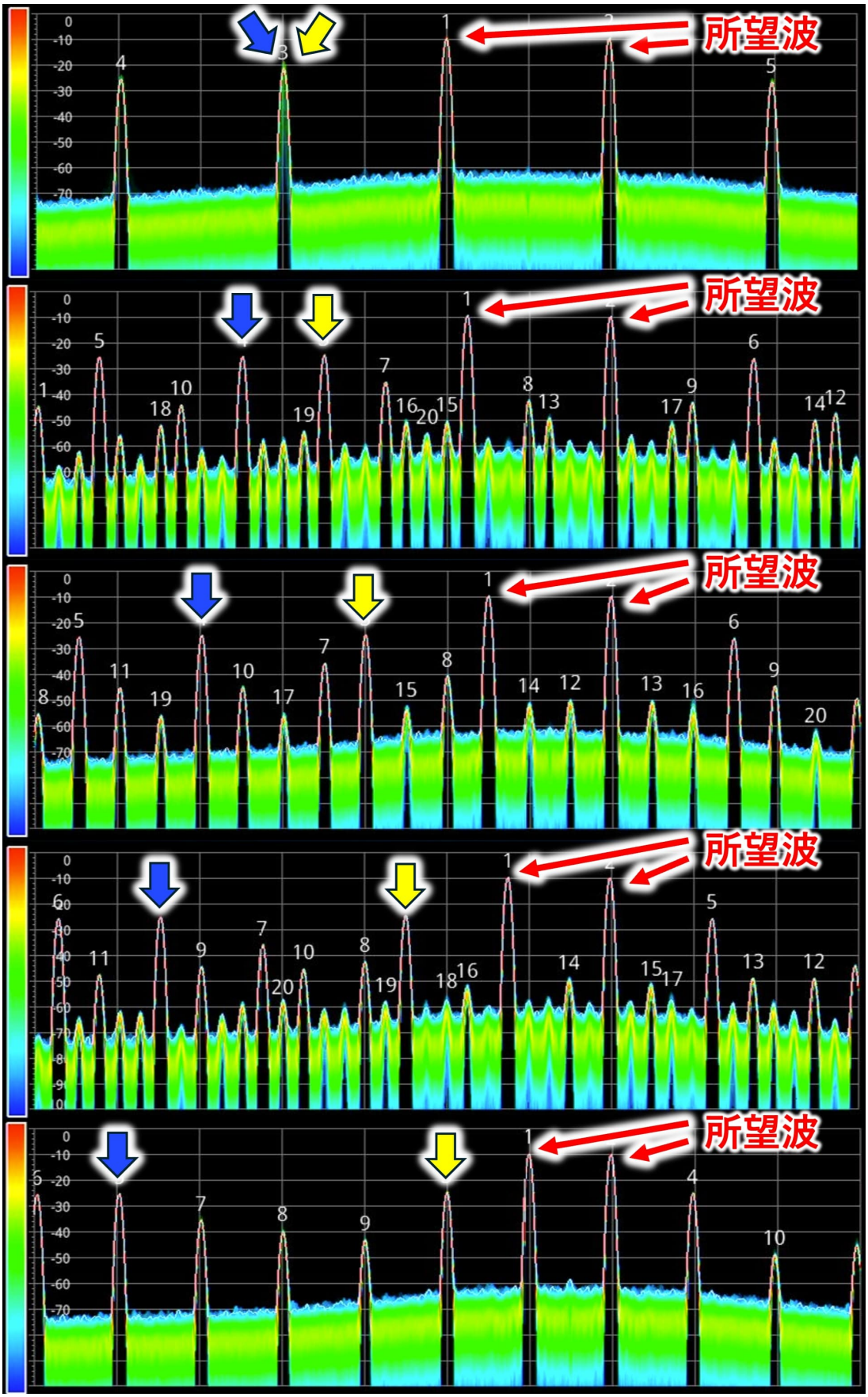

図 3.7 Δf_1 を 0kHz から 200kHz まで変化させた場合のスプリアスの変化．上から順に Δf_1 が 0kHz，50kHz，100kHz，150kHz，200kHz の時のスペクトラムを表示している．

3.3.2　三次相互変調の影響

SDR デバイスは非線形回路であり，非線形性により相互変調が発生する．入力の二乗に比例する成分は，

$$(e^{j\times 2\pi f_1 t} + e^{j\times 2\pi f_2 t})^2$$

$$= e^{j2\times 2\pi f_1 t} + e^{j2\times 2\pi f_2 t} + 2e^{j\times 2\pi (f_1+f_2)t}$$

と表され，$|f_1 \pm f_2|$ 成分が二次相互変調積によるスプリアスとして発生する．これらは所望波から大きく離れておりバンドパスフィルタによる除去が可能である．入力の三乗に比例する成分は，

$$(e^{j\times 2\pi f_1 t} + e^{j\times 2\pi f_2 t})^3$$

$$= e^{j3\times 2\pi f_1 t} + 3e^{j\times 2\pi (2f_1+f_2)t} + 3e^{j\times 2\pi (2f_2+f_1)t} + e^{j3\times 2\pi f_2 t}$$

と表され，$|2f_1 \pm f_2|$ 成分や $|2f_2 \pm f_1|$ 成分が三次相互変調積によるスプリアスとして発生する．項の間の符号が正の場合所望波から大きく離れる為バンドパスフィルタによる除去が可能である．一方符号が負の場合，f_1 と f_2 の値が近いとき三次相互変調積も所望波近傍に発生する．

以上のことから，$2f_1 - f_2$ や $2(f_0 - \Delta f_1) - f_2$ は三次相互変調によるスプリアスであると推測される．$2f_1 - f_2$ 成分は二つの所望波による三次相互変調積であり，$2(f_0 - \Delta f_1) - f_2$ も $(f_0 - \Delta f_1)$ 成分と f_2 の三次相互変調である．$(f_0 - \Delta f_1)$ 成分について考える．

図 3.3 に示すように，1ch の送信の場合には比較的大きなスプリアスが 4 成分発生する．2ch での送信時にも，それぞれのチャンネルに対し同様のスプリアスが発生していると考えられる．その様子を図 3.8 に示す．$f_1 = 80.2$MHz(赤色)，$f_2 = 80.4$MHz(青色)のそれぞれに対し表 3.1 に示す成分が発生していると考えられる為，それらを重ねて表示した．相互変調は一方の所望波と，もう一方の所望波に起因するスプリアス（図 3.3 に示したスプリアス）との間でも発生していると考えられる．よって $(f_0 - \Delta f_1)$ 成分は，図 3.3 における①に相当する IQ Imbalance に起因するスプリアスと推測され，$(f_0 - \Delta f_1)$ 成分と f_2 の三次相互変調積が青矢印のスプリアスの原因であると考えられる．

他のスプリアスも同様に相互変調によるスプリアスであると考えられる．これは SDR 出力段の非線形性による相互変調であり，三次相互変調積は出力レベルを 1dB 下げた場合 3dB 低減する為，大幅に抑制することができる．出力を 40dB 下げる為，UHD:USRP

Sink ブロックの前に multiply const ブロックを接続して振幅を 1/100 倍した．フローグラフを図 3.9，スペクトラムを図 3.10 に示す．DC 成分を除く多数のスプリアスがノイズフロア以下に低減していることがわかる．

図 3.10 より，出力強度を 40dB 下げた場合の所望波の強度は −50dBm 程度であることがわかる．これは受信デバイスに使用する FM ラジオ受信用 IC の要求する強度よりも十分に強い為，受信に問題を生ずることなく出力を下げることができる．

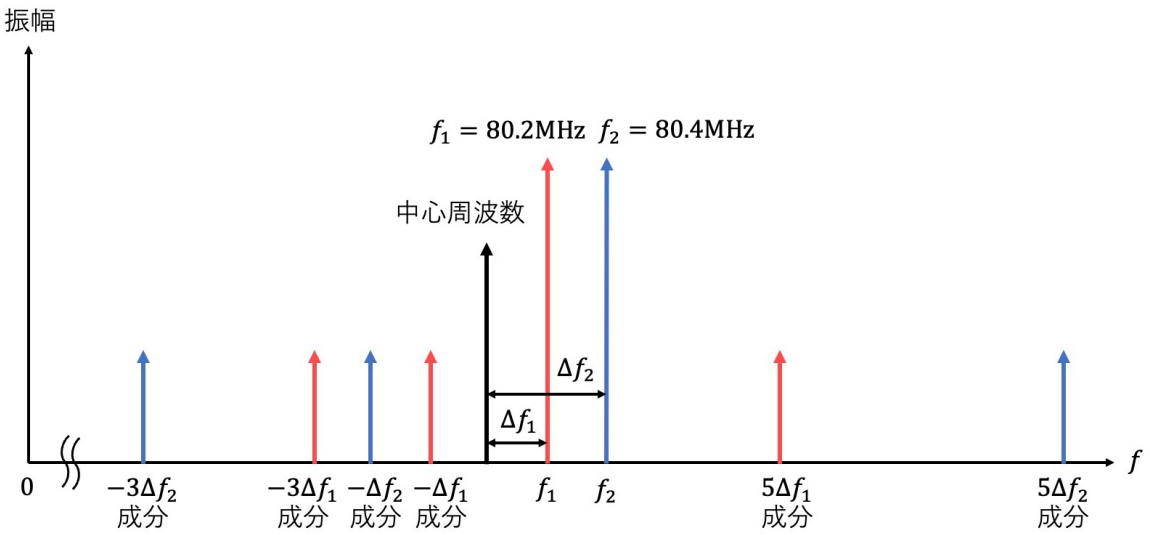

図 3.8 図 3.3 に示したスプリアスを 2ch 分重ね合わせた図．2ch の場合にも発生していると考えられる．

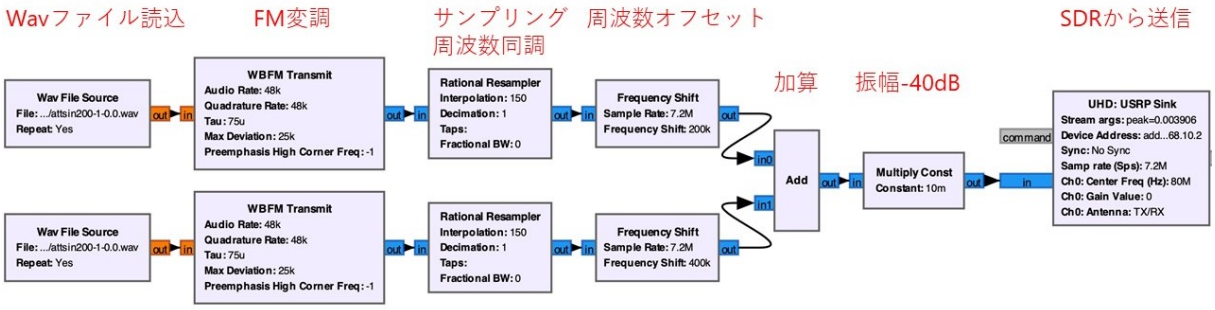

図 3.9 振幅を 1/100 にするフローグラフ

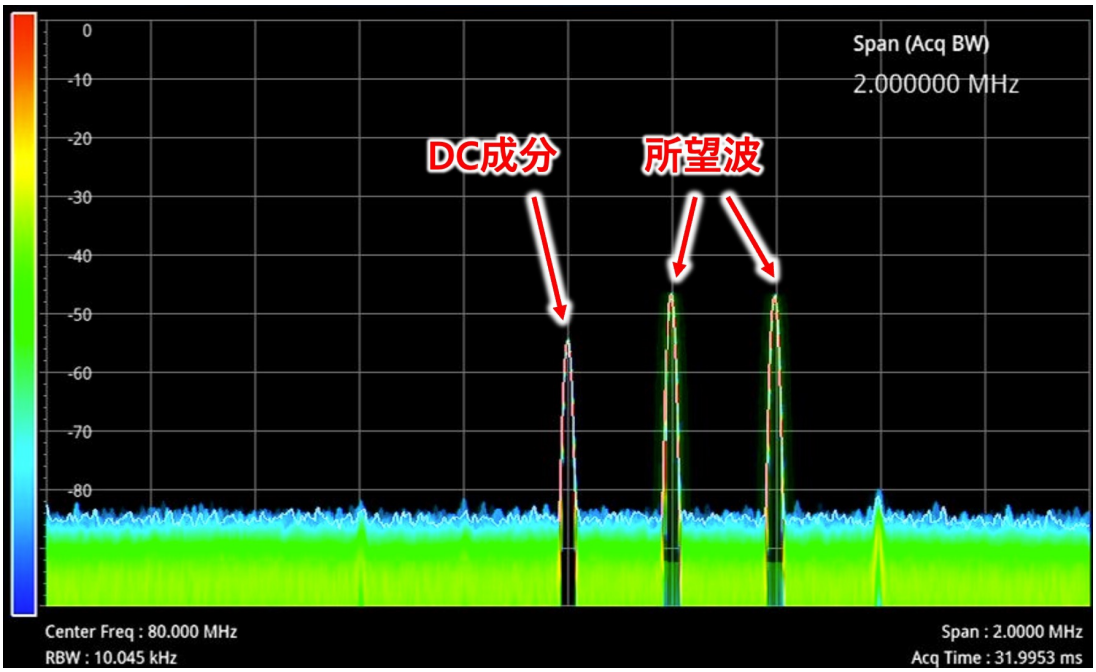

図 3.10　出力を-40dB した場合のスペクトラム

3.3.3　チャンネルを増やした場合の実験結果

文献 [1] の実験は，複数の変調波を加算した被変調波を複数のデバイスに送信し個別に復調できることを示すもので，最小構成の 2ch で実験が行われている．チャンネル数を実用的な数まで増やした場合にも同様に送信が可能であるか，構成を複雑化した場合にのみ起こり得る問題等の有無について確認した．同様の実験でチャンネル数を 40ch に増やして行った場合のスペクトラムを図 3.11，3.12 に示す．図 3.11 がデフォルトの出力強度で，図 3.12 が -40dB したものである．チャンネルは 76.0MHz から 84.0MHz まで，中心周波数の 80.0MHz を除き 200kHz 間隔で設置している．この実験ではオフセット周波数の最大値が 4.0MHz となり，サンプリング周波数 7.2MSps の場合のナイキスト周波数 3.6MHz を超えてしまう為，サンプリング周波数を 9.6MSps に変更した．2ch の場合と同様に，出力を下げることで多数のスプリアスの抑制が可能であることがわかる．スペクトラム両端付近に発生しているのはナイキスト周波数に近いオフセット周波数を使用したことによる折り返し雑音である．$(80.0 \pm \Delta f)$MHz 成分 (Δf: オフセット周波数 (MHz)) の折り返し雑音は複素信号の為 $(80.0 \pm (\Delta f - f_s))$MHz に発生する ($f_s$: サンプリング周波数)．

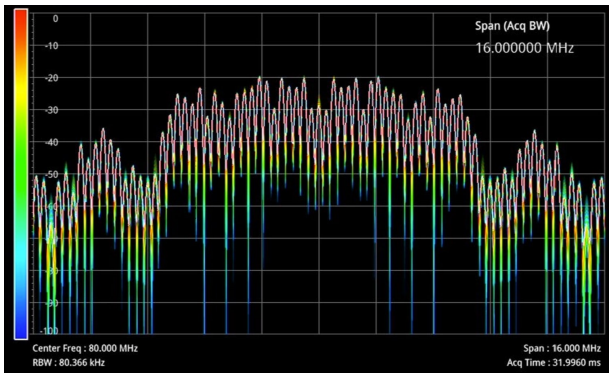

図 3.11 40ch の送信 (デフォルトの出力)

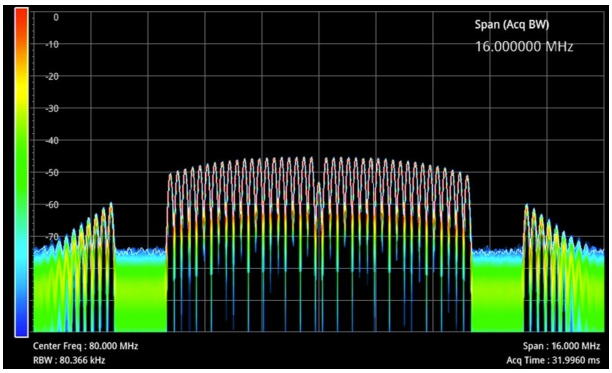

図 3.12 40ch の送信 (出力 −40dB)

3.4 結言

2 チャンネル以上の同時伝送時に発生するスプリアスは，系の非線形性による相互変調に起因するもので，出力レベルを下げることでノイズフロア以下まで抑制することができた．一方で，異なる条件下で発生する異なる原因のスプリアスが存在する．現時点で

1. 3.3.3 項で述べたナイキスト周波数付近まで周波数オフセットを掛けた場合の折り返し雑音 (図 3.10)
2. サンプリング周波数が大きい場合に発生するスプリアス (図 3.13)

が挙げられる．数十チャンネル以上を扱う処理では干渉の要因になり得る．これらのスプリアスについても抑制方法の検討が必要である．

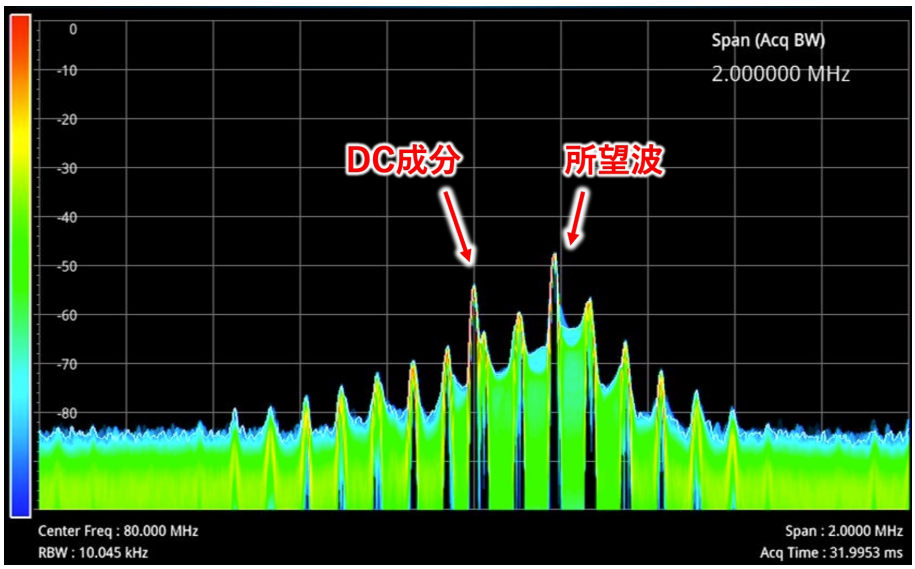

図 3.13 サンプリング周波数が大きい場合に発生するスプリアス．この図では 32MSps で実験を行っている．

第 4 章　ユーザ入力をトリガとする任意波形送信

本章では，GRC で自動生成されるプログラムにおける課題である任意波形送信ができないことについて，その原因の調査と解決手法を提案する．

4.1　GRC で自動生成されるプログラムについて

文献 [1] 及び前章で用いている GRC で自動生成されたプログラムは，予め作成した Wav ファイル形式の振動波形を FM 変調してオフセットを掛け送信するという処理を連続的に繰り返すというものである．フローチャートを図 4.1 に示す．送信するチャンネル (オフセット周波数) や振動周波数を変更できるのは実行前であり，VR や AR と組み合わせて映像や操作と同調した触覚呈示にそのまま利用することができない．GRC には if 文のような条件分岐を実装するブロックが実装されておらず，GRC のみで映像との同期やユーザ入力をトリガとした動作は実装できない．

GNU Radio ライブラリには GRC で自動生成することができない要素があり，それらは GRC を使わず直接スクリプトを記述することで実装可能である．また直接記述することでフローグラフでは実装できない条件分岐を利用することができ，実用的に動作するシステムの実現が期待できる．その為本章では GRC で自動生成されたスクリプトの問題点を調査し，スクリプトを編集して GRC では扱うことができない目標の動作を実装する．

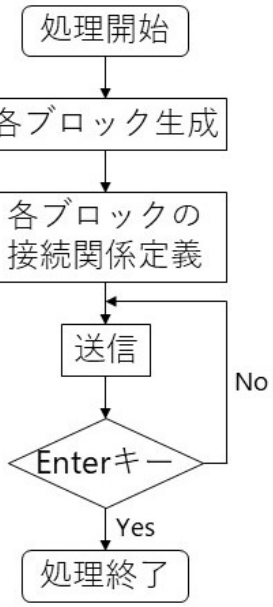

図 4.1 GRC で自動生成されるスクリプトのフローチャート

4.2 GRC で作成したプログラムの課題点

GRC で作成したプログラムを実行すると，実際にスペクトラムアナライザで波形が表示されるようになるまで数秒を要する．このレスポンスの悪化の原因を調査した．

図 3.2 を基に，構成を単純化して考える．Wav File Source ブロックから Frequency Shift ブロックまでを全て削除し，Signal Source ブロック (単調な正弦波を発生させるブロック) に置き換えた．フローグラフを図 4.2 に示す．このフローグラフにより，信号波を FM 変調する処理がなく，搬送波のみを生成する．発生させる正弦波の周波数は 200kHz に設定しており，搬送波周波数は 80.2MHz となる．

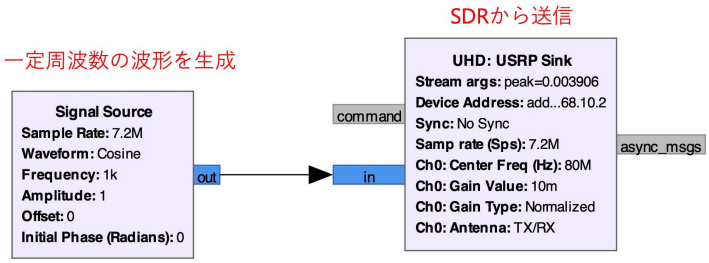

図 4.2 構成を単純化したフローグラフ

このフローグラフを実行した場合にも，同様の遅れが発生した．このことから Wav File Source - Frequency Shift ブロック間の FM 変調に係る処理が遅延の原因ではないこ

とがわかる．

次に，図 3.2 の UHD: USRP Sink ブロックを QT GUI Sink ブロック (コンピュータ上の GUI でスペクトラムを表示する (図 4.4) ブロック) に置き換えた．フローグラフを図 4.3 に示す．このフローグラフでは USRP との接続を行わず，被変調波を GUI 上に表示する．

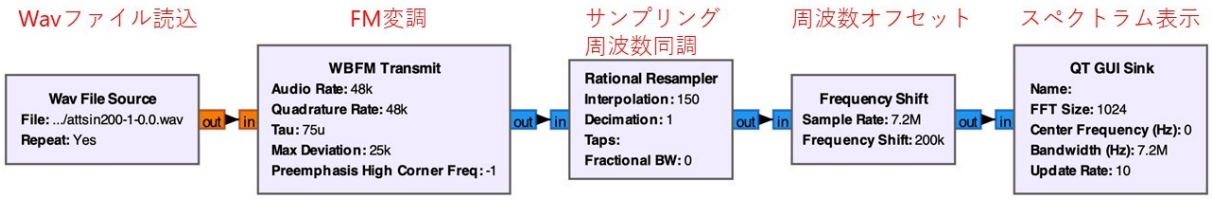

図 4.3　QT GUI Sink ブロックを用いたフローグラフ

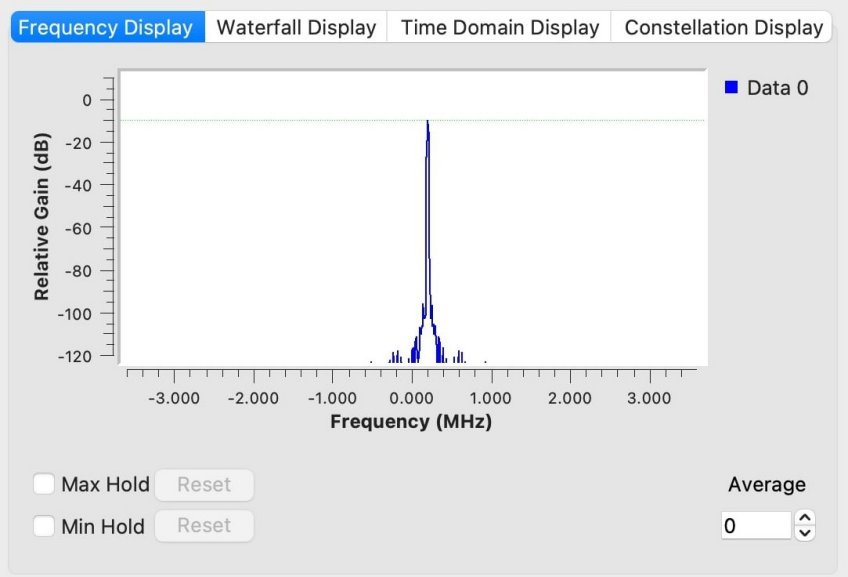

図 4.4　QT GUI Sink ブロックでコンピュータ上に表示させたスペクトラム

このフローグラフを実行した場合には遅延が大幅に減少した．このことから UHD: USRP Sink ブロックがレスポンスを悪化させていると考えられる．残る遅延は Python プログラム実行時に必ず発生するものであり，この遅延はプログラムを停止させずに処理を行うことで初回実行時以外は回避することが可能である．

GRC で自動生成されたプログラムを実行する場合，振動周波数や周波数オフセットを変更するたびに USRP との接続を切断し，実行時に再度接続するという処理が必要となる．USRP を用いないフローグラフで実験した結果遅延が減少したことから，USRP デバイスとの接続に係る処理がレスポンスを悪化させていることがわかった．そこで，USRP との接続を維持したまま，送信する振動波形の Wav ファイルやオフセット周波数のみ変更するというプログラムを考える．

4.3 GRC のプログラムとの変更点

GRC で自動生成されたスクリプトを付録 A.1 に，スクリプトを編集して所望の動作を可能にしたものを付録 A.2 に示す．A.2 では，実行後にキーボードの入力によってオフセット周波数を変更して振動波形を一度送信するというものを掲載した．フローチャートを図 4.5 に示す．

表 4.1 キー入力とオフセット周波数の関係

キー入力	オフセット周波数
a	-400kHz
s	-200kHz
d	+200kHz
f	+400kHz
g	+600kHz

実装したプログラムはキー入力に応じて異なるオフセットを掛けた触覚刺激波形を送信する．表 4.1 にキー入力とオフセット周波数の対応関係を示す．現段階でのインターフェースはターミナル上で直接キーボード入力を行う方式であるが，インターフェースの改変により専用 GUI を作成したり専用の入力デバイスを用いた制御も可能である．

本論文で用いているフローグラフにおいて，GRC で自動生成されるスクリプトでは使用されない要素が disconnect 関数である [8]．図 4.5 中の “各ブロックの接続解除”に相当する処理である．GRC で自動生成されるスクリプトにも含まれる connect 関数 (図中の “各ブロックの接続関係定義”) で各ブロックの接続関係を定義しており，そのままでは実行を停止するまで接続関係が維持されるが，disconnect 関数を用いることで実行中に接続関係を解除することができる．接続関係が定義された後は各ブロックのパラメータ変更や再生成ができない為，disconnect 関数を用いて各ブロックの接続関係を解除することで Wav File Source ブロックの再生成やオフセット周波数の変更が可能になる．

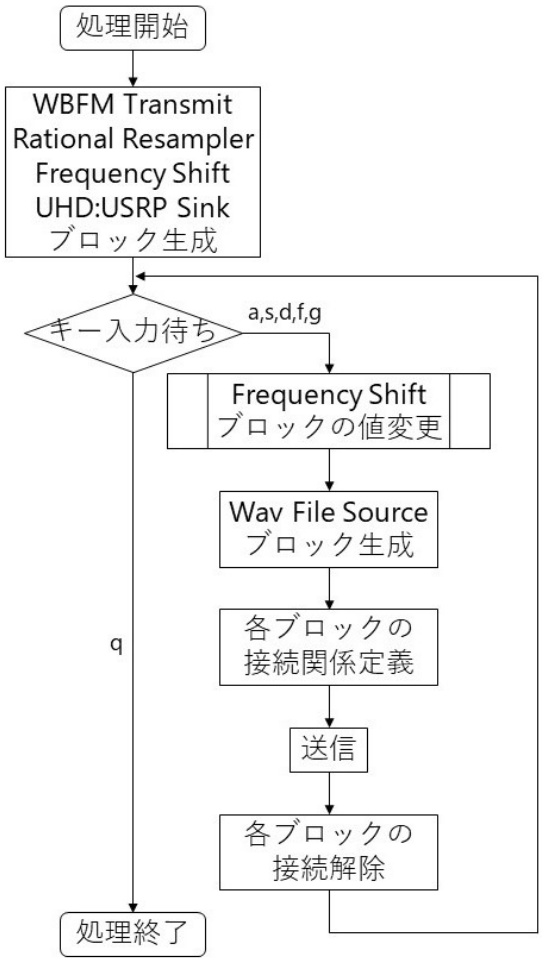

図 4.5　自動生成されたスクリプトを編集して実装したフローチャート

4.4　結言

本章で実装したプログラムで自由度の高い制御ができるシステムが実装できたが，76.0MHz から 108.0MHz までの全ての周波数帯にチャンネルを設置して実用的に運用するには処理負荷が大きいという課題がある．

4.4.1　周波数オフセットと処理負荷の関係

本論文で用いている周波数オフセットとは，変調信号に所望するオフセット周波数の単調な信号を乗算するというものである．キャリア周波数が大きくなると，すなわちチャンネル数が増えるほど変調信号に乗算するオフセット周波数も大きくなる．オフセット周波数がナイキスト周波数を超えることがないようにサンプリング周波数を設定する必要がある．しかしサンプリング周波数を増加させることでコンピュータの負荷は増大する為，設置できるチャンネル数と処理負荷の大きさはトレードオフとな

る．本研究で用いた受信デバイスで 76.0MHz から 108.0MHz に 200kHz 間隔でチャンネルを配置した場合，160ch まで増設可能である．しかし 76.0MHz から 108.0MHz までの全てのチャンネルに振動波形を送信する場合，オフセット周波数の最大値は中心周波数 92.0MHz の時 16.0MHz となる．この場合サンプリング周波数は 32.0MSps 以上に設定する必要があり，一般的なコンピュータで安定して動作させることは難しい．システム全体を身体に装着できるサイズにすると想定した場合，コンピュータの大きさも必然的に小型となる為，処理負荷の軽いプログラムが望まれる．

4.4.2 処理負荷が大きいプログラムの有用性

高い処理性能を持つコンピュータは大型であり，身体に取り付けられるような小型軽量のシステムで実行することは難しく，現行の VR 機器にもあるように据え置き型のコンピュータとケーブルで接続して処理を行うことになる．ケーブルの届く範囲内でしか動けない為，運動等の激しい動きに対応したシステムに組み込むことは難しい．一方手持ちのコントローラーのみで完結するような操作や，操作の必要がない映像コンテンツ等に対して触覚刺激を返すシステムであれば実現の可能性がある．

第 5 章　低負荷で実行できるシステムの提案

本章では前章で作成したプログラムの課題である処理負荷が大きいことについて，その解決手法し提案する．

5.1　処理負荷が高くなる要因と抑制の手法

3 章，4 章で用いているプログラムは処理負荷が大きく，処理性能がある程度高いコンピュータで実行する必要がある．本章ではコンピュータ，SDR デバイスを含めた系を身体に装着することを想定した場合のシステムを考える．

5.1.1　処理負荷を抑える方法

GRC で作成したプログラムで，処理負荷を増大させる要因となっている最も大きな要素はサンプリング周波数である．サンプリング周波数を下げる為にはオフセット周波数を小さくする必要があるが，前述の通り受信デバイスは 76.0MHz から 108.0MHz までの周波数に対応している．周波数オフセットを掛ける方式でサンプリング周波数を下げるとオフセットを掛けられる範囲が小さくなり，できる限り広い周波数帯を活用する為には現実的ではない．

そこで周波数オフセットを掛けず，SDR の中心周波数を随時変更するという方式で被変調波の周波数を変更するプログラムを作成した．ブロック線図を図 5.1 に示す．図 2.4 に示す前章まで用いた概念図では中心周波数 f_0 を固定し，それぞれの変調信号に乗算する $e^{jN\times 2\pi\Delta f}$ の N を変更することでキャリア周波数を変更していた．本章では f_0 の値を変更することでオフセットを掛けずにキャリア周波数を変更できる機能を実装した．この方式では一度に送信できる触覚刺激波形は 1ch のみに限定されるが，周波数オフセットを掛けない為，サンプリング周波数の設定にあたり変調信号の周波数以外を考慮する必要がない．変調信号の周波数はオフセット周波数と比較して十分に小さい為，処理負荷の大幅な低減が可能である．5.2 節では，作成したプログラムを小型のコンピュータで実行し，安定して動作させられる処理負荷を実現可能であるか調査した．

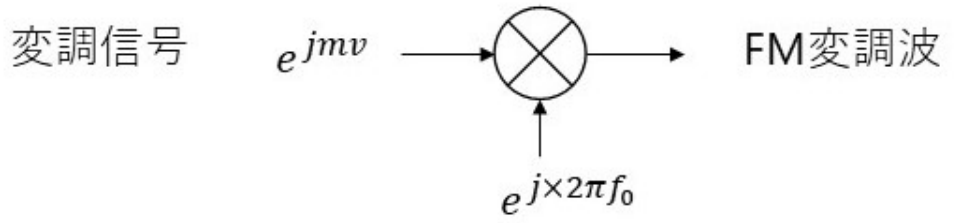

図 5.1 周波数オフセットを用いない 1 チャンネル FM 変調波の生成

5.2 小型のコンピュータを用いた実験

5.2.1 使用機材

小型のコンピュータには RaspberryPi 4 を用いる．SDR デバイスは N210 を使用するが，本研究で SDR デバイスに N210 を使用した理由は本機がドライバなどを必要とせずにコンピュータと接続可能である為である．本機よりも小型安価である SDR デバイスでは不足する要素があるわけではなく，文献 [1] で用いられている USRP B205mini-i のような小型の SDR デバイスと RaspberryPi を組み合わせたシステムは十分に実現可能である．

5.2.2 4 章のプログラムとの相違点

4 章で作成したプログラムとの相違点について説明する．

このプログラムではオフセット周波数を設定せず，Wav ファイル形式の振動波形を FM 変調し，変調信号を随時変更可能な中心周波数で送信する．チャンネル数は 1ch に限定されるが表 4.1 に示す周波数オフセットを掛けた被変調波と同等の被変調波を得られる機能を実装した．周波数オフセットを掛けない為ナイキスト周波数を考慮する必要がなく，送信毎に中心周波数を変更することでサンプリング周波数に依存せず自由に周波数を変更できる．サンプリング周波数は GNU Radio の制約内の下限値である 195312Sps(約 195kSps) まで下げることができた．

5.3　実験結果

3.2.2 項で作成した GRC フローグラフを RaspberryPi 4 で実行した場合のスペクトラムを図 5.2 に示す．1ch のみの送信で 3 章と 4 章で用いたプログラムで最も処理負荷が軽いものであるが，大量のランダムノイズが発生しており，ターミナル上でアンダーフローの発生を示すエラーが多発した．また実行後にコンピュータがフリーズして数秒間動作不能になる．このままでは実用的な触覚呈示は不可能である為，次に RaspberryPi 4 用に改変したプログラムを実行した．その時のスペクトラムを図 5.3 に示す．ランダムノイズやエラーは発生せず，実行後も正常に動作した．また，図 5.2 にもある通り周波数オフセットを掛ける方式では中心周波数帯に DC 成分によるスプリアスが発生するが，図 5.3 では周波数オフセットを掛けていない為，DC 成分にあたるスプリアスが発生しない．

また，実行中の CPU 使用率をモニタリングしたところ，20% 未満で安定している．これよりも多少処理負荷の大きいプログラムでも十分に動作すると考えられ，プログラムの改変の余地があると期待できる．

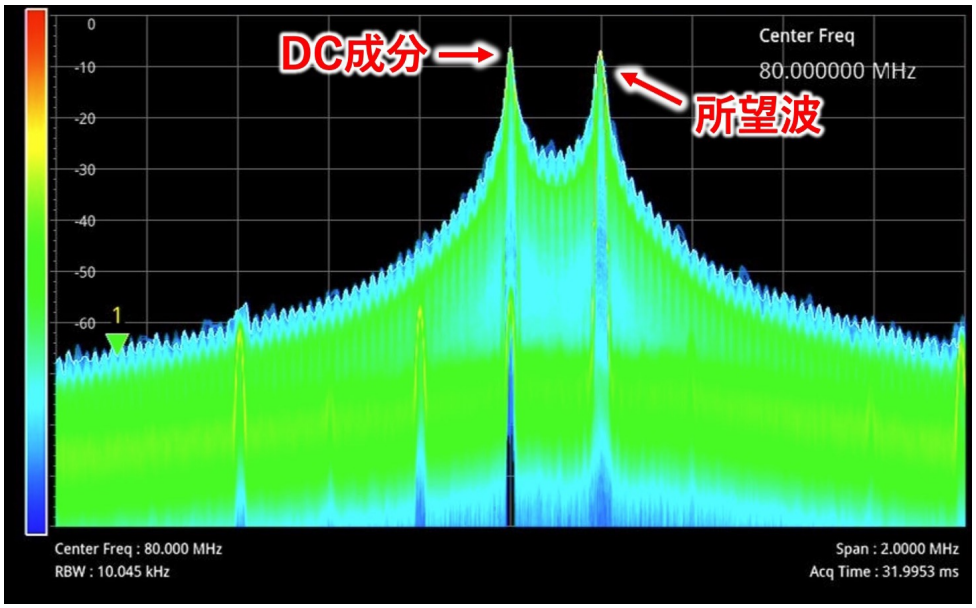

図 5.2　RaspberryPi 4 で図 3.2 を実行した場合のスペクトラム．大量のランダムノイズが発生している．

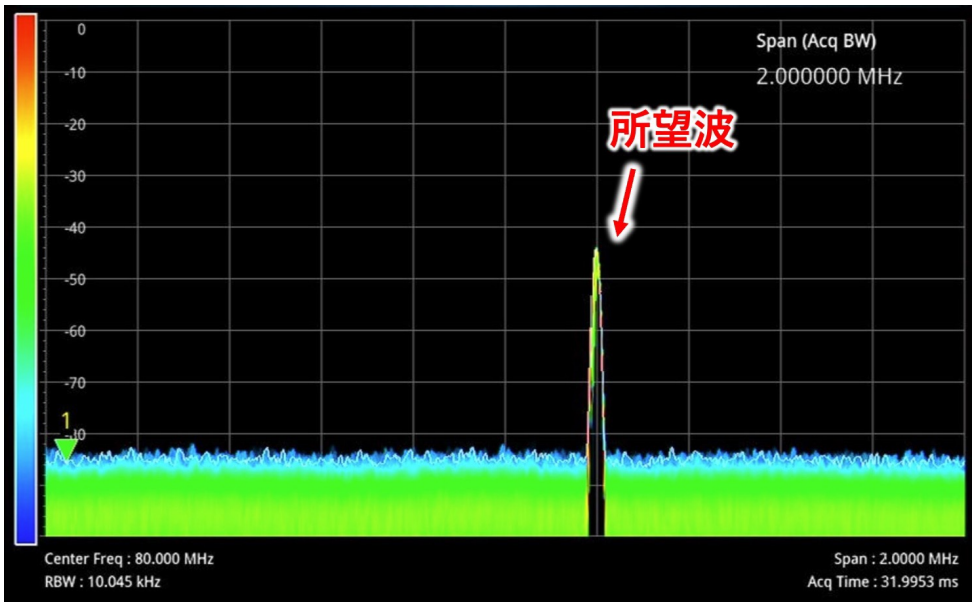

図 5.3 RaspberryPi 4 用に処理負荷を抑えたプログラムを実行した場合のスペクトラム

5.4 結言

5.4.1 1ch に限定されるプログラムの有用性

本章で実装したプログラムにおける 1ch に限定されるという制約は，一度に送信できるチャンネル数が 1ch に限定されるという意味である．すなわち複数のチャンネルで同時に振動波形を送信することはできないが，タイミングの異なる振動波形送信であれば実現可能である．全身に触覚刺激を与えるシステムにおいて，複数チャンネルで同時に触覚刺激を与える必要のあるシチュエーションが存在するか，また回避可能であるかは未検証である．このプログラムの有用性を示すには，実用上で単一チャンネルのみの送信でも触覚フィードバックとしての表現が可能であるかの検証が必要である．

5.4.2 本章における SDR を使用することの合理性

本研究では多チャンネルの変調信号を加算した被変調波を得るための手段として SDR を用いている．一方本章で提案したプログラムではチャンネル数が 1ch に限定されるため，SDR の採用理由と相反する．この場合 SDR ではなく市販の FM トランスミッタを使用することでも同様の被変調波を得ることができると考えられる．FM トランスミッタであれば SDR デバイスと比較してかなり小型軽量で且つ小電力である．しかし FM トランスミッタは周波数を瞬時に複数回切り替えながら使用することは通常想定されておらず，そのような動作においての遅延の有無や耐久性の検証が必要である．

5.4.3　低負荷と多チャンネル同時送信を両立させるシステムの考察

ここで 4 章と本章のプログラムを組み合わせて考えられるプログラムについて述べる．中心周波数を随時変更し，オフセット周波数が最小となるような組み合わせを計算するプログラムが考えられる．本章のプログラムではサンプリング周波数が約 195kSps でオフセットを掛けるには不十分であるが，400kSps 以上あれば隣のチャンネル (オフセット周波数 ±200kHz) を用いた多チャンネル同時送信が可能である．オフセット周波数をなるべく小さく抑えることでサンプリング周波数を下げることができ，処理負荷を抑えながら多チャンネルに同時送信が可能になる．一方この方式は複数のチャンネルの周波数帯が大きく離れた場合には 4 章で実装したプログラムと同等の処理負荷が要求される．多くの場合で複数チャンネルの周波数帯が大きく離れることがない配置や処理の方法が示されれば，このプログラムが有用であると考えられる．

5.4.4　安価に実装できるシステムの実装例

RaspberryPi と組み合わせた場合に合理的なデバイスとして ADALM-PLUTO(Analog Devices) [9] が挙げられる．ADALM-PLUTO は N210 や文献 [1] で用いられた B205mini-i よりも安価に入手可能で，こうした安価なデバイスでも実装が可能であることが示されると実現の期待が高まる．N210 を ADALM-PLUTO に置き換えた実験系を図 5.4 に示す．図 2.1 では電圧を重畳する電源に直流電源装置を用いたが，ここでは USB 電源を使用する DC-DC converter に置き換えることで系を大幅に小型化できる．RaspberryPi と ADALM-PLUTO の電源を接続すると，本章で実装したプログラムを動作させることが可能である．

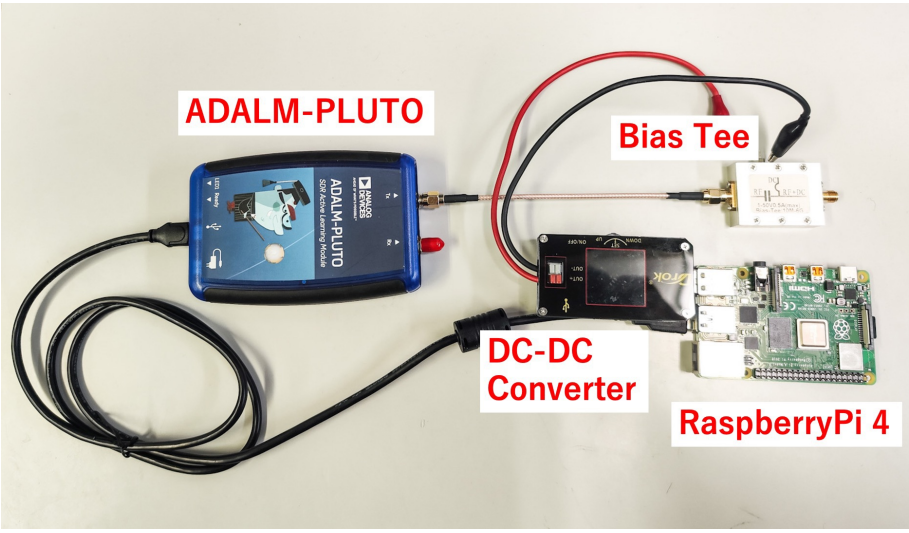

図 5.4　ADALM-PLUTO を使用するシステムの実装例．身体に取り付けての使用が可能な大きさに収まっている．

第 6 章　まとめ

本論文では，SDR を用いた触覚刺激波形の送信に係る課題についての調査，解決を述べた．スプリアスの発生については，原因が系の非線形性による相互変調であることが判り，出力強度を下げることでスプリアスの強度をノイズフロア以下まで抑制することができた．ユーザ入力をトリガとした任意波形送信は，自動生成されたスクリプトを編集することでキー入力に応じて触覚刺激を与える機能を実装できた．処理負荷が大きくなる課題については，1ch のみという制約はあるが低負荷で動作するシステムを実現できた．一方，提案したプログラムは処理負荷やそれに伴う制約により，そのまま自在な触覚呈示システムに利用することは難しい．さらに制約を軽減する手法の検討や，配置するボイスコイルアクチュエータの数や同時に送信するチャンネル数などの具体的な仕様の検討が必要である．先行研究で原理的な可能性のみ示されていた触覚刺激手法であるが，本研究で得た成果により SDR と多チャンネル FM 伝送を用いた実用的な触覚フィードバックシステムの実現が期待できる．

謝辞

本論文の執筆及び本研究においてご指導して頂いた野田 聡人准教授に心より感謝申し上げます.

また，本論文の執筆にあたり助言，協力いただいた野田研究室，小林研究室の学生諸氏に心より感謝申し上げます.

並びに，本研究のプログラム実装において助言を頂いた情報学群 田之上 陽向氏に心より感謝申し上げます.

参考文献

[1] 野田聡人，“導電布伝送路上でのアナログ波形の多チャンネル同時送信”，信学技報，vol. 122，no. 277，pp. 19–22，2022．

[2] 篠田 裕之，“二次元通信による生活支援空間”，横幹連合コンファレンス予稿集，vol. 2007，pp. 36–36，2007．

[3] A. Noda and H. Shinoda, “Frequency-division-multiplexed signal and power transfer for wearable devices networked via conductive embroideries on a cloth,” in *2017 IEEE MTT-S International Microwave Symposium (IMS)*, 2017, pp. 537–540.

[4] ——, “Simplex Inter-IC for Wearables and Its Applications,” *IEEE Access*, vol. 9, pp. 69 654–69 662, 2021.

[5] A. Noda, “Multi-channel fm transmission of vibrotactile signals on 2-d communication textile,” *IEICE Communications Express*, vol. 11, no. 5, pp. 195–201, 2022.

[6] “GNU Radio - The Free & Open Source Radio Ecosystem,” https://www.gnuradio.org/.

[7] Dave Frizelle，“Transmit LO Leakage(LOL)—An Issue of Zero-IF That Isn’t Making People Laugh Out Loud”，analog devices，2017．

[8] GNU Radio, “Handling Flowgraphs.” [Online]. Available: https://wiki.gnuradio.org/index.php/Handling_Flowgraphs

[9] “Adalm-pluto 評価用ボード—アナログ・デバイセズ”，https://www.analog.com/jp/design-center/evaluation-hardware-and-software/evaluation-boards-kits/ADALM-PLUTO.html#eb-overview．

付録 A　作成したプログラム

A.1　GRC で生成したプログラム

GRC で作成した，図 3.2 に示すフローグラフで実行されるプログラムを以下に示す．

Listing A.1　GRC で作成したプログラム

```
1  from gnuradio import analog
2  from gnuradio import blocks
3  import math
4  from gnuradio import filter
5  from gnuradio.filter import firdes
6  from gnuradio import gr
7  from gnuradio.fft import window
8  import sys
9  import signal
10 from argparse import ArgumentParser
11 from gnuradio.eng_arg import eng_float, intx
12 from gnuradio import eng_notation
13 from gnuradio import uhd
14 import time
15
16 class TX(gr.top_block):
17
18     def __init__(self):
19         gr.top_block.__init__(self, "Not titled yet", catch_exceptions=True)
20
21 # Variables
22         self.samp_rate = samp_rate = 72e5
23
24 # Blocks
25
26         self.uhd_usrp_sink_0 = uhd.usrp_sink(
27             ",".join(('addr=192.168.10.2', "")),
28             uhd.stream_args(
29                 cpu_format="fc32",
30                 args='peak=0.003906',
31                 channels=list(range(0,1)),
32             ),
33             '',
34         )
35         self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
36         self.uhd_usrp_sink_0.set_center_freq(1e9, 0)
37         self.uhd_usrp_sink_0.set_antenna('TX/RX', 0)
38         self.uhd_usrp_sink_0.set_gain(50e-2, 0)
39         self.rational_resampler_xxx_1_0_0_0_2_0 = filter.rational_resampler_ccc(
40                 interpolation=150,
41                 decimation=1,
42                 taps=[],
```

```
43                  fractional_bw=0)
44          self.blocks_wavfile_source_1_0_0_0_2_0 = blocks.wavfile_source('attsin200-1-0.0.wav',
                False)
45          self.blocks_multiply_const_vxx_0 = blocks.multiply_const_cc(0.01)
46          self.blocks_freqshift_cc_0_2_0 = blocks.rotator_cc(2.0*math.pi*2e5/samp_rate)
47          self.analog_wfm_tx_1_0_0_0_2_0 = analog.wfm_tx(
48                  audio_rate=48000,
49                  quad_rate=48000,
50                  tau=(75e-6),
51                  max_dev=25e3,
52                  fh=(-1.0),
53          )
54
55  # Connections
56          self.connect((self.blocks_freqshift_cc_0_2_0, 0), (self.blocks_multiply_const_vxx_0, 0))
57          self.connect((self.blocks_multiply_const_vxx_0, 0), (self.uhd_usrp_sink_0, 0))
58          self.connect((self.analog_wfm_tx_1_0_0_0_2_0, 0), (self.
                rational_resampler_xxx_1_0_0_0_2_0, 0))
59          self.connect((self.blocks_wavfile_source_1_0_0_0_2_0, 0), (self.
                analog_wfm_tx_1_0_0_0_2_0, 0))
60          self.connect((self.rational_resampler_xxx_1_0_0_0_2_0, 0), (self.
                blocks_freqshift_cc_0_2_0, 0))
61
62  def main(top_block_cls=TX, options=None):
63      tb = top_block_cls()
64
65      def sig_handler(sig=None, frame=None):
66          tb.stop()
67          tb.wait()
68
69          sys.exit(0)
70
71      signal.signal(signal.SIGINT, sig_handler)
72      signal.signal(signal.SIGTERM, sig_handler)
73
74      tb.start()
75
76      try:
77          input('Press Enter to quit: ')
78      except EOFError:
79          pass
80      tb.stop()
81      tb.wait()
82
83  if __name__ == '__main__':
84      main()
```

A.2 編集したプログラム

上記のソースコードを直接編集し，所望する動作を実現したプログラムを以下に示す．

Listing A.2 編集したプログラム

```
1  from gnuradio import analog
2  from gnuradio import blocks
3  import math
4  from gnuradio import filter
```

```
5  from gnuradio.filter import firdes
6  from gnuradio import gr
7  from gnuradio.fft import window
8  import sys
9  import signal
10 from argparse import ArgumentParser
11 from gnuradio.eng_arg import eng_float, intx
12 from gnuradio import eng_notation
13 from gnuradio import uhd
14 import time
15
16 class nogui(gr.top_block):
17
18     def __init__(self,freq):
19         gr.top_block.__init__(self, "Not titled yet", catch_exceptions=True)
20
21 # Variables
22         self.samp_rate = samp_rate = 7142857
23         self.freq = freq
24
25 # Blocks
26         self.uhd_usrp_sink_0 = uhd.usrp_sink(
27             ",".join(('addr=192.168.10.2', "")),
28             uhd.stream_args(
29                 cpu_format="fc32",
30                 args='peak=0.003906',
31                 channels=list(range(0,1)),
32             ),
33             '',
34         )
35         self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
36         self.uhd_usrp_sink_0.set_center_freq(1e9, 0)
37         self.uhd_usrp_sink_0.set_antenna('TX/RX', 0)
38         self.uhd_usrp_sink_0.set_gain(50e-3, 0)
39         self.rational_resampler_1 = filter.rational_resampler_ccc(
40                 interpolation=150,
41                 decimation=1,
42                 taps=[],
43                 fractional_bw=0)
44         self.blocks_multiply_const = blocks.multiply_const_cc(0.01)
45         self.frequency_shift_1 = blocks.rotator_cc(2.0*math.pi*freq/samp_rate)
46         self.wbfm_tx_1 = analog.wfm_tx(
47                 audio_rate=48000,
48                 quad_rate=48000,
49                 tau=(75e-6),
50                 max_dev=25e3,
51                 fh=(-1.0),
52         )
53
54 def main(top_block_cls=nogui, options=None):
55
56     def sig1_start():
57         tb.connect((tb.frequency_shift_1, 0), (tb.blocks_multiply_const, 0))
58         tb.connect((tb.blocks_multiply_const, 0), (tb.uhd_usrp_sink_0, 0))
59         tb.connect((tb.wbfm_tx_1, 0), (tb.rational_resampler_1, 0))
60         tb.connect((tb.wave_1, 0), (tb.wbfm_tx_1, 0))
61         tb.connect((tb.rational_resampler_1, 0), (tb.frequency_shift_1, 0))
62         tb.start()
63         time.sleep(0.5)
64         tb.disconnect((tb.frequency_shift_1, 0), (tb.blocks_multiply_const, 0))
```

```
65          tb.disconnect((tb.blocks_multiply_const, 0), (tb.uhd_usrp_sink_0, 0))
66          tb.disconnect((tb.wbfm_tx_1, 0), (tb.rational_resampler_1, 0))
67          tb.disconnect((tb.wave_1, 0), (tb.wbfm_tx_1, 0))
68          tb.disconnect((tb.rational_resampler_1, 0), (tb.frequency_shift_1, 0))
69          tb.stop()
70          tb.wait()
71
72      tb = top_block_cls(0)
73      samp_rate = tb.samp_rate
74      while True:
75          user_input = input('input >> ')
76          if user_input == 'q':
77              break
78          elif user_input == 'a':
79              tb.wave_1 = blocks.wavfile_source('attsin200-1-0.0.wav', False)
80              freq = -4e5
81              tb.frequency_shift_1 = blocks.rotator_cc(2.0*math.pi*freq/samp_rate)
82          elif user_input == 's':
83              tb.wave_1 = blocks.wavfile_source('attsin200-1-0.0.wav', False)
84              freq = -2e5
85              tb.frequency_shift_1 = blocks.rotator_cc(2.0*math.pi*freq/samp_rate)
86          elif user_input == 'd':
87              tb.wave_1 = blocks.wavfile_source('attsin200-1-0.0.wav', False)
88              freq = 2e5
89              tb.frequency_shift_1 = blocks.rotator_cc(2.0*math.pi*freq/samp_rate)
90          elif user_input == 'f':
91              tb.wave_1 = blocks.wavfile_source('attsin200-1-0.0.wav', False)
92              freq = 4e5
93              tb.frequency_shift_1 = blocks.rotator_cc(2.0*math.pi*freq/samp_rate)
94          elif user_input == 'g':
95              tb.wave_1 = blocks.wavfile_source('attsin200-1-0.0.wav', False)
96              freq = 6e5
97              tb.frequency_shift_1 = blocks.rotator_cc(2.0*math.pi*freq/samp_rate)
98          else:
99              freq = 0
100             tb.frequency_shift_1 = blocks.rotator_cc(2.0*math.pi*freq/samp_rate)
101             print('N/A')
102         sig1_start()
103
104 if __name__ == '__main__':
105     main()
```