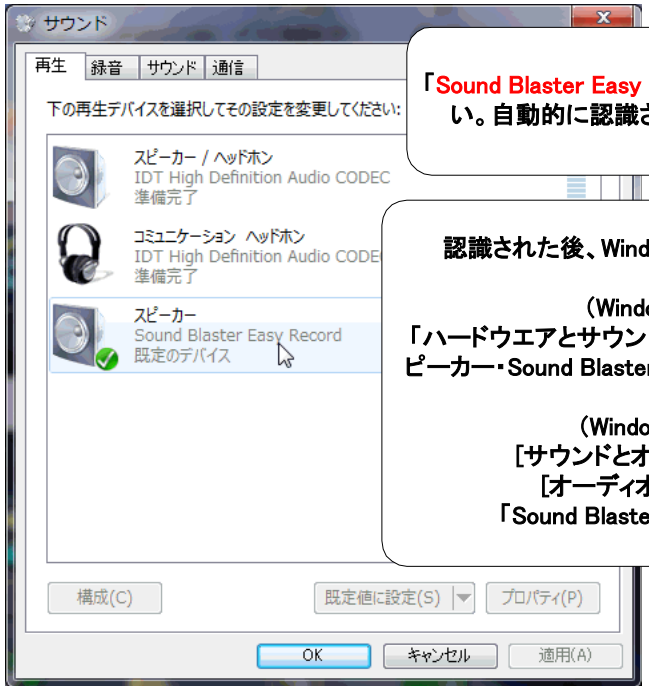
再生デバイスの調整

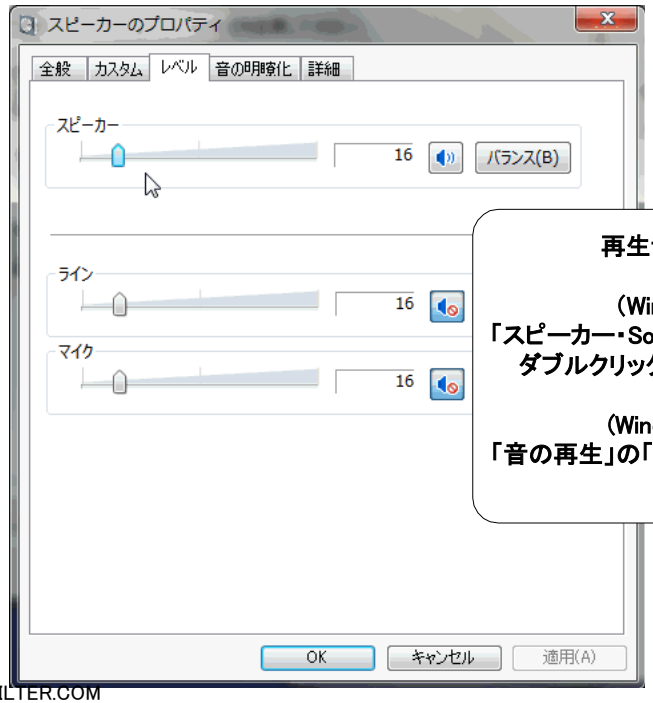


「Sound Blaster Easy Record」をUSBにさして下さい。自動的に認識されます（ドライバ不要）。

認識された後、Windowsのコントロールパネルで

（Windows 7の場合）
「ハードウエアとサウンド」→「サウンド」→「再生」→「スピーカー・Sound Blaster Easy Record」を既定値に設定

（Windows XPの場合）
[サウンドとオーディオデバイス]→
[オーディオ]の「音の再生」で
「Sound Blaster Easy Record」を選択



再生音量を調整します

（Windows 7の場合）
「スピーカー・Sound Blaster Easy Record」をダブルクリックして「レベル」で調整する

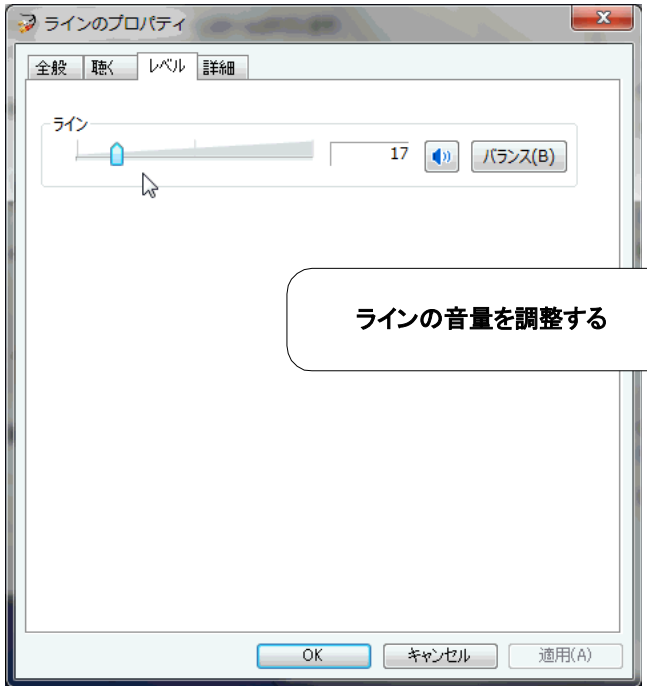
（Windows XPの場合）
「音の再生」の「音量」のスピーカーの音量を少し上げる

録音デバイスの調整

録音音量を調整します

（Windows 7の場合）
「録音」→「ライン・Sound Blaster Easy Record」を既定値に設定し、ダブルクリックでレベル調整

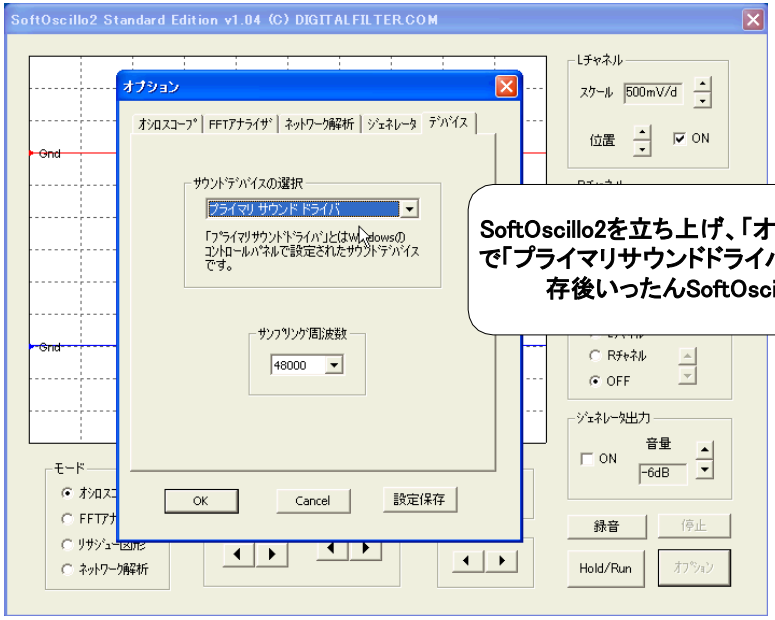
（Windows XPの場合）
「音の録音」で「Sound Blaster Easy Record」を選択し「音量」をクリック



ラインの音量を調整する

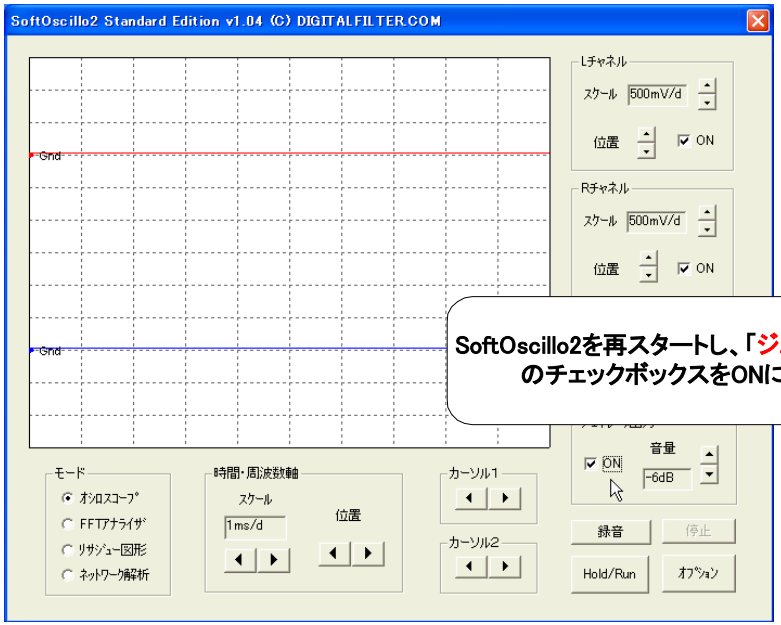
SoftOscillo2 Standard Edition(*)から音を出す

（＊）配布USBメモリに収録。serial no. 0581-3932-3851



SoftOscillo2を立ち上げ、「オプション」→「デバイス」で「プライマリサウンドドライバ」を選択（＊）、設定保存後いったんSoftOscillo2を終了する。

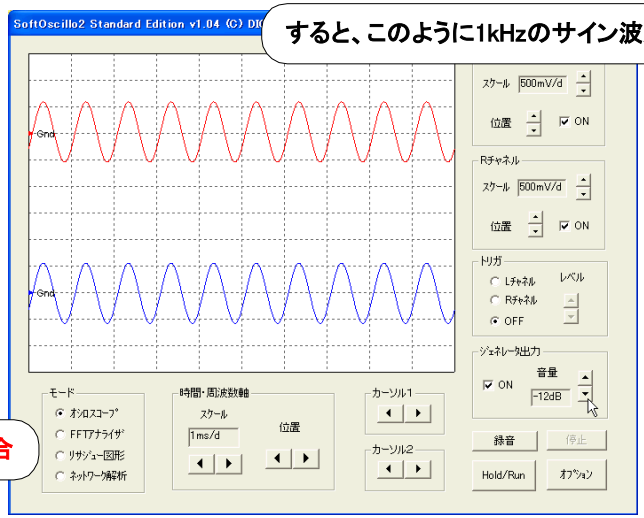
（＊）Sound Blaster Easy Recordを選択しても良い。



SoftOscillo2を再スタートし、「ジェネレータ」のチェックボックスをONにする

USBオーディオの音をUSBオーディオで観察する

このようにスピーカー出力（ジャック）とライン入力（プラグ）をつなぐ。



すると、このように1kHzのサイン波があらわれる

波形が現れない場合

スピーカー出力（ジャック）にヘッドフォンをつないで、音を聞いてみましょう

- 音が出ない → コントロールパネルに戻って音が出るようにする
- 音が出る → SoftOscillo2のオプション（デバイス、ジェネレータ）の確認

書籍18ページをチェック。または付属CD-ROMのSoftOscillo2¥TroubleShoot¥Trouble.html

書籍11ページ

MPLAB IDEとC30のセットアップ

dsPICマイコン統合開発環境

Cコンパイラ

MPLAB IDEのダウンロード

www.microchip.com からダウンロード

頒布USBメモリに MPLAB_IDE_8_73a.zip

MPLABXではビルドエラーが出ます。
さしあたりMPLABを使用してください。

MPLAB IDEのインストール

特に気を付けることはありません。ZIPを解凍した後、普通にインストールしましょう。

最後に「High Tech C」のインストーラが現れるが、これはキャンセルしてよい。

C30のダウンロード

www.microchip.com からダウンロード （英語サイトがベター）

頒布USBメモリにmplabc30-v3.30b-windows-installer.exe

最初はダウンロード不可になっている。Registerの後、Sign Inするとダウンロード可。

C30のインストール

MPLABの後にインストールします。

インストールディレクトリ選択はどちらでもよいがLegacyで

"Lite Compiler"を選択する。（"最適化"などのオプションが使えなくなるだけ）。

MPLABでCコーディング

書籍12ページ　MovingAverageディレクトリのコピー　付属CD-ROMのMPLAB¥Sourceにある。　HDD上の適当なディレクトリにコピーする

書籍12ページ　Project Wizartdでプロジェクト作成　デバイス(dsPIC30F2012)を選択 コンパイラ(Microchip C30 C Compiler)を選択　ソース(MovingAverage.c)をAdd ライブラリ(libdsp-coff.a)をAdd(*)

(*) C:¥Program Files¥Microchip¥mplabc30¥v3.31¥libなどにある

書籍14ページ　プロジェクトのビルド　Build Allアイコン(またはメニューのProject→Build All)　Build Suceededが出たら成功。

⇩

MovingAverage.hexが出来ている

dsPIC書き込みには新ソフトdspicguy64を使います

書籍で使っている"dspicguy.exe"はWindows Vista/7の64ビット版OSでは動きません
"dspicguy64.exe"は64ビットでも32ビットでも動きます

書籍14ページ　書き込みの準備　ﾊﾟｿｺﾝとﾄﾚｰﾆﾝｸﾞ基板をｼﾘｱﾙｹｰﾌﾞﾙで繋ぐ JP1のジャンパは2と3に JP2はPWMに配置する。　付属dsPIC基板のｽﾗｲﾄﾞｽｲｯﾁをLDにして 子基板のﾌﾟｯｼｭｽｲｯﾁSW1を押す。(＊) 付属dsPIC基板の赤・緑LEDともに点灯するのを確認

(*)ﾄﾚｰﾆﾝｸﾞ基板にもSW1があるので注意

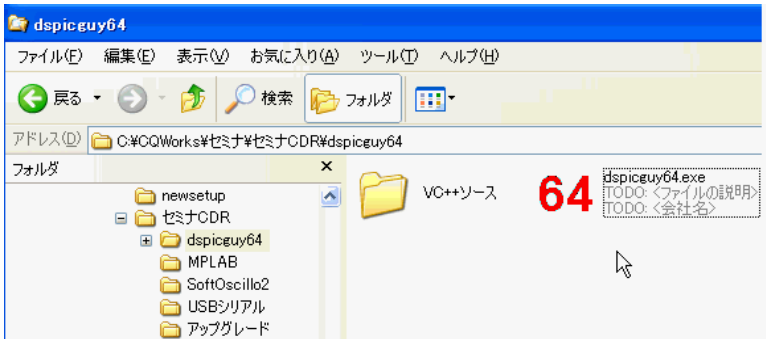
⇩

準備が整ったらdspicguy64.exeを実行

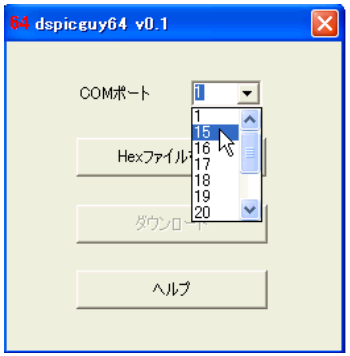
新ソフトは配布したUSBメモリのセミナCDR¥dspicguy64ディレクトリ
または　http://digitalfilter.comからダウンロード

新ダウンロードアプリケーション　dspicguy64.exeを使う

①　配布したUSBメモリのセミナCDR¥dspicguy64ディレクトリのdspicguy64.exeを実行する



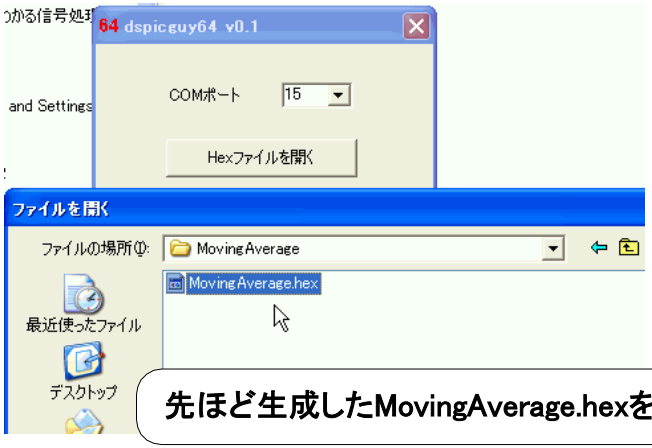
②　COMポートを選択する



COMの番号はデバイスマネージャで調べる
「プログラムとファイルの検索」を選択し、"sysdm.cpl"と入力すると「システムのプロパティ」が現れる。「ハードウエア」タブの「デバイスマネージャ」をクリックし、ポート（COMとLPT）からそれらしきものを見つける。

XPの場合はの「スタート」から「ファイル名を指定して実行」でsysdm.cpl

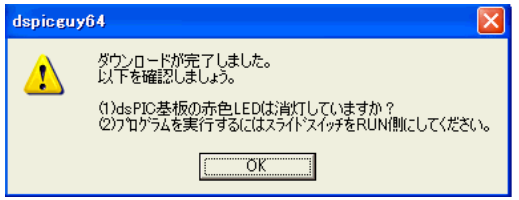
③　HEXファイルを開く



先ほど生成したMovingAverage.hexを選択

④　ダウンロードボタンをクリック

⑤　以下の2つを確認



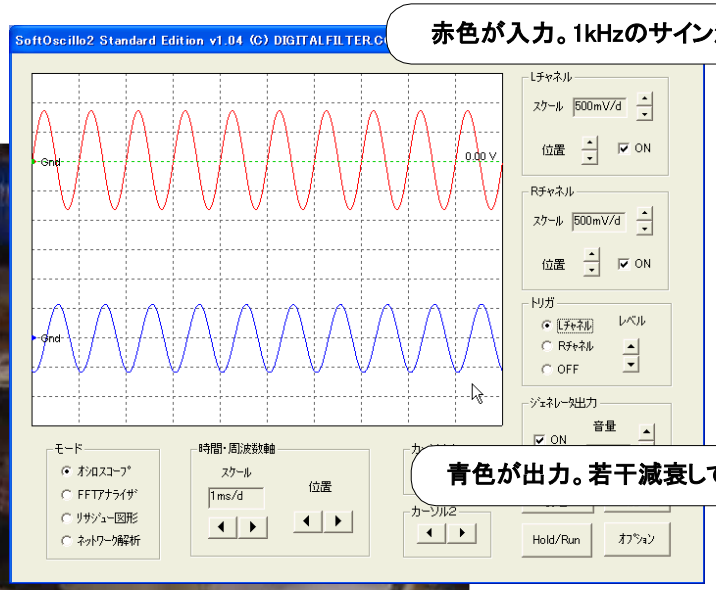
赤は消灯する。**緑**は点滅のち点灯。
その後このダイアログが現れるので
スライドSWを**RUN**側にする

うまく書き込めないときは・・・

COMポートは正しいか
JP1のジャンパは2と3に配置されているか

トレーニングボードの入出力をUSBオーディオで観察する

MovingAverageプロジェクトを焼いた時の入出力



赤色が入力。1kHzのサイン波。

青色が出力。若干減衰している。

先ほどのMovingAverageプロジェクトが書き込まれている。

ライン入力をオシロとして使う

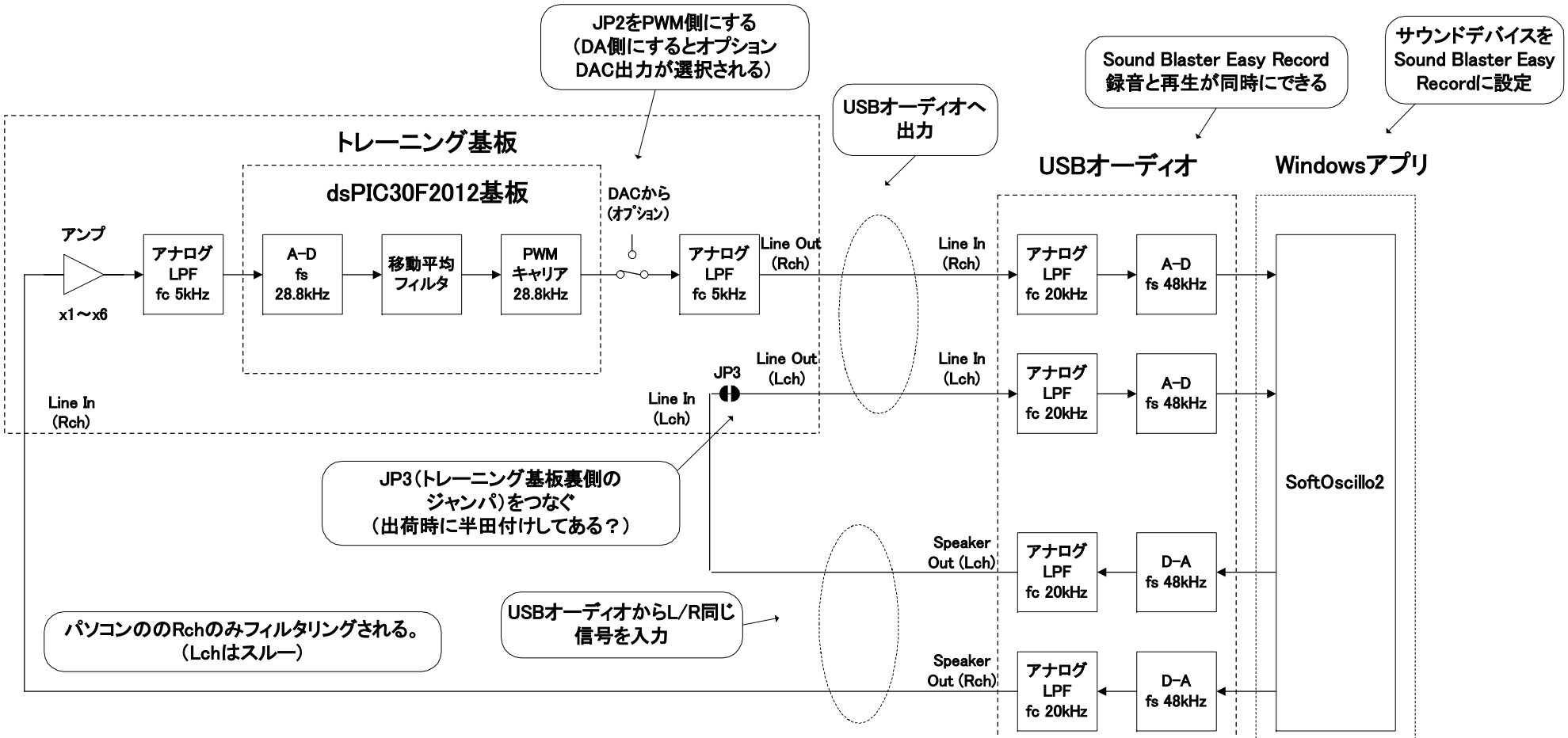
スピーカー出力をジェネレータとして使う

書籍17ページ～

基板とSoftOscillo2でやっていること

うまく波形が表示されないときは→

書籍18ページを確認してください。
また、付属CD-ROMにSoftOscillo2のトラブルシューティングがあります。



fs：サンプリング周波数
fc：カットオフ周波数

信号の流れ

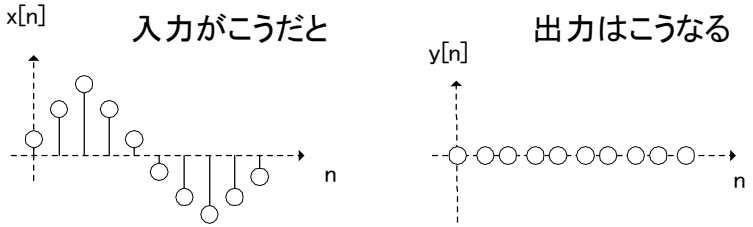
書籍21ページ～

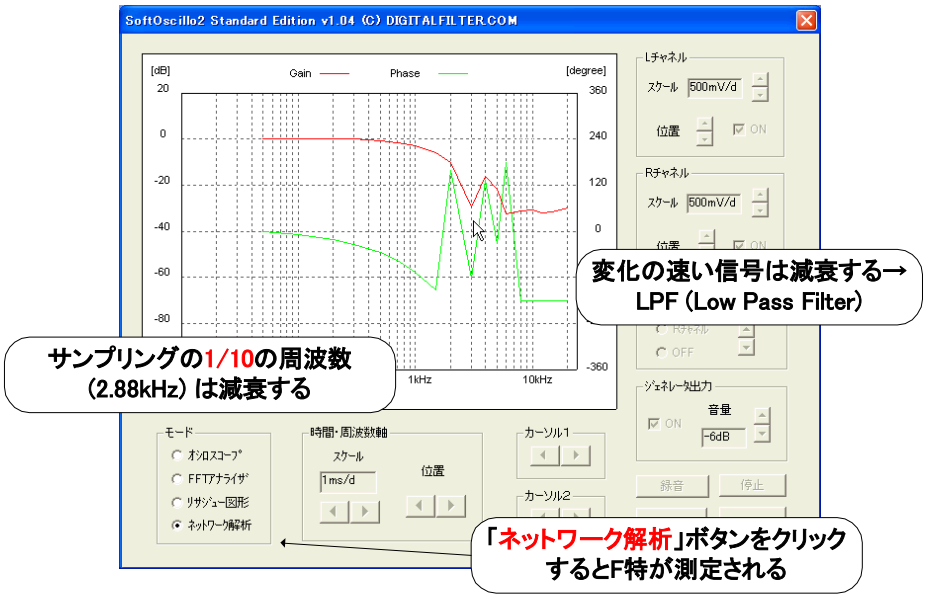
近傍のデータ10個の平均をとる

x[n] : 現在のデータ
x[n-1] : 1サンプル前のデータ
x[n-2] : 2サンプル前のデータ
：　　：
x[n-9] : 9サンプル前のデータ

書籍図1-7の意味は・・・

これらを平均して足し込む





変化の速い信号は減衰する→
LPF (Low Pass Filter)

サンプリングの1/10の周波数
(2.88kHz) は減衰する

「ネットワーク解析」ボタンをクリック
するとF特が測定される

これらDSP関数の中身はアセンブラなので高速に処理できる

係数0.1を代入
（書籍リスト1-3）

プログラム開始時に一度だけ実行

遅延器の配列のプッシュ
VectorCopy関数使用
（書籍リスト1-4）

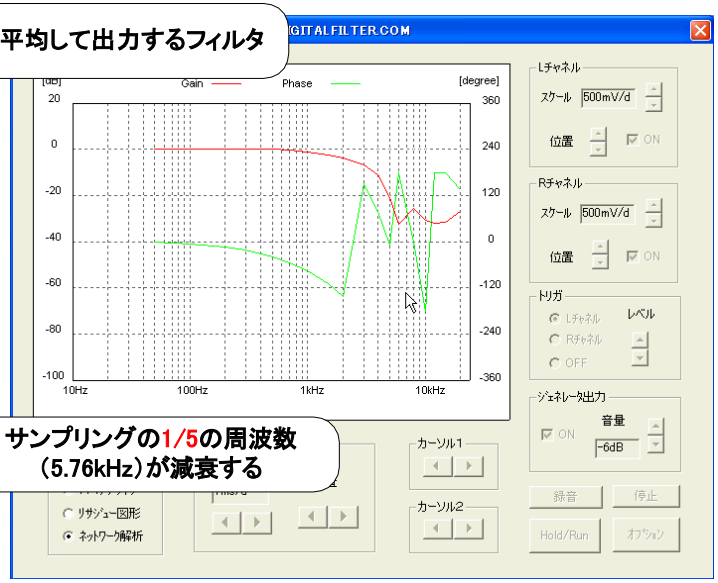
遅延器と係数の積和演算
VectorDotProduct関数使用
（書籍リスト1-6）

タイマ3割込
（A-D変換）毎に実行
（書籍リスト1-1）

```
void CoeffInit() { // 移動平均フィルタ（10タップ）の係数を代入
  int i;
// +1.0 -> 32767(0x7FFF), -1.0 -> -32768(0x8000)
// したがって +0.1 -> 3276 となる。
      for(i = 0; i < 5; i++) Coeff[i] = 6552;
}
```

リスト1-3をこのように変えると・・・

5個平均して出力するフィルタ



サンプリングの1/5の周波数
(5.76kHz)が減衰する

F得を測る手段は2通りある

F特（周波数特性）を測りたい！

ネットワーク解析

いろいろな周波数の
サイン波を入力

⇩

各周波数の出力／入力
を測定してプロット

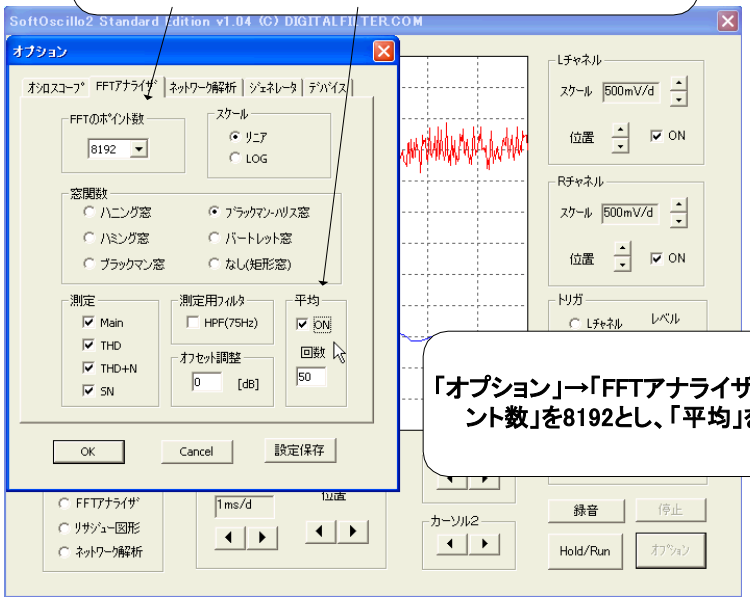
白色雑音をFFT

白色雑音を入力

⇩

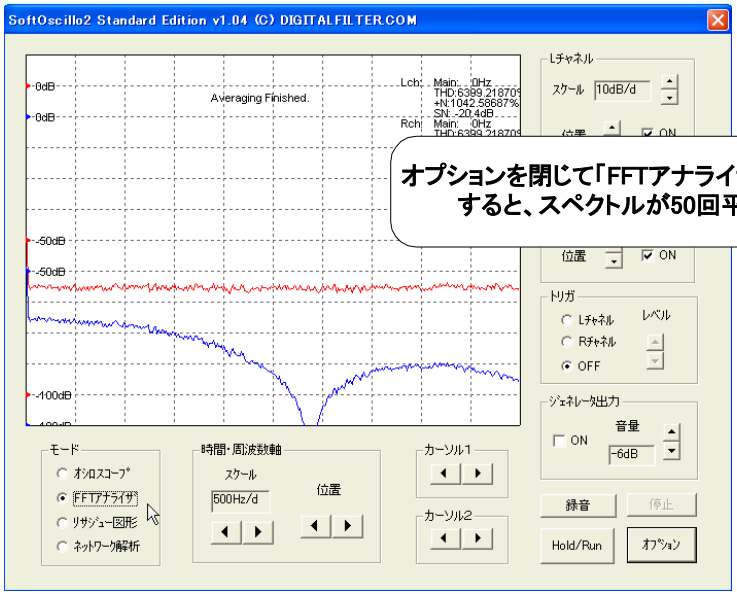
出力をFFTする

Standard Editionの機能（CQ Editionにはありません）

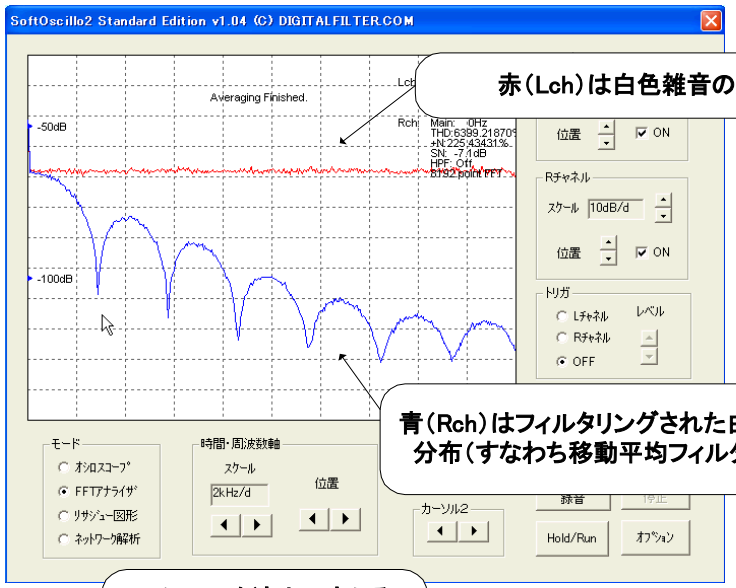


「オプション」→「FFTアナライザ」の「FFTのポイント数」を8192とし、「平均」をONにする。

白色雑音をFFTしてみる



オプションを閉じて「FFTアナライザ」ボタンを選択すると、スペクトルが50回平均される。



赤（Lch）は白色雑音の周波数分布

（＊）白色雑音は周波数によらず均一なスペクトルを持つ

青（Rch）はフィルタリングされた白色雑音の周波数分布（すなわち移動平均フィルタの周波数特性）

スケールを適当に変える

書籍26ページ～

係数を変えるだけであらゆる特性を実現できる

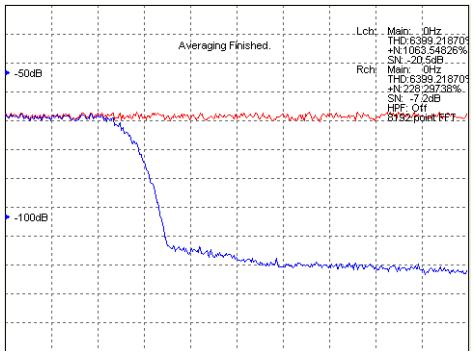
StaticAdaptiveプロジェクトのソース

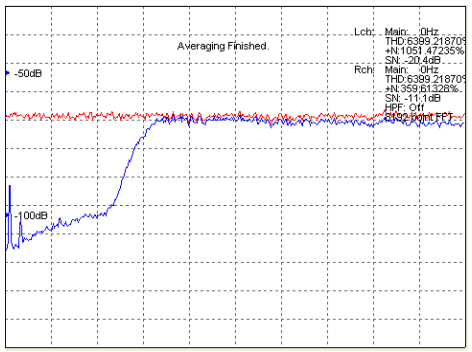
```
void CoeffInitLpf() { // FIR型LPFの係数を代入
(1kHz以下を通すフィルタ)
//Generated by DSPLinks
//U_4
//Remez Algorithm LPF
//Sampling Frequency =    28800.0
//cutoff1 = 1100.0000000
//cutoff2 = 1900.0000000
//Tap Count = 127
//attenuate =   -80.00
//ripple factor =    0.100000
//Quantized by 16 [bits]
Coeff[   0      ] =      -3      ;
Coeff[   1      ] =      -4      ;
Coeff[   2      ] =      -6      ;
Coeff[   3      ] =      -8      ;
Coeff[   4      ] =      -11     ;
Coeff[   5      ] =      -14     ;
Coeff[   6      ] =      -15     ;
Coeff[   7      ] =      -16     ;
Coeff[   8      ] =      -15     ;
Coeff[   9      ] =      -12     ;
Coeff[   10     ] =      -6      ;
Coeff[   11     ] =      1       ;
Coeff[   12     ] =      11      ;
Coeff[   13     ] =      22      ;
```

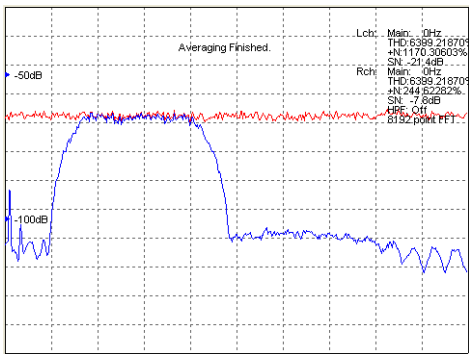
```
void CoeffInitHpf() { // FIR型HPFの係数
を代入(2kHz以上を通すフィルタ)
//Generated by DSPLinks
//U_11
//Remez Algorithm HPF
//Sampling Frequency =    28800.0
//cutoff1 = 1100.0000000
//cutoff2 = 1900.0000000
//Tap Count = 127
//attenuate =   -80.00
//ripple factor =    0.100000
//Quantized by 16 [bits]
Coeff[   0      ] =      52      ;
Coeff[   1      ] =      -12     ;
Coeff[   2      ] =      -14     ;
Coeff[   3      ] =      -17     ;
Coeff[   4      ] =      -19     ;
Coeff[   5      ] =      -21     ;
Coeff[   6      ] =      -20     ;
Coeff[   7      ] =      -16     ;
Coeff[   8      ] =      -9      ;
Coeff[   9      ] =      0       ;
Coeff[   10     ] =      12      ;
Coeff[   11     ] =      24      ;
Coeff[   12     ] =      36      ;
Coeff[   13     ] =      44      ;
```

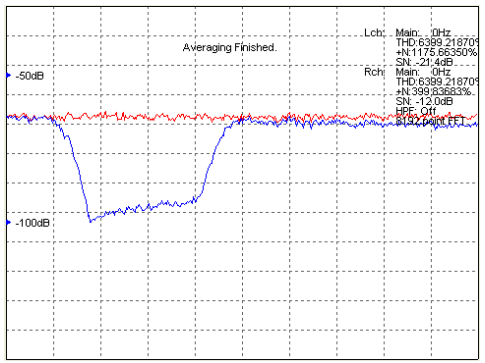
```
void CoeffInitBpf() { // FIR型BPFの係数を代入
(1kHz～2kHzを通すフィルタ)
//Generated by DSPLinks
//U_12
//Remez Algorithm BPF
//Sampling Frequency =    28800.0
//cutoff1 =  550.0000000
//cutoff2 =  950.0000000
//cutoff3 = 2100.0000000
//cutoff4 = 2500.0000000
//Tap Count = 127
//attenuate =   -55.00
//ripple factor =    0.300000
//Quantized by 16 [bits]
Coeff[   0      ] =      48      ;
Coeff[   1      ] =      -56     ;
Coeff[   2      ] =      -10     ;
Coeff[   3      ] =      26      ;
Coeff[   4      ] =      61      ;
Coeff[   5      ] =      95      ;
Coeff[   6      ] =      123     ;
Coeff[   7      ] =      140     ;
Coeff[   8      ] =      140     ;
Coeff[   9      ] =      117     ;
Coeff[   10     ] =      69      ;
Coeff[   11     ] =      1       ;
Coeff[   12     ] =      -77     ;
Coeff[   13     ] =      -156    ;
```

```
void CoeffInitBrf() { // FIR型BRFの係数を
代入(1kHz～2kHzをカットするフィルタ)
//Generated by DSPLinks
//U_13
//Remez Algorithm BRF
//Sampling Frequency =    28800.0
//cutoff1 =  550.0000000
//cutoff2 =  950.0000000
//cutoff3 = 2100.0000000
//cutoff4 = 2500.0000000
//Tap Count = 127
//attenuate =   -55.00
//ripple factor =    0.300000
//Quantized by 16 [bits]
Coeff[   0      ] =      -25     ;
Coeff[   1      ] =      326     ;
Coeff[   2      ] =      -512    ;
Coeff[   3      ] =      -7      ;
Coeff[   4      ] =      242     ;
Coeff[   5      ] =      277     ;
Coeff[   6      ] =      216     ;
Coeff[   7      ] =      132     ;
Coeff[   8      ] =      59      ;
Coeff[   9      ] =      9       ;
Coeff[   10     ] =      -11     ;
Coeff[   11     ] =      -4      ;
Coeff[   12     ] =      28      ;
Coeff[   13     ] =      80      ;
```









StaticAdaptiveプロジェクトの周波数特性(青色)

書籍55ページ～

ブラックボックスを使ってみる（FIR関数）

FIR関数とは・・・

①遅延素子の配列のプッシュ
②各遅延素子と係数の乗算
③乗算結果の総和

これらをまとめた関数

（書籍図2-2参照）

FIR関数を使う手順

①係数を決める ← 16ビットで量子化。FirFunc.cでは係数をプログラムメモリに割り当てている（リスト2-1）。(*)

(*) データーメモリ節約のため。

②遅延素子を用意する ← タップ数の2倍以上の領域をY-Memoryに（リスト2-1）。

③構造体を作る ← タップ数、係数や遅延素子のアドレスなどを定義する（リスト2-2）。

④遅延素子の初期化 ← 遅延素子の配列を0で埋める（リスト2-2）

④A-Dの割り込みを待つ ← A-Dから入力が1個来るたびにFIR関数を実行し、出力を1個計算する（リスト2-3）

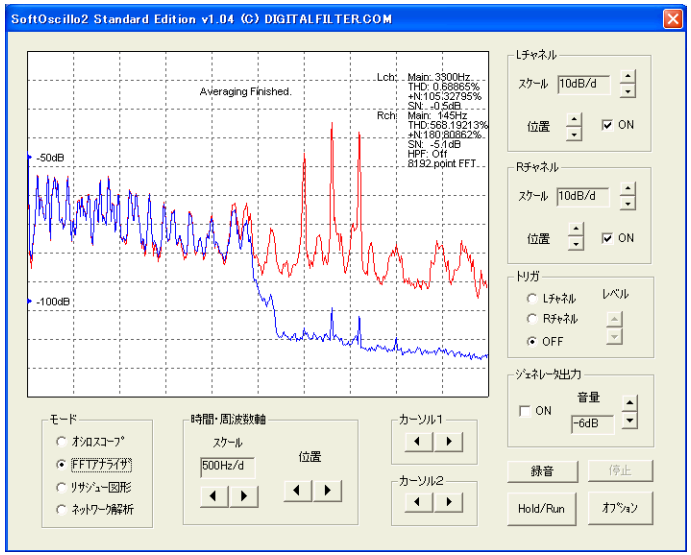
FIR関数の引き数

```
FIR( 1, &filterOutData, &filterInData, &firFilter );
```

- 出力を1個計算
- 入出力はポインタ（アドレス）で指定
- 構造体もポインタ（アドレス）で指定

●音楽に重畳したノイズを消してみよう

FirFuncプロジェクト（＊）でclassic_a_hinz.wavを再生



127次FIRフィルタでは、高域ノイズが大幅に減衰している

（＊）最新のC30コンパイラ(v3.31)にはバグがある模様。
このプロジェクトはうまく動作しません

頒布USBメモリにあるv3.30bを使ってください。また
microchip.comのアーカイブにv3.30cがあります。

書籍62ページ～

DSPLinksで得た係数をIIRTransposed関数で使う

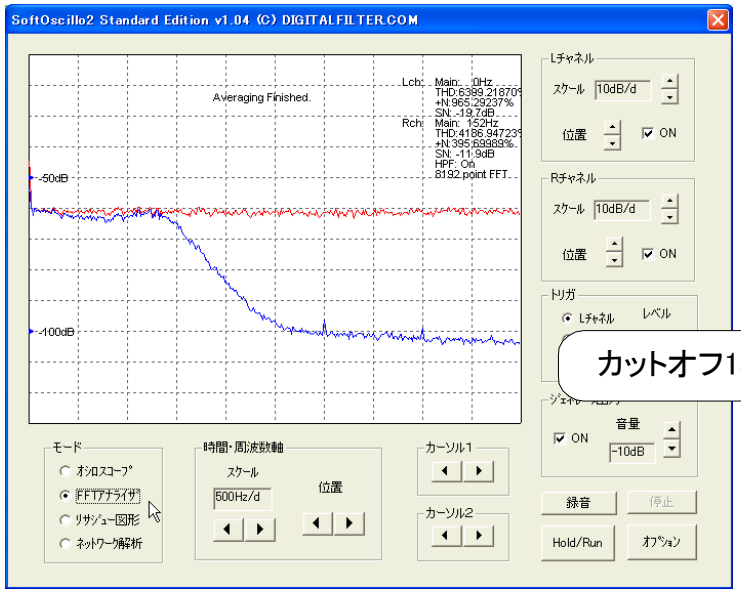
Biquad（2次IIRフィルタ）では深い減衰量が取れない
（書籍図2-7、8参照）

⇩

IIRTransposed関数で
Biquadの多段縦続接続を一気に計算（書籍図2-6参照）

⇩

（FIRと比較して）少ない演算の割には急峻な減衰量が得られる

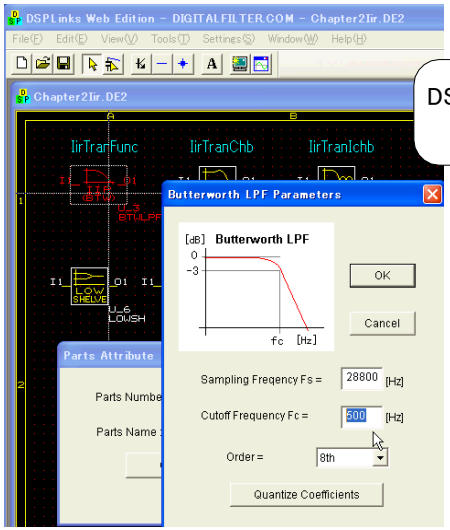


カットオフ1500Hz

IirTranFuncプロジェクトの周波数特性

●DSPLinksを立ち上げてカットオフ周波数を変える

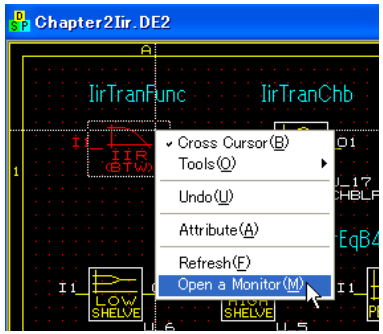
（書籍155ページ参考）



DSPLinksでChapter2Iir.DE2（＊）を開いてIirTranFuncをダブルクリック

（＊）CD-ROMのDSPLinks¥Schematic以下をHDDにコピーしてから開く

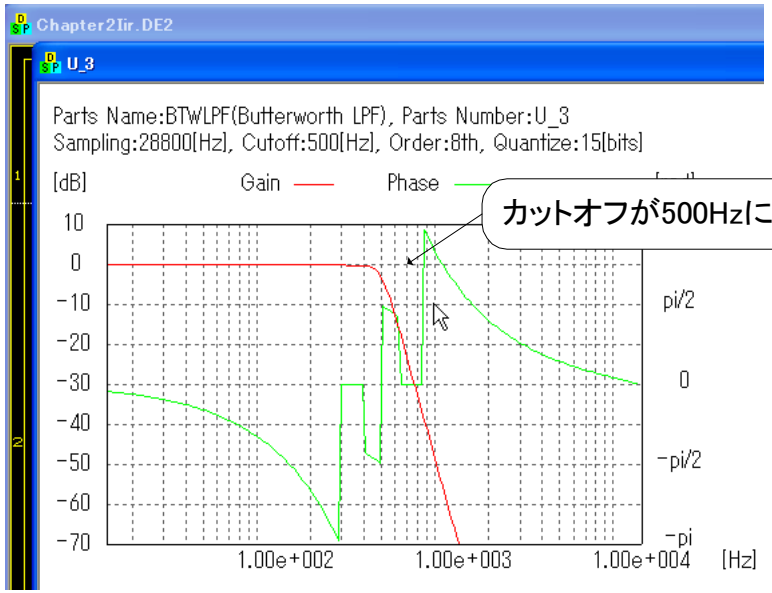
Change Parametersでカットオフを1500Hzから500Hzに変更



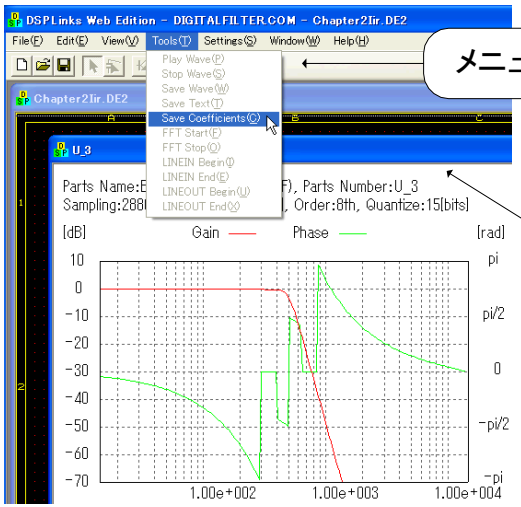
右クリック→Open a monitorで特性を見る。

書籍155ページ～

IIRフィルタの係数を変えてみよう



カットオフが500Hzになっている



メニューから係数をセーブする

Monitorウインドウを最前面に出す必要あり

```
U_3
Butterworth LPF
Sampling Frequency =   28800.0
cutoff1 =      500.0
Order = 8th
Quantized by 15 [bits]
**** 1st buquad ****
a0 =           47
a1 =           95
a2 =           47
b1 =       -31895
b2 =        15702
**** 2nd buquad ****
  :            :
```

15ビットで量子化している。（16ではない）

この2係数の符号に注意！

DSPLinksでセーブした係数ファイル(coeff1.txt)

●書籍リスト8-4（156頁）を参考にして IirTranFuncプロジェクトの係数を変更してみましょう

その後カットオフが1500Hzから500Hzに変わっていることを確認

IIRフィルタは以下の2点に注意

15ビットで量子化する理由 ⇒ IIRフィルタは係数の値が1.0を超える場合がある

書籍リスト8-2（152ページ）参照。

（＊） インターフェース誌2009年1月号特集・第9の壁もご参照

フィードバック側の係数の符号 ⇒ DSPLinksで得られた係数を使う場合はフィードバック側の係数を反転させる。

書籍リスト8-1（150ページ）参照。

（＊） インターフェース誌2009年1月号特集・第2の壁もご参照

書籍81ページ～

FFTアナライザを作ろう

音楽に載ったノイズがが何Hzかわからない

⇩

FFTアナライザでスペクトルを見れば分かる

書籍写真3-1はFftAnalyzerプロジェクトを実行し、
classic_a_1k.wavを再生するようす（1kHzのノイズあり）

●FftAnalyzerプロジェクトを作成しよう

FftAnalyzer以下のCファイル、
Hファイルをすべて ADDする

(*) このデバイス(dsPIC30F2012)においてFFTは64ポイントが限界のようです。128ポイントではメモリサイズオーバになります。

●SoftOscillo2のディレクトリにいろいろなWAVEファイルがあるので、再生してスペクトルを見てみましょう。

クイズ① 1kHzの矩形波のスペクトルはどうなるでしょうか？

（ヒント） sq1k.wavを再生してみましょう。

（ヒント） トレーニング基板のSW1を押さないと0Hz～3750Hz、押すと4000Hz～7750Hzが表示されます。

（＊）LCDの表示がおかしい時はジェネレータのボリュームを調整してみましょう

クイズ② 1kHzの三角波のスペクトルはどうなるでしょうか？

（ヒント） tri1k.wavを再生してみましょう。

クイズ③ 1kHzののこぎり波のスペクトルはどうなるでしょうか？

（ヒント） saw1k.wavを再生してみましょう。

クイズ③ 白色雑音のスペクトルはどうなるでしょうか？

（ヒント） whitenoise.wavを再生してみましょう。

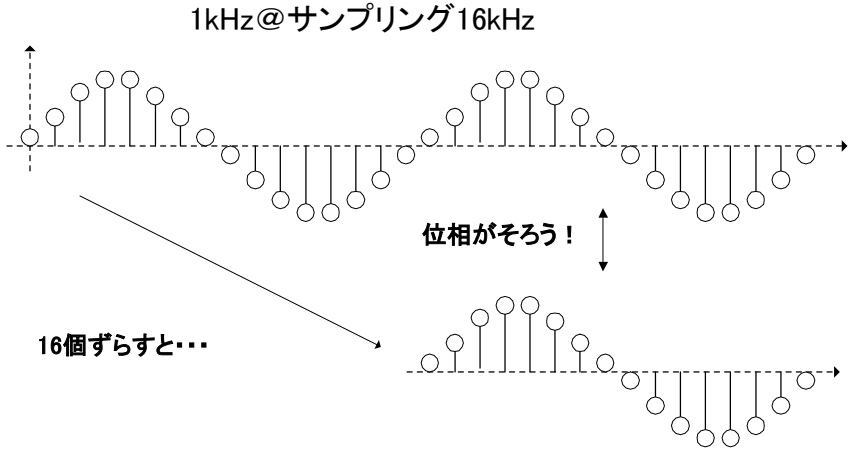
書籍90ページ～

自己相関を利用して周期信号を除去

●VdotCorrGraphプロジェクトをdsPICに書き込みましょう

●sin1k.wav, sin2k.wav, sin4k.wav, whiteniose.wavを入力し、写真G-1（書籍95ページ）の様になるか確認。

なぜそうなるのか？ ⇩



●位相がそろったところで自己相関は極大になる。
（その理由は書籍103ページ）

周期信号（サイン波など）→位相がそろう場合あり→自己相関が大きくなる
非周期信号（白色雑音など）→位相がそろわない→自己相関は小さい

⇩

自己相関により周期信号のある／なしを判別できる

（自己相関はFFTと比べて演算量が少ない）

●AutoCorrプロジェクトをdsPICに書き込みましょう

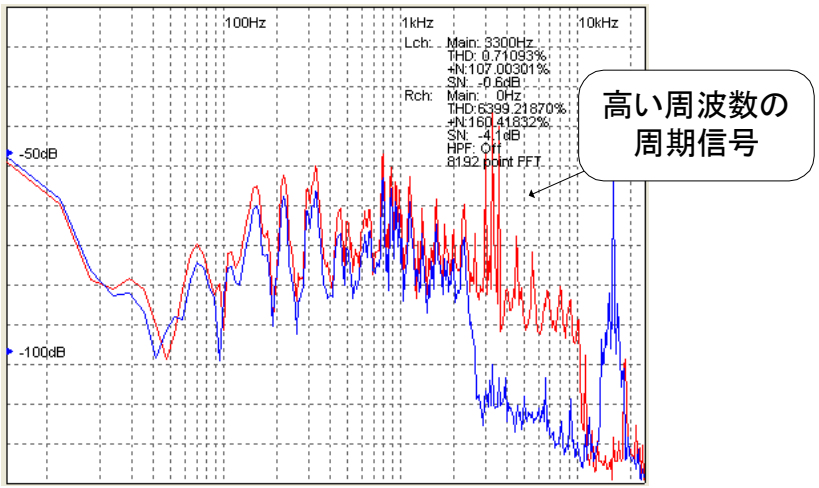
（＊）MPLAB+C30のバージョンアップにより、書籍CD-ROMのソースでは動かなくなりました。頒布USBメモリの「アップグレード」にあるAutoCorr.cを使ってください。

●classic_a_hinz.wavを入力し、写真4-1（書籍108ページ）の様になるか確認。

classic_a.wavを再生　音楽だけ→自己相関小さい→スルー（緑色LED）

⇩ ⇧ ←　2つのWAVEファイルを交互に再生してみましょう

classic_a_hinz.wavを再生　音楽＋周期信号→自己相関大→LPF（赤色LED）



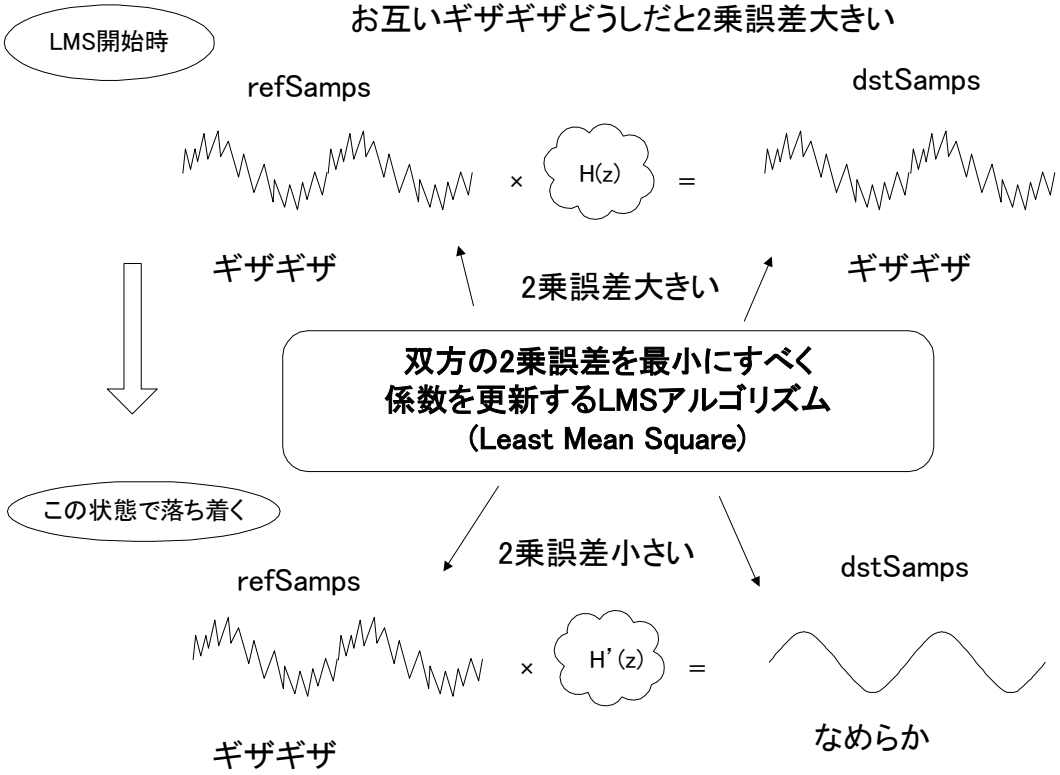
LPFに切り替わったときのスペクトル分布（青が出力）

書籍109ページ～

FIRLMS関数による白色雑音の除去

●今度は相関のない信号を取り去ります

書籍図4－6のrefSampsとdstSampsに注目



●FIRLMS関数でLMSアルゴリズムを実装しよう

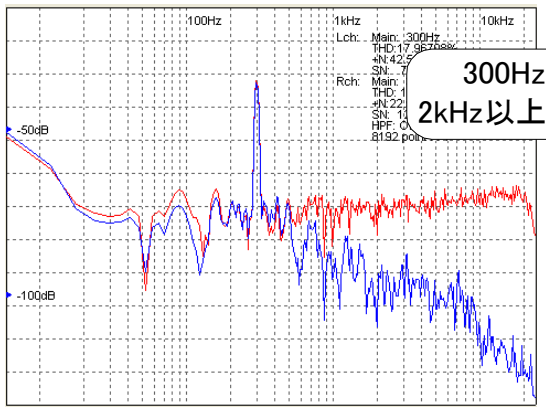
（＊）係数は逐次更新されるので、データメモリ（X-Memory）におかれる

●LmsFuncプロジェクトをdsPICに書き込む

●sin1kwn.wav, sin3kwn.wav, sin300wn.wav, sq1kwn.wavを再生し、図4-7～4-10の様になるか確認。

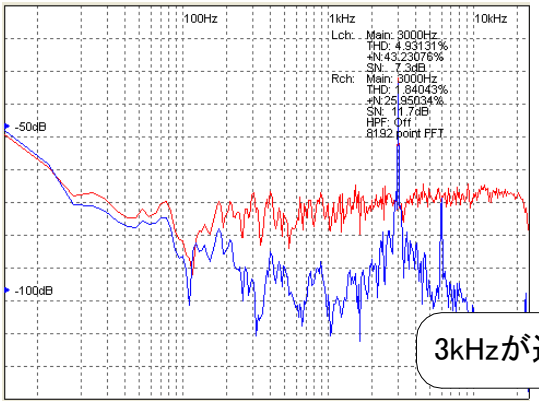
（これらwaveファイルは付録CD-ROMのSoftOscillo2¥Soundディレクトリにあります）

●スペクトルも見てみましょう



sin300wn.wav再生時　300Hz＋白色雑音

**FIRLMS関数で
フィルタ係数を
自動切り替え**



sin3kwn.wav再生時　3kHz＋白色雑音

●LMSによって「スペクトル強調器」が実現した！

その他お役立ち情報

●パソコンとシリアル通信するには

書籍121ページ～

第5章「マルチレート信号処理」では、パソコンからのコマンドを受ける→パソコンへデータをアップロードする、といった双方向のシリアル通信をしています（InterpFuncプロジェクトなど）。
また、Windows側アプリのソース（InterpMonなど、Visual C++ 2008プロジェクト）も付属CD-ROMに収録しています。

（＊）MPLAB+C30のバージョンアップにより、書籍CD-ROMのソースでは動かなくなりました。頒布USBメモリの「アップグレード」にあるInterpFunc.cを使ってください。

（＊）なぜか赤色LEDが点灯したままですが、問題なく通信できます。

●サンプリング周波数を変更するには

書籍103ページ～

AutoCorrプロジェクトではA-Dのサンプリング、PWMのキャリアを16kHzにしています。他のプロジェクト（MovingAverageなど）は28.8kHzなので、それらとの違いを確認しましょう。

●その他ご不明な点は・・・

筆者Webサイト http://digitalfilter.com に書籍のサポートサイトがあります。掲示板もありますのでぜひご利用下さい。

●さらなるステップアップに・・・

D-Aコンバータ内蔵の高性能なdsPIC33FJ128GP804を搭載した基板！

dsPIC30F2012と比べると桁外れにパワフル！

	dsPIC33FJ128GP804	dsPIC30F2012
プログラムメモリ	128 kByte	12 kByte
データRAM	16 kByte	1 kByte
ピン数（内I/O）	44 (35)	28 (20)
A/Dコンバータ	12bit @500kHz	12bit @200kHz
D/Aコンバータ	16bit @100kHz	なし
DMA チャネル数	8	なし
電源電圧範囲	3.0V～3.6V	2.5V～5.5V

http://digitalfilter.comでお買い求めいただけます(¥22,200)。