オーディオ信号処理で学ぶDSP

第6回　エフェクタを作ってみる――あふれと飽和

堀江 誠一

今回はADSP-BF537に内蔵されているI²Cインターフェースを使って，外部周辺機器を接続する．例として，A-Dコンバータ AD7998を接続し，割り込みを使用したI²Cの制御と，オーディオ・フレームワークの拡張を行う．（編集部）

オーディオの分野では，いかに良い音を人に伝えるかが重要です．つまりひずみのないことが要求されます．実際，オーディオ用のCODECの*S*/*N*比は100dBを軽く超えるものが一般的です．EZ-KIT BF537にもオーディオ用のD-Aコンバータ（DAC）であるAD1854が搭載されており（**写真1**），その*S*/*N*比は112dBもあります．これほど性能が上がっても，CODECごとに音の違いが聞き分けられるそうです．ただし，いずれの音が良いかは好き好きではないかという意見もあります．

そこまで性能を追求しなくても，一般論としてひずみは嫌なものです．ところが，固定小数点DSPの場合，信号データの表現範囲が±1の範囲に限られるため，どうしてもデータのあふれがおきることを意識せざるをえません．データのあふれはひずみに直結します．そこで，今回はデータのあふれが生じたときに行う飽和処理（クリッピング）について説明します．さらにその応用としてギター・エフェクタを作ってみます[注1]．

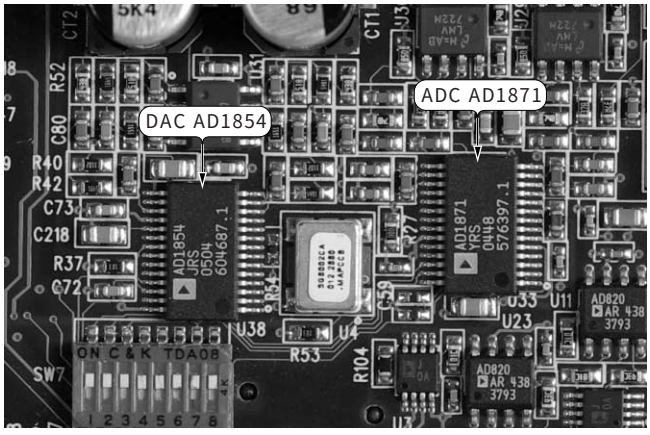


写真1　EZ-KIT Lite BF537のADCとDAC

注1：サンプル・プログラムは，CQ出版社のWebサイトからダウンロードできる（http://www.cqpub.co.jp/interface/download/）．

1．あふれとは何か

● 目で見る演算結果のあふれ

演算結果が値で表現できる範囲を超えてしまうと，あふれが生じます．ではあふれが起きるとどうなるのか，簡単な例を見てみましょう．**リスト1**は正弦波を生成するプログラムで，オーディオ・フレームワークを使って音として信号を出力します．このプログラムで使っている正弦波生成は単純なテーブル参照です．テーブルに格納してある値はScilabで作成しました．Excelのような表計算ソフトウェアでも作成できます．波形を確認してみると，きれいな正弦波になっています（**図1**）．

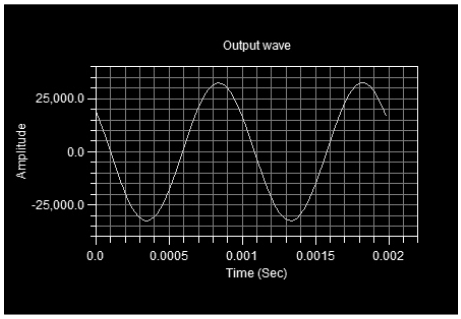
リスト1　正弦波の生成

```
shortfract osc(void);

void processData(
                const shortfract leftIn[],
                const shortfract rightIn[],
                shortfract leftOut[],
                shortfract rightOut[],
                int count
               )
{

    // 引き数配列のすべてのデータを処理する
    for ( int i=0; i<count; i++ ){
          // 正弦波を出力する
        rightOut[i] = leftOut[i] = osc();
    }

}   // processData

shortfract osc(void)
{
    shortfract result;
    // static でなければならない
    static int index = 0;

    // 正弦波テーブルからサンプル取り出し
    result = table[index++];
    // テーブルの端まで来たら端から繰り返し
    if ( index >= tableSize )
        index = 0;

    return result;
}
```

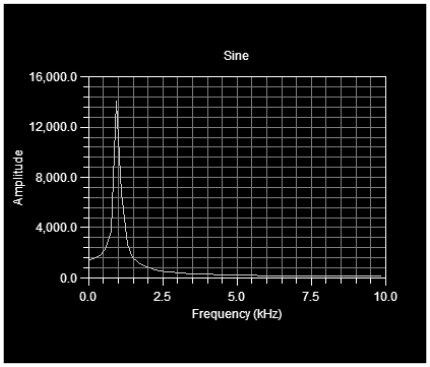
（フレームワークの関数を使う）

（ループの中を変更）

（正弦波発振器関数を追加．1kHz専用）



(**a**) 波形



(**b**) スペクトラム

図1
正弦波出力

さて，このルーチンで生成した正弦波データを使ってあふれの実験をしてみます．**リスト2**にそのコードを示します．元のプログラムは16ビットの範囲内でいっぱいに振れる正弦波を生成していました．それをこのプログラムは1.2倍しています．1.2倍したことによって，16ビットの変数に格納するとあふれが生じます．

あふれが生じた結果を**図2**に示します．もとの波形がどうだったか，ぱっと見たくらいでは分からないほど形が変わっています．ひずんでいるというよりは，壊れていると言いたくなるほどです．実際にこの波形を聞いてみても，とても元の波形が正弦波であったとは思えません．

● なぜあふれで波形が大きく崩れるのか

このような激しい結果になるのは，2の補数表現ではあふれが生じたときに非常に大きな値の変化がおきるからです．**図3**を見ながらその様子を見てみましょう．値を1LSBずつ増やしていくと，2の補数表示ではあふれがおきた瞬間，表現できる最大値から最小値へと値が急変します．その結果，16ビットの符号付き整数では，それまでの1LSBずつの変化が，一気に－65535LSBもの変化になります．雑音が大きいのも当然です．

このあふれ方[注2]は，2の補数を使う汎用プロセッサでは普通のものです．つまり，x86系パソコン用プロセッサや，

リスト2　ラップ・アラウンド

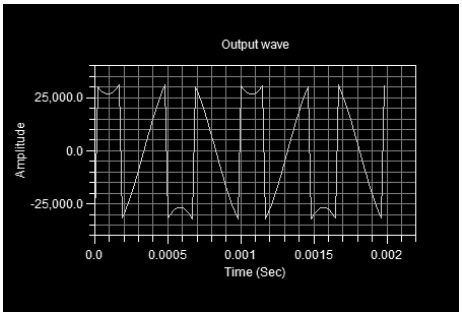
```
void processData(
                const shortfract leftIn[],
                const shortfract rightIn[],
                shortfract leftOut[],
                shortfract rightOut[],
                int count
               )
{
        int tmp;

        for ( int i=0;i<count;i++ ){
        //引き数配列のすべてのデータを処理する
                tmp  =  osc().v * 1.2;
                //振幅が1.2の正弦波を作る
                rightOut[i] = leftOut[i] = tmp;
                //振幅1のところでラップ・アラウンド
        }
}   // processData
```

フレームワークの関数を使う

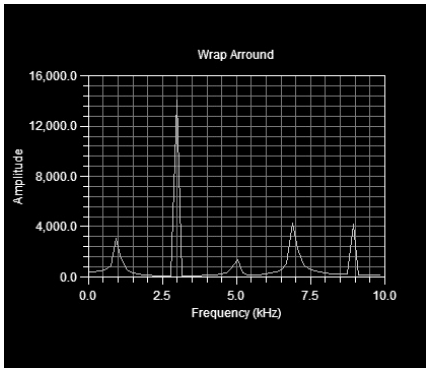
ループの中を変更

もともと最大振幅で振れていたものを1.2倍にしたことで，shortの中に納まらなくなった．C++コンパイラでこのような値をshortに格納するとラップ・アラウンドを起こす

注2：正の最大値から負の最小値へと値が回ることから，ラップ・アラウンドと呼ぶ．



(**a**) 波形



(**b**) スペクトラム

図2
ラップ・アラウンドがおきた波形