オーディオ信号処理で学ぶDSP

第7回　Sin波を作る――テーブル参照のテクニック

堀江 誠一

音声処理に欠かせない関数として，SinとCosがある．メモリ容量や演算速度に余裕のある環境であれば，巨大なテーブルや複雑な演算式で計算できる．しかし，組み込み機器では限られたメモリ容量とCPUパワーで現実的な精度の演算結果を得る必要がある．今回はDSPでSinとCosを実現するための方法について解説する．（編集部）

Sin波とCos波は，発振器のみならず信号処理を行う上であちこちに登場する重要な信号です．ディジタル信号処理に限らず，実際の電子回路においても発振器の設計は重要なポイントとして扱われます．

Sin（正弦）関数，Cos（余弦）関数といった三角関数は広く使われるので，短い時間で精度良く計算することが求められます．三角関数の計算方法にはいろいろありますが，今回はテーブルを使うものについて紹介します．また，三角関数を作りながら，関連する話として丸め誤差についても考えてみます．

最後に，出来上がった発振器を使って少し遊んでみましょう[注1]．

1. テーブル参照による三角関数の計算

テーブル参照による三角関数の計算について考えてみましょう．

テーブルに三角関数を入れておいてそれを引く，という仕組みはすぐに思い付くアイデアですが，あっという間に壁にぶつかってしまいます．それは，精度を高めようとするとテーブルが大きくなり過ぎるということです．円周を16ビット精度で分割すると，必要なテーブルの大きさは64Kエントリになります．これでは話になりません．SDRAMに格納することが前提ならば，不可能な数字ではありませんが，容量が小さな高速内部メモリに置くことはできません．SDRAMに置いた場合もキャッシュは使えるものの，かなりの速度低下を甘んじて受け入れなければなりません．

しかし，三角関数の公式の一つを使えば，巨大なテーブルを分割して小さくし，内部メモリに配置できます．その方法を紹介しましょう．

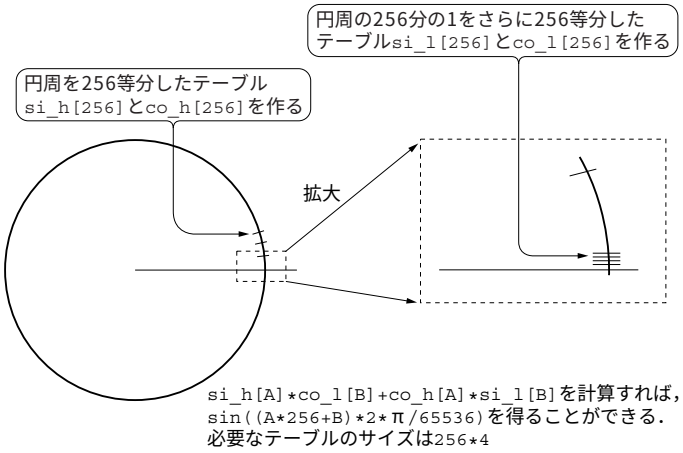

図1　テーブルによる三角関数の計算

● 三角関数の和の公式を使ってテーブルを縮小

三角関数については，高校の数学の時間に教わった以下の公式を覚えているかと思います．

$$\cos(a+b)=\cos(a)\cos(b)-\sin(a)\sin(b) \quad \cdots\cdots(1)$$

$$\sin(a+b)=\sin(a)\cos(b)+\cos(b)\sin(a) \quad \cdots\cdots(2)$$

これらの公式を使うと，64Kエントリの巨大なテーブルを1Kエントリに圧縮できます．その原理を**図1**を見ながら説明しましょう．ポイントは式(1)のa+bを使って，三角関数の引き数を分割するというアイデアです．

先に仮定したように円周を16ビット精度で表すとします．このとき，図のように，円周を256分割する荒っぽいテーブルと，円周の256分の1をさらに256分割する細かいテーブルがあると考えてください．それぞれのエントリのインデックスをA，Bとすると，それぞれのインデックスは円周を表す16ビット・インデックスの上位，下位に

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/

KEYWORD――Blackfin，Sin波，発信機，シングル・トーン，DSB変調，AM変調，FM変調，フランジャ

リスト1　Scilabによるテーブル生成

```
index=[0:255]*2*%pi/256;

co=cos(index)*32768;
write( '¥hco.txt', int(co)', '(i8)');

si=sin(index)*32768;
write( '¥hsi.txt', int(si)', '(i8)');

index=[0:255]*2*%pi/65536;

co=cos(index)*32768;
write( '¥lco.txt', int(co)', '(i8)');

si=sin(index)*32768;
write( '¥lsi.txt', int(si)', '(i8)');
```

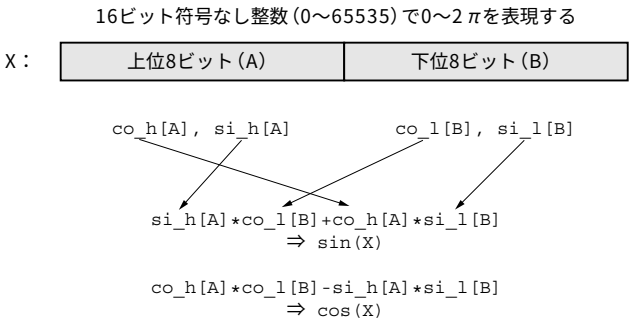

図2　添え字の分解

相当することになります．

この分割したインデックスでおのおののテーブルを引き，式(1)を使って値を合成すれば，16ビット精度で円周を分割した三角関数を作り出すことができます．このとき必要なテーブルは256エントリのものが四つなので，合計で1024エントリとなります．16ビットDSPなら2Kバイトです．

● テーブル参照によるプログラム

四つのテーブルはどのようにして作っても構いませんが，今回は科学技術計算用ソフトウェアScilab[注2]を使ってみました．**リスト1**にその様子を示します．生成されるファイルは，整数が1行に一つある形式です．これに行末のコンマを追加して，C言語で読める形式にします．こうして作ったテキスト・ファイルをソース・コード中の#include文で読み込み，配列の初期値として利用します．

shortfract型を使っているので，整数ではなく固定小数点数表記のまま読み込ませればいいのですが，あえて整数を使っているのには理由があります．Analog Devices社のコンパイラには制限があり，shortfract型変数を静的に初期化すると，C++の初期化用のコードが生成されてしまうからです．これは小規模のテーブルでは問題ありませんが，今回のようにそれ相当の規模になるとプログラムが膨れ上がってしまうのです．

Analog Devices社のツール・グループには改善を提案していますが，残念ながら現時点では大きなテーブルの初期化は整数を使ってごまかすしかありません．

プログラムの流れは至極単純です．与えられた引き数（正弦波の位相にあたる）は16ビット符号なし整数です．先に説明したように円周は65536分割し，16ビットで全周を表します．このように決めた引き数を上位と下位の8ビットに分割し，それぞれを添え字としてSinおよびCosのテーブルを引きます（**図2**）．最後に先の三角関数の和の公式を使って精密な正弦関数の値を計算します．

● 誤差をx86上のシミュレーション機能で評価する

関数を作ることができたので，その誤差を評価してみましょう．プログラムをシミュレータで走らせ，余弦関数の計算結果をprintf()文を使って65536点すべての計算結果をファイルに出力します．

この計算にはVisualDSP++のcompiled simulation機能を使いました．この機能はBlackfinプロセッサのアセンブリ・プログラムをx86のアセンブリ・プログラムに変換して実行するもので，シミュレーション速度が100倍程度になる非常に高速なシミュレーション機能です．評価キットのライセンスでは，インストールから90日間だけ使える機能です．計算だけならばEZ-KIT lite上でやっても構わないのですが，EZ-KIT Liteはprintf関数の実行が遅いため，結果的にcompiled simulationが最速になります．

計算結果をExcelに取り込んでcos()の計算結果と比較したのが**図3**です．縦軸は誤差をLSBで表しています．一見して，正負に振れる大きな誤差があることが分かります．実はこの誤差は関数計算の過程ではなく，係数テーブルを作る過程で入り込んだものです．

● より誤差の少ない表を作る

このプログラムの係数を作るために使ったScilabのプログラムは，計算結果を整数化するためにint()関数を使っています．この関数は小数部を単純に切り捨てるため，誤差の絶対値が最大で1になります．話はそれだけではすみません．切り捨ての方向は0に向かうため，正負で値の切り捨て方向が違うのです．図の誤差に途中から大きなオフ

注2：Scilabに関してはInterface誌2006年9月号の特集記事で扱っている．