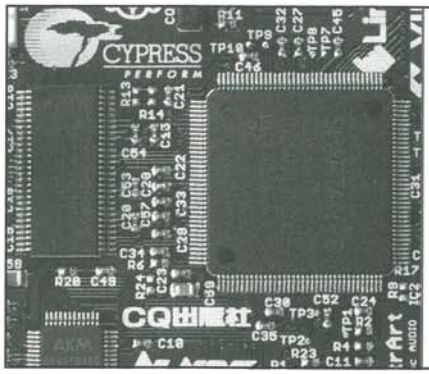


折り返し雑音を阻止!

DAC PCM1795内部の64倍フィルタと合わせてトータル256倍!
CDを176.4 kHzで再生! FPGA搭載USB基板で作る4倍アップサンプラ

第2回　アップサンプラのしくみと作り方

田力 基
Motoi Tariki

本連載では4倍アップサンプラを製作します.

シングル・レート(44.1 k/48 kHz)のPCM信号をFPGAであらかじめ4倍にアップサンプリングしてからD-Aコンバータへ入力します. 実験に使用するUSB-FPGA基板は, 本誌2010年2月号で開発したものです. これにつながるDAC基板には, テキサス・インスツルメンツのΔΣ D-Aコンバータ PCM1795が搭載されています. このICは, デフォルトの設定で, 入力信号がシングル・レートのとき, 内部の8倍オーバー・サンプリング・フィルタで処理を行います. さらに何らかの手段で8倍にアップサンプリングし, トータルで64倍のサンプリング周波数でΔΣ変調器を動かしています. D-Aコンバータで64倍オーバー・サンプリングされるので, 元信号に対して256倍オーバー・サンプリング動作していることになり, ノイズの周波数も同じ比率で高い方へシフトします.

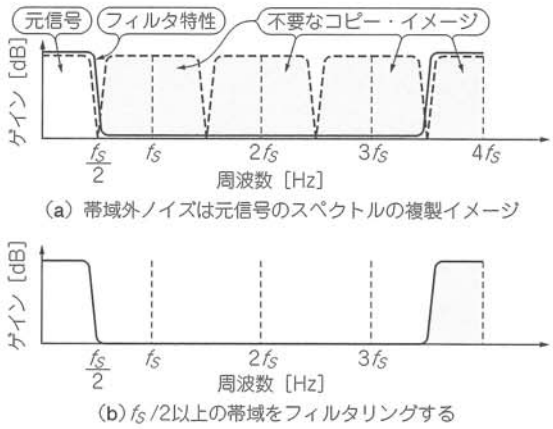


図1　ディジタル・オーディオの宿命…サンプリング周波数f_Sのn倍の周波数に不要なノイズが現れる

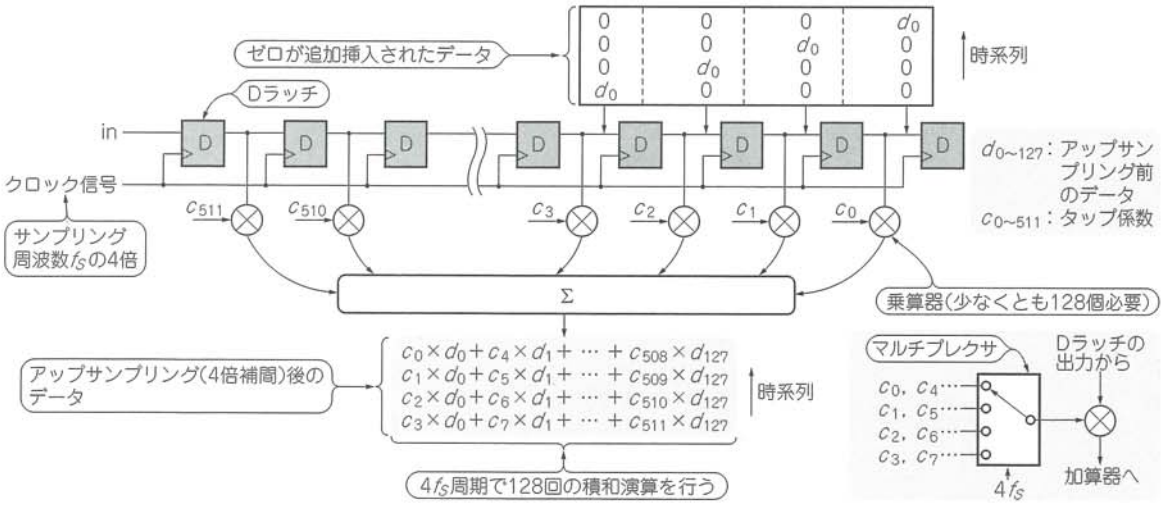


図2　最初に思いつく4倍アップサンプラの回路構成(128個もの乗算器を作り込めるFPGAは存在しない)

データ・レートを4倍に引き上げるため入力データはあらかじめ1個おきに3個のゼロを挿入する. Dラッチを128段にした上で, 右下図のように回路を工夫すると乗算器を512個から128個に減らせる可能性がある

アップサンプラの効能

● サンプリングされた信号はf_Sのn倍の周波数に雑音が発生している

周波数f_Sでサンプリングされた信号は，図1(a)の斜線領域で示すように，f_Sの周期で不要な帯域外ノイズを持ちます．これは元信号のスペクトルの複製イメージです．図1(b)のように，もとの信号の有効スペクトラムである$f_S/2$以下の帯域だけ残すようにフィルタリングする必要があります．

● 遮断特性の緩やかな位相変化が少ないアナログ・フィルタで雑音を除去できる

アップサンプラを使うと，D-Aコンバータの出力後の信号に残る高周波ノイズの帯域をより高い周波数へシフトできます．これにより，D-Aコンバータの出力後に接続するアナログ・ローパス・フィルタを簡単なものにできるという効果が期待できます．

たった1個の積和演算ユニットで作れる4倍アップサンプラ

● ゼロを3個追加してFIRフィルタでならす

図2に，タップ係数が512の一般的なFIRフィルタの回路構成を示します．4倍アップサンプリング処理を行うため，クロック周波数を4倍にしてゼロを追加して，データ・レートを4倍に引き上げます．

そのためにまず，図2に示すチャートのように，あらかじめ入力データ1個おきに3個のゼロを挿入し，回路に入力します．必要な計算のうち4分の3はゼロを掛けるだけなので，最初から計算不要です．4倍に補間されたデータ・レートでみると，結果的に$4f_S$の周期の間に，512÷4 = 128個の積和演算を行えばよいことになります．

● 一般的なFIRフィルタの回路構成そのままでは必要な演算器が多すぎて実現不可能

今回は，CDからリッピングしたオーディオ・ファイル相当である16ビット，シングル・レートの信号を4倍アップサンプラの入力として想定します．FIRフィルタのタップ係数は32ビットに量子化します．この状態では16×32の乗算器が少なくとも128個必要です．実際にはステレオなので合計256個必要です．今回使用するFPGA(XC6SLX4)に搭載された乗算ユニットは8個なので到底足りません．それどころか，どんなFPGAやDSPをもってしても足りません．

● サンプリングの1周期の間に乗算ユニットを512回動かす

サンプリング周波数がf_Sのシングル・レートの入力データにおいて，USB-FPGA基板の動作クロックは$512f_S$です．そこで，並列処理回路を512倍の速度で動作する直列処理回路に置き換えます．具体的には図3のように，入力データをf_Sレートで書き込み，$512f_S$

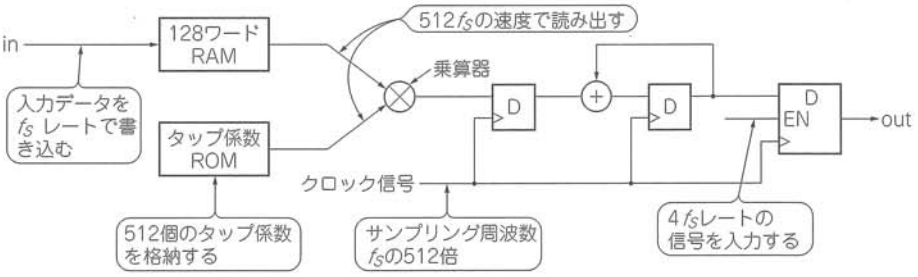


図3 製作したアップサンプラの回路構成(乗算器は1個だけ)
512f_Sレートごとに乗算されたデータを同レートで加算しながらアキュムレータに蓄積し，レジスタ値を4f_Sのレートで読み出せば，4倍アップサンプリングされた信号が取り出せる

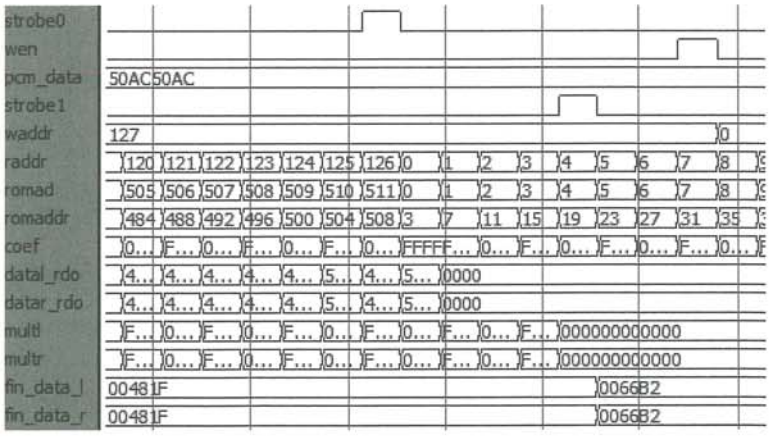


図4 直列動作に置き換えたFIRフィルタのタイムチャート
512f_Sレートで繰り返し新しい値がアキュムレータで蓄積(加算)され，"strobe1" フラグが 'H' のタイミングで4倍アップサンプリングされた信号が出力される

レートで読み出す128ワードの2ポートRAM，512個のタップ係数を512 f_Sレートで読み出すROM，RAMから読み出した入力信号とROMから読み出したタップ係数を掛け算する乗算器，512 f_Sレートごとに乗算されたデータを同レートで加算しながら蓄積するアキュムレータという構成です．アキュムレータのレジスタ値を4 f_Sのレートで読み出せば，4倍アップサンプリングされた信号が取り出せます．この構成であればチャネルあたりの積和演算ユニットはたった1個ですみます．このように，リソースを節約する目的で直列動作にしたり，速度を稼ぐ目的で並列動作にしたり自由に回路構成を変更できるのがディジタル演算処理の特徴です．

▶タイムチャートで動作を確認してみる

図4に示すのは，直列動作に置き換えたFIRフィルタのタイムチャートです．"wen"，"waddr"は，入力データ信号"pcm_data"をRAMへ書き込むライ

2倍処理を2回にわけて4倍アップサンプリングするとメモリを節約できる

今回の製作例では，サンプリング周波数44.1 kHz(48 kHz)の信号を一気に4倍の176.4 kHz(192 kHz)までアップサンプリングしました．元々の各サンプル間に3個のゼロ・データを挿入してから仕上がり周波数の4分の1に帯域を制限するフィルタをかけます．そのため急峻なフィルタが必要となり，タップ係数が512個とかなり大きなフィルタとなりました．

● タップ係数の大きさはチップ面積に影響する

その結果，512ワード×32ビットのROMが必要になります．これは，アップサンプリングを実現するためのチップ面積において大きなインパクトがあるということを意味しています．半導体の集積度は，飛躍的に上がっているため，ロジックの増減はチップ面積にそれほど影響を及ぼさなくなってきています．ところが，メモリの増加はチップ面積へ大きな影響を与えます．

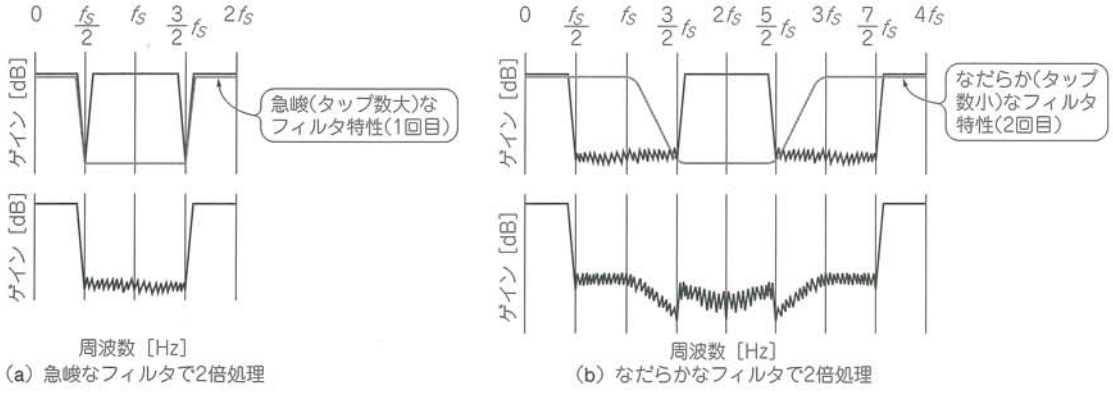
FPGAやDSPは，もともと内蔵リソースとしてメモリが積まれているので，使える分だけ目いっぱい使って性能を追求すればよいのですが，単体ICとしてのD-Aコンバータやディジタル・フィルタICにとって，性能を維持したままチップ面積を可能な限り小さく作るという事はコストに直結する最優先事項です．

● 専用ICは複数回のフィルタ処理をするのが一般的

そこで，上記のような専用ICでディジタル・フィルタを作りこむ場合，チップ面積を可能な限り小さくするために複数回のフィルタ処理を行います．

図Aのように急峻な2倍処理となだらかな2倍処理を順番に行うことで，メモリを節約できます．処理が2回になると必要な乗算器が2個から4個に増えます．ただし，ロジックの面積増加分は微々たるものです．逆にメモリを減らせればチップ面積が減らせます．

図A(b)を見ていただくと，1回処理方式と複数回処理のフィルタ特性は，阻止域での様子が違って見えます．単体ICのD-Aコンバータのデータシートを見ていただくと，複数回処理の特性になっているものがほとんどではないでしょうか．皆さんもそういった面からデータシートを観察されてみると面白いかと思います．



図A　急峻な2倍処理となだらかな2倍処理を順番に行うとメモリが節約できる

1回処理方式と複数回処理のフィルタ特性は，阻止域での様子が違う．単体ICのD-Aコンバータのデータシートを見ると，複数回処理の特性になっているものがほとんどである

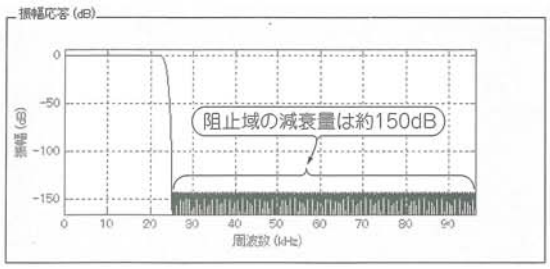
ト・イネーブル信号と書き込みアドレス信号です．書き込まれたシングル・スピード・レートの信号は512f_Sクロックごとに読み出しアドレス信号“raddr”に従って読み出され(“datal_rdo”，“datar_rdo”)，同じクロックでアドレス信号“romaddr”に従って読み出されたタップ係数“coef”と掛け合わされます．ROMの読み出しアドレス“romaddr”はクロックごとに‘4’ずつ飛ばされた値になるようアドレス生成されています．乗算器の出力“multl”，“multr”は，512f_Sレートで，繰り返し新しい値がアキュムレータで蓄積(加算)され，“strobe1”フラグが‘H’のタイミングで4倍アップサンプリングされた信号として出力されます．“strobe0”フラグが‘H’のタイミングで“raddr”が初期値にセットされ，“romaddr”がクリアされています．

これらのアドレスやタイミング・フラグについてはTOPの“USB_AUDIO.vhd”に記述し，アキュムレータとリミッタの機能は別ファイルである“FIRCULC.vhd”に記述します．ある程度長い時間軸で見ないと理解が難しいと思いますので，実際にシミュレーションしながら動作を追ってみることをおすすめします．

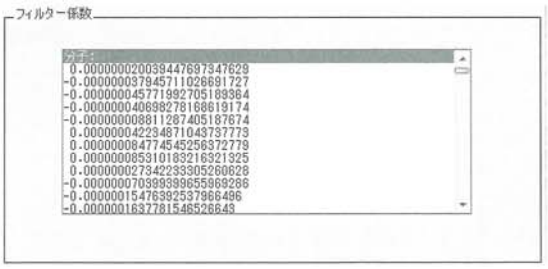
MATLAB & EXCELで! FIRフィルタを設計

MATLABのSignal processing toolboxから利用できるFDAtoolで，タップ係数を求めます．今回は，阻止領域においておよそ100 dB以上の減衰量が取れれば良しと考えました．

(a) 設計仕様を入力する



(b) 阻止領域でおよそ150 dB近い減衰量となる



(c) タップ係数(倍精度浮動小数点による値)が算出された

図5 MATLABのFDAtoolでFIRフィルタのタップ係数を求めた
等リップル近似というアルゴリズムを選択しタップ係数を512とした

	A	B	C
1	2E-08	172	000000AC
2	-3.8E-08	-326	FFFFFFFEBA
3	-4.6E-08	-393	FFFFFFFE77
4	-4.1E-08	-350	FFFFFFFEA2
5	-8.8E-09	-76	FFFFFFFFB4

B列の数式
=ROUND(A1＊2^33，0)

C列の数式
=DEC2HEX(B1，8)

図6　タップ係数をEXCELで16進8桁の符号付きデータに量子化する
カラムCに512個の32ビット係数データが符号付き16進数で求まる

本来は要求スペックを満たせる最小のタップ係数を求めるのが筋ですが，FPGAに搭載された未使用のRAM容量が大きいこともあり(タップ係数と必要なRAMサイズは比例する)，設計をできるだけ簡略化するため，タップ係数を512として算出し，元信号のサンプリング周波数は48 kHz，アップサンプリング後のサンプリング周波数は192 kHzとします．

● STEP1：タップ係数を算出する

図5(a)に示す設計仕様をMATLABのFDAtoolに入力します．等リップル近似というアルゴリズムを選択し，タップ係数を512とすることで，阻止領域でおよそ150 dB近い減衰量(図5(b))のタップ係数(倍精度浮動小数点による値)が求まりました．タップ係数(図5(c))を10進数でテキストに保存します．

● STEP2：タップ係数をROMへ格納するため量子化する

タップ係数を32ビットに丸めるため，EXCELを使って16進8桁の符号付きデータ形式で量子化します．

図6のようにカラムAに512個の浮動小数点係数データを展開し，カラムBを“＝ROUND(A1＊2^33，0)”，カラムCを“＝DEC2HEX(B1，8)”として1行目から512行目までコピーするとカラムCに512個の32ビット係数データが符号付き16進数で求まります．係数によっては，後段のゲイン調整のためにカラムBを2^33から2^32に変更が必要となる場合があります．

```
RAMB16_S36_inst : RAMB16_S36↓
generic map (↓
   INIT => X"000000000",  --  Value of output RAM registers at startup↓
   SRVAL => X"000000000", --  Ouput value upon SSR assertion↓
   WRITE_MODE => "WRITE_FIRST", --  WRITE_FIRST, READ_FIRST or NO_CHANGE↓
   -- The following INIT_xx declarations specify the initial contents of the RAM↓
   -- Address 0 to 127↓
   INIT_00 => X"000002DD000002D80000016BFFFFFFB4FFFFFEA2FFFFFE77FFFFFEBA000000AC",↓
   INIT_01 => X"000009C7000008BB00000387FFFFFDC3FFFFFA81FFFFFACFFFFFFDA3000000EB",↓
   INIT_02 => X"000019DB0000145A0000058CFFFFF6D4FFFFEFB9FFFFF255FFFFFB54000004CA",↓
```

EXCELで量子化したタップ係数

図7　VHDLで記述されたザイリンクス社のRAMモデルでタップ係数を定義する
ザイリンクス社のFPGA開発ツールに付属する“CORE_Generator”で生成した書式とは異なる可能性がある

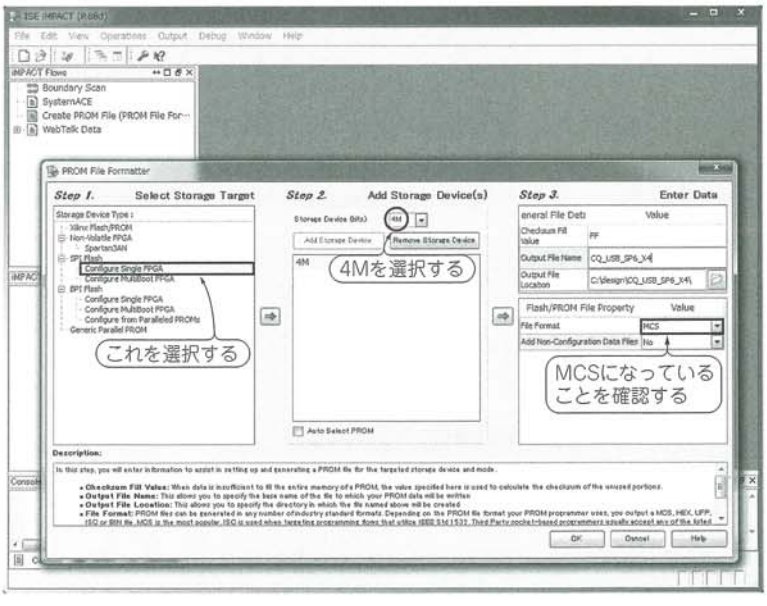
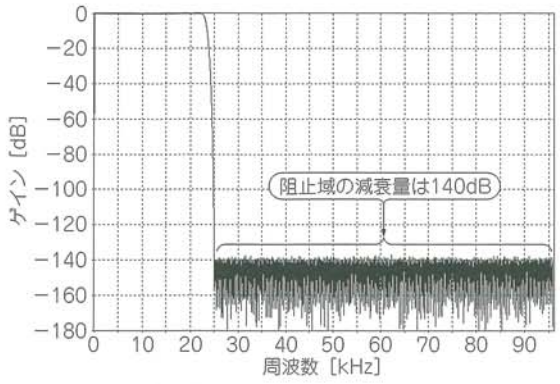
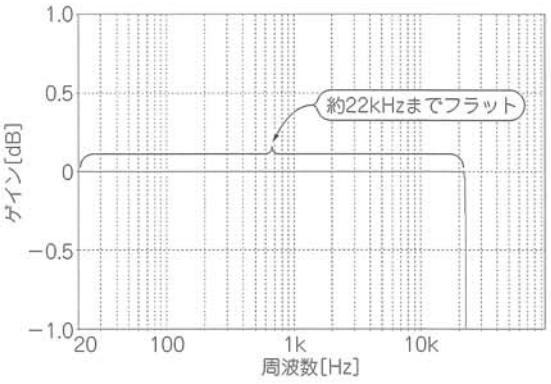


図8　ザイリンクス社のISE Design SuiteでROM用のデータを作成する
ソースコードやROMのデータ・フォーマットに誤りがなければ自動的にフィッティングまで終了する



(a) 阻止域の減衰量は140 dB程度



(b) 通過帯域は約22 kHzまでフラット

図9 設計したFIRフィルタの周波数特性を実測した結果，妥当な結果が得られた
MATLABでタップ係数を求めたときより，わずかに阻止域の減衰量が少ないように見えるのは，係数データをROMに納める前に倍精度浮動小数点から32ビット固定小数点に丸めたため

● STEP3：FPGAのROMモデルでタップ係数を定義する

求めた512個のデータを**図7**のようにRAMの初期値としてVHDLで記述されたザイリンクス社のRAMモデルで定義します．これをROMモデルとして使います．

ザイリンクス社のFPGA開発ツールに付属する“CORE_Generator”で生成した書式とは異なる可能性があります．本連載は，開発ツールの使い方を説明するのが趣旨ではなく，機能の回路的実現方法を説明するのが目的なので，書式の詳細については省略します．書式が少々ややこしいので，CQ出版のホームページに変換ツールを公開します．

● STEP4：ROM用のデータを作成する

VHDLのソースコードやアップサンプラのタップ係数などを変更したら論理合成とフィッティングを行ってFPGAのROMデータを作成します．

ザイリンクス社の“ISE Design Suite”で“Design”タブのProcessesウィンドウから“Configure Target Device”を展開して“Generate Target PROM”をダブルクリックします．ソースコードやROMのデータ・フォーマットに誤りがなければ自動的にフィッティングまで終了し，**図8**のような“PROM File Formatter”ダイアログ・ウィンドウが立ち上がります．

USB-FPGA基板ではXilinx標準のROMではなく，汎用のSPI ROMを使用しているので，“STEP1. Select Storage Target”ウィンドウの中の“SPI FLASH”を展開して“Configure Single FPGA”をクリックし，右矢印を押して“STEP2. Add Storage Device(s)”へ移動します．ウィンドウの中のリストダウンBOXから基板に実装されたROMサイズである“4 M(bytes)”を選択し，右矢印を押して“STEP3. Enter Data”へ移動します．File Formatが“MCS”であることを確認し，出力ファイルのロケーションと名前を確認してOKボタンを押してROMデータを生成します．

● STEP5：実際の回路で特性を確認する

図9に示すのは，周波数特性の実測結果です．MATLABでタップ係数を求めたとき(**図5(b)**)に比べて，わずかに阻止域の減衰量が少ないように見えます．係数データをROMに納める前に倍精度浮動小数点から32ビット固定小数点に丸めているので，多少は特性劣化もあり得ます．大体妥当な結果が出ているのではないでしょうか．

Appendix

USB-DACファームウェアの処理のキモ

音切れ&音飛びなし! 実際のプログラムを公開!

パソコンから外部のハードウェアへオーディオ・データを転送するには，USBが最も多く使われています．音切れや音飛びがなくオーディオ・データを再生するには，ホスト(パソコンなど)とターゲット(USB-DACなど)の連携動作が重要なカギを握っています．この連携動作について，FX2LP(USBマイコン)を事例に実際のソースコードを参照しながら説明します．他のマイコンを使った場合でも基本的な考え方は同じです．

オーディオ・クラス1.0と2.0には，USBファームウ

リストA　データ・パケットがホストからFX2LPに送られてくると割り込みが発生するように設定する

```
EZUSB_IRQ_ENABLE();                    // Enable USB interrupt (INT2)
EZUSB_ENABLE_RSMIRQ();                 // Wake-up interrupt

INTSETUP |= (bmAV2EN | bmAV4EN);       // Enable INT 2 & 4 autovectoring

USBIE |= bmSUDAV | bmSUTOK | bmSUSP | bmURES | bmHSGRANT | bmSOF;
EA = 1;                                // Enable 8051 interrupts
```

リストB　GotSUDフラグが立つと標準またはオーディオ・クラスのリクエストであるかを判断する関数が実行される

```
// Setup Data Available Interrupt Handler
void ISR_Sudav(void) interrupt 0
{
  GotSUD = TRUE;          // Set flag
  EZUSB_IRQ_CLEAR();
  USBIRQ = bmSUDAV;       // Clear SUDAV IRQ
}
```

ェア，FPGA回路の実装面とも大きな違いはほとんどありません．クラス1.0が把握できれば，2.0の実装は，仕様書を参照すれば難しいことでありません．興味のある方はチャレンジされると面白いと思います．

リストC　SETUPDAT［0］をチェックしてリクエストの種類を判断する

SETUPDAT[1]をチェックしてリクエストの内容を判断し，必要な処理を行う関数へ分岐する

```
// Device request parser
void SetupCommand(void)
{
  void  *dscr_ptr;

  if(SETUPDAT[0] == 0x22)
  // If the device request belongs to Audio class 1.0
  {
    switch(SETUPDAT[1])
    {
      case SC_SET_CUR:
        DR_SetCur();
        break;
      default:                          // Invalid request
        EZUSB_STALL_EP0();              // Stall End Point 0
    }
    // Acknowledge handshake phase of device request
    EP0CS |= bmHSNAK;
  }
  else if(SETUPDAT[0] == 0xA2)
  {
    switch(SETUPDAT[1])
    {
      case SC_GET_CUR:
        DR_GetCur();
        break;
      default:                          // Invalid request
        EZUSB_STALL_EP0();              // Stall End Point 0
    }
    // Acknowledge handshake phase of device request
    EP0CS |= bmHSNAK;
  }
  else
  {
    switch(SETUPDAT[1])
    {
      case SC_GET_DESCRIPTOR:           // *** Get Descriptor
        if(DR_GetDescriptor())
          switch(SETUPDAT[3])
          {
            case GD_DEVICE:             // Device
              SUDPTRH = MSB(pDeviceDscr);
              SUDPTRL = LSB(pDeviceDscr);
              break;
            case GD_DEVICE_QUALIFIER:   // Device Qualifier
              // only retuen a device qualifier if this is
              // a high speed capable chip.
```

リクエストの種類を判断する

● パソコンがターゲットをオーディオ・デバイスと認識するまで

コントロール転送におけるセットアップ・ステージにおいて，データ・パケットがホストからFX2LP（USBマイコン）に送られてくると割り込みが発生するように設定します．具体的には，**リストA**に示すように，FX2LPのファームウェアfw.cのmain関数内でUSBIEレジスタを設定します．

この割り込みによってperiph.cのvoid ISR_Sudav(void)関数内でGotSUDフラグが立ち（**リストB**），fw.cの中のSetupCommand()関数が実行されます（**リストC**）．この関数の中で，8バイトのうちの最初のバイトの値（SETUPDAT[0]）によってリクエストが標準リクエストかオーディオ・クラスのリクエストであるかを判断しています．それぞれのリクエストに応じて，今度は2番目のバイトの値（SETUPDAT[1]）をチェックして，リクエストの内容を判断し，必要な処理を行う関数へ分岐していきます．それぞれの分岐動作は，ディスクリプタのGetであったり（標準リクエスト），サンプリング周波数の通知（クラス・リクエスト）であったりします．

このように，コントロール転送がすべての動作の最初の取っ掛かりです．動作を把握しておいて損はないと思います．紙面の都合もありますので，これ以上の動作に関しては，CQ出版のホームページに公開したソースコードを参照してください．

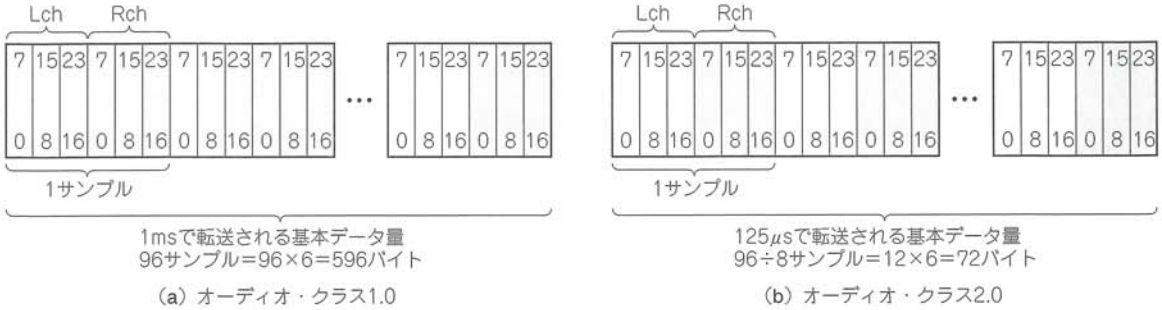
● オーディオ・データ転送時におけるFPGAとFX2LPの連携動作

オーディオ・デバイスとして正しく認識されれば，その後はデータの転送動作になります．オーディオ・データの転送について，シミュレーション波形を使って説明します．

▶USB-FPGA基板はホストからエンドポイント2へオーディオ・データを転送する

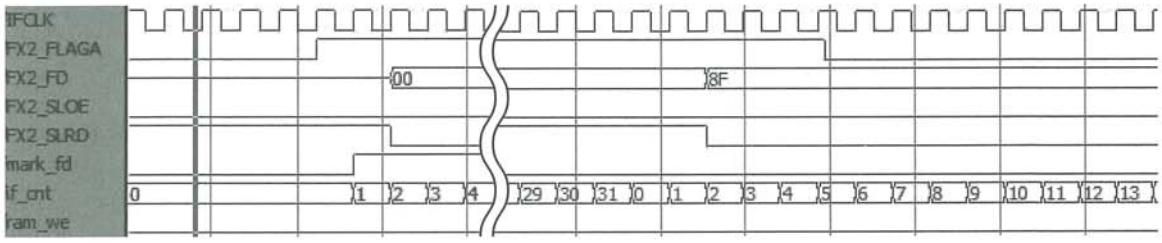
コントロール転送によってサンプリング周波数が通知された後，アイソクロナス転送によってオーディオ・データが転送されます．

FX2LP（USBマイコン）のファームウェアは下記Webページからダウンロードできます．
http：//toragi.cqpub.co.jp/tabid/548/Default.aspx



図B　オーディオ・クラスのバージョンによって量が異なるデータをFX2LPのエンドポイント2に格納する

図C　8マイクロ・フレームに1度だけ576バイトのデータがホストから転送される



図D　SLRDの立ち下がりタイミングでFX2LP内のエンドポイント2にあるデータをFPGA内の2kワードFIFOへ移動開始
最後のデータがエンドポイント2から読み出されると，FLAGAが“L”になってデータ転送が終了する

24ビット/96 kHzを例にとってみると，**図B**に示すように，オーディオ・クラス1.0の場合，1 msのフレーム期間(ホスト側のクロック・レート)に96×6＝576バイトのデータがホストから送信されます．オーディオ・クラス2.0の場合，125 μsのマイクロ・フレーム期間に96×6÷8＝72バイトのデータがホストから送信されます．このデータは，最初にFX2LPに設けられているエンドポイント2に格納されます．そのようにディスクリプタに記述しているので，エニュメレーションの際にホストはオーディオ・データをエンドポイント2へ転送すればよいことを知ります．USB-FPGA基板の場合，オーディオ・クラス1.0をUSB2.0ハイスピードとして動作させるので，厳密には8マイクロ・フレームに1度，1マイクロ・フレーム期間に576バイトのデータがホストから転送されます(**図C**)．

▶エンドポイント2のエンプティフラグ検出がFPGAの起動トリガだ

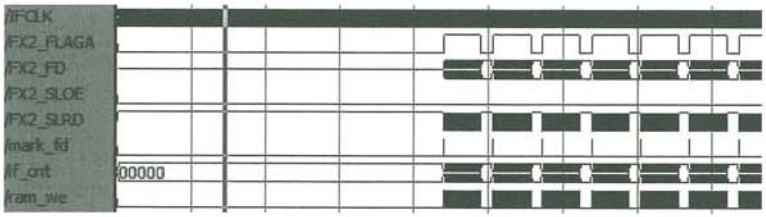
エンドポイント2にデータが入ったことをFPGAが知ることが，USB-FPGA基板内でのオーディオ・データ転送の最初の取っ掛かりになります．periph.cのinit()関数の中でPINFLAGSABレジスタを設定することで，FX2LPからFPGAへ入力するFLAGAピンが，エンドポイント2のエンプティ・フラグとして動作するようにします．

エンプティ・フラグは，FIFOのデータが空になったことを知らせるためのフラグです．デフォルトでアクティブ“L”として動作するので，エンドポイント2にデータが入ると逆にこのピンは“H”に変化します．FPGAはこの変化を捉えてスレーブFIFOモードを起動し，FX2LPからオーディオ・データを抜き出します．

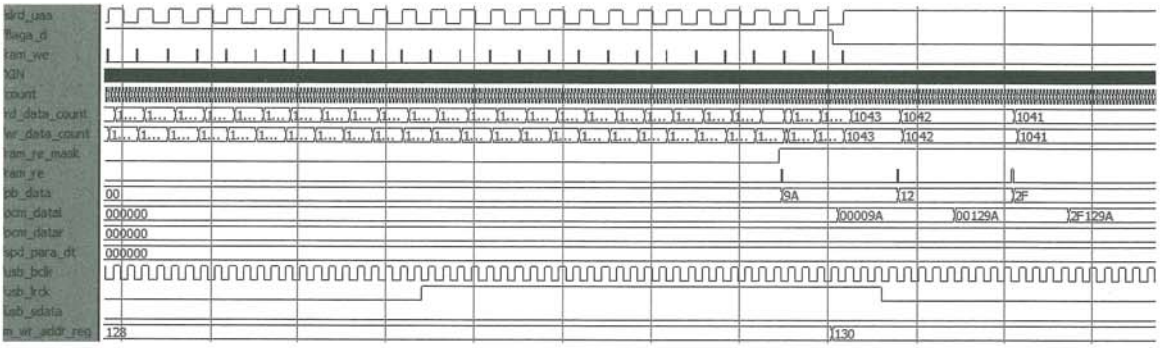
▶FLAGAを検出したFPGAはFX2LPをスレーブFIFOモードにしてデータを読む

FPGAのファームウェアUSB_AUDIO.vhdの中で，FLAGAの立ち上がりを検出し，if_cntを回してタイミングを作り，FX2LPへ読み出して伝えるSLRD信号(slrd_uaa)を生成しています．

SLOE信号も本来は操作が必要ですが，USB-FPGA基板では再生動作のみ(FPGAから見て読み込み)しか行いませんので“L”で固定しています．FLAGAの立ち上がりを検出してエンドポイント2にデータが入ったことを知ると，FPGAはSLRDを立ち下げて読み込みを開始します．SLRDの立ち下がりでFX2LPはデータを出力(更新)しますので，SLRDの立ち上がりタイミングでデータをFPGA内の2kワードFIFOへ



図E　シミュレーションの結果，全オーディオ・データをFPGA内の2kワードFIFOへ読み込みができていることがわかった



図F　FPGAからのオーディオ信号の読み出しタイミングはcount信号の値によって管理される

格納するように，このタイミングでram_weを有効にするようにしています(図D)．

この一連の動作でFX2LPからFPGA内の2kワードFIFOへデータを移すことができます．SLRDが立ち下がって最後のデータがエンドポイント2から読み出されると，FLAGAが“L”になってその(マイクロ)フレーム内でのデータ転送が終了します．

▶ホストからの転送に遅れず読み込めるか確認

図Eに示すのは，オーディオ・クラス2.0の24ビット/192 kHz転送時のシミュレーション波形です．SLRD周期でマイクロフレーム(125 μs)内のエンドポイント2から遅れなく全オーディオ・データをFPGA内の2kワードFIFOへ読み込めていることがわかります．

● FPGA内2kワードFIFOに転送されたデータをDACに送り出す動作

USB-FPGA基板が動作中のとき，フリーランのオーディオ・クロックであるXINパルスによって，メインのオーディオ信号用カウンタであるcount信号を回しています．XINは水晶発振器の出力と等価なクロックです．FPGAからのオーディオ信号読み出しタイミングはすべてこのcount信号の値によって管理されます．以下，図Fに示すタイミング・チャートを使って説明します．

▶FIFOに半分データが溜まると読み出しが可能になる

2kワードFIFOにデータが入り始めると，FIFOのwr_data_count，rd_data_count信号がインクリメントを開始します．これらはFIFO内のデータ数を示しており，USB-FPGA基板では，rd_data_count信号のMSBが“H”になったとき，すなわちFIFOに半分データが溜まった時点でram_re_mask信号を“H”にしてFIFOからの読み出しが可能になったことを示すフラグとしています．このフラグが“H”になっている間，count信号の値とタイミングを取ってFIFOからの読み出しタイミング・イネーブル・パルスであるram_re信号を生成しています．

▶読み出したデータからI^2Sの信号とクロックを生成

同時にFIFOから読み出した8ビットのオーディオ・データ列pb_dataを24ビットのPCMオーディオ・データ列pcm_data1(r)に並びなおします．count信号とタイミングを取り，I^2S信号を作るためのテンポラリなオーディオ信号であるspd_para_dt信号を作り，最終的にI^2S信号であるusb_sdataを生成します．同様にcount信号とタイミングを取ってI^2Sのシリアル・クロックであるusb_bclkやフレーム・クロックであるusb_lrckを生成します．

▶2kワードFIFOの残りデータ量の管理

さらに，FLAGAの立ち下がりでwr_data_count信号の上位8ビットの値をram_wr_addr_reg信号にコピーしています．この信号はFX2LPによってI^2C経由で読み出せるようにしています．

USB-FPGA基板は，ホストからフィードバック・エンドポイントであるエンドポイント6にフィードバック・データの転送要求があると，FX2LPが，その都度FPGAからこの値を読み出し，取得したFIFO内部の現在のデータ・バイト数に応じてホストへ返すフィードバック量を設定しています．