卒 業 研 究 報 告

題　　目

ＡＤＰＣＭ方式による音声のエンコーダ
及びデコーダの設計

指 導 教 員

橘 昌良 助教授

報 告 者

川渕 稔弘

平成　14 年 2 月 7 日

高知工科大学 電子･光システム工学科

目次

はじめに

最近のオーディオはデジタルオーディオが主流となってきており、音声をアナログからデジタルにする時の音声のフォーマットには、無圧縮のフォーマットや多数の可逆フォーマット、非可逆フォーマットがある。各種フォーマットはいかに容量を減らし、原音に近くするためにいくつかの工夫がしてある。今回はその中で基本的な非可逆フォーマットの一つである、ADPCM についての原理と VHDL による設計をして学んでいく。

第１章　音声の符号化について
　アナログオーディオからデジタルオーディオに変換する為の手順や解説について説明している。

第２章　ＡＤＰＣＭについて
　ＡＤＰＣＭ方式乃エンコーダやデコーダを作成するためにまずそのフォーマットの原理を学んでいく。

第３章　ＷＡＶフォーマットについて
　ＷＩＮＤＯＷＳ上で標準的な音声フォーマットであるＷＡＶ方式について解説していく。

第４章　ＡＤＰＣＭのデコーダの設計
　Ｖｉｓｕａｌ　Ｂａｓｉｃ(WINDOWS)とＶＨＤＬ上で動くデコーダを設計する。

第５章　ＡＤＰＣＭのエンコーダの設計
　Ｖｉｓｕａｌ　Ｂａｓｉｃ(WINDOWS)とＶＨＤＬ上で動くエンコーダを設計する。

第６章　まとめ
　デコーダとエンコーダを設計した感想や改良するべき点を記述する。

第1章　音声の符号化について

音とは、鼓膜を振動させて音として感じさせることで、連続的に変化する圧力である。これをコンピュータが認識する非連続的なデータとして取り出しビットに置き換えることをデジタイズという。

標準的なＰＣＭサウンドの場合、定期的に音圧レベルをサンプリングし、その測定値を何段階かのデジタル値（符号）に置き換えていく。グラフにすると、アナログ信号はあくまで切れ目のない曲線的なカーブを描く、デジタル化した音は連続した棒グラフを描いていくようなイメージとなる（図１－１）。

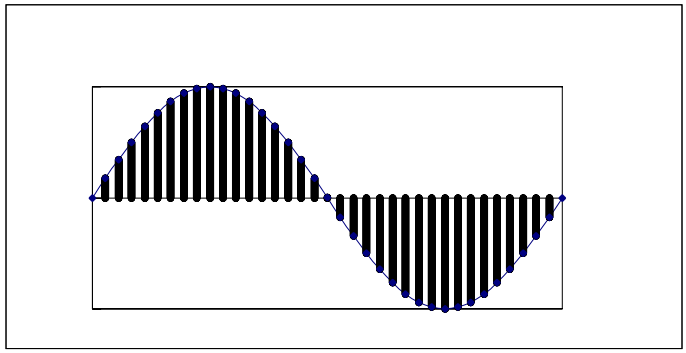

図１－１

この測定の周期を標本化周波数（サンプリングレート）といい、デジタル値に変換する際のビット（bit）数を量子化ビット数という。 音楽ＣＤの場合は、44.1ｋＨｚ（毎秒44100回測定する）のサンプリングレートを適用し、それぞれの測定値を１６ビット（２の１６乗＝６５５３６段階）のレベルに置き換えていく。こうしてできあがったデジタルサウンドは、元の音とは構造上異なるにもかかわらず、人間の聴覚を満足させるレベルであれば自然な音として聴くことが可能である。

つまり、サンプリングレートと量子ビット数を調整することにより、音の品質を変えることが可能である。

一般的な人間の聴覚で感じられる周波数は２０－２０,０００Hz、ダイナミックレンジは１２０ｄｂと言われている。そして２０，０００Ｈｚをデジタルで再現するためには、その倍の周波数でサンプリングしなければならない。（シャノンの定理）

よって音楽ＣＤに使われている４４，１００Ｈｚでサンプリングすれば問題は無い。

ＰＣＭの場合は、ビット数に「２倍の音圧比」に相当する６ｄＢをかければダイナミックレンジを算出する事が可能である。１６ビットなら９６ｄＢ（16 x 6）となる。これは、アナログのレコードの４８ｄＢをはるかに凌ぐが、人間の聴覚は１２０ｄＢ程度のダイナミックレンジまで感知する事が可能であるという説もあり、これを実現するには量子化ビット数を２０ビットまで上げねばならない。（ＤＶＤオーディオは最大２４ビットに対

応。）

よって１秒間のＣＤＤＡにおけるＰＣＭのデータサイズはサンプリングレート４４，１００Ｈｚビットレート１６bit ステレオでサンプリングすると４４，１００Ｈｚ×１６×２ｃｈで1，４１１，２００ｂｐｓ　1.4Ｍｂｐｓ　１７０ｋｂｙｔｅになる。

第2章　ＡＤＰＣＭについて

２．１　ＡＤＰＣＭとは

ＡＤＰＣＭとは、英語名で Adaptive Differential Pulse Code Modulation の略であり、日本語名では適応差分 PCM と呼ばれている。ADPCM は、音声データを圧縮する手法の1つで、完全に元の状態にもどすことができない非可逆フォーマットの一つである。

ADPCM(適応差分 PCM)もＰＣＭと同様に音声信号をサンプリングする。しかし、量子化の方法が PCM とは異なっていて、音声信号のサンプル値の間には相関があり、過去の入力信号から現在の入力信号を予測することが可能であるという特徴を利用して、予測誤差(サンプル値の差分)を量子化している。

これにより16bitの信号を3bit～8bitにすることで圧縮することが可能である。

そして同一の量子化ビット数の場合, PCM よりも ADPCM のほうが音質は良くデータ量も少なくなるが、回路は複雑になる。

２．２　他のフォーマットとの比較

他のフォーマット

・　ＡＴＲＡＣ(Adaptive Transform　Acoustic Coding)

音声を高音質のまま圧縮して少ない容量のメディアにも収まるようにする技術で、この圧縮法は人の耳に聞こえにくい大きな音に埋もれた小さな音のみを間引いて圧縮する。その結果ほぼ原音に忠実に拡張して復元、再生する。

ＡＴＲＡＣ2はＭＤに採用。圧縮率は約1／5である。

ＡＴＲＡＣ3はメモリースティックウォークマンや、新しいメモリーウォークマンＭＤＬＰなどに採用されている。圧縮率は約1／10である。

・　CCITT A-Law

CCITT G.711 で規定される非直線量子化の一つで電話回線につかわれており、圧縮率は1／2となっている。

・ CCITT μ-Law

CCITT G.711 で規定される非直線量子化の一つで、日本の電話回線に使われている。圧縮率は上と同じで1／2となっている。

・ CELP (code-excited linear prediction)

ＣＥＬＰとは符号励振線形予測の略語で、CELP では、あらかじめ用意したコードブックを合成し音声を表す。伝送されるのはデジタル化された音声信号ではなく、入力音声にもっとも近い音声を与えるコードブックのインデックスや合成のパラメータが伝送され、これにより圧縮される。

・ ＭＰ３ (MPEG1 Audio Layer-3)

MPEG1 という動画圧縮技術に含まれる音声圧縮技術で、圧縮率は約 1/20～1/10 である。ＭＰ３では、人間の耳で聴こえない周波数の情報やその瞬間に聞こえにくい音をカットしてしまうという工夫により、周波数をブロックごとに分けてこれを圧縮する。ステレオ/16bit/44.1kHz の CD クオリティの音声ならば 128kbps のビットレートでもほとんど音質を変えることなく圧縮することが可能である。

・ TwinVQ (Transform-domain weighted interleave Vector Quantization)

NTT サイバースペース研究所が開発した高品質音声圧縮技術で、MP3 と同程度の音質を維持しながら音楽 CD の 1/18 程度に圧縮できる。ただ、MP3 と比較してエンコードに時間がかかるのが難点である。圧縮方法は対象とする音声を分析して、予めデータベース内に用意していた標準パターンと比較し、これに近いものを圧縮符号に置き換えていく。音によっては音質にばらつきがでてくる。

比較

フォーマット	長所	短所
ATRAC	圧縮率が良い。	回路が複雑になる。
CCITT A-Law	回路が簡単。	圧縮率が悪い。
CCITT μ-Law	回路が簡単。	圧縮率が悪い。
CELP	圧縮率が良い、音声の劣化が少ない。	音声が非常に劣化する。
MP3	圧縮率が良い、音声の劣化が少ない。	回路が複雑になる。
TwinVQ	圧縮率が良い、音声の劣化が少ない。	データベースの保存のためにメモリが必要になり、回路が複雑になる。
ADPCM	回路が比較的簡単、音声の劣化が少ない	ステップサイズのためにメモリが必要。
PCM	回路が簡単、音声の劣化は無い。	無圧縮のフォーマットなので、ファイルのサイズが大きくなる。

表2－1　フォーマットの長所や短所

以上比較してみたところＡＤＰＣＭ方式が圧縮率、回路規模の面から見て適している。

第3章 WAVフォーマットについて

3．1 ヘッダ情報について

音声フォーマットであるWAVファイルには、ファイルの先頭にヘッダ情報が記されており、その後に波形データ部分が記録している。標準的なPCMのヘッダ情報は44 byteで構成されている。

実際のWAVフォーマットデータのHEXを見てみると、表3－1の様な形になっている事が分かる。

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	52	49	46	46					57	41	56	45	66	6D	74	20
内容	RIFF				ﾌｧｲﾙｻｲｽﾞ－8				WAVE				fmt			
01	10	00	00	00	01	00	02	00	44	AC	00	00	10	B1	02	00
内容	fmt のバイト数				ID		ステレオ		44100(ｻﾝﾌﾟﾘﾝｸﾞﾚｰﾄ)				1秒間のデータ量			
02	04	00	10	00	64	61	74	61								
内容	ﾌﾞﾛｯｸｻｲｽﾞ		16(bit 数)		data				波形データサイズ				波形データ			

表3－1 ヘッダ情報

ヘッダ情報は波形データサイズまでがヘッダ情報であり、その後はデータ部分となる。

そして表3－1を具体的に説明したのが表3－2であり、これを見ると標準的な無圧縮PCM、44100Hz、16 bitステレオのWAVデータの場合は、ファイルサイズと波形データのバイト数以外は変わらないということが分かる。

バイト数	内容	説明
4 Byte	RIFF	RIFFヘッダ
4 Byte	ファイルサイズ－8	これ以降のファイルサイズ
4 Byte	WAVE	WAVE ヘッダ
4 Byte	fmt	fmt ヘッダ
4 Byte	Fmt ヘッダのバイト数	標準のPCMなら 10 00 00 00
2 Byte	フォーマットID	WAVのフォーマットを示す表3－3参照
2 Byte	チャンネル数	ステレオだと 0200 モノラルだと 0100
4 Byte	サンプリングレート	44100 の場合 44 AC 00 00
4 Byte	1秒間のデータの量	44100Hz16bit ステレオであれば 44100*2*2=176400(10 B1 02 00)
2 Byte	ブロックサイズ	16bit ステレオなら 2*2=4(0400)
2 Byte	量子化ビット数	16bit の場合 10 00
4 Byte	Data	文字列
4 Byte	波形データのバイト数	波形データのバイト数

表3－2ヘッダ情報の詳細

0x0000	Unknown
0x0001	PCM
0x0002	MS ADPCM
0x0005	IBM CSVD
0x0006	A-Law
0x0007	μ-Law
0x0010	OKI ADPCM
0x0011	IMA/DVI ADPCM
0x0012	MediaSpace ADPCM
0x0013	Sierra ADPCM
0x0014	ADPCM (G.723)
0x0015	DIGISTD
0x0016	DIGIFIX
0x0020	YAMAHA ADPCM
0x0021	SONARC
0x0022	TrueSpeech
0x0023	Echo Speech1
0x0024	AF36 (Audiofile)
0x0025	Apix
0x0026	AF10 (Audiofile)
0x0030	AC2 (Dolby)
0x0031	GSM 6.10
0x0033	ANTEX ADPCM
0x0034	VQLPC (Control Resources)
0x0035	DIGIREAL
0x0036	DIGIADPCM
0x0037	CR10 (Control Resources)
0x0040	ADPCM (G.721)
0x0101	IBM μ-LAW
0x0102	IBM A-LAW
0x0103	IBM ADPCM
0x0200	Creative Labs ADPCM
0x0300	FM TOWNS
0x1000	Olivetti GSM
0x1001	Olivetti ADPCM
0x1002	Olivetti CELP
0x1003	Olivetti SBC
0x1004	Olivetti OPR

表3－3フォーマットID

3．2　WAVデータ内の波形情報について

3．1にて解説したWAVデータのヘッダ情報の後にWAVデータの波形データが並ぶ。そしてその波形データの形式がPCMの量子化ビット数が8 bit の場合と16 bit の場合では異なっている。8 bit なら符号無し(unsigned) 0～255，無音時は128となっている。そして、16 bit の場合は、符号付き (signed) －32768～＋32768，無音時は0)となっている。双方の違いを図に表したのが図3－1と図3－2である。

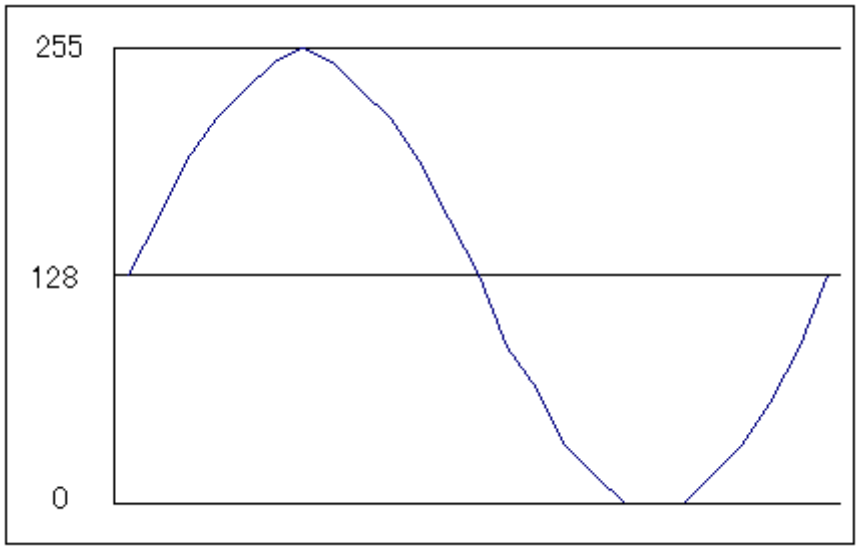

図3－1　8bitPCMデータの場合

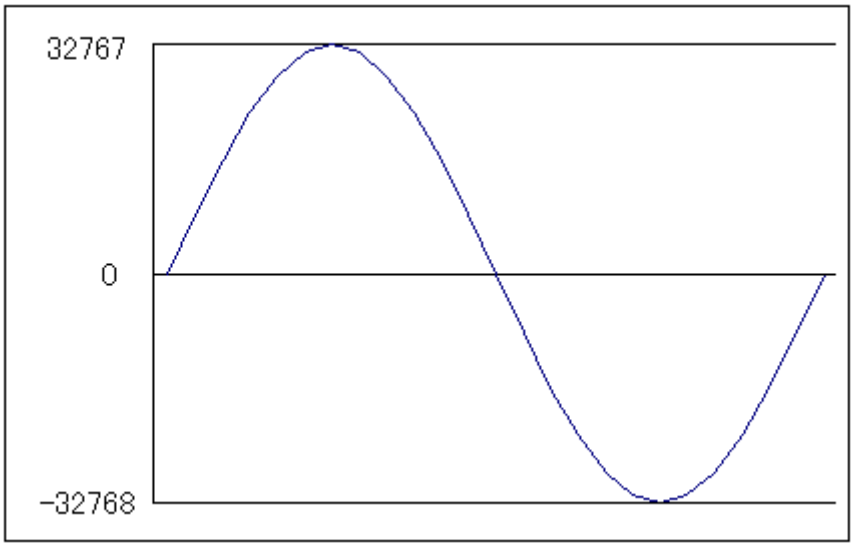

図3－2　16bitデータの場合

WAVファイルでは、波形データはL，R、L、Rという順番で時間順に記録されて行く。そして記録されていく数値はバイトオーダ[0]をリトルエンディアン[1]にしなければならない。
以上が無圧縮PCMであるWAVファイルのデータ構造である。

[0] 複数バイトの整数を並べるとき、どのような順番で並べて置くのか
[1] リトルエンディアン。番地が小さい方に下位バイトを置く。逆にビッグエンディアンというのは番地が小さい方に上位バイトを置く

第4章　ＡＤＰＣＭのデコーダの設計

この章では、ＷＩＮＤＯＷＳ上で動作するソフトウェアとＶＨＤＬによるＡＤＰＣＭのデコーダを設計していく。

４．１　ＡＤＰＣＭのデコーダの概要

- 圧縮率は約1／5から1／2とする。
- ＷＩＮＤＯＷＳ版のソフトは Visual Basic を使うこととする。
- ＷＩＮＤＯＷＳ版のソフトは無圧縮ＰＣＭへの変換ソフトとする。
- ＷＩＮＤＯＷＳ版のソフトを使ってＶＨＤＬのソフトの動作を照合して確認する。

４．２　ＡＤＰＣＭデコーダの設計

まずＡＤＰＣＭのデコーダのブロック図を書くと図４－１となる。

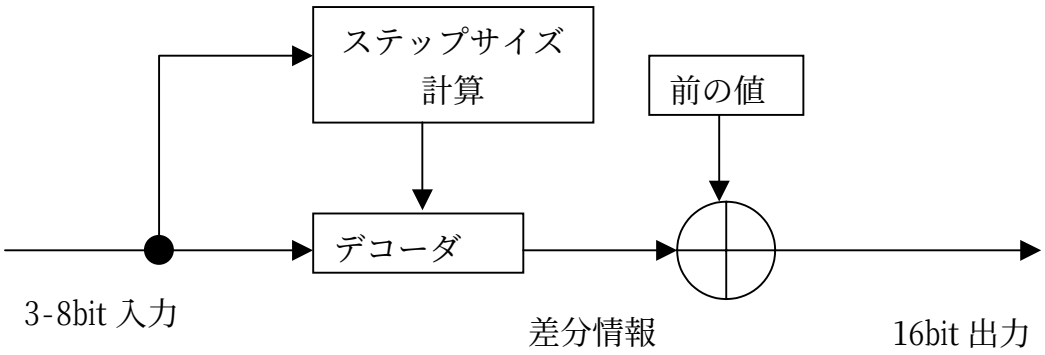

図４－１デコーダのブロック図

ADPCM は次の符号を予測して圧縮するのであるが、その予測するための計算がステップ計算であり、計算されデコーダにわたされる。その後、デコーダで作成された差分情報を元に前回の値を参照して16 bit のデータが出力される。

さて実際の計算であるが、ステップサイズは 8～32767(表4－1を参照)まで88段階に増減する。その増減を決めるのは入力の値であり、その値は4 bit 以上の場合は、表4－2のようになる。
3bit の場合は表４－３のようになる。

8, 9, 10, 11, 12, 13, 14, 16, 17,19, 21, 23, 25, 28, 31, 34, 37, 41, 45,50, 55, 60, 66, 73, 80, 88, 97, 107, 118,130, 143, 157, 173, 190, 209, 230, 253, 279, 307,337, 371, 408, 449, 494, 544, 598, 658, 724, 796,876, 963, 1060, 1166, 1282, 1411, 1552, 1707, 1878, 2066,2272, 2499, 2749, 3024, 3327, 3660, 4026, 4428, 4871, 5358,5894, 6484, 7132, 7845, 8630, 9493, 10442, 11487, 12635, 13899,15289, 16818, 18500, 20350, 22385, 24623, 27086, 29794, 32767

表４－１　ステップサイズ表

入力(上位4 bit)	ステップサイズの増減段階
0000 or 1000	-1
0001 or 1001	-1
0010 or 1010	-1
0011 or 1011	-1
0100 or 1100	+2
0101 or 1101	+4
0110 or 1110	+6
0111 or 1111	+8

表4－2　4bit以上場合の増減段階

入力(3bit)	ステップサイズの増減段階
000 or 100	-1
001 or 101	-1
010 or 110	+2
011 or 111	+4

表4－3　3bitの場合の増減段階

このステップサイズの増減式はエンコーダとデコーダに共通でつかわれる。これを Visual Basic で書くと、始めにステップサイズを2次元配列に代入する。

```
stepsize(1, 1) = 8
stepsize(2, 1) = 9
stepsize(3, 1) = 10
stepsize(4, 1) = 11
stepsize(5, 1) = 12
stepsize(6, 1) = 13
stepsize(7, 1) = 14
stepsize(8, 1) = 16
stepsize(9, 1) = 17
        ・
        ・
        ・
```

ENC＝入力
stadr=ステップサイズのアドレス
Combo1.Text=圧縮率　Combo1.Text/16

```
If ENC(Combo1.Text - 2) = 0 Then
    stadr = stadr - 1
Else
  stadr = stadr + 2
  If Combo1.Text = 3 Then
       If ENC(Combo1.Text – 3) = 1 Then stadr = stadr + 2
  Else
       If ENC(Combo1.Text - 3) = 1 Then stadr = stadr + 4
       If ENC(Combo1.Text - 4) = 1 Then stadr = stadr + 2
  End If
End If
```

まず上位の 2bit 目が0ならば、ステップサイズを-1 とする。0でなければステップサイズを+2する。そして圧縮率が3／16の場合は上位の3 bit 目が1ならばステップサイズを+2加えて計算を終える。圧縮率が4／16以上の場合はステップサイズをさらに+4加える。そして上位の4 bit 目が1であれば、ステップサイズをさらに+2する。以上の処理でステップサイズが求められる。

しかし88段階目を越えてしまうと数がおかしくなるために、

```
If stadrL < 1 Then stadrL = 1
If stadrL > 88 Then stadrL = 88
```

以上のプログラムを追加して1未満になった場合には1にし、88より大きくなった場合は88に固定するようにしている。

次はデコーダの部分であるが、まずエンコードされた情報の上位1 bit は正負を定める。そして計算式は以下の様になる

sabun = 差分情報
old = 前のデコーダの出力
decode = デコーダの出力
L()＝入力
stepsize=現在のステップサイズ
N=圧縮率 N/16

$$sabun = stepsize \div 2^{N-1} + \sum_{k=1}^{N-1}\left(stepsize \times L(N-k) \div 2^{k-1}\right)$$

$$decode = old + sabun \times -1^{L(N)}$$

以上の様に数式で表すことが可能である。

これを Visual Basic で書くとまず

$$stepsize(stadr, N) = stepsize \div 2^{N}$$

を定義する。

```
For I = 1 To 88
   stepsize(I, 2) = stepsize(I, 1) ¥ 2
   stepsize(I, 3) = stepsize(I, 1) ¥ 4
   stepsize(I, 4) = stepsize(I, 1) ¥ 8
   stepsize(I, 5) = stepsize(I, 1) ¥ 16
   stepsize(I, 6) = stepsize(I, 1) ¥ 32
   stepsize(I, 7) = stepsize(I, 1) ¥ 64
   stepsize(I, 8) = stepsize(I, 1) ¥ 128
Next I
```

デコーダの Visual Basic のソースは

ENCL = 入力(Combi1.Text bit)

Combo1.Text = 圧縮率　Combo1.Text /16

sabun = 差分情報

decode = デコード出力

stadr = ステップサイズのアドレス

```
For i1 = 1 To Combo1.Text - 1
   sabun = sabun + stepsize(stadr, i1) * ENCL(Combo1.Text - i1 - 1)
Next i1
sabun = sabun + stepsize(stadr, Combo1.Text)
If ENCL(Combo1.Text - 1) = 1 Then sabun = sabun * -1
decode = decode + sabun
```

以上のプログラムでデコードが可能である。しかしオーバーフローする可能性があるので、以下の様に対策している。

```
If decode > 37267 Then decode = 37267
If decode < -37267 Then decode = -37267
```

4．3 VHDLの設計

WINDOWS上のプログラムを参考にしてステップサイズの計算とデコーダを作る。まずステップサイズの計算だが、まずステップサイズを2進数にして配列に代入する。下のプログラムではSTEPSIZE(STEPADR)でステップサイズが求められる様に配列に2次元配列に代入している。

```
subtype STEPU is std_logic_vector(15 downto 0);
type STEPSZ is array (0 to 87) of STEPU;
constant STEPSIZE : STEPSZ := (
"0000000000001000",
"0000000000001001",
"0000000000001010",
        •
        •
        •
              );
variable    STEP: std_logic_vector(15 downto 0);
variable STEPADR : integer range 0 to 87 :=0;
```

ステップサイズの増減は以下の様なプログラムで動作する。

```
if (N=3) then
    case A((N-2) downto (N-3)) is
          when "10" => K := 2;
          when "11" => K := 4;
          when others => K := -1;
          end case;
else
    case A((N-2) downto (N-4)) is
    when "100" => K := 2;
    when "101" => K := 4;
    when "110" => K := 6;
    when "111" => K := 8;
    when others => K := -1;
    end case;
end if;
```

ステップサイズの増減は3 bit の場合と4 bit 以上の場合計算式がちがうためにN bit の場合のデコードは以下なプログラムで作る。

```
for I in 2 to N loop
        if (A(N-I)='1') then
                if (I= 2) then
                        SAB := STEP;
                else
                        SAB := SAB + (ZERO(I-3 downto 0) & STEP ( 15 downto I-2));
                end if;
        end if;
end loop;
SAB := SAB + ("000" & STEP (15 downto 3));
```

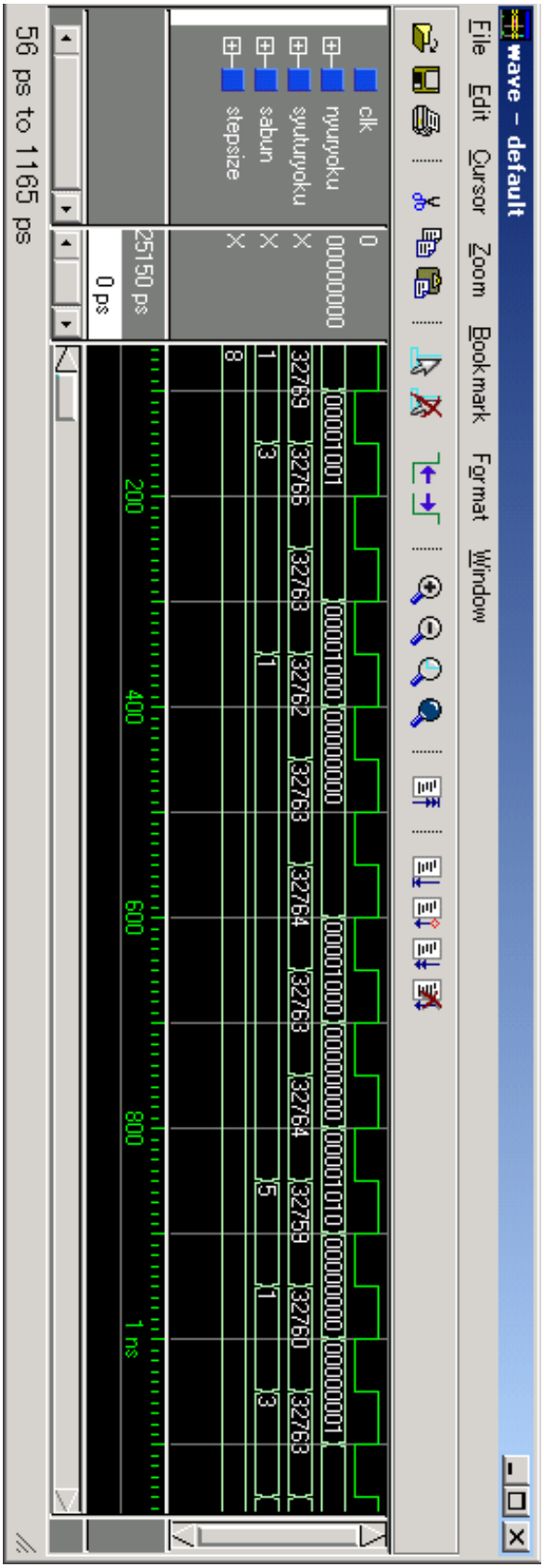

図4－2　デコーダのシミュレーション結果

信号の解説

clk クロック

nyuuryoku エンコーダされたデータの入力

syuturyoku デコード出力

sabun 前のデータとの差分

stepsize ステップサイズ

動作確認のために初めの 3 クロック分を表に取り出すと

Clk	nyuuryoku	syuturyoku	sabun	Stepsize
1	1001	32766	3	8
2	1001	32763	3	8
3	1000	32762	1	8

表4－4 動作確認表

1クロック目から2クロック目に変わる時

$$sabun = 8\times0+4\times0+2\times1+1\times1=3$$

入力値の上位１ビットが１のため

$$sabun = 3\times(-1)=-3$$

$$syuturyoku = 32766-3=32763$$

となるので、2クロック目の数字になる。

次に2クロック目から3クロック目に変わる時

$$sabun = 8\times0+4\times0+2\times0+1\times1=1$$

入力値の上位１ビットが１のため

$$sabun = 1\times(-1)=-1$$

$$syuturyoku = 32763-1=32762$$

となるので、3クロック目の数字になる。

ステップサイズのアドレスは

2BIT目がすべて0のために全てアドレスが1引かれる。しかし、現在アドレスが1のためにこれ以上引かれることはない。よってステップサイズは8のままである。

図4－2に載っている全ての信号で計算してみた所、確実に動作しているのが分かった。

第5章　ＡＤＰＣＭのエンコーダの設計

5．1　ＡＤＰＣＭのエンコーダの概要

- 圧縮率は約1／5から1／2とする。
- ＷＩＮＤＯＷＳ版のソフトは Visual Basic を使うこととする。
- ＷＩＮＤＯＷＳ版のソフトは無圧縮からの変換ソフトとする。
- ＷＩＮＤＯＷＳ版のソフトを使ってＶＨＤＬのソフトのソースとなるデータを作成する。

5．2　ＡＤＰＣＭエンコーダの設計

まずＡＤＰＣＭのエンコーダのブロック図を書くと図5－1となる。

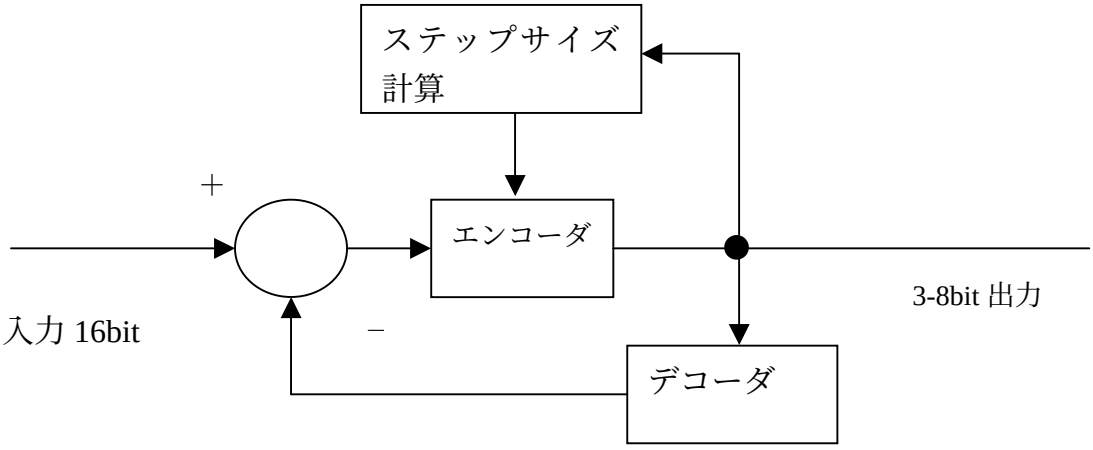

図５－１エンコーダのブロック図

ADPCM のエンコーダであるが、ブロック図を参照してみれば、第4章で説明した ADPCM のデコーダにエンコーダ部分を結合すると ADPCM のエンコーダが完成するのが分かる。エンコーダは16 bit の入力を受け取り3～8 bit のデータを出力する。この際音声は多少劣化する。そのために可逆圧縮では必要のないデコードの作業が必要となる。

エンコードの計算式を書くと、

X = エンコードされたデータ (N bit)

D = 前のデータとの差分

Stepsize = ステップサイズ

$$X_2 = \frac{D \times 2^{N-2}}{stepsize}_{10}$$

以上のような式でエンコードが可能である。
これを Visual Basic で表すと
ENC＝出力
diff=差分
stadr=ステップサイズのアドレス
Combo1.Text=圧縮率　Combo1.Text/16

```
For i1 = 1 To Combo1.Text - 1
    If diffL >= stepsize(stadr, i1) Then ENC(Combo1.Text - i1 - 1) = 1: diff = diff -
stepsize(stadr, i1)
Next i1
```

以上の式で求めることが可能である。

5．3　VHDLの設計

WINDOWS上のプログラムを参考にしてエンコーダを作る。
ステップサイズとデコーダの部分は 4 章で製作したため今回はエンコードする部分のみを作成する。
まず前の値よりも大小を判定し上位の1 bit に記録する。

```
D:= (not X(15) & X(14 downto 0));
HANTEI := ('1' & D) - ('0' & GENZAI);
if (HANTEI(16)='1') then
    D := D - GENZAI;
        L(N-1) := '0';
else
        D := GENZAI - D;
        L(N-1) := '1';
end if;
```

残りのビットで圧縮する。

```
for I in 2 to N loop
    HANTEI := ('1' & D) - (ZERO(N-2 downto 0) & STEP(15 downto N-2));
    if (HANTEI(16) ='1') then
            L(N-I) := '1';
        D := HANTEI(15 downto 0);
                else
        L(N-I) := '0';
         end if;
end loop;
```

次のページに分かりやすい様に数を変えて、実行した時のタイムチャートを載せた。エンコーダの中にデコーダが内蔵されているために、デコードとエンコードが確実に動作していないと、タイムチャートをみれば一目瞭然である。

さて、上記のプログラムで実際にＡＤＰＣＭに圧縮が可能である。しかし問題があり、出力が8 bit に固定されているために4 bit で出力しても上位4 bit に不定のデータが入ってしまう。これを解決するのはデータ長ごとに output の変数を変えることにより、可変することが可能である。

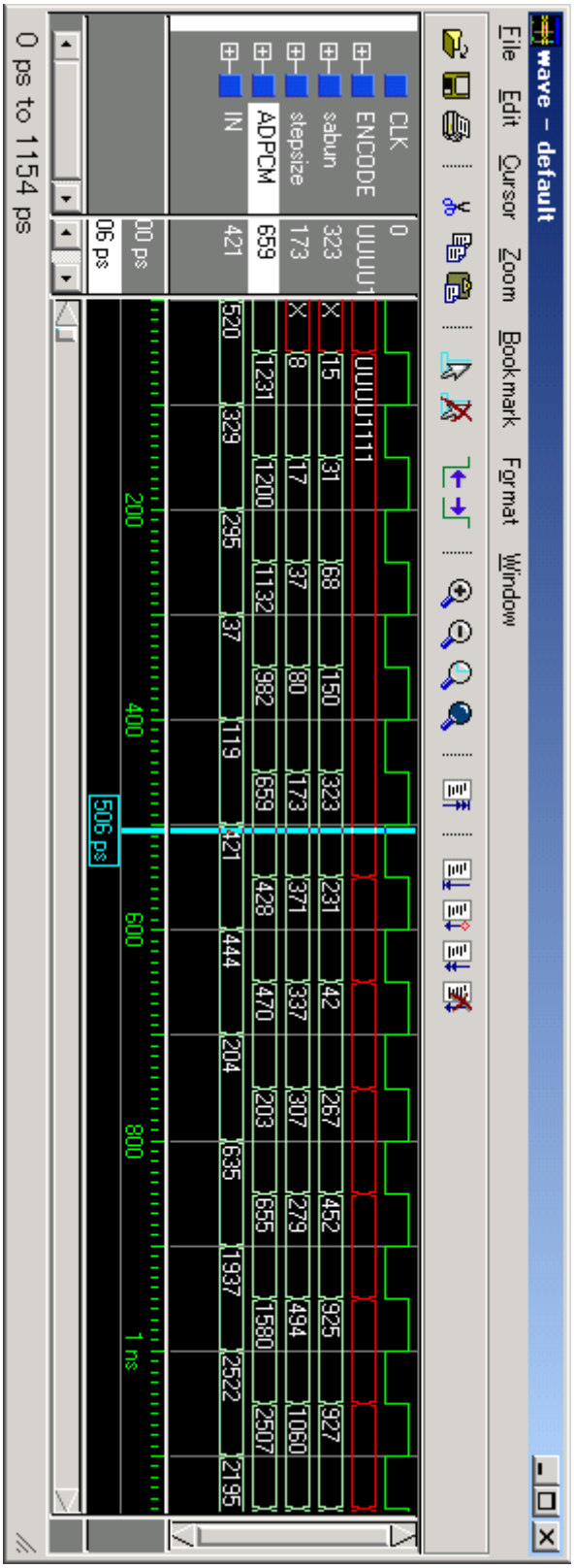

図５－２　エンコーダのシミュレーション結果

初期値　１２４６

信号の解説

CLK　　クロック

ENCODE エンコード出力

sabun　ADPCMで計算された予測値を修正した差分値

step　　ADPCMで計算された次の値との差の予測値

ADPCM　入力の信号をエンコードした後デコードした値

lN　　入力の信号の値

同期式のためにクロックに応じて出力が変化するようになっている。

500psまで入力の値とADPCM方式でエンコードした後デコードした値の誤差が大きいが、その後はほぼデコードした値と入力の値の誤差が少なくなっていく。VHDLの動作を確認するために、動作確認のために初めの 3 クロック分を表に取り出すと

CLK	ENCODE	sabun	stepsize	ADPCM	IN
1	1111	15	8	1231	520
2	1111	31	17	1200	329
3	1111	68	37	1132	295

表5－1エンコーダの動作確認表

1クロック目から2クロック目に変わるとき

$$sabun = stepsize \div 2^{N-1} + \sum_{k=1}^{N-1}\left(stepsize \times L(N-k) \div 2^{k-1}\right)$$

$$D = 329 - 1231 = -902$$

$$ENCODE = D \div stepsize = -902/17 = -53$$

ENCODEは２進数４ＢＩＴで*ENCODE*が負の数なので
上位の1*BIT*は１となり、残りの３ＢＩＴも－５３だと
オーバーフローしてしまうので最大値の111と上位１ＢＩＴ
を足して１１１１となる

差分は

$$sabun = stepsize \div 2^{N-1} + \sum_{k=1}^{N-1}\left(stepsize \times L(N-k) \div 2^{k-1}\right)$$

この式を使って

$$sabun = 31$$

$$ADPCM = 1231 - 31 = 1200$$

となるので、2クロック目の数字になる。

ステップサイズのアドレスはエンコード出力が 1111 のために＋８アドレスが加算される。
その結果ステップサイズは３１となる。

次に2クロック目から3クロック目に変わる時

$D = 295 - 1200 = -905$
$ENCODE = D \div stepsize = -905 / 37 = -24$
ENCODEは２進数４ＢＩＴで*ENCODE*が負の数なので
上位の1*BIT*は１となり、残りの３ＢＩＴも－５３だと
オーバーフローしてしまうので最大値の111と上位１ＢＩＴ
を足して１１１１となる
$sabun = 68$
$ADPCM = 1200 - 68 = 1132$
となるので、3クロック目の数字になる。
ステップサイズのアドレスはエンコード出力が 1111 のために＋８アドレスが加算される。
その結果ステップサイズは８０となる。

このような計算を図５－２に載っている信号すべてについて計算してみると、確実に計算式の通りにＶＨＤＬが動作しているのが分かる。
５．３のＶＨＤＬの設計に解説している通りエンコーダが８ｂｉｔまで扱える様に設計しているために、データ長を８ｂｉｔに設計している。そのために上位４ｂｉｔが不定となる。

第7章 まとめ

卒業研究で、Visual BasicとVHDLのADPCMのエンコーダおよびデコーダを作成してみたが考えている以上に困難だった。まずVisual Basicではファイルの読み込みが10進数でなければならないという事と、ファイルの書きこみも10進数でなければならない事、16進数から10進数に変換すること等メインのADPCMのエンコーダやデコーダ以外の事に手間がかかり、VHDLについては、2進数で計算しなければならない事やオーバーフロー対策に苦労した。

色々工夫をしてみたが、プログラムはリアルタイムに動かなかった。これについては1秒間に44,100個もの符号化されたデータを取り扱わねばならない事と、Visual Basic で作成したプログラムではステレオに対応しているために、その倍88,200個もの符号化されたデータを扱わないといけないという問題点が有る。Visual Basic で変換作業をすると、アスロン 1.5Ghz のマシンで元ファイルの時間の、デコード作業やエンコード作業で20倍もの時間がかかってしまう。

VHDLについては、シミュレーションまで動作を確認したが、FPGA化するのはエラーでできなかった。これについてはメモリ関係が原因の可能性が大きい。

まだ考えられる改良点は、音の初めの無音部分と音の終わりの無音部分を圧縮することにより、効率の良い圧縮が可能である。圧縮法としてはランレングス方式を採用する。ランレングス方式とは、1が100回続けば 1×100 というような圧縮方法である。これを音の始めと終わりに採用すれば、ADPCMの圧縮率があげることが可能である。しかし問題点があり、Visual Basic であれば実装は容易であるが、VHDLの場合出力と入力のタイミングが合わなくなる可能性がある。しかし、バッファを置くことにより解決することが可能である。

音質面では、ADPCM の圧縮後の音は4 bit の場合は元のPCM 16bit の音に比べて違和感があまりなく、波形が元のPCM16 bit に比べほぼ同等のカーブになっており、下の図6－1の様な波形になる。

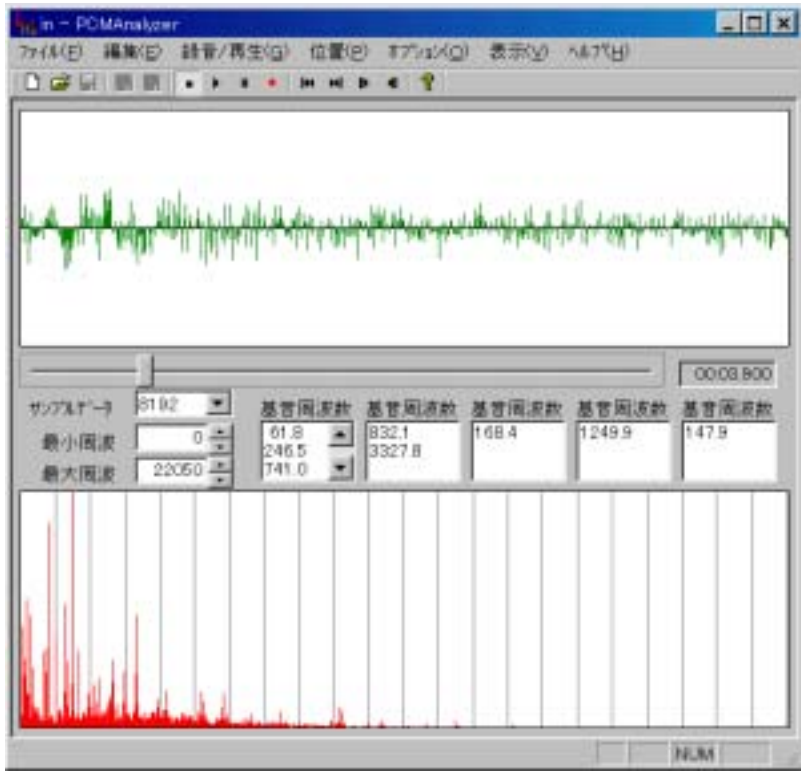

図6－1　無圧縮16 bit PCMの波形

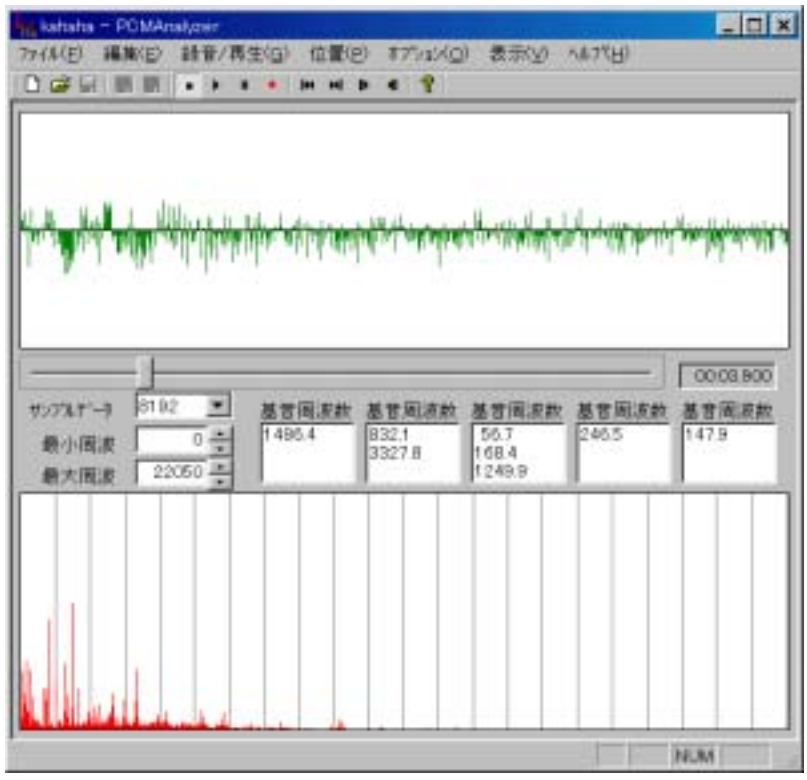

図6－2　ADPCM4 bit の波形

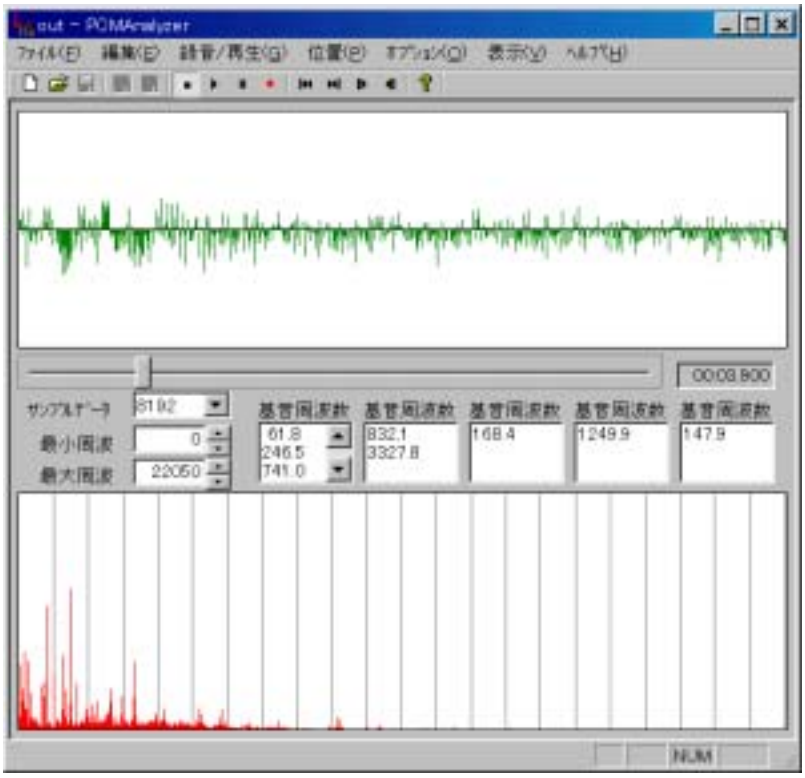

図6－3　ADPCM8 bit の波形

以上の6－1～6－3の3つの図を比べてみると、無圧縮のPCM16 bit の波形に比べ、ADPCMの8 bit で圧縮した波形は一部低音部分が再現されていないが、高音部分はほとんど変わらない。ADPCMの4 bit と8 bit で比較するとADPCM化する際にある程度損失があるためあまり信号では差がない。その結果 4bit 時のほうが圧縮率も高く効率がよい。次に3 bit の場合で検証した結果4 bit の時よりも、音が劣化しているが十分聞ける音質であった。

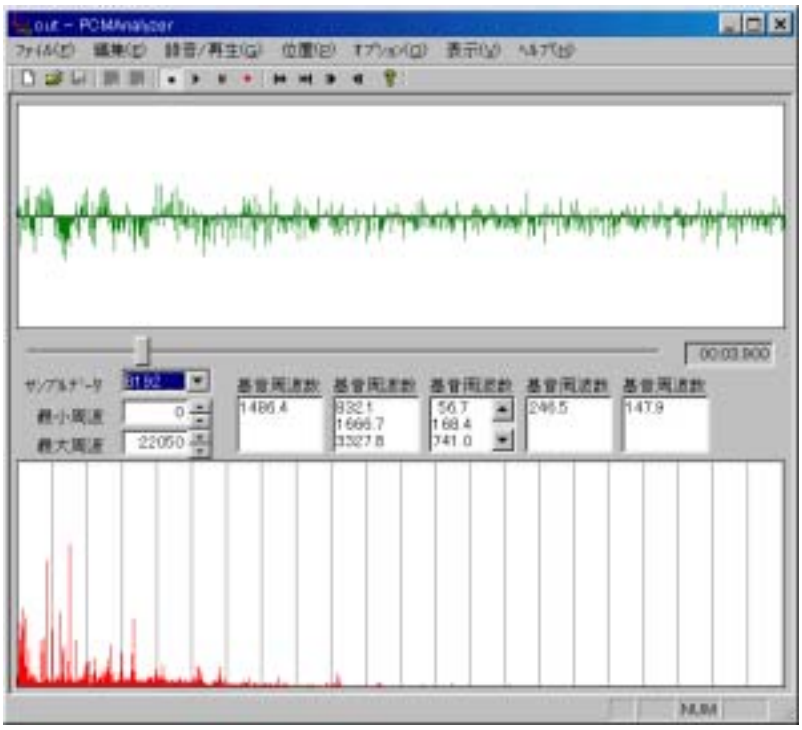

図6－4　ＡＤＰＣＭ3 bit の波形

謝辞

本研究の全過程を通じて懇切な指導をいただいた橘　昌良助教授、適宜指導をいただいた原　央教授、矢野政顯教授に厚く御礼申し上げます。

参考文献

[1] 「ＨＤＬによるＶＬＳＩ設計」 深山正幸　北川章夫　秋田純一　鈴木政国著
[2] 「ＶＨＤＬの基礎」Ｚ･ナビバ著/王一幸訳
[3] 「基礎からのデジタル信号処理」宇山靖政著

付録

ここでは、 Visual Basic のソースと VHDL のソースを載せる。まずVisualBasicのソースで
(General) (Declarations)

```
Option Explicit
Dim I As Long
Dim i1 As Integer
Dim stepsize(1 To 88, 1 To 8) As Integer
Dim SizeB(1 To 4) As Byte
Dim Size As Long
Dim WAVL(1 To 2) As Byte
Dim WAVR(1 To 2) As Byte
Dim WAVLTEN As Long
Dim WAVRTEN As Long
Dim decodeL As Long
Dim decodeR As Long
Dim tempL As Integer
Dim tempR As Integer
Dim sizesyutokuL As String
Dim sizesyutokuR As String
Dim syokiL(1 To 16) As Byte
Dim syokiR(1 To 16) As Byte
Dim syutuL As String
Dim syutuR As String
Dim diffL As Long
Dim diffR As Long
Dim stadrL As Integer
Dim stadrR As Integer
Dim ENCL(0 To 7) As Byte
Dim ENCR(0 To 7) As Byte
Dim Lsabun As Long
Dim Rsabun As Long
Dim OpenForms As Integer
Dim temp2L As String
```

```
Dim temp2R As String
Dim yomikomiL As String
Dim yomikomiR As String
Dim decodewavsize As Long
Dim decodesize As Long
Dim decsyutuL As Long
Dim decsyutuR As Long
Dim nagasa As Integer
Dim hexmaeL As Long
Dim hexmaeR As Long
Dim hexL As String
Dim hexR As String
Dim hexsize As String
Dim headhexsize As String
Dim DECL(0 To 7) As Byte
Dim DECR(0 To 7) As Byte
Dim hexsyutu As String
Dim hexyou As Byte
```

エンコーダ

```
syokiL(16) = 0
syokiR(16) = 0
stadrL = 1
stadrR = 1

For I = 1 To 88
   stepsize(I, 2) = stepsize(I, 1) ¥ Text3.Text
   stepsize(I, 3) = stepsize(I, 1) ¥ Text4.Text
   stepsize(I, 4) = stepsize(I, 1) ¥ Text5.Text
   stepsize(I, 5) = stepsize(I, 1) ¥ Text6.Text
   stepsize(I, 6) = stepsize(I, 1) ¥ Text7.Text
   stepsize(I, 7) = stepsize(I, 1) ¥ Text8.Text
   stepsize(I, 8) = stepsize(I, 1) ¥ Text9.Text
Next I
```

```
Open “in.wav” For Binary Access Read As #1
'wav データサイズ取得
    Get #1, 41, SizeB(1)
    Get #1, 42, SizeB(2)
    Get #1, 43, SizeB(3)
    Get #1, 44, SizeB(4)
Size = SizeB(1) + (SizeB(2) * 256) + SizeB(3) * 65536 + SizeB(4) * 16777216
Text2.Text = Size

For I = 45 To Size + 45 Step 4
'初期クリア
  Lsabun = 0
  Rsabun = 0
  For i1 = 0 To Combo1.Text - 1
      ENCL(i1) = 0: ENCR(i1) = 0
  Next i1

'カウンタ
      OpenForms = DoEvents
      Text1.Text = I

'データ読み込み

  Get #1, I, WAVL(1)
  Get #1, I + 1, WAVL(2)
  Get #1, I + 2, WAVR(1)
  Get #1, I + 3, WAVR(2)
'１０進データに変換
        sizesyutokuL = WAVL(2)
      sizesyutokuR = WAVR(2)
      WAVLTEN = (sizesyutokuL * 256) + WAVL(1)
      If WAVLTEN >= 32769 Then WAVLTEN = WAVLTEN - 65536: syokiL(16) = 1
      WAVRTEN = (sizesyutokuR * 256) + WAVR(2)
      If WAVRTEN >= 32769 Then WAVRTEN = WAVRTEN - 65536: syokiR(16) = 1
```

```
        If I = 45 Then
'初期値出力
            decodeL = WAVLTEN
            decodeR = WAVRTEN

            For i1 = 1 To 15
                If i1 = 1 Then tempL = Abs(WAVLTEN): tempR = Abs(WAVRTEN)
                syokiL(i1) = tempL Mod 2: tempL = tempL ¥ 2
                syokiR(i1) = tempR Mod 2: tempR = tempR ¥ 2
            Next i1

            For i1 = 16 To 1 Step -1
                syutuL = syutuL & syokiL(i1)
                syutuR = syutuR & syokiR(i1)
            Next i1

        Else
            diffL = WAVLTEN - decodeL
            diffR = WAVRTEN - decodeR
            If diffL < 0 Then ENCL(Combo1.Text - 1) = 1: diffL = diffL * -1
            If diffR < 0 Then ENCR(Combo1.Text - 1) = 1: diffR = diffR * -1

            For i1 = 1 To Combo1.Text - 1
                If diffL >= stepsize(stadrL, i1) Then ENCL(Combo1.Text - i1 - 1) = 1:
diffL = diffL - stepsize(stadrL, i1)
                If diffR >= stepsize(stadrR, i1) Then ENCR(Combo1.Text - i1 - 1) = 1:
diffR = diffR - stepsize(stadrR, i1)
            Next i1
    '出力
              For i1 = Combo1.Text - 1 To 0 Step -1
              syutuL = syutuL & ENCL(i1)
              syutuR = syutuR & ENCR(i1)
            Next i1
```

```
            For i1 = 1 To Combo1.Text - 1
                Lsabun = Lsabun + stepsize(stadrL, i1) * ENCL(Combo1.Text - i1 - 1)
                Rsabun = Rsabun + stepsize(stadrR, i1) * ENCR(Combo1.Text - i1 - 1)
            Next i1

            Lsabun = Lsabun + stepsize(stadrL, Combo1.Text)
            If ENCL(Combo1.Text - 1) = 1 Then Lsabun = Lsabun * -1
            Rsabun = Rsabun + stepsize(stadrL, Combo1.Text)
            If ENCR(Combo1.Text - 1) = 1 Then Rsabun = Rsabun * -1
            decodeL = decodeL + Lsabun
            decodeR = decodeR + Rsabun
            If decodeL > 37267 Then decodeL = 37267
            If decodeR > 37267 Then decodeR = 37267
            If decodeL < -37267 Then decodeL = -37267
            If decodeR < -37267 Then decodeR = -37267
'ステップサイズ計算
            If Combo1.Text = 3 Then

                If ENCL(Combo1.Text - 2) = 0 Then
                    stadrL = stadrL - 1
                Else
                    stadrL = stadrL + 2
                    If ENCL(Combo1.Text - 3) = 1 Then stadrL = stadrL + 2
                End If
            Else
                If ENCL(Combo1.Text - 2) = 0 Then
                    stadrL = stadrL - 1
                Else
                    stadrL = stadrL + 2
                    If ENCL(Combo1.Text - 3) = 1 Then stadrL = stadrL + 4
                    If ENCL(Combo1.Text - 4) = 1 Then stadrL = stadrL + 2
                End If
            End If
```

```
If stadrL < 1 Then stadrL = 1
If stadrL > 88 Then stadrL = 88

If Combo1.Text = 3 Then

        If ENCR(Combo1.Text - 2) = 0 Then
                stadrR = stadrR - 1
        Else
                stadrR = stadrR + 2
                If ENCR(Combo1.Text - 3) = 1 Then stadrR = stadrR + 2
        End If
Else
        If ENCR(Combo1.Text - 2) = 0 Then
                stadrR = stadrR - 1
        Else
                stadrR = stadrR + 2
                If ENCR(Combo1.Text - 3) = 1 Then stadrR = stadrR + 4
                If ENCR(Combo1.Text - 4) = 1 Then stadrR = stadrR + 2
        End If
End If
If stadrR < 1 Then stadrR = 1
If stadrR > 88 Then stadrR = 88
If I = 49 Then
Open "syutuL.txt" For Output As #2
Print #2, syutuL;
Close #2
syutuL = ""
Open "syutuR.txt" For Output As #2
Print #2, syutuR;
Close #2
syutuR = ""
If (I Mod 10000) - Combo1.Text < 0 Then
Open "syutuL.txt" For Append As #2
Print #2, syutuL;
```

```
                Close #2
                syutuL = ""
                Open "syutuR.txt" For Output As #2
                Print #2, syutuR;
                Close #2
                syutuR = ""
                End If

                If (I Mod 10000) - Combo1.Text < 0 Then
                Open "syutuL.txt" For Append As #2
                Print #2, syutuL;
                Close #2
                syutuL = ""
                Open "syutuR.txt" For Append As #2
                Print #2, syutuR;
                Close #2
                syutuR = ""
                End If


        End If

Next I
                Open "syutuL.txt" For Append As #2
                Print #2, syutuL;
                Close #2
                syutuL = ""
                Open "syutuR.txt" For Append As #2
                Print #2, syutuR;
                Close #2
                syutuR = ""
Close #1
End Sub
```

デコーダ

```
For I = 1 To 88
   stepsize(I, 2) = stepsize(I, 1) ¥ Text3.Text
   stepsize(I, 3) = stepsize(I, 1) ¥ Text4.Text
   stepsize(I, 4) = stepsize(I, 1) ¥ Text5.Text
   stepsize(I, 5) = stepsize(I, 1) ¥ Text6.Text
   stepsize(I, 6) = stepsize(I, 1) ¥ Text7.Text
   stepsize(I, 7) = stepsize(I, 1) ¥ Text8.Text
   stepsize(I, 8) = stepsize(I, 1) ¥ Text9.Text
Next I
yomikomiL = ""
yomikomiR = ""
Lsabun = 0
Rsabun = 0
decsyutuL = 0
decsyutuR = 0
stadrL = 1
stadrR = 1
decodesize = FileLen("syutuL.txt")
decodewavsize = (decodesize - 16 + Combo1.Text) * (4 / Combo1.Text)

Open “out.wav” For Binary Access Write As #3
Open "syutuL.txt" For Input As #1
yomikomiL = Input(16, #1)



Open "syutuR.txt" For Input As #2
yomikomiR = Input(16, #2)

For I = 2 To 16
  decsyutuL = decsyutuL + Mid(yomikomiL, I, 1) * (65536 / (2 ^ I))
Next I
If Mid(yomikomiL, 1, 1) = 1 Then decsyutuL = decsyutuL * -1
```

```
For I = 2 To 16
  decsyutuR = decsyutuR + Mid(yomikomiR, I, 1) * (65536 / (2 ^ I))
Next I

If Mid(yomikomiR, 1, 1) = 1 Then decsyutuR = decsyutuR * -1

'初期値ＨＥＸ出力

If decsyutuL < 0 Then hexmaeL = 65536 + decsyutuL Else hexmaeL = decsyutuL
If decsyutuR < 0 Then hexmaeR = 65536 + decsyutuR Else hexmaeR = decsyutuR

hexL = Hex(hexmaeL)
hexR = Hex(hexmaeR)
'ビット長を合わせる
For I = 1 To 4
  nagasa = Len(hexL)
  If nagasa = 4 Then Else hexL = 0 & hexL
Next I

For I = 1 To 4
  nagasa = Len(hexR)
  If nagasa = 4 Then Else hexR = 0 & hexR
Next I

'ビットオーダを合わせる
hexL = Mid(hexL, 3, 2) & Mid(hexL, 1, 2)
hexR = Mid(hexR, 3, 2) & Mid(hexR, 1, 2)

Text2.Text = decodewavsize * (Text1.Text / 4)
hexsize = Hex(decodewavsize)
headhexsize = Hex(decodewavsize + 36)
For I = 1 To 8
  nagasa = Len(hexsize)
  If nagasa = 8 Then Else hexsize = 0 & hexsize
Next I
```

```
hexsize = Mid(hexsize, 7, 2) & Mid(hexsize, 5, 2) & Mid(hexsize, 3, 2) & Mid(hexsize, 1, 2)
For I = 1 To 8
  nagasa = Len(headhexsize)
  If nagasa = 8 Then Else headhexsize = 0 & headhexsize
Next I
headhexsize = Mid(headhexsize, 7, 2) & Mid(headhexsize, 5, 2) & Mid(headhexsize, 3, 2) &
Mid(headhexsize, 1, 2)




hexsyutu = "52494646" & headhexsize &
"57415645666D74201000000001000020044AC000010B10200040010006461746 1" & hexsize &
hexL & hexR

For I = 17 To decodesize Step Combo1.Text
Lsabun = 0
Rsabun = 0
OpenForms = DoEvents
Text1.Text = I
yomikomiL = Input(Combo1.Text, #1)
yomikomiR = Input(Combo1.Text, #2)

      For i1 = Combo1.Text - 1 To 0 Step -1

            DECL(i1) = Mid(yomikomiL, Combo1.Text - i1, 1)
            DECR(i1) = Mid(yomikomiR, Combo1.Text - i1, 1)

      Next i1

      For i1 = 1 To Combo1.Text - 1
            Lsabun = Lsabun + stepsize(stadrL, i1) * DECL(Combo1.Text - i1 - 1)
            Rsabun = Rsabun + stepsize(stadrR, i1) * DECR(Combo1.Text - i1 - 1)
      Next i1
```

```
    Lsabun = Lsabun + stepsize(stadrL, Combo1.Text)
    If DECL(Combo1.Text - 1) = 1 Then Lsabun = Lsabun * -1
    Rsabun = Rsabun + stepsize(stadrL, Combo1.Text)
    If DECR(Combo1.Text - 1) = 1 Then Rsabun = Rsabun * -1

    decsyutuL = decsyutuL + Lsabun
    decsyutuR = decsyutuR + Rsabun

'オーバーフロー対策
    If decsyutuL > 37267 Then decsyutuL = 37267
    If decsyutuL > -37267 Then decsyutuL = -37267
    If decsyutuR < 37267 Then decsyutuR = 37267
    If decsyutuR < -37267 Then decsyutuR = -37267

    If Combo1.Text = 3 Then
            If DECL(Combo1.Text - 2) = 0 Then

                stadrL = stadrL - 1
            Else
                stadrL = stadrL + 2
                If DECL(Combo1.Text - 3) = 1 Then stadrL = stadrL + 2
            End If
    Else

        If DECL(Combo1.Text - 2) = 0 Then

                stadrL = stadrL - 1
        Else

                stadrL = stadrL + 2
                If DECL(Combo1.Text - 3) = 1 Then stadrL = stadrL + 4
                If DECL(Combo1.Text - 4) = 1 Then stadrL = stadrL + 2
        End If
    End If
```

```
    If stadrL < 1 Then stadrL = 1
    If stadrL > 88 Then stadrL = 88

    If Combo1.Text = 3 Then
            If DECR(Combo1.Text - 2) = 0 Then

                stadrR = stadrR - 1
            Else
                stadrR = stadrR + 2
                If DECR(Combo1.Text - 3) = 1 Then stadrR = stadrR + 2
            End If
    Else

        If DECR(Combo1.Text - 2) = 0 Then

                stadrR = stadrR - 1

        Else

                stadrR = stadrR + 2
                If DECR(Combo1.Text - 3) = 1 Then stadrR = stadrR + 4
                If DECR(Combo1.Text - 4) = 1 Then stadrR = stadrR + 2

        End If
    End If
    If stadrR < 1 Then stadrR = 1
    If stadrR > 88 Then stadrR = 88
'出力
    If decsyutuL < 0 Then hexmaeL = 65536 + decsyutuL Else hexmaeL = decsyutuL
    If decsyutuR < 0 Then hexmaeR = 65536 + decsyutuR Else hexmaeR = decsyutuR

    hexL = Hex(hexmaeL)
    hexR = Hex(hexmaeR)
```

```
    For i1 = 1 To 4
            nagasa = Len(hexL)
            If nagasa = 4 Then Else hexL = 0 & hexL
    Next i1

    For i1 = 1 To 4
            nagasa = Len(hexR)
            If nagasa = 4 Then Else hexR = 0 & hexR
    Next i1

'ビットオーダを合わせる

    hexL = Mid(hexL, 3, 2) & Mid(hexL, 1, 2)
    hexR = Mid(hexR, 3, 2) & Mid(hexR, 1, 2)
    hexsyutu = hexsyutu & hexL & hexR

'ファイルの出力

    If I Mod 10000 - Combo1.Text < 0 Then
        For i1 = 1 To Len(hexsyutu) Step 2
            hexyou = "&H" & Mid(hexsyutu, i1, 2)
            Put #3, , hexyou
        Next i1
        hexsyutu = ""
    End If
Next I
    For i1 = 1 To Len(hexsyutu) Step 2
        hexyou = "&H" & Mid(hexsyutu, i1, 2)
        Put #3, , hexyou
    Next i1
Close #1
Close #2
Close #3
End Sub
```

Form Load

```
stepsize(1, 1) = 8
stepsize(2, 1) = 9
stepsize(3, 1) = 10
stepsize(4, 1) = 11
stepsize(5, 1) = 12
stepsize(6, 1) = 13
stepsize(7, 1) = 14
stepsize(8, 1) = 16
stepsize(9, 1) = 17
stepsize(10, 1) = 19
stepsize(11, 1) = 21
stepsize(12, 1) = 23
stepsize(13, 1) = 25
stepsize(14, 1) = 28
stepsize(15, 1) = 31
stepsize(16, 1) = 34
stepsize(17, 1) = 37
stepsize(18, 1) = 41
stepsize(19, 1) = 45
stepsize(20, 1) = 50
stepsize(21, 1) = 55
stepsize(22, 1) = 60
stepsize(23, 1) = 66
stepsize(24, 1) = 73
stepsize(25, 1) = 80
stepsize(26, 1) = 88
stepsize(27, 1) = 97
stepsize(28, 1) = 107
stepsize(29, 1) = 118
stepsize(30, 1) = 130
stepsize(31, 1) = 143
stepsize(32, 1) = 157
stepsize(33, 1) = 173
```

stepsize(34, 1) = 190
stepsize(35, 1) = 209
stepsize(36, 1) = 230
stepsize(37, 1) = 253
stepsize(38, 1) = 279
stepsize(39, 1) = 307
stepsize(40, 1) = 337
stepsize(41, 1) = 371
stepsize(42, 1) = 408
stepsize(43, 1) = 449
stepsize(44, 1) = 494
stepsize(45, 1) = 544
stepsize(46, 1) = 598
stepsize(47, 1) = 658
stepsize(48, 1) = 724
stepsize(49, 1) = 796
stepsize(50, 1) = 876
stepsize(51, 1) = 963
stepsize(52, 1) = 1060
stepsize(53, 1) = 1166
stepsize(54, 1) = 1282
stepsize(55, 1) = 1411
stepsize(56, 1) = 1552
stepsize(57, 1) = 1707
stepsize(58, 1) = 1878
stepsize(59, 1) = 2066
stepsize(60, 1) = 2272
stepsize(61, 1) = 2499
stepsize(62, 1) = 2749
stepsize(63, 1) = 3024
stepsize(64, 1) = 3327
stepsize(65, 1) = 3660
stepsize(66, 1) = 4026
stepsize(67, 1) = 4428

```
stepsize(68, 1) = 4871
stepsize(69, 1) = 5358
stepsize(70, 1) = 5894
stepsize(71, 1) = 6484
stepsize(72, 1) = 7132
stepsize(73, 1) = 7845
stepsize(74, 1) = 8630
stepsize(75, 1) = 9493
stepsize(76, 1) = 10442
stepsize(77, 1) = 11487
stepsize(78, 1) = 12635
stepsize(79, 1) = 13899
stepsize(80, 1) = 15289
stepsize(81, 1) = 16818
stepsize(82, 1) = 18500
stepsize(83, 1) = 20350
stepsize(84, 1) = 22385
stepsize(85, 1) = 24623
stepsize(86, 1) = 27086
stepsize(87, 1) = 29794
stepsize(88, 1) = 32767
End Sub
```

VHDLのエンコーダのソース

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity encoder is
    port(CLK : in std_logic;
              X    : in std_logic_vector(15 downto 0);
         ECD : out std_logic_vector(7 downto 0);
                   syoki : in std_logic_vector(15 downto 0)
);

end encoder;

architecture Behavioral of encoder is
signal Y: std_logic_vector(7 downto 0);
signal unX: std_logic_vector(15 downto 0);
subtype STEPU is std_logic_vector(15 downto 0);
type STEPSZ is array (0 to 87) of STEPU;
constant N :integer :=4 ;
constant STEPSIZE : STEPSZ := (
"0000000000001000",
"0000000000001001",
"0000000000001010",
"0000000000001011",
"0000000000001100",
"0000000000001101",
"0000000000001110",
"0000000000010000",
"0000000000010001",
"0000000000010011",
"0000000000010101",
"0000000000010111",
"0000000000011001",
```

"0000000000011100",
"0000000000011111",
"0000000000100010",
"0000000000100101",
"0000000000101001",
"0000000000101101",
"0000000000110010",
"0000000000110111",
"0000000000111100",
"0000000001000010",
"0000000001001001",
"0000000001010000",
"0000000001011000",
"0000000001100001",
"0000000001101011",
"0000000001110110",
"0000000010000010",
"0000000010001111",
"0000000010011101",
"0000000010101101",
"0000000010111110",
"0000000011010001",
"0000000011100110",
"0000000011111101",
"0000000100010111",
"0000000100110011",
"0000000101010001",
"0000000101110011",
"0000000110011000",
"0000000111000001",
"0000000111101110",
"0000001000100000",
"0000001001010110",
"0000001010010010",

"0000001011010100",
"0000001100011100",
"0000001101101100",
"0000001111000011",
"0000010000100100",
"0000010010001110",
"0000010100000010",
"0000010110000011",
"0000011000010000",
"0000011010101011",
"0000011101010110",
"0000100000010010",
"0000100011100000",
"0000100111000011",
"0000101010111101",
"0000101111010000",
"0000110011111111",
"0000111001001100",
"0000111110111010",
"0001000101001100",
"0001001100000111",
"0001010011101110",
"0001011100000110",
"0001100101010100",
"0001101111011100",
"0001111010100101",
"0010000110110110",
"0010010100010101",
"0010100011001010",
"0010110011011111",
"0011000101011011",
"0011011001001011",
"0011101110111001",
"0100000110110010",
"0100100001000100",

```
"0100111101111110",
"0101011101110001",
"0110000000101111",
"0110100111001110",
"0111010001100010",
"0111111111111111"
                        );

begin
    process (CLK,X)
variable   SAB: std_logic_vector(15 downto 0);
variable   K:integer range -1 to 8;
variable L: std_logic_vector(7 downto 0);
variable STEP: std_logic_vector(15 downto 0);
variable HANTEI: std_logic_vector(16 downto 0);
variable ZERO: std_logic_vector(7 downto 0):="00000000";
variable D: std_logic_vector(15 downto 0);
variable STEPADR : integer range 0 to 87 :=0;
variable   GENZAI: std_logic_vector(15 downto 0):="1000000000000000";
variable TEMP:std_logic:='0';
begin
if(CLK'event and CLK='1') then
  if (TEMP='0') then
       GENZAI := syoki;
       TEMP :='1';
  end if;
  STEP :=STEPSIZE(STEPADR);
  SAB :="0000000000000000";
  D:= (not X(15) & X(14 downto 0));
  HANTEI := ('1' & D) - ('0' & GENZAI);
  if (HANTEI(16)='1') then
     D := D - GENZAI;
     L(N-1) := '0';
  else
```

```
            D := GENZAI - D;
            L(N-1) := '1';
end if;
for I in 2 to N loop
     HANTEI := ('1' & D) - (ZERO(I-2 downto 0) & STEP(15 downto I-2));
   if (HANTEI(16) ='1') then
                L(N-I) := '1';
                D := HANTEI(15 downto 0);
       else
                L(N-I) := '0';
   end if;
end loop;

for I in 2 to N loop
        if (L(N-I)='1') then
              if (I= 2) then
                        SAB := STEP;
              else
                        SAB := SAB + (ZERO(I-3 downto 0) & STEP ( 15 downto I-2));
              end if;
        end if;
   end loop;
   SAB := SAB + ("000" & STEP (15 downto 3));


                if (L(N-1)='0') then
                      HANTEI := ('0' & GENZAI) + ('0' & SAB);
                          if (HANTEI(16)='1') then
                                  GENZAI := "1111111111111111";
                              else
                                  GENZAI := GENZAI + SAB;
                          end if;
        elsif (L(N-1)='1') then
                      HANTEI := ('1' & GENZAI) - ('0' & SAB);
```

```
                        if (HANTEI(16)='1') then
                            GENZAI := GENZAI - SAB;
                        else
                            GENZAI := "0000000000000000";
                        end if;
                end if;
    if (N=3) then
        case L((N-2) downto (N-3)) is
                when "10" => K := 2;
                when "11" => K := 4;
                when others => K := -1;
                end case;
    else
        case L((N-2) downto (N-4)) is
          when "100" => K := 2;
        when "101" => K := 4;
          when "110" => K := 6;
          when "111" => K := 8;
          when others => K := -1;
          end case;
    end if;
    if (STEPADR + K > 87) then
            STEPADR := 87;
    elsif (STEPADR +K < 0) then
            STEPADR := 0;
   else
            STEPADR := STEPADR + K;
     end if;
Y <= L;
end if;
end process;
unX <= (not X(15) & X(14 downto 0));
ECD <= Y ;
end Behavioral;
```

ＶＨＤＬのデコーダのソース

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity ADPCMD is
    port(CLK : in std_logic;
              A : in std_logic_vector(7 downto 0);
              X : out std_logic_vector(15 downto 0);
              syoki : in std_logic_vector(15 downto 0)
);
end ADPCMD;
architecture RTL of ADPCMD is
subtype STEPU is std_logic_vector(15 downto 0);
type STEPSZ is array (0 to 87) of STEPU;
signal    Y: std_logic_vector(15 downto 0);
constant N :integer :=4 ;
constant STEPSIZE : STEPSZ := (
"0000000000001000",
"0000000000001001",
"0000000000001010",
"0000000000001011",
"0000000000001100",
"0000000000001101",
"0000000000001110",
"0000000000010000",
"0000000000010001",
"0000000000010011",
"0000000000010101",
"0000000000010111",
"0000000000011001",
"0000000000011100",
"0000000000011111",
"0000000000100010",
"0000000000100101",
```

"0000000000101001",
"0000000000101101",
"0000000000110010",
"0000000000110111",
"0000000000111100",
"0000000001000010",
"0000000001001001",
"0000000001010000",
"0000000001011000",
"0000000001100001",
"0000000001101011",
"0000000001110110",
"0000000010000010",
"0000000010001111",
"0000000010011101",
"0000000010101101",
"0000000010111110",
"0000000011010001",
"0000000011100110",
"0000000011111101",
"0000000100010111",
"0000000100110011",
"0000000101010001",
"0000000101110011",
"0000000110011000",
"0000000111000001",
"0000000111101110",
"0000001000100000",
"0000001001010110",
"0000001010010010",
"0000001011010100",
"0000001100011100",
"0000001101101100",
"0000001111000011",

"0000010000100100",
"0000010010001110",
"0000010100000010",
"0000010110000011",
"0000011000010000",
"0000011010101011",
"0000011101010110",
"0000100000010010",
"0000100011100000",
"0000100111000011",
"0000101010111101",
"0000101111010000",
"0000110011111111",
"0000111001001100",
"0000111110111010",
"0001000101001100",
"0001001100000111",
"0001010011101110",
"0001011100000110",
"0001100101010100",
"0001101111011100",
"0001111010100101",
"0010000110110110",
"0010010100010101",
"0010100011001010",
"0010110011011111",
"0011000101011011",
"0011011001001011",
"0011101110111001",
"0100000110110010",
"0100100001000100",
"0100111101111110",
"0101011101110001",
"0110000000101111",

```
"0110100111001110",
"0111010001100010",
"0111111111111111"
                        );
begin
    process (CLK,A)
variable ZERO: std_logic_vector(7 downto 0):="00000000";
variable   SAB: std_logic_vector(15 downto 0);
variable   STEP: std_logic_vector(15 downto 0);
variable STEPADR : integer range 0 to 87 :=0;
variable   HANTEI: std_logic_vector(16 downto 0);
variable   GENZAI: std_logic_vector(15 downto 0):=”1000000000000000”;
variable   K:integer range -1 to 8;
variable TEMP:std_logic:='0';
begin
  if(CLK'event and CLK='1') then
  if (TEMP='0') then
      GENZAI := syoki;
      TEMP :='1';
  end if;
      SAB :="0000000000000000";
      STEP :=STEPSIZE(STEPADR);
      for I in 2 to N loop
        if (A(N-I)='1') then
            if (I= 2) then
                    SAB := STEP;
            else
                    SAB := SAB + (ZERO(I-3 downto 0) & STEP ( 15 downto I-2));
            end if;
        end if;
    end loop;
    SAB := SAB + ("000" & STEP (15 downto 3));
                if (A(N-1)='0') then
                    HANTEI := ('0' & GENZAI) + ('0' & SAB);
                    if (HANTEI(16)='1') then
```

```
                    GENZAI := "1111111111111111";
                else
                    GENZAI := GENZAI + SAB;
                end if;
        elsif (A(N-1)='1') then
                HANTEI := ('1' & GENZAI) - ('0' & SAB);
                if (HANTEI(16)='1') then
                    GENZAI := GENZAI - SAB;
                else
                    GENZAI := "0000000000000000";
                end if;
        end if;
    if (N=3) then
        case A((N-2) downto (N-3)) is
            when "10" => K := 2;
            when "11" => K := 4;
            when others => K := -1;
            end case;
    else
        case A((N-2) downto (N-4)) is
         when "100" => K := 2;
        when "101" => K := 4;
         when "110" => K := 6;
         when "111" => K := 8;
         when others => K := -1;
         end case;
    end if;
          if (STEPADR + K > 87) then
              STEPADR := 87;
          elsif (STEPADR +K < 0) then
              STEPADR := 0;
          else
              STEPADR := STEPADR + K;
          end if;
```

```
                Y <= GENZAI;
  end if;

end process;

X <= Y;
end RTL;
```

ＶＨＤＬのエンコーダのテストベンチ

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
use IEEE.std_logic_unsigned.all;
ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

        COMPONENT encoder
        PORT(
                CLK : IN std_logic;
                X : IN std_logic_vector(15 downto 0);
       syoki : in std_logic_vector(15 downto 0);
                ECD : OUT std_logic_vector(7 downto 0)
                );
        END COMPONENT;
        SIGNAL CLK :    std_logic:='0';
        SIGNAL X :    std_logic_vector(15 downto 0):="0000000000000000";
        SIGNAL ECD :    std_logic_vector(7 downto 0);
        SIGNAL syoki : std_logic_vector(15 downto 0);
BEGIN
        uut: encoder PORT MAP(
                CLK => CLK,
                X => X,
                ECD => ECD,
                syoki => syoki
        );

process begin
syoki <="0000010011011110";
  for I in 1 to 1000 loop
    CLK <= '0'; wait for 50 ps;
```

```
   CLK <= '1'; wait for 50 ps;
  end loop;
  wait;
  end process;

     tb : PROCESS
     BEGIN
X <="1000001000001000";
wait for 100 ps;
X <="1000000101001001";
wait for 100 ps;
X <="1000000100100111";
wait for 100 ps;
X <="1000000000100101";
wait for 100 ps;
X <="1000000001110111";
wait for 100 ps;
X <="1000000110100101";
wait for 100 ps;
X <="1000000110111100";
wait for 100 ps;
X <="1000000011001100";
wait for 100 ps;
X <="1000001001111011";
wait for 100 ps;
X <="1000011110010001";
wait for 100 ps;
X <="1000100111011010";
wait for 100 ps;
X <="1000100010010011";
wait for 100 ps;
X <="1000011011000101";
wait for 100 ps;
X <="1000010101010110";
```

```
wait for 100 ps;
X <="1000010010100000";
wait for 100 ps;
X <="1000010010100011";
wait for 100 ps;
X <="1000010000101000";
wait for 100 ps;
X <="1000010000010000";
wait for 100 ps;
X <="1000010101001111";
wait for 100 ps;
X <="1000010101111110";
wait for 100 ps;
X <="1000001000101010";
wait for 100 ps;
X <="1000010000101110";
wait for 100 ps;
X <="1000101000100110";
wait for 100 ps;
X <="1000101111111001";
wait for 100 ps;
X <="1000100111111010";
wait for 100 ps;
X <="1000011011101011";
wait for 100 ps;
X <="1000001100100100";
wait for 100 ps;
X <="1000000101011001";
wait for 100 ps;
X <="1000010001110100";
wait for 100 ps;
X <="1000010010110011";
wait for 100 ps;
```

```
    END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

VHDL のデコーダのテストベンチ

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
use IEEE.std_logic_unsigned.all;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

        COMPONENT adpcmd
        PORT(
                CLK : IN std_logic;
                A : IN std_logic_vector(7 downto 0);
                X : OUT std_logic_vector(15 downto 0);
                syoki : IN std_logic_vector(15 downto 0)
                );
        END COMPONENT;
        SIGNAL CLK :    std_logic:= '0';
        SIGNAL A :    std_logic_vector(7 downto 0) :="00000000";
        SIGNAL X :    std_logic_vector(15 downto 0);
        SIGNAL syoki :    std_logic_vector(15 downto 0) :="1000000000000000";

BEGIN
        uut: adpcmd PORT MAP(
                CLK => CLK,
                A => A,
                X => X,
       syoki => syoki
                );
process begin
  for I in 1 to 10000 loop
    CLK <= '0'; wait for 50 ps;
```

```
   CLK <= '1'; wait for 50 ps;
  end loop;
  wait;
  end process;
-- *** Test Bench - User Defined Section ***
      tb : PROCESS
      BEGIN

A <="00000000" ;
wait for 100 ps;
A <="00001000" ;
wait for 100 ps;
A <="00000000" ;
wait for 100 ps;
A <="00001000" ;
wait for 100 ps;
A <="00000000" ;
wait for 100 ps;
A <="00001000" ;
wait for 100 ps;
A <="00000000" ;
wait for 100 ps;
A <="00001000" ;
wait for 100 ps;
A <="00000000" ;
wait for 100 ps;
A <="00001000" ;
wait for 100 ps;
A <="00000000" ;
wait for 100 ps;
A <="00001000" ;
wait for 100 ps;
A <="00000000" ;
wait for 100 ps;
```

```
end process;
END;
```