テストベンチを含めた磁気ディスク信号処理回路のVHDL記述とシミュレーション

谷口　研二[*1]・山田　貴裕[*1]・中司　賢一[*2]

A VHDL Simulation of Magnetic Disk Signal Processing Circuits

Kenji TANIGUCHI[*1], Takahiro YAMADA[*1] and Kenichi NAKASHI[*2]

Abstract:

Operation of Viterbi decoder applied to magnetic disk under random bit pattern signals and random noises is described and has been successfully simulated in VHDL. Besides digital functions, analog functions have been realized utilizing arithmetic functions of VHDL, performing almost equally to its conventional C-language counterpart, with an ease of adaptation towards hardware implementation.

Key Words: Magnetic disk drive, Viterbi decoder, VHDL, Simulation, Signal processing

1．はじめに

近年における磁気ディスク等の記録密度の向上や記録容量の大形化には著しいものがあり，今後の発展も期待されている。

本研究では，磁気記録再生系を大幅に変えずに信号の処理により記録密度を1.2～1.5倍に高めていく技術として注目されているPRML（Partial Response Maximum Likelihood）方式を使ったビタビ回路について着目する。ビタビ復号回路は，すでに磁気ディスク用として実用化されているが，より高度な回路の検討が今後も求められていくと見られるほか，この技術が将来のVLSIなどの高密度信号処理系へ適用されるものと期待されている。

磁気ディスク信号処理系はアナログディジタル信号処理系であること，回路規模も数千ゲートに及び本研究室で今後研究対象として取り上げる電子システムの研究の一環としても格好の題材となると考えられる。

本研究は当初九州大学を中心に行ってきたものであるが昨年度から本学で，特に設計記述言語VHDLを用いた検討を取り上げてきた。

すでに，独自の差分演算方式ビタビ回路[1), 3), 4)]，C言語によるテストベンチを含めた磁気ディスク信号処理系[2), 5), 6)]，VerilogHDLによるビタビ回路本体の記述とシミュレーション[1)]などが行われてきたが，本研究では，これらを基に，上記C言語系に相当する高度なテストベンチを持つVHDLやVerilogHDL記述ビタビ復号回路の構成とシミュレーションの検討を行いその手法を確立することを狙っている。

本研究では以下の3つの達成を目的とする。

1）ビタビ復号回路本体をVHDL記述で構築し，シミュレーションする技術の確立を図る。

2）ビタビ復号回路本体をテストするテストベンチもVHDLで構築する。

3）上記の系をウインドウズPCやUNIX WS上でシミュレーションする技術を確立する。

2．ビタビ復号回路について

ビタビ復号回路は，磁気記録パターンに対応して再生すべき信号と実際再生された信号のユークリッド距離の自乗の累積和といった確率の指数が最小のものを残すこ

[*1] 電子情報工学科　　[*2] 九州大学
平成13年9月28日受理

とを特徴とする回路である。この回路のユークリッド距離の計算部分は従来乗算器が用いられてきたが，乗算器の代わりに差分演算器を用いる方式を使うことで，回路の高速化，小形化を検討してきている[1)~4)]。

ビタビ復号はデータ間に相関が存在する場合，その相関を利用して最も確からしい系列を選択することによりデータを検出する方法で，再生信号が雑音を含む場合に相対的に誤りを減らすことができる。データ間に既知の相関があると，理想的には再生波形は限られたパターンしかとらない。雑音を含む実際の波形はこのパターンからはずれることがあるが，入力信号と相関から推測される値との誤差が常に最小となるように信号系列を選択することで最も確からしいデータ系列を見つけることができる。

もっとも簡単な，2状態のビタビ復号では，データ系列に1-Dの相関，つまり現在の値から1つ前の時間における値を引いた値が入力として入ってくる。ここで2状態というのは存在確率を求めるために残す状態の数のことである。この相関を持たせるフィルタ関数を以下ではPR functionと表現している。

ビタビ回路のVHDL記述は図1のように本体のViterbiとその構成要素である組み込み部品のbitshift, addsub, AND10, shiftregister, newand2, newxor2, invertからなる。これらの各部品はentity-architecture対で構成され，Parts.vhdファイルに収容する。以下にViterbi本体のVerilogHDL記述を参考にVHDL記述を行った。リスト1にそのentityとarchitectureの一部分を示す。

なお“－”に続く行はコメントを表す。－はハイフン2個が続くものであるが印刷上一つの文字に見えている。

3．ビタビ復号回路のテストベンチ

(1) ブロック図

図1は，このように構成した差分演算を利用したビタビ復号回路の構成を，これをテストするテストベンチを含めて描いたブロック図である。

磁気ディスクの書き込みデータに対応するランダムビットデータをMPSS部で発生し，それをPR functionへ導く。

PR functionでは最初のビットデータVA1を1と定め，次のビットデータをVA2とし，VA1-VA2をVAとし，2ビット目以降は，VA1にVA2を代入することによって1-Dの相関を作っている。

この信号に対してランダムなアナログ雑音を加算するために雑音発生器Noisegeneratorを設けている。

リスト1　ビタビ復号回路のVHDL記述

```
－ 以下は使用ライブラリの宣言
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use work.all;

－ 回路名entityとその入出力ポートの定義
entity viterbi is
    port (x: in std_logic_vector(5 downto 0);
          ck: in std_logic;
          acc: in std_logic_vector(9 downto 0);
          out1,out2: out std_logic;
          accin: out std_logic_vector(9 downto 0)
          dctl1,dctl2: inout std_logic;
          reg1,reg2: inout std_logic_vector(19 downto 0)
    );
－ 上記でreg1, reg2は下位の部品であるシフト
－ レジスタ内の信号で,
－ viterbiからこれを見るために設けたポート
end viterbi;

－ entityの実現方法
architecture VHDL of viterbi is
  signal ref:std_logic_vector(9 downto 0):="0000010000";
  signal in1,add1,sub1,ctl1,ctl2,ctl3,ctl4,sel1,sel2:
          std_logic_vector(9 downto 0);
  signal s1msb,s2msb,xorout,dctl1,dctl2,notxorout,vain:
          std_logic;
  signal ONE:std_logic:='1';
  signal ZERO:std_logic:='0';
  component bitshift
        port(x:in std_logic_vector(5 downto 0);
              in1:out std_logic_vector(9 downto 0));
  end component;
－ 以下同様のcomponent宣言
begin
  －使用部品の組み込み，即ちインスタンス化。
  －これらの組み込みは組み合わせ論理回路と
  －同様に並列動作となる。
   bs: bitshift port map(x,in1);
   as1: addsub port map(ZERO,in1,ref,add1);
－ 以下同様
－ 以下の各プロセスは順序動作となるが,
－ 複数個のプロセスは並列に動作する。
viterbi: process
begin
－ 定数の代入
ref<= "0000001000";
  wait;  －　シミュレーション中有効とする
         －ために必要
end process;
initial: process
begin
wait for 2 ns;  －　Viterbiプロセスが有限の時間
     －に終ることを規定するために必要な記述
end process;
end VHDL;
```

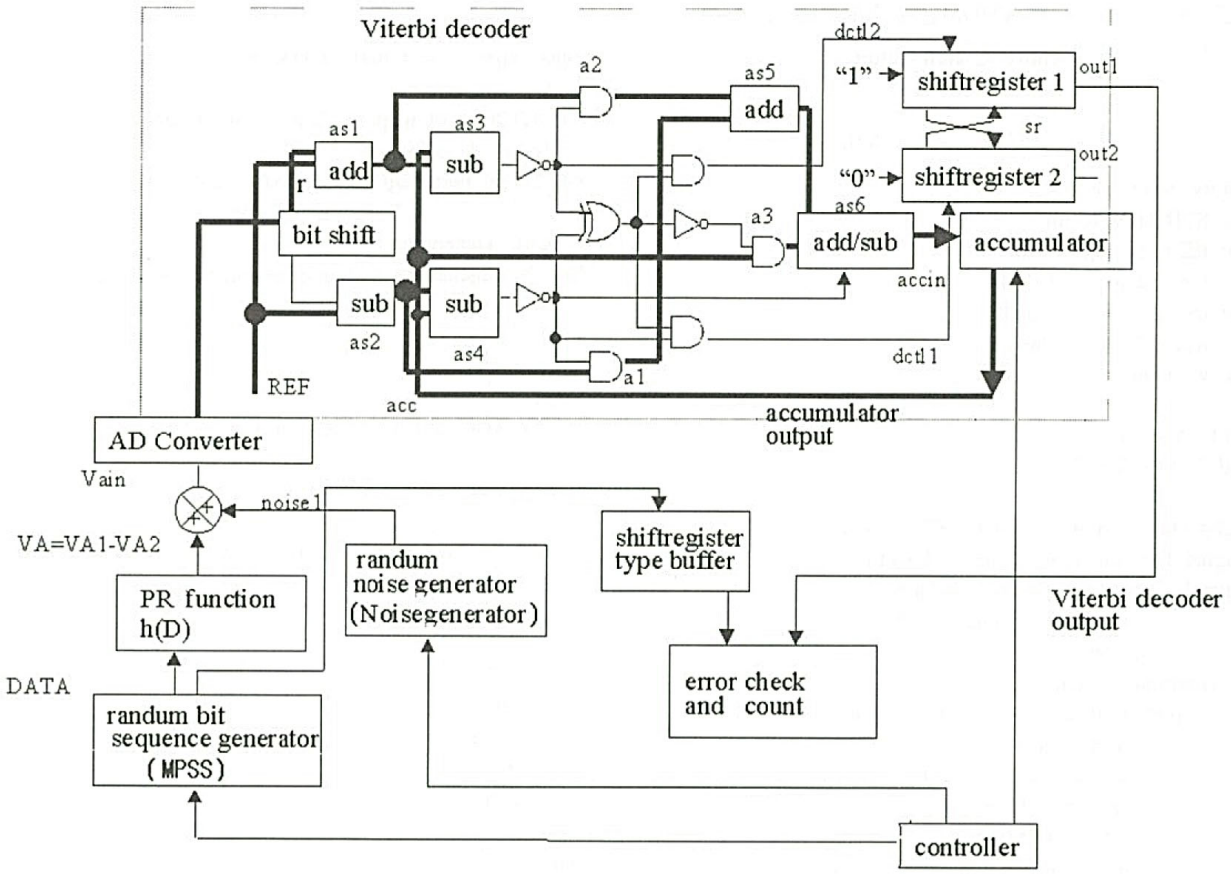

図1　ビタビ復号回路とテストベンチンチのブロック図

Error Check and Count では，シフトレジスタの20ビット目の値とこれに対応する磁気ディスク信号としての入力データとを比較して，エラーナンバーの回数をカウントして所定の回数だけ計算を行うようにする。実際には VHDL の代入に伴う遅れに対応して21ビット目と比較する。

本研究では，VHDL 記述とシミュレーションを Interactive Image Technology 社の multiSIM に備わっている VHDL シミュレーション機能を用い，アナログ部分については multiSIM のアナログシミュレーション機能を用いることを当初想定していたが，検討の結果アナログ部分についても VHDL 記述で行う方法を案出することができた。即ち，図1に示した Noisegenerator や AD 変換などのテストベンチを含めて VHDL シミュレーションを行うことにした。これに相当する機能は従来 C 言語プログラムで実現されていたが[2), 5)]，VHDL 記述では独自の工夫が必要となった。

ビタビ本体を含め，VHDL 記述とシミュレーションで考慮した事項を表1に示す。

表1　VHDL シミュレーションでの考慮事項－

1) 素機能を entity-architecture，function，procedure で記述する。
2) 信号（global）と変数（local）を適宜定義し，使い分ける。
3) 同時処理（コンカレンシー）と順序処理を使い分ける。
   2種類の代入の使い分け，process におけるセンシティビティリストと wait 文の関係，clock の位相等に注意する。
4) Parts の組み込みなど階層構造はインスタンスで表現する。
5) アナログ，数値演算部分は real（実数）や integer（整数）などを用い，論理部分との間はタイプを変換する。
6) デッバグのために，シミュレータの波形観測機能のほか以下のような工夫をする。
   ・file 入出力機能を使用する。
   ・観測点を下位の階層にも及ぼすため，適宜 parts の port を増設する。
7) シミュレータでの compile では階層を分けてやる必要がある。

(2) テストベンチの VHDL 記述詳細

リスト 2 にその entity と architecture の一部分を示す。

リスト 2　テストベンチの VHDL 記述

```
library STD,ieee;
use STD.TEXTIO.all;
use IEEE.std_logic_textio.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use work.all;

entity Viterbi_TEST is
end Viterbi_TEST;

architecture VHDL of Viterbi_TEST is
 signal Do: std_logic_vector(5 downto 0):= "000000";
 signal x:std_logic_vector(5 downto 0):= "000000";
      -- 以下同様に，signal と必要な初期値，
      -- constant の宣言
   component viterbi
      port(x: in std_logic_vector(5 downto 0);
           ck,ck1: in std_logic;
           acc: in std_logic_vector(9 downto 0);
           out1: out std_logic;
           out2: out std_logic;
           accin: out std_logic_vector(9 downto 0);
           dctl1,dctl2: inout std_logic;
           reg1,reg2: inout std_logic_vector(19 downto 0)
           );
   end component;
   component AD6bit
      port(ck: in std_logic;
           vmax,vmin,ain: in real;
           Do: out std_logic_vector(5 downto 0)
           );
   end component;
   component mpss15
      port(ck: in std_logic;
           p: out std_logic
           );
   end component;
   component Noisegenerator
      port (ck: in std_logic;irn:in integer;
            average,sigma:in real;noise1: out real
            );
   end component;

  -- 以下ファイル変数を定義
file invectors:STD.TEXTIO.TEXT;
file outvectors:STD.TEXTIO.TEXT;
file outvectors1:STD.TEXTIO.TEXT;
begin
--以下インスタンスの記述
mpss: mpss15 port map(ck,DATA);
-- A/D converting
AD:AD6bit port map(ck1,Vmax,Vmin,VAIN,x);
-- Viterbi decoding ---
vit: viterbi port map(x,ck,ck1,ACC, out1,out2,
                      ACCIN,dctl1,dctl2,reg1,reg2);
-- Noise Generation
NG: Noisegenerator port map(ck,irn,average,sigma,noise1);

--以下はプロセスの記述
initial:process
begin
--  for ADC and noise generation
  Vmax<= 2.0;
  Vmin<= -2.0;

-- エラー回数やカウンタ，乱数初期値，
-- その他定数を記述
  Enum<= 0;
     n1<= 0;
     n2<= 0;
     irn<= 5;
     average<= 0.0;
     sigma<= 0.1;
     reset<= '0';
 wait;
end process;

-- クロック ck,ck1 を定義
clock:process begin
 ck<= '0';
 wait for STEP/100;
 ck<= '1';
 wait for STEP/100;
 ck<= '0';
 wait for 5 ns;
end process;
clock1:process begin
 ck1<= '0';
 wait for 5 ns;
 wait for STEP/100;
 ck1<= '1';
 wait for STEP/100;
end process;
process(ck)
variable Dotemp:std_logic_vector(5 downto 0);
variable VA,VA2,VA10,VA20:integer;
variable VAIN0:real;
variable DATA0:std_logic;
variable Lo,Lo1,Li:Line;     --ファイル処理のため定義
begin
  count<= count +1;  --定常ルーチンまで待つため
-- のカウンタ
FILE_OPEN(invectors,"viterbi.in",READ_MODE);
FILE_OPEN(outvectors,"viterbi.out",WRITE_MODE);
FILE_OPEN(outvectors1,"viterbi1.out",WRITE_MODE);
```

```
if(ck'event and ck='1') then
    if(reset='1') then
    ACC<= "0000000000";
    else
    ACC<= ACCIN;
    end if;
VA10:=VA1;
   DATA0 := DATA(0);
   VA2 := CONV_INTEGER(DATA0);
   VA20:= VA2;
   VA  := VA2-VA1;
   VA1<= VA2;
   VAIN<= real(VA);
   VAIN0:= real(VA);
   VAIN<= VAIN0+noise1;
    -- MPSS 出力の一時記憶
   for i in 49 downto 1 loop
      reserve(i)<= reserve(i-1);
    end loop;
      reserve(0)<= DATA0;
 if((count>50))then

  -- ビタビデコードエラーのカウント
    if (reserve(21)/= out1) then
    Enum<= Enum+1;
   end if;
   end if;
write(Lo,count,RIGHT,6);
write(Lo,NOW,RIGHT,6);
write(Lo,DATA(0),RIGHT,5);
write(Lo,DATA0,RIGHT,5);
--以下同様なファイル処理の記述
writeline(outvectors,Lo);     --ファイル出力
end if;
end process;
end VHDL;
```

リスト 3　ランダムビットパターン発生器（MPSS）の VHDL 記述

```
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
entity mpss15 is
  port (ck:in std_logic; p: out std_logic_vector(14 downto
                                0));
end  mpss15;
architecture VHDL of mpss15 is
  signal i: integer;       --整数としての変数宣言
  signal p0:std_logic_vector(14 downto
0):="111111111111111";
  --15ビットの std_logic と定義し，その初期値を
  --すべて "1" にしている。
begin
 initial: process  begin
  wait;
 end process;
 randombit: process(ck)
    --ローカルな変数の定義
  variable ptemp:std_logic_vector(14 downto 0);
  variable x1:std_logic;
  begin
   ptemp := p0;
   if(ck'event and ck='1') then
    x1 := ptemp(0) xor ptemp(14);
    for i in 14 downto 1 loop
    --i が14から 1 までの間実行
     ptemp(i) := ptemp(i-1);
    end loop;
     ptemp(0) := x1;
     p<= ptemp;          --テストベンチへの返り値
     p0<= ptemp;
   end if;
 end process;
end VHDL;
```

以下に，上記のテストベンチ専用のモジュールについて説明する。

(a) MPSS（ランダムビットパターン発生器）

ランダムビットパターン発生回路は LFFR（Linear Feedback Shift Register）で構成し，これを VHDL 記述した。シフトレジスタは，15ビット構成として初期値はすべてに 1 を入力した。リスト 3 に MPSS の VHDL 記述を示す。シフトレジスタの内容は p0 を初期値としてクロック（ck）が立ち上がる毎に xor 動作とシフト動作を繰り返してランダムになっていくが p0 の値は，MPSS が呼ばれる毎に変化していくようになっているところが工夫した点である。

(b) Noise 発生器の VHDL 記述

まず乱数の種の初期値を 5 として，大きな数を掛けている。

これを順番に掛けていき，乱数を出力するためにこのような計算をしている。そしてこれを繰り返して乱数の発生を行っている。

リスト 4 に乱数発生器（Noisegenerator）の VHDL 記述を示す。

乱数の種は最初は "5" となっており，ix0 となる変数に格納されるが，与えられた平均値，標準偏差を持つ乱数発生の毎に ix0 が書き換えられていく，これによって乱数性が保たれるように工夫してある。

リスト4　乱数発生器（Noisegenerator）の VHDL 記述

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
entity Noisegenerator is
  port (ck:in std_logic;irn:in integer;average,sigma:in
  real;noise1: out real);
end Noisegenerator;
architecture VHDL of Noisegenerator is
begin
random:process(ck)
  variable ix: integer;
  variable ix0: integer:=5;  -- 乱数の種の初期値を 5
                                --に設定 --
  variable ox,a: real;
begin
 if (ck'event and ck='1') then  -- クロッパルスの発生
    a := 0.0;                    --  a を0.0に設定
    ix := ix0;                   --  ix を ix0 に設定
  for i in 1 to 12 loop
    ix := ix*48828125;     -- 数学的数字の大きな
                              --数をかける
 if(ix<0) then             -- 乱数出力の計算 --
    ix := (ix+2147483647)+1;
 end if;
    ox := real(ix)*0.4656613e-9;
    a := a + ox;
  end loop;
    noise1<= (a-6.0)*sigma+average;  -- 平均値,
                 --標準偏差を指定の値に設定
    ix0     := ix;  -- ix を ix0 に代入し，次の呼び出
                 --し時の乱数の種とする
   end if;
  end process;
end VHDL;
```

（c）AD 変換器

ビタビデコーダ内はディジタルとして扱われ，ランダムビットパターン発生器（MPSS）出力に雑音が加わった信号はアナログとして扱われているため，ディジタルに直すために AD 変換を行う。磁気ディスク信号に1-D相関を与えた場合ビタビデコーダへの入力は，−1 から 1 の間で変化するので雑音分を考えて，最小値を−2.0 最大値を2.0とする[5]。

AD 変換は 6 bit で表され，その情報を64に分ける。一番下を 0 と置き，上を63とし，その真ん中を32とする。6 ビットのディジタル値に変換した後に32を上限とし，真ん中を 0 と置き，下限は補数をとって−32にしている。この負の部分は 2 の補数として扱っている。そしてここで得られた情報が，ビタビデコーダに入ることになる。

アナログ値の分割と量子化ために real，integer の使いわけや AD 変換の前にアナログ値やディジタル値のシフトをしているところに工夫がある。

リスト 5 に AD 変換器の VHDL 記述を示す。

リスト 5　AD 変換器の VHDL 記述

```
begin
initial:process
 variable max :real := 2.0;  --  最大値を2.0に設定
 begin
 me<= max*2.0/64.0;          --  右辺を me に代入
 wait;
end process;
conversion: process(ck)
  variable a0,a2temp:integer;
  variable areal1:real;
  variable d1,d2,h:std_logic_vector(5 downto 0);
begin
if(ck'event and ck='1') then  --クロック・パルスの発生
  h := "100000";            --  6 bit の情報を32に設定
  areal1 := ain/me;         --  areal1 を実数に設定
  a0 := integer(areal1);    --  a0 を整数 areal1 に設定
  a2temp := a0;
 if((a2temp>=32)) then
  a2temp := 32;
 elsif((a2temp<=-33))then
  a2temp := -32;
 end if;
  a2temp := a2temp + 32;      --  整数値で表すために
                                --  +32を加える
  a2<= a2temp;
  d1 := CONV_std_logic_vector(a2temp,6);
  d2 := d1+h;                 --  補数を表現
   Do<= d2;
  end if;
end process;                  --  プロセスの終了
end VHDL
```

（d）MPSS，Noisegenerator，AD 変換器のシミュレーション

図 2 は，ランダムビットパターン MPSS の multiSIM でシミュレーションの結果をファイル処理した結果であり，時間とともにビットパターンがランダムになっていくことがわかる。

表 2 は A/D 変換器と雑音発生器のシミュレーションでファイル処理した結果をに示す。同図で VAIN 0 はアナログ値として入力した値であり noise 1 は雑音発生器出力であり VAIN はこれらが重畳したものである。

D0 は，VAIN を A/D 変換して得た 6 ビット数値であ

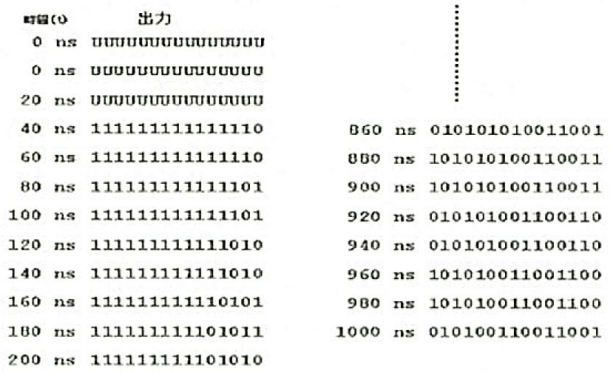

図 2　出力結果（ランダムビットパターン）

表 2　A/D 変換器と雑音発生器のシミュレーション結果

NOW	VAIN0	VAIN	noisel	Do
0 ns	0.000000	0.000000	0.000000	000000
40 ns	1.000000	0.000000	0.132091	000000
80 ns	1.500000	1.132091	0.130582	000000
120 ns	2.000000	1.630582	0.136799	010010
160 ns	−1.000000	2.136799	0.133836	011010
200 ns	−1.500000	−0.866164	0.134745	100000
240 ns	−2.000000	−1.365255	0.134372	110010

A/D 変換

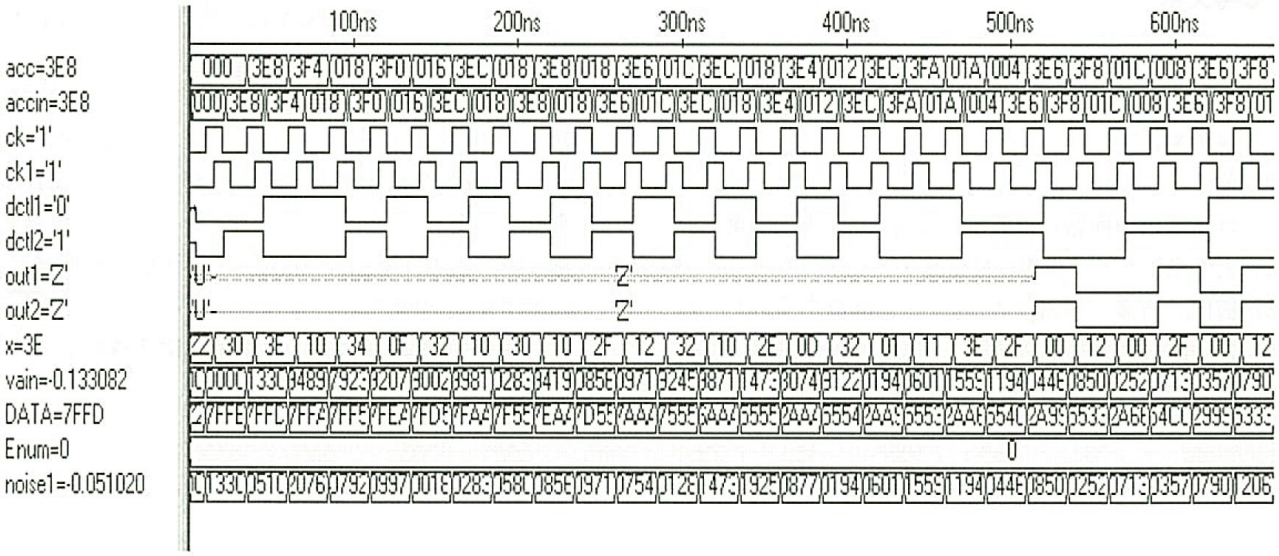

図 3　ビタビ復号回路の VHDL シミュレーション結果

（注）acc：アキュームレタからビタビへの入力
accin:：ビタビかアキュームレータへの出力
ck,ck1：クロック
dctl1,dctl2：シフトレジスタへの入力
out1,out2：シフトレジスタからの出力
x：ビタビ入力
vain：1-D 信号＋noise1
DATSA：MPSS 出力
Enum：エラー累積値
Noise1：乱数発生器出力

る。ノイズがのる前の信号の入力が VAIN 0。これにノイズがのってくる状態を調べたものでありそれぞれの正常な動作を確認できる。

4. ビタビ復号回路のシミュレーション

ビタビ復号回路を構成する各コンポーネントやテストベンチ用モジュールについて動作は上記のように確認した[9), 10)]。テストベンチを含めたビタビ復号回路のシミュレーションは， multiSIM に Parts.vhd， Viterbi.vhd， Testbench.vhd なる 3 つのファイルを加えて行った。結果を図 3 に示す。同図のシミュレーション時間を長くしてもビタビ動作のエラー累積数 E_{NUM} が 0 であることを確認した。その他各コンポーネントの動作が正常なことも確認することができた。

5. む す び

テストベンチを含む磁気ディスク信号処理回路のVHDL記述を用いてシミュレーションを行いその妥当性を確認した。

(1) テストベンチを含めたビタビ復号回路のVHDLシミュレーションの結果，正常な動作を確認をした。

(2) テストベンチに用いられたのは以下のモジュールであり，このようなアルゴリズムを含むVHDL記述を実用化することができた。

① ランダムビットパターン発生回路

② 雑音発生器

③ A/D変換

参考文献

1) 秀島，“Verilog-HDLによるビタビ復号回路の高性能化に関する検討”，平成10年度九州大学電気工学科卒業論文

2) 前田，“磁気ディスクの信号再生用ビタビ復号アルゴリズムの回路化に関する研究”，平成10年度九州大学電子デバイス専攻修士論文

3) 前田，吉澤，中司，谷口，“差分演算を用いた二状態ビタビ復号回路の検討”，平成9年度電気関係九州支部連合大会論文集，No.309

4) 前田，秀島，吉澤，中司，谷口，“差分演算を用いたビタビ復号回路の集積回路化”，平成10年度電気関係九州支部連合大会論文集，No.626

5) 松尾，“磁気記録再生系における信号処理の計算機シミュレーションによる検討”，平成9年度九州大学電気工学科卒業論文

6) 光山，“磁気ディスク信号処理用ビタビ復号回路の回路化と高性能化の検討”，平成12年度久留米工業大学電子情報工学科研究生報告

7) 谷口，光山，小路口，山田，中司，“磁気ディスク信号処理回路のSPICEとHDLによるシミュレーションの検討”，平成12年度電気関係九州支部連合大会論文集

8) 谷口，山田，中司，“磁気ディスク信号処理回路のシミュレーションとFPGA化の検討”，平成13年度電気関係九州支部連合大会論文集

9) 有馬，泊，仲間，吉村，“磁気ディスク信号処理回路のEWBによるシミュレーションとFPGA化の検討”，平成12年度久留米工業大学電子情報工学科卒業論文

10) 真栄田，山本，青山，“磁気ディスク信号処理回路のVHDLとSPICEによるシミュレーション技術の検討”，平成12年度久留米工業大学電子情報工学科卒業論文