2次元積符号デコーダの設計

課題　　上級課題
3.8Gbps デコーダの設計

チーム名　　チーム Veritak

代表者氏名　菅原孝幸（社会人）

設計者　　菅原孝幸
陸　偉良
菅原明美(3 名）

所属　菅原システムズ(菅原孝幸、菅原明美）
上海ウイリンエレクトロニックテクノロジー株式会社（陸　偉良）

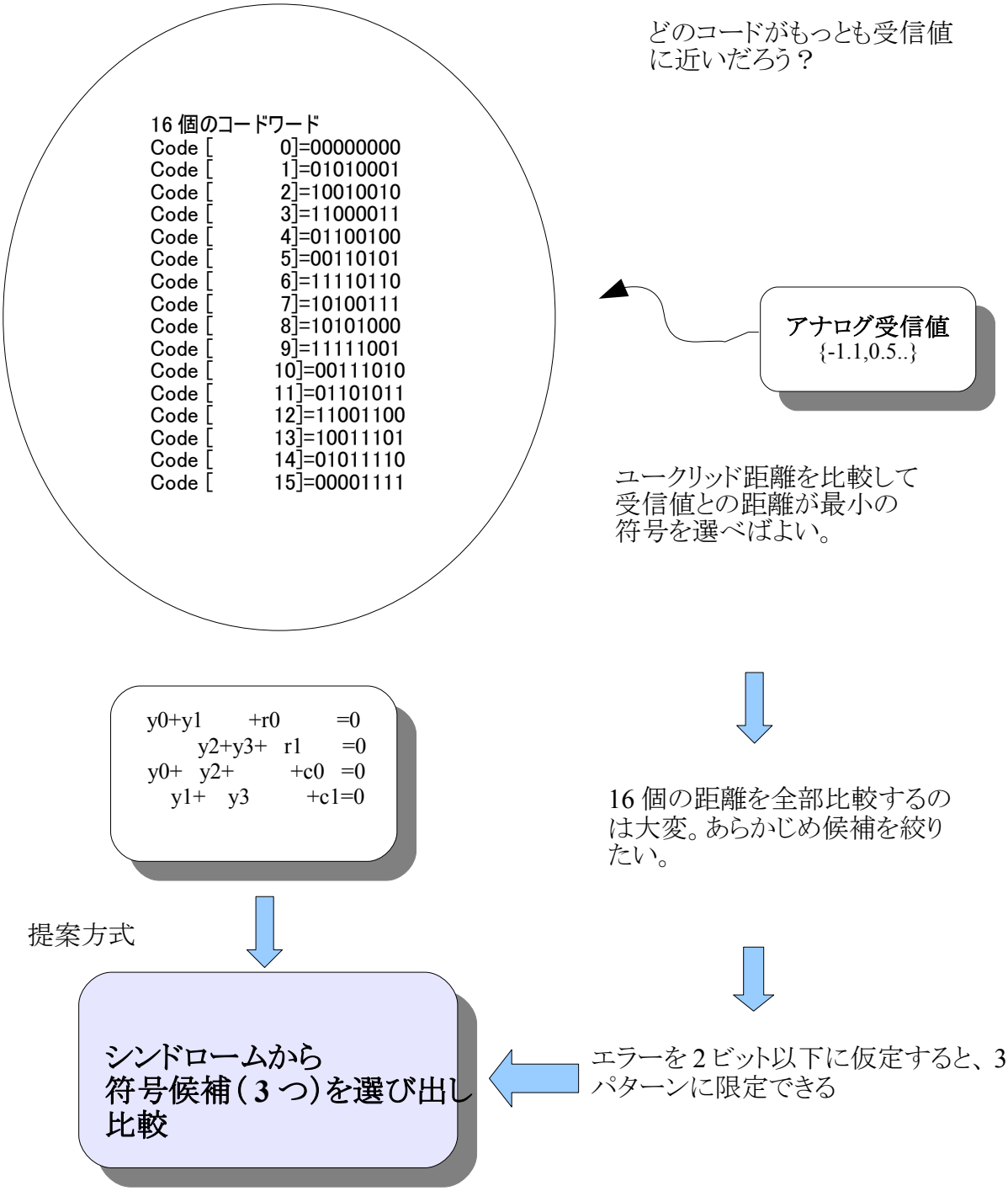
1.概略

繰り返し復号を使わずに、まったく新しい別のアプローチで、速度/LUT 数最大を目指す設計を行いました。また、AGWN 生成器を設計したデコーダと共に、一つの FPGA 上に搭載し、エラーレートも実測しました。下に設計したデコーダの仕様を示します。

表1

項	*項目*	*仕様*	*備考*
1	訂正能力	理想デコーダより 0.5dB 以下の劣化(BER1e-5)	
2	スループット	3.8Gbps EP2S15F484C3 1.5Gbps XC3S200-4	（情報転送レート 1.9Gbps/750Mbps）
3	LUT 数	309 (EP2S15F484C3) 291 (XC3S200-4)	Stratix Ⅱ Spartan3

2. 設計
2.1提案方式の概要



2.2 最小距離復号法

ノイズがガウス性の場合は、ベクトル符号語 x と受信語 y とのユークリッド距離

$$D\ (\ \mathrm{x}\ 、\ \mathrm{y}\ ) = \sqrt{\Sigma (x_i - y_i)^2}$$

--(1)

が小さいほど尤度 P(y| x) は、大きくなることが知られています。(参考文献＊1)

今回の符号は符号長が極端に短い(8 ビット)ので、例示の繰り返し復号に性能・コスト比で対抗できる可能性があると考え、検討を開始しました。

2.3 最小距離復号の簡単化

データは、4 ビットしかないので、符号としてありえるのは 16 個です。この 16 個のデータの内、受信値との距離が最小の符号を選べば、Maximum Likelihood decoding となります。比較だけの問題なので、(1)式のルートは、外せ、

$$\Sigma (x_i^2 - 2\mathrm{x}_i y_i + y_i^2) \qquad \text{--(2)}$$

を計算すれば、よいことになりますが、 $\Sigma (x_i^2 + y_i^2)$ 受信語は比較コードワードによらず同じですから比較対象から外せます。従って

$$\Sigma (-\ 2\ \mathrm{x}_i y_i) \qquad \text{--(3)}$$

だけを対象にすればよいことになります。ここでは、

x_i :1 => -y
x_i :-1 => y --(4)

と計算することにします。(4) について、8 ビットの総和を求め比較し、最小のパターン符号を選びだせばよいことになります。

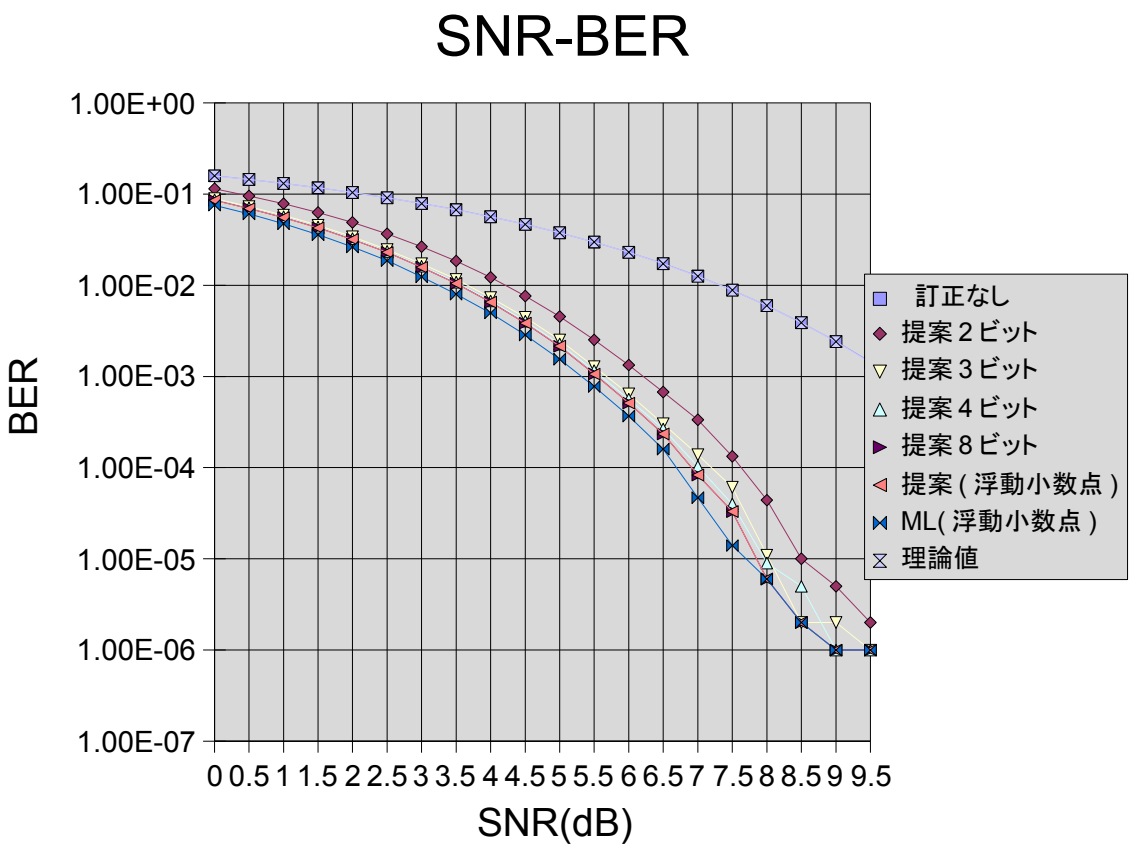
2.4 Near Optimum な性能

大分簡単になりましたが、それでも 16 個のデータパスを計算するのは、大変です。そこで、少し性能を犠牲にして回路を大幅に簡単化する方法を検討しました。この符号の最小距離は、添付プログラム 1 により 3 であることが分かっています。そこで、2 ビット以下のエラーを仮定してみます。実際に起こりえる全ての 2 ビット以下エラーに対するシンドローム(パリティエラーの出方)をプログラム3)により調べ、最長 3 符号の候補を持てば、十分であることが分かりました。

Maximum Likelihood decoding では、3 ビット以上でも訂正可能な場合がありますが、その発生頻度は、実用的な BER 付近では、1 ビットエラーや、2 ビットエラーと比べ少ないと予想され、劣化は小さいと予想できます。この考えが妥当かどうかを、シミュレーションを行い検証しました。

2.5　提案方式の検証

下図は、AWGN の S/N に対する BER のシミュレーションです。(　添付プログラム 2)　各 S/N　1e6 ビット固定でシミュレーションしてみました。



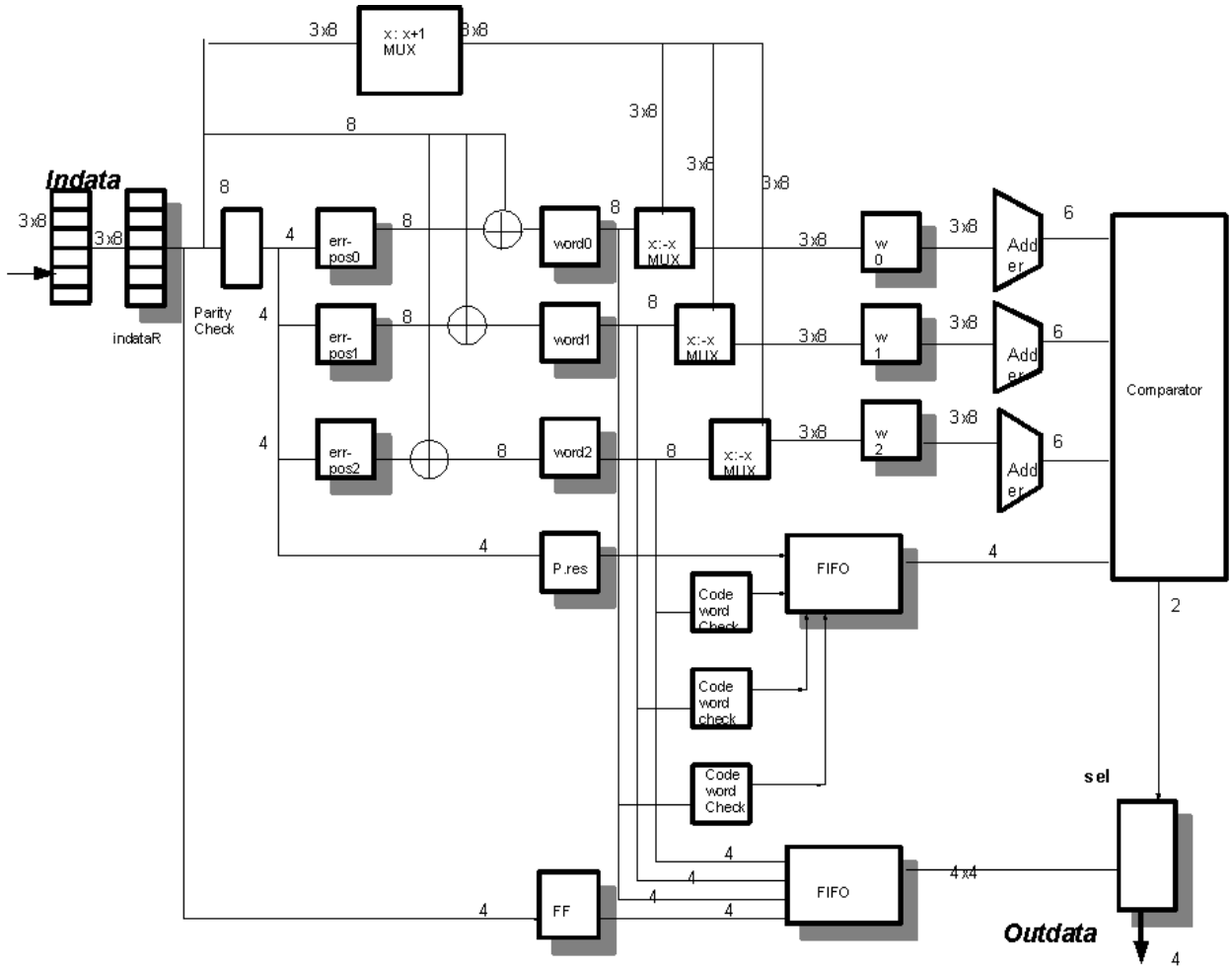
ML（Maximum Likelihood)の結果が理想値で、それより、0.5dB 以下の劣化にとどめようとすると 3 ビット以上が必要であることが分かります。また、提案方式で、浮動小数による演算と 3 ビット演算の差は、BER=1e-3 において、0.2dB 以下であり 3 ビットでも十分な効果を得ることができます。
ビット幅は、トップの階層 parameter を換えるだけで簡単に再合成することができるように記述しています。スパルタン3による合成結果を下に示します。

Bit幅	フリップフロップ数	LUT数	遅延(ns)	周波数MHz
2	196	225	5.73	174.67
3	255	291	5.37	186.22
4	318	385	6.47	154.63
8	554	763	8.47	118.06

これよりビット幅のコストは、大きいことが分かります。以上の検討からビット幅は、3 ビットに決定しました。

3. 実装

3.1 ブロックダイアグラム

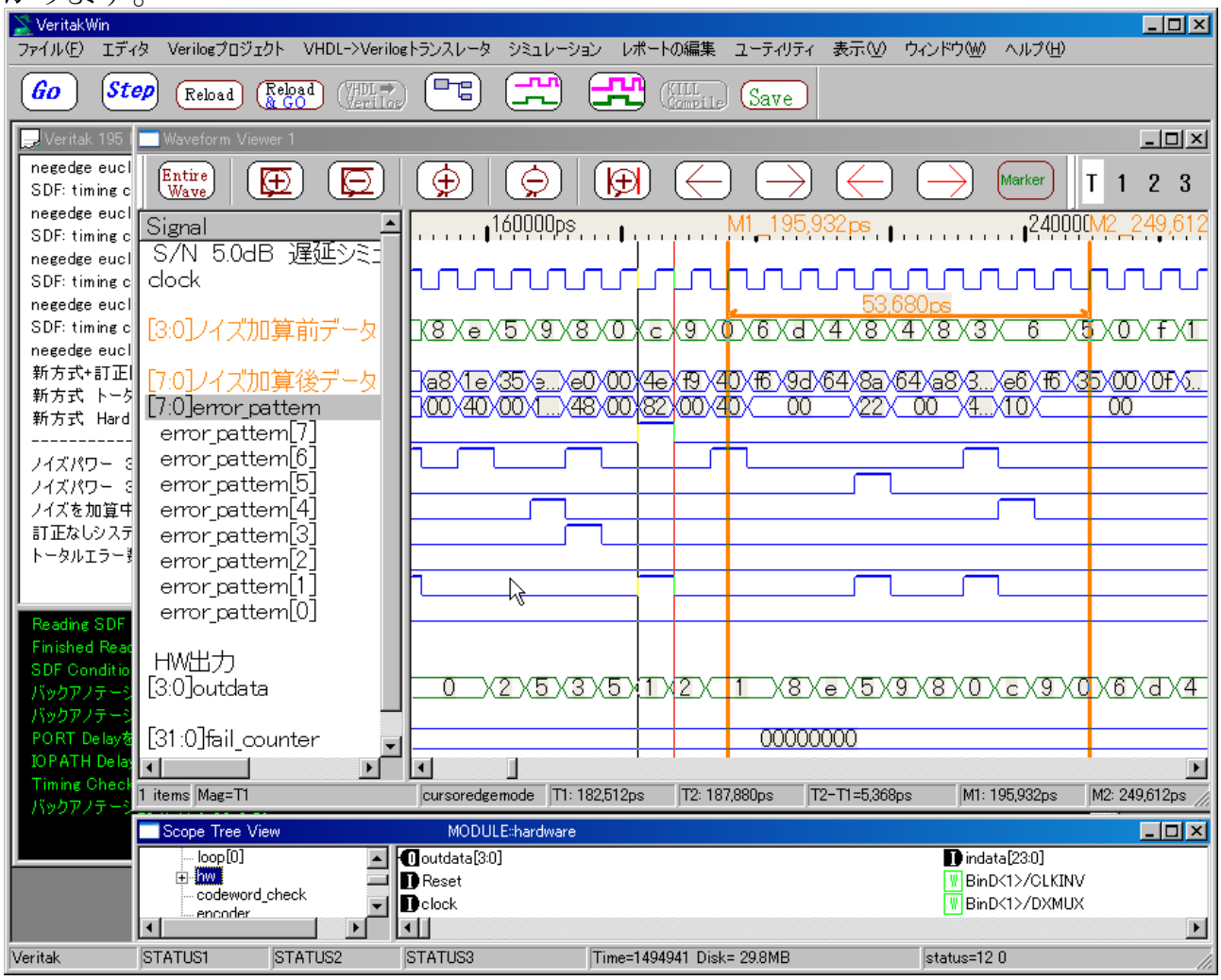


＜ブロック図の説明＞

1)3x8 ビットの量子化したデータを Indata として受け取ります。
2）パリティチェックを行い、3 つのエラーパターン候補を生成します。(err_pos0/1/2)
3）3 つの符号候補を生成します。(word 0/1/2)
4)3つの符号候補についてユークリッド距離を計算します。(w0/1/2 adder)
5）コンパレータでユークリッド距離を比較します。
6)シンドローム状況、及びユークリッド距離から、4つのパターン（訂正なし、word0/1/2）からひとつを選択し Outdata として出力します。

3.2 遅延シミュレーション

スパルタン3で、合成、バックアノテーション後、遅延シミュレーションを行いました。（添付プログラム　2　Define　SPARTAN_DELAY 設定）
下で、ノイズ加算前データに、ノイズ(バイナリで見るとerror_pattern)が加算されています。この例では、1ビットないし2ビットのエラーが発生していますが、10クロック後 outdata では、元のノイズ加算前のデータが再現していることがわかります。



同様にして、Altera　StratixⅡ についても遅延シミュレーションで動作を確認しました。（添付プログラム　2　Define　ST2_DELAY 設定）

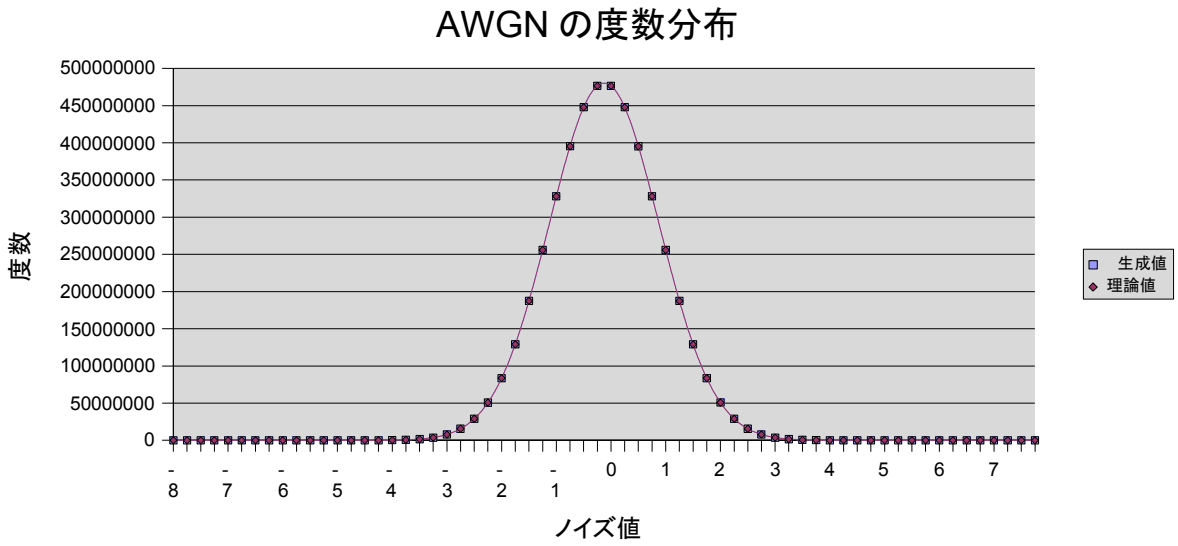
3.3 AWGN 生成

手持ちの FPGA ボードで、

- 各 S/N での AWGN 生成、
- エラー数のカウント
- エラー状況出力

まで、自律的に試験したいと思いました。というのは、一度走らせれば、後はテスト終了まで PC の介在が不要となり時間の許す限りの多くのビットをテストすることができるからです。そのためには、AWGN をメモリに蓄えるのではなく自律的に生成できる必要があります。AWGN の生成方法としては、Box Muller[2], Wallace[3], ZIGGURAT[4] の方法が知られていますが、いずれも浮動小数点の演算が必要となります。ハードウェアで作成すると、それだけで一つの IP 開発が必要となりますので今回は、ソフトウェアで生成する方法を取りました。プロセッサには、Opencores に登録してある自作プロセッサ YACC を使いました。また、AWGN 生成の方法としては、比較的高速な ZIGGURAT の方法を採用し、浮動小数演算ライブラリとしては、newlib を使用しました。

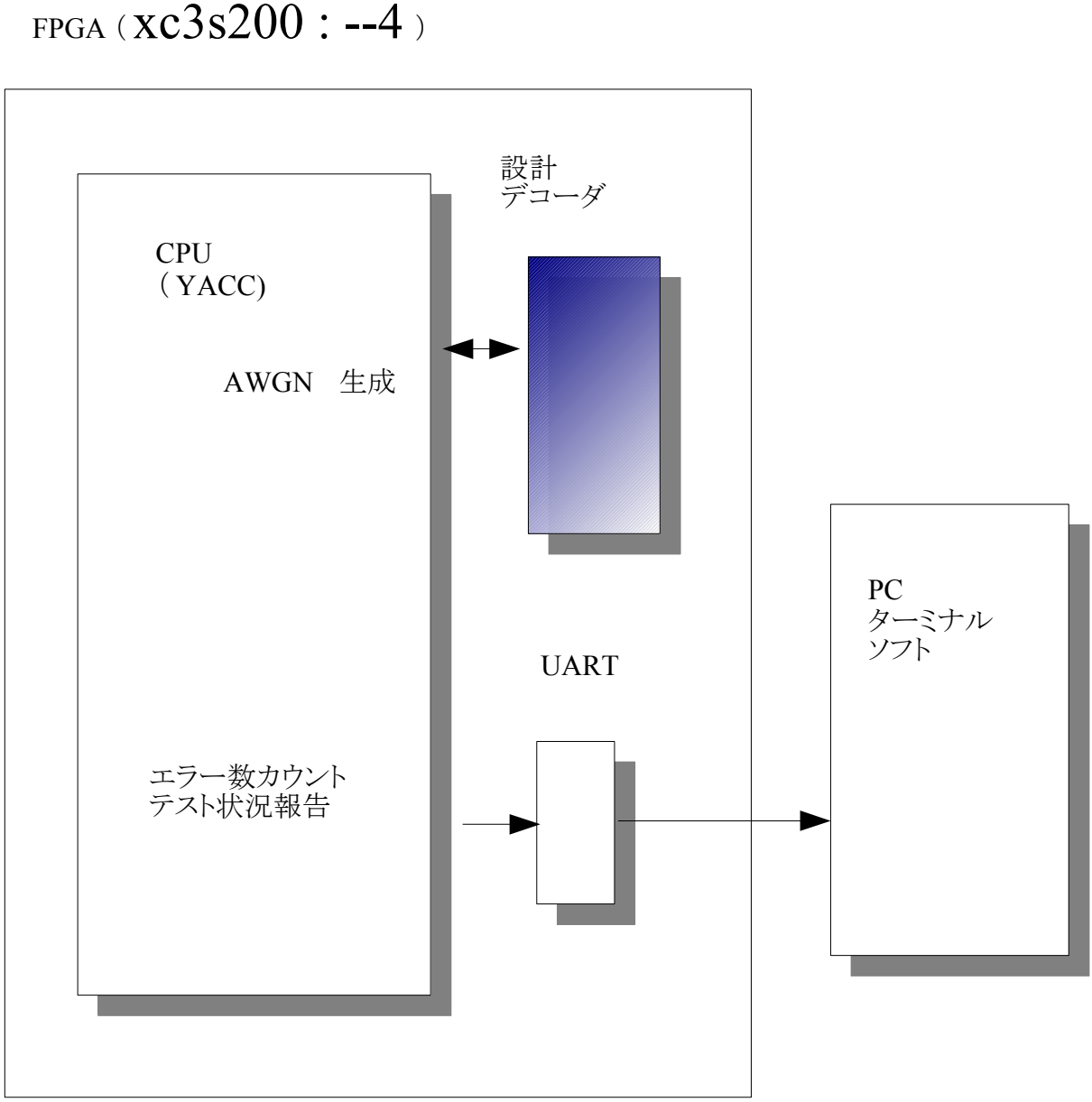
なお、AWGNプログラム（添付プログラム4）の検証として、ノイズ値[-8,8] を63区間に分けカイ自乗テストを行いました。（4e9ビット）



有意水準5%、63区間でのカイ自乗値は、82.53で、上の度数分布での値は54.88です。従い生成したAWGNは、統計的に見てガウス分布と見てよいという結果が得られました。

3.4 検証ブロック

エラーレートの実測ブロック図を下図に示します。設計したデコーダは、CPU のメモリ空間上にマップされます。CPU から見て IO デバイスになります。生成した AWGN は、量子化処理した後、デコーダに書き込まれます。デコーダからの出力は、CPU でエラーカウント処理し、エラーが 1000 個溜まったところで、結果を UART に書き込みます。PC のターミナルソフトでそれを拾います。これらの処理を各 S/N について行います。(添付プログラム 4)

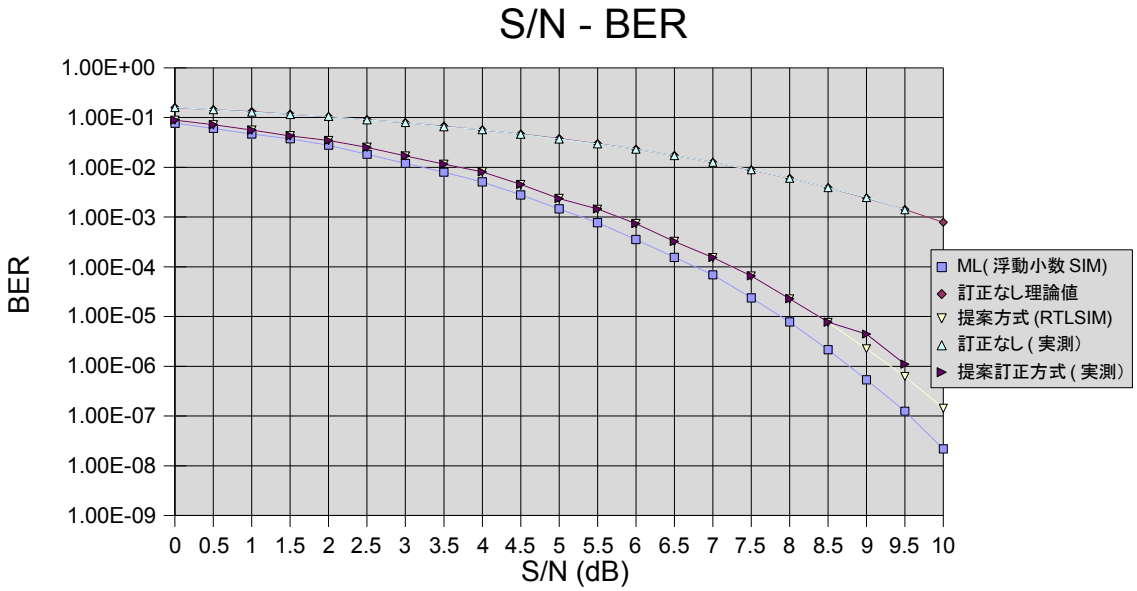


エラーレート実測ブロック図

3.5　エラーレート実測

下図に訂正後の実測値を示します。1E-5 では、理想値（ML 浮動小数 SIM)から、0.5db 以内に収まっています。

エラー数が 1000 個になるまで、テストビット数を増やしていくテスト方法を取っていますが、9dB と 9.5dB の実測値だけエラー数を 10 個にしています。その結果、RTLSIM（VPI を用いた高速 RTL シミュレーション値　添付プログラム6）と乖離が見られます。その他のデータは、良く一致していますので、エラー数が同じく 1000 個程度になるまで、テストビット数を増やせば、RTLSIM の値に収束するものと考えます。

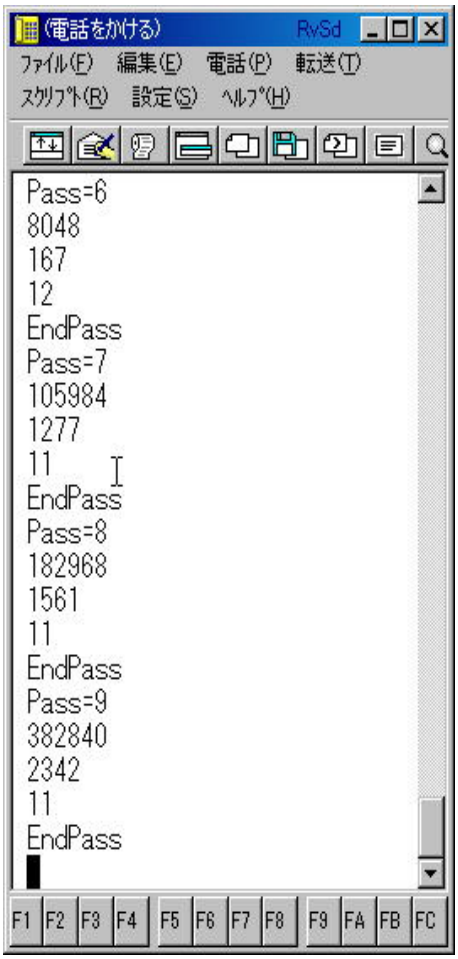


S/N[dB]	ML(浮動小数SIM)	訂正なし理論値	提案方式(RTLSIM)	訂正なし(実測)	提案訂正方式(実測)	テストビット数	訂正なしエラー数	提案方式実測エラー数
0	7.72E-02	1.5866E-01	8.88E-02	1.6011E-01	8.8830E-02	1.13E+04	1806	1002
0.5	6.05E-02	1.4474E-01	7.19E-02	1.4350E-01	7.1786E-02	1.40E+04	2005	1003
1	4.68E-02	1.3093E-01	5.60E-02	1.2866E-01	5.6003E-02	1.79E+04	2302	1002
1.5	3.76E-02	1.1732E-01	4.33E-02	1.1619E-01	4.3043E-02	2.33E+04	2702	1001
2	2.78E-02	1.0403E-01	3.46E-02	1.0580E-01	3.4815E-02	2.88E+04	3042	1001
2.5	1.83E-02	9.1180E-02	2.53E-02	9.0325E-02	2.5234E-02	3.97E+04	3583	1001
3	1.20E-02	7.8896E-02	1.70E-02	7.9586E-02	1.7052E-02	5.87E+04	4672	1001
3.5	8.00E-03	6.7296E-02	1.16E-02	6.5473E-02	1.1612E-02	8.62E+04	5644	1001
4	5.10E-03	5.6495E-02	8.15E-03	5.7040E-02	8.1358E-03	1.23E+05	7018	1001
4.5	2.79E-03	4.6595E-02	4.49E-03	4.6430E-02	4.5370E-03	2.21E+05	10244	1001
5	1.46E-03	3.7679E-02	2.38E-03	3.7248E-02	2.3545E-03	4.25E+05	15836	1001
5.5	7.71E-04	2.9806E-02	1.44E-03	2.9763E-02	1.4527E-03	6.89E+05	20508	1001
6	3.52E-04	2.3007E-02	7.42E-04	2.3103E-02	7.3789E-04	1.36E+06	31341	1001
6.5	1.55E-04	1.7279E-02	3.24E-04	1.7136E-02	3.2378E-04	3.09E+06	52979	1001
7	6.91E-05	1.2587E-02	1.52E-04	1.2686E-02	1.5366E-04	6.51E+06	82647	1001
7.5	2.37E-05	8.8611E-03	6.51E-05	8.9047E-03	6.6028E-05	1.52E+07	134998	1001
8	7.76E-06	6.0044E-03	2.25E-05	6.0006E-03	2.2741E-05	4.40E+07	264134	1001
8.5	2.15E-06	3.8986E-03	7.67E-06	3.9083E-03	7.7151E-06	1.30E+08	507084	1001
9	5.36E-07	2.4133E-03	2.25E-06	2.4661E-03	4.4361E-06	2.48E+06	6115	11
9.5	1.25E-07	1.4161E-03	6.23E-07	1.4067E-03	1.1067E-06	9.94E+06	13982	11
10	2.18E-08	7.8270E-04	1.43E-07					

テストの様子

スパルタン3　スタータキット

UART　ターミナル



3.6 回路の合成結果
下表に示します。

合成ソフト/デバイス	合成対象	LUT数	遅延(ns)	周波数（MHz）	LUT数比	速度比	転送レート(Gbps)
Quartus５．1/	リファレンス回路	14	10.57	94.60	1	1	
EP2S15F484C3	hardw are.v	309	2.11	473.26	22.07	5.003	3.79
ISE7.1/	リファレンス回路	17	12.73	78.57	1	1	
XC3S200-4	hardw are.v	291	5.37	186.22	17.12	2.370	1.49

4.所感
今年は、中国のエンジニア陸 偉良とのコラボレーションです。まだ、顔を合わせたことのない者同士が知恵を出しあい、作業を分担しながら進めました。ここまで、一つのレポートにまとめることができ感慨深いものがあります。
また来年も設計できる機会があることを楽しみにしたいと思います。

5.参考資料

[1]符号理論 今井秀樹 電子情報通信学会

[2] D. Lee, W. Luk, J. Villasenor, and P. Cheung, “A Gaussian noise generator for hardware-based simulations,” *IEEE Transactions on Computer*s, vol. 53, no. 12, pp. 1523–1534, 2004.

[3] C. Wallace, “Fast pseudorandom generators for normal and exponential variates,” *ACM Transations on Mathematical Software (TOMS*), vol. 22, no. 1, pp. 119–127, 1996.

[4] G. Marsaglia and W. W. Tsang, “The Ziggurat Method for Generating Random Variables,” *Journal of Statistical Soft-war*e, vol. 5, no. 8, 2000.

6.添付資料

プログラム１ - 符号の最小距離を調べる
プログラム２ - System/RTL/遅延シミュレーション及びビット幅検討
プログラム３ - 1ビット、2ビットエラーによるシンドローム検討
プログラム４ - AWGN 度数分布及び実機 C プログラム
プログラム５ - 実機ハードウェア記述（TOP 階層のみ）
プログラム６ - VPI による高速 RTL シミュレーション

プログラム1 - 符号の最小距離を調べる

```
//課題の符号を全部列挙する
//Brute Force に最小距離を調べる
module dwm1_test;

	reg [3:0] data;//Encode する元 Data の WORK
	reg [7:0] code_array[0:15];//全コードワードを格納する
	reg [7:0] work,a,b,c,d;

	wire [7:0] encoded_data;//encoder からの出力を受け取る
	integer i,j,flag,m,k,n;

	encoder en1(data,encoded_data);

	initial begin
		//符号の列挙
			for (i=0;i<16;i=i+1) begin//
				data=i[3:0];
				#10;
				code_array[i]=encoded_data;//配列に格納
				$display("Code [%d]=%b 0x%h",i,encoded_data,encoded_data);
			end
			for (i=0;i<16;i=i+1) begin//
				if (i==0) $write("codewords[16]={");
				if (i !=15)$write("0x%h,",code_array[i]);
				else $display("0x%h};",code_array[i]);
			end
		//CODE Word のビットの任意1ビットを変化させて別符号語に一致するかを見る。
			flag=0;
			$display("1 ビットエラーをチェックしています。");
			for (i=0;i<16;i=i+1) begin//
				work=code_array[i];//元の Data に
				for ( j=0;j<8;j=j+1) begin
					a=work ^ (1 <<j);//1bit エラーを乗せて
					for (m=0; m<16;m=m+1) begin//同じ符号語があるかどうかをチェック
						if (a==code_array[m]) flag=1;//同じ符号語が見つかったら フラグ ON
					end
				end
			end
			if (!flag) $display( "1 ビットエラーで、他の符号語になることはありませんでした。最小距離は、1より大きいです。\n");
		//CODE Word のビットの任意2ビットを変化させて別符号に一致するかを見る。
			flag=0;
			$display("2 ビットエラーをチェックしています。");
			for (i=0;i<16;i=i+1) begin//
				work=code_array[i];//元の Data に
				for ( j=0;j<8;j=j+1) begin
					a=work ^ (1 <<j);//1bit エラーを乗せて
					for (k=0;k<8;k=k+1) begin
						b=a ^(1<< k);//もう 1 ビットエラーを乗せて
						if (work !=b) begin//同じ符号は除いて
							for (m=0; m<16;m=m+1) begin//同じ符号語があるかどうかをチェック
								if (b==code_array[m]) flag=1;//同じ符号語が見つかったら フラグ
ON
							end
						end
					end
				end
			end
			if (!flag) $display( "2 ビットエラーで、他の符号語になることはありませんでした。最小距離は、2 より大きいです。\n");
```

```
                //CODE  Word のビットの任意 3 ビットを変化させて別符号に一致するかを見る。
                        flag=0;
                        $display("3 ビットエラーをチェックしています。");
                        for (i=0;i<16;i=i+1) begin//
                                work=code_array[i];//元の Data に
                                for ( j=0;j<8;j=j+1) begin
                                        a=work ^ (1 <<j);//1bit  エラーを乗せて
                                        for (k=0;k<8;k=k+1) begin
                                                b=a ^(1<< k);//もう 1 ビットエラーを乗せて
                                                for (n=0;n<8;n=n+1) begin
                                                        c=b^(1<<n);//もう 1 ビットエラーを乗せ
                                                        if (work !=c) begin//同じ符号語は除いて
                                                                for (m=0; m<16;m=m+1) begin
                                                                        if (c==code_array[m]) begin//同じ符号語があるかどうか
をチェック
                                                                                //$display("%b %b",work,c);
                                                                                flag=1;//同じ符号語が見つかったら  フラグ
ON
                                                                        end
                                                                end
                                                        end
                                                end
                                        end
                                end
                        end
                        if (!flag) $display( "3 ビットエラーで、他の符号語になることはありませんでした。最小距離は、2 より大きいです。");
                        else $display("3 ビットエラーで、他の符号語になることがあります。最小距離は、3 以下です。");

        end



endmodule
module encoder(input [3:0] in,
                                output wire [7:0]  out);
                reg r0,r1,c0,c1;
                wire [3:0] y=in[3:0];

                assign out[3:0]=in;
                assign out[4]=r0;
                assign out[5]=r1;
                assign out[6]=c0;
                assign out[7]=c1;

        always @* begin
                r0=y[0] ^ y[1];
                r1=y[2] ^ y[3];
                c0=y[0] ^ y[2];
                c1=y[1] ^ y[3];
        end

endmodule
```

プログラム2　System/RTL/遅延シミュレーション及びビット幅検討

```
//Nov.2.2005
//Nov.3.2005
`timescale 1ns/1ps
//`define SPARTAN
//`define SPARTAN_DELAY
//`define ST2_DELAY
`ifdef SPARTAN_DELAY
        `define CYCLE (5.368/2)
`elsif ST2_DELAY
        `define CYCLE (4.7/2) //4.5 FAIL due to input delay

`else
        `define CYCLE (5)
`endif


module euclidean_error_rate;
        parameter integer RATE=2;
    parameter integer No_Of_Data_Bits=10000;
        parameter integer No_Of_Array=No_Of_Data_Bits/4;
        parameter integer Total_Bits=No_Of_Data_Bits*RATE;
        parameter integer SEED=1;

        real expected_sigma;
        real noise;

        real noise_array[ 0 :Total_Bits-1];
        real r_array [0 :Total_Bits-1];
        reg  [7:0] data_array  [0 : No_Of_Array-1];
        reg  [7:0] decode_array [0 : No_Of_Array-1];
        reg  [3:0] hard_correction_array [0 : No_Of_Array-1];
        reg  [3:0] euclidean_correction_array [0 : No_Of_Array-1];
        reg  [3:0] hard_and_euclidean_correction_array [0 : No_Of_Array-1];
        integer i;
        integer F1;
        parameter  File_name="error_rate.txt";
        parameter integer BW=3;
        wire [3:0] outdata;
        reg clock=0,Reset;
        integer hw_index;
        wire [BW*8-1:0] indata;
        always #(`CYCLE) clock=~clock;

        genvar gen;
        generate
                for (gen=0; gen<8;gen=gen+1) begin :loop
                        assign indata[ gen*BW +:BW]= get_hw_input_vector(hw_index,gen);
                end
        endgenerate

        hardware # ( .BW(BW)) hw (  .indata(indata),  .outdata(outdata), .clock(clock), .Reset(Reset)          );



        function signed  [BW-1:0]  get_hw_input_vector (input integer index, bit_no);
                integer centor_div;
                integer mul;
                integer qvalue;
                real rvalue;

                begin
                        mul=(2**BW)/2;
                        if (BW>=3) centor_div=2;//量子化の1をどこに置くかのパラメータ
                        else            centor_div=1;

                        rvalue=(r_array[index*8+bit_no]*(mul/centor_div));
                        if (r_array[index*8+bit_no]<0) rvalue=rvalue-1;
                        //量子化規則
                        // 2 倍し四捨五入する
                        //  負数なら  -1 にする  ->HW 内で距離を計算するとき元に戻す
                        // 量子　＋3、－4で飽和させる


                        qvalue=$rtoi(rvalue);//
                        if (qvalue>=mul) qvalue=mul-1;//プラス飽和演算
```

```
                        if (qvalue<=-mul) qvalue=-mul;//マイナス飽和演算
                        get_hw_input_vector=qvalue;
        end
endfunction




initial begin
        Reset=1;
        #100
        Reset=0;
        make_random_data;//10000 個のランダムデータを作る 結果は=>data_array  LSB 側 4bit
        encode_data;//10000 個のパイティを生成する  結果は=>data_array  MSB 側 4bit
        F1=$fopen(File_name);
        if (!F1) begin
                $display("File が開けません");
                $finish;
        end
        write_row_line;
        for (i=00;i<=100;i=i+5) begin//0dB ->10dB  Step=0.5dB
                do_corrections($itor(i)/10.0);
        end
        $finish;
end

task do_corrections(input real noise_db);
        begin
                add_noise(noise_db);//d B ノイズを加える

                raw_decode;//デコードする
        //      hard_correction;
        //      hard_and_euclidean_correction(2);//ビット幅のパラメータ
                hard_and_euclidean_correction(3);
        //      hard_and_euclidean_correction(4);
        //      hard_and_euclidean_correction(8);
        //      hard_and_euclidean_correction(0);
        //      hard_and_euclidean_correction(8);

                euclidean_decode;//理想 ML
        end
endtask

task write_row_line;
        begin
                $fdisplay(F1," dB, 訂正なし,新方式 3 ビット,新方式ハード");
        end
endtask
task make_random_data;
        integer i;
        integer work,counter;
        begin
                $display("%d 個のランダムデータを生成中です。",No_Of_Data_Bits);
                for (i=0;i<No_Of_Array-8; i=i+8) begin//4x8 ビットづつランダムデータを生成
                        work=$random;
                        {data_array[i+3][0 +:4],data_array[i+2][ 0 +:4],data_array[i+1][0 +:4],data_array[i+0][0 +:4]}=work[15:0];
                        {data_array[i+7][0 +:4],data_array[i+6][ 0 +:4],data_array[i+5][0 +:4],data_array[i+4][0 +:4]}=work[31:16];
                end
                work=$random;
                counter=0;
                for (i=No_Of_Array-8;i<No_Of_Array; i=i+1) begin//最後は、4 ビットづつ
                        data_array[i][0 +:4]=work[counter*4 +:4];
                        counter=counter+1;
                end
        end
endtask



task encode_data;
        integer i;
        begin
                $display("パリティを生成中です。");
                if ( (Total_Bits/RATE)% No_Of_Array)begin//assert(0);
                        $display(" プログラムエラー");
                        $finish;
                end
                for (i=0;i<No_Of_Array; i=i+1) begin
                        data_array[i][4 +:4]=encoder(data_array[i][ 0 +:4]);//データを渡してパリティを受け取る

                end

        end
endtask

function [3:0] encoder (input [3:0] y);//パリティ生成器
        reg r0,r1,c0,c1;
        begin
                r0=y[0] ^ y[1];
```

```
                r1=y[2] ^ y[3];
                c0=y[0] ^ y[2];
                c1=y[1] ^ y[3];
                encoder ={c1,c0,r1,r0};
        end
endfunction



task make_noise( input real db);//所望の S/N[dB]を入力
        real sigma2,sigma;
        integer seed,i;
        real noise;
        real noise_sum;
        begin
                $display("--------------------------------------------------");
                $display("ノイズパワー  %f[dB] 狙いで生成中です。",db);
                sigma2=$pow(10.0,-db/10.0);//$pow は、Veritak  Unique
                sigma=$sqrt(sigma2);//$sqrt は、Veritak  Unique
                seed=0;
                noise_sum=0;
                for (i=0;i<Total_Bits;i=i+1) begin
                        noise=$normal_vtak(seed,0.0,sigma);//$normal_vtak は、Veritak  Unique  平均値0、σ の AWGN を生成
                        noise_array[i]=noise;//ノイズ配列に収納する
                        noise_sum=noise_sum+noise*noise;
                end
                $display("ノイズパワー  %f[dB]の生成をしました。",-10*$log10(noise_sum/Total_Bits));
        end
endtask

task add_noise( input real db);//所望の S/N[dB]を入力
        real sigma2,sigma;
        integer counter,i,j;
        real noise,RData;
        begin
                make_noise(db);//所望の S/N  Noize を noise_array に収納する
                $display("ノイズを加算中です。");
                counter=0;
                for (i=0;i<No_Of_Array; i=i+1) begin
                        for (j=0;j<8;j=j+1) begin
                                if (data_array[i][j]==1'b1)             RData=$itor(-1);//データ1＝＞－1.0 にエンコード
                                else            RData=$itor(1);//データ0＝＞1.0 にエンコード
                                RData=RData+noise_array[counter];//計算しておいた AWGN ノイズを加算する
                                r_array[counter]=RData;//r_array にノイズが重畳した real data を Save する。
                                counter=counter+1;
                        end
                end

                if ( counter !=Total_Bits) begin//assert(0);
                        $display(" プログラムエラー");
                        $finish;
                end
                $fwrite(F1,"%f  ",db) ;
        end
endtask

task raw_decode();
        integer i,j;
        integer index;
        integer error_count;
        begin
                error_count=0;
                index=0;
                for (i=0;i<Total_Bits;i=i+8) begin
                        for (j=0;j<8;j=j+1) begin
                                if (r_array[i+j]>0) decode_array[index][j]=0;//パリティも含めハードデコードする
                                else                            decode_array[index][j]=1;
                                if (j<4) begin//データ部はビットエラーをカウントする
                                        if (decode_array[index][j] !==data_array[index][j]) error_count=error_count+1;
                                end
                        end
                        index=index+1;
                end
                $display("訂正なしシステム結果");
                $display("トータルエラー数＝%d エラーレート=%e",error_count, $itor(error_count)/No_Of_Data_Bits);
            $fwrite(F1,", %e ",$itor(error_count)/No_Of_Data_Bits);
        end
endtask


task hard_correction();
        reg p0,p1,p2,p3;
        reg [3:0] parities;
        reg d0,d1,d2,d3;
        integer index,error_count,i;
        begin
                error_count=0;
                for (index=0;index<No_Of_Array; index=index+1) begin
                        {d3,d2,d1,d0}=decode_array[index][3:0];//work
                        p0=d0 ^ d1 ^ decode_array[index][4];//パリティを計算
```

```
                                        p1=d2 ^ d3 ^ decode_array[index][5];//パリティを計算
                                        p2=d0 ^ d2 ^ decode_array[index][6];//パリティを計算
                                        p3=d1 ^ d3 ^ decode_array[index][7];//パリティを計算

                                        parities={p3,p2,p1,p0};
                                        hard_correction_array[index][3:0]={d3,d2,d1,d0};
                                        if (parities==4'b0000);//No Error Nothing to Do
                                        else if (parities==4'b0101) hard_correction_array[index][3:0]={d3,d2,d1,~d0};//1bit 訂正
                                        else if (parities==4'b1001) hard_correction_array[index][3:0]={d3,d2,~d1,d0};//1bit 訂正
                                        else if (parities==4'b0110) hard_correction_array[index][3:0]={d3,~d2,d1,d0};//1bit 訂正
                                        else if (parities==4'b1010) hard_correction_array[index][3:0]={~d3,d2,d1,d0};//1bit 訂正
                                        else ;//parity error or uncorrectable error.
                                        //エラー数をカウント
                                        if (data_array[index][3:0] !==hard_correction_array[index][3:0]) begin//ワード比較して
                                                for (i=0;i<4;i=i+1) begin//違っていたらエラー数をカウント
                                                        if (data_array[index][i] !=hard_correction_array[index][i]) error_count=error_count+1;
                                                end
                                        end
                                end
                                $display("ハード訂正回路結果");
                                $display("トータルエラー数＝%d エラーレート=%e",error_count, $itor(error_count)/No_Of_Data_Bits);
                                //$display("\n");
        end
endtask


task hard_and_euclidean_correction(input integer no_of_bits); // no_of_bits 0 => real
        reg p0,p1,p2,p3;
        reg [3:0] parities;
        reg [7:0] err_pat[0:2];
        reg [7:0] pat[0:2];
        reg [7:0] pattern;
        reg d0,d1,d2,d3;
        integer index,error_count,i;
        integer fail_counter;
        reg [3:0] fail_pat;
        reg [7:0] error_pattern;
        real distance[0:2];
        real min_dist;
        real r1;
        integer min_index;
        reg [7:0] cpat,ccpat;
        real dr0,dr1,dr2;
        begin
                error_count=0;
                fail_counter=0;
                for (index=0;index<No_Of_Array; index=index+1) begin
                        {d3,d2,d1,d0}=decode_array[index][3:0];//work
                        p0=d0 ^ d1 ^ decode_array[index][4];//パリティを計算
                        p1=d2 ^ d3 ^ decode_array[index][5];//パリティを計算
                        p2=d0 ^ d2 ^ decode_array[index][6];//パリティを計算
                        p3=d1 ^ d3 ^ decode_array[index][7];//パリティを計算

                        parities={p3,p2,p1,p0};
                        hard_and_euclidean_correction_array[index][3:0]={d3,d2,d1,d0};
                        if (parities==4'b0000);//No Error Nothing to Do
                        else begin
                                err_pat[0]=err_pos0(parities);
                                err_pat[1]=err_pos1(parities);
                                err_pat[2]=err_pos2(parities);

                                min_index=0;
                                min_dist=1.1e5;//big value
                                for (i=0;i<3;i=i+1) begin :label// 3 エラーパターンの内、距離最小を求める

                                        pat[i]=err_pat[i]^decode_array[index];

                                        pattern=pat[i];
                                        distance[i]=get_distance(pat[i],index,no_of_bits);

                                        if (i==0) dr0=distance[i];
                                        if (i==1) dr1=distance[i];
                                        if (i==2) dr2=distance[i];
                                        if (!codeword_check(pat[i]) ) disable label;//Not codeword. Continue

                                        if (min_dist> distance[i]) begin
                                                min_index=i;
                                                min_dist=distance[i];
                                        end
                                end
                            if (min_dist >1e5) begin
                                        $display("program error ");
                                        $stop;
                                end

                                cpat=data_array[index];
                                hard_and_euclidean_correction_array[index]=pat[min_index][3:0];
                        end
                `ifdef SPARTAN_DELAY
                        @(negedge clock);
```

```
                        `elsif ST2_DELAY
                                @(posedge clock);
                                #(0.1);

                        `else
                                @(negedge clock);
                        `endif
                                cpat=decode_array[index];
                                ccpat=data_array[index];
                                error_pattern=cpat^{get_parity(ccpat),ccpat};
                                r1=r_array[index*8+4];
                                hw_index=index;
                        `ifdef ST2_DELAY
                                @(negedge clock);
                        `endif
                                fail_pat=hard_and_euclidean_correction_array[index];
`ifdef SPARTAN_DELAY
        `define DELAY 10
`elsif  ST2_DELAY
        `define DELAY 10
`else

        `define DELAY 9
`endif


                                if (index>`DELAY) begin
                                        if (data_array[index-`DELAY][3:0] !==outdata) begin

                                                for (i=0;i<4;i=i+1) begin//違っていたらエラー数をカウント
                                                        if (data_array[index-`DELAY][i] !=outdata[i]) fail_counter=fail_counter+1;
                                                end

                                        end
                                end
                                //エラー数をカウント
                                if (data_array[index][3:0] !==hard_and_euclidean_correction_array[index][3:0]) begin//ワード比較して
                                        for (i=0;i<4;i=i+1) begin//違っていたらエラー数をカウント
                                                if (data_array[index][i] !=hard_and_euclidean_correction_array[index][i])
error_count=error_count+1;
                                        end
                                end

                        end
                        $display("新方式+訂正回路結果 No_of_bits=%d",no_of_bits);
                        $display("新方式 トータルエラー数＝%d エラーレート=%e",error_count, $itor(error_count)/No_Of_Data_Bits);
                        $display("新方式  Hard トータルエラー数＝%d エラーレート=%e",fail_counter, $itor(fail_counter)/No_Of_Data_Bits);

                         $fdisplay(F1,", %e, %e ",$itor(error_count)/No_Of_Data_Bits,$itor(fail_counter)/No_Of_Data_Bits);

        end
endtask
function [3:0]get_parity (input [7:0] pat);
                        begin
                                get_parity[0]=pat[0] ^ pat[1] ^pat[4];//パリティを計算
                                get_parity[1]=pat[2] ^ pat[3] ^ pat[5];//パリティを計算
                                get_parity[2]=pat[0] ^ pat[2] ^ pat[6];//パリティを計算
                                get_parity[3]=pat[1] ^ pat[3] ^ pat[7];//パリティを計算
                        end
endfunction
function  codeword_check (input [7:0] pat);
        reg p0,p1,p2,p3;
        begin
                                p0=pat[0] ^ pat[1] ^pat[4];//パリティを計算
                                p1=pat[2] ^ pat[3] ^ pat[5];//パリティを計算
                                p2=pat[0] ^ pat[2] ^ pat[6];//パリティを計算
                                p3=pat[1] ^ pat[3] ^ pat[7];//パリティを計算
                                if (p0 || p1 || p2 || p3) begin//パリティエラーなら Codeword ではないので
                                        codeword_check=1'b0;//Not codeward
                                end else        codeword_check=1'b1;//OK proceed
        end
endfunction
function real get_distance (input [7:0] pat, input integer index, input integer no_of_bits);
        real bit_distance,flip;
        real distance;
        reg [7:0] cpat,dpat;
        integer i;
        begin
                        if (no_of_bits) begin
                                get_distance=get_distance_w_Quantization(pat,index,no_of_bits);//量子化ビット数がある場合
                        end else begin//ビット数指定がないなら real 演算
                                cpat=data_array[index];
                                dpat=pat;
                                distance=0.0;
                                for (i=0;i<8;i=i+1) begin
                                        if (pat[i]) flip=1.0;
                                        else flip=-1.0;
                                        bit_distance=r_array[index*8+i]* flip;//-2*(r_array[i+k]* data_r)
                                        distance=distance+bit_distance;
                                end
```

```
                                        get_distance=distance;
                                end

        end
endfunction



function real get_distance_w_Quantization (input [7:0] pat, input integer index, input integer no_of_bits);
        integer bit_distance;
        integer distance;
        reg [7:0] cpat,dpat;
        integer i;
        integer mul;
        real rvalue;
        integer qvalue;
        integer flip;
        integer centor_div;
        begin
                mul=(2**no_of_bits)/2;
                if (no_of_bits>=3) centor_div=2;//量子化の1をどこに置くかのパラメータ
                else            centor_div=1;
                cpat=data_array[index];
                dpat=pat;
                distance=0.0;
                for (i=0;i<8;i=i+1) begin
                        if (pat[i]) flip=1;
                        else flip=-1;


                        rvalue=r_array[index*8+i]*(mul/centor_div);
                        qvalue=$rtoi(rvalue);//
                        if (qvalue>=mul) qvalue=mul-1;//プラス飽和演算
                        if (qvalue<=-mul) qvalue=-mul+1;//マイナス飽和演算
                        if (flip==-1) qvalue=-qvalue;//
                        bit_distance=qvalue ;//r_array[index*8+i]* flip;//-2*(r_array[i+k]* data_r)

                        distance=distance+bit_distance;
                end
                get_distance_w_Quantization=$itor(distance);
        end
endfunction

task euclidean_decode();
        integer i,j,k,m;
        integer index;
        integer error_count;
        reg [7:0] min_code_word;
        reg [7:0] data;
        real distance,min_distance;
        real bit_distance;
        real data_r;
        integer bit_error_count;
        integer one_bit_error_correction;
        integer two_bit_error_correction;
        integer three_bit_error_correction;
        integer more_than_four_bit_error_correction;
        integer flip;
        begin
                error_count=0;
                index=0;
                one_bit_error_correction=0;
                two_bit_error_correction=0;
                three_bit_error_correction=0;
                more_than_four_bit_error_correction=0;
                for (i=0;i<Total_Bits;i=i+8) begin
                        //符号の列挙

                        min_distance=1e5;//Big Value
                        for (j=0;j<16;j=j+1) begin//全コードワードについて
                                data={encoder(j[3:0]),j[3:0]};//符号の列挙
                                //$display("Code [%d]=%b",j,data);

                                distance=0.0;
                                for (k=0;k<8;k=k+1) begin//ユークリッド距離を調べる
                                        data_r=data[k]==1'b1 ? -1.0 : 1.0;//エンコード

        `define RELATIVE_COMPARISON
                                `ifdef RELATIVE_COMPARISON
                                        if(data_r  <0.0 ) flip=1;
                                        else flip=-1;

                                        bit_distance=r_array[i+k]* flip;//-2*(r_array[i+k]* data_r)
                                        distance=distance+bit_distance;
                                `else
                                        bit_distance=r_array[i+k]- data_r;//ビット毎の距離
                                        distance=distance+ bit_distance**2;//2乗して足す
                                `endif
```

```
                                        end
                                        if (distance <min_distance) begin//距離が最小か？ 比較だけなのでルートは不要
                                                min_distance=distance;//距離を Save
                                                min_code_word=data;//コードワードを覚えておく
                                        end

                                end
                                euclidean_correction_array[index]=min_code_word;
                                if ( (min_code_word ===data_array[index])//訂正が成功した場合の訂正ビット数の統計をとる
                                        && (decode_array[index] !==min_code_word))begin
                                        bit_error_count=0;
                                        for (m=0;m<8;m=m+1) begin//違っていたらビットエラー数をカウント
                                                if (min_code_word[m] !==decode_array[index][m]) bit_error_count=bit_error_count+1;
                                        end
                                        case (bit_error_count)
                                                1: one_bit_error_correction=one_bit_error_correction+1;
                                                2: two_bit_error_correction=two_bit_error_correction+1;
                                                3: three_bit_error_correction=three_bit_error_correction+1;
                                                4,5,6,7,8:more_than_four_bit_error_correction=more_than_four_bit_error_correction+1;
                                         endcase
                                end
                                if (euclidean_correction_array[index][3:0] !==data_array[index][3:0]) begin//コードワードを比較して
                                        for (m=0;m<4;m=m+1) begin//違っていたらビットエラー数をカウント
                                                if (data_array[index][m] !=euclidean_correction_array[index][m]) error_count=error_count+1;
                                        end
                                end
                                index=index+1;
                        end
                        $display("ML システム結果");
                        $display(" 1bit =%d 2bit=%d 3bit=%d 4bit>=%d",one_bit_error_correction,two_bit_error_correction,
three_bit_error_correction,more_than_four_bit_error_correction);
                        $display("トータルエラー数＝%d エラーレート=%e",error_count, $itor(error_count)/No_Of_Data_Bits);
                        $fwrite(F1,", %e\n",$itor(error_count)/No_Of_Data_Bits);
                end
        endtask

        function [7:0] err_pos0 (input [3:0] parity_error);
                begin
                        case(parity_error)
                                4'b0001://Peq=0 single bit parity error
                                        err_pos0=8'b0001_0000;
                                4'b0010://Peq=1 single bit parity error
                                        err_pos0=8'b0010_0000;
                                4'b0100://Peq=2 single bit parity error
                                        err_pos0=8'b0100_0000;
                                4'b1000://Peq=3 single bit parity error
                                        err_pos0=8'b1000_0000;
                                4'b0101://Peq=0Peq=2
                                        err_pos0=8'b0000_0001;//position j=      0
                                 4'b1001://Peq=0Peq=3
                                        err_pos0=8'b0000_0010;//position j=      1 ,
                                 4'b0110://Peq=1Peq=2
                                        err_pos0=8'b0000_0100;//position j=      2 ,
                                 4'b1010://Peq=1Peq=3
                                        err_pos0=8'b0000_1000;//position j=      3 ,
                                 4'b1100://Peq=2Peq=3
                                        err_pos0=8'b0000_0011;//position j=      0 k=      1,
                                 4'b0011://Peq=0Peq=1
                                        err_pos0=8'b0000_0101;// position j=      0 k=      2,
                                 4'b1111://Peq=0Peq=1Peq=2Peq=3
                                        err_pos0=8'b0000_1001;//position j=      0 k=      3,
                                 4'b0111://Peq=0Peq=1Peq=2
                                        err_pos0=8'b0001_0100;//position j=      2 k=      4,
                                 4'b1011://Peq=0Peq=1Peq=3
                                        err_pos0=8'b0010_0010;//position j=      1 k=      5,
                                 4'b1101://Peq=0Peq=2Peq=3
                                        err_pos0=8'b0100_0010;//position j=      1 k=      6,
                                 4'b1110://Peq=1Peq=2Peq=3
                                        err_pos0=8'b1000_0100;//position j=      2 k=      7,
                                default:
                                        err_pos0=8'b0000_0000;
                        endcase
                end
        endfunction

        function [7:0] err_pos1 (input [3:0] parity_error);
                begin
                        case(parity_error)
                                4'b0001://Peq=0 single bit parity error
                                        err_pos1=8'b0100_0001;//position j=      0 k=      6
                                4'b0010://Peq=1 single bit parity error
                                        err_pos1=8'b0100_0100;//position j=      2 k=      6
                                4'b0100://Peq=2 single bit parity error
                                        err_pos1=8'b0010_0100;//position j=      2 k=      5,
                                4'b1000://Peq=3 single bit parity error
                                        err_pos1=8'b0001_0010;//position j=      1 k=      4,
                                4'b0101://Peq=0Peq=2
                                        err_pos1=8'b0101_0000;//position j=      4 k=      6,
                                4'b1001://Peq=0Peq=3
```

```
                                        err_pos1=8'b1001_0000;//position j=     4 k=     7,
                                4'b0110://Peq=1Peq=2
                                        err_pos1=8'b0110_0000;//position j=     5 k=     6,
                                4'b1010://Peq=1Peq=3
                                        err_pos1=8'b1010_0000;//position j=     5 k=     7,
                                4'b1100://Peq=2Peq=3
                                        err_pos1=8'b0000_1100;//position j=     2 k=     3,
                                4'b0011://Peq=0Peq=1
                                        err_pos1=8'b0000_1010;//position j=     1 k=     3,
                                4'b1111://Peq=0Peq=1Peq=2Peq=3
                                        err_pos1=8'b0000_0110;//position j=     1 k=     2,
                                4'b0111://Peq=0Peq=1Peq=2
                                        err_pos1=8'b0010_0001;//position j=     0 k=     5,
                                4'b1011://Peq=0Peq=1Peq=3
                                        err_pos1=8'b0001_1000;//position j=     3 k=     4,
                                4'b1101://Peq=0Peq=2Peq=3
                                        err_pos1=8'b1000_0001;//position j=     0 k=     7,
                                4'b1110://Peq=1Peq=2Peq=3
                                        err_pos1=8'b0100_1000;//position j=     3 k=     6,
                                default:
                                        err_pos1=8'b0000_0000;
                        endcase
                end
        endfunction

        function [7:0] err_pos2 (input [3:0] parity_error);
                begin
                        case(parity_error)
                                4'b0001://Peq=0 single bit parity error
                                        err_pos2=8'b1000_0010;//position j=     1 k=     7
                                4'b0010://Peq=1 single bit parity error
                                        err_pos2=8'b1000_1000;//position j=     3 k=     7
                                4'b0100://Peq=2 single bit parity error
                                        err_pos2=8'b0001_0001;//position j=     0 k=     4
                                4'b1000://Peq=3 single bit parity error
                                        err_pos2=8'b0010_1000;//position j=     3 k=     5,
                                 4'b1100://Peq=2Peq=3
                                        err_pos2=8'b1100_0000;//position j=     6 k=     7,
                                 4'b0011://Peq=0Peq=1
                                        err_pos2=8'b0011_0000;//position j=     4 k=     5,
                                default:
                                        err_pos2=8'b0000_0000;
                        endcase
                end
        endfunction


endmodule
```

合成对象 Xilinx 用

```
module hardware # (parameter BW=3)
                ( input [BW*8-1:0] indata,
                  output reg [3:0]  outdata,
                  input clock, Reset         );
        localparam integer FIFO_DEPTH=6;
        reg [BW*8-1:0] indataR,indataR_D;

        wire [7:0] Bin={indataR[BW*8-1], indataR[BW*7-1],indataR[BW*6-1],indataR[BW*5-1],
                                        indataR[BW*4-1], indataR[BW*3-1],indataR[BW*2-1],indataR[BW*1-1]};
        wire [3:0] parities=parity_check(Bin);

        wire [7:0] err_pat0=err_pos0(parities);
        wire [7:0] err_pat1=err_pos1(parities);
        wire [7:0] err_pat2=err_pos2(parities);

        reg p_res0,p_res1,p_res2,P_res;
        reg p_res0_D,p_res1_D,p_res2_D,P_res_D;
        reg p_res0_DD,p_res1_DD,p_res2_DD,P_res_DD;
        reg p_res0_DDD,p_res1_DDD,p_res2_DDD,P_res_DDD;

        reg p_res2_DDDD;
        reg P_res_DDDD,P_res_DDDDD;

        reg [7:0] word0,word1,word2;
        reg signed [BW-1+3:0] add0,add1,add2;
        reg signed  [BW-1+3:0] add2_D;
        reg [3:0] BinD;

        reg c0;
        reg signed [BW+3-1:0] comp0;
        reg [1:0] sel;


        always @(posedge clock) begin
                indataR<=indata;//Receiver Register
                indataR_D<=indataR;//1-st Stage
        end
```

```
//1-st Stage
        always @(posedge clock ,posedge Reset) begin
                if (Reset) word0<=0;
                else       word0<=err_pat0^Bin;
        end

        always @(posedge clock ,posedge Reset) begin
                if (Reset) word1<=0;
                else       word1<=err_pat1^Bin;
        end

        always @(posedge clock ,posedge Reset) begin
                if (Reset) word2<=0;
                else       word2<=err_pat2^Bin;
        end

        always @(posedge clock, posedge Reset) begin
                if (Reset) BinD<=0;
                else       BinD<=Bin[3:0];
        end

        always @(posedge clock, posedge Reset) begin
                if (Reset) P_res<=0;
                else       P_res<=parities==4'b0000;
        end

//2nd stage
        genvar ge;
        generate
                for (ge=0; ge<8 ; ge=ge+1) begin :lp
                        reg signed [BW-1:0] w0,w1,w2;
                        wire signed [BW-1:0] temp;

                                assign temp=indataR_D[(ge+1)*BW -1] ? indataR_D[ge*BW +:BW]+1'b1: indataR_D[ge*BW  +:BW] ;
                                always @ (posedge clock)  begin
                                        w0 <=!word0[ge] ? -temp :temp;
                                        w1 <=!word1[ge] ? -temp :temp;
                                        w2 <=!word2[ge] ? -temp :temp;
                                end

                end

        endgenerate
        reg signed [BW+2-1 :0] add0_0,add0_1;
        reg signed [BW+2-1 :0] add1_0,add1_1;
        reg signed [BW+2-1 :0] add2_0,add2_1;

        reg signed [BW+1-1 :0] add0_00,add0_01;
        reg signed [BW+1-1 :0] add1_00,add1_01;
        reg signed [BW+1-1 :0] add2_00,add2_01;

        reg signed [BW+1-1 :0] add0_10,add0_11;
        reg signed [BW+1-1 :0] add1_10,add1_11;
        reg signed [BW+1-1 :0] add2_10,add2_11;



    always @(posedge clock) begin
                                add0_00<=lp[0].w0+lp[1].w0;//3rd
                                add0_01<=lp[2].w0+lp[3].w0;//3rd

                                add0_10<=lp[4].w0+lp[5].w0;//3rd
                                add0_11<=lp[6].w0+lp[7].w0;//3rd

                                add0_0<=add0_00+add0_01;//4th
                                add0_1<=add0_10+add0_11;//4th
                                add0<=add0_0+add0_1;//5th
        end
    always @(posedge clock) begin
                                add1_00<=lp[0].w1+lp[1].w1;
                                add1_01<=lp[2].w1+lp[3].w1;

                                add1_10<=lp[4].w1+lp[5].w1;
                                add1_11<=lp[6].w1+lp[7].w1;

                                add1_0<=add1_00+add1_01;
                                add1_1<=add1_10+add1_11;
                                add1<=add1_0+add1_1;

        end
         always @(posedge clock) begin
                                add2_00<=lp[0].w2+lp[1].w2;
                                add2_01<=lp[2].w2+lp[3].w2;

                                add2_10<=lp[4].w2+lp[5].w2;
                                add2_11<=lp[6].w2+lp[7].w2;
```

```
                add2_0<=add2_00+add2_01;
                add2_1<=add2_10+add2_11;
                add2<=add2_0+add2_1;
                add2_D<=add2; //6th
        end

    generate
        genvar g;
        for (g=0;g <FIFO_DEPTH; g=g+1) begin :fifo
                reg [3:0] BinDD;
                reg [3:0] word0_D;
                reg [3:0] word1_D;
                reg [3:0] word2_D;

          if (g==0) begin :if_label //why xilinx needs label here?
                always @(posedge clock) begin
                        BinDD<=BinD;//2nd
                        word0_D<=word0[3:0];//2nd
                        word1_D<=word1[3:0];
                        word2_D<=word2[3:0];
                end
          end  else begin :else_label//why xilinx needs label here?
                always @(posedge clock) begin
                       BinDD<=fifo[g-1].BinDD;//
                        word0_D<=fifo[g-1].word0_D;//
                        word1_D<=fifo[g-1].word1_D;
                        word2_D<=fifo[g-1].word2_D;
                end

          end //if
        end //for
   endgenerate

//Parity Check
                reg [7:4] word0p_D;
                reg [7:4] word1p_D;
                reg [7:4] word2p_D;
        always @(posedge clock) begin

                        word0p_D<=word0[7:4];//2nd
                        word1p_D<=word1[7:4];
                        word2p_D<=word2[7:4];

                 p_res0_D<=codeword_check({word0p_D,fifo[0].word0_D});//3rd
                 p_res1_D<=codeword_check({word1p_D,fifo[0].word1_D});
                 p_res2_D<=codeword_check({word2p_D,fifo[0].word2_D});

                 //p_res0_D<=p_res0;//3rd
                 p_res0_DD<=p_res0_D;//4th
                 p_res0_DDD<=p_res0_DD;//5th

                 //p_res1_D<=p_res1;
                 p_res1_DD<=p_res1_D;
                 p_res1_DDD<=p_res1_DD;

                 //p_res2_D<=p_res2;
                 p_res2_DD<=p_res2_D;
                 p_res2_DDD<=p_res2_DD;
                 p_res2_DDDD<=p_res2_DDD;//6th

                 P_res_D<=P_res; //2nd
                 P_res_DD<=P_res_D;//3rd
                 P_res_DDD<=P_res_DD;//4th
                 P_res_DDDD<=P_res_DDD;//5th
                 P_res_DDDDD<=P_res_DDDD;//6th
        end



//6th Stage
        wire add0_is_less_or_equal=add0<= add1;
        always @(posedge clock) begin
                case ( {p_res1_DDD,p_res0_DDD} )
                        2'b00: begin
                                        comp0<={ 1'b0, {(BW+2) {1'b1}} } ;// どちらも codeword でないのプラス最大値を返す
                                        c0<=0;
                                end
                        2'b01: begin
                                        comp0<=add0;//0 のみ  Codeword
                                        c0<=0;
                                end
                        2'b10: begin
                                        comp0<=add1;//1 のみ  Codeword
                                        c0<=1;
                                end
                        default: begin
                                        c0<=add0_is_less_or_equal;//add==add1 => add0
                                        comp0<=add0_is_less_or_equal ? add0: add1;//どちらも codeword
                                 end
                endcase
```

```
	end

//7th
	always @(posedge clock) begin
		case ( {p_res2_DDDD,P_res_DDDDD} )
			default:  sel<=2'b00 ;// エラーなし、
			2'b00:    sel<={1'b1,c0};//2 は  Code  Word でないので、0 か1のどちらか
			2'b10:    if (add2_D <comp0) sel<=2'b01;// 2 が最小
                                  else sel<= {1'b1,c0};//0/1 の小さいほう
		endcase
	end

//8th
	always @(posedge clock) begin
		case (sel)
                        default: outdata<=fifo[FIFO_DEPTH-1].BinDD;
                        2'b11: outdata<=fifo[FIFO_DEPTH-1].word0_D;
                        2'b10: outdata<=fifo[FIFO_DEPTH-1].word1_D;
                        2'b01: outdata<=fifo[FIFO_DEPTH-1].word2_D;
		endcase
	end

	function [3:0] parity_check(input [7:0] pat);

		begin
                        parity_check[0]=pat[0] ^ pat[1] ^ pat[4];//パリティを計算
                        parity_check[1]=pat[2] ^ pat[3] ^ pat[5];//パリティを計算
                        parity_check[2]=pat[0] ^ pat[2] ^ pat[6];//パリティを計算
                        parity_check[3]=pat[1] ^ pat[3] ^ pat[7];//パリティを計算

		end
	endfunction

	function  codeword_check (input [7:0] pat);
		reg [3:0] parity;
		begin
                                                                parity=parity_check(pat);

                        codeword_check=parity ==4'b0000;
		end
	endfunction

	function [7:0] err_pos0 (input [3:0] parity_error);
		begin
			case(parity_error)
                                4'b0001://Peq=0 single bit parity error
                                        err_pos0=8'b0001_0000;
                                4'b0010://Peq=1 single bit parity error
                                        err_pos0=8'b0010_0000;
                                4'b0100://Peq=2 single bit parity error
                                        err_pos0=8'b0100_0000;
                                4'b1000://Peq=3 single bit parity error
                                        err_pos0=8'b1000_0000;
                                4'b0101://Peq=0Peq=2
                                        err_pos0=8'b0000_0001;//position j=        0
                                 4'b1001://Peq=0Peq=3
                                        err_pos0=8'b0000_0010;//position j=        1 ,
                                 4'b0110://Peq=1Peq=2
                                        err_pos0=8'b0000_0100;//position j=        2 ,
                                 4'b1010://Peq=1Peq=3
                                        err_pos0=8'b0000_1000;//position j=        3 ,
                                 4'b1100://Peq=2Peq=3
                                        err_pos0=8'b0000_0011;//position j=        0 k=        1,
                                 4'b0011://Peq=0Peq=1
                                        err_pos0=8'b0000_0101;// position j=        0 k=        2,
                                 4'b1111://Peq=0Peq=1Peq=2Peq=3
                                        err_pos0=8'b0000_1001;//position j=        0 k=        3,
                                 4'b0111://Peq=0Peq=1Peq=2
                                        err_pos0=8'b0001_0100;//position j=        2 k=        4,
                                 4'b1011://Peq=0Peq=1Peq=3
                                        err_pos0=8'b0010_0010;//position j=        1 k=        5,
                                 4'b1101://Peq=0Peq=2Peq=3
                                        err_pos0=8'b0100_0010;//position j=        1 k=        6,
                                 4'b1110://Peq=1Peq=2Peq=3
                                        err_pos0=8'b1000_0100;//position j=        2 k=        7,
                                default:
                                        err_pos0=8'b0000_0000;
			endcase
		end
	endfunction

	function [7:0] err_pos1 (input [3:0] parity_error);
		begin
			case(parity_error)
                                4'b0001://Peq=0 single bit parity error
                                        err_pos1=8'b0100_0001;//position j=        0 k=        6
                                4'b0010://Peq=1 single bit parity error
                                        err_pos1=8'b0100_0100;//position j=        2 k=        6
                                4'b0100://Peq=2 single bit parity error
                                        err_pos1=8'b0010_0100;//position j=        2 k=        5,
```

```
                                        4'b1000://Peq=3 single bit parity error
                                                err_pos1=8'b0001_0010;//position j=      1 k=      4,
                                        4'b0101://Peq=0Peq=2
                                                err_pos1=8'b0101_0000;//position j=      4 k=      6,
                                        4'b1001://Peq=0Peq=3
                                                err_pos1=8'b1001_0000;//position j=      4 k=      7,
                                        4'b0110://Peq=1Peq=2
                                                err_pos1=8'b0110_0000;//position j=      5 k=      6,
                                        4'b1010://Peq=1Peq=3
                                                err_pos1=8'b1010_0000;//position j=      5 k=      7,
                                        4'b1100://Peq=2Peq=3
                                                err_pos1=8'b0000_1100;//position j=      2 k=      3,
                                        4'b0011://Peq=0Peq=1
                                                err_pos1=8'b0000_1010;//position j=      1 k=      3,
                                        4'b1111://Peq=0Peq=1Peq=2Peq=3
                                                err_pos1=8'b0000_0110;//position j=      1 k=      2,
                                        4'b0111://Peq=0Peq=1Peq=2
                                                err_pos1=8'b0010_0001;//position j=      0 k=      5,
                                        4'b1011://Peq=0Peq=1Peq=3
                                                err_pos1=8'b0001_1000;//position j=      3 k=      4,
                                        4'b1101://Peq=0Peq=2Peq=3
                                                err_pos1=8'b1000_0001;//position j=      0 k=      7,
                                        4'b1110://Peq=1Peq=2Peq=3
                                                err_pos1=8'b0100_1000;//position j=      3 k=      6,
                                        default:
                                                err_pos1=8'b0000_0000;
                                endcase
                        end
        endfunction

        function [7:0] err_pos2 (input [3:0] parity_error);
                        begin
                                case(parity_error)
                                        4'b0001://Peq=0 single bit parity error
                                                err_pos2=8'b1000_0010;//position j=      1 k=      7
                                        4'b0010://Peq=1 single bit parity error
                                                err_pos2=8'b1000_1000;//position j=      3 k=      7
                                        4'b0100://Peq=2 single bit parity error
                                                err_pos2=8'b0001_0001;//position j=      0 k=      4
                                        4'b1000://Peq=3 single bit parity error
                                                err_pos2=8'b0010_1000;//position j=      3 k=      5,
                                        4'b1100://Peq=2Peq=3
                                                err_pos2=8'b1100_0000;//position j=      6 k=      7,
                                        4'b0011://Peq=0Peq=1
                                                err_pos2=8'b0011_0000;//position j=      4 k=      5,
                                        default:
                                                err_pos2=8'b0000_0000;
                                endcase
                        end
        endfunction


endmodule
```

合成対象 Altera　用

//Verilog 2001　Generate 及び Local　Scope の変数宣言が Altera　Quartas5.1 では、合成不能だっため、Xilinx から必要な部分を 1995 に書き換えた。

```
module hardware # (parameter BW=3)
                        ( input [BW*8-1:0] indata,
                          output reg [3:0]  outdata,
                          input clock, Reset         );
        localparam integer FIFO_DEPTH=6;
        reg [BW*8-1:0] indataR,indataR_D;

        wire [7:0] Bin={indataR[BW*8-1], indataR[BW*7-1],indataR[BW*6-1],indataR[BW*5-1],
                                        indataR[BW*4-1], indataR[BW*3-1],indataR[BW*2-1],indataR[BW*1-1]};
        wire [3:0] parities=parity_check(Bin);

        wire [7:0] err_pat0=err_pos0(parities);
        wire [7:0] err_pat1=err_pos1(parities);
        wire [7:0] err_pat2=err_pos2(parities);

        reg p_res0,p_res1,p_res2,P_res;
        reg p_res0_D,p_res1_D,p_res2_D,P_res_D;
        reg p_res0_DD,p_res1_DD,p_res2_DD,P_res_DD;
        reg p_res0_DDD,p_res1_DDD,p_res2_DDD,P_res_DDD;

        reg p_res2_DDDD;
        reg P_res_DDDD,P_res_DDDDD;

        reg [7:0] word0,word1,word2;
        reg signed [BW-1+3:0] add0,add1,add2;
        reg signed  [BW-1+3:0] add2_D;
        reg [3:0] BinD;

        reg c0;
```

```
        reg signed [BW+3-1:0] comp0;
        reg [1:0] sel;


        always @(posedge clock) begin
                indataR<=indata;//Receiver Register
                indataR_D<=indataR;//1-st Stage
        end



//1-st Stage
        always @(posedge clock ,posedge Reset) begin
                if (Reset) word0<=0;
                else       word0<=err_pat0^Bin;
        end

        always @(posedge clock ,posedge Reset) begin
                if (Reset) word1<=0;
                else       word1<=err_pat1^Bin;
        end

        always @(posedge clock ,posedge Reset) begin
                if (Reset) word2<=0;
                else       word2<=err_pat2^Bin;
        end

        always @(posedge clock, posedge Reset) begin
                if (Reset) BinD<=0;
                else       BinD<=Bin[3:0];
        end

        always @(posedge clock, posedge Reset) begin
                if (Reset) P_res<=0;
                else       P_res<=parities==4'b0000;
        end

//2nd stage
/*      genvar ge;
        generate
                for (ge=0; ge<8 ; ge=ge+1) begin :lp
                        reg signed [BW-1:0] w0,w1,w2;
                        wire signed [BW-1:0] temp;

                                assign temp=indataR_D[(ge+1)*BW -1] ? indataR_D[ge*BW +:BW]+1'b1: indataR_D[ge*BW  +:BW] ;
                                always @ (posedge clock)  begin
                                        w0 <=!word0[ge] ? -temp :temp;
                                        w1 <=!word1[ge] ? -temp :temp;
                                        w2 <=!word2[ge] ? -temp :temp;
                                end

                end

        endgenerate */

    wire signed [BW-1 : 0] temp0,temp1,temp2,temp3,temp4,temp5,temp6,temp7;

    assign temp0=indataR_D[BW-1]? indataR_D[0 +:BW]+1'b1 : indataR_D[0 +:BW];

         reg signed [BW-1 :0] lp0_w0,lp0_w1,lp0_w2;
        reg signed [BW-1 :0] lp1_w0,lp1_w1,lp1_w2;
        reg signed [BW-1 :0] lp2_w0,lp2_w1,lp2_w2;
        reg signed [BW-1 :0] lp3_w0,lp3_w1,lp3_w2;

         reg signed [BW-1 :0] lp4_w0,lp4_w1,lp4_w2;
        reg signed [BW-1 :0] lp5_w0,lp5_w1,lp5_w2;
        reg signed [BW-1 :0] lp6_w0,lp6_w1,lp6_w2;
        reg signed [BW-1 :0] lp7_w0,lp7_w1,lp7_w2;
          always @ (posedge clock) begin

            lp0_w0 <= !word0[0] ? -temp0 :temp0;
            lp0_w1 <= !word1[0] ? -temp0 :temp0;
            lp0_w2 <= !word2[0] ? -temp0 :temp0;
    end

    assign temp1=indataR_D[2*BW-1]? indataR_D[1*BW +:BW]+1'b1 : indataR_D[BW +:BW];
    always @ (posedge clock) begin

            lp1_w0 <= !word0[1] ? -temp1 :temp1;
            lp1_w1 <= !word1[1] ? -temp1 :temp1;
            lp1_w2 <= !word2[1] ? -temp1 :temp1;
    end

    assign temp2=indataR_D[3*BW-1]? indataR_D[2*BW +:BW]+1'b1 : indataR_D[2*BW +:BW];
    always @ (posedge clock) begin

            lp2_w0 <= !word0[2] ? -temp2 :temp2;
            lp2_w1 <= !word1[2] ? -temp2 :temp2;
            lp2_w2 <= !word2[2] ? -temp2 :temp2;
    end
```

```
assign temp3=indataR_D[4*BW-1]? indataR_D[3*BW +:BW]+1'b1 : indataR_D[3*BW +:BW];
always @ (posedge clock) begin

        lp3_w0 <= !word0[3] ? -temp3 :temp3;
        lp3_w1 <= !word1[3] ? -temp3 :temp3;
        lp3_w2 <= !word2[3] ? -temp3 :temp3;
end

assign temp4=indataR_D[5*BW-1]? indataR_D[4*BW +:BW]+1'b1 : indataR_D[4*BW +:BW];
always @ (posedge clock) begin

        lp4_w0 <= !word0[4] ? -temp4 :temp4;
        lp4_w1 <= !word1[4] ? -temp4 :temp4;
        lp4_w2 <= !word2[4] ? -temp4 :temp4;
end

assign temp5=indataR_D[6*BW-1]? indataR_D[5*BW +:BW]+1'b1 : indataR_D[5*BW +:BW];
always @ (posedge clock) begin

        lp5_w0 <= !word0[5] ? -temp5 :temp5;
        lp5_w1 <= !word1[5] ? -temp5 :temp5;
        lp5_w2 <= !word2[5] ? -temp5 :temp5;
end

assign temp6=indataR_D[7*BW-1]? indataR_D[6*BW +:BW]+1'b1 : indataR_D[6*BW +:BW];
always @ (posedge clock) begin

        lp6_w0 <= !word0[6] ? -temp6 :temp6;
        lp6_w1 <= !word1[6] ? -temp6 :temp6;
        lp6_w2 <= !word2[6] ? -temp6 :temp6;
end

assign temp7=indataR_D[8*BW-1]? indataR_D[7*BW +:BW]+1'b1 : indataR_D[7*BW +:BW];
always @ (posedge clock) begin

        lp7_w0 <= !word0[7] ? -temp7 :temp7;
        lp7_w1 <= !word1[7] ? -temp7 :temp7;
        lp7_w2 <= !word2[7] ? -temp7 :temp7;
end



    reg signed [BW+2-1 :0] add0_0,add0_1;
    reg signed [BW+2-1 :0] add1_0,add1_1;
    reg signed [BW+2-1 :0] add2_0,add2_1;

    reg signed [BW+1-1 :0] add0_00,add0_01;
    reg signed [BW+1-1 :0] add1_00,add1_01;
    reg signed [BW+1-1 :0] add2_00,add2_01;

    reg signed [BW+1-1 :0] add0_10,add0_11;
    reg signed [BW+1-1 :0] add1_10,add1_11;
    reg signed [BW+1-1 :0] add2_10,add2_11;



always @(posedge clock) begin
                        add0_00<=lp0_w0+lp1_w0;//3rd
                        add0_01<=lp2_w0+lp3_w0;//3rd

                        add0_10<=lp4_w0+lp5_w0;//3rd
                        add0_11<=lp6_w0+lp7_w0;//3rd

                        add0_0<=add0_00+add0_01;//4th
                        add0_1<=add0_10+add0_11;//4th
                        add0<=add0_0+add0_1;//5th
    end
always @(posedge clock) begin
                        add1_00<=lp0_w1+lp1_w1;
                        add1_01<=lp2_w1+lp3_w1;

                        add1_10<=lp4_w1+lp5_w1;
                        add1_11<=lp6_w1+lp7_w1;

                        add1_0<=add1_00+add1_01;
                        add1_1<=add1_10+add1_11;
                        add1<=add1_0+add1_1;

    end
     always @(posedge clock) begin
                        add2_00<=lp0_w2+lp1_w2;
                        add2_01<=lp2_w2+lp3_w2;

                        add2_10<=lp4_w2+lp5_w2;
                        add2_11<=lp6_w2+lp7_w2;

                        add2_0<=add2_00+add2_01;
                        add2_1<=add2_10+add2_11;
                        add2<=add2_0+add2_1;
                        add2_D<=add2; //6th
    end
```

```
/*    generate
            genvar g;
            for (g=0;g <FIFO_DEPTH; g=g+1) begin :fifo
                        reg [3:0] BinDD;
                        reg [3:0] word0_D;
                        reg [3:0] word1_D;
                        reg [3:0] word2_D;

              if (g==0) begin :if_label //why xilinx needs label here?
                                    always @(posedge clock) begin
                                                BinDD<=BinD;//2nd
                                                word0_D<=word0[3:0];//2nd
                                                word1_D<=word1[3:0];
                                                word2_D<=word2[3:0];
                        end
              end  else begin :else_label//why xilinx needs label here?
                                    always @(posedge clock) begin
                                               BinDD<=fifo[g-1].BinDD;//
                                                word0_D<=fifo[g-1].word0_D;//
                                                word1_D<=fifo[g-1].word1_D;
                                                word2_D<=fifo[g-1].word2_D;
                                    end

              end //if
            end //for
   endgenerate */
            reg [3:0] fifo0_word0_D;
            reg [3:0] fifo0_word1_D;
            reg [3:0] fifo0_word2_D;
            reg [3:0] fifo0_BinDD;

            reg [3:0] fifo1_word0_D;
            reg [3:0] fifo1_word1_D;
            reg [3:0] fifo1_word2_D;
            reg [3:0] fifo1_BinDD;

            reg [3:0] fifo2_word0_D;
            reg [3:0] fifo2_word1_D;
            reg [3:0] fifo2_word2_D;
            reg [3:0] fifo2_BinDD;

            reg [3:0] fifo3_word0_D;
            reg [3:0] fifo3_word1_D;
            reg [3:0] fifo3_word2_D;
            reg [3:0] fifo3_BinDD;

            reg [3:0] fifo4_word0_D;
            reg [3:0] fifo4_word1_D;
            reg [3:0] fifo4_word2_D;
            reg [3:0] fifo4_BinDD;

            reg [3:0] fifo5_word0_D;
            reg [3:0] fifo5_word1_D;
            reg [3:0] fifo5_word2_D;
            reg [3:0] fifo5_BinDD;

            always @(posedge clock) begin


                        fifo0_BinDD<=BinD;//2nd
                        fifo0_word0_D<=word0[3:0];//2nd
                        fifo0_word1_D<=word1[3:0];
                        fifo0_word2_D<=word2[3:0];
            end

            always @(posedge clock) begin

                         fifo1_BinDD<=fifo0_BinDD;//
                        fifo1_word0_D<=fifo0_word0_D;//
                        fifo1_word1_D<=fifo0_word1_D;
                        fifo1_word2_D<=fifo0_word2_D;
            end

            always @(posedge clock) begin

                         fifo2_BinDD<=fifo1_BinDD;//
                         fifo2_word0_D<=fifo1_word0_D;//
                         fifo2_word1_D<=fifo1_word1_D;
                         fifo2_word2_D<=fifo1_word2_D;
            end

            always @(posedge clock) begin

                        fifo3_BinDD<=fifo2_BinDD;//
                         fifo3_word0_D<=fifo2_word0_D;//
                         fifo3_word1_D<=fifo2_word1_D;
                         fifo3_word2_D<=fifo2_word2_D;
            end
```

```
	always @(posedge clock) begin

		fifo4_BinDD<=fifo3_BinDD;//
		fifo4_word0_D<=fifo3_word0_D;//
		fifo4_word1_D<=fifo3_word1_D;
		fifo4_word2_D<=fifo3_word2_D;
	end

	always @(posedge clock) begin

		fifo5_BinDD<=fifo4_BinDD;//
		fifo5_word0_D<=fifo4_word0_D;//
		fifo5_word1_D<=fifo4_word1_D;
		fifo5_word2_D<=fifo4_word2_D;
	end



//Parity Check
		reg [7:4] word0p_D;
		reg [7:4] word1p_D;
		reg [7:4] word2p_D;
	always @(posedge clock) begin

			word0p_D<=word0[7:4];//2nd
			word1p_D<=word1[7:4];
			word2p_D<=word2[7:4];

		p_res0_D<=codeword_check({word0p_D,fifo0_word0_D});//3rd
		p_res1_D<=codeword_check({word1p_D,fifo0_word1_D});
		p_res2_D<=codeword_check({word2p_D,fifo0_word2_D});

		//p_res0_D<=p_res0;//3rd
		p_res0_DD<=p_res0_D;//4th
		p_res0_DDD<=p_res0_DD;//5th

		//p_res1_D<=p_res1;
		p_res1_DD<=p_res1_D;
		p_res1_DDD<=p_res1_DD;

		//p_res2_D<=p_res2;
		p_res2_DD<=p_res2_D;
		p_res2_DDD<=p_res2_DD;
		p_res2_DDDD<=p_res2_DDD;//6th

		P_res_D<=P_res; //2nd
		P_res_DD<=P_res_D;//3rd
		P_res_DDD<=P_res_DD;//4th
		P_res_DDDD<=P_res_DDD;//5th
		P_res_DDDDD<=P_res_DDDD;//6th
	end



//6th Stage
	wire add0_is_less_or_equal=add0<= add1;
	always @(posedge clock) begin
		case ( {p_res1_DDD,p_res0_DDD} )
			2'b00: begin
					comp0<={ 1'b0, {(BW+2) {1'b1}} } ;// どちらも codeword でないのプラス最大値を返す
					c0<=0;
				end
			2'b01: begin
					comp0<=add0;//0 のみ  Codeword
					c0<=0;
				end
			2'b10: begin
					comp0<=add1;//1 のみ  Codeword
					c0<=1;
				end
			default: begin
					c0<=add0_is_less_or_equal;//add==add1 => add0
					comp0<=add0_is_less_or_equal ? add0: add1;//どちらも codeword
				 end
		endcase
	end

//7th
	always @(posedge clock) begin
		case ( {p_res2_DDDD,P_res_DDDDD} )
			default:  sel<=2'b00 ;// エラーなし、
			2'b00:    sel<={1'b1,c0};//2 は  Code  Word でないので、0 か1のどちらか
			2'b10:    if (add2_D <comp0) sel<=2'b01;// 2 が最小
					else sel<= {1'b1,c0};//0/1 の小さいほう
		endcase
	end

//8th
	always @(posedge clock) begin
		case (sel)
```

```
                                        default: outdata<=fifo5_BinDD;
                                        2'b11: outdata<=fifo5_word0_D;
                                        2'b10: outdata<=fifo5_word1_D;
                                        2'b01: outdata<=fifo5_word2_D;
                endcase
end

function [3:0] parity_check(input [7:0] pat);

                begin
                                        parity_check[0]=pat[0] ^ pat[1] ^ pat[4];//パリティを計算
                                        parity_check[1]=pat[2] ^ pat[3] ^ pat[5];//パリティを計算
                                        parity_check[2]=pat[0] ^ pat[2] ^ pat[6];//パリティを計算
                                        parity_check[3]=pat[1] ^ pat[3] ^ pat[7];//パリティを計算

                end
endfunction

function  codeword_check (input [7:0] pat);
                reg [3:0] parity;
                begin
                                                                                        parity=parity_check(pat);

                                        codeword_check=parity ==4'b0000;
                end
endfunction

function [7:0] err_pos0 (input [3:0] parity_error);
                begin
                                case(parity_error)
                                        4'b0001://Peq=0 single bit parity error
                                                err_pos0=8'b0001_0000;
                                        4'b0010://Peq=1 single bit parity error
                                                err_pos0=8'b0010_0000;
                                        4'b0100://Peq=2 single bit parity error
                                                err_pos0=8'b0100_0000;
                                        4'b1000://Peq=3 single bit parity error
                                                err_pos0=8'b1000_0000;
                                        4'b0101://Peq=0Peq=2
                                                err_pos0=8'b0000_0001;//position j=        0
                                         4'b1001://Peq=0Peq=3
                                                err_pos0=8'b0000_0010;//position j=        1 ,
                                         4'b0110://Peq=1Peq=2
                                                err_pos0=8'b0000_0100;//position j=        2 ,
                                         4'b1010://Peq=1Peq=3
                                                err_pos0=8'b0000_1000;//position j=        3 ,
                                         4'b1100://Peq=2Peq=3
                                                err_pos0=8'b0000_0011;//position j=        0 k=        1,
                                         4'b0011://Peq=0Peq=1
                                                err_pos0=8'b0000_0101;// position j=        0 k=        2,
                                         4'b1111://Peq=0Peq=1Peq=2Peq=3
                                                err_pos0=8'b0000_1001;//position j=        0 k=        3,
                                         4'b0111://Peq=0Peq=1Peq=2
                                                err_pos0=8'b0001_0100;//position j=        2 k=        4,
                                         4'b1011://Peq=0Peq=1Peq=3
                                                err_pos0=8'b0010_0010;//position j=        1 k=        5,
                                         4'b1101://Peq=0Peq=2Peq=3
                                                err_pos0=8'b0100_0010;//position j=        1 k=        6,
                                         4'b1110://Peq=1Peq=2Peq=3
                                                err_pos0=8'b1000_0100;//position j=        2 k=        7,
                                        default:
                                                err_pos0=8'b0000_0000;
                                endcase
                end
endfunction

function [7:0] err_pos1 (input [3:0] parity_error);
                begin
                                case(parity_error)
                                        4'b0001://Peq=0 single bit parity error
                                                err_pos1=8'b0100_0001;//position j=        0 k=        6
                                        4'b0010://Peq=1 single bit parity error
                                                err_pos1=8'b0100_0100;//position j=        2 k=        6
                                        4'b0100://Peq=2 single bit parity error
                                                err_pos1=8'b0010_0100;//position j=        2 k=        5,
                                        4'b1000://Peq=3 single bit parity error
                                                err_pos1=8'b0001_0010;//position j=        1 k=        4,
                                        4'b0101://Peq=0Peq=2
                                                err_pos1=8'b0101_0000;//position j=        4 k=        6,
                                        4'b1001://Peq=0Peq=3
                                                err_pos1=8'b1001_0000;//position j=        4 k=        7,
                                         4'b0110://Peq=1Peq=2
                                                err_pos1=8'b0110_0000;//position j=        5 k=        6,
                                         4'b1010://Peq=1Peq=3
                                                err_pos1=8'b1010_0000;//position j=        5 k=        7,
                                         4'b1100://Peq=2Peq=3
                                                err_pos1=8'b0000_1100;//position j=        2 k=        3,
                                         4'b0011://Peq=0Peq=1
                                                err_pos1=8'b0000_1010;//position j=        1 k=        3,
                                         4'b1111://Peq=0Peq=1Peq=2Peq=3
```

```
                                        err_pos1=8'b0000_0110;//position j=      1 k=      2,
                                4'b0111://Peq=0Peq=1Peq=2
                                        err_pos1=8'b0010_0001;//position j=      0 k=      5,
                                4'b1011://Peq=0Peq=1Peq=3
                                        err_pos1=8'b0001_1000;//position j=      3 k=      4,
                                4'b1101://Peq=0Peq=2Peq=3
                                        err_pos1=8'b1000_0001;//position j=      0 k=      7,
                                4'b1110://Peq=1Peq=2Peq=3
                                        err_pos1=8'b0100_1000;//position j=      3 k=      6,
                                default:
                                        err_pos1=8'b0000_0000;
                        endcase
                end
        endfunction

        function [7:0] err_pos2 (input [3:0] parity_error);
                begin
                        case(parity_error)
                                4'b0001://Peq=0 single bit parity error
                                        err_pos2=8'b1000_0010;//position j=      1 k=      7
                                4'b0010://Peq=1 single bit parity error
                                        err_pos2=8'b1000_1000;//position j=      3 k=      7
                                4'b0100://Peq=2 single bit parity error
                                        err_pos2=8'b0001_0001;//position j=      0 k=      4
                                4'b1000://Peq=3 single bit parity error
                                        err_pos2=8'b0010_1000;//position j=      3 k=      5,
                                 4'b1100://Peq=2Peq=3
                                        err_pos2=8'b1100_0000;//position j=      6 k=      7,
                                 4'b0011://Peq=0Peq=1
                                        err_pos2=8'b0011_0000;//position j=      4 k=      5,
                                default:
                                        err_pos2=8'b0000_0000;
                        endcase
                end
        endfunction


endmodule
```

プログラム３　１ビット、２ビットエラーによるシンドローム検討

```
//課題の符号を全部列挙する
//1bit/2bit エラーの全ての組み合わせを起こしシンドロームを調べる
module dwm2_test;

	reg [3:0] data;//Encode する元 Data の WORK
	reg [7:0] code_array[0:15];//全コードワードを格納する
	reg [7:0] work,a,b,c,d;

	wire [7:0] encoded_data;//encoder からの出力を受け取る
	integer i,j,flag,m,k,n;

	encoder en1(data,encoded_data);

	initial begin
		//符号の列挙
			for (i=0;i<16;i=i+1) begin//
				data=i[3:0];
				#10;
				code_array[i]=encoded_data;//配列に格納
				$display("Code [%d]=%b",i,encoded_data);
			end
		//CODE  Word のビットの任意1ビットを変化させて別符号語に一致するかを見る。
			flag=0;
			$display("1 ビットエラーをチェックしています。");
			for (i=0;i<1;i=i+1) begin//
				work=code_array[i];//元の Data に
				for ( j=0;j<8;j=j+1) begin
					a=work ^ (1 <<j);//1bit  エラーを乗せて
					$write("position j=%d ,",j);
					if(a[0]^a[1]^a[4])  $write("Peq=0");
					if(a[2]^a[3]^a[5])  $write("Peq=1");
					if(a[0]^a[2]^a[6])  $write("Peq=2");
					if(a[1]^a[3]^a[7])  $write("Peq=3");
					$display("");
				end
			end
			$display( "1 ビットエラー終わり");
		//CODE  Word のビットの任意2ビットを変化させて別符号に一致するかを見る。
			flag=0;
			$display("2 ビットエラーをチェックしています。");
			for (i=0;i<1;i=i+1) begin//
				work=code_array[i];//元の Data に
				for ( j=0;j<8;j=j+1) begin
					a=work ^ (1 <<j);//1bit  エラーを乗せて

				    for (k=j ; k<8;k=k+1) begin
						b=a ^(1<< k);//もう 1 ビットエラーを乗せて
						$write("position j=%d k=%d,",j,k);
						if (work !=b) begin//同じ符号は除いて
							if(b[0]^b[1]^b[4]) $write("Peq=0");
							if(b[2]^b[3]^b[5]) $write("Peq=1");
							if(b[0]^b[2]^b[6]) $write("Peq=2");
							if(b[1]^b[3]^b[7]) $write("Peq=3");
						end
						$display("");
					end
				end
			end
			$display("2 ビットエラー終わり");
	end



endmodule
module encoder(input [3:0] in,
			output wire [7:0]  out);
		reg r0,r1,c0,c1;
		wire [3:0] y=in[3:0];

		assign out[3:0]=in;
		assign out[4]=r0;
		assign out[5]=r1;
		assign out[6]=c0;
		assign out[7]=c1;

	always @* begin
```

```
		r0=y[0] ^ y[1];
		r1=y[2] ^ y[3];
		c0=y[0] ^ y[2];
		c1=y[1] ^ y[3];
end

function [7:0] err_pos0 (input [3:0] parity_error);
		begin
			case(parity_error)
				4'b0001://Peq=0 single bit parity error
					err_pos0=8'b0001_0000;
				4'b0010://Peq=1 single bit parity error
					err_pos0=8'b0010_0000;
				4'b0100://Peq=2 single bit parity error
					err_pos0=8'b0100_0000;
				4'b1000://Peq=3 single bit parity error
					err_pos0=8'b1000_0000;
				4'b0101://Peq=0Peq=2
					err_pos0=8'b0000_0001;//position j=	0
				 4'b1001://Peq=0Peq=3
					err_pos0=8'b0000_0010;//position j=	1 ,
				 4'b0110://Peq=1Peq=2
					err_pos0=8'b0000_0100;//position j=	2 ,
				 4'b1010://Peq=1Peq=3
					err_pos0=8'b0000_1000;//position j=	3 ,
				 4'b1100://Peq=2Peq=3
					err_pos0=8'b0000_0011;//position j=	0 k=	1,
				 4'b0011://Peq=0Peq=1
					err_pos0=8'b0000_0101;// position j=	0 k=	2,
				 4'b1111://Peq=0Peq=1Peq=2Peq=3
					err_pos0=8'b0000_1001;//position j=	0 k=	3,
				 4'b0111://Peq=0Peq=1Peq=2
					err_pos0=8'b0001_0100;//position j=	2 k=	4,
				 4'b1011://Peq=0Peq=1Peq=3
					err_pos0=8'b0010_0010;//position j=	1 k=	5,
				 4'b1101://Peq=0Peq=2Peq=3
					err_pos0=8'b0100_0010;//position j=	1 k=	6,
				 4'b1110://Peq=1Peq=2Peq=3
					err_pos0=8'b1000_0100;//position j=	2 k=	7,
				default:
					err_pos0=8'b0000_0000;
			endcase
		end
endfunction

function [7:0] err_pos1 (input [3:0] parity_error);
		begin
			case(parity_error)
				4'b0001://Peq=0 single bit parity error
					err_pos1=8'b0100_0001;//position j=	0 k=	6
				4'b0010://Peq=1 single bit parity error
					err_pos1=8'b0100_0100;//position j=	2 k=	6
				4'b0100://Peq=2 single bit parity error
					err_pos1=8'b0010_0100;//position j=	2 k=	5,
				4'b1000://Peq=3 single bit parity error
					err_pos1=8'b0001_0010;//position j=	1 k=	4,
				4'b0101://Peq=0Peq=2
					err_pos1=8'b0101_0000;//position j=	4 k=	6,
				4'b1001://Peq=0Peq=3
					err_pos1=8'b1001_0000;//position j=	4 k=	7,
				 4'b0110://Peq=1Peq=2
					err_pos1=8'b0110_0000;//position j=	5 k=	6,
				 4'b1010://Peq=1Peq=3
					err_pos1=8'b1010_0000;//position j=	5 k=	7,
				 4'b1100://Peq=2Peq=3
					err_pos1=8'b0000_1100;//position j=	2 k=	3,
				 4'b0011://Peq=0Peq=1
					err_pos1=8'b0000_1010;//position j=	1 k=	3,
				 4'b1111://Peq=0Peq=1Peq=2Peq=3
					err_pos1=8'b0000_0110;//position j=	1 k=	2,
				 4'b0111://Peq=0Peq=1Peq=2
					err_pos1=8'b0010_0001;//position j=	0 k=	5,
				 4'b1011://Peq=0Peq=1Peq=3
					err_pos1=8'b0001_1000;//position j=	3 k=	4,
				 4'b1101://Peq=0Peq=2Peq=3
					err_pos1=8'b1000_0001;//position j=	0 k=	7,
				 4'b1110://Peq=1Peq=2Peq=3
					err_pos1=8'b0100_1000;//position j=	3 k=	6,
				default:
					err_pos1=8'b0000_0000;
			endcase
		end
endfunction

function [7:0] err_pos2 (input [3:0] parity_error);
		begin
			case(parity_error)
				4'b0001://Peq=0 single bit parity error
					err_pos2=8'b1000_0010;//position j=	1 k=	7
				4'b0010://Peq=1 single bit parity error
```

```
                                                err_pos2=8'b1000_1000;//position j=      3 k=      7
                                        4'b0100://Peq=2 single bit parity error
                                                err_pos2=8'b0001_0001;//position j=      0 k=      4
                                        4'b1000://Peq=3 single bit parity error
                                                err_pos2=8'b0010_1000;//position j=      3 k=      5,
                                         4'b1100://Peq=2Peq=3
                                                err_pos2=8'b1100_0000;//position j=      6 k=      7,
                                         4'b0011://Peq=0Peq=1
                                                err_pos2=8'b0011_0000;//position j=      4 k=      5,
                                        default:
                                                        err_pos2=8'b0000_0000;
                                endcase
                        end
                endfunction



endmodule
/*
F:\regression_test\ldgm\dwm_test2.v(3)::dwm2_test
Verilog のシミュレーションの準備が完了しました。スタートは,Go ボタンを押してください。
---------- シミュレーションを開始します。----------

Code [         0]=00000000
Code [         1]=01010001
Code [         2]=10010010
Code [         3]=11000011
Code [         4]=01100100
Code [         5]=00110101
Code [         6]=11110110
Code [         7]=10100111
Code [         8]=10101000
Code [         9]=11111001
Code [        10]=00111010
Code [        11]=01101011
Code [        12]=11001100
Code [        13]=10011101
Code [        14]=01011110
Code [        15]=00001111
1 ビットエラーをチェックしています。
position j=       0 ,Peq=0Peq=2
position j=       1 ,Peq=0Peq=3
position j=       2 ,Peq=1Peq=2
position j=       3 ,Peq=1Peq=3
position j=       4 ,Peq=0
position j=       5 ,Peq=1
position j=       6 ,Peq=2
position j=       7 ,Peq=3
1 ビットエラー終わり
2 ビットエラーをチェックしています。
position j=       0 k=      0,
position j=       0 k=      1,Peq=2Peq=3
position j=       0 k=      2,Peq=0Peq=1
position j=       0 k=      3,Peq=0Peq=1Peq=2Peq=3
position j=       0 k=      4,Peq=2
position j=       0 k=      5,Peq=0Peq=1Peq=2
position j=       0 k=      6,Peq=0
position j=       0 k=      7,Peq=0Peq=2Peq=3
position j=       1 k=      1,
position j=       1 k=      2,Peq=0Peq=1Peq=2Peq=3
position j=       1 k=      3,Peq=0Peq=1
position j=       1 k=      4,Peq=3
position j=       1 k=      5,Peq=0Peq=1Peq=3
position j=       1 k=      6,Peq=0Peq=2Peq=3
position j=       1 k=      7,Peq=0
position j=       2 k=      2,
position j=       2 k=      3,Peq=2Peq=3
position j=       2 k=      4,Peq=0Peq=1Peq=2
position j=       2 k=      5,Peq=2
position j=       2 k=      6,Peq=1
position j=       2 k=      7,Peq=1Peq=2Peq=3
position j=       3 k=      3,
position j=       3 k=      4,Peq=0Peq=1Peq=3
position j=       3 k=      5,Peq=3
position j=       3 k=      6,Peq=1Peq=2Peq=3
position j=       3 k=      7,Peq=1
position j=       4 k=      4,
position j=       4 k=      5,Peq=0Peq=1
position j=       4 k=      6,Peq=0Peq=2
position j=       4 k=      7,Peq=0Peq=3
position j=       5 k=      5,
position j=       5 k=      6,Peq=1Peq=2
position j=       5 k=      7,Peq=1Peq=3
position j=       6 k=      6,
position j=       6 k=      7,Peq=2Peq=3
position j=       7 k=      7,
2 ビットエラー終わり

---------- シミュレーションを終了します。time=160----------
Peq=0
```

position j= 4
position j= 0 k= 6
position j= 1 k= 7

Peq=1
position j= 5 ,
position j= 2 k= 6
position j= 3 k= 7

Req=2
position j= 6 ,
position j= 2 k= 5,
position j= 0 k= 4

Peq=3
position j= 7 ,
position j= 1 k= 4,
position j= 3 k= 5,

Peq=0Peq=2
position j= 0 ,
position j= 4 k= 6,

Peq=0Peq=3
position j= 1 ,
position j= 4 k= 7,

Peq=1Peq=2
position j= 2 ,
position j= 5 k= 6,

Peq=1Peq=3
position j= 3 ,
position j= 5 k= 7,

Peq=2Peq=3
position j= 0 k= 1,
position j= 2 k= 3,
position j= 6 k= 7,

Peq=0Peq=1
position j= 0 k= 2,
position j= 1 k= 3,
position j= 4 k= 5,

Peq=0Peq=1Peq=2Peq=3
position j= 0 k= 3,
position j= 1 k= 2,

Peq=0Peq=1Peq=2
position j= 2 k= 4,
position j= 0 k= 5,

Peq=0Peq=2Peq=3
position j= 1 k= 6,
position j= 0 k= 7,

Peq=0Peq=1Peq=3
position j= 1 k= 5,
position j= 3 k= 4,

Peq=1Peq=2Peq=3
position j= 2 k= 7,
position j= 3 k= 6,

*/

プログラム４　AWGN 度数分布及び実機 C プログラム

```
/*
 Nov.29.2005 Tak.Sugawara :Use taus88 as uniform random generator.

  gauss.c - gaussian random numbers, using the Ziggurat method
 *
 * Copyright (C) 2005  Jochen Voss.
 *
 * For details see the following article.
 *
 *    George Marsaglia, Wai Wan Tsang
 *    The Ziggurat Method for Generating Random Variables
 *    Journal of Statistical Software, vol. 5 (2000), no. 8
 *    http://www.jstatsoft.org/v05/i08/
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * $Id: gauss.c 6509 2005-07-07 18:31:10Z voss $
 */

#include <math.h>
#ifdef DOS
#include <stdlib.h>
#endif
//#include <assert.h>

//#include <gsl/gsl_rng.h>


/* position of right-most step */

#ifndef RAM32K//16KB

                #define print_port      0x8000  //
    #define print_char_port 0x8001
    #define print_int_port  0x8002  //
    #define print_num_port 0x8004  //

                #define int_set_address 0x8008 //
                #define uart_port 0x800c //
                #define uart_wport uart_port
                #define uart_rport uart_port

                 #define  REG_PORT_START_ADDRESS  0x8010 //Your Register File Starts from here
                 #define  REG_PORT_END_ADDRESS    0x8010 //  to here

                 #define HARD_BLOCK_START_ADDRESS     0x8000  //Start Address of Hardware Block
                 #define HARD_BLOCK_END_ADDRESS       0x8fff  //Start Address of Hardware Block



#else// RAM32K
                #define print_port                      0x7ff0
                #define print_char_port 0x7ff1
                #define print_int_port  0x7ff2
                #define print_num_port 0x7ff4




                #define uart_port                       0x07ffc //for 16KRAM
                #define uart_wport uart_port
                #define uart_rport uart_port
                #define int_set_address 0x07ff8 //for 16KRAM
```

```
#endif

void print_num(unsigned);
void print_uart(unsigned char* ptr);//



/* automatically generated by ziggurat.c */

/* number of STATISTICS_LEVELS in the ziggurat */
#define LEVELS 32

/* position of right-most step */
#define PARAM_R 2.96454468156

/* level values */
static const double ytab[32] = {
  1,
  0.907475617638,
  0.8387295158,
  0.780775785072,
  0.729446868113,
  0.682776316768,
  0.639647716015,
  0.599353201849,
  0.561409662765,
  0.525469625896,
  0.491273228111,
  0.458620229136,
  0.427352696597,
  0.397343775955,
  0.368490127285,
  0.340706676394,
  0.313922886649,
  0.288080067698,
  0.263129417761,
  0.23903060658,
  0.215750777731,
  0.193263899438,
  0.171550433862,
  0.150597335281,
  0.130398439093,
  0.110955385052,
  0.0922793705114,
  0.0743943543072,
  0.0573431209268,
  0.0411998625242,
  0.0261009770388,
  0.0123479825627,
};

/* quick acceptance check */
static const unsigned long ktab[32] = {
  0,
  12465559,
  14143380,
  14859376,
  15254596,
  15503949,
  15674743,
  15798352,
  15891342,
  15963265,
  16020003,
  16065358,
  16101882,
  16131333,
  16154939,
  16173564,
  16187797,
  16198017,
  16204425,
  16207053,
  16205755,
  16200181,
  16189707,
  16173323,
  16149413,
  16115345,
  16066638,
  15995077,
  15883832,
  15691829,
  15281721,
  15063248
};

/* quick value conversion */
```

```
static const double wtab[32] = {
  2.6265107239e-08,
  3.5349827263e-08,
  4.19328107856e-08,
  4.73449074891e-08,
  5.20705831605e-08,
  5.63468970139e-08,
  6.03100167664e-08,
  6.40468149156e-08,
  6.76171489147e-08,
  7.10648776583e-08,
  7.44238783956e-08,
  7.77216053723e-08,
  8.09813473119e-08,
  8.42237615336e-08,
  8.74679979284e-08,
  9.0732597892e-08,
  9.4036290297e-08,
  9.73987784303e-08,
  1.00841606117e-07,
  1.04389204949e-07,
  1.08070261837e-07,
  1.11919620405e-07,
  1.15981071511e-07,
  1.20311664676e-07,
  1.24988739931e-07,
  1.30122131768e-07,
  1.35877026736e-07,
  1.42521234138e-07,
  1.50537318078e-07,
  1.60949814958e-07,
  1.76700632665e-07,
  1.96806467179e-07
};



unsigned long s1=0xffffffff,s2=0xffffffff,s3=0xffffffff;
unsigned long taus88_int ()
{/*Generates numbers between 0 and 1.*/
                unsigned b;

                b =(((s1 <<13)^s1)>>19);
                s1 =(((s1 &4294967294)<<12)^b);
                b =(((s2 <<2)^s2)>>25);
                s2 =(((s2 &4294967288)<<4)^b);
                b =(((s3 <<3)^s3)>>11);
                s3 =(((s3 &4294967280)<<17)^b);
                return ((s1 ^s2 ^s3));
}

double taus88 ()
{
                return (taus88_int()*2.3283064365e-10);
}



double
gsl_ran_gaussian_ziggurat ()
{
  unsigned long  U, sign, i, j;
  double  x, y;

  while (1) {
    U = taus88_int();
 //   i = U & 0x0000007F;                        /* 7 bit to choose the step */
 //   sign = U & 0x00000080;             /* 1 bit for the sign */
    i = U & 0x0000001F;                        /* 7 bit to choose the step */
    sign = U & 0x00000020;             /* 1 bit for the sign */

   // j = U>>8;                                  /* 24 bit for the x-value */
    j = U>>6;                                  /* 24 bit for the x-value */

    x = j*wtab[i];
    if (j < ktab[i])  break;

  //if (i<127) {
     if (i<32) {

       double  y0, y1;
       y0 = ytab[i];
       y1 = ytab[i+1];
       y = y1+(y0-y1)*taus88();
    } else {
       x = PARAM_R - log(1.0-taus88())/PARAM_R;
       y = exp(-PARAM_R*(x-0.5*PARAM_R))*taus88();
    }
    if (y < exp(-0.5*x*x))  break;
```

```
  }
  return  sign ? x : -x;
}


#define No 1000
#define SIGMA_LIMIT  8
#define ERROR_THRESHOLD 1000
#define STATISTICS_LEVELS 32

  #ifdef STTISTICS
        unsigned counter_array[STATISTICS_LEVELS*2];
  #endif
        unsigned bit_error_counter;
        unsigned raw_bit_error_counter;
        unsigned bit_counter;
        unsigned total_bits;

#ifdef DOS

        #ifdef SD_DISPLAY
                double array[No];
        #endif
#endif

unsigned quantize(double rvalue)
{
        double R;
        int qdata;
        R=rvalue *2;
//      R=rint(R);
        qdata=R;
        if (rvalue<0) qdata -=1;//負値のマッピング
        if (qdata>3) qdata=3;//飽和演算
        if (qdata<-4) qdata=-4;//飽和演算
        qdata &=0x7;//3bit 切り出し
        return qdata;

}
#ifdef STATISTICS
void statistics(double noise)
{
                int index;
                unsigned I;

                index=(noise/SIGMA_LIMIT)*STATISTICS_LEVELS;
                if (index>=STATISTICS_LEVELS) index=STATISTICS_LEVELS-1;
                if (noise<0) index -=1;
                if (index<=-STATISTICS_LEVELS) index=-STATISTICS_LEVELS+1;
                I=index+STATISTICS_LEVELS;
                counter_array[I] +=1;

                #ifdef SD_DISPLAY

                        array[i]=noise;
                        sum +=array[i];
                #endif

}
#endif

unsigned get_random_data()
{
        unsigned y;
        unsigned r0,r1,c0,c1;
        unsigned y0,y1,y2,y3;
        unsigned outdata;

                y=taus88_int();//ランダムデータを得る。下位 4 ビットしか使っていない
        //ビット切り出し
                y0=y & 0x01;
                y1=(y >> 1) & 0x01;
                y2=(y >> 2) & 0x01;
                y3=(y >> 3) & 0x01;

        //パリティ計算
                r0=y0 ^ y1;
                r1=y2 ^ y3;
                c0=y0 ^ y2;
                c1=y1 ^ y3;

                outdata=y0;
                outdata |=y1 <<1;
                outdata |=y2 <<2;
                outdata |=y3 <<3;
                outdata |=r0 <<4;
                outdata |=r1 <<5;
                outdata |=c0 <<6;
                outdata |=c1 <<7;
                outdata <<=24;// 上位 8 ビットが訂正前ノイズ加算前エンコードデータ
```

```
                return outdata;
}

unsigned make_input_data(double target_sd)
{

        double rvalue;
        unsigned temp;
        unsigned i;
        double noise;
        unsigned outdata;

                outdata=get_random_data();

                //3 ビットアナログ受信値の作成
                for (i=0;i<8 ;i++) {

                        noise=gsl_ran_gaussian_ziggurat();//1.0 ガウスノイズ生成
                        #ifdef STATISTICS
                                statistics(noise);//ガウスノイズの度数分布処理
                        #endif
                        noise *=target_sd;//ノイズシグマ乗算
                        if ( (outdata >> (24+i)) & 0x01) {
                                rvalue=-1.0;//1-> -1 に
                        }else rvalue=+1.0;//0-> +1 にエンコード
                        rvalue +=noise;
                        temp=quantize(rvalue);//3 ビットに量子化
                        temp <<=3*i;
                        outdata |=temp;
                }
                return outdata;
}

void data_write_and_compare(unsigned data_in)
{
        unsigned i;
        unsigned data_out;
        unsigned temp,temp2;
        *(volatile unsigned*)REG_PORT_START_ADDRESS=data_in ;//書き込み
        //wait 9 clocks
        {//準備ができるまで、生エラーをカウント
                temp=data_in;
                temp2=data_in>>2;//MSB を持ってくる
                for (i=0;i <4 ;i++){
                        temp=data_in >> (24+i);
                        if ( (temp ^temp2) & 0x01) ++raw_bit_error_counter;
                        temp2 >>=3;
                }
                data_in >>=24;
        }
        data_out=*(volatile unsigned*)          REG_PORT_START_ADDRESS;//読み出し


//      if (data_out !=data_in){
                data_out ^=data_in;
                for (i=0;i <4;i++) {
                        if ( (data_out >>i) & 0x01) ++bit_error_counter ;

                }
//      }


}
void print_uart(unsigned char* ptr)//
{
        #define WRITE_BUSY 0x0100
        unsigned uport;
        while (*ptr) {
                do {
                  uport=*(volatile unsigned*)             uart_port;
                } while (uport & WRITE_BUSY);
                *(volatile unsigned char*)uart_wport=*(ptr++);
        }
}
char *strrev(char *s) {
        char *ret = s;
        char *t = s;
        char c;

        while( *t != '\0' )t++;
        t--;

        while(t > s) {
                c = *s;
                *s = *t;
                *t = c;
                s++;
                t--;
        }
```

```
		return ret;
}


void utoa(unsigned val, char *s) {
		char *t;
		int mod;


		t = s;

		while(val) {
				mod = val % 10;
				*t++ = (char)mod + '0';
				val /= 10;

		}

		if(s == t)
				*t++ = '0';

		*t = '\0';


		strrev(s);
}
void print_num(unsigned u)
{
		char buffer[12];
		utoa(u,buffer);
		print_uart(buffer);
		buffer[0]=0x0a;
		buffer[1]=0x00;
		print_uart(buffer);
}
double target_sds[]={//0dB -10dB STEP 0.5dB
		1.000,
0.944,
0.891,
0.841,
0.794,
0.750,
0.708,
0.668,
0.631,
0.596,
0.562,
0.531,
0.501,
0.473,
0.447,
0.422,
0.398,
0.376,
0.355,
0.335,
0.316};

void main()
{
		double sum,avg,a,target_sd;
		double noise;
		int i,j,index,I;
		unsigned data_in,data_out;


		int n=No;
#ifdef RTL_SIM
		print_uart("V1.3\n");

#endif
		total_bits=0;
		sum=0;

		#ifdef STATISTICS
				for (i=0;i< (STATISTICS_LEVELS*2);i++) counter_array[i]=0;
		#endif

		for(j=0;j< 21 ;j++) {
				target_sd=target_sds[j];
				bit_counter=0;
				bit_error_counter=0;
				raw_bit_error_counter=0;
				do{

						data_in=make_input_data(target_sd);
						#ifndef DOS
								data_write_and_compare(data_in);
						#endif
						bit_counter +=4;
```

```
                                        total_bits  +=4;

                                        if (bit_error_counter> ERROR_THRESHOLD){
                                                print_uart("Pass=");
                                                print_num(j);
                                                print_num(bit_counter);
                                                print_num(raw_bit_error_counter);
                                                print_num(bit_error_counter);
                                                print_uart("EndPass\n");
                                                break;
                                        }
                                }while(1);
                        }
  #ifdef DOS
                         #ifdef SD_DISPLAY
                                avg=sum/No;
                        sum=0;
                        for (i=0;i<No;i=i+1){
                                a=array[i]-avg;
                                sum=sum+a*a;
                        }

                                sd=sqrt(sum/No);
                        printf("Done %d samples. avg=%f target_sd=%f sd=%f",n,avg,target_sd,sd);
                         #endif
                           printf("Done %d samples.  target_sd=%f\n",n,target_sd);
                                for (i=0;i< (STATISTICS_LEVELS*2);i++){
                                        printf("%d,%d\n",i-STATISTICS_LEVELS,counter_array[i]);
                                }
  #else
    print_uart("Done All.\n");

  #endif
}
```

プログラム5　実機ハードウェア記述（TOP 階層のみ）

```
//Jun.30.2004 blez bgtz bug fix
//Jul.7.2004 int bug fix
//Jul.11.2004 bgezQ,bltzQ
//Apr.2.2005 Change Port Address, change uart interface port
//Apr.3.2005 bgtz bug fix
//Apr.13.2005 Add stratix2 workaround
//Aug.1.2005 Separate Hardware/Ram Blocks

`include "define.h"
`ifdef RTL_SIMULATION
module yacc(clock,Async_Reset,MemoryWData,MWriteFF,data_port_address,
                RXD,TXD);
        input clock;
        input Async_Reset;
        output [31:0] MemoryWData;
        output [15:0] data_port_address;



        output MWriteFF;
        input RXD;
        output TXD;
`else

module yacc(clock,Async_Reset,          RXD,TXD);
        input clock;
        input Async_Reset;
        input RXD;
        output TXD;

`endif

        wire [31:0] MOUT,IRD1;
        wire [31:0] Hard_or_Memory_output;//Aug.1.2005
        reg [31:0] Hard_OUT;//Aug.1.2005
        wire hardware_output_true;//Aug.1.2005
        reg hardware_output_trueD;//Aug.1.2005

        wire RegWriteD2;
        wire [1:0] A_Right_SELD1;
        wire [1:0] RF_inputD2;
        wire M_signD1,M_signD2;
        wire [1:0] M_access_modeD1,M_access_modeD2;
        wire [3:0] ALU_FuncD2;
        wire [1:0] Shift_FuncD2;
        wire [25:0] IMMD2,IMMD1;
        wire [4:0] source_addrD2,target_addrD2;
        wire [4:0] source_addrD1,target_addrD1,Shift_amountD2;
        wire [4:0] RF_input_addr;
        wire [2:0] PC_commandD1;
        wire [7:0] uread_port;
        wire takenD2;//


        wire [25:0] PC;
        wire [25:0] DAddress;//
        reg [25:0] int_address;//interim
        reg [25:0] PCD1,PCD2;
        reg [25:0] IRD2;
        reg [25:0] DAddrD;
        wire [25:0] PCCDD;

        wire [31:0] memory_indata;//
        wire memory_sign;
        wire [1:0] memory_access_mode;

        wire [31:0] ea_reg_out;
        wire [31:0] regfile_indata,regfile_indata_temp;//
        wire reg_compare;
        wire beqQ,bneQ,blezQ,bgtzQ;
        wire [25:0] IMM;
        wire clear_int;
        wire jumpQ,branchQQ;
        wire [31:0] memory_wdata;
        wire [31:0] alu_source,alu_target;
```

```
		wire [1:0] RRegSelD1;
		wire A_Left_SELD1;
		wire [1:0] mul_alu_selD2;
		wire [3:0] mul_div_funcD2;
//registers
		reg sync_reset;
		wire [31:0] forward_source_reg,forward_target_reg;
//		reg [31:0] MOUT_ff;
		reg takenD3;
		reg int_req;


		reg beqQQ,bneQQ,blezQQ,bgtzQQ;
		reg bgezQQ,bltzQQ;
		reg MWriteFF;
		wire MWriteD2,MWriteD1;
		reg [31:0] MemoryWData;
		wire NOP_Signal;

		wire [7:0] control_state;
		wire [15:0] data_port_address=DAddrD;
		wire [3:0] mult_func;
		wire pause_out;
		wire Shift_Amount_selD2;
		wire source_zero;//Jun.30.2004
		wire int_req_uport;
		wire uart_write_req;
		wire uart_write_done,uart_write_busy;
		wire int_stateD1;
		wire bgezQ,bltzQ;

decoder d1(.clock(clock),.sync_reset(sync_reset),.MWriteD1(MWriteD1),
		.RegWriteD2(RegWriteD2),.A_Right_SELD1(A_Right_SELD1),.RF_inputD2(RF_inputD2),
		.RF_input_addr(RF_input_addr),.M_signD1( M_signD1),.M_signD2(M_signD2),
		.M_access_modeD1(M_access_modeD1),.M_access_modeD2(M_access_modeD2),
		.ALU_FuncD2(ALU_FuncD2),.Shift_FuncD2(Shift_FuncD2),
		.source_addrD1(source_addrD1),.target_addrD1(target_addrD1),.IMMD2(IMMD2),
		.source_addrD2(source_addrD2),.target_addrD2(target_addrD2),

.Shift_amountD2(Shift_amountD2),.PC_commandD1(PC_commandD1),.IMMD1(IMMD1),.IRD1(IRD1),.takenD3(takenD3),.takenD2(takenD2),.beqQ(beqQ),.bneQ(bneQ)
,.blezQ(blezQ),.bgtzQ(bgtzQ),
		.DAddress(DAddress),.PC(PC),.memory_indata(memory_indata),.MOUT(MOUT),.IMM(IMM),
		.branchQQ(branchQQ),.jumpQ(jumpQ),.int_req(int_req),.clear_int(clear_int),
		.int_address(int_address),.A_Left_SELD1(A_Left_SELD1),.RRegSelD1(RRegSelD1),
		.MWriteD2(MWriteD2),.NOP_Signal(NOP_Signal),.mul_alu_selD2(mul_alu_selD2),
		.mul_div_funcD2(mul_div_funcD2),.pause_out(pause_out),.control_state(control_state),
		.Shift_Amount_selD2(Shift_Amount_selD2),
		.int_stateD1(int_stateD1),.bgezQ(bgezQ),.bltzQ(bltzQ),.hard_access(hardware_output_true));//Aug.1.2005



pc_module pc1(.clock(clock),.sync_reset(sync_reset),.pc_commandD1(PC_commandD1),.PCC(PC),
		.imm(IMM),.ea_reg_source(alu_source),.takenD2(takenD2),.takenD3(takenD3),
		.branchQQ(branchQQ),.jumpQ(jumpQ),.NOP_Signal(NOP_Signal),
		.control_state(control_state),.IMMD1(IMMD1),.PCCDD(PCCDD));



Pipelined_RegFile pipe(.clock(clock),.sync_reset(sync_reset),
		.dest_addrD2(RF_input_addr),.source_addr(IMM[25:21]),.target_addr(IMM[20:16]),
		.wren(RegWriteD2),.memory_wdata(memory_wdata),
		.A_Right_SELD1(A_Right_SELD1),.A_Left_SELD1(A_Left_SELD1),.PCD1(PCD1),
		.IMMD1(IMMD1[15:0]),.ALU_FuncD2(ALU_FuncD2),.Shift_FuncD2(Shift_FuncD2),
		.Shift_amountD2(Shift_amountD2),.RRegSelD1(RRegSelD1),.MOUT(Hard_or_Memory_output),//Aug.1.2005
		.RF_inputD2(RF_inputD2),.alu_source(alu_source),.alu_target(alu_target),
		.MWriteD2(MWriteD2),.MWriteD1(MWriteD1),.mul_alu_selD2(mul_alu_selD2),
		.mul_div_funcD2(mul_div_funcD2),.pause_out(pause_out),
		.Shift_Amount_selD2(Shift_Amount_selD2),.int_stateD1(int_stateD1),.PCCDD(PCCDD));

//sync_reset
`ifdef Stratix2
		always @(posedge clock ) begin//Workaround for stratix2
				sync_reset <=!Async_Reset;
		end
`else
		always @(posedge clock , negedge Async_Reset) begin
				if (!Async_Reset) sync_reset <=1'b1;
				else  sync_reset <=!Async_Reset;
		end
`endif

//PCD1,PCD2
		always @(posedge clock) begin
				PCD1 <=PC+4;
		end

		always @(posedge clock) begin
				PCD2 <=PCD1;
		end
```

```
//
		always @(posedge clock) begin
		IRD2 <=IRD1;
		end

		always @(posedge clock) begin
			if (sync_reset) MWriteFF<=1'b0;
			else		MWriteFF <=MWriteD2;
		end

		assign memory_access_mode=M_access_modeD1;
		assign memory_sign=M_signD1;

//Apr.14.2005		assign DAddress=alu_source[25:0]+{ {6{IRD2[15]}},IRD2[15:0]};
		assign DAddress=alu_source[25:0]+{ {10{IRD2[15]}},IRD2[15:0]};

//
		always @(posedge clock) begin
		DAddrD <=DAddress;
		end
//
always @(posedge clock) begin
		MemoryWData <=memory_wdata;
		end

		assign memory_indata=memory_wdata;




		assign reg_compare=( alu_source==alu_target);



		always @(posedge clock) begin
		  if (!NOP_Signal) begin//Jun.29.2004
					beqQQ<=beqQ;
			bneQQ<=bneQ;
			bgtzQQ<=bgtzQ;
			blezQQ<=blezQ;
					bgezQQ<=bgezQ;//Jul.11.2004
			bltzQQ<=bltzQ;//Jul.11.2004
				end
		end

		always @( beqQQ ,bneQQ,bgtzQQ,blezQQ,bgezQQ,bltzQQ,reg_compare,alu_source) begin//Jul.11.2004
			takenD3=  ( beqQQ   && reg_compare) ||
				( bneQQ   && !reg_compare) ||
				( bgtzQQ  && !alu_source[31] && !reg_compare) || //Apr.3.2005 bug fix $s >0 Jun.30.2004
				( blezQQ  && (alu_source[31]  || reg_compare )) ||
				( bgezQQ  && (!alu_source[31] || reg_compare )) || //Jul.11.2004
				( bltzQQ  && (alu_source[31]  )); //Jul.11.2004//$s <0=Jun.30.2004
		end

//Hardware Blocks Aug.1.2005




reg [31:0] my_reg_file;//Your Hardware Register File difinition
 wire [23:0] indata=my_reg_file[23:0];
 wire [3:0] outdata;
hardware # (.BW(3)) hw
		( .indata(indata), .outdata(outdata), .clock(clock), .Reset(!Async_Reset)	);

		reg hardware_output_trueDD;

		uart_read  uread( .sync_reset(sync_reset),.clk(clock), .rxd(RXD),
		.buffer_reg(uread_port),.int_req(int_req_uport));

		uart_write uwite( .sync_reset(sync_reset), .clk(clock), .txd(TXD),.data_in(MemoryWData[7:0]) ,
		.write_request(uart_write_req),.write_done(uart_write_done),.write_busy(uart_write_busy));

		assign hardware_output_true=`HARD_BLOCK_START_ADDRESS<=DAddress
									&& `HARD_BLOCK_END_ADDRESS>=DAddress;
		wire my_reg_file_access=`REG_PORT_START_ADDRESS<=DAddrD
									&& `REG_PORT_END_ADDRESS>=DAddrD;

		wire uread_access=`UART_PORT_ADDRESS==DAddrD;

		always @(posedge clock) begin
			if (sync_reset) hardware_output_trueD<=0;
			else hardware_output_trueD<=hardware_output_true;
		end

		always @(posedge clock) begin
			if (sync_reset) hardware_output_trueDD<=0;
			else hardware_output_trueDD<=hardware_output_trueD;
		end
```

```
	always @(posedge clock) begin
		if (sync_reset) Hard_OUT<=0;
		else if(uread_access)
					Hard_OUT <={23'h00_0000,uart_write_busy,uread_port};
		else if (my_reg_file_access) begin
//					Hard_OUT <=my_reg_file[DAddrD[12:2]];
						Hard_OUT <={24'h00_0000,outdata};
		end
	end
	assign Hard_or_Memory_output=hardware_output_trueDD ? Hard_OUT: MOUT;//Hardware/RAM multiplexer

`ifdef RAM16K
	assign uart_write_req= DAddrD==`UART_PORT_ADDRESS && MWriteFF  ;//`UART_WRITE_PORT_ADDRESS ;
		always @ (posedge clock) begin
			if (sync_reset) int_address<=0;
			else if (DAddrD==`INTERUPPT_ADDRESS & MWriteFF) int_address<=MemoryWData;
			else if (my_reg_file_access & MWriteFF) my_reg_file<=MemoryWData;
	end
`endif



//state machine
//latch with one shot pulse
//clear by clear_int
	always @(posedge clock) begin
		if (sync_reset) int_req <=1'b0;
		else if (clear_int) int_req <=1'b0;// assume one shot(1clk) pulse
		else if ( int_req_uport) int_req<=1'b1;//
	end

endmodule
```

プログラム6 - VPI による高速 RTL シミュレーション

```
//VPI ソース


static int sys_my_random_for_dwm_size_tf()//Aug.42003
{
        return 32;
}
 /* position of right-most step */
#define PARAM_R 2.96454468156

/* level values */
static const double ytab[32] = {
  1,
  0.907475617638,
  0.8387295158,
  0.780775785072,
  0.729446868113,
  0.682776316768,
  0.639647716015,
  0.599353201849,
  0.561409662765,
  0.525469625896,
  0.491273228111,
  0.458620229136,
  0.427352696597,
  0.397343775955,
  0.368490127285,
  0.340706676394,
  0.313922886649,
  0.288080067698,
  0.263129417761,
  0.23903060658,
  0.215750777731,
  0.193263899438,
  0.171550433862,
  0.150597335281,
  0.130398439093,
  0.110955385052,
  0.0922793705114,
  0.0743943543072,
  0.0573431209268,
  0.0411998625242,
  0.0261009770388,
  0.0123479825627,
};

/* quick acceptance check */
static const unsigned long ktab[32] = {
  0,
  12465559,
  14143380,
  14859376,
  15254596,
  15503949,
  15674743,
  15798352,
  15891342,
  15963265,
  16020003,
  16065358,
  16101882,
  16131333,
  16154939,
  16173564,
  16187797,
  16198017,
  16204425,
  16207053,
  16205755,
  16200181,
  16189707,
  16173323,
  16149413,
  16115345,
  16066638,
  15995077,
  15883832,
  15691829,
  15281721,
  15063248
};
```

```
/* quick value conversion */
static const double wtab[32] = {
  2.6265107239e-08,
  3.5349827263e-08,
  4.19328107856e-08,
  4.73449074891e-08,
  5.20705831605e-08,
  5.63468970139e-08,
  6.03100167664e-08,
  6.40468149156e-08,
  6.76171489147e-08,
  7.10648776583e-08,
  7.44238783956e-08,
  7.77216053723e-08,
  8.09813473119e-08,
  8.42237615336e-08,
  8.74679979284e-08,
  9.0732597892e-08,
  9.4036290297e-08,
  9.73987784303e-08,
  1.00841606117e-07,
  1.04389204949e-07,
  1.08070261837e-07,
  1.11919620405e-07,
  1.15981071511e-07,
  1.20311664676e-07,
  1.24988739931e-07,
  1.30122131768e-07,
  1.35877026736e-07,
  1.42521234138e-07,
  1.50537318078e-07,
  1.60949814958e-07,
  1.76700632665e-07,
  1.96806467179e-07
};



unsigned long s1=0xffffffff,s2=0xffffffff,s3=0xffffffff;
unsigned long taus88_int ()
{/*Generates numbers between 0 and 1.*/
                unsigned b;

                b =(((s1 <<13)^s1)>>19);
                s1 =(((s1 &4294967294)<<12)^b);
                b =(((s2 <<2)^s2)>>25);
                s2 =(((s2 &4294967288)<<4)^b);
                b =(((s3 <<3)^s3)>>11);
                s3 =(((s3 &4294967280)<<17)^b);
                return ((s1 ^s2 ^s3));
}

double taus88 ()
{
                return (taus88_int()*2.3283064365e-10);
}



double
gsl_ran_gaussian_ziggurat ()
{
  unsigned long  U, sign, i, j;
  double  x, y;

  while (1) {
    U = taus88_int();
 //   i = U & 0x0000007F;                        /* 7 bit to choose the step */
 //   sign = U & 0x00000080;             /* 1 bit for the sign */
    i = U & 0x0000001F;                         /* 7 bit to choose the step */
    sign = U & 0x00000020;              /* 1 bit for the sign */

   // j = U>>8;                                  /* 24 bit for the x-value */
    j = U>>6;                                   /* 24 bit for the x-value */

    x = j*wtab[i];
    if (j < ktab[i])  break;

  //if (i<127) {
     if (i<32) {

       double  y0, y1;
       y0 = ytab[i];
       y1 = ytab[i+1];
       y = y1+(y0-y1)*taus88();
    } else {
       x = PARAM_R - log(1.0-taus88())/PARAM_R;
       y = exp(-PARAM_R*(x-0.5*PARAM_R))*taus88();
```

```
    }
    if (y < exp(-0.5*x*x))  break;
  }
  return  sign ? x : -x;
}



unsigned quantize(double rvalue)
{
        double R;
        int qdata;
        R=rvalue*1;

        if (R>=1.5) qdata=3;
        else if (R>=1.0) qdata=2;
        else if (R>=0.5) qdata=1;
        else if (R>=0) qdata=0;
        else if (R>=-0.5) qdata=-1;
        else if (R>=-1.0) qdata=-2;
        else if (R>=-1.5) qdata=-3;
        else  qdata=-4;


        qdata &=0x7;//3bit 切り出し
        return qdata;

}


unsigned get_random_data()
{
        unsigned y;
        unsigned r0,r1,c0,c1;
        unsigned y0,y1,y2,y3;
        unsigned outdata;

                y=taus88_int();//ランダムデータを得る。下位 4 ビットしか使っていない
        //ビット切り出し
                y0=y & 0x01;
                y1=(y >> 1) & 0x01;
                y2=(y >> 2) & 0x01;
                y3=(y >> 3) & 0x01;

        //パリティ計算
                r0=y0 ^ y1;
                r1=y2 ^ y3;
                c0=y0 ^ y2;
                c1=y1 ^ y3;

                outdata=y0;
                outdata |=y1 <<1;
                outdata |=y2 <<2;
                outdata |=y3 <<3;
                outdata |=r0 <<4;
                outdata |=r1 <<5;
                outdata |=c0 <<6;
                outdata |=c1 <<7;
                outdata <<=24;// 上位 8 ビットが訂正前ノイズ加算前エンコードデータ
                return outdata;
}

unsigned make_input_data(double target_sd)
{

        double rvalue;
        unsigned temp;
        unsigned i,j;
        double noise;
        unsigned outdata;

    unsigned word;
  //  unsigned temp;

        outdata=get_random_data();//エンコーダ上位 8 ビットにエンコードデータ


                //3 ビットアナログ受信値の作成
                for (i=0;i<8 ;i++) {
                        noise=gsl_ran_gaussian_ziggurat();//1.0 ガウスノイズ生成
                        #ifdef STATISTICS
                                statistics(noise);//ガウスノイズの度数分布処理
                        #endif
                        noise *=target_sd;//ノイズシグマ乗算
                        if ( (outdata >> (24+i)) & 0x01) {
                                rvalue=-1.0;//1-> -1 に
                        }else rvalue=+1.0;//0-> +1 にエンコード
                        rvalue +=noise;
                        temp=quantize(rvalue);//3 ビットに量子化
```

```
                                        temp <<=3*i;
                                        outdata |=temp;
                                }
        return outdata;
}

 static int sys_my_random_for_dwm(char*  name)
{

 vpiHandle systfref,argsiter,argh;
  t_vpi_value value;
        double r;

        systfref = vpi_handle(vpiSysTfCall, NULL); /* get system function that invoked C routine */
        argsiter = vpi_iterate(vpiArgument, systfref);/* get iterator (list) of passed arguments */
        argh = vpi_scan(argsiter);/* get the one argument - add loop for more args */
        if(!argh){
                                        vpi_printf("$VPI missing parameter.\n");
                                        return 0;
        }

        value.format = vpiRealVal;
    vpi_get_value(argh, &value);
    r = value.value.real;

        value.value.integer =make_input_data(r)   ;
        value.format = vpiIntVal;/* return the result */
        vpi_put_value(systfref, &value, NULL, vpiNoDelay);

        vpi_free_object(argsiter);

        return(0);

 }


extern "C" void sys_math_vpi_register()//VPI   DLL ロード時 CALL され、$ Function が登録される。
{
      s_vpi_systf_data tf_data;



      //Dec.30.2005 FOR DWM
      tf_data.type      = vpiSysFunc;//Define as function
      tf_data.subtype         =vpiIntFunc;//return by integer
      tf_data.tfname    = "$my_random_for_dwm";
      tf_data.user_data = "$my_random_for_dwm";
      tf_data.calltf    = sys_my_random_for_dwm;
      tf_data.compiletf = 0;
      tf_data.sizetf    = sys_my_random_for_dwm_size_tf;//func なので返す
      vpi_register_systf(&tf_data);

}

//テストベンチ

//Nov.2.2005
//Nov.3.2005
`timescale 1ns/1ps
//`define SPARTAN
//`define SPARTAN_DELAY
`define ST2_DELAY
`ifdef SPARTAN_DELAY
        `define CYCLE (5.368/2)
`elsif ST2_DELAY
        `define CYCLE (4.7/2) //4.5 FAIL

`else
        `define CYCLE (5)
`endif


module euclidean_error_rate;

        parameter integer ERROR_LIMIT=1000;
        parameter integer BW=3;
        parameter integer FIFO_DEPTH=9;

        reg clock=0;
        reg [31:0]  VPI_Data;
        wire [23:0] indata=VPI_Data[23:0];
        wire [3:0] correct_data=VPI_Data[27:24];
        wire [3:0] outdata;
        reg Reset=1;

        integer counter=0;
        integer bit_counter=0;
        integer bit_error_counter=0;
        real DB;
        real sigma;
        initial begin
```

```
		#100;
		Reset=0;
		DB=0.0;
		set_sigma;


	end
	generate
		genvar gen;
		for (gen=0;gen<FIFO_DEPTH;gen=gen+1) begin :loop
			reg [4:0] D;
			if  (gen==0) begin

					always  @(posedge clock) begin
						D<={Reset,correct_data};
					end
			end else if(gen==FIFO_DEPTH-1) begin
					always  @(posedge clock) begin
						D<=loop[gen-1].D;
						@(negedge clock);
						if (D[4]==0 ) begin
							bit_counter=bit_counter+4;
							count_bit_error(D[3:0],outdata);
						end
					end

			end else begin
					always  @(posedge clock) begin
						D<=loop[gen-1].D;
					end

			end
		end

	endgenerate
	 task count_bit_error( input [3:0] in1,in2) ;
		integer i;
		real rate;
		begin
			if ( in1 !==in2) begin
				for (i=0;i<4;i=i+1)  begin
					if (in1[i] !==in2[i]) bit_error_counter=bit_error_counter+1;
				end
			end
			if (bit_error_counter >ERROR_LIMIT) begin
				rate=bit_error_counter;
				rate=rate/bit_counter;
				$display("DB=%f bits=%d error=%d error_rate=%e sigma=%f",DB,bit_counter,bit_error_counter,rate,sigma);
				DB=DB+0.5;
				bit_counter=0;
				bit_error_counter=0;
				set_sigma;
			end
		end
	endtask

	task set_sigma;
		real sigma2;
		begin
			sigma2=$pow(10.0,-DB/10.0);//
			sigma=$sqrt(sigma2);//$sqrt
		end
	endtask


	always @(negedge clock) begin
		#1;
		if (Reset) ;
		else begin

			VPI_Data=$my_random_for_dwm(sigma);

		end

	end

	always #(`CYCLE) clock=~clock;


	hardware # ( .BW(BW)) hw (  .indata(indata),  .outdata(outdata), .clock(clock), .Reset(Reset)		);

endmodule
```