ディジタル **FM** レシーバ回路の設計

課題　　基本課題
-10Mbps FM　Demodulator 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
菅原明美(2 名）

所属　　菅原システムズ

住所　　〒981-3215　宮城県仙台市泉区北中山 3-24-13

1.概略

基本課題は、+1/-1 のデータを復調するものです。これは、FSK（Frequency Shift Keying）として、デジタルデータを復調する課題と捉えることもできます。
FSK としては、DECT/GSM 等の携帯電話や Bluetooth 等で用いられる GFSK が代表的です。そこで、コンテスト課題で、自由度が大きい部分は、Bluetooth から Spec.を引用し拘束条件としました。速度最速、かつ FPGA の特徴を生かした、スライス数最小を目指す設計としました。

表1

項	項目	仕様	引用 ***Spec.***
1	キャリア周波数（fc）	fc	
2	周波数偏移(fm)	0.01（fcに対し±1%）	コンテスト課題例示より
3	ベースバンドフィルタ	ガウスフィルタ BT=0.5	Bluetooth Spec.より
4	変調指数(fi)	0.3	Bluetooth Spec.より
5	ベースバンド転送レート	fc*fm*2/fi	1-4 項より左式で決まる
6	入出力ビット幅	入力 8 ビット、出力 12 ビット	コンテスト課題より
7	デバイス	Xc3s200-4ft256	Xilinx Starter Kit (手持ちデバイス)

この条件の下に、fcを出来る限り高く、かつ消費スライス数を最小にする設計を目標としました。

2．アルゴリズム
2．1　従来方式
　デジタル FM 復調方式として、下図 $\tan^{-1}$ の差分方式が一般的です。

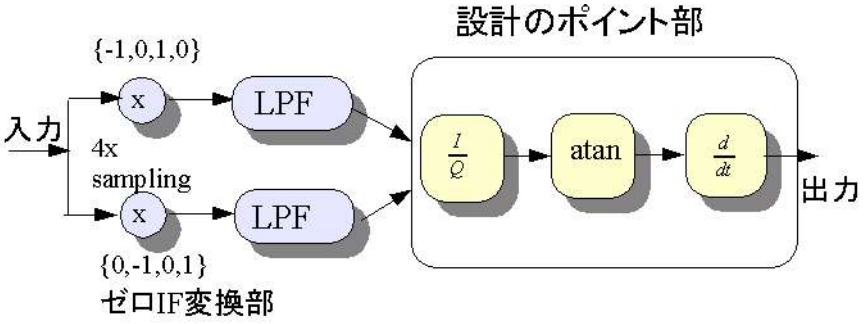
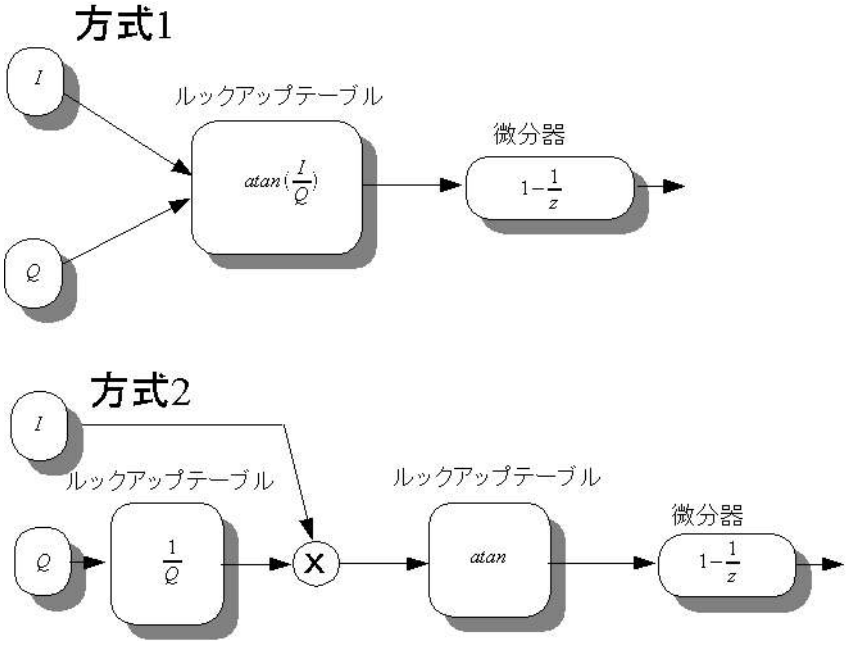


図1　FM 復調器のブロック図

この方式を採用した場合のポイントは、I/Q 除算、及び ARCTAN をいかに計算するかです。計算方法として、ハードウェア化しやすい以下の方法が考えられます。



　方式1は、arctan(I/Q) をテーブルとして持つ方法です。ビット幅が小さい場合は、テーブルも小さく済みますが、今回の例示にある 8 ビット幅の場合、単純には、64KB のテーブルが必要であり現実的ではありません。方式2は、逆数テーブル(256words)　と arctan テーブル（256words）があればよいので、今回の 8 ビット幅では、現実解になり得ます。しかし、汎用的に考えた場合、たとえば、16 ビットでは、64KB ものテーブルが必要となり、やはり大きなリソースを必要とする問題があります。（16 ビット幅で IQ が出力される IC として AD9870 が参考文献1に紹介されています。）

２．２　新方式

Xilinx の Spartan3 は、専用乗算器(18x18 ビット)という新しい FPGA アーキテクチャを採用しています。FPGA の貴重な LUT スライスを消費せず高速な乗算が利用可能です。これを利用した、新しい方式を考案しました。

２．２．１　周波数検出アルゴリズム

位相の変化＝周波数ですから、位相を微分すれば、周波数を得ることができます。これを式に展開すると

$$\frac{d\,atan(\frac{I}{Q})}{dt}=\frac{\frac{d}{dt}(\frac{I}{Q})}{1+(\frac{I}{Q})^2}=\frac{\dot{I}Q-I\dot{Q}}{I^2+Q^2}$$ --式(1)

理想的には、分母は1です。フェージングがあった場合は、値が変動しますが、フェージングなしと考えると、定数と考えることができ、分子だけを考えればよいことになります。 $\dot{I}$, $\dot{Q}$ 項は、微分値ですが、デジタル値の差分で、近似できます。kサンプル目の値は、pを遅延時間(サンプル)とすると

$$\dot{I}=I(k)(1-z^{-p})$$

$$\dot{Q}=Q(k)(1-z^{-p})$$

これより(1)式分子項は、

$$I(k)Q(k-p)-Q(k)I(k-p)$$ --式(2)

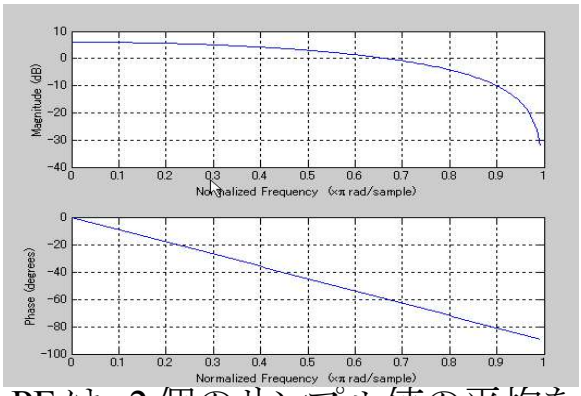
となり、乗算器2個と減算器一個で、容易に計算できます。

２．２．２　周波数変換方式

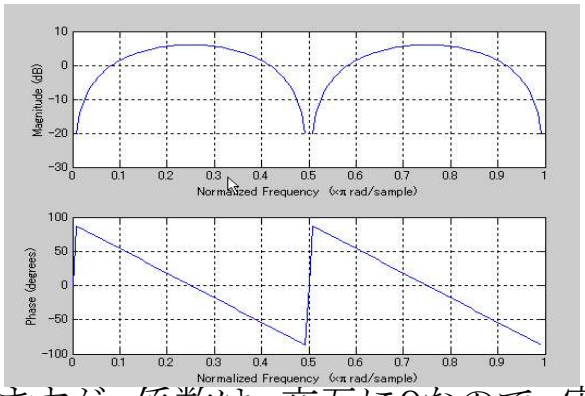
図１で、律速する部分は、明らかに、式(2)乗算部です。従って、4 倍サンプリング方式では、乗算器の最高動作周波数の 4 分の 1 以下にしか、キャリア周波数を上げられません。適用可能なキャリア周波数範囲を上げるには、乗算器の高速化が必要な事は勿論です。しかし、最近のシンセサイザは、パイプライン段数まで指定できてしまうので、シンセサイザに任せることにします。ここでは、低レートでサンプリングする方法を検討しました。

図１の周波数変換の乗算により、和(2 倍周波数成分)と差の成分に分解されます。ここで、不要な 2 倍周波数成分は、LPF(下図左)で落とします。LPF は、簡単に2次の FIR で落とすことが出来ます。4倍サンプリング周波数の2分の1(=キャリア周波数の2倍)で Null 特性となります。 同じくキャリアの2倍周波数の影響をなくすために、微分器は、 $Differentiator = 1 - Z^{-2n}$ (nは設計パラメータで整数)とします。下図右は、n=2 の例です。

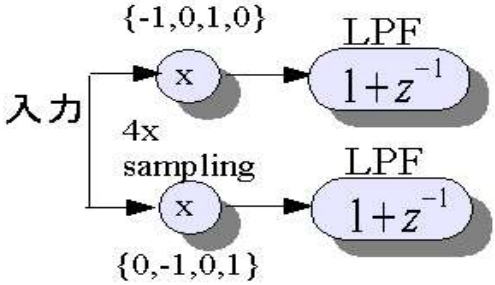
$LPF = 1 + Z^{-1}$



$Differentiator = 1 - Z^{-4}$



LPF は、2 個のサンプル値の平均をとりますが、係数は、交互に0なので、実質的にこのブロックでの計算は、必要のないことがわかります。また、IQ を組にして四相と考えることができますが、後段の微分器を n=2m(m は整数)とすることで、同じ相だけの演算に限定できます。そこで一相のみを計算する(デシメート)を行えば、サンプル数を少なくできます。



たとえば、表1の条件下で、ベースバンド転送レートを 10Mbps としたとき、キャリア周波数は、150MHz と求まります。4 倍サンプリング方式だと 600MHz での乗算が必要になります。このとき、ベースバンドの転送レート間に 60 個のサンプル値を計算することになりますが、復調後の後段に接続するベースバンドフィルタ処理で 60 個は、多すぎます。そこで、今回の設計では4相に一回だけ、演算します。結果、キャリアと同じ周波数で乗算器は動作すればよいことになり、600MHz・60 サンプリング/Symobl から 150MHz・15 サンプリング/Symbol となりました。また、IQ(8 ビット x2)は、ホストからもらう仕様としました。

3. 設計したFM復調器回路

3．1　設計仕様

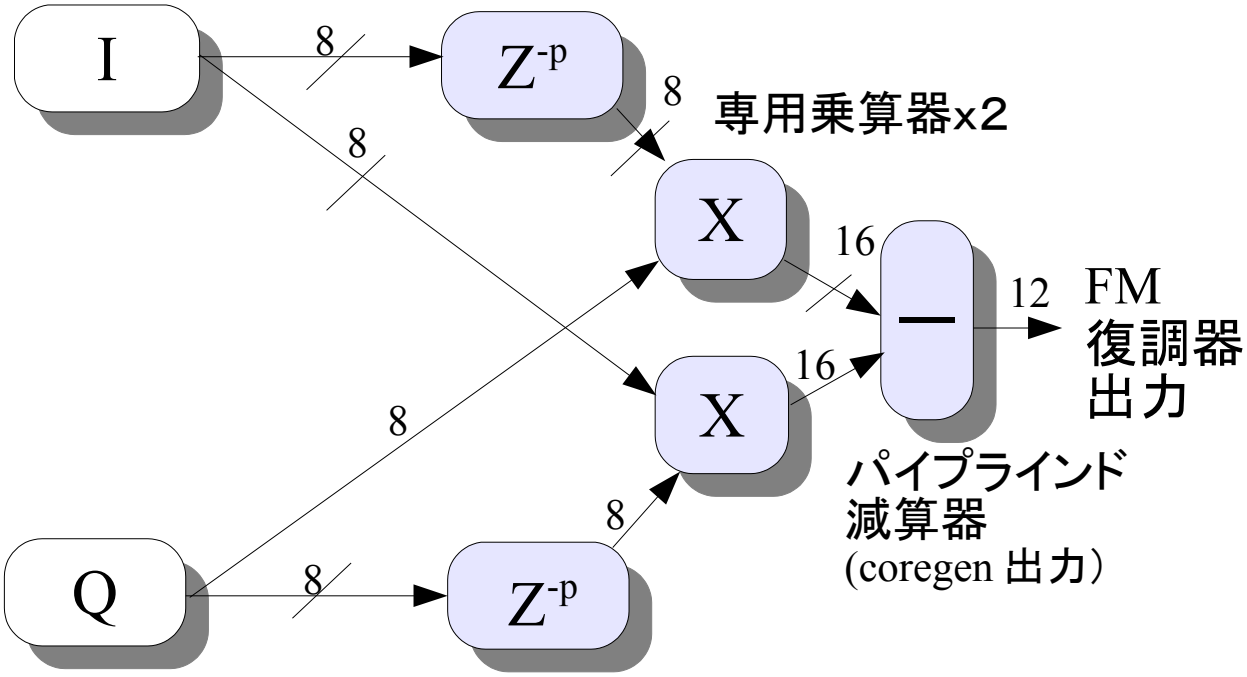
2．2の検討により、以下の設計仕様にまとめました。

表2

項	項目	仕様	引用 ***Spec.***/備考
1	キャリア周波数（fc）	150MHz	FPGA 動作周波数に同じ
2	周波数偏移(fm)	0.01（fcに対し±1%）	コンテスト課題例示より
3	ベースバンドフィルタ	ガウスフィルタ BT=0.5	Bluetooth Spec.より
4	変調指数(fi)	0.3	Bluetooth Spec.より
5	ベースバンド転送レート	fc*fm*2/fi=10Mbps	1-4 項より左式で決まる
6	入出力ビット幅	IQ 入力各 8 ビット、出力 12 ビット	コンテスト課題より
7	デバイス	Xc3s200-4ft256	Xilinx Starter Kit (手持ちデバイス)

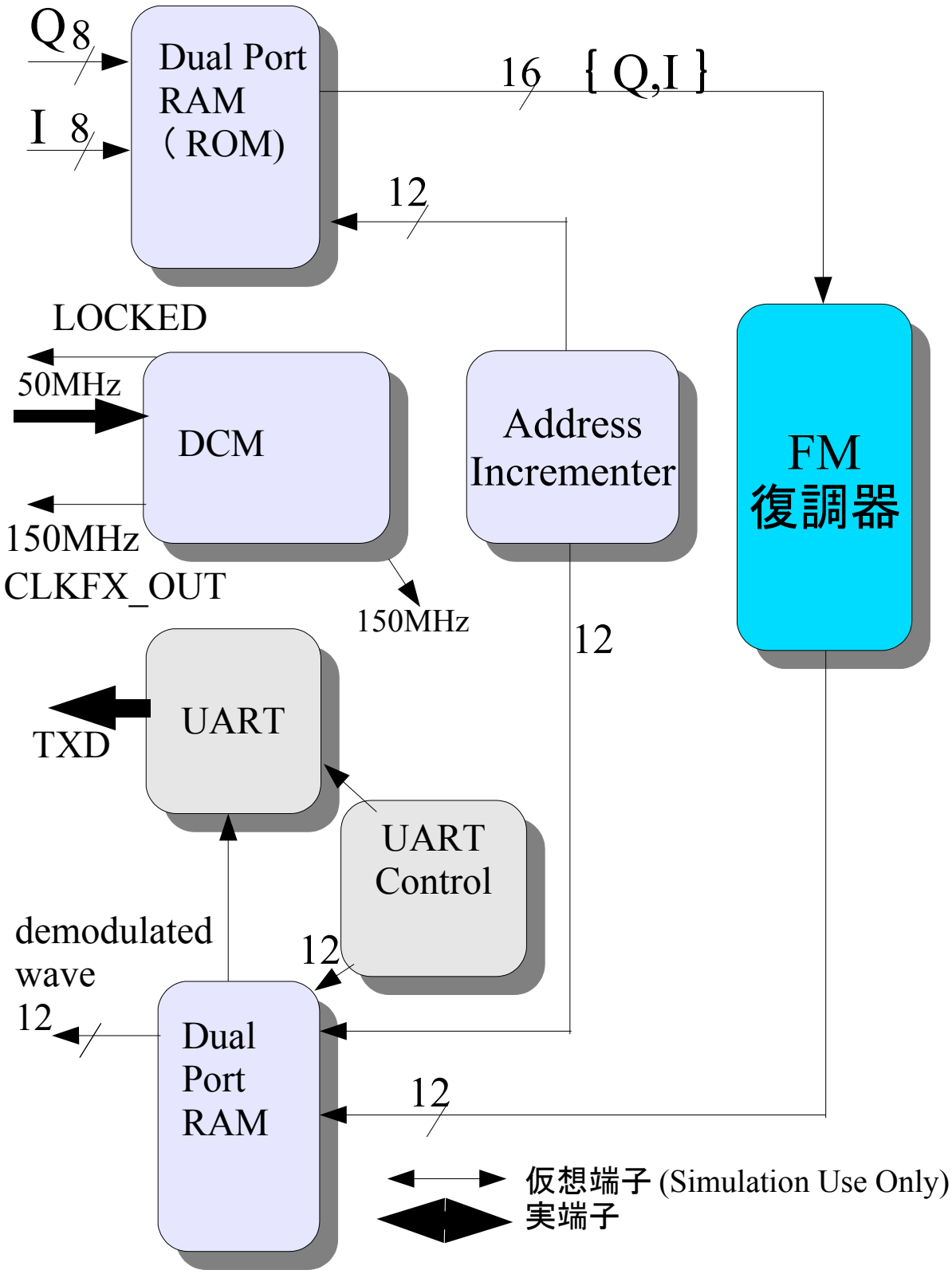
３．２　ブロック図
３．２．１　FM 復調回路

FM 復調器回路ブロック図を下に示します。



FM 復調器のブロック図
(module:demodulator_hardware_new_sampling)

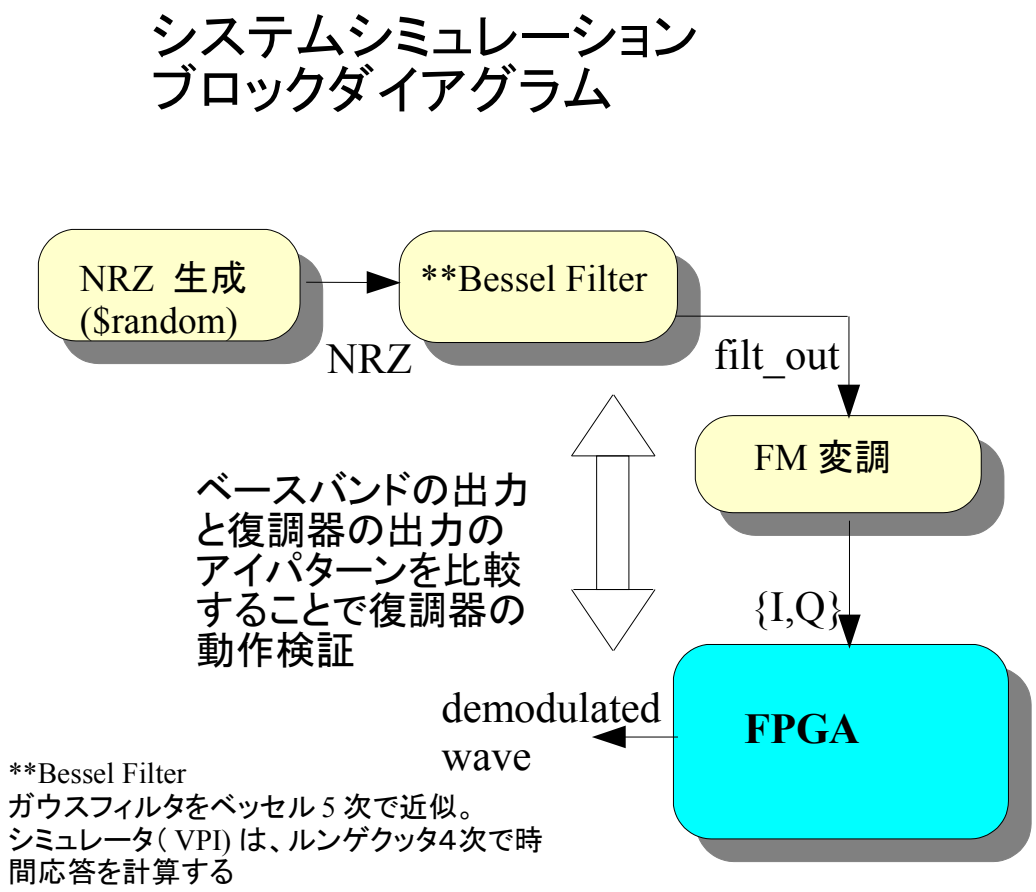
3．2．2　実機検証回路（FPGA)のブロック図



実機検証用 FPGA
ブロックダイアグラム

３．３　検証
３．３．１　検証方法

　下図のような、システムシミュレーションを行いました。また、このなかで、設計パラメータの検討も行いました。モデリング記述とシミュレーションは、すべてVerilogHDL シミュレータ上で行いました。



３．３．２　パラメータ検証

設計した、復調器で唯一の設計パラメータは、微分器の差分の遅延時間(式(2)におけるp)です。遅延時間が小さければ、微分近似が理想に近づき、微分近似誤差による波形歪は小さくなりますが、差分値が小さいため、量子化による影響が大きくなります。反対に、差分遅延時間が大きいと量子化による影響は小さくなりますが、微分近似誤差が大きくなり、波形が歪みます。このトレードオフは、アイパターンで評価しました。

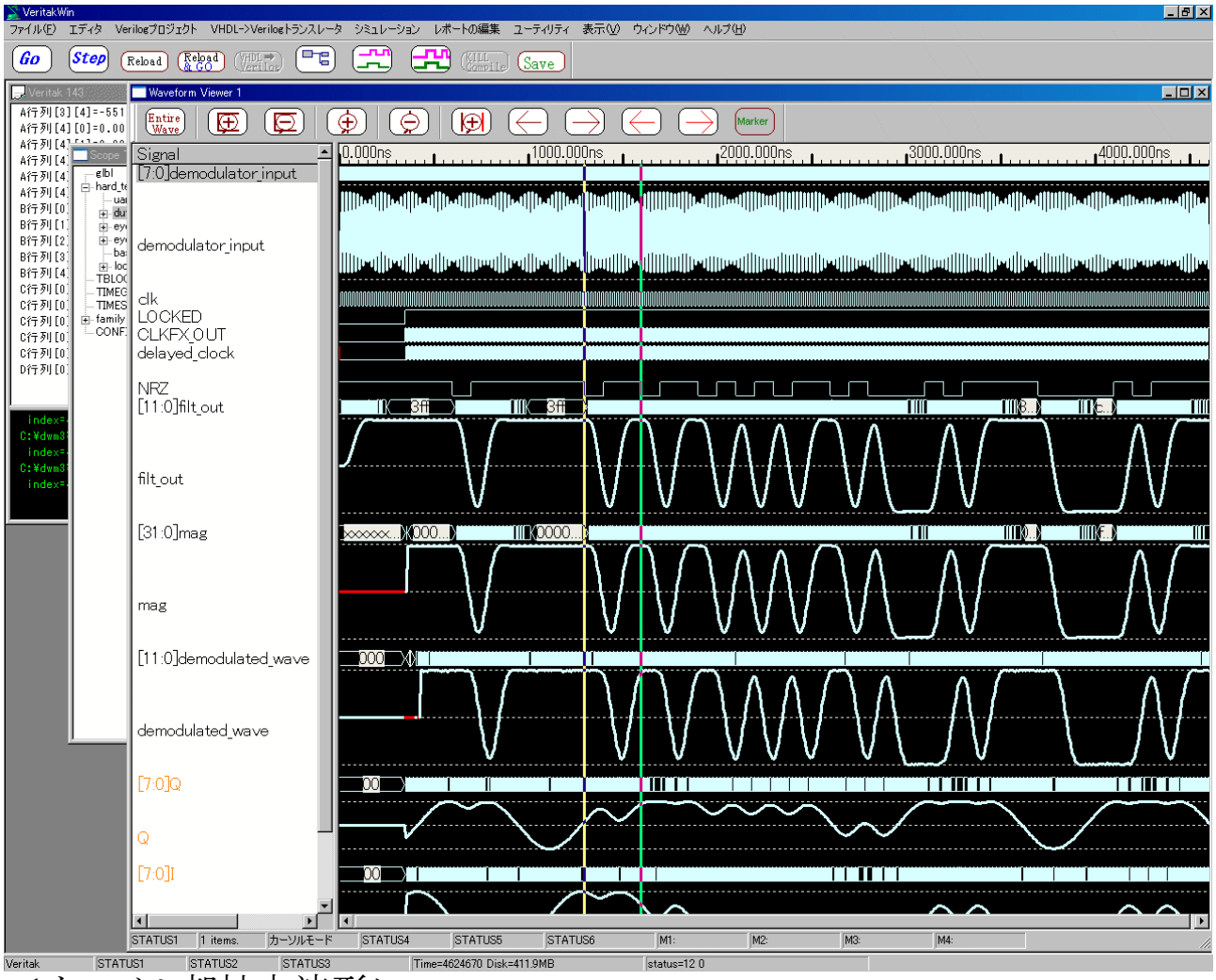
3．3．3．シミュレーション結果

シミュレーションは、量子化誤差が入らない Verolog real（浮動小数点）による結果と、実機検証用に論理合成用に記述した、8 ビット量子化での RTL シミュレーションでの結果を示します。

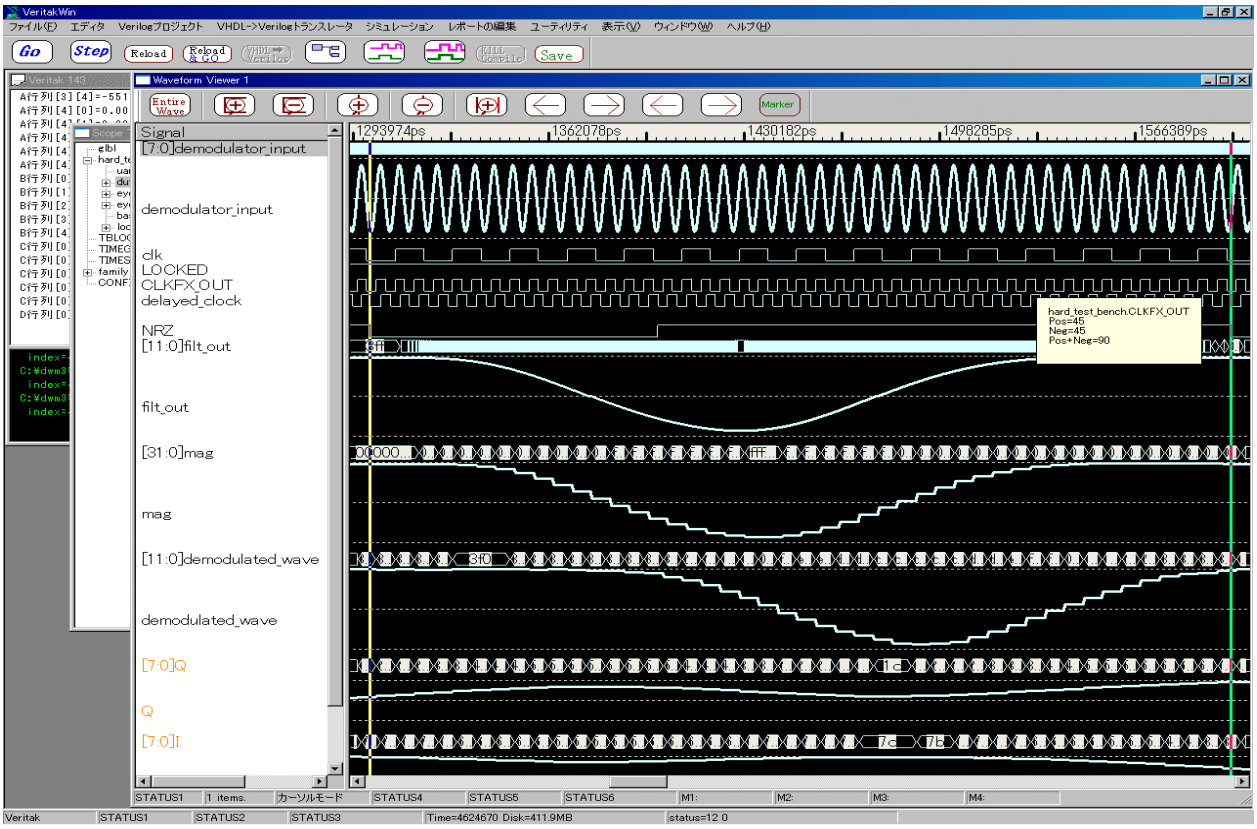
（1） 時間応答波形

＜波形の説明＞

Item	***TB/ HW***	ビット幅	内容
NRZ	TB	1	Verilog $random で生成される情報シンボル信号
demodulated_input	TB	8	変調信号(150MHz)
clk	HW	1	システムクロック(50MHz)
LOCKED	HW	1	XILINX DCM 内信号 H で DLL が LOCK したことを示す
CLK_FXOUT	HW	1	XILINX DCM 出力信号。システムクロックが3逓倍される。(150MHz)
delayed_clk	TB	1	TB 内で IQ を生成させるため、CLK_FXOUT を 90 度ずらしている。
filt_out	TB	12	NRZ を入力とするベッセルフィルタ出力信号
mag	TB	32	浮動小数点演算で計算した復調出力
demodulated_wave	HW	12	復調器出力


VeritakWin
Waveform Viewer 1
Signal
[7:0]demodulator_input
demodulator_input
clk
LOCKED
CLKFX_OUT
delayed_clock
NRZ
[11:0]filt_out
filt_out
[31:0]mag
mag
[11:0]demodulated_wave
demodulated_wave
[7:0]Q
Q
[7:0]I
0.000ns
1000.000ns
2000.000ns
3000.000ns
4000.000ns

＜カーソル部拡大波形＞


VeritakWin
Waveform Viewer 1
Signal
[7:0]demodulator_input
demodulator_input
clk
LOCKED
CLKFX_OUT
delayed_clock
NRZ
[11:0]filt_out
filt_out
[31:0]mag
mag
[11:0]demodulated_wave
demodulated_wave
[7:0]Q
Q
[7:0]I
1293974ps
1362078ps
1430182ps
1498285ps
1566389ps
hard_test_bench.CLKFX_OUT
Pos=45
Neg=45
Pos+Neg=90

(2)アイパターン評価

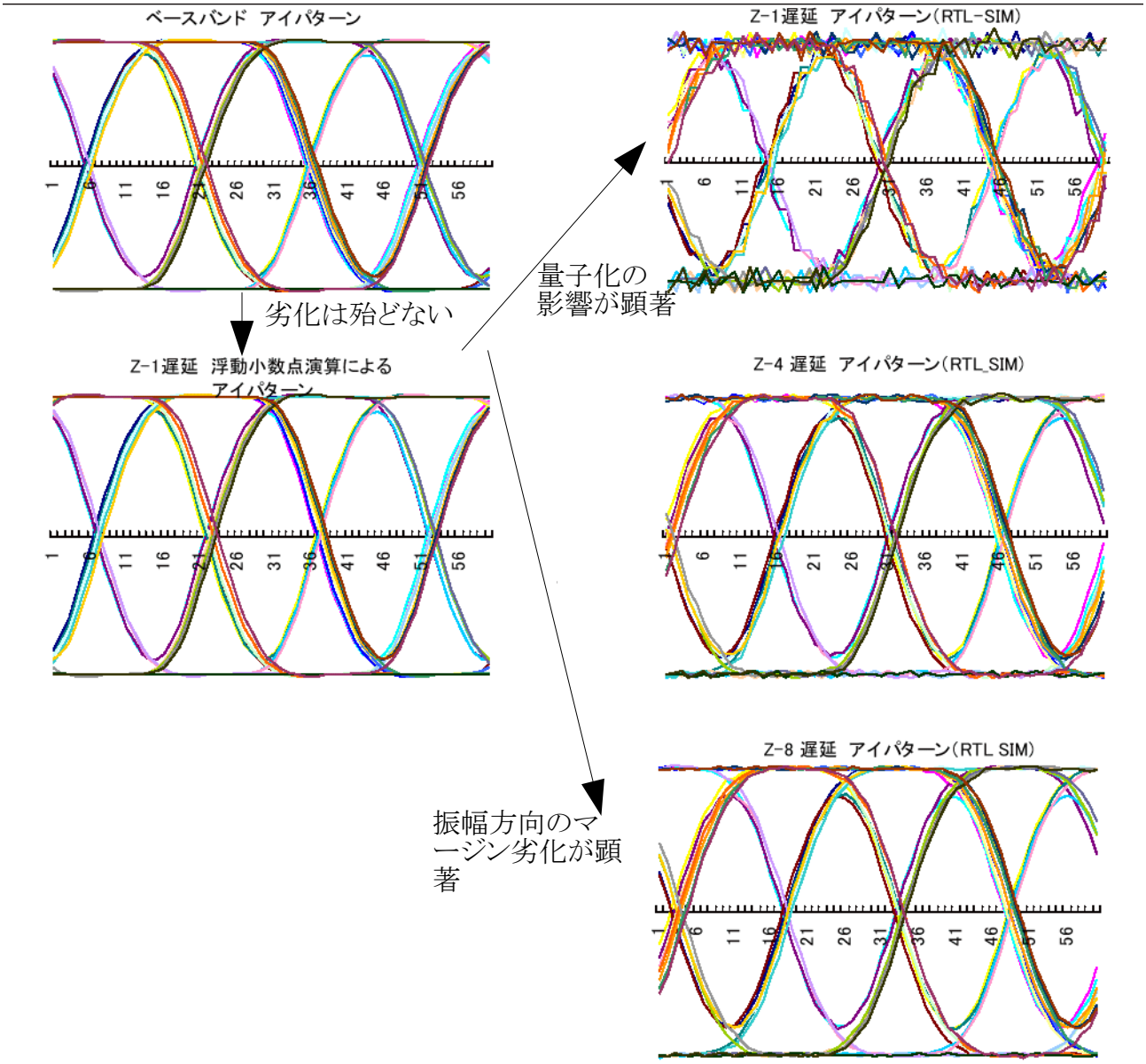


**図 Z-xは、式(2)における p

考察

・浮動小数点演算した波形(mag)は、ベースバンド波形(filt_out)とほぼ同じであり、式(2)による FM 復調の原理検証ができたと考えます。

・8 ビット量子化を行った Z-1(RTL SIM　:demodulated_wave)では、ノイズが重畳しているように見えます。後段にベースバンドフィルタを配置すれば、除去可能と考えられますが、簡単には、微分差分値遅延時間時間を大きくしていけば、見かけ上の S/N は高めることができます。

・しかしながら、大きくしすぎた遅延時間は、理想微分特性から遠のくことを意味するので、あまり大きくはできません。今回の設計値は、上記波形から p=4 としました。

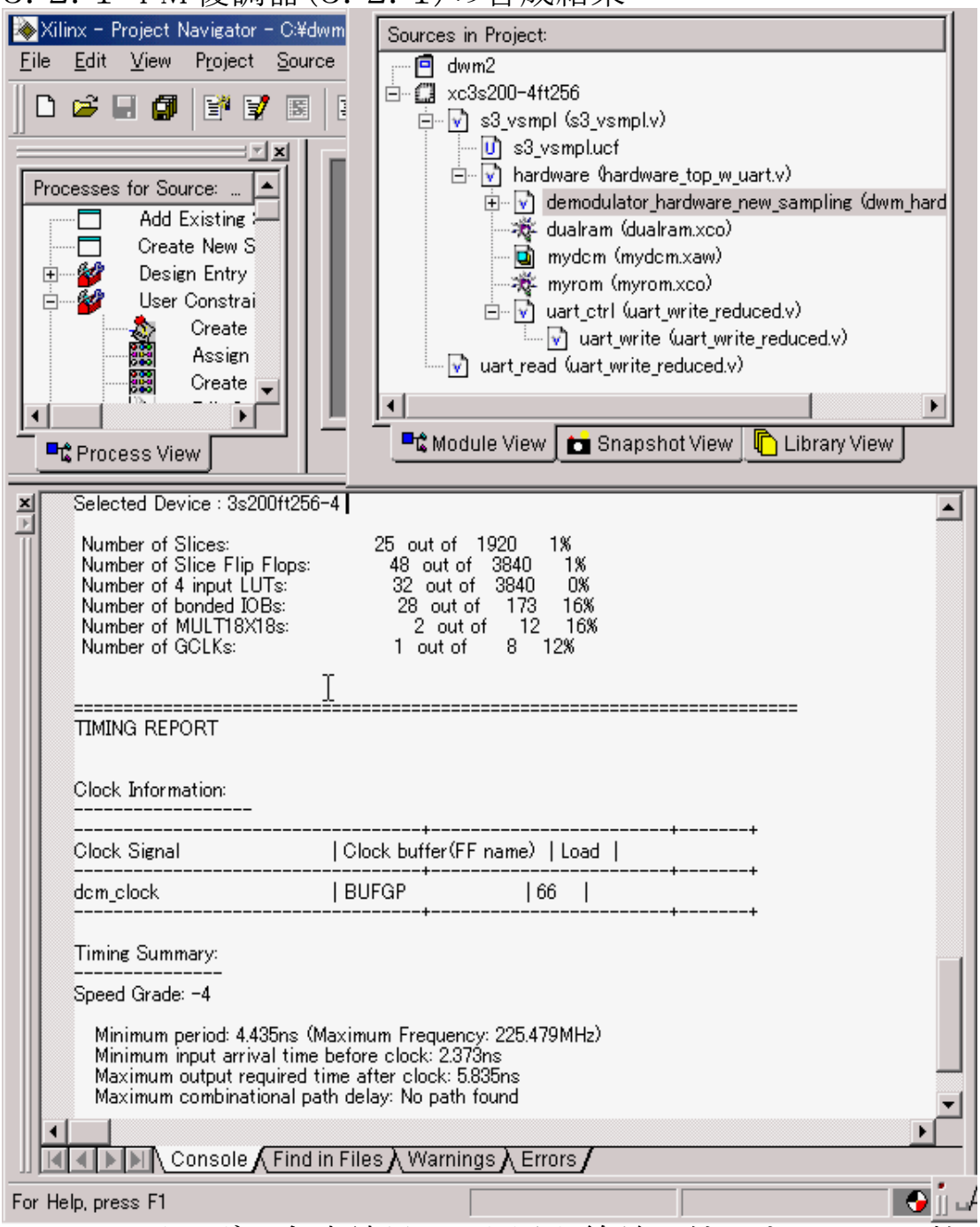
3．4　実機 FPGA 検証

今回、動作速度が、150MHz と速いため、リアルタイムでの検証は、困難です。そこで、復調器自体は、150MHz で動かしながら、結果を RAM に蓄えておいて、UART で出力する方式としました。また、150MHz で IQ をリアルタイムに与えることも困難です。そこで、これもあらかじめ計算しておいた結果(システムシミュレーション時に生成される結果)を初期値として持つ内蔵 RAM から与えるものとしました。(3．2．2　FPGA ブロック図を参照)

3．4．1　合成結果

合成環境は、Xilinx　ISE　FOUDATION 6．3 i です。

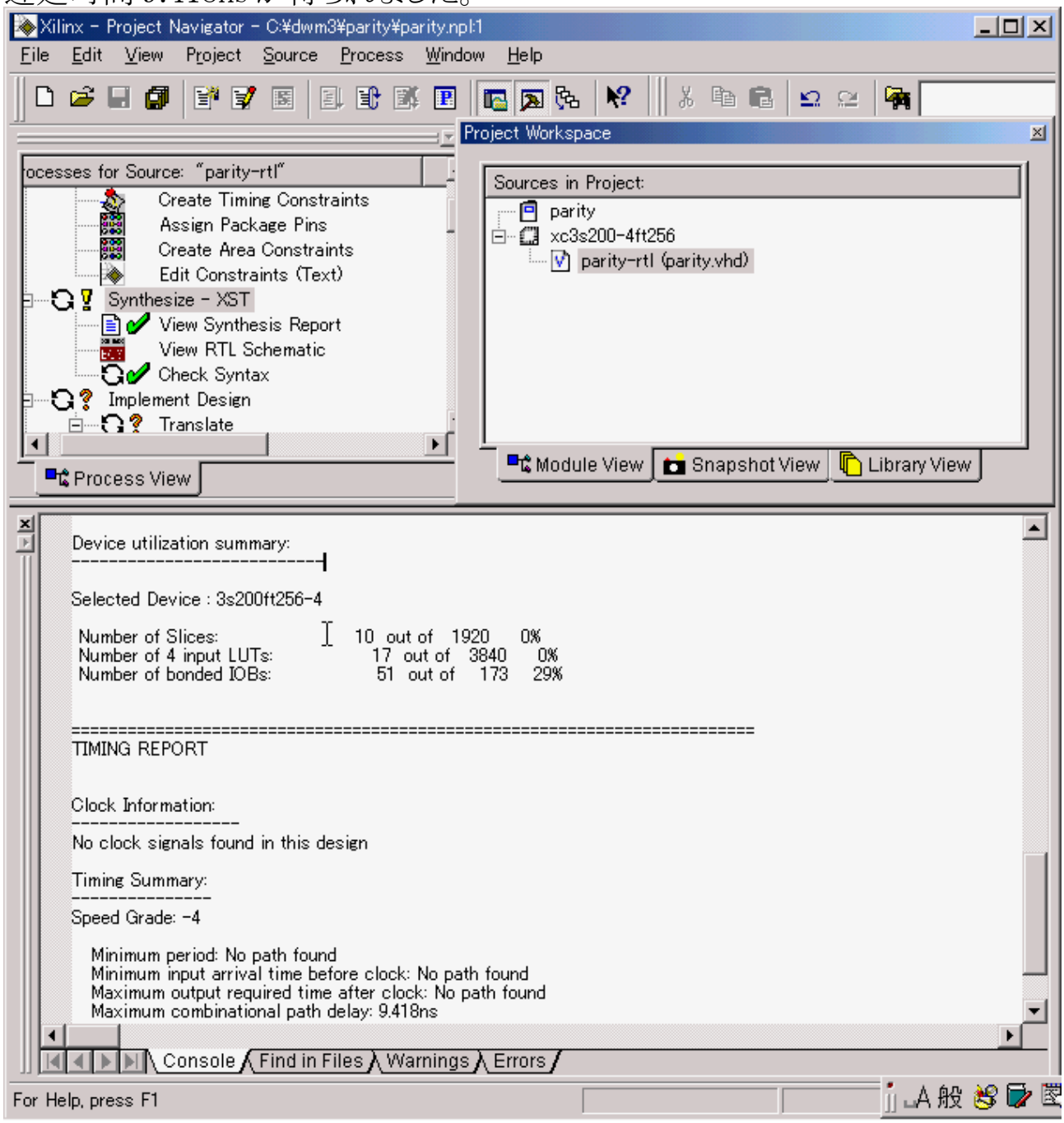
(1)3．2．1　FM 復調器(3．2．1)の合成結果



Xilinx　シンセサイザの合成結果で、配置配線前の値です。スライス数 25、速度 225MHz が得られました。

(2)リファレンス回路(parity.vhd)の合成結果

Xilinx シンセサイザの合成結果で、配置配線前の値です。スライス数 10、遅延時間 9.418ns が得られました。



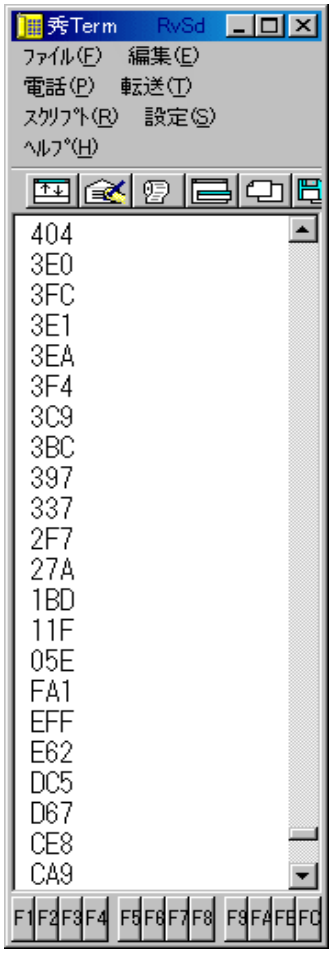
3．4．2　合成結果の検証

3．2．2の回路を合成しました。合成、ポストレイアウト後の速度は、FPGA 全体で、151MHz でした。ポストレイアウトゲート遅延シミュレーションを Veritak で行い、問題なく 150MHz で、RTL と同じ結果が得られることを確認しました。

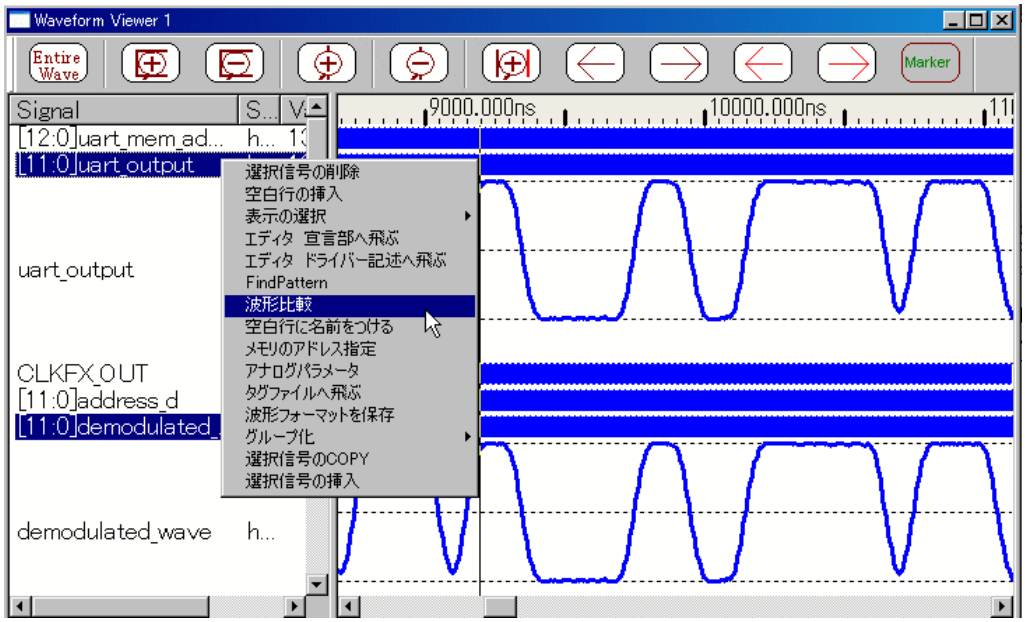
3．4．3　FPGA 実機検証

FPGA の UART から送られてくるデータを PC のターミナルソフトで拾った後、テキストファイルとしてセーブしました。セーブしたテキストファイルを Verilog で読み込み、RTL シミュレーション結果とコンペアし一致していることを確認しました。

ターミナルソフトで、***FPGA*** からの ***Data*** を受信



Spartan3-Starter Kit に ***RS232C*** ケーブルを接続



波形比較コマンドで、***UART*** からのデータ***(uart_output)*** と ***RTL　SIM*** との一致を確認

4. 提案方式の評価

項	評価
速度	Spartan3 Normal Speed ランクで 150MHz。情報転送レートで、課題例示より 1000 倍程度高速。
ゲート規模	25スライス。専用乗算器 2 個使用。
ユニーク	ユニーク。同方式の発表例はない。
実現	Xilinx Starter Kit、150MHz で実装検証済み
方式評価	18x18 専用乗算器で、テーブルルックアップでは対応できない 16 ビット幅にも対応できる。PLL 方式との違いは、Latency が殆どないこと。$\tan^{-1}$ の差分方式との違いは、位相の飛びを考慮する必要がないこと。

5. まとめ

FM 復調器として、FPGA の特徴を生かした、新しいデジタル復調方式を提案しました。VerilogHDL シミュレータ上で、

- システムシミュレーション
- RTL シミュレーション
- ポストレイアウトゲートシミュレーション

を行い方式の検証を行いました。また、FPGA 実機で、動作周波数 150MHz、ベースバンド転送レート 10Mbps の検証を行いました。

6. 参考文献

1:西村芳一； ディジタル FM レシーバの実現方式　デザインウェーブマガジン　2005　1 月号

7. 添付資料

1　FM 復調器ソースリスト
2　FPGA ソースリスト(トップ階層のみ)
3　テストベンチソースリスト

添付資料1　FM 復調器ソースリスト

```
//Jan.27.2005 Refine

//FM 復調器
module demodulator_hardware_new_sampling #(parameter Demo_MSB=7) (
                                input dcm_clock,//150MHz
                                input async_reset,
                                input signed [Demo_MSB:0] modulated_wave_sin,//150MHz
                                input signed [Demo_MSB:0] modulated_wave_cos,//150MHz
                                output signed [Demo_MSB+4:0] demodulated_wave_final);//150MHz

        reg signed [Demo_MSB:0] input_reg_sin,input_reg_cos;
        reg signed [Demo_MSB:0] sin_input,cos_input,cos_i,sin_i,cos_r,sin_r;
        reg signed [Demo_MSB:0] sin_z2,cos_z2, sin_z4,cos_z4,sin_z1,sin_z3,cos_z1,cos_z3;
        reg signed [Demo_MSB:0] sin_z5,cos_z5, sin_z6,cos_z6,sin_z7,sin_z8,cos_z7,cos_z8;

        reg signed [(Demo_MSB+1)*2-1:0] mult1,mult2;
        wire signed [(Demo_MSB+1)*2-1:0] sin_mul,cos_mul;
        wire signed [(Demo_MSB+1)*2-1:0] sin_ex_z4,cos_ex_z4;
        wire signed  [(Demo_MSB+1)*2-1:0] diff_reg;


       always @(posedge dcm_clock ) //復調器入力を一旦 FF で受ける
                input_reg_sin<=modulated_wave_sin;

       always @(posedge dcm_clock ) //復調器入力を一旦 FF で受ける
                input_reg_cos<=modulated_wave_cos;


       always @(posedge dcm_clock) begin//ディレイ  Z8 まであるが、合成されるのは、Z4 まで
                sin_z1<=input_reg_sin;
                sin_z2<=sin_z1;
                sin_z3<=sin_z2;
                sin_z4<=sin_z3;
                sin_z5<=sin_z4;
                sin_z6<=sin_z5;
                sin_z7<=sin_z6;
                sin_z8<=sin_z7;
                cos_z1<=input_reg_cos;
                cos_z2<=cos_z1;
                cos_z3<=cos_z2;
                cos_z4<=cos_z3;
                cos_z5<=cos_z4;
                cos_z6<=cos_z5;
                cos_z7<=cos_z6;
                cos_z8<=cos_z7;
       end

       assign sin_ex_z4=sin_z4;//アイパターン検討結果、Z4 を選択
       assign cos_ex_z4=cos_z4;//アイパターン検討結果、Z4 を選択


       always @(posedge dcm_clock ) begin
                begin
                        mult1<=input_reg_sin*cos_ex_z4;//符号付乗算器記述
                        mult2<=input_reg_cos*sin_ex_z4;//符号付乗算器記述
                end

       end
```

```
        mysub sub(//Xilinx パイプラインド減算モジュール  Coregen で生成
        .A(mult1),
        .B(mult2),
        .Q(diff_reg),
        .CLK(dcm_clock));    // synthesis black_box

        assign demodulated_wave_final=diff_reg[Demo_MSB+4+2:2];//復調器出力
endmodule
```

添付資料2　FPGA ソースリスト(トップ階層のみ)

```
//Jan.27.2005 Refine

//FPGA 回路記述モジュール
//RTL シミュレーション時は、HOST から IQ をもらう 、復調出力 12 ビット 150MHz
//実機では、内蔵 ROM により IQ を生成 UART 115.2Kbps で、RAM に Save された復調出力を出力
//
module hardware #(parameter INPUT_DATA_MSB=7,
                                parameter HOST_DATA_MSB=(INPUT_DATA_MSB+1)*2-1,
                                parameter ADDRESS_MSB=11,
                                parameter OUTPUT_DATA_MSB=11            )


                               (  input RST_In,
                                  input CLKIN_IN,
                                  input HOST_CLK,
                                  input [ADDRESS_MSB :0] HOST_ADDRESS,
                                  input HOST_WRITE_ENABLE,
                                  input [HOST_DATA_MSB :0] HOST_DATA,
                                  output signed [OUTPUT_DATA_MSB :0] output_data,
                                  output CLKFX_OUT,
                                  output TXD);


  reg [ADDRESS_MSB:0] address,address_d,address_dup1,address_dup2,address_dup3,address_dup4,address_dup5;

  wire signed [INPUT_DATA_MSB:0] data_in1,data_in2;


//uart
   wire [ADDRESS_MSB:0] address_for_uart;
   reg sync_reset_uart;
   wire clk0;
   wire signed  [OUTPUT_DATA_MSB:0] demodulated_wave_final;

//XilinX DCM モジュール: 50MHz システムクロックを入力 =>150MHz を出力
    mydcm dcm(.CLKIN_IN(CLKIN_IN),
            .RST_IN(RST_In),
            .CLKFX_OUT(CLKFX_OUT),
            .CLKIN_IBUFG_OUT(),
            .CLK0_OUT(clk0),
            .LOCKED_OUT());

//復調器モジュール: 150MHz クロックで、I/Q を入力、=>復調出力を 12 ビット幅で出力
 demodulator_hardware_new_sampling #(.Demo_MSB(INPUT_DATA_MSB)) demodulator(
                                .dcm_clock(CLKFX_OUT),//
                                .async_reset(RST_In),
                                .modulated_wave_sin(data_in1),//
                                .modulated_wave_cos(data_in2),//
                                .demodulated_wave_final(demodulated_wave_final));//
```

```
//Dual Port モジュール：1 2ビットx4 KWords
//復調出力 Save のための Dual Port RAM 入力は、復調出力（12ビット幅）=> 出力は、
//UART に出力しないのなら、RTL シミュレーション用に 150MHz で復調出力を出す
//UART 選択時は、UART_CNTRL は、アドレスを制御する
//`define UART_TX_OUTPUT
	`ifndef UART_TX_OUTPUT //UART を使わないのなら

		dualram  dual_portram(
		.addra(address_dup4),
		.addrb(address_dup5),
		.clka(CLKFX_OUT),
		.clkb(CLKFX_OUT),
		.dina(demodulated_wave_final),
		.dinb(),
		.douta(),
		.doutb(output_data),
		.wea(1'b1),//Jan.23.2005
		.web(1'b0));    // synthesis black_box

	`else//uart 出力

		dualram  dual_portram(
		.addra(address_dup4),
		.addrb(address_for_uart),
		.clka(CLKFX_OUT),
		.clkb(clk0),//uart を選択すると出力は 50MHz  Clock になるので
		.dina(demodulated_wave_final),//RAM 入力も 50MHz サンプリングになる。
		.dinb(),
		.douta(),
		.doutb(output_data),//RAM からの出力 Data
		.wea(1'b1),  //Jan.23.2005
		.web(1'b0));    // synthesis black_box

		uart_ctrl  uart_write_port(
					.ASYNC_RESET(RST_In),
					.clk_50M(clk0),
					.ram_data(output_data),
					.ram_address(address_for_uart),
					.TXD(TXD));


	`endif
```

```
//初期化付 RAM モジュール：16bits（IQ)x 4KWords
//`define USE_ROM_DATA  //HOST から生成 Data を使う場合には、コメントアウト
//RTL シミュレーション時は、HOST からの DATA(150MHz)をバッファリングする
//実機時は、HOST   DATA は、無視。RAM 初期値を使う

        myrom  rom(
        .addra(address_dup2),
        .addrb(HOST_ADDRESS),
        .clka(CLKFX_OUT),
        .clkb(HOST_CLK),
        .dina(),
        .dinb(HOST_DATA),
        .douta( {data_in2,data_in1}),
        .doutb( ),

        .wea(1'b0),

`ifdef USE_ROM_DATA //ROM を使うなら WRITE しない
        .web(1'b0));

`else
        .web(HOST_WRITE_ENABLE));

`endif




  always @(posedge CLKFX_OUT or posedge RST_In) begin
                if (RST_In) address<=0;
                else address<=address+1;

        end

        always @(posedge CLKFX_OUT) address_d<=address;
        always @(posedge CLKFX_OUT) address_dup1<=address;
        always @(posedge CLKFX_OUT) address_dup2<=address_dup1;
        always @(posedge CLKFX_OUT) address_dup3<=address_dup2;
        always @(posedge CLKFX_OUT) address_dup4<=address_dup3;
        always @(posedge CLKFX_OUT) address_dup5<=address_dup4;

endmodule
```

添付資料3　テストベンチ ソースリスト

```
//Jan.27.2005 Refine 2
//Jan.21.2005 Refine
//Jan.19.2005 最終版  Baseband/Demodulator の Eye  生成
//                                        Coe FILE  生成

`timescale 1ps/1ps
module hard_test_bench;
//Hard パラメータ
        parameter INPUT_DATA_MSB=7;
        parameter HOST_DATA_MSB=((INPUT_DATA_MSB+1)*2-1);
        parameter OUTPUT_DATA_MSB=11;
        parameter ADDRESS_MSB=11;


//テストベンチパラメータ
        parameter real Freq =150e6*1000/999;//150MHz Carrior Frequency
        parameter real Modulation =0.01;// +-1% GFSK Modulation
        parameter integer Clock_Period=10;//10ps 10ps resolution test bench
        parameter File_Name="bessel5.txt";//System Matrix A,B,C,D file for bessl filter
                                                                        // 3dB cutoff=5MHz
        parameter  integer Filter_Degree= 5;//System Matrix 's Degree
        parameter  integer CYCLE=Clock_Period*3*333 ;//
        parameter  integer BasebandCycle=1000*100/(Clock_Period*2);//10Mbps
        localparam integer Item_Counter_Max=2048*2-1;//4KB
        parameter integer IQ_Delay=1667;//90Degree shift at 150MHz
        parameter integer HOLD_TIME=1;

        real fc;//Carrier Frequency
        real fb;//BaseBand Frequency
        real fm;//Modulated Frequency

//rungekutta systems matrix input/output array
        real read_array[0:10];// Input vector for rungekutta
        real write_array[0:10];//Output vector from rugekutta

        real t;
        reg clock=0;
        reg signed [11:0] filt_out;//for monitoring

        real filt_outr;
        reg sync_reset1,sync_reset;

        real modulated_waveform;//変調波形
        reg  signed [INPUT_DATA_MSB:0] demodulator_input;//8 x2 ビットサンプリング信号
                                                                //  復調器の入力になる

        real theta,delta_theta,omega;//Dec.22.2004

        wire NRZ;//Baseband NRZ
        wire TXD;//UART 出力
        wire signed [OUTPUT_DATA_MSB:0] demodulated_wave;//復調器出力 Data
        integer fp=0;

//Hardware IF Signals
        reg clk=0;
        reg RST_In;

        reg  [ADDRESS_MSB:0] HOST_ADDRESS=0;
         reg [HOST_DATA_MSB:0] HOST_DATA=0;
```

```
        reg HOST_WRITE_ENABLE=0;
        wire CLKFX_OUT;
        wire dcm_clock=CLKFX_OUT;//150MHz FPGA output CLOCK
        wire  delayed_clock;
        wire  HOST_WRITE_CLOCK;

//システムシミュレーション用
        real cos_real,sin_real;
        real cos_real_z1,sin_real_z1;
        real diff_real;
        integer mag;


assign  #(HOLD_TIME) HOST_WRITE_CLOCK=delayed_clock;//RTL シミュレーション時、
                                                    //HOST に IQ を Write するためのクロック
assign #(IQ_Delay) delayed_clock =  dcm_clock;//CLKFX_OUT;//CLKFX_OUT を 90 度
                                                    //シフトした 150MHz  クロック

//テストベンチクロック  ベッセルフィルタの時間応答計算に使用するので、十分細かくする
//DCM に端数がでないように 3 の倍数(150/50)とすること。そうでないとジッタが発生する
        always #Clock_Period clock=~clock;

//FPGA システムクロック 50MHz Starter  Kit 仕様
        always #(CYCLE) clk=~clk;

//NRZ よりベッセルフィルタの時間応答を計算する。
//
//`define USE_ROM_DATA //ROM DATA を使うならこれを ON にする
`ifndef USE_ROM_DATA //ROM DATA を使わない場合、NRZ より送信変調波を作成する
                                        //  ROM  DATA を使う場合は、SIM 時間短縮のため
                                        //以下は実行しない
        always @(   clock) begin//Jan.19.2005

            t=$time()*1e-12;

            read_array[0]=(NRZ==1 ? 1 :-1);//1/0 => 1/-1 にマッピング
            $runge_kutta(File_Name,1);//アナログシミュレータで、ベッセルフィルタ出力計算
             filt_outr=write_array[Filter_Degree-1];//出力を REAL で得る  Get filtered outout
            filt_out=$rtoi($rint(filt_outr*1023));//フィルタ出力のモニタ用

            fc=Freq;
            fb=Freq*Modulation*filt_outr;
            fm=fc+fb;
            omega=2.0*fm*$M_PI;
            delta_theta=(Clock_Period)*1e-12*omega;//Jan.19.2005
            theta=theta+delta_theta;
            modulated_waveform=$sin(theta);//変調した送信波  150MHz
            //量子化した、変調した送信波  150MHz オーバフロー考慮、四捨五入後、整数変換
           demodulator_input=$rtoi($rint(modulated_waveform*(  2**INPUT_DATA_MSB -1.0)));
        end
`else

`endif

//coe file 生成
//`define MAKE_COE_FILE // XILINX  RAM  初期化ファイル生成を生成するなら ON
`ifdef MAKE_COE_FILE
        initial begin
                fp=$fopen("modulator_out.coe","w");
                $fwrite(fp,"memory_initialization_radix=16;\n");
                $fwrite(fp,"memory_initialization_vector=\n");
        end
`endif
        integer item_counter=0;
```

```
        integer dcm_instead_counter=0;
        integer cos_monitor;
        always @(posedge dcm_clock) begin//cos エッジ
                        #1;
                        HOST_DATA[INPUT_DATA_MSB:0]=demodulator_input;//Set  LSB WORD
                        cos_real_z1=cos_real;
                        cos_real=modulated_waveform;
                        cos_monitor=$rtoi(1000*cos_real);
        end

        always @(posedge delayed_clock) begin//sin エッジ
                        #1;
                        HOST_DATA[HOST_DATA_MSB: INPUT_DATA_MSB+1]=demodulator_input;
                        //Set MSB WORD
                        if (!HOST_WRITE_ENABLE) HOST_ADDRESS=0;//最初は0
                        else                                    HOST_ADDRESS=HOST_ADDRESS+1;
                        //次からインクリメント

                        HOST_WRITE_ENABLE=1;//Write Start
                        sin_real_z1=sin_real;
                        sin_real=modulated_waveform;
                        diff_real=cos_real*sin_real_z1 -sin_real*cos_real_z1;
                        diff_real=diff_real*1023*16;
                        mag=$rtoi(diff_real);

`ifdef MAKE_COE_FILE // XILINX   RAM   初期化ファイル生成を生成するなら
                        if (item_counter==Item_Counter_Max)       begin//RAM 最終アドレスまで書いたら
                                                                        //終わり
                                $fwrite(fp,"%4h;\n",HOST_DATA);
                                $fclose(fp);
                                $finish;
                        end else begin
                                $fwrite(fp,"%4h,\n",HOST_DATA);
                        end
`endif
                        item_counter=item_counter+1;
        end

//NRZ 生成
        baseband #(.counter_max(BasebandCycle)) base(clock,NRZ);//NRZ 生成

//アイパターン生成

`define EYE_BASEBAND //ベースバンドのアイパターンを生成するとき  これを ON
`define EYE_DEMODULATOR //復調器出力のアイパターンを生成するとき  これを ON

`ifdef EYE_BASEBAND
         eye_monitor #(.samples_per_scan(15*2*2),.max_columns(32)
         ,.Eye_Save_Data_File("eye_base.txt"))  eye_base(dcm_clock,filt_out);
`endif

`ifdef EYE_DEMODULATOR
         eye_monitor #(.samples_per_scan(15*2*2),.max_columns(32)) eye_demo(dcm_clock,mag);
         //mag,demodulated_wave
`endif
```

```
//Reset
        initial begin

                $runge_kutta(File_Name,0,read_array,write_array);//アナログシミュレータ
                                                                 //のオブジェクト初期化
                RST_In=0;
                #(10*1000);
                RST_In=1;
                #(100*1000);
                RST_In=0;

        end

//復調器
        hardware   #( .INPUT_DATA_MSB(INPUT_DATA_MSB),
                        .HOST_DATA_MSB(HOST_DATA_MSB),
                        .ADDRESS_MSB(ADDRESS_MSB),
                        .OUTPUT_DATA_MSB(OUTPUT_DATA_MSB)          )

                    dut  (  .RST_In(RST_In),//Reset
                       . CLKIN_IN(clk),//FPGA 入力 CLOCK=50MH z
                       . HOST_CLK(HOST_WRITE_CLOCK),// 復調器 RAM へ Write   入力 CLOCK
                       . HOST_ADDRESS(HOST_ADDRESS),//復調器入力   RAM   Address 指定
                       . HOST_WRITE_ENABLE(HOST_WRITE_ENABLE),//復調器入力 Write
                       .HOST_DATA(HOST_DATA), // 復調器入力 Data
                       .output_data(demodulated_wave),    //復調器出力 Data
                       .CLKFX_OUT(CLKFX_OUT),//150MHz FPGA 出力
                       .TXD(TXD));//UART 出力   115.2Kbps


//uart read port
  wire [7:0] buffer_reg;
  wire int_req;
  localparam LF=8'h0d;//改行コード

        always @(posedge clk, posedge RST_In) begin
                if (RST_In) sync_reset <=1'b1;
                else sync_reset<=1'b0;
        end

//`define READ_GENERATED_UART_DATA //FPGA   UART からのデータを TEXT   Save
                                                                    //=>uart.txt
//RTL とコンペアをとる場合は、コメントをトル
`ifdef READ_GENERATED_UART_DATA
        parameter uart_file="uart.txt";//実機が出力した UART   テキストファイル
        reg  [OUTPUT_DATA_MSB:0] uart_mem [0 :4096*2-1];//実機のデータ保持メモリ
        reg signed [OUTPUT_DATA_MSB:0] uart_output;//メモリ内容を引っ張り出す
        reg  [ADDRESS_MSB+1:0]  uart_mem_address=-(13'h100a-13'h0c70);
                                                          //垂れ流し UART   DATA の調整値

         initial begin
                $readmemh(uart_file,uart_mem);//uart_mem に読み込んだデータを格納
        end

     always @* uart_output=uart_mem[uart_mem_address];//150MHz   クロック同期
    always @(posedge      CLKFX_OUT) uart_mem_address<=uart_mem_address+1;//
`endif

   uart_read   uart_read_port( .sync_reset(sync_reset), .clk(clk)
        , .rxd(TXD),.buffer_reg(buffer_reg), .int_req(int_req));//実機の TX port のテスト用として
```

```
//UART からのデータを 12'h xxx CR で表示する
	always @(posedge int_req) begin
		begin :local

			reg [7:0] local_mem [0:3];
			integer i=0;

			if (i>=4) $stop;//Assert(0);

			if (buffer_reg==LF) begin :local2 //pop stack
				integer j;
				j=0;
				while( j < i) begin
					$write( "%c",local_mem[j]);
					j=j+1;
				end
				$write("     : time=%t\n",$time);
				i=0;//clear stack
			end else begin//push stack

				local_mem[i]=buffer_reg;
				i=i+1;
			 end
		end

	end

endmodule

//ランダム NRZ を生成する
module baseband
		#(parameter counter_max=1000)

		(
			input clock,
			output reg NRZ=1
		);

	localparam  integer MAX=counter_max-1;
	localparam integer initial_value_counts=4;//最初は、Bessel 安定期間
	localparam integer finish_value_counts=counter_max-5;//最後も1にしているが、
                                                          //位相があっていないので、Wrap 時に Noise がでる。

	reg [15:0] counter=0;
	reg [15:0] baseband_counter=0;

	always @(posedge clock) begin
		if (counter==MAX) counter<=0;//10Mbps counter
		else	counter<=counter+1;
	end

    always @(posedge clock) begin
		if (counter==MAX) begin
			baseband_counter <=baseband_counter+1;
			if ( baseband_counter <=initial_value_counts ||
			    baseband_counter >=finish_value_counts)  NRZ<=1;
			else NRZ<=$random %2==1;//ランダム   10Mbps NRZ

		end
	end



endmodule
```

```
//アイパターンを作る。EXCEL 読み込み用のコンマセパレート  テキストファイルを生成する
//EXCEL 読み込み時に、コンマ を指定すること。
module eye_monitor #(parameter samples_per_scan=32,//行数
                      max_columns=64,//列数
                      Eye_Save_Data_File="eye_demo.txt" //ファイル名
                                  )
                    (
                            input clock,
                            input signed [11:0] baseband_out
                    );



        integer  sample_counter=0;
        integer  column_counter=0;
        integer fi;
        integer i,j;

        reg  [11:0]  saved_wave [0:samples_per_scan][0:max_columns];


        always @(posedge clock) begin
                        saved_wave[sample_counter][column_counter] =baseband_out;
                        if  (sample_counter==samples_per_scan-1)  begin
                                        sample_counter<=0;
                                        column_counter<=column_counter+1;
                                        if (column_counter==max_columns) begin
                                                save_data_to_file;
                                                $stop;
                                        end
                        end else sample_counter <=sample_counter+1;
        end


        task save_data_to_file;
                begin
                        fi=$fopen(Eye_Save_Data_File);
                        for (i=0;i<samples_per_scan;i=i+1) begin
                                for (j=0;j< max_columns;j=j+1) begin
                                    if (j==0)                   $fwrite(fi,"%d ",$signed(saved_wave[i][j]));
                                    else if (j==max_columns-1) $fdisplay(fi,", %d ",$signed(saved_wave[i][j]));
                                    else            $fwrite(fi,",%d",$signed(saved_wave[i][j]));
                                end
                        end
                        $fclose(fi);
                end
        endtask

endmodule
```