BCH 符号エラー訂正回路　設計仕様書 (Ver 1.0)

琉球大学工学部情報工学科　和田　知久

---

[0] はじめに

さて、今年度の LSI デザインコンテスト設計課題ですが、昨年と同様に基本に帰って基本的なエラー訂正方式である BCH 符号にトライしたいと思います。BCH 符号は 1959 および 1960 年にボッケンジェム（Hocquenguem）およびボーズ（Bose）とチョドーリ（Chanduri）によって発見された符号です。ガロア体と代数計算で実装できますので、初心者のデジタル回路設計には打ってつけの課題です。学生対象のコンテストですので、小さめのデジタル回路を設計することを念頭に、符号長を実際に使用されている長さよりかなり短くし、取り扱いやすい大きさとします。また、いくつの設計オプションを設定し、各個人やチームによって色々な工夫ができるように、実現方法には自由度をある程度与えています。

要求されている設計内容は HDL（VHDL もしくは VerilogHDL）による設計と論理合成です。FPGA 等の合成ツールでも参加できます。HDL 設計に興味のある学生はどしどし参加してください。また、余裕のある方は FPGA 等で実装すれば、努力を認めて高い評価が得られると思います。FPGA 等での実装もぜひトライしてみてください。

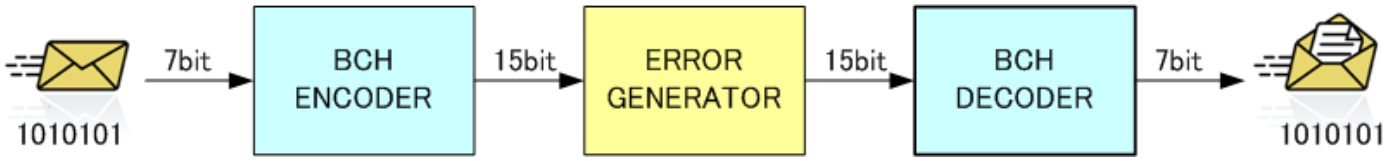

図1　送受信システム図

図1に今回設計するデジタルデータの送受信システムのブロック図を示します。システムは大きく分けて、符号化器（ENCODER）と解読器（DECODER）に分かれますが、今回の設計課題は解読器（DECODER）です。実際の伝送システムを想定し、伝送途中でビットエラーが発生することを過程しています。7 ビット長であるアスキーコードの送信も可能なように、7ビットの情報ビットに8ビットのパリティビットを追加し、15ビットの符号語を生成する BCH(15,7)符号とします。この BCH 符号では 15 ビット中 2 ビットのランダム誤りを訂正できますので、比較的強力なエラー訂正能力をもちます。図1では Error Generation ブロックがエラーを発生させます。

受信機ではこのエラーを含んだ 15 ビットの符号を受け取り、エラー訂正処理を行います。エラー訂正処理には有名なところではリード･ソロモン方式などがありますが、処理が複雑で学生対象のコンテストにはやや不向きです。しかしながら、このサイズのちいさな BCH 符号方式では、除算回路と ROM などでエラー訂正をやることができ、回路的にはシンプルです。

今回の課題では BCH 符号の知識がなくても設計できるように、以下の仕様書で工夫をしています。設計すべきことは比較的単純ですので、デジタル設計の知識のある学生は自信を持って、課題に取り組んでください。

---

[1] BCH(15,7)符号のイントロダクション

BCH 符号を文献等で調査するとガロア体の多項式等をもちいたややこしい理論が説明されています。BCH 符号のアルゴリズムを理解するには、ガロア体の多項式からは逃れられません。しかし、ここでは符号理論に詳しくない方、むしろデジタル回路設計者のような他分野の方に理解しやすいように、ガロア体の多項式とデジタル回路の実装を対応づけて説明し、BCH 符号を用いたエラー訂正方法を紹介します。

今回設計するシステムでは先にも説明したように、7 ビットの情報ビットに対して 8 ビットのエラー訂正用のビットを付加し、15 ビットのデータを送信します。表 1 に、7 ビットの情報ビットとして(1)={0, 0, 0, 0, 0, 0, 0}, (2)={1, 1, 0, 1, 0, 1, 1}, (3)={1, 1, 1, 1, 1, 1, 1}の3種類の情報を送る場合の例を示します。エラー訂正用の 8 ビットは表2のようになります。この生成方法は後ほど説明します。

		エラー訂正用(8 ビット)								情報ビット(7 ビット)						
(1)	情報ビット(7 ビット)									0	0	0	0	0	0	0
	送信ビット(15 ビット)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2)	情報ビット(7 ビット)									1	1	0	1	0	1	1
	送信ビット(15 ビット)	1	1	0	0	0	1	0	0	1	1	0	1	0	1	1
(3)	情報ビット(7 ビット)									1	1	1	1	1	1	1
	送信ビット(15 ビット)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

表 1　情報ビット(7 ビット)と送信ビット(15 ビット)の例

---

[2] 送信ビットの生成方法

送信回路や受信回路のシンドローム計算回路では多項式を割り算し余りを求める回路が必要になりますので、多項式を用いた説明を以下行います。

7 ビットの情報ビットを{u0, u1, u2, u3, u4, u5, u6 }とし、以下の多項式 U(x)で表します。ここで多項式を用いていますが、変数xを解くような意味はなく、X^n がそのビットの位置を表しています。

$$U(x) = u_0 + u_1 \cdot x^1 + u_2 \cdot x^2 + u_3 \cdot x^3 + u_4 \cdot x^4 + u_5 \cdot x^5 + u_6 \cdot x^6 \quad \text{---- (1)}$$

そうすると、15 ビットの送信ビットは以下のような多項式 SB(x)で示すことができます。

$$SB(x) = u_0 \cdot x^8 + u_1 \cdot x^9 + u_2 \cdot x^{10} + u_3 \cdot x^{11} + u_4 \cdot x^{12} + u_5 \cdot x^{13} + u_6 \cdot x^{14}$$
$$+ r_0 + r_1 \cdot x^1 + r_2 \cdot x^2 + r_3 \cdot x^3 + r_4 \cdot x^4 + r_5 \cdot x^5 + r_6 \cdot x^6 + r_7 \cdot x^7 \quad \text{---- (2)}$$

これを表で示すと、以下の表2のようになります。表2からもわかりますが、情報ビットを右側に配置しましたので、Xの時数 8 次から 14 次の部分の情報ビットが割り当てられ、0 次から 7 次にエラー訂正用のパリティビットが割り当てられています。

	エラー訂正用（8 ビット）								情報ビット（7 ビット）						
情報ビット（7 ビット）									u0	u1	u2	u3	u4	u5	u6
送信ビット（15 ビット）	r0	r1	r2	r3	r4	r5	r6	r7	u0	u1	u2	u3	u4	u5	u6
X の次数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

表2　情報ビット（7ビット）と送信ビット（15ビット）と多項式の係数

すなわち、

$$R(x) = r_0 + r_1 \cdot x^1 + r_2 \cdot x^2 + r_3 \cdot x^3 + r_4 \cdot x^4 + r_5 \cdot x^5 + r_6 \cdot x^6 + r_7 \cdot x^7 \quad \text{---- (3)}$$

を求めれば、U(x)が与えられているので、容易に SB(x）を生成することが可能となります。実は R(x)は

$$U(x) \cdot x^8 = u_0 \cdot x^8 + u_1 \cdot x^9 + u_2 \cdot x^{10} + u_3 \cdot x^{11} + u_4 \cdot x^{12} + u_5 \cdot x^{13} + u_6 \cdot x^{14} \quad \text{---- (4)}$$

を

$$G(x) = 1 + x^4 + x^6 + x^7 + x^8 \quad \text{----(5)}$$

で割り算を行った余りとなります。 ここで、G(X)のことを生成多項式と呼びます。

（4）式は U(x)に X^8 をかけたものですが、この X^8 をかけることにより、情報ビットが上位に8ビットシフトしたことになります。すなわち、デジタル回路での n ビットシフト処理は、ガロア多項式ではXの n 乗の乗算にあたることになります。

（5）式の次数は8次なので、（4）式を（5）式で割った余りは7次式以下の次数となり、（3）式の R(X)の次数と合います。但し、ガロア体という普通の数学ではない割り算の余りを求めることになりますが、ガロア体での多項式の余りを求める方法は実は単純であり、

$$x^8 = 1 + x^4 + x^6 + x^7 \quad \text{----(6)}$$

を用いて、8 次の項を7次以下の式への変換を繰り返し用いることで、(4)式の次数を 7 次以下に下げるように変換することで求めることができます。但しこのとき、係数どうしの加算や減算は EXOR で代用します。すなわち、

$$x^2 + x^2 = (1 \oplus 1) \cdot x^2 = 0 \cdot x^2 = 0$$
$$x^2 + x^2 + x^2 = (1 \oplus 1 \oplus 1) \cdot x^2 = 1 \cdot x^2 = x^2 \quad \text{----(7)}$$

のように計算します。

---

[3] 符号化器(ENCODER)の回路構成と動作説明

図2に送信機の回路図を示します。クロックに同期して、SIN 入力より情報ビットを u6, u5, ..., u0 の順に入力します。このとき2つのスイッチは A 側に接続しておきます。そうすると、SB 出力端子より u6, u5, ..., u0 がそのまま出力されると同時に r7, r6, ..., r0 フリップフロップと排他的論理輪 EXOR で構成されるガロア体の割り算の余り計算回路で(3)式で示される係数が計算されます。

$$G(x) = 1 + x^4 + x^6 + x^7 + x^8 \quad \text{----(5)}$$

で割り算を行いますので、

$$x^8 = 1 + x^4 + x^6 + x^7 \quad \text{----(6)}$$

なる式を用いて式の次数を下げることになります。すなわち、SIN からの信号は8次の係数に対応し、8次の係数が1の場合、0次、4次、6次、7次の係数に’1’を EXOR とることになります。回路をみるとまさに(6)式で示す計算をやっていることになります。結果的に、IN に7個の値をすべて入力すると、r7, r6, ..., r0 なるフリップフロップにガロア体の割り算の余りの係数が発生します。

その後にスイッチを B 方向に接続すると、r7, r6, ..., r0 なる係数は右方向へ順にシフトし、OUT 端子から出力されます。この詳しい動作タイミングを図3に示します。

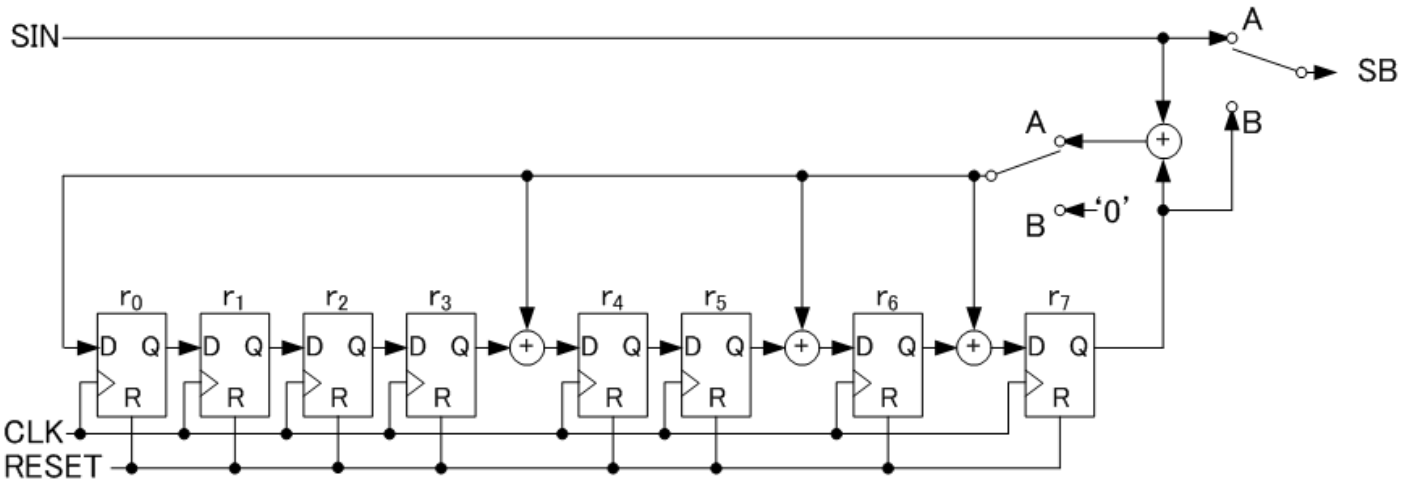

図2　送信機の回路図

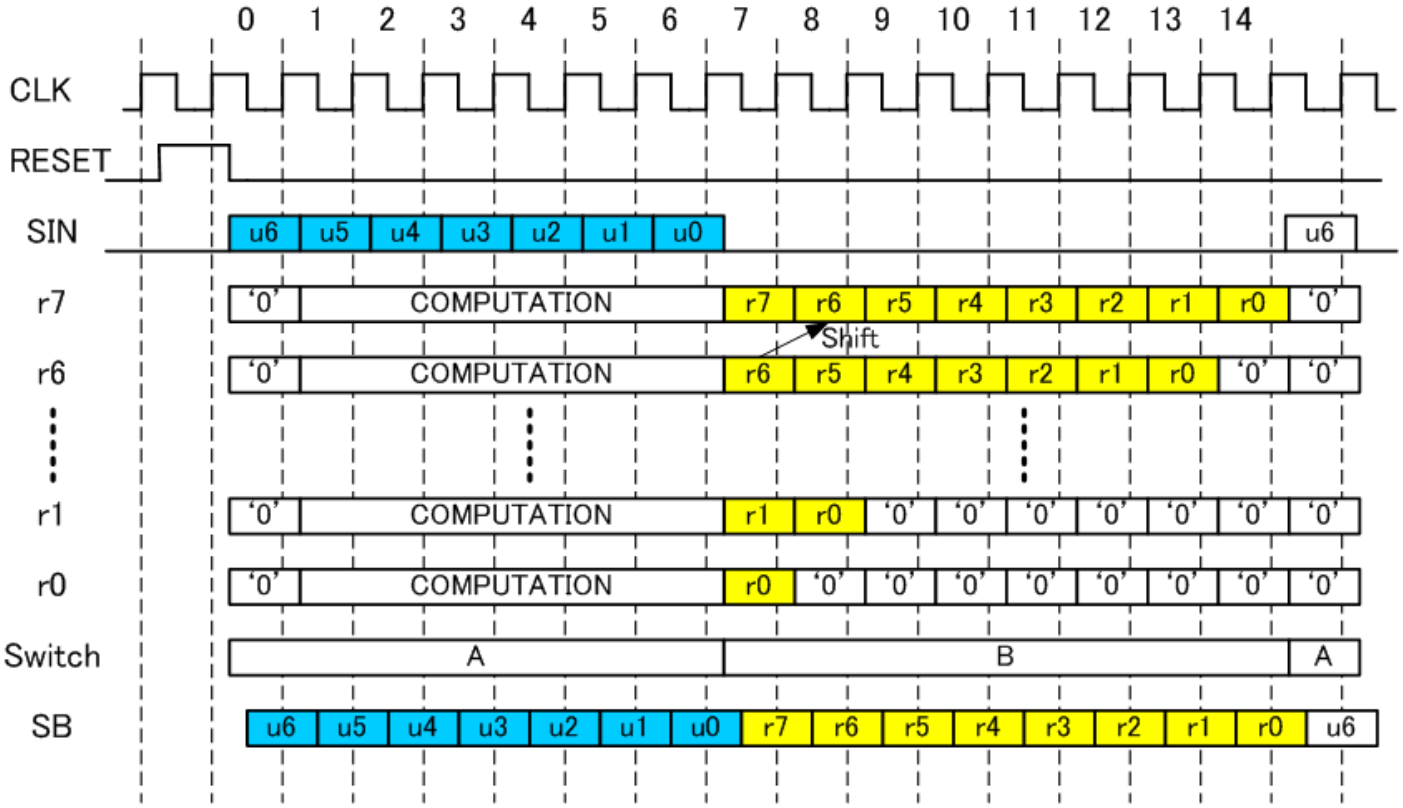

図3　送信機の動作波形図

図3では RESET 信号を用い、RESET 信号により r7, r6, ..., r0 なるフリップフロップの同期リセットも行っています。

表3および表4に情報ビットが(0,0,0,0,0,0,1) の場合と、(1,0,0,0,0,0,0) の場合のレジスタの値の時間的変化を示しておきます。

情報ビットが(0,0,0,0,0,0,1) の場合は、u0=1 すなわち、8次の係数が1であり、

$$x^8 = 1 + x^4 + x^6 + x^7 \quad \text{----(6)}$$

すなわち、r7=r6=r4=r0=1 なる処理を1回やっていることが理解できると思います。

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
AB Switch	–	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B
IN	–	u6 0	u5 0	u4 0	u3 0	u2 0	u1 0	u0 1	–	–	–	–	–	–	–	–
r7	0	0	0	0	0	0	0	1								
r6	0	0	0	0	0	0	0	1								
r5	0	0	0	0	0	0	0	0								
r4	0	0	0	0	0	0	0	1								
r3	0	0	0	0	0	0	0	0								
r2	0	0	0	0	0	0	0	0								
r1	0	0	0	0	0	0	0	0								
r0	0	0	0	0	0	0	0	1								
OUT	–	u6 0	u5 0	u4 0	u3 0	u2 0	u1 0	u0 1	r7 1	r6 1	r5 0	r4 1	r3 0	r2 0	r1 0	r0 1

表 3　情報ビット=(0,0,0,0,0,0,1) を送信する場合の送信回路の動作

情報ビットが(1,0,0,0,0,0,0)の場合は、u6=1 すなわち、14 次の係数が1であり、

$$x^8 = 1 + x^4 + x^6 + x^7 \quad \text{----(6)}$$

なる処理をサイクル1で行うことで、その後の 6 サイクルでシフト動作を行うことで等価的に、

$$x^{14} = x^8 \cdot x^6 = (1 + x^4 + x^6 + x^7) \cdot x^6 \quad \text{----(8)}$$

やっていることが理解できると思います。

以下に動作を説明しますので、これを丁寧に理解することで、ENCODE 方法を理解することができると思います。

Cycle=1 では、u6=1 が入力され、r7=r6=r4=r0=1 なる処理を行います。

Cycle=2 では、まずシフト動作により r7=r5=r1 をたてようし、u5=0 とシフトであふれた r7=1 の exor＝1により、r7= r6= r4= r0 に1を立てようとしますが、この両者の exor により結果的には r7=0, r6=r5=r4=1, r3=r2=0, r1=r0=1 という状態になります。

Cycle=2から Cycle=3 への変化時には、u4=0 でかつ以前の r7=0 であったので、exor=0 となり、ここではシフト動作のみが発生します。

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
AB Switch	–	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B
IN	–	u6 1	u5 0	u4 0	u3 0	u2 0	u1 0	u0 0	–	–	–	–	–	–	–	–
r7	0	1	0	1	0	0	0	1								
r6	0	1	1	1	0	0	1	1								
r5	0	0	1	1	0	1	1	1								
r4	0	1	1	0	1	1	1	0								
r3	0	0	0	0	1	1	0	1								
r2	0	0	0	1	1	0	1	0								
r1	0	0	1	1	0	1	0	0								
r0	0	1	1	0	1	0	0	0								
OUT	–	u6 1	u5 0	u4 0	u3 0	u2 0	u1 0	u0 0	r7 1	r6 1	r5 1	r4 0	r3 1	r2 0	r1 0	r0 0

表 4　情報ビット=(1,0,0,0,0,0,0) を送信する場合の送信回路の動作

---

[4]　解読器（DECODER）のアルゴリズム

さて、いよいよ DECODER の仕様を説明します。ここでは、アルゴリズムの詳細は説明せず、どのような演算をもちいればエラー訂正できるかという方針で、説明を行います。今回の課題である BCH(15,7)符号では以下の生成多項式を用いました。

$$G(x) = 1 + x^4 + x^6 + x^7 + x^8 \quad \text{----(5)}$$

$$SB(x) = u_0 \cdot x^8 + u_1 \cdot x^9 + u_2 \cdot x^{10} + u_3 \cdot x^{11} + u_4 \cdot x^{12} + u_5 \cdot x^{13} + u_6 \cdot x^{14} + r_0 + r_1 \cdot x^1 + r_2 \cdot x^2 + r_3 \cdot x^3 + r_4 \cdot x^4 + r_5 \cdot x^5 + r_6 \cdot x^6 + r_7 \cdot x^7 \quad \text{---- (2)}$$

すなわち、符号を示す多項式 SB(x)は、（2）式の形をしているので、生成多項式 G(x)で割るとあまりが0となる性質を持っています。すなわち、割り算のあまりを計算して0であれば、エラーがないと判断することができます。

BCH(15,7)符号ではデータが 7 ビット、冗長（パリティ）ビットが 8 ビットで、15 ビットの符号中 2 ビットの誤り訂正が可能な符号です。

以下（9）式に示すように、生成多項式 G(x)は

$$G_1(x) = 1 + x + x^4$$
$$G_2(x) = 1 + x + x^2 + x^3 + x^4$$
$$G(x) = G_1(x)G_2(x)$$
$$= (1 + x + x^4)(1 + x + x^2 + x^3 + x^4)$$
$$= 1 + (1 \oplus 1)x + (1 \oplus 1)x^2 + (1 \oplus 1)x^3 + (1 \oplus 1 \oplus 1)x^4 + (1 \oplus 1)x^5 + x^6 + x^7 + x^8$$
$$= 1 + x^4 + x^6 + x^7 + x^8 \quad \text{----(9)}$$

となり、G1(x)と G2(x)の積です。したがって、SB(x)は G1(x)でも、G2(x)でも割り切れることになります。すなわち、エラーがなければ、2つの余りは0となります。

送信信号 SB(x)を送信して、RB(x)なる信号を受信したとします。送信中にエラーがなければ、RB(x)は SB(x)と同一であり、RB(x)を G1(x)もしくは G2(x)で割ると余りは0となります。もし送信中にエラーが発生すると、RB(x)を G1(x)もしくは G2(x)で割った余りは0ではなく、エラーの発生を検知することができます。

この RB(x)を G1(x)で割った余りを S1(x)シンドロームとし、G2(x)で割ったあまりを S2(x)シンドロームとすると、S1(x)および S2(x)は以下のような 3 次の多項式となります。

$$S_1(x) = s_{10} + s_{11} \cdot x^1 + s_{12} \cdot x^2 + s_{13} \cdot x^3$$
$$S_2(x) = s_{20} + s_{21} \cdot x^1 + s_{22} \cdot x^2 + s_{23} \cdot x^3 \quad \text{---(10)}$$

2つのシンドロームビット(S10, S11, S12, S13)と(S20, S21, S22, S23)の計8ビットにより、エラーのあるなし、エラー訂正可能かどうか、エラー訂正可能な場合に、反転すべきビット位置はどの位置かを知ることができます。ここでは、複雑なアルゴリズムでの解法はあきらめ、テーブルでの実装をします。というのも、シンドロームはすべてで8ビットであり、256通りの場合しかなく、Table すなわち ROM 等で実装が可能だからです。

表5にシンドロームとエラー位置の関係を示します。黄色のセルはシンドロームがともに0の場合に対応しており、エラーのない場合で、None と表示されています。青色のセルは1ビットエラーに対応しており、符号語15ビットのエラーの位置15通りを示しています。エラー位置は表6に示すように、(2)式のXの指数と合わせています。赤色のセルは2ビットエラーに対応しており、15*14/2=105 通りのパターンを示しています。各セルには2つの数値がしめされており、これらが2ビットエラーの位置に対応します。他のセルは3ビット以上のエラーがあり、エラー訂正不可能の状態を示しています。エラー発生は検知したが、訂正できない状態で135通りあります。

Error Position		S2(x) (S_{20},S_{21},S_{22}, S_{23})															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
S1(x) (S_{10},S_{11},S_{12}, S_{13})	0000	None															
	0001	8,13	3					2,6	0,14			10,12	1,9	5,11	4,7		
	0010	7,12		2			3,6		4,10		0,8		9,11	1,5		13,14	
	0011	1,11			2,3	6			5,9		0,13	7,10			4,12	8,14	
	0100	6,11			12,13	1			0,4		8,10	2,5			7,14	3,9	
	0101	4,11			8,12		1,3	2,11			10,13	0,7		5,6			9
	0110	0,10			7,13		3,11	1,2		5			6,9		12,14	4,8	
	0111	1,6			7,8	11			10,14		3,5	0,12			2,9	4,13	
	1000	5,10			2,8		6,13	11,12		0			1,4		7,9	3,14	
	1001	4,9			2,13		5,8	1,7			0,3	5,12		10,11			14
	1010	3,13	8					7,11	4,5			0,2	6,14	1,10	9,12		
	1011	3,8	13					1,12	9,10			5,7	4,11	0,6	2,14		
	1100	9,14			3,7		11,13	6,12			5,8	2,10		0,1			4
	1101	2,12		7			8,11		0,9		5,13		1,14	6,10		3,4	
	1110	0,5			3,12		1,8	6,7		10			11,14		2,4	9,13	
	1111	2,7		12			1,13		5,14		3,10		4,6	0,11		8,9	

表 5　シンドロームとエラー位置の対応表

	X^{14}	X^{13}	X^{12}	X^{11}	X^{10}	X^{9}	X^{8}	X^{7}	X^{6}	X^{5}	X^{4}	X^{3}	X^{2}	X^{1}	1
SB	u6	u5	u4	u3	u2	u1	u0	r7	r6	r5	r4	r3	r2	r1	r0
Bit position	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

表6　エラー位置

---

[5]　解読器(DECODER)の設計例

前セクションで説明したように、15ビットの符号を G1(x), G2(x)で割ったあまり、すなわちシンドロームを求め、そのシンドロームから表5を参照すれば、エラー訂正が可能となります。

以下図4にシンドローム計算回路の例を示します。2種類のシンドロームを計算するために、2つのリニアフィードバックシフトレジスタを並列に配置した構成にしました。リニアフィードバックレジスタのフリップフロップは t**と記載されており、最終的な計算結果は s**レジスタに現れます。図5はこの回路の動作波形図を示したもので、最初の11サイクルはスイッチ＝Aであり、フィードバック動作をしています。これは最初の11サイクルは14次から4次までの次数であり、シンドロームの次数より大きいので、フィードバック

処理を行っているからです。その後の4サイクルはスイッチ＝Bであり、3次以下に対応しており、単なる exor による加算を実施し、結果を S**シフトレジスタに転送しています。

先頭を示すために START=1 がアサートされ、START は t**のフリップフロップの出力を AND ゲートにより’0’にすることで、初期状態からの計算開始を実現しています。図5からもわかるように、14サイクル目に s**レジスタにシンドロームが計算されます。

この値を用いて、表5に示すテーブルを検索すれば、訂正すべきエラー発生ビットが判明し、エラー訂正を行うことができます。

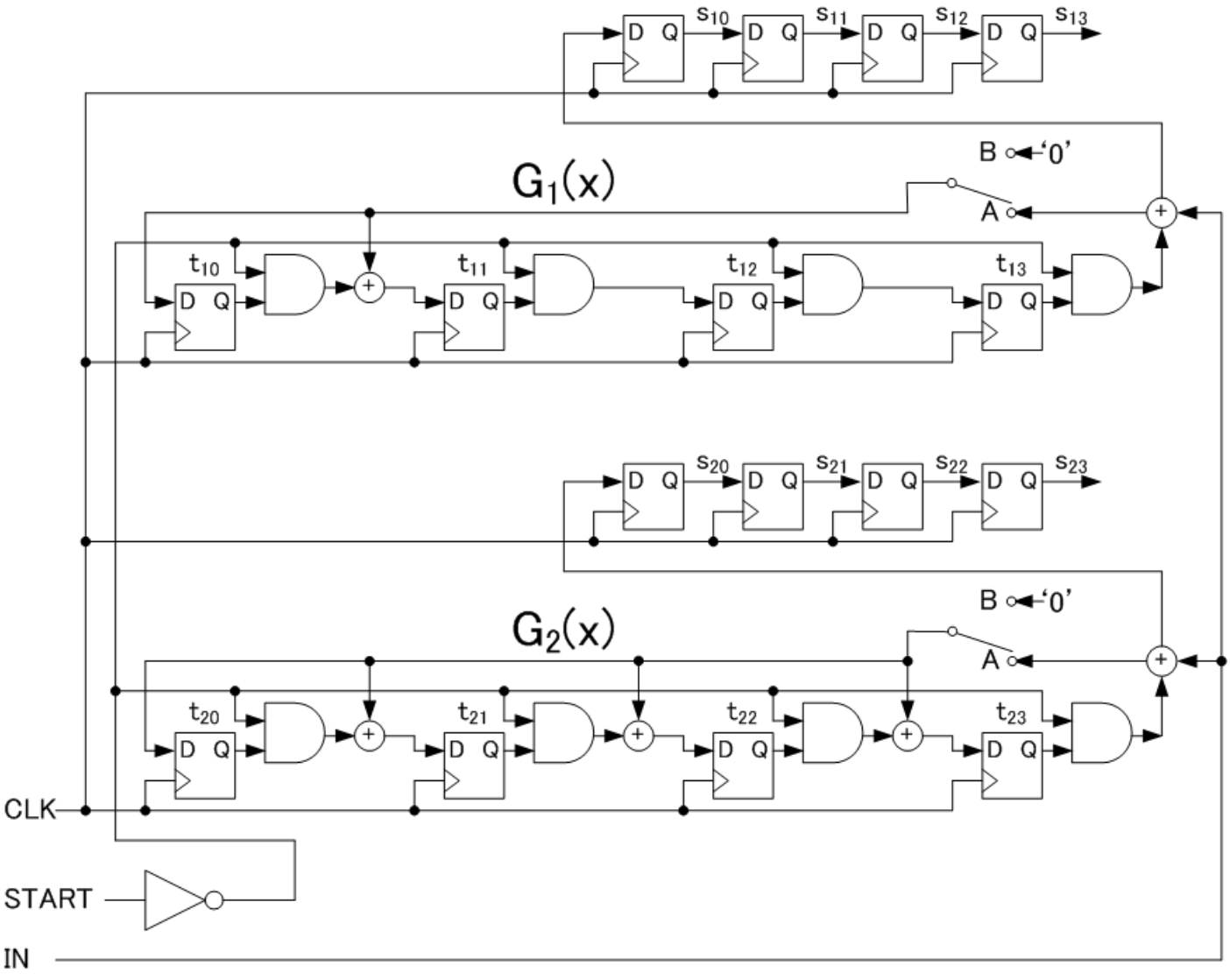

図4　シンドローム計算回路

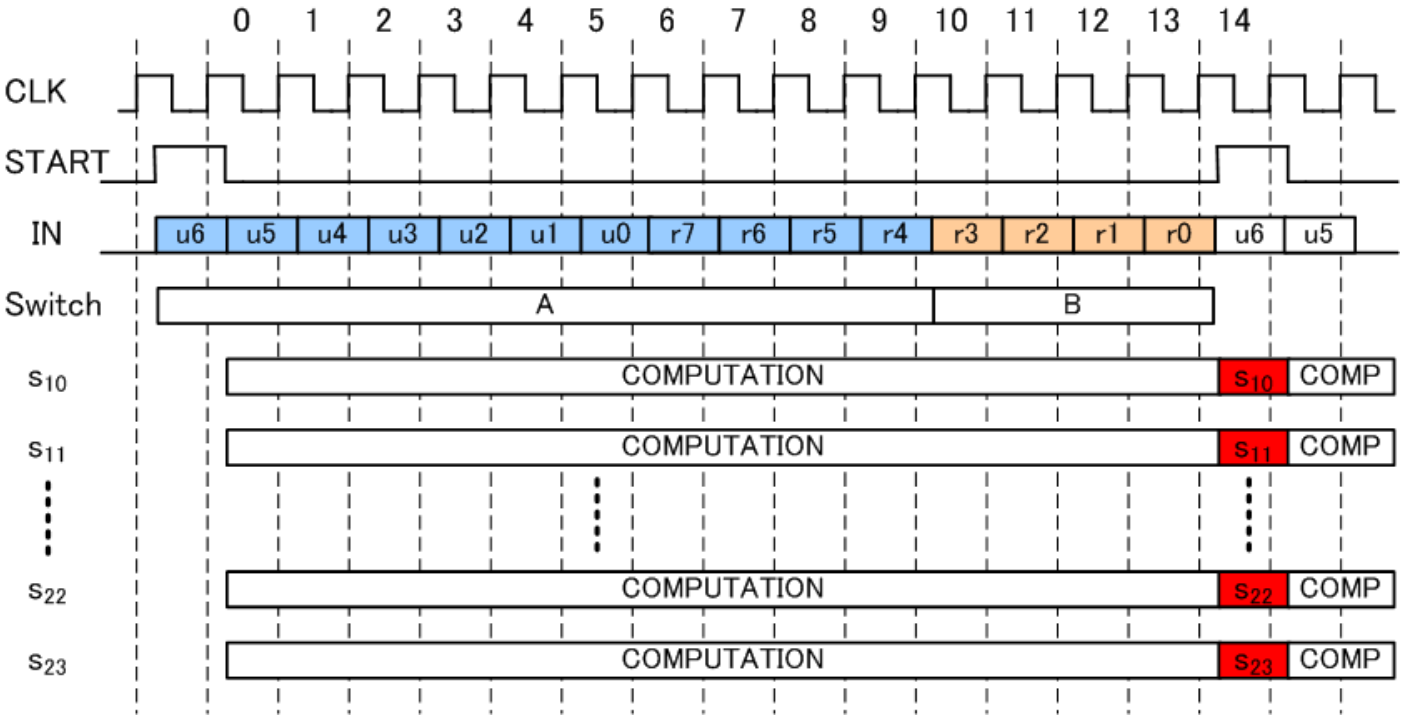

図5　シンドローム計算回路動作波形図

以下では、シンドローム計算の途中を EXCEL を用いて示して説明をします。図6は黄色のセルに示される(1,0,0,0,0,0,0,1,1,1,0,1,0,0,0)の入力から、シンドローム計算を実行している例です。この入力にはエラーがなく、計算されたシンドローム（赤色のセル）の値はともに0となっています。

図7は黄色のセルに示される(1,0,0,1,0,0,0,1,1,1,0,1,1,0) の入力からシンドローム計算を実行している例です。この入力は r2(ビット位置 2)と u3（ビット位置 11）に2ビットのエラーがあります。シンドローム計算の結果(s10, s11,s12,s13)=(0101), (s20, s21,s22,s23)=(0110)が計算されています。この2つのシンドローム値より、表5を参照すると2，１１となり2箇所を訂正することで、エラー訂正をすることができます。

上記で説明した、シンドローム計算回路の他に、表5に対応するテーブル回路（ROM や組み合わせ回路）や、入力データを記憶しておいて、エラー訂正としてビット反転を行う回路が必要です。

Syndrome Computation

1) INPUT 15bit code in yellow cells
2) Get Syndrome OUTPUTs in RED cells

	INPUT			G1(x)				G2(x)			
				t10	t11	t12	t13	t20	t21	t22	t23
				0	0	0	0	0	0	0	0
14	u6=	1	0	1	1	0	0	1	1	1	1
13	u5=	0	1	0	1	1	0	1	0	0	0
12	u4=	0	2	0	0	1	1	0	1	0	0
11	u3=	0	3	1	1	0	1	0	0	1	0
10	u2=	0	4	1	0	1	0	0	0	0	1
9	u1=	0	5	0	1	0	1	1	1	1	1
8	u0=	0	6	1	1	1	0	1	0	0	0
7	r7=	1	7	1	0	1	1	1	0	1	1
6	r6=	1	8	0	1	0	1	0	1	0	1
5	r5=	1	9	0	0	1	0	0	0	1	0
4	r4=	0	10	0	0	0	1	0	0	0	1
3	r3=	1	11								
2	r2=	0	12								
1	r1=	0	13								
0	r0=	0	14	0	0	0	0	0	0	0	0
				s10	s11	s12	s13	s20	s21	s22	s23

図6　シンドローム計算例（エラーなしの場合）

			Syndrome Computation												
	1) INPUT 15bit code in yellow cells														
	2) Get Syndrome OUTPUTs in RED cells														
					G1(x)						G2(x)				
					t10	t11	t12	t13			t20	t21	t22	t23	
	INPUT				0	0	0	0			0	0	0	0	
14	u6=	1		0	1	1	0	0			1	1	1	1	
13	u5=	0		1	0	1	1	0			1	0	0	0	
12	u4=	0		2	0	0	1	1			0	1	0	0	
11	u3=	1		3	0	0	0	1			1	1	0	1	
10	u2=	0		4	1	1	0	0			1	0	0	1	
9	u1=	0		5	0	1	1	0			1	0	1	1	
8	u0=	0		6	0	0	1	1			1	0	1	0	
7	r7=	1		7	0	0	0	1			1	0	1	0	
6	r6=	1		8	0	0	0	0			1	0	1	0	
5	r5=	1		9	1	1	0	0			1	0	1	0	
4	r4=	0		10	0	1	1	0			0	1	0	1	
3	r3=	1		11											
2	r2=	1		12											
1	r1=	0		13											
0	r0=	0		14	0	1	0	1			0	1	1	0	
					s10	s11	s12	s13			s20	s21	s22	s23	

図7　シンドローム計算例（2ビットエラーの場合）

以下のこの計算を実行した EXCEL ファイルをリンクしておきます。自由にご活用ください。

- シンドローム計算:　syndrome.xls

---

[6]システム構成

今回設計するシステムの構成を図8に示します。システムは大きくわけて、ENCODER と DECODER、そして受信機でエラー訂正された受信機の出力（RBEC）のエラーの数をカウントするための ERRCNT（エラーカウント機）からなり、設計してもらうのは、DECODER と ERRCNT です。実際に回路合成を行い、回路規模やスピードを求めるのは RECEIVER だけとなり、ERRCNT は自分で設計した RECEIVER のエラー訂正の能力の評価に使用します。

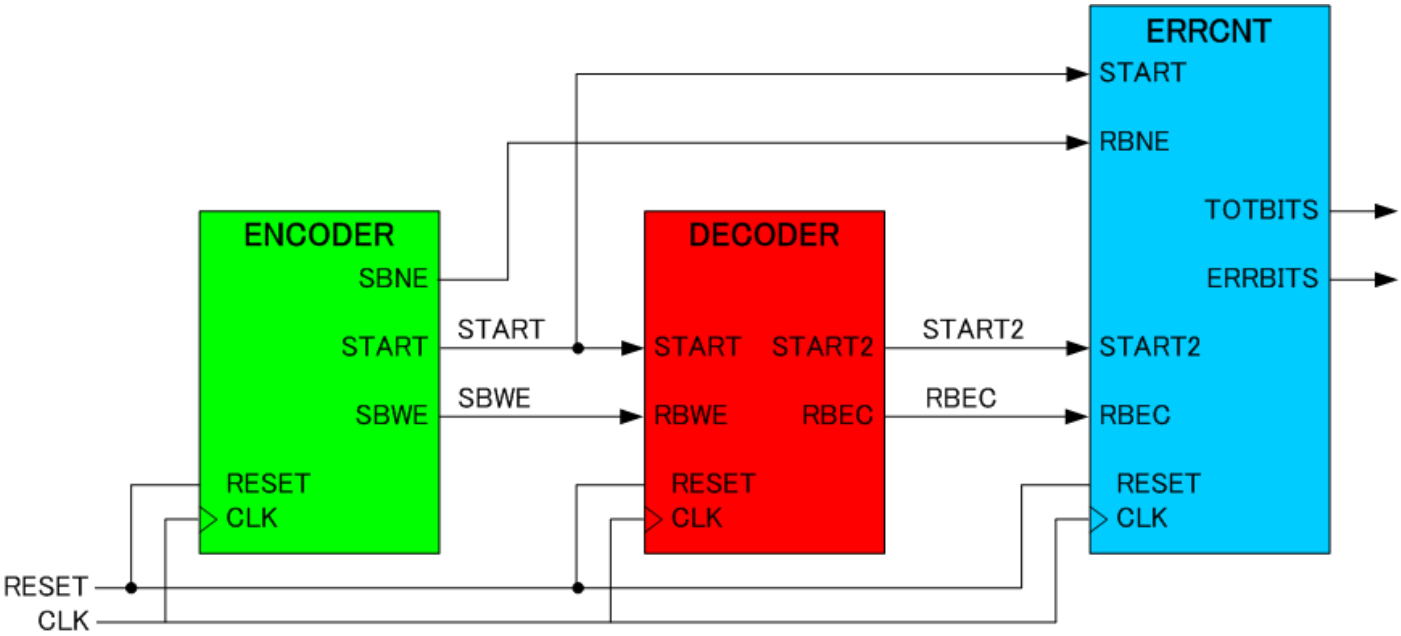

図8　システム構成図

（**ENCODER の説明**）

ENCODER は送信側であり、7 ビットの情報ビット列 U(x)を生成し、8 ビットのエラー訂正用ビット列 R(x)を付加して、15 ビットの送信データ SB(x)を生成します。

さらに ENCODER は適当なエラーを発生させ、その出力を SBWE(Send Bit With Error)端子に出力します。SBWE には連続的に1かOの信号がクロック CLK に同期して出力されます。この 15 ビットの送信は連続的に発生し、常に 15 サイクルごとに 7 ビットの情報ビット列と 8 ビットのエラー訂正用ビット列が送信されます。

START 信号は 15 サイクル中 1 サイクル期間のみ、1を出力し、15 ビットの送信データの先頭位置を示します。

SB は RECEIVER によるエラー訂正がうまくできているかを調べる目的に用意された信号であり、SBWE 端子の出力に対応したエラーが含まれてない正しい信号を出力します。実際の通信システムではこのような信号はありませんが、ここではエラー訂正の評価のために特別に用意します。

（**DECODER の説明**）

DECODER は受信側であり、クロックに同期して、ENCODER からの送信信号 SBWE を受け取ります。15 サイクルごとにエラー訂正を行い、エラー訂正された 15 ビットを RBEC 端子に出力します。この出力の 15 ビットにはエラー訂正用ビットも含まれますが、すべてのエラー訂正を行い出力します。また、RBEC の先頭位置を示すために、START2 信号を用います。これは START 信号を DECODER 内でのエラー訂正のレイテンシー遅延させた信号です。

（**ERRCNT**：エラーカウント機）

ERRCNT は受信機の評価用のブロックであり、回路合成等は行いません。ENCODER より START 信号とエラーのない送信信号 SB を受け取ります。したがって、いつも正しい情報ビットを知ることができます。また、ERRCNT は DECODER よりエラー訂正された情報ビットを RBEC 端子より受け取ります。ERRCNT は正しい情報ビットとエラー訂正された情報ビットを比較を行い、内部に持つ適当な大きさのレジスタに累積のエラーの発生回数を記憶し、外部へ ERRBITS 端子を通して出力します。同時に、累積のトータル受信ビット数を外部へ TOTBITS 端子を通して出力します。したがって、ERRBITS/TOTBITS を計算すればビットエラー率を知ることができます。

DECODER はエラー訂正出力を出すまでのレイテンシー（遅延）がありますので、ERRCNT はその遅延を考慮してエラー数をカウントするように設計してください。

---

[7]動作タイミング例

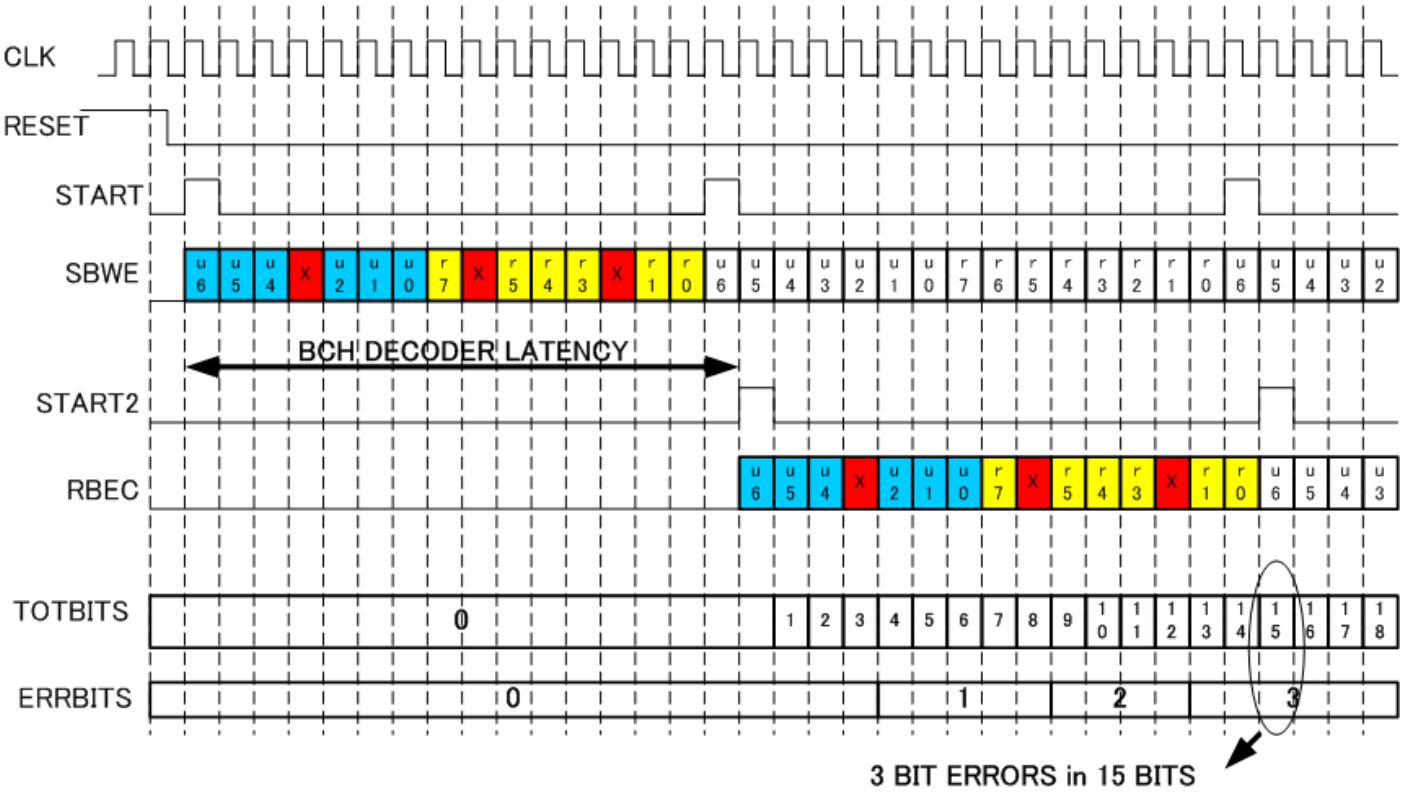

図9　システム動作タイミング例

図9に動作タイミングの例を示します。RESET 信号の解除の後、ENCODER は SBWE（Sent Bits With Error）信号、SB（Sent Bits without error）信号、START 信号を出力します。15 サイクルがひとつの単位となっており、最初の 7 サイクルで情報ビットが、その後の 8 サイクルでエラー訂正用ビットが送信されます。SBWE はエラー付の信号であり、ある確率でエラーが発生します。詳しくは与えられた ENCODER の VHDL 記述を見てください。

DECODER は SBWE 信号を受信し、エラー訂正を行い、RBEC（Received Bits Error Corrected）にエラー訂正をした情報ビットを出力します。エラー訂正用ビットもエラー訂正処理を行い、ERRCNT で計測できるようにします。

ERRCNT は 15 ビットの情報ビットおよぼパリティビットに対するエラーの累積値と処理したトータルビット数を示すものです。

---

[8]LEVEL1：初心者用課題

初心者用 DECODER は以下のピン配置をもつこととします。

Receiver			
信号名	入出力	ビット幅	説明
CLK	IN	1	クロック入力
RESET	IN	1	'1'でリセット
START	IN	1	情報ビットの先頭位置を示す
RBWE	IN	1	エラー付の受信信号
START2	OUT	1	エラー訂正後の情報ビットの先頭位置を示す
RBEC	OUT	1	エラー訂正処理された受信信号

表7　RECEIVER の初心者用ピン配置

以下に送信機 ENCODER、全体シュミレーション、ならびに受信機のテンプレートの VHDL ファイルをリンクします。必要に応じて修正して使用してください。

- 送信機: encoder.vhd
- 全体シミュレーション：　testbench_bch.vhd
- 設計すべき受信機のテンプレート：　decoder.vhd

以下図10にVHDLシミュレーションの波形の一部を示します。情報ビット=(1,0,0,0,0,0,0) を送信する場合の送信回路の動作を示しています。

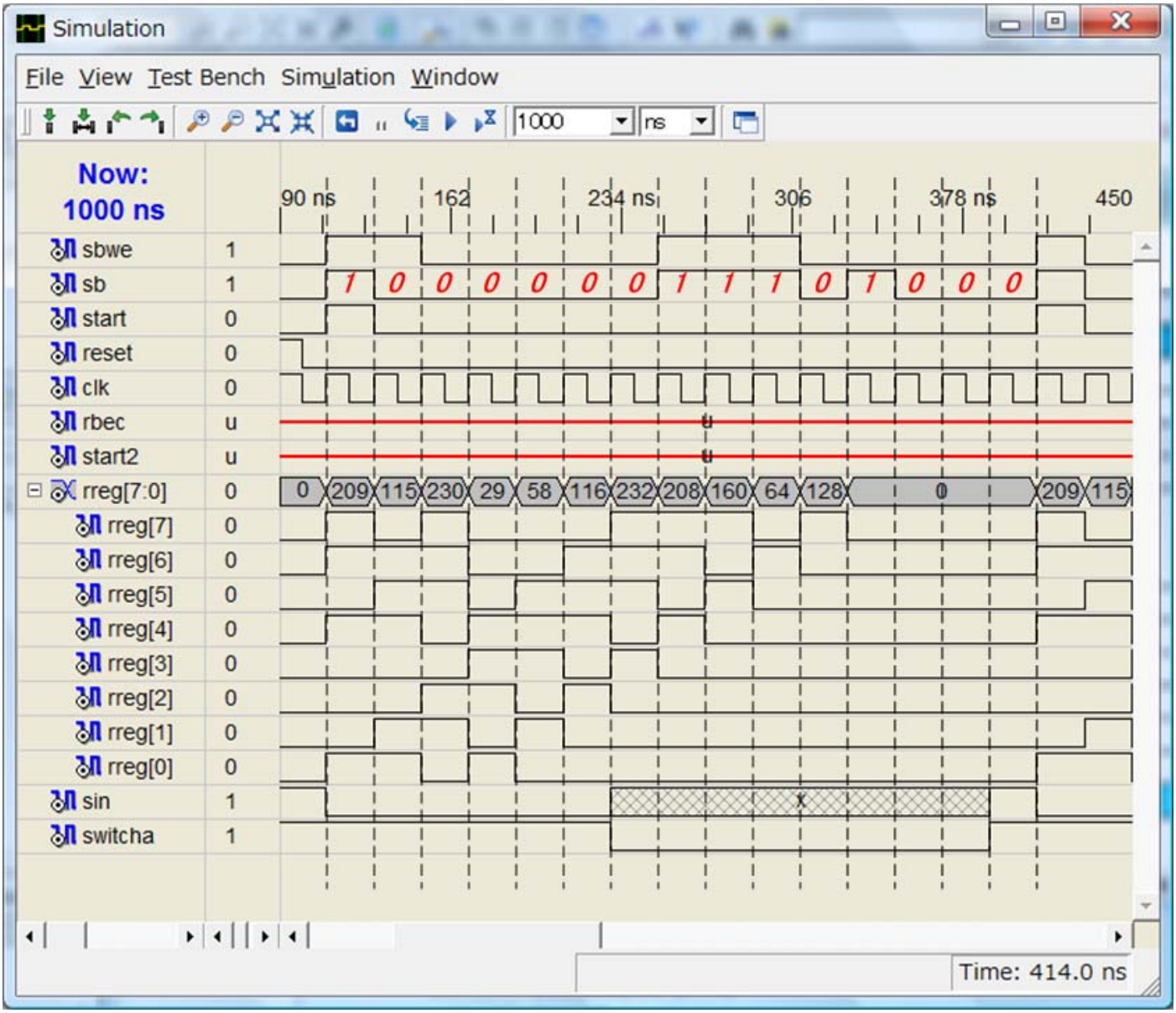

図10　VHDL シミュレーション例

---

[9]LEVEL2：上級者用課題

上級者用課題では、基本課題を自由に変更してオリジナリティのある設計を行ってください。

期待される内容としては、

1）シンドロームからエラー訂正位置の計算に、例では Table を用いましたが、エラー訂正の教科書等には別のアルゴリズミックな方法も用いられています。ROM に代わる実装も大歓迎です。

2）例題では、15ビットの符号はシリアル転送していますが、パラレル転送にし、非常な短時間でのエラー訂正処理の実現も面白い有効なアプローチです。

3）その他、自由に面白い工夫を行ってください。

---

[10] スピードおよび回路規模の測定単位

色々なメンバーが参加し、回路規模やスピードの比較が困難な状況になっていますが、せめてもの比較のために、50 入力の EXOR 回路の回路規模と速度を単位面積、および単位時間とします。あなたが設計した、DECODER の面積（回路規模）が 50EXOR の何倍であるか？　速度が何倍であるかをレポートしてください。

50入力の EXOR 回路：　parity.vhd

---

[11] レポート

レポートには以下の内容を含めること。また、ページ数を少なめにコンパクトにまとめること。

表紙	1	代表者の氏名、チーム名、大学院修士/大学学部生/高専生の区別
	2	代表者の連絡先、学年、学校名、住所、電話、email 等
	3	共同設計者全員の名前（最高3名まで）
	4	取り組んだ課題（LEVEL1／LEVEL2）
内容	1	設計した回路ブロックの構成説明（ブロック図と説明）
	2	設計した回路ブロックの動作説明（動作波形図やパイプライン動作等の説明）
	3	工夫した点、オリジナリティを出した点（アピールが重要！）
	4	クリティカルパスのスピード、論理合成後の回路規模
	5	VHDL もしくは Verilog のコード
	6	正常動作している VHDL/Verilog シミュレーション波形
	7	その他自由意見など

レポートは PDF ファイルにて下記に EMAIL にて提出すること！

締め切りは 2010 年1月 29 日（金）必着です。

---

[12] 審査のポイント

- 速度、回路規模だけでなく、アーキテクチャのユニークさ、アイデア、面白さを十分考慮して審査します。
- 大学院修士、学部生、高専生のレベルに応じて審査をします。
- 仕様書に従ってまじめに作るのも良いが、オモロイアイデアを歓迎します。他人と違ったことをしよう！
- 仕様の部分変更など、柔軟に受け付けます。

ENJOY　HDL!　沖縄で会おう！