「Design Wave 設計コンテスト2002」の実施要領

Design Wave Magazine 編集部では，昨年度に引き続き，「Design Wave 設計コンテスト2002」を行います．

■ ねらい

ハードウェア設計は，HDL を使用する方法が主流となってきましたが，HDL の文法やツールの使いかたを学ぶことはできても，実際にあるシステムの要求仕様から設計を進め，実際に動作する回路を実現するまでを経験する機会がない，という方は少なくないでしょう．また，同じ仕様書で，ほかの設計者はどのように解決するのか知りたい，みずからの設計技術が客観的にどれくらいか知りたい，と思われている方もおられるのではないでしょうか．若い人たちに，回路設計の多様性，おもしろさをもっと理解してほしい，と思っている先輩設計者の方も多いでしょう．

そこで，Design Wave Magazine では設計コンテストを行いたいと考えました．より多くの方に「ハードウェア・システム設計」に参加していただき，ご自分の設計力やアイデアをアピールしてみてはいかがでしょうか．少し競争しながら設計するのも，きっと楽しいことだと思います．

■ 種目

ハードウェア設計者のキャリアが短い方や学生の方でも気軽に参加できるように，シンプルで具体的な課題が用意されています．また，初心者がより参加しやすいように，初心者向けコースも用意されています．参加資格は，学生と社会人を区別する以外は特に設けません．また，社会人のみ，匿名での参加も受け付けます（連絡用に本名の明記は必要です）．

― 課題　差集合巡回符号エラー訂正回路の設計 ―

テレビの文字多重放送に用いられているエラー訂正方式である差集合巡回符号（Difference-set Cyclic Code）を用いたエラー訂正回路の設計を行います．設計するのは受信機の回路です．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にしています．

設計仕様の詳細は，次ページからの記事で解説します．

■審査基準

審査は，基本的に次の項目を基準として行います．

(1) 速度，(2) ゲート規模，(3) ユニーク性，(4) 実現

「速度」と「ゲート規模」は，各参加者から提出されたシミュレーション結果で判定します．各参加者が使用する開発環境は異なりますので，審査時にそのことは考慮されます．「ユニーク性」とは，おもにアーキテクチャを評価するものです．再利用性やハードウェア回路らしいユニークなアーキテクチャなどを評価します．「実現」とは，実際に基板上に回路を実現し，動作させることです．論理合成だけで終わるのではなく，実際のPLD/FPGA（基板）上で実現し動作させた方は，審査の評価の対象となります．

上記のように，審査は，必ずしも数値的な要素だけで優劣を決めるとはかぎりません．結果的に提出いただくレポート自体も評価対象となります．あらかじめ，ご了承ください．

審査は，設計者，研究者の方から構成された，Design Wave 設計コンテスト審査委員会で行うことになります．

■ スケジュール

締め切りは，

2002年1月31日

といたします．ファイルによるE-mail 送付または郵送で受け付けます．

発表は，

本誌2002年5月号（2002年4月10日発売予定）

を予定しています．

優秀作品については，その製作レポートを本誌で掲載することがあります．

■ 説明

本コンテストは，琉球大学工学部情報工学科と共同で進めていきます．同学科から発表されている学生向けの「シリコン・シーベルト・デザイン・コンテスト2002」と同じ課題です．Design Wave 設計コンテストについて，学生（大学，大学院，工業高等専門学校など）の方が参加される場合は，琉球大学側で審査を行い，最終審査に残った場合は，琉球大学で行われるデザイン・コンテスト2002 最終発表会（2002年3月8日予定）に招待されます．社会人の方が参加される場合は，CQ 出版社側で審査し，優秀な設計をされた方には，社会人設計の代表として，琉球大学での上記発表会に招待いたします．

■ 賞品

優秀な設計をされた方には，賞品を贈呈します．前回の賞品は以下のとおりでした．今回も前回同様に，結果発表時点における注目製品を贈呈します．

― 2001年の課題1（CDMA 電話機）―

社会人部門

第1位　沖縄3泊4日旅行およびネットワークウォークマン
第2位　16インチ液晶モニタ
第3位　ゲームボーイアドバンス
第4位　ゲームボーイアドバンス

学生部門

You are No.1 賞	5万円相当
Originality Design 賞	3万円相当
High Performance 賞	3万円相当
Interesting Design 賞	3万円相当

1次審査通過チーム　琉球大学における発表会への招待

― 2001年の課題2（RSA 暗号器）―

第1位　16インチ液晶モニタ
第2位　ネットワークウォークマン
第3位　ネットワークウォークマン

なお，コンテストのためのホームページは，

http://www.cqpub.co.jp/dwm/contest/

に設置しています．お問い合わせは，E-mail で contest.dwm@cqpub.co.jp までお願いします．（編集部）

Design Wave 設計コンテスト 2002

差集合巡回符号 エラー訂正回路設計仕様書

Difference-set Cyclic Code

和田知久

今回はテレビの文字多重放送に用いられているエラー訂正方式である差集合巡回符号（Difference-set Cyclic Code）を用いたエラー訂正回路の設計を行います．とはいえ，コンテストの課題ですので，小さめのディ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，いくつかの設計オプションを設定し，各個人やチームによっていろいろなくふうができるように，実現方法にはある程度自由度を与えています．要求される設計内容は，HDL（VHDL もしくはVerilog HDL）による設計と論理合成です．論理合成ツールに制約はなく，無償で提供されている論理合成ツールでも参加できます．HDL 設計に興味のある方はどしどし参加してください．また，余裕のある方はFPGA などへ実装すれば，努力を認めて高い評価が得られると思います．ぜひトライしてみてください．

課題の概要

図1 に今回設計するディジタル・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，送信機（transmitter）と受信機（receiver）に分かれますが，今回の設計課題は受信機（receiver）です．

送信者はビット列を出力します．送信機はシリアルに入力されるビット列11 ビットに対して差集合巡回符号の符号化処理（encode）を行い，11 ビットの入力にエラー訂正用のビット列10 ビットを付加して21 ビットの符号とします．この21 ビットが伝送されるわけですが，伝送中にビット・エラーが発生します．**図1** ではエラー・ジェネレータのブロックがエラーを発生させます．

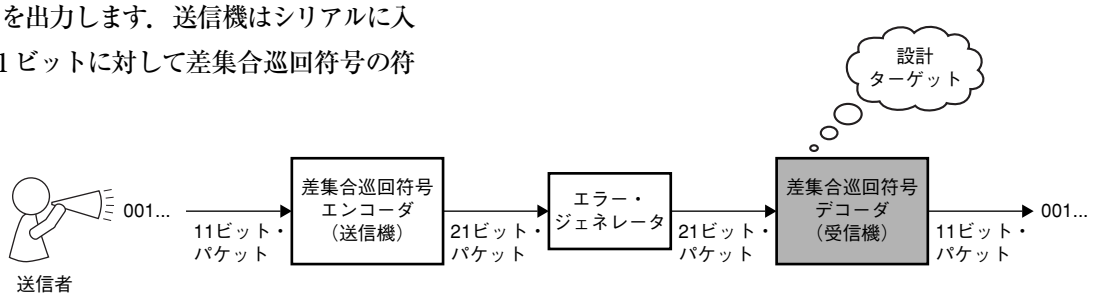
受信機ではこのエラーを含んだ21 ビットの符号を受け取り，エラー訂正処理を行います．エラー訂正処理には有名なところではリード・ソロモン方式などがありますが，処理が複雑でコンテストの課題には不向きです．しかし，差集合巡回符号方式では，多数決回路でエラー訂正を行うことができ，回路的には非常にシンプルです．

今回の課題では差集合巡回符号の細かい知識がなくても設計できるように，仕様をくふうしています．設計すべきことは比較的単純ですので，ディジタル設計の知識のある方は自信を持って，課題に取り組んでください．

差集合巡回符号について

●パリティ・チェックの復習

ディジタル・データの伝送におけるエラーの有無をチェックする場合，パリティ・チェックという方式がよく用いられます．**表1** に，8 ビットのデータに1 ビットのパリ



〔図1〕
送受信システム図

ティ・ビットを付加した例を示します．8ビット・データの中の‘1’の個数が偶数であればパリティ・ビットは‘0’であり，‘1’の個数が奇数であればパリティ・ビットは‘1’となります．したがって，パリティ・ビット生成回路は非常に簡単であり，多入力のXOR（排他的論理和）となります．

したがって，パリティ・ビットを付加した9ビット幅のデータを送信し，受信側で｛D_7，D_6，D_5，D_4，D_3，D_2，D_1，D_0，$Parity$｝のXORをとれば，その出力が‘1’のときはどこかにエラーが発生したことが検知でき，XORの出力が‘0’のときはエラーが発生しなかったと判断できます．ただし，2ビットのエラーは検知できませんし，送信した9ビットのうちどのビットがエラーで反転したかを知ることはできません．

●差集合巡回符号のイントロダクション

差集合巡回符号を文献などで調査すると，ガロア体の多項式などを用いたややこしい理論が説明されています．しかし，ここでは符号理論に詳しくない方，むしろディジタル回路設計者のような方に理解しやすいように，差集合巡回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，11ビットの情報ビットに対して10ビットのエラー訂正用のビットを付加し，21ビットのデータを送信します．この21ビットには以下に示すようなおもしろい特性があります．

表2に，11ビットの情報ビットとして｛1，1，1，1，1，1，1，1，1，1，1｝というすべて‘1’からなる情報を送る場合の例を示します．エラー訂正用の10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．

この21ビットの送信ビットに対して以下の**表3**に示すような5種類A_1～A_5のパリティ・チェックを考えます．

パリティ・チェックA_1ではビット番号で9，12，13，18，20のところに1があります．この五つの位置に対応する送信ビット$SB(9)$，$SB(12)$，$SB(13)$，$SB(18)$，$SB(20)$のXORを取ると‘0’になります．同様にパリティ・チェックA_2～A_5でもすべて‘0’となります．まとめると**表4**のようになります．

ここで送信ビットの20ビット目$SB(20)$がエラーで反転した場合を考えます．$SB(20)$は5種類のパリティ・チェック式にすべて含まれています．したがって，五つのパリティ・チェック式A_1からA_5はすべて‘1’となります．

また，送信ビットの20ビット目以外で1ヵ所のエラーが発生した場合を考えます．**表3**を見てわかるように，送信ビットの20ビット目以外のビットは5種類のパリティ・チェック式のどれかに1回のみ使用されます．つまり，A_1からA_5のうちどれか一つが‘1’となります．

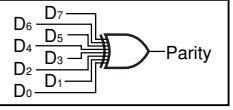
したがって，上記パリティ・チェック式を計算し，その多数が‘1’となると$SB(20)$にエラーが発生していることが検知され，$SB(20)$のビットを反転することでエラーの訂正を行うことができます．

本設計コンテストでは，

$$A_1 + A_2 + A_3 + A_4 + A_5 \quad >= 3$$

〔表1〕8ビット・パリティの例とパリティ発生回路

8ビットのデータ								パリティ・ビット	パリティ発生回路
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Parity	
1	0	1	1	0	0	0	1	0	
0	0	0	0	0	0	0	0	0	
0	1	0	0	0	0	0	0	1	



〔表2〕情報ビット（11ビット）と送信ビット（21ビット）の例

	エラー訂正用（10ビット）										情報ビット（11ビット）										
情報ビット（11ビット）											1	1	1	1	1	1	1	1	1	1	1
送信ビット（21ビット）	0	0	1	1	0	0	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1

〔表3〕情報ビット（11ビット）と送信ビット（21ビット）の例

ビットの位置番号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
送信ビット（*SB*）	0	0	1	1	0	0	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1
パリティ・チェックA_1										1			1	1					1		1
パリティ・チェックA_2		1										1			1	1					1
パリティ・チェックA_3					1		1										1			1	1
パリティ・チェックA_4	1					1		1										1			1
パリティ・チェックA_5			1	1					1		1										1

〔表4〕5種類のパリティ・チェック

パリティ・チェックA_1	A_1=*SB*(9) xor *SB*(12) xor *SB*(13) xor *SB*(18) xor *SB*(20)=0
パリティ・チェックA_2	A_2=*SB*(1) xor *SB*(11) xor *SB*(14) xor *SB*(15) xor *SB*(20)=0
パリティ・チェックA_3	A_3=*SB*(4) xor *SB*(6) xor *SB*(16) xor *SB*(19) xor *SB*(20)=0
パリティ・チェックA_4	A_4=*SB*(0) xor *SB*(5) xor *SB*(7) xor *SB*(17) xor *SB*(20)=0
パリティ・チェックA_5	A_5=*SB*(2) xor *SB*(3) xor *SB*(8) xor *SB*(10) xor *SB*(20)=0

であれば，すなわち三つ以上のパリティ・チェック式がエラーを検知した場合に，エラー訂正を行うことにしておきます．このようにすることで，2ビットのエラー発生に対して訂正を行えるようになります．

●ほかのビットのエラー訂正

上記の例ではSB(20)のエラー訂正の方法を説明しました．このセクションではほかのビット番号の場所のエラー訂正の方法を説明します．

表5に$SB(19)$を訂正するためのパリティ・チェックの方法を示します．これは単に**表3**の$SB(20)$を訂正するパリティ・チェックを示す'1'を，全体に左に1ビットシフトしたものとなっています．そして，いちばん左端であふれた'1'を右端に巡回させています．このようにシフトしたパリティ・チェックを用いれば$SB(19)$ビットを同様に訂正することができます．

同様にパリティ・チェック式のシフトを行うことで，ほかのビット位置のエラーを訂正することが可能となります．

差集合巡回符号による符号化とは，これまで説明したような複数のパリティ・チェックがシフトした形式(巡回した形式)でも成立するように符号化(11ビットを21ビット化)することとなります．

差集合巡回符号による符号化は，複数のパリティ・チェックを実現し，その複数のパリティ・チェックにある程度以上のエラーが検出されれば，特定のビットのエラーを検出できる方法です．

送信ビットの生成方法

これまで複雑な多項式を用いず，例題を用いて説明してきました．しかし，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で行います．

11ビットの情報ビットを$\{u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$とし，多項式$U(x)$で表します．

$$U(x) = u_0 + u_1 \cdot x^1 + u_2 \cdot x^2 + u_3 \cdot x^3 + u_4 \cdot x^4 + u_5 \cdot x^5 + u_6 \cdot x^6 + u_7 \cdot x^7 + u_8 \cdot x^8 + u_9 \cdot x^9 + u_{10} \cdot x^{10} \quad \cdots\cdots (1)$$

すると，21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．

$$SB(x) = u_0 \cdot x^{10} + u_1 \cdot x^{11} + u_2 \cdot x^{12} + u_3 \cdot x^{13} + u_4 \cdot x^{14} + u_5 \cdot x^{15} + u_6 \cdot x^{16} + u_7 \cdot x^{17} + u_8 \cdot x^{18} + u_9 \cdot x^{19} + u_{10} \cdot x^{20} + r_0 + r_1 \cdot x^1 + r_2 \cdot x^2 + r_3 \cdot x^3 + r_4 \cdot x^4 + r_5 \cdot x^5 + r_6 \cdot x^6 + r_7 \cdot x^7 + r_8 \cdot x^8 + r_9 \cdot x^9 \quad \cdots\cdots (2)$$

これを表で示すと，**表6**のようになります．すなわち，

$$R(x) = r_0 + r_1 \cdot x^1 + r_2 \cdot x^2 + r_3 \cdot x^3 + r_4 \cdot x^4 + r_5 \cdot x^5 + r_6 \cdot x^6 + r_7 \cdot x^7 + r_8 \cdot x^8 + r_9 \cdot x^9 \quad \cdots\cdots (3)$$

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，

$$X(x) = u_0 \cdot x^{10} + u_1 \cdot x^{11} + u_2 \cdot x^{12} + u_3 \cdot x^{13} + u_4 \cdot x^{14} + u_5 \cdot x^{15} + u_6 \cdot x^{16} + u_7 \cdot x^{17} + u_8 \cdot x^{18} + u_9 \cdot x^{19} + u_{10} \cdot x^{20} \quad \cdots\cdots (4)$$

を，

$$G(x) = 1 + x^2 + x^4 + x^6 + x^7 + x^{10} \quad \cdots\cdots (5)$$

で割り算を行った余りとなります．ただし，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．

(5)式の次数は10次なので，(4)式を(5)式で割った余りは9次式以下の次数となります．ガロア体での多項

〔表5〕
$SB(19)$を訂正するパリティ・チェック

ビットの位置番号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
送信ビット(SB)	0	0	1	1	0	0	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1
パリティ・チェックA_1									1			1	1					1		1	
パリティ・チェックA_2	1										1			1	1					1	
パリティ・チェックA_3				1		1										1			1	1	
パリティ・チェックA_4					1		1										1			1	1
パリティ・チェックA_5		1	1					1		1										1	

〔表6〕
情報ビット(11ビット)と送信ビット(21ビット)の例

	エラー訂正用(10ビット)										情報ビット(11ビット)										
情報ビット(11ビット)											u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
送信ビット(21ビット)	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}

式の余りを求める方法は実は単純であり，

$$x^{10} = 1 + x^2 + x^4 + x^6 + x^7 \quad \cdots\cdots\cdots (6)$$

を用いて，10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9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．ただしこのとき，係数どうしの加算や減算はXORで代用します．すなわち，

$$x^2 + x^2 = (1 \oplus 1) \cdot x^2 = 0 \cdot x^2 = 0$$
$$x^2 + x^2 + x^2 = (1 \oplus 1 \oplus 1) \cdot x^2 = 1 \cdot x^2 = x^2 \quad \cdots\cdots (7)$$

のように計算します．

送信機(transimitter)の回路構成と動作説明

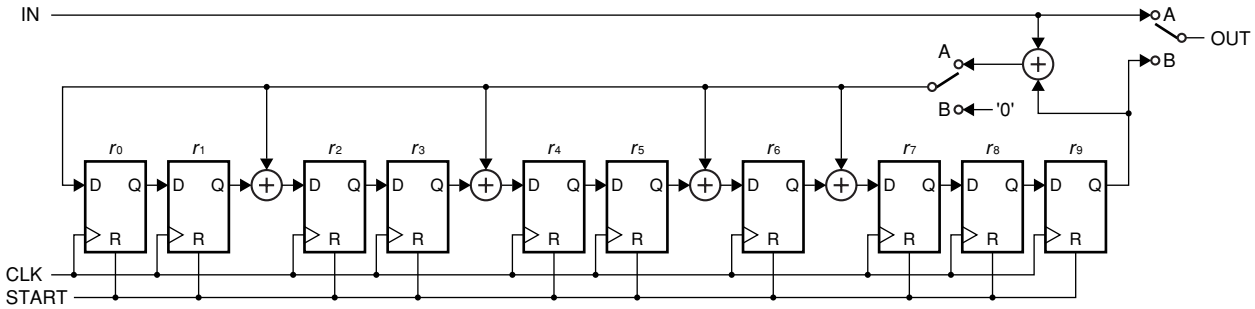
図2に送信機の回路図を示します．クロックに同期して，IN入力より情報ビットをu_{10}，u_9，…，u_0の順に入力します．このとき二つのスイッチはA側に接続しておきます．そうすると，OUT出力端子よりu_{10}，u_9，…，u_0がそのまま出力されると同時にr_9，r_8，…，r_0のフリップフロップと排他的論理輪XORで構成されるガロア体の割り算の余り計算回路で(3)式で示される係数が計算されます．(5)式で割り算を行いますので，(6)式を用いて式の次数を下げることになります．すなわち，INからの信号は10次の係数に対応し，10次の係数が1の場合，0次，2次，4次，6次，7次の係数に'1'をXOR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11個の値をすべて入力すると，r_9，r_8，…，r_0の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，r_9，r_8，…，r_0の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3では情報ビットの入力を示すために，START号を用い，START信号によりr_9，r_8，…，r_0になるフリップフロップの同期リセットも行っています．START信号は21サイクルごとに入力することにします．

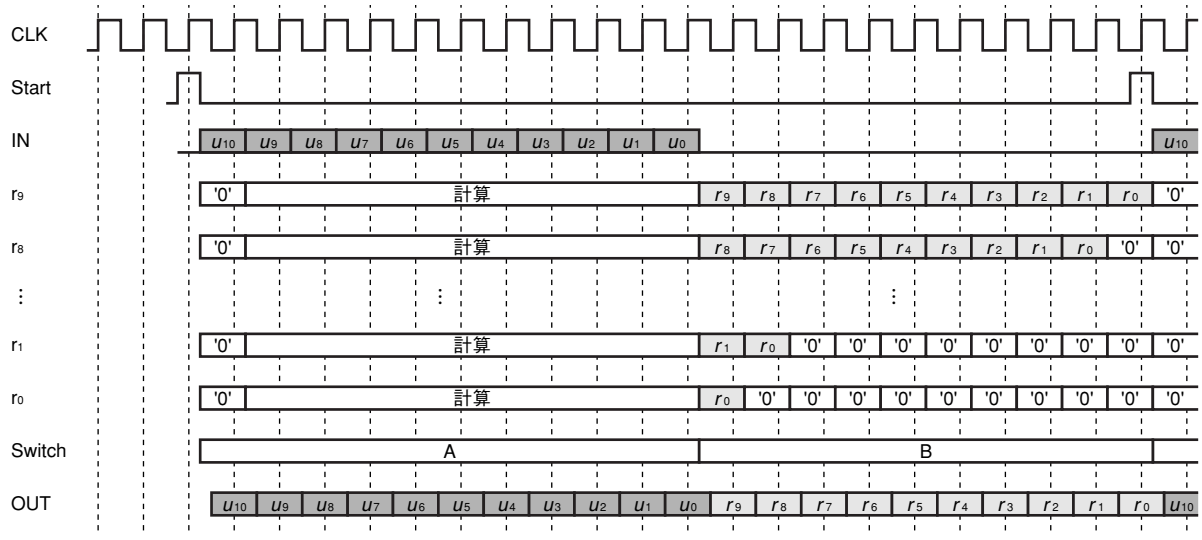
情報ビットが(0，0，0，0，0，0，0，0，0，0，1)の場合と，(1，1，1，1，1，1，1，1，1，1，1)の場合のレジスタの値の時間的変化を**表7**と**表8**に示します．

情報ビットが(0，0，0，0，0，0，0，0，0，0，1)の場合は，$u_0 = 1$，すなわち，10次の係数が1であり，(6)



▲〔図2〕送信機の回路図

▼〔図3〕送信機の動作タイミング図



式の処理を1回やっていることが理解できると思います．

情報ビットが(1，1，1，1，1，1，1，1，1，1，1)の場合は，$u_{10}=1$，すなわち，20次の係数が1であり，(6)式の処理をサイクル1で行うことで，その後の10サイクルでシフト動作を行うことで等価的に，

$$x^{20}=\left(1+x^2+x^4+x^6+x^7\right)\cdot x^{10} \quad \cdots\cdots\cdots (8)$$

を行っていることが理解できると思います．

受信機(Receiver)の設計(初心者用)

さて，いよいよ受信機の仕様を説明します．受信機では，**表4**で示したA_1からA_5のパリティ・チェックを行う必要があります．もちろん，**表4**に示したパリティ・チェックを行ってもかまいませんが，後ほどの性能改善のためにシンドロームを計算する方法を説明します．

先に説明したように，21ビットの送信信号は(2)式のようになります．

$$SB(x)=U(x)\cdot x^{10}+R(x)=X(x)+R(x) \quad \cdots\cdots (9)$$

$R(x)$は$X(x)$を(5)式の$G(x)$で割った余りであるので，$SB(x)$は$G(x)$で割り切れることになります．

今，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$G(x)$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$G(x)$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．

今，この$RB(x)$を$G(x)$で割った余りを$S(x)$シンドロームとすると，$S(x)$は以下のような9次の多項式となります．

$$S(x)=s_0+s_1\cdot x^1+s_2\cdot x^2+s_3\cdot x^3+s_4\cdot x^4+s_5\cdot x^5+s_6\cdot x^6+s_7\cdot x^7+s_8\cdot x^8+s_9\cdot x^9 \quad \cdots\cdots\cdots (10)$$

実は前に説明した5つのパリティ・チェック式は(10)式のシンドローム係数により簡単に計算することができます．この方法を**表9**に示します．

シンドロームは$RB(x)$を$G(x)$で割った余りなので，送信機で用いた回路(正式名：リニア・フィードバック・シフトレジスタ)と同じ回路を用いることで，$S(x)$を，すなわち係数s_0，s_1，…，s_9を求めることができます．

リニア・フィードバック・レジスタにより余りを計算

〔表7〕情報ビット=(0，0，0，0，0，0，0，0，0，0，1)を送信する場合の送信回路の動作

信号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
AB スイッチ	—	A	A	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B	B
IN	—	u_{10} 0	u_9 0	u_8 0	u_7 0	u_6 0	u_5 0	u_4 0	u_3 0	u_2 0	u_1 0	u_0 1	—	—	—	—	—	—	—	—	—	—
r_9	0	0	0	0	0	0	0	0	0	0	0	0										
r_8	0	0	0	0	0	0	0	0	0	0	0	0										
r_7	0	0	0	0	0	0	0	0	0	0	0	1										
r_6	0	0	0	0	0	0	0	0	0	0	0	1										
r_5	0	0	0	0	0	0	0	0	0	0	0	0										
r_4	0	0	0	0	0	0	0	0	0	0	0	1										
r_3	0	0	0	0	0	0	0	0	0	0	0	0										
r_2	0	0	0	0	0	0	0	0	0	0	0	1										
r_1	0	0	0	0	0	0	0	0	0	0	0	0										
r_0	0	0	0	0	0	0	0	0	0	0	0	1										
OUT	—	u_{10} 0	u_9 0	u_8 0	u_7 0	u_6 0	u_5 0	u_4 0	u_3 0	u_2 0	u_1 0	u_0 1	r_9 0	r_8 0	r_7 1	r_6 1	r_5 0	r_4 1	r_3 0	r_2 1	r_1 0	r_0 1

〔表8〕情報ビット=(1，1，1，1，1，1，1，1，1，1，1)を送信する場合の送信回路の動作

信号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
AB スイッチ	—	A	A	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B	B
IN	—	u_{10} 1	u_9 1	u_8 1	u_7 1	u_6 1	u_5 1	u_4 1	u_3 1	u_2 1	u_1 1	u_0 1	—	—	—	—	—	—	—	—	—	—
r_9	0	0	0	1	0	0	0	0	0	1	1	0										
r_8	0	0	1	0	0	0	0	0	1	1	0	0										
r_7	0	1	0	0	0	0	0	1	1	0	0	0										
r_6	0	1	1	0	1	1	0	0	1	0	0	1										
r_5	0	0	1	1	0	1	1	0	1	0	1	0										
r_4	0	1	1	0	1	1	0	1	0	1	0	0										
r_3	0	0	1	1	0	1	0	1	0	0	0	1										
r_2	0	1	1	0	1	0	1	0	0	0	1	1										
r_1	0	0	1	1	1	0	1	1	1	1	1	0										
r_0	0	1	1	1	0	1	1	1	1	1	0	0										
OUT	—	u_{10} 1	u_9 1	u_8 1	u_7 1	u_6 1	u_5 1	u_4 1	u_3 1	u_2 1	u_1 1	u_0 1	r_9 0	r_8 0	r_7 0	r_6 1	r_5 0	r_4 0	r_3 1	r_2 1	r_1 0	r_0 0

し，すなわちシンドロームを計算し，A_1からA_5のパリティ・チェックを行い，三つ以上のパリティ・チェックが1になった場合にエラーを発見する（ERROR_FOUND信号を1とする）方式を用いた受信機の回路例を，**図4**に示します．

21ビットの受信信号の入力が終了し，シンドローム値s_0，s_1，…，s_9が計算されると，最上位の受信信号ビットRB(20)用のパリティが計算され，多数決回路により，エラーが発生したかどうかが計算されます．そして，エラーが発生した場合，21ビット・シフト・レジスタの右側にXORによりエラーが訂正されます．その後に，EN信号を'0'とし，ANDゲートの出力を'0'に固定して，シンドローム・ビットを同じ回路で1ビットシフトすると，RB(19)用のパリティが計算され，またRB(19)が21ビット・シフト・レジスタより出力され，RB(19)にのエラー訂正をすることができます．この処理を繰り返すことで，21ビットの受信信号すべてにエラー訂正処理を行うことができます．

図中，CLK信号の引き回しは省略しています．

システム構成

今回設計するシステムの構成を**図5**に示します．システムは大きく分けて，TRANSMITTER（送信機）とRECEIVER（受信機），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ERRORCNT（エラー・カウント機）で構成されます．設計してもらうのは，RECEIVERとERRORCNTです．実際に回路合成を行い，回路規模や速度を求めるのはRECEIVERだけです．ERRORCNT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．

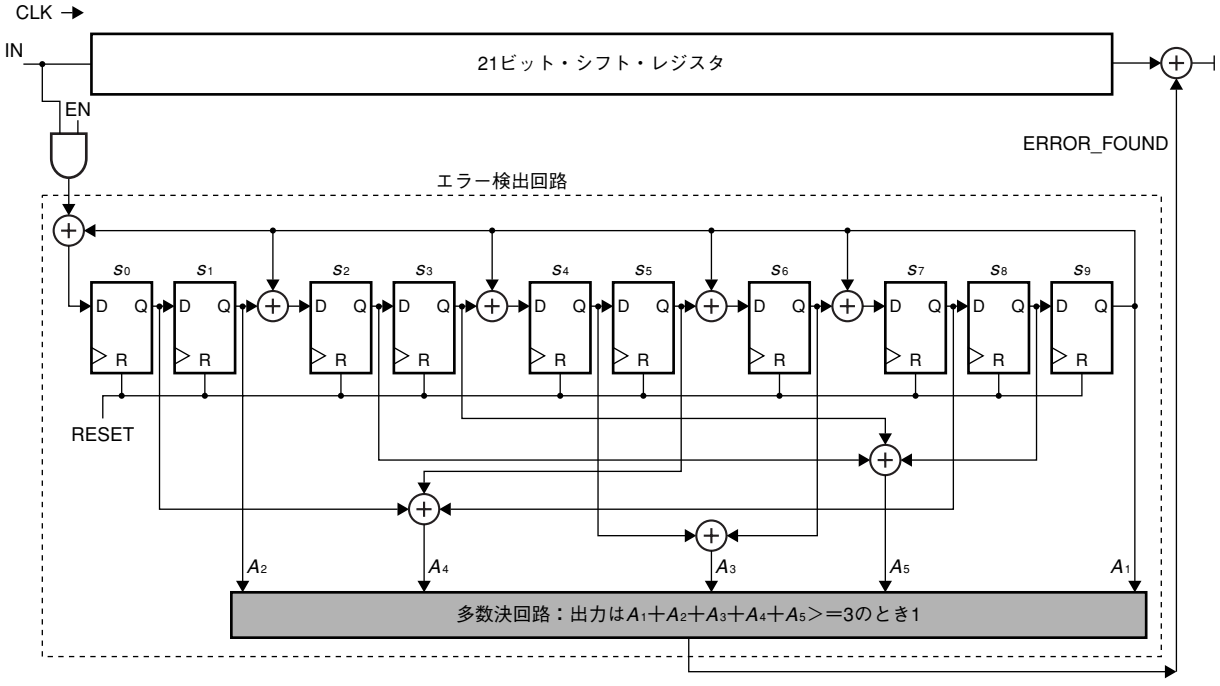
●TRANSMITTER：送信機の説明

TRANSMITTERは送信機であり，11ビットの情報ビット列$U(x)$を生成し，10ビットのエラー訂正用ビット列$R(x)$を付加して，21ビットの送信データ$SB(x)$を生成します．

送信機は21ビットの送信データの前に同じく21ビットの決まったシーケンス（同期信号と呼ぶ）を付加しま

〔表9〕シンドロームによる5種類のパリティ・チェックの導出

パリティ・チェックA_1	$A_1=s_9=SB(9)$ xor $SB(12)$ xor $SB(13)$ xor $SB(18)$ xor $SB(20)$
パリティ・チェックA_2	$A_2=s_1=SB(1)$ xor $SB(11)$ xor $SB(14)$ xor $SB(15)$ xor $SB(20)$
パリティ・チェックA_3	$A_3=s_4$ xor $s_6=SB(4)$ xor $SB(6)$ xor $SB(16)$ xor $SB(19)$ xor $SB(20)$
パリティ・チェックA_4	$A_4=s_0$ xor s_5 xor $s_7=SB(0)$ xor $SB(5)$ xor $SB(7)$ xor $SB(17)$ xor $SB(20)$
パリティ・チェックA_5	$A_5=s_2$ xor s_3 xor $s_8=SB(2)$ xor $SB(3)$ xor $SB(8)$ xor $SB(10)$ xor $SB(20)$



〔図4〕初心者用受信回路の例

す．この連続した値はMSB→LSBの順に示して‘00110101111010111000000’とします．送信機ではMSBから順に出力されます．

さらに送信機は同期信号が付加され42ビット長となった信号に適当なエラーを発生させ，その出力をSBWE端子に出力します．SBWEには連続的に1か0の信号がクロックCLKに同期して出力されます．説明したように，42サイクルで42ビットを送信し，この42サイクルごとに一つの11ビットの情報ビット列を送信することになります．この42ビットの送信は連続的に発生し，つねに42サイクルごとに11ビットの情報ビット列が送信されることになります．

START信号は42サイクル中1サイクル期間のみ，1を出力し，21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どう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．

●RECEIVER：受信機の説明

RECEIVERは受信機であり，クロックに同期して，TRANSMITTERからの送信信号SBWEを受け取ります．42サイクルごとにそこに含まれる21ビットの送信データを用いて，エラー訂正された11ビットの情報ビット列をRBEC端子に出力します．

21ビットの送信データの先頭位置はSTART信号を用いて知ることができます．しかしながら，上級者用の課題ではRECEIVERはSTART信号を使用せず，RBWEに含まれる21ビットの決まった同期信号を用いてSTART信号を内部で発生させます．

ただし，21ビットの同期信号にもエラーが発生しているので，注意が必要です．

●ERRORCNT：エラー・カウント機

ERRORCNTは受信機の評価用のブロックであり，回路合成などは行いません．TRANSMITTER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，ERRORCNTはRECEIVER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．ERRORCNT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8ビットのレジスタに累積のエラーの発生回数を記憶し，外部へERRCNT端子を通して出力します．8ビットですので，255までのエラーをカウントできることにします．

●動作タイミング

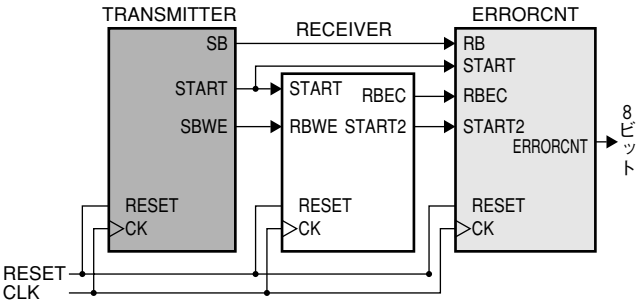
図6に動作タイミング図を示します．RESET信号の解除の後，TRANSMITTERはSBWE（Sent Bits With Error）信号，SB（Sent Bits without error）信号，START信号を出力します．42サイクルが一つの単位となっており，最初の21サイクルは決まった値が送信され，その後の11サイクルで情報ビットが，その後の10サイクルでエラー訂正用ビットが送信されます．SBWEはエラー付の信号であり，ある確率でエラーが発生します．詳しくは与えられたTRANSMITTERのVHDL記述（**リスト1**．稿末）を見てください．もちろん，最初の同期用の21サイクルの信号にもエラーが発生しますので，レベル3の上級者課題を行う方は注意が必要です．

この同期用の21ビットの値はMSB→LSBの順に示して‘00110101111010111000000’で，MSBを先頭に送信されます．

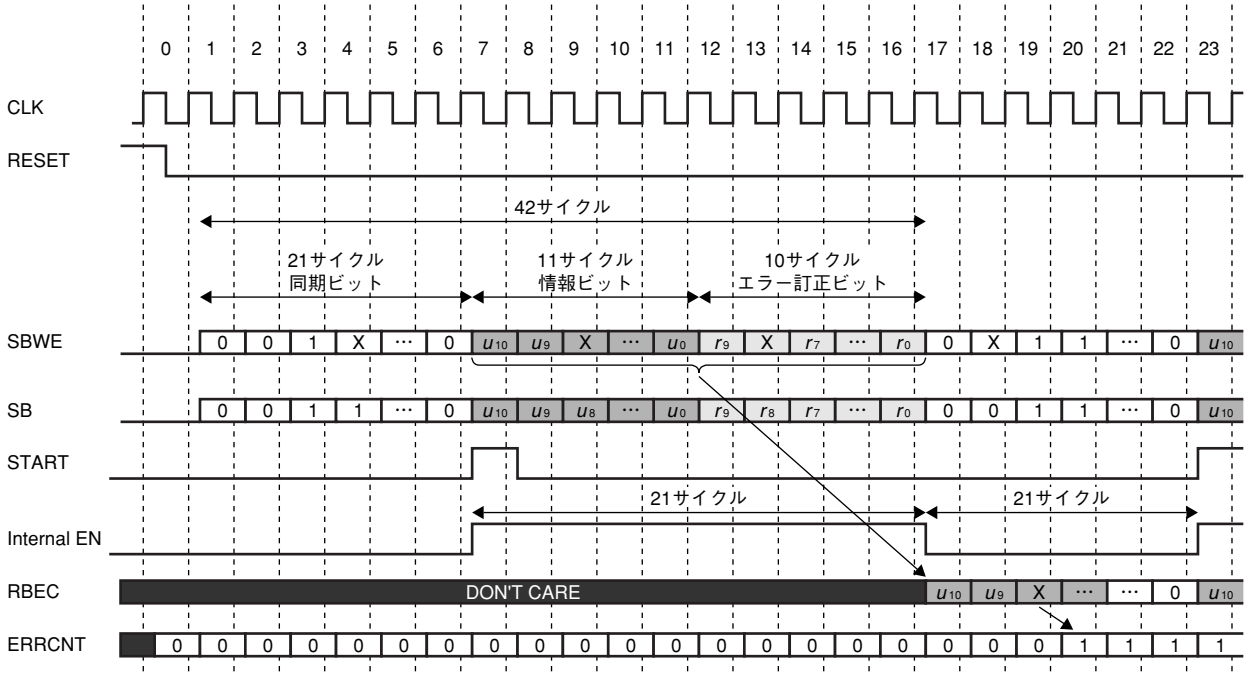
RECEIVERはSBWE信号を受信し，エラー訂正を行い，RBEC（Received Bits Error Recovered）にエラー訂正をした情報ビットを出力します．エラー訂正用ビットもエラー訂正処理を行うことが可能ですが，それを出力するかしないかは設計者の自由とします．

ERRCNTは11ビットの情報ビットに対するエラーの累積値を示すもので，一応8ビット（＝255）までとしますが，自由に変更してもかまいません．

内部EN信号Internal EN信号は，STARTの入力サイクルから21サイクルの間‘1’となり，その後の21サイクルは‘0’となる信号です．この信号は**図4**と**図7**のEN

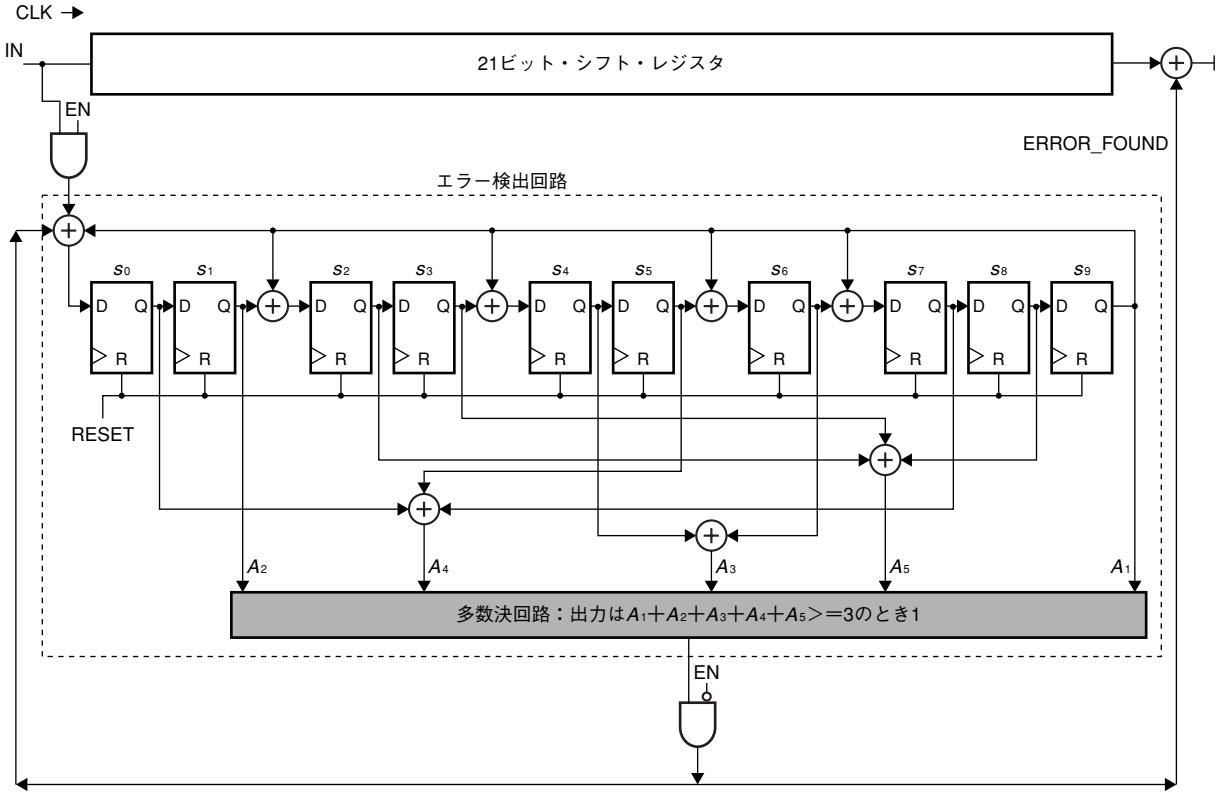


〔**図5**〕システム構成図



▲〔図6〕システム動作タイミング図

▼〔図7〕改良されたERROR DETECTION LOGIC



〔表10〕RECEIVERの初心者用ピン配置

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	エラー付きの受信信号
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11〕RECEIVERの上級者用ピン配置

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
RBWE	IN	1	エラー付きの受信信号
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

信号と対応しており，EN= ‘1’ の区間はシンドローム・レジスタへの値の取り込み，EN= ‘0’ の区間はエラー検知信号ERROR_FOUND 信号の発生区間を意味します．

課題

●LEVEL1 ：初心者用課題

初心者用RECEIVER は，**表10** に示すピンを持つこととします．

送信機TRANSMITTER，エラー・カウント機ERRORCNT，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**リスト1** ～**リスト4**（稿末）に示します．必要に応じて修正して使用してください．

●LEVEL2:中級者用課題

中級者用RECEIVER は初心者用と同じピン配置をもつこととしますが，以下に示すエラー訂正回路に以下に示す改良を行います．

図7 は**図4** の ERROR DETECTION LOGIC の Majority logic からのエラー検知信号ERROR_FOUND をシンドローム・レジスタにフィードバックしたものです．すなわち，Majority Logic にてエラーの発生が検知されたら実際にエラー訂正を行うので，そのエラー訂正に対応してシンドロームを変更します．このようにすると，うまくすべてのエラーが訂正されたならば，すべてのシンドローム・ビット0，s_1，…，s_9 はすべて0 になります．

このような改良をすることで，エラーの修正能力が向上します．

●LEVEL3 ：上級者用課題

上級者用はSTART 信号入力がない，**表11** のピン配置でかつ，中級者用の改良されたエラー訂正方式とします．

START 信号がないので，最初の21 サイクルの同期信号によりSTART 信号を発生させる必要があります．ただし，同期信号にもエラーが発生するのでややくふうが必要です．

速度の測定単位

筆者の大学では米国Synopsys 社のDesign Compiler を使用しますが，琉球大学以外からの参加の場合，同じ論理合成用ライブラリを使用することが困難であると思いますので，**リスト5** のような多段XOR 回路を例に従って合成し最適化していただき，その1段あたりの遅延時間を単位時間として速度を評価します．

〔リスト5〕多段XOR 回路のVHDL ソース例（50 入力のXOR 回路）

```
library  IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;

entity PARITY is
    port ( A : in  unsigned(49 downto 0);
           Y : out std_logic );
end PARITY;

architecture RTL of PARITY is
begin

    process(A)
    variable TMP : std_logic;
    begin
        TMP := '0';
        for i in 0 to 49 loop
            TMP := TMP xor A(i);
        end loop;

        Y <= TMP;
    end process;
end RTL;
```

この例では**図8** のように6段のXOR 段が合成され，クリティカル・パス遅延はreport_timing コマンド（**図9**）により7.17 であったので，7.17/6 = 1.195 を単位（UNIT）とします．

例えば，ある遅延が20 ならば，20/1.195 = 17.74 UNIT 遅延となります．ちなみに面積はreport_area コマンド（**図10**）のtotal cell area にあります．

提出レポートについて

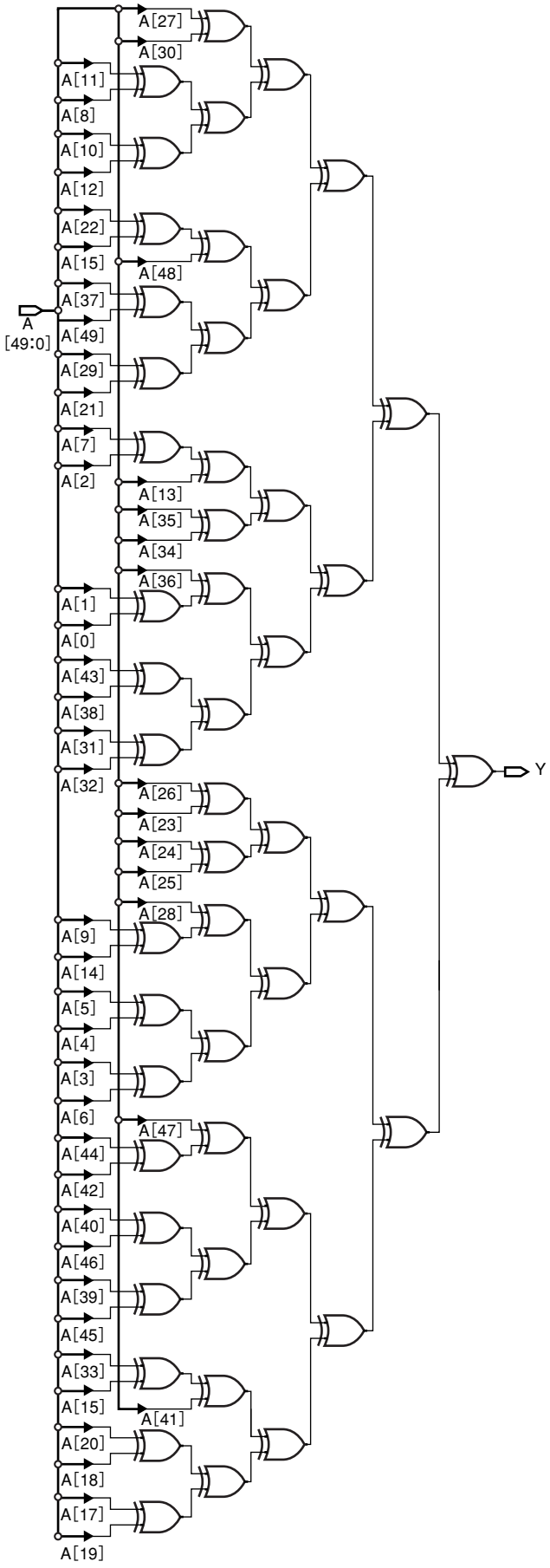
レポートには以下の内容を含めてください．また，ページ数を少なめにコンパクトにまとめてください．

＜表紙＞

1）代表者の氏名，チーム名，社会人/大学院修士/大学学部生/高専生の区別
2）共同設計者全員の名前（最高3 名まで），会社名/学校名（学籍番号，学年），住所，電話，E-mail などの連絡先
3）取り組んだ課題（初級/中級/上級）

＜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



〔図8〕リスト5の論理合成後の回路図

〔図9〕リスト5のタイミング・レポート

```
Generating schematic for design: PARITY
The schematic for design 'PARITY' has 1 page(s).

1
design_analyzer> report_timing
Information: Updating design information... (UID-85)

            :
          中略
            :

Operating Conditions:
Wire Load Model Mode: top

  Startpoint: A[19] (input port)
  Endpoint: Y (output port)
  Path Group: (none)
  Path Type: max

  Des/Clust/Port     Wire Load Model   Library
  ------------------------------------------------
  PARITY             05x05             class

  Point                                Incr       Path
-------------------------------------------------------------
  input external delay                 0.00       0.00 f
  A[19] (in)                           0.00       0.00 f
  U27/Z (EO)                           1.22       1.22 f
  U28/Z (EO)                           1.22       2.43 f
  U36/Z (EO)                           1.22       3.65 f
  U43/Z (EO)                           1.22       4.86 f
  U55/Z (EO)                           1.22       6.08 f
  U13/Z (EO)                           1.09       7.17 f
  Y (out)                              0.00       7.17 f
  data arrival time                               7.17
-------------------------------------------------------------
(Path is unconstrained)
```

〔図10〕面積レポート

```
design_analyzer> report_area
Information: Updating design information... (UID-85)

            :
          中略
            :

 Library(s) Used:

    class (File:/usr/open/synopsys/1999.05/libraries/syn/
                                                  class.db)

Number of ports:               51
Number of nets:                99
Number of cells:               49
Number of references:           1

Combinational area:   147.000000
Noncombinational area:  0.000000
Net Interconnect area: undefined  (Wire load has zero net
                                                    area)

Total cell area:      147.000000
Total area:            undefined
```

プライン動作などの説明）
3）くふうした点，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・パスの速度，論理合成後の回路規模
5）VHDLもしくはVerilog HDLのコード
6）正常動作しているVHDL/Verilog HDLシミュレーション波形
7）そのほか自由意見など

レポートはPDFファイルにて下記にE-mailで提出してください（ほかの形式での提出希望あれば，相談してください）．

社会人か学生かで評価の方法が異なります．それにともない，レポートの送付先と締め切り日が異なります．

社会人：contest.dwm@cqpub.co.jp
締め切りは2002年1月31日（木）必着．

学生　：wada@ie.u-ryukyu.ac.jp
締め切りは2002年2月15日（金）必着．

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，おもしろさを十分考慮して審査します（ちゃんとアピールしてね！）．

社会人は，Design Wave Magazine編集部が審査を行います．学生は，琉球大学で第1次審査を行い，最終審査に残ったチームは，2002年3月8日に琉球大学で開催予定の「シリコン・シーベルト・デザイン・コンテスト最終発表会」に招待され，その場で最終審査を行います．

仕様書に従ってまじめに作るのもよいが，おもしろい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！

わだ・ともひさ
琉球大学工学部 情報工学科

〔リスト1〕送信機（transmitter.vhd）

```
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;

entity TRANSMITTER is
    port (SBWE     : out std_logic;
          SB       : out std_logic;
          START    : out std_logic;
          RESET    : in  std_logic;
          CLK      : in  std_logic );
end entity TRANSMITTER;

architecture RTL of TRANSMITTER is
    -- phase counts from 0 to 41
    -- phase  0 to 20 : sync bits are transmitted
    -- phase 21 to 31 : information bits transmitted
    -- phase 32 to 41 : parity bits transmitted
    signal  phase    : unsigned (5 downto 0);
    -- 11bits information bits
    signal  infbit   : unsigned (10 downto 0);
    -- internal start signal
    signal  intstart : std_logic;
    -- internal sb signal
    signal  intsb,intsb2    : std_logic;
    -- 10 parity bits
    signal  parity   : unsigned (9 downto 0);
    -- error signal
    signal  errsig   : std_logic;
    -- error counter
    signal  errcnt   : unsigned (9 downto 0);
begin
-------------------------------------------------
-- 42 phase (0 to 41 counter) generation unit
-------------------------------------------------
    PHASE_CNT: process(CLK,RESET)
    begin
        if (RESET='1') then
            phase <= "000000";
        elsif rising_edge(CLK) then
            if (phase="101001") then -- if phase=41
                phase <= "000000";
            else
                phase <= phase + 1;
            end if;
        end if;
    end process PHASE_CNT;
-------------------------------------------------
-- 11 bits information generation unit
-- count down from 1111111111 by 1
-------------------------------------------------
    INF_GEN: process(CLK, RESET)
    begin
        if (RESET='1') then
            infbit <= "00000000010";
        elsif rising_edge(CLK) then
            if (phase="010100") then -- if phase=20
                infbit <= infbit - 1;
            end if;
        end if;
    end process INF_GEN;
-- internal start (intstart)  generation
-------------------------------------------------
    START_GEN: process(CLK, RESET)
    begin
        if (RESET='1') then
            intstart <= '0';
        elsif rising_edge(CLK) then
            if (phase="010101") then -- if phase=21
               intstart <= '1';
```

〔リスト1〕送信機(transmitter.vhd)(つづき)

```
            else intstart <= '0';
            end if;
        end if;
    end process START_GEN;
-------------------------------------------------
-- internal sb signal (intsb) generation
-------------------------------------------------
    SB_GEN: process(CLK, RESET)
    begin
        if (RESET='1') then
            intsb <= '0';
        elsif rising_edge(CLK) then
            case phase is
              when "000000"=> intsb <= '0'; -- 0
              when "000001"=> intsb <= '0'; -- 1
              when "000010"=> intsb <= '1'; -- 2
              when "000011"=> intsb <= '1'; -- 3
              when "000100"=> intsb <= '0'; -- 4
              when "000101"=> intsb <= '1'; -- 5
              when "000110"=> intsb <= '0'; -- 6
              when "000111"=> intsb <= '1'; -- 7
              when "001000"=> intsb <= '1'; -- 8
              when "001001"=> intsb <= '1'; -- 9
              when "001010"=> intsb <= '1'; --10
              when "001011"=> intsb <= '0'; --11
              when "001100"=> intsb <= '1'; --12
              when "001101"=> intsb <= '1'; --13
              when "001110"=> intsb <= '1'; --14
              when "001111"=> intsb <= '0'; --15
              when "010000"=> intsb <= '0'; --16
              when "010001"=> intsb <= '0'; --17
              when "010010"=> intsb <= '0'; --18
              when "010011"=> intsb <= '0'; --19
              when "010100"=> intsb <= '0'; --20
              when "010101"=> intsb <= infbit(10); --21
              when "010110"=> intsb <= infbit(9);  --22
              when "010111"=> intsb <= infbit(8);  --23
              when "011000"=> intsb <= infbit(7);  --24
              when "011001"=> intsb <= infbit(6);  --25
              when "011010"=> intsb <= infbit(5);  --26
              when "011011"=> intsb <= infbit(4);  --27
              when "011100"=> intsb <= infbit(3);  --28
              when "011101"=> intsb <= infbit(2);  --29
              when "011110"=> intsb <= infbit(1);  --30
              when "011111"=> intsb <= infbit(0);  --31
              when  others => intsb <= 'X';  --others
            end case;
        end if;
    end process SB_GEN;
-------------------------------------------------
-- parity calculation
-------------------------------------------------
    PARITY_CAL: process(CLK, RESET)
    begin
        if (RESET='1') then
            parity <= "0000000000";
        elsif rising_edge(CLK) then
            if (phase="010101") then
                 parity <= "0000000000";
            elsif (phase>="010110" and phase<="100000")
then
                parity(9) <= parity(8);
                parity(8) <= parity(7);
                parity(7) <= parity(6) xor intsb xor
parity(9);
                parity(6) <= parity(5) xor intsb xor
parity(9);
                parity(5) <= parity(4);
                parity(4) <= parity(3) xor intsb xor
parity(9);
                parity(3) <= parity(2);
                parity(2) <= parity(1) xor intsb xor
parity(9);
                parity(1) <= parity(0);
                parity(0) <= intsb xor parity(9);
            else
                parity <= parity(8 downto 0) & parity(9);
            end if;
        end if;
    end process PARITY_CAL;
-------------------------------------------------
-- error interval counter
-------------------------------------------------
    ERR_CNT: process (CLK, RESET) begin
        if (RESET='1') then errcnt <= "0000000000";
        elsif rising_edge(CLK) then
            if (errcnt = "0000001001") then -- max=9 (0-9)
                  errcnt <= "0000000000";
            else errcnt <= errcnt +1;
            end if;
        end if;
    end process ERR_CNT;
-------------------------------------------------
-- error signal generation
-------------------------------------------------
    ERROR_GEN: process (CLK, RESET)
    begin
        if (RESET='1') then
           errsig <= '0';
        elsif rising_edge(CLK) then
           if (errcnt="0000000000") then
               errsig <= '1';  -- error happens every 10
cycle
           else errsig <= '0';
           end if;
        end if;
    end process ERROR_GEN;
-------------------------------------------------
-- output signals
-------------------------------------------------
    SBOUT: process (CLK, RESET)
    begin
        if (RESET='1') then
            intsb2 <= '0';
        elsif rising_edge(CLK) then
            if(phase >="100001" or phase="000000") then
                intsb2 <= parity(9);
            else
                 intsb2 <= intsb;
            end if;
        end if;
    end process SBOUT;

    STARTOUT: process (CLK, RESET)
    begin
        if (RESET='1') then
            START <= '0';
        elsif rising_edge(CLK) then
            START <= intstart;
        end if;
    end process STARTOUT;

    SB   <= intsb2;
    SBWE <= intsb2 xor errsig;

end architecture RTL;
```

〔リスト2〕エラー・カウント機(errorcnt.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;

entity ERRORCNT is
    port (START    : in  std_logic;
          RB       : in  std_logic;
          START2   : in  std_logic;
          RBEC     : in  std_logic;
          ERRCNT   : out unsigned(7 downto 0);
          RESET    : in  std_logic;
          CLK      : in  std_logic );
end entity ERRORCNT;

architecture RTL of ERRORCNT is

-- phase counter
signal phase   : unsigned (4 downto 0);
-- enable signal
signal en      : std_logic;
-- 21 bit signal shift register
signal data    : unsigned (20 downto 0);
-- RBEC input register
signal rbecin  : std_logic;
-- RBEC phase counter
signal phase2  : unsigned (4 downto 0);
-- enable2 signal
signal en2      : std_logic;
-- 8bit error counter
signal counter : unsigned (7 downto 0);

begin
-----------------------------------------------
-- phase generation
-- count 20 downto 0
-----------------------------------------------
PHASE_GEN: process (CLK, RESET) begin
    if (RESET='1') then phase <= "00000";
    elsif rising_edge(CLK) then
        if (START='1') then phase <= "10100"; -- set 20
        elsif (phase="00000") then phase <= "00000";
        else  phase <= phase -1;
        end if;
    end if;
end process PHASE_GEN;
-----------------------------------------------
-- ENABLE generation, COMBINATIONAL LOGIC
-- EN is high while START=1 or phase is not 0
-----------------------------------------------
EN_GEN: process (START, phase) begin
    if(START='1' or phase /= "00000") then en <='1';
    else en <='0';
    end if;
end process EN_GEN;
-----------------------------------------------
-- 21 bit shift register
-----------------------------------------------
SHIFT_REG: process(CLK, RESET) begin
    if (RESET='1') then data <= (others=>'0');
    elsif rising_edge(CLK) then
        if (en='1') then
              data <= data(19 downto 0) & RB;
        end if;
    end if;
end process SHIFT_REG;
-----------------------------------------------
-- PHASE2 generation
-----------------------------------------------
PHASE2_GEN: process(CLK, RESET) begin
    if (RESET='1') then phase2 <= "00000";
    elsif rising_edge(CLK) then
        if (START2='1') then phase2 <= "10100"; -- set 20
        elsif (phase2="00000") then phase2 <= "00000";
        else phase2 <= phase2 -1;
        end if;
    end if;
end process PHASE2_GEN;
-----------------------------------------------
-- ENABLE2 generation
-- EN2 is high while START2=1 or phase is not 0
-----------------------------------------------
EN2_GEN: process (CLK, RESET) begin
    if (RESET='1') then en2 <='0';
    elsif rising_edge(CLK) then
        if(START2='1' or phase2 /= "00000") then en2 <='1';
        else en2 <='0';
        end if;
    end if;
end process EN2_GEN;
-----------------------------------------------
-- RBECIN input register
-----------------------------------------------
INPUT_REG: process(CLK, RESET) begin
    if (RESET='1') then rbecin <= '0';
    elsif rising_edge(CLK) then
        rbecin <= RBEC;
    end if;
end process INPUT_REG;
-----------------------------------------------
-- error counter
-----------------------------------------------
ERR_CNT: process(CLK,RESET) begin
    if (RESET='1') then counter <= "00000000";
    elsif rising_edge(CLK) then
        if (en2='1' and data(TO_INTEGER(phase2)) /= rbecin)
then
            counter <= counter+1;
        end if;
    end if;
end process ERR_CNT;

ERRCNT <= counter;

end architecture RTL;
```

〔リスト3〕全体シミュレーション(test_dcc21.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;

entity TEST_DCC21 is
end entity TEST_DCC21;

architecture TESTBENCH of TEST_DCC21 is
    component TRANSMITTER
        port ( SBWE  : out  std_logic;
               SB    : out  std_logic;
               START : out  std_logic;
               RESET : in   std_logic;
               CLK   : in   std_logic );
    end component;
    component RECEIVER
        port ( START : in   std_logic;
               RBWE  : in   std_logic;
               START2: out  std_logic;
               RBEC  : out  std_logic;
               RESET : in   std_logic;
               CLK   : in   std_logic );
    end component;
    component ERRORCNT
        port ( START : in   std_logic;
               RB    : in   std_logic;
               START2: in   std_logic;
               RBEC  : in   std_logic;
               ERRCNT: out  unsigned(7 downto 0);
               RESET : in   std_logic;
               CLK   : in   std_logic );
    end component;
    signal     SBWE   :  std_logic;
    signal     SB     :  std_logic;
    signal     START  :  std_logic;
    signal     RESET  :  std_logic := '1';
    signal     CLK    :  std_logic := '0';
    signal     RBEC   :  std_logic;
    signal     START2 :  std_logic;
    signal     ERRCNT :  unsigned(7 downto 0);
begin
    -- clock generation
    CLK <= not CLK after 10 ns;

    TX: TRANSMITTER port map
    (SBWE, SB, START, RESET, CLK);

    RX: RECEIVER port map
    (START, SBWE, START2, RBEC, RESET, CLK);

    EC: ERRORCNT port map
    (START,SB,START2,RBEC,ERRCNT,RESET,CLK);

    process
    begin
        RESET_LOOP: for N in 0 to 3 loop
        wait until falling_edge(CLK);
        end loop RESET_LOOP;
            RESET <= '0';

        CAL_LOOP: for N in 0 to 500 loop
        wait until falling_edge(CLK);
        end loop CAL_LOOP;

        wait;
    end process;
end architecture TESTBENCH;

configuration CFG_DCC21 of TEST_DCC21 is
    for TESTBENCH
    end for;
end configuration CFG_DCC21;
```

〔リスト4〕設計すべき受信機のテンプレート(receiver.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.NUMERIC_STD.all;

entity RECEIVER is
    port (START    : in  std_logic;
          RBWE     : in  std_logic;
          START2   : out std_logic;
          RBEC     : out std_logic;
          RESET    : in  std_logic;
          CLK      : in  std_logic );
end entity RECEIVER;

architecture RTL of RECEIVER is

-- HERE, WRITE YOUR ORIGINAL HDL CODE!

-- GOOD LUCK!


end architecture RTL;
```