卒 業 研 究 報 告

題　　目

差集合巡回符号エラー訂正回路の設計

指 導 教 員

矢野　政顕　教授

報 告 者

石川　純平

平成 14 年　 2 月　 7 日

高知工科大学 電子･光システム工学科

目次

第１章　はじめに

現在、パソコンによる「ディジタル通信」や、近年開始された「BS ディジタル放送」が盛んになるにつれ、「ディジタル」や「文字多重」などの言葉を耳にする機会が増えてきた。その中でも、「文字多重放送」は、身近なところでテレビやＦＭラジオなどを介して我々の日々の生活に定着してきている。技術の発展に伴い、ディジタル放送等が盛んになるにつれ、送受信される情報データの規模が大きくなってきているのも事実である。しかし、送受信される情報データが大きくなればなるほど、送受信の過程で、それらの情報データの中にエラー（誤り）が生じる可能性が高くなるのも否めない事実である。ゆえに、受信側にはそのエラーを発見・訂正し、元の情報データに戻す機能が求められるようになってきた。そこで、卒業研究では、テレビ等の文字多重放送の、エラー訂正方法である差集合巡回符号について学び、実際に VHDL 等を用いて、エラー訂正ための回路を設計する技術を習得することを目的とした。

本論文の第 2 章では、卒業研究の背景となっている文字多重放送について説明する。第 3 章では、文字多重放送で用いられる「パリティ・チェック」と「差集合巡回符号」について、その構成とエラー訂正のメカニズムについて述べる。第 4 章では、文字多重放送に用いられる「差集合巡回符号エラー訂正回路」の VHDL 等による設計等について述べる。第 5 章では、卒業研究で得られた成果を要約し、今後に残された課題について述べる。

なお、本研究の課題は「DesignWave」誌の「設計コンテスト 2002」の中級レベルに相当し、本研究の成果をコンテストに応募中である。

第２章　文字多重放送

本章では、卒業研究テーマである「差集合巡回符号　エラー訂正回路設計」を研究していく上で、「エラー訂正」が使われている分野の一つであるテレビや「文字多重放送」について、文献やインターネットにて調べた基本的な知識をまとめて紹介する。具体的には、文字多重放送がどのようなものか、どんな状況で使われているのか、どんな方法が使われているのか順に説明する。さらに、実際に「エラー訂正」がどのような場面で使われているのか、なぜ必要なのか、またテレビの「文字多重放送」と同様な技術を用い、近年主流になりつつあるＦＭラジオ放送の「文字多重放送」についても述べる。[1],[2],[10],[11],[12]

2.1　テレビの「文字多重放送」

本節では、テレビの「文字多重放送」について、その概要を説明し、さらに具体的な例を挙げて紹介する。

2.1.1　テレビの「文字多重放送」とは？

そもそも、現在、我々の生活に密着している「文字多重放送」は，昭和６０年（1985 年）の１１月２９日にＮＨＫにて開始された。解りやすい例としては、最近 JR の電車等の中などで見られる文字放送で、地上波のすき間「空き走査線 VBI」（「2.1.3」節参照）に文字や図形のデータを送るものがあげられる。また、BS では某ゲームハード用のゲームソフトを送ったり、CS では Sky PerfecTV が Sky PerfecPC サービスとしてニュースや気象情報などを送っている。

では、放送システムの中にデジタル技術が登場したのはいつのことか。実はこれがつい最近のことではない。テレビの初期に、東京発からローカルの番組に切り替える時に画面の隅に三角のマークいわゆる「三角パンチ」を一瞬出し、これを合図に切り替えていた。このような原始的な方法を、なんとか自動化するためデジタル技術が初めてテレビ信号の中に登場した。

現在は文字放送やデータ放送が利用している「空き走査線 VBI」に、切り替えなどのためデジタル信号“I”“Q”を送るようになった（1968）。しかし、これは

放送局間で使われ、一般の家庭では利用できない舞台裏の信号であった。実際に、家庭までデジタル信号が届き利用するようになったのは、前述のように、NHK で開始された、テレビの文字多重放送の実用化が最初である。

しかし、これも未だ完全に表舞台への登場とはいえない。はじめて放送の映像・音声がデジタル化されたのは衛星放送の実用化の時（1989 本放送）であるが、音声のみ（A モード、B モードのデジタル音声）のデジタル化であった。映像はまだアナログであるので、アナログ／デジタルの混合（ハイブリッド）方式である。日本の文字放送は、字幕放送専門の欧米とは異なり、文字、図形、音声まで放送できる高性能なもので、マルチメディア時代の主柱の一つとして大きく期待されている。

現在の文字放送はデジタル技術を利用している為、コンピュータとドッキングし易いわけである。デジタル信号を送り、受信機側のデコーダー（文字受信機）が信号を解読して、テレビ画面にその内容を映し出すものである。エレクトロニクスの最先端テクノロジーのデジタル技術を利用し、テレビ画面の走査線５２５本の内４本を使ってＮＨＫの総合テレビに多重している文字番組は、１００番組以上で、テレビの放送時間中、朝から深夜までとぎれなく放送されている。ちなみに、番組の種類は、身近なところで言えば、ニュース、天気予報、株価情報、競馬情報等、実に様々ある。

つまり、ユーザー自身が見たいときに何時でも見ることができる放送、いわば自分が主役の番組、それが文字放送である。

2.1.2　テレビの「文字多重放送」の概要

テレビの「文字多重放送」は、2.11 節で簡単に述べた通り地上波のすき間「空き走査線 VBI」に文字や図形のデータを送るものだが、それを説明する前に、実際にどのように情報の送受信をするのか、以下の図 2.1 を用いてシステムの概要を具体的に説明する。

送信側の、ＴＶ主整室では、一般のテレビ放送と同じ方法で制作された番組の、音声信号は何も手を加えずに、そのままＴＶ送信機に送るのだが、映像信号は多重化装置によって文字信号を挿入してからＴＶ送信機に送る。そして、両信号は合成され送信信号としてアンテナから電波となって送信される。

一方、受信側の映像復調器では、アンテナから受信した受信信号の中の映像信号から文字信号を抜き取り、デコーダ（解読器）で文字切り替えや信号処理

等の過程を経てデコード（解読）し、映像信号と一緒にブラウン官に表示するとともに、音声信号をスピーカより出力する。

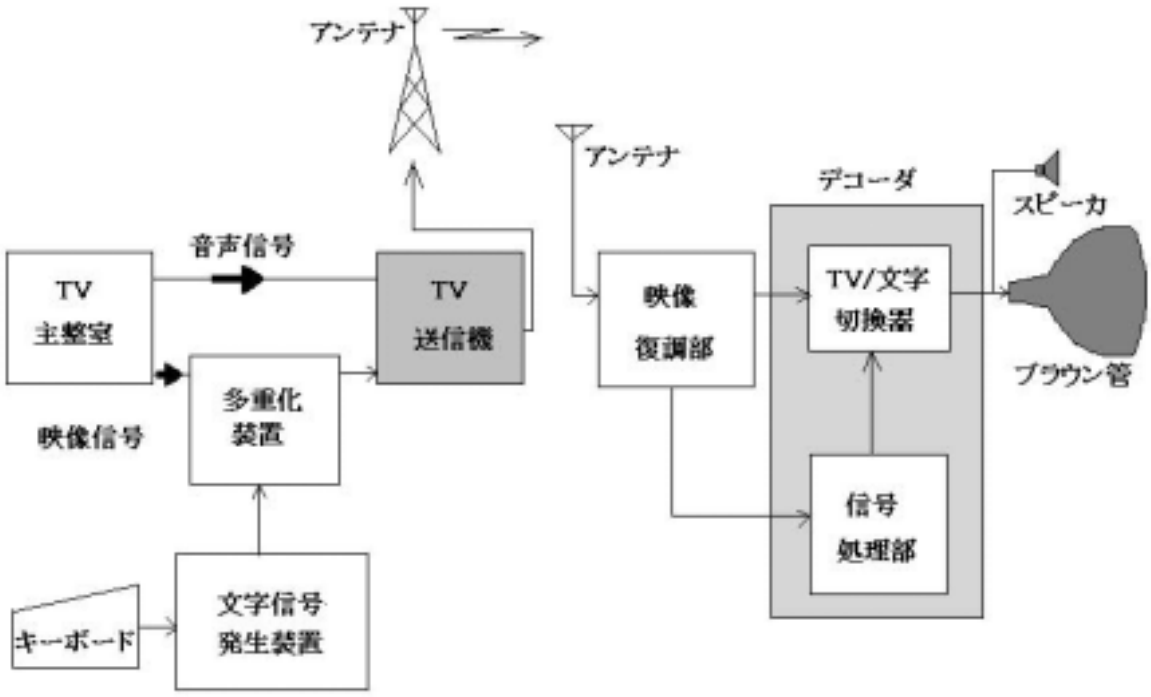

図 2.1　文字多重放送システムの概要

2.1.3　テレビの「文字多重放送」に用いられている技術

つぎに、文字多重放送に用いられている地上波のすき間「空き走査線 VBI」に文字や図形等のデジタル信号を多重化して送信する方式について、具体的な例を用いて説明する。

2.1.3.1　地上波のすき間「空き走査線 VBI」とは

文字多重放送やデータ放送で使われている、地上波の「すき間」について説明する。テレビは画面上を水平方向に走る走査線が上から下に送られることによって 1 画面を構成している。これを毎秒 30 回、繰り返して動画を見せているわけである。この走査線が下から上に戻る時に「すき間」が生じる。この普段使用していない空いた空間を「垂直帰線消去期間 VBI：Vertical Blanking Interval（すき間）」もしくは、「垂直ブランキング」という。

この垂直帰線消去期間は、図 2.2 で示すように、受像機の垂直同期を無理にズラした時画面に表示される水平なバーの部分であり、従来は放送局間の伝送回線試験信号が挿入され特性チェックなどに使用されていた。しかし、様々な実験により、特定の位置ならば他の信号を挿入しても画面の乱れがなく、かつ幾分かの余裕があることが判明したので、放送局間でこの期間に文字信号を挿入し、受信側で抜き取り解読するようになったのが始まりである。 また、図 2.3 は、図 2.2 の中の「VBI」を拡大したものである。

具体的には、走査線 525 本中、すき間は 21 本あり、文字多重放送は 1985 年に、データ放送は 1995 年の時点で、それぞれ４本ずつ利用できるようになっている。

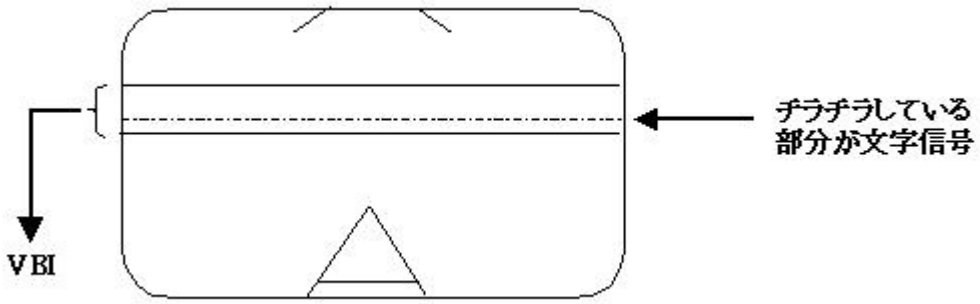

図 2.2.　VBI

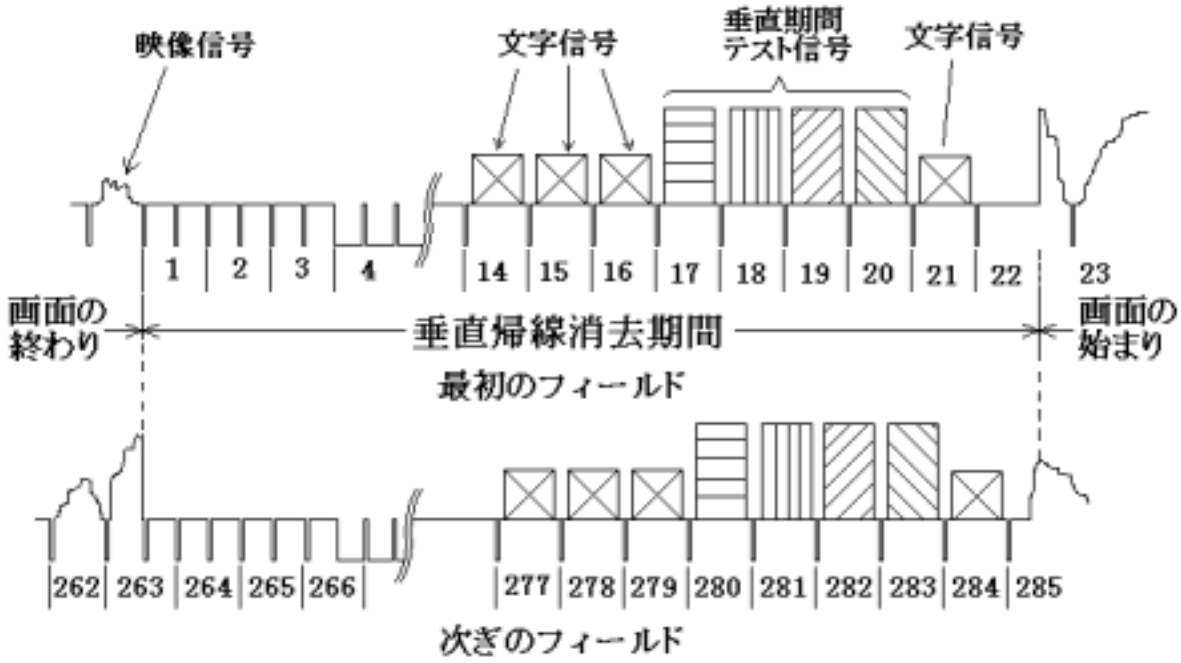

図 2.3　VBI への文字の挿入

また、「VBI」は水平走査期間を 22H含んでいる。この中で文字信号を挿入しても良い位置は、郵政省令「テレビジョン文字多重放送に関する送信の標準方式」により、水平走査期間番号で数えて最初のフィールドでは第 14H,15H,16H, および 21H である。(次のフィールドでは 277H,278H,279H,283H である。) 残りの第 17H から 20H は従来どおりの「垂直帰線期間テスト信号 VITS」が挿入され、特性チェックなどに使用される。このとき、文字信号は 2 値 (“ 0 ”か“ 1 ”) で表される。

2.1.3.2 「VBI」による画像表示の仕組み

VBI は言わば、ちょうどフィルムのコマとコマを仕切る空間のようなものである。もちろん非常に狭い空間なので、この中に大量の情報を詰め込むことは不可能である。しかし、一つの「VBI」は、デジタルデータにして毎秒 10KB の容量がある。これはテキストデータ程度の軽いデータを送るには十分な容量である。そこでこの空間に、テレビ番組の内容を補完するようなデータを詰め込んで送れないかという試みが、まず最初にはじまった。そこで生まれたのが、字幕などをテレビ画面に映し出す「文字多重放送」というわけである。

データの送信にはテレビの放送設備を使用するため、データ放送を提供するのも必然的にテレビ局またはその関連会社ということになる。テレビ放送で、画面を描き終わってから次の画面を描きはじめるまでのインターバル期間（垂直帰線消去期間）には、画面を描きはじめるタイミングをとるための信号（垂直同期信号）等も含まれているが、一部はいわゆる冗長部分になっており、これを利用して文字多重放送やデータ多重放送が行なわれている。

テレビの映像は、インタレース（飛び越し走査）といって、最初に 1 行おきに走査して粗い画面を描き、次の 262.5 回でその間を埋める画面を描く。2 回の走査で描かれる画面はフレーム、1 フレームを構成する 2 つの粗い画面はフィールドと呼ばれ、1 秒間に 30 回フレーム（60 フィールド）の静止画を送ることによって、動きのある映像を作っている。

画面の描画には、525 本の走査線が用意されているが、この 525 本分すべてが画面表示に使われているわけではなく、各フィールドの最初の 21 本相当のタイミングには、映像信号は含まれておらず、VBI として割り当てられている。21 本のうち、9 本分は、同期用に使われているが、残りの 12 本は基本的には空いており、このうちの 4 本分（1 秒あたり 240 本）を使った文字多重放送が、1983 年からスタートしている。1996 年には、残りの 8 本分（1 秒あたり 480

本）を使用したデータ多重放送の認可が下り、4 月にはフジテレビの「VBI」を使った E-NEWS（イーニュース）がスタートしている。その後も、テレビ東京やテレビ朝日、TBS……と次々にデータ多重放送局が誕生している。

余談であるが、パソコンやインターネットが普及する以前は、文字多重放送くらいにしか使い道の無かった VBI だが、パソコンやインターネットによる通信が一般的に行われてくるようになると、当然デジタルデータの送信にも使いたいという要望が生まれてくる。これが、パソコン向けのデータ放送を生むきっかけとなった。

しかしパソコンにデータを送るとなると、毎秒 10KB の伝送能力ではやや役不足である。そこで現在のデータ放送では「VBI」を 4 つ束ね、毎秒 40KB のデータ伝送能力を持たせている。これでアナログモデムでインターネットにアクセスするのと同程度の環境が実現したわけである。

文字多重放送が、文字コードやパターン等のプロトコルまで定めた規格であるのに対し、データ多重放送は多重化の方法（文字多重放送同様、1 走査線単位に 272 ビットのデータパケットを送る）だけを標準化したもので、自由なプロトコルでさまざまなデータが送信できるようになっている。具体的な方式には、テレビ東京のインターテキスト（アイティービジョンとも）、テレビ朝日のアダマス（ADAMS:TV-Asahi Data and Multimedia Service）、TBS のビットキャスト等があり、PC 向けに HTML 形式のコンテンツの配信も行なわれている。

2.1.4　文字多重放送の受信方法

文字多重放送を受信するには、前述のように、文字信号が VBI 期間に符号化されて放送局から送信されているので、抜き取る回路と解読する回路が必要であり、これらをまとめて「デコーダ」という。このデコーダについて、文字多重放送受信機の構成の面から図 2.4 および図 2.5 を用いて説明する。

映像から抜き取られた文字信号は、電波伝搬路でのゴーストや送信機の特性などによるひずみを持っているので、波形等化回路で補正される。その後、符号識別回路にて 2 値（“ 0 ”か“ 1 ”）化される。2 値化されたデータは次のデータ処理・復号回路で解読され、誤り訂正が行われる。この回路が、今回、研究テーマに挙げた「差集合巡回符号エラー訂正回路」の主な部分に、相当する。この出力は表示回」に送られ TV/文字切換え回路を通って、ブラウン官に表示される。同期再生回路は、これらの一連の処理に必要なタイミング信号を再生する。TV/文字切換え回路は、通常の放送番組と文字多重放送番組とを画面ごとに

切り換えるか、または、通常の放送番組に文字信号を「スーパーインポーズ（2種類以上の番組等を同時に表示する方法)」するかを選択をする回路である。

また、文字多重放送における音は付加音と呼ばれる。文字信号の中に符号化した付加音情報を送出すると、受像機内のデコーダが自動的にこれを検出し、シンセサイザから電子音を発生し、音声信号と同様にスピーカより出力するようになっている。

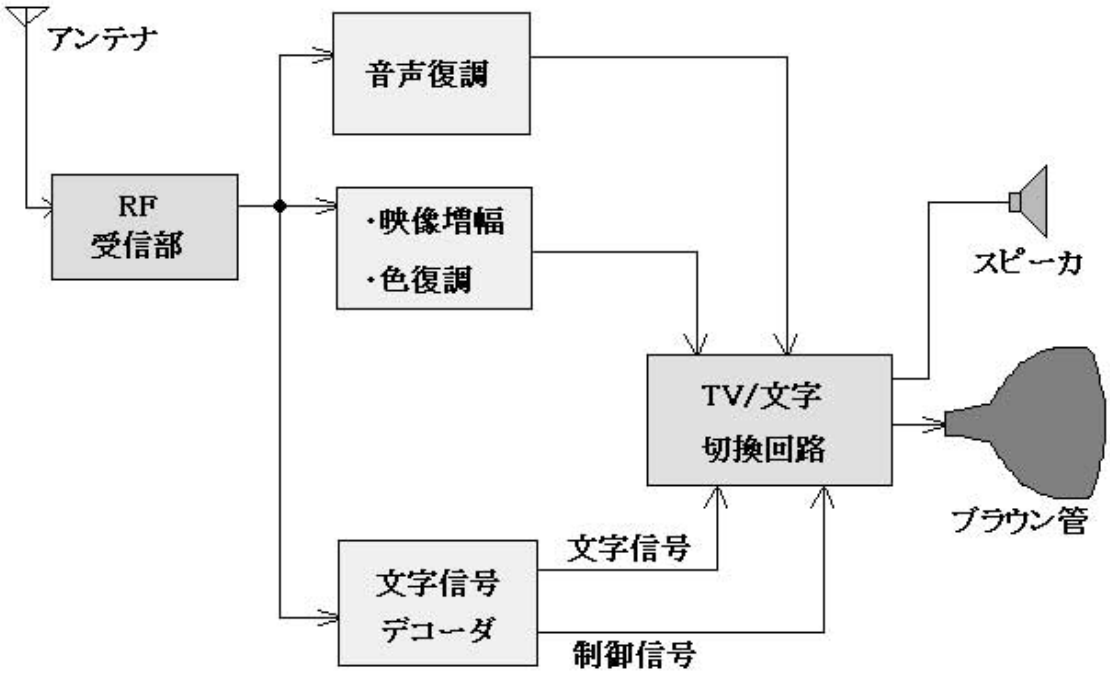

図 2.4　文字多重放送受信機の構成

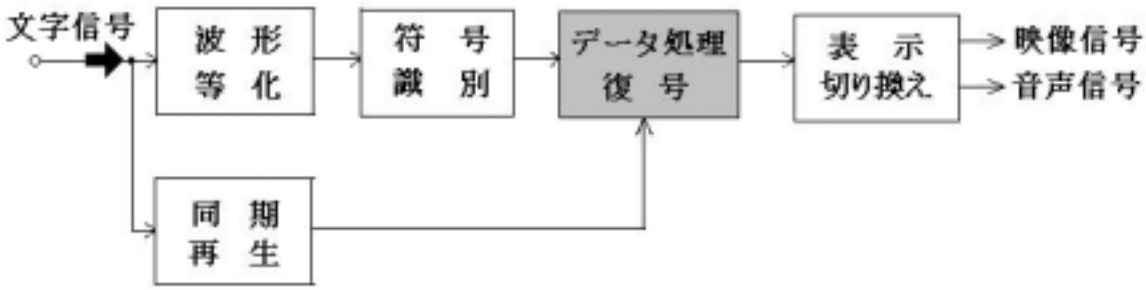

図 2.5　文字多重放送のデコーダの構成

また、文字多重放送の表示画面は、図 2.6 に示すように周辺領域、ヘッダ文表示領域、本文表示領域に分けられおり、ヘッダ文表示領域には日付、番組名、ページ番号などが表示される。

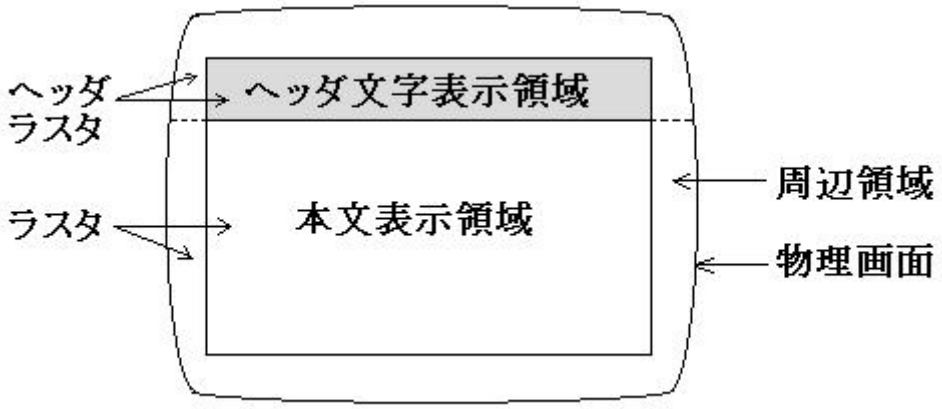

図 2.6　文字多重放送の表示画面

2.1.5　文字多重放送のエラー訂正の必要性

前述したように、一般的に、文字多重放送では「文字信号」などの「ディジタル信号」は、伝送路による妨害（ゴースト、自動車の点火雑音、受信入力電界低下など）や送信機の特性によるひずみを受け、エラー（符号誤り）発生する。この結果、文字多重放送では、文字信号が 2 値（“ 0 ”か“ 1 ”）化されていることにより、表示文字の脱字、誤字などになって現れる。
例えば、平仮名 50 音「**あ**」から「**ん**」までを 2 値で表すとするならば、

「**あ**」⇒「**000000**」、「**い**」⇒「**000001**」、「**う**」⇒「**000010**」

・・・・・「**を**」⇒「**110000**」、「**ん**」⇒「**110001**」

と、6 ビットから成る数値で表すことができる。

ところが、送信するときの文字信号が「**い**」の「**000001**」であったのに対し、送受信中に「エラー」が生じて、受信した際に「**000000**」となっていた場合、表示される時に「**あ**」となって現れる。

よって、文字多重放送において、「エラー（符号誤り）」が与える影響は非常に大きいと言える。ゆえに、受信機には、精度の高いエラーの検出と強力な訂正能力が必要不可欠である。

今回、研究テーマに設定した「差集合巡回符号　エラー訂正回路」も、この一例であり、これからよりいっそう盛んになっていくであろう、テレビの「文字多重放送」に接していく上で「エラー訂正回路」を研究し設計することが、有意義であると判断し卒業研究テーマに設定した。

「文字多重放送」を開始した、NHK では、特にこのエラー訂正能力を「BEST（Burst and random Error System for Teletext）方式と呼んでいる。この BEST 方式によると、１データパケット中に生じた８ビットの全ての誤りを訂正する能力をもっている。これに加え、二重、三重のエラー検出を行っているので、誤字、脱字のほとんどないシステムになっている。

2.1.6　文字多重放送の特徴、必要とされた理由

次に、テレビの「文字多重放送」には、他のメディアにはない幾つかのユニークな特徴があるので、その代表例を以下のように示す。

- 見たい番組を何時でもすぐ見ることができる「随時性」
- さまざまな番組の中から、知りたい情報を選べる「選択性」
- 常に新しい情報が更新されている「速報性」
- 専用プリンターで必要な画面のコピーができる「記録性」

したがってこれらの特徴は、その性質上テレビを見る者の選択にまかされているので、テレビの「文字多重放送」はいわば、「放送」と「新聞」が一体となったようなものであるといえる。

文字多重放送を始めるきっかけになったのは、当時、国民の約 70%が、「放送」と「新聞」が一体になった、ラジオテレビ欄のような、「番組のタイムスケジュール」等の表形式の一覧表示を、望んでいたことである。また、「番組のお知らせ」形式の、文字による情報提供への期待度も高く、約 60%とを占める。しかも、統合サービス型テレビは、文字・静止画・動画を組み合わせたマルチメディア表示を行うので、番組表の中に動画の番組案内を組み合わせることが可能である。中でも、ハイビジョンディスプレーは、標準テレビに比べて非常に多くの文字を表示できるため、一覧性の面において、良い表示が行える（当時の実験番組では、約三千文字を表示した。）ので、特に注目されている。

2.2　FM ラジオの「文字多重放送」

本節では、FM ラジオの「FM 文字多重放送」を、概要などを具体的な例を挙げて紹介する。

2.2.1　「FM 文字多重放送」とは？

「ＦＭ文字多重放送」とは、ＦＭ音声放送と同じチャンネルに文字情報データを重ねて放送することである。FM 文字多重放送は、通常のラジオの FM 放送の音声で使われている電波のすき間に、文字や図形等のデジタル信号を多重化して送信する放送で、ニュースや天気予報などの文字情報を無料で提供している。このような性質上、テレビの文字多重放送と同じようなものだと考えられる。

身近な所では、1994 年に FM 東京が「見えるラジオ」という愛称で初の放送を開始した。現在は、この他には、NHK や民放各局が、番組と連動した、あるいは独立した文字情報サービスを提供している。また、「道路交通情報通信システム」の情報配信にも利用されている。

なお、近頃、巷でよく耳にする、この見えるラジオとは、「 FM 文字多重放送」を利用し、ラジオの液晶モニターに様々な情報を文字として表示しているものである。 ちなみに、「見えるラジオ」とは、JFN(Japan FM Network)で使われている愛称である。他には、J-WAVE では「アラジン」、FM802 では「Watch-me」、Kiss-FM KOBE では「Kiwi」という愛称がついている。

2.2.2　「FM 文字多重放送」の原理

FM 放送は、もともとが左右の和（L＋R＝モノラル）と差（L－R）という 2 つの音声信号（50～15kHz）を多重化し、ひとつの搬送波（carrier）に乗せたもので、送信時の変調方式に文字通りの「FM（Frequency Modulation～周波数変調[*1]）」が使われている。音声信号の多重化は、差信号を 38kHz で変調し（この搬送波は副搬送波[sub carrier]という）、和信号の上の帯域に移動させる[*2]。デジタル信号も同様の方法で多重化しており、こちらは 76kHz の副搬送波を使い、差信号のさらに上に重畳する。

デジタル信号の多重化では、ＮＨＫが開発した「ＤＡＲＣ（ダーク：DAta Radio Channel（2.2.4 節参照））と呼ばれる方式が用いられている。このＤＡＲＣでは、

デジタル変調にＬＭＳＫ（Level control MSK[*3]）という新しく開発した変調方式を採用している。音声信号の大きさに合わせて、デジタル信号側のレベルをダイナミックに調整することにより、音声信号との干渉を最小限に押さえているのが大きな特徴である。伝送速度は 16kbps である。エラー訂正用のパリティを除いた実データ[*4] は約 8kbps であり、通常の文字多重放送では、15 文字×2 列のディスプレイに、様々な文字情報が表示されるようになっている。

注）

[*1] ある信号を別の信号に乗せることを変調といい、この時使用する搬送用の電波のことを搬送波という。ＡＭ放送で用いている「ＡＭ（Amplitude Modulation～振幅変調）」が、信号波の振幅に応じて搬送波の振幅を変化させるタイプであるのに対し、「ＦＭ」は搬送波の周波数を変化させる方式で、混信や雑音が少なく、忠実度が高いのが特徴である。ちなみに、位相を変化させるタイプもあり、こちらは「ＰＭ（Phase Modulation～位相変調）」という。

[*2] 和信号の 50～15kHz に対し、差信号は 38kHz±15kHz に移動する。ひとつの帯域が 2 チャンネルに分割されたことになる。

[*3] AM、FM、PM のデジタル変調版を、ASK（Amplitude Shift Keying）、FSK（Frequency Shift Keying）、PSK（Phase Shift Keying）という。MSK（Minimum Shift Keying）は、「0」と「1」の状態に異なる周波数を割り当てる FSK の一種で、ビットの変わり目に、位相が必ず±90 度になるような最小の周波数差を割り当てたもの。

[*4] FM 文字多重の符号化方式は、テレビの場合と同様、BEST（Burst and random Error correction System for Teletext）と呼ばれるもので、テレビの文字多重放送や衛星ディジタルにもこの方式が用いられている。エラー訂正は、テレビの場合と同様「差集合巡回符号」である。

2.2.3　FM文字多重放送のデータの流れ

文字多重放送のデータの流れについて、送信機から受信機に送られる信号にしたがって説明する。

そもそも、人間の声は、アナログ信号なので、これをまずディジタル化する必要がある。そのための必要とされる、『ディジタル・エンコーダ（変調器）/デコーダ（復調器）（二つを一体として CODEC という）』で、音声をバイナリ・データに変換する。

ワード間の無音、ディジタル信号処理（DSP）の負荷をできるだけ低減するために（より小型の DSP で対応するために）、通常このディジタル化処理では音声に含まれる冗長な信号成分を除去してデータ量を大幅に圧縮する。

ディジタル化されたデータは，ひとつながりのデータ列となるが（これをディジタル・データ・ストリームという）、そこにリード・ソロモンなどのエラー訂正符号、あるいは最新手法の「ターボ符号」を追加する。

これらの符号が必要な理由は、伝送路上で受けるノイズなどの影響により、伝送中にビット・エラーが発生するためである。エラーを訂正するには、伝送するデータ自身に冗長性をもたせる必要がある。つまり、エラー訂正符号などを組み込まなければならないのである。

こうして作られた一連の、エラー訂正符号を追加された信号は、送信機モジュールに送られ、受信機に向け伝送され、受信機モジュールにて受信され、エラー検出・訂正され、再び復調される。

2.2.4　その他の FM 多重方式の例「ＤＡＲＣ」

その他の、FM 多重方式の例として、「DARC」が上げられる。「DARC」について説明する。

2.2.4.1　「DARC」とは

「DARC」（DAta Radio Channel）とは、前述したように、ＮＨＫで開発されたＦＭ文字多重方式である。デジタル変調にＬＭＳＫという、変調方式を採用している。音声信号の大きさに合わせて、デジタル信号側のレベルをダイナミックに調整することにより、音声信号との干渉を最小限に押さえているのが大きな特徴である。伝送ビットレートは、１６kbit/sec である。

2.2.4.2　「DARC」」の特徴と階層構造

「DARC」の、主な特徴を示す。

- 従来のＦＭ音声放送と共存できる。
- ＦＭステレオ受信ができるエリアであれば、DARC 文字多重も受信できる。
- 誤り訂正能力が比較的高い。
- 自動車などの移動体でも受信できる。
- 既存の放送設備に多重化器を追加するだけで機器は準備できるので、数千万円程度の 追加設備投資で放送できる。

「DARC」には、大きく分けて以下のような、「１～６」の階層がある。

- 階層１　伝送路

  ＬＭＳＫ　１６ｋｂｉｔ/ｓの変調を行う。

- 階層２　誤り訂正

  差集合巡回符号を２重に用いた積符号によって誤りを訂正する。

●階層３　データパケット

誤り訂正後に取り出される 176bit 構成のデータである。「データパケット」は、「プリフィックス」と「データブロック」から成り、「プリフィックス 32bit」の 【構成１】と「プリフィックス 16bit」の【構成２】がある。

・構成１：文字・図形・交通情報（VICS）などの一般的な番組から構成される。

・構成２：放送局名・年月日（ユリウス暦）・時刻・代替周波数（受信状態が 悪くなったときに、同一放送局の他の周波数を選択する目的などに使う。）から構成される。

●階層４　データグループ

データパケットの同一の番組番号種別・同一のデータグループ番号をデータパケット 番号の順に並べるとデータグループになる。データグループ内でも 16bit のＣＲＣによる誤り訂正 を行う。１データグループが１ページに当たる。

●階層５　番組

番組は、複数のデータグループ（ページ）から構成される。また、表示に必要なデータユニットから構成される。

●階層６　提示

表示方法が規定されている。

・字数：15.5 文字Ｘ2.5 行・15.5 文字Ｘ8.5 行　他

・文字種類：８単位符号系　JIS コード(SHIFT JIS ではない)・追加文字　他

・着色：４０９６色中３２色を文字単位毎

・属性：点滅・アンダーライン・反転　他

・図形

・その他

●サービスレベル

・レベル１

１５.５文字 Ｘ ２.５行（例：ＪＦＮの「見えるラジオ」）

・レベル２

１５.５文字 Ｘ ８.５行、図形

・レベル３

交通情報データ

●その他のサービス

ＤＧＳＰ・ページング・他 様々な応用

以上により、本章での、卒業研究テーマである「差集合巡回符号　エラー訂正回路設計」を研究していく上で、「エラー訂正」が使われている分野の一つであるテレビや「文字多重放送」について、またテレビの「文字多重放送」と同様な技術を用い、近年主流になりつつあるＦＭラジオ放送の「文字多重放送」についての紹介を終える。

第 3 章

パリティ・チェックと差集合巡回符号

本章では、前章で述べたようにテレビの「文字多重放送」における情報データの送受信中に様々な理由により生じるエラーに対して、一般的に用いられる訂正方法であり、かつ今回の研究テーマである「差集合巡回符号　エラー訂正回路設計」に用いる方式「パリティ・チェック」と「差集合巡回符号」について例を挙げて順に具体的に説明する。[9]

なお、例として用いている数値およびビット数等については、今回応募した「Design Wave 設計コンテスト 2002」の仕様にて設定された物も含まれている。また、実際に一般にテレビの文字多重放送にて使われている物と異なることを予め了承頂きたい。

3.1　パリティ・チェック

本節では、テレビの「文字多重放送」において情報データに生じるエラーの基本的な訂正方法である、パリティ・チェックについて具体的な例を用いて説明する。

3.1.1　パリティ (parity)とは

パリティとは、原義は「等価」という意味の名詞である。しかし、実際に用いられる場合は、「parity」は「奇偶（奇数と偶数）」という意味で、コンピュータ関連では、データの誤りを検出する目的で、データに付加されるビット情報またはこのパリティ・ビットを使用してデータの誤りを検出することを表す。

すなわち、送受信したデジタルデータが、送信側と受信側とで間違いなく同一内容かどうかを確認するしくみの一種として、「パリティ・チェック(parity check)」という技法が普及している。また、実際にこの用途に使われる１ビットのことを「パリティ・ビット(parity bit)」と呼ぶ。

3.1.2　パリティを用いる利点

「パリティ」を使えば、たとえば 2 つのデバイス間でデータを転送する場合に、送信元からデータといっしょに送られたパリティと、受信先に届いたデータから計算したパリティを比較することで、そのデータが正しく転送できたかどうかを判断できる。また、PC 互換機のメモリシステムでは、8bit につき 1bit のパリティを付加していることが多い。

3.1.3「ビットの奇偶」で検査するパリティ・チェック

本節では前節までに示した「パリティ」を用いて「ビットの奇遇」で検査するエラー（誤り）検出技法の一つである「パリティ・チェック」について説明する。

3.1.3.1　パリティ・チェックとは

パリティ・チェックとは、デジタルデータの送受信の際などに用いられるデータエラー検出技法の一つであり、エラーのため受信データが、送信データと異なっていないかどうか、チェックするのに用いられる。

コンピュータ内部のデータは､0 または 1 からなる 2 値符号で表現されている。このとき、ひとまとまりとして処理されるデータ（1byte のデータなど）において、1 の出現回数を計数し、その数が偶数個か、奇数個かを表す 1bit のデータを冗長ビット（パリティ・ビットと呼ばれる）としてデータに付加しておく。そしてこのデータを他のデバイスに転送したとき、受け取った側でもデータに含まれる 1 の数を同じように計数し、結果がパリティ・ビットの結果と一致するかどうかを確認する。このとき結果がパリティ・ビットの値と異なるなら、データを構成するどこかのビット情報が不正であると判断できる。

つまり、データを受信したとき、もう 1 度パリティを計算し、受信したパリティビットと一致しなければ、どれかのビットのデータが変わった（反転した）ことが分かる。

ただしパリティによるエラー検出では、どのビットがエラーを起こしているのかは分からない。また 1 の数が偶数個か奇数個かしか検査していないので、データ内部でエラーが偶数個あると、エラーの検出自体にも失敗する。つまり、

言い換えるならば、パリティでは、パリティ・ビットも含めて、奇数個のビットの誤りは検出できるが、偶数個のデータの誤りは検出できないということである。

パリティ・チェックは、エラー検出のために必要な冗長ビットが少ないので手軽に低コストで実装できるが、それにより可能なエラー検出能力には限界がある。

3.1.3.2 パリティ・チェックの利点・欠点

データ中の“１”もしくは、“0”の数を数える演算（詳しくは後の項目参照）を、データ送信（保存）時と受信（読み出し）時に行うことによって、データの受け手側は、独自に演算した結果のパリティが“０”のはずなのに“１”だった場合、またはその逆の場合に、そのデータが誤りであると判定し、しかるべき処理(通信等なら再送要求。メモリエラーならシステムの緊急停止など)を行うことになる。

ただし、上の例でいえば、もし「101（=5)」であるべきデータが「000(=0)」、「011(= 3)」、「110(=６)」、「1001(=９)」……などに化けて受け渡されても、パリティ・ビットは同様に“０”となり検出できない理屈になる。また、この方式には「１データ内のエラー発生個所が奇数の場合にしか発見できない」という限界がある。したがって、エラーが１度だけ発生した場合に発見できる確率は、約２分の１にすぎない。

しかし、エラーが２回起きれば、発見できない確率は２分の１の自乗＝４分の１となり、逆に「どちらか１回のエラーが検出できる確率」が４分の３に上がる。３回エラーが起きれば、この確率は８分の７になる。このように、エラーの発生回数が多ければ多いほど、遅かれ早かれ発見できる確率は100%に近づいてゆく。

以上に述べたようにパリティ・チェック方式は、実用上エラー検出確率は十分高く、かつ、処理も簡単なため、通信やメモリー(RAM)のエラー検出技法など、パソコン関連分野で非常によく使われている。

3.1.3.3　エラー訂正が可能な ECC

パリティ･チェックで検出できるのは「エラーが発生したかどうか」ということだけであり、前述したとおり、ごくまれには「発見できない」ということも起こり得る。

そこで、エラーの発生をより確実に検出し、かつ、エラーを含むデータ自体とあわせて演算することでデータを復元できる符号“ ECC(誤り訂正符合、Error-Correcting Code)”も考案されており、サーバなど高度なデータ信頼性を要求される用途では、この技術が採用されている。

現在パソコン用にも普及しつつある ECC メモリ(Error Check and Correct memory)もその活用例のひとつで、パリティ・ビット方式より冗長なエラー検出･訂正用符号を使う仕組みになっている。

当然ながら、ECC なし(パリティ方式、あるいはそれもない)メモリより高価にはなるが、(ECC メモリの機能を利用可能なマザーボードを採用している PC では)安定度は向上する。

3.1.3.4　パリティ・チェックの原理

元データに含まれる「1」または「0」の数を計算し、そのひとまとまりのデータ（一般に７～８ビット）に対して、１つのパリティ・ビット(冗長ビット)を付加する。3.15 節より具体的な例を用いて説明する。

3.1.4 「８ビットデータ」の場合のパリティ･チェック

例えば、８ビットのデータに対して計算した結果をパリティ・ビットに入れた例を図 3.1 に示す。

10 進数	８ビット･データ	パリティ
0	00000000	0
1	00000001	1
2	00000010	1
3	00000011	0
4	00000100	1
5	00000101	0
6	00000110	0
7	00000111	1
8	00001000	1
9	00001001	0

図 3.1　8 ビットデータのパリティ･ビットの例

図 3.1 の具体例は、8 ビットの元データの中の、「０」と「１」の個数を数え、「０」の数（もしくは、「1」）の数が、偶数なら「０」、奇数ならば「１」を、パリティとして、パリティ･ビットに付加したものである。

つまり、図 3.1 から、

（１）１0 進数「６」の場合、８ビットデータは「00000110」であり、この一塊のデータの中には、

- 「0」⇒　6 個（偶数）
- 「1」⇒　2 個（偶数）

が、含まれているので、この場合は、「0」、「1」の数が異なってはいても、共に「偶数」である。したがって、パリティ・ビットには「0」を付加する。

しかし、

（2）10 進数「7」の場合、8 ビットデータが「00000111」であり、この一塊のデータの中には、

- 「0」⇒　5 個（奇数）
- 「1」⇒　3 個（奇数）

が、含まれているので、この場合は、「0」、「1」の数が異なってはいても、共に「奇数」である。したがって、パリティ・ビットには「1」を付加する。

以上の様に、他の場合（図 3.1 の 10 進数 0～9）においても、同様に、パリティの値を容易に求められる。

3.1.5 「９ビットデータ」場合のパリティ･チェック

次に、データが 9 ビットの奇数ビットについて考えてみる。仮に、図 3.2 のような、9 ビットデータとパリティがあったとする。

10 進数	９ビット･データ	パリティ
0	000000000	0
1	000000001	1
2	000000010	1
3	000000011	0
4	000000100	1
5	000000101	0
6	000000110	0
7	000000111	1
8	000001000	1
9	000001001	0

図 3.2　9 ビット･データのパリティビットの例

図 3.2 の場合の具体的な計算は、９ビットの元データの中の、「０」と「１」の個数を数え、「０」の数（もしくは、「1」）の数が、偶数なら「０」、奇数ならば「１」を、パリティの値として求めたものである。

ここで上記の図 3.2　から、

（1）10 進数「6」の場合、9ビットデータは「000000110」であり、この一塊のデータの中には、

- 「0」⇒　7個（奇数）
- 「1」⇒　2個（偶数）

が、含まれているので、「0」の数が「奇数」、「1」の数が「偶数」である。

また、

（2）10 進数「7」の場合、9ビットデータが「000000111」であり、この一塊のデータの中には、

- 「0」⇒　6個（偶数）
- 「1」⇒　3個（奇数）

が、含まれているので、「0」の数が「偶数」、「1」の数が「奇数」である。

したがって、前述の8ビットのデータの時のように、「0」と「1」の数が共に「偶数」や「奇数」ではなく、異なっているので、8ビットの場合と同様にして、パリティの値を求めることができない。

では、どのようにして、図 3.2 の場合のパリティの値を求めるか、その具体的な方法を 3.1.5.1 節に示す。

3.1.5.1　奇数ビットデータにおけるパリティの求め方「偶パリティ」

次に奇数ビットデータにおけるパリティの求め方を図 3.3 を用いて具体的に示す。

奇数ビットのデータのパリティを求めるのによく用いられる方法は「偶パリティ」である。この方法は、元となる奇数ビットの情報そのものに、新たに、パリティビットとして、もう 1 ビットを付加する方式である。新たに付加する 1 ビットは、元となる奇数ビットの情報の中に含まれる“ 1 ”の数に応じて、“ 1 ”か“ 0 ”に変化する。すなわち、このパリティ・ビットを含めて、全ビットの“ 1 ”の数が偶数になるように調整される。

（3.1.5 節の場合とは異なり、“ 0 ”の数に関わらず、情報ビットの中の“ 1 ”の数にのみ、依存する。）

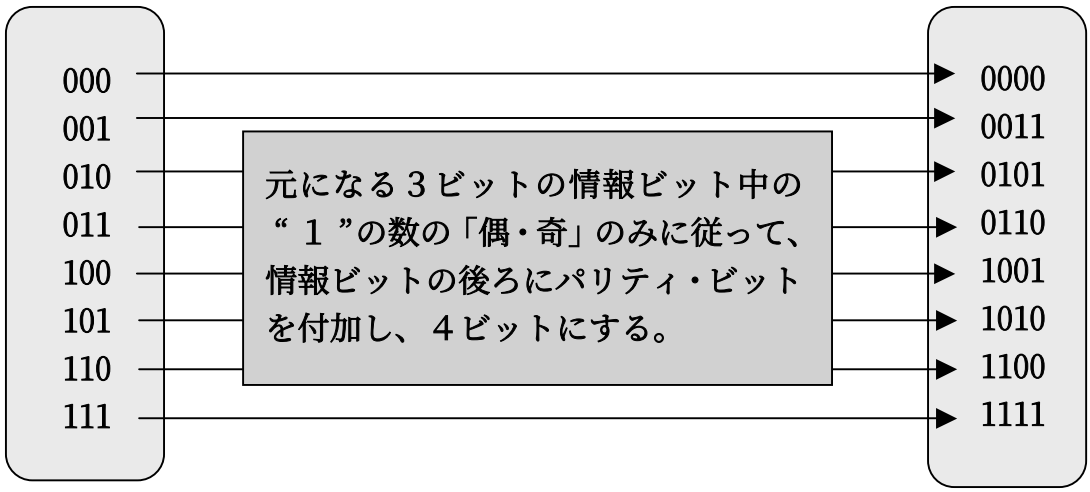

図 3.3　パリティビット生成

以上のように、パリティ・ビットを付加する方法を、特に「偶パリティ」と呼ぶ。（また、逆に“ 1 ”の数が奇数になるようにパリティ・ビットを付加する場合は、「奇パリティ」と呼ぶ。）

3.1.6　パリティ･ビットによるエラー検知

3.1.6.1　パリティ･ビットとは

ディジタル･データ伝送においては、エラーのチェックにパリティ･チェックが用いられ、そのチェック方式に用いられるのがパリティ･ビットである。

- ８ビット･データ中の‘１’の個数が偶数･･･パリティ･ビットは‘０’
- ８ビット･データ中の‘１’の個数が奇数･･･パリティ･ビットは‘１’

式で表すと、

$$\text{“Parity bit”} = D_0 \text{ xor } D_1 \text{ xor } D_2 \text{ xor } D_3 \text{ xor } D_4 \text{ xor } D_5 \text{ xor } D_6 \text{ xor } D_7$$

$$= (D_0 \text{ xor } D_1) \text{ xor } (D_2 \text{ xor } D_3) \text{ xor } (D_4 \text{ xor } D_5) \text{ xor } (D_6 \text{ xor } D_7)$$

となる。（Parity bit：パリティ･ビットを表す）

ちなみに、パリティ･ビットは、具体的には表 3.1 に例に示すように求められ、図 3.4 の様な多入力の XOR 回路によって生成できる。

表 3.1　パリティ･ビット発生

８ビットのデータ								パリティ･ビット
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

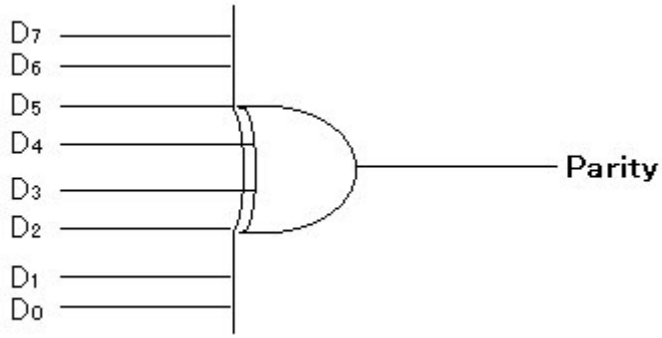

図 3.4　パリティ・ビット発生回路 1

　また、前述のように多入力 XOR で構成されるパリティ・ビット発生回路を実際に論理ゲートを用いて表したものを図 3.5 に示す。

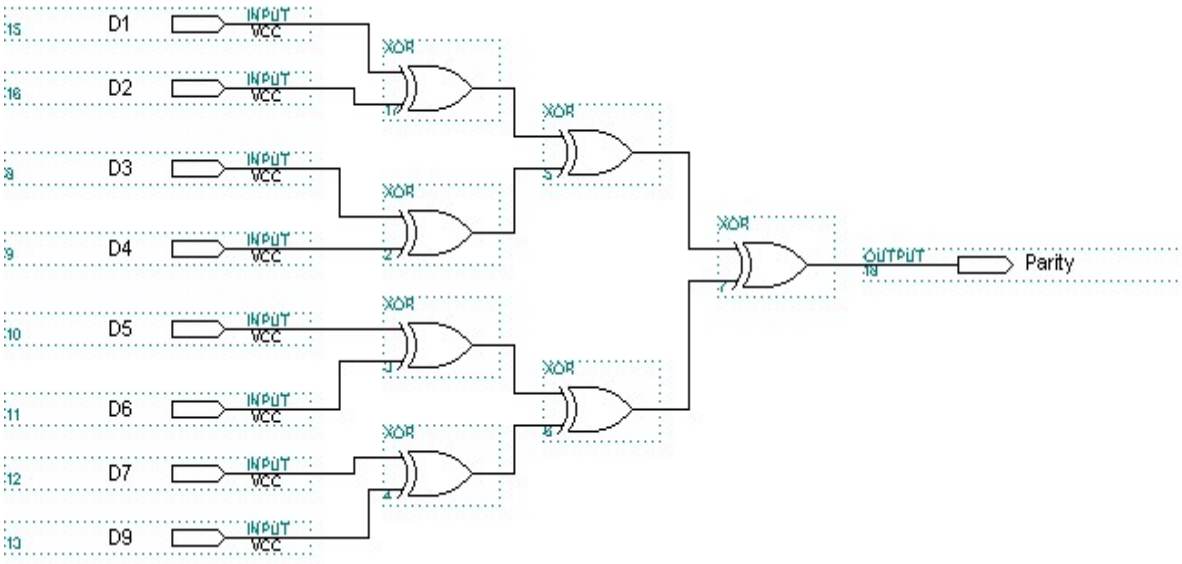

図 3.5　パリティ・ビット発生回路 2

3.1.6.2　パリティ･ビットによるエラー検出方法

前述のように、８ビットデータに生成したパリティ･ビットを加えた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 を取れば、その出力が“１”の時は、何処かのビットでエラーが起きていることを検出できる。

式で表すと、

$$\text{“Error”} = D_0 \text{ xor } D_1 \text{ xor } D_2 \text{ xor } D_3 \text{ xor } D_4 \text{ xor } D_5 \text{ xor } D_6 \text{ xor } D_7 \text{ xor } P$$

となる。（Error：エラー検知回路出力、P：Parity bit を表す）

ただし、

- “ Error ”＝1 のとき、エラー有り（検出）
- “ Error ”＝0 のとき、エラー無し

（欠点としては、2 ビットのエラーが検出不可能であること、またエラーしたビットの位置を検出することが不可能であることが挙げられる。）

エラー検出回路は、具体的には表 3.2 に例に示すように求められ、図 3.6 のような多入力の XOR 回路によって作成できる。（ 回路の出力を“ Error ”とする。）

表 3.2　エラー検出ビット発生

９ビットのデータ									エラー（Error)
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	P	Error
1	0	1	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0

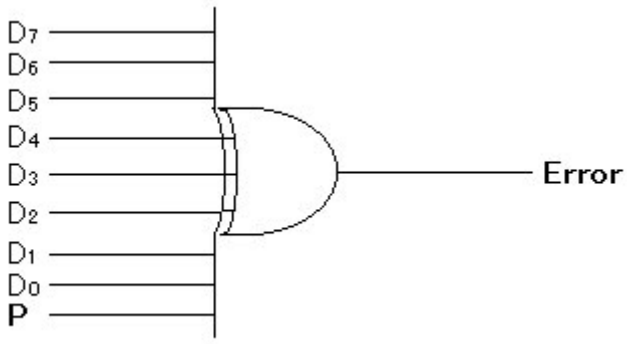

図 3.6　エラー検知ビット発生回路 1

図 3.7 は、多数入力 XOR で構成されるエラー検出ビット発生回路を、実際に論理ゲートを用いて構成したものである。

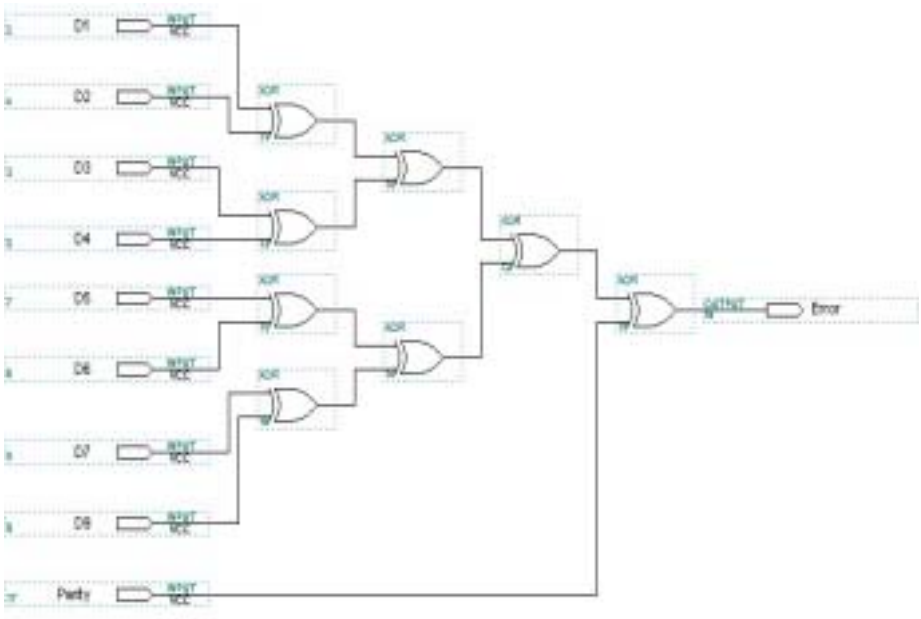

図 3.7　エラー検出ビット発生回路 2

検出方法の原理は、通常は 9 ビット幅のデータ中の“ 1 ”は、常にパリティ・ビットが入っていることにより偶数に保たれているが、回路の出力が「“ Error ”＝1」ということは、9 ビット幅のデータの中の“ 1 ”の数が、奇数になっていることになるので、何処かに間違ったデータを含んでいることを意味し、エラーを検出できる。

また、表 3.1 に示すように、エラーが検出される「“ Error ”＝１」になる場合の例を表 3.3 を用いて示す。

表 3.3　エラー検出

９ビットのデータ									エラー（Error）
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	P	Error
0	0	1	1	0	0	0	1	0	1
1	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	1	1

3.2　差集合巡回符号

本節では、テレビの「文字多重放送」において情報データに生じるエラーの基本的な訂正方法であり、今回の研究テーマである「差集合巡回符号　エラー訂正回路設計」に用いる方式「差集合巡回符号」について具体的な例を用いて説明する。

3.2.1　差、差集合（Difference）とは

「差、差集合（difference)」とは、第１の物体と、第２の物体を完全に反転させたものとの 「交差（intersection)」 を取る操作のことである。したがって、オブジェクト「A」の内側と、オブジェクト「B」の外部の点だけが、両方の物体の 「difference」 となる。

結果として、図 3.8 に示すように、第１の物体から第２の物体を除く、引き算になる。つまり、図 3.8 の灰色部分に相当する。

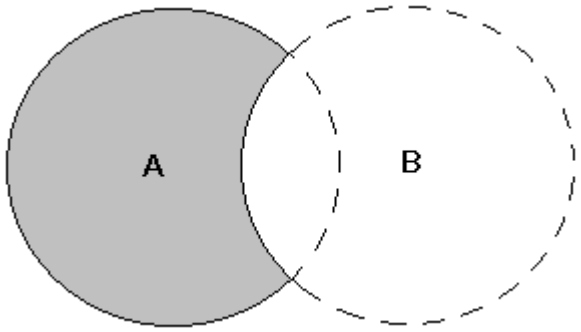

図 3.8　２つの物体の 差集合

ここで、差集合について、理解し易い計算例を以下の様に示す。

集合 A={1,2,3,4,6,12}, B={1,2,3,6,9,18} に対して差集合を計算する。

●差集合が Ａ－Ｂ ならば、

Ａ－Ｂ＝{1,2,3,4,6,12}－{1,2,3,6,9,18}＝{4,12}　　となる。

なお、集合計算には他に、和集合と共通集合があり、それらの計算についても、差集合の場合と同様、集合Ａ，Ｂを用いて表せる。

また、図 3.9 と図 3.10 は、それぞれ差集合と同様に、計算結果で得られる和集合および共通集合を示したものである。

●和集合がＡ∪Ｂならば、

Ａ∪Ｂ＝{1,2,3,4,6,12}∪{1,2,3,6,9,18}＝{1,2,3,4,6,9,12,18}　　となる。

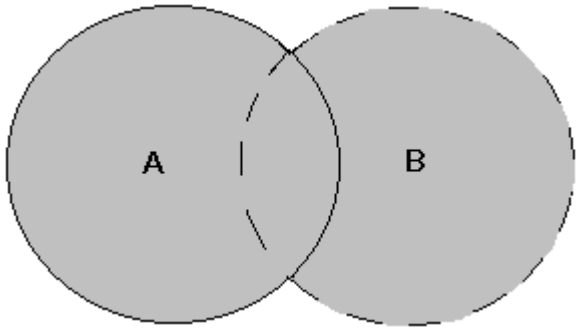

図 3.9　和集合

●共通集合がＡ∩Ｂならば、

Ａ∩Ｂ＝{1,2,3,4,6,12}∩{1,2,3,6,9,18}＝{1,2,3,6,}　　となる。

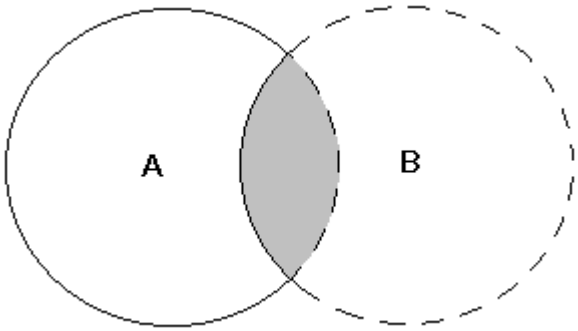

図 3.10　共通集合

3.2.2　巡回符号とは

巡回符号とは、ある幾つかの符号の並びを左や右に、１符合ずつ巡回（シフト）させ、再び元の符号の並びに戻すことのできる符号である。

『S = {A1,A2,A3,A4,A5}』という、5 種類の符号の並びを用いて、左に 1 符合ずつ巡回する場合の具体的な動作を次に示す。

S = { A1,A2,A3,A4,A5}　⇒　S = { A2,A3,A4, A5, A1}

⇒　S = {A3,A4,A5, A1,A2}　⇒　S = {A4, A5, A1,A2,A3}

⇒　S = {A5, A1 A2,A3,A4}　⇒　S = { A1,A2,A3, A4, A5, }　　となる。

以上のような動作をする幾つかの符号の並びを、特に「巡回符号」と呼ぶ。

3.2.3　差集合巡回符号とは

差集合巡回符号とは、「3.2.1」節や「3.2.2」節で示したような性質を持っているものであるが、特に文字多重放送などのエラー訂正方法として用いられる場合は、前述した「パリティ・チェック」を複数備え、それらを巡回（シフト）させ、エラーを検出し、訂正することを目的とする。

ちなみに、差集合巡回符号による符号化とは、この場合、複数のパリティ・チェックをシフトした形式でも、成立するように符号化することを意味する。また性質上、その複数のパリティ・チエェックに、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と判断できる方法である。

3.2.4　差集合巡回符号の計算

本節では前節で述べたパリティ・チェックを持つ、「差集合巡回符号」の基本的な動作について、具体例を用いて説明する。

例として、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ビットを付加し、21 ビットのデータを送信する差集合巡回符号をとりあげ、そのエラー訂正方法について、表 3.4 及び表 3.5 を用いて説明する。

表 3.4　21 ビットの送信ビット

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 3.5　パリティ・チィック生成

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

3.2.4.1

全てのパリティ･チェックにおいて、XOR 出力が“ 0 ”になる場合

表 3.4 の中の、各パリティ･チェック「A_1～A_5」に、“ 1 ”に対応する送信ビット（SB：Sent Bits）の排他的論理和（XOR）をそれぞれ取ると、どのパリティ･チェックの場合においても、XOR の結果は“ 0 ”となる。その計算式を、表 3.5 に示す。

各パリティチェックにおいて、対応する送信ビットの排他的論理和をとると結果が“ 0 ”になる理由は、3.1 節で示したとおり、複数の値の排他的論理和をとると、“ 1 ”の数が奇数の場合、結果は“ 1 ”となり、“ 1 ”の数が偶数の場合は、結果は“ 0 ”となるように付加ビットを決めているからである。

このことより、例えば、表 3.5 の中の例を用いて考えるならば、パリティ・チェック A_1 の場合は、

（パリティ･チェック A_1）

= SB(9) xor SB(12) xor SB(13) xor SB(18) xor SB(20)

= “0” xor “1” xor “1” xor “1” xor “1”

= “0” となる。

なぜならば、“ 1 ”の数が「偶数」だからである。

他のパリティ・チェックの場合も同様に求められる。以上のようにして求められた結果を表 3.5 に示す。

3.2.4.2　SB(20)がエラー発生し反転している場合

表 3.4 の中の送信ビット SB(20)がエラー発生し反転し、”0”になっている場合表 3.6 のように、送信ビット 20（SB(20)）の 5 つのパリティ･チェック式「A_1～A_5」は全て“1”となる。

表 3.6　SB(20)のエラー発生

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	0
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

●（パリティ･チェック A_1）

= SB(9) xor SB(12) xor SB(13) xor SB(18) xor SB(20)

= “0” xor “1” xor “1” xor “1” xor “0”

= 1　　となる。

なぜならば、上記 5 ビット中の “ 1 ” の数が奇数である。

「パリティ A_2～A_5」についても、同様に計算すると、

●（パリティ･チェック A_2）

= SB(1) xor SB(11) xor SB(14) xor SB(15) xor SB(20)

= “0” xor “1” xor “1” xor “1” xor “0”

= 1　　（∵上記５ビット中の“1”の数が奇数）

●（パリティ・チェック A_3）

= SB(4) xor SB(6) xor SB(16) xor SB(19) xor SB(20)

= “0” xor “1” xor “1” xor “1” xor “0”

= 1　　（∵上記５ビット中の“1”の数が奇数）

●（パリティ・チェック A_4）

= SB(0) xor SB(5) xor SB(7) xor SB(17) xor SB(20)

= “0” xor “0” xor “0” xor “1” xor “0”

= 1　　（∵上記５ビット中の“1”の数が奇数）

●（パリティ・チェック A_5）

= SB(2) xor SB(3) xor SB(8) xor SB(10) xor SB(20)

= “1” xor “1” xor “0” xor “1” xor “0”

= 1　　（∵上記５ビット中の“1”の数が奇数）

となり、「A_1～A_5」5 種類全てのパリティ・チェックが、排他的論理和（XOR）の結果として、“ 1 ” を出力している。

3.2.4.3　SB(20)以外で一箇所エラーした場合

本節では、例として表 3.4 の送信ビット SB(0)目がエラーし、反転して“0”→“1”になっている場合を用いて示す。

表 3.7　SB(0)にエラー発生

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)	1	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

●(パリティ・チェック A_1)

= SB(9) xor SB(12) xor SB(13) xor SB(18) xor SB(20)

= “0” xor “1” xor “1” xor “1” xor “1”

= “0”

なぜならば、上記 5 ビット中の “ 1 ” の数が奇数である。

●(パリティ・チェック A_2)

= SB(1) xor SB(11) xor SB(14) xor SB(15) xor SB(20)

= “0” xor “1” xor “1” xor “1” xor ”1”

= “0“ （∵上記５ビット中の“1”の数が偶数）

●(パリティ・チェック A_3)

= SB(4) xor SB(6) xor SB(16) xor SB(19) xor SB(20)

= “0” xor “1” xor “1” xor “1” xor ”1”

= “0” （∵上記５ビット中の“1”の数が偶数）

●(パリティ・チェック A_4)

= SB(0) xor SB(5) xor SB(7) xor SB(17) xor SB(20)

= “1” xor “0” xor “0” xor “1” xor “1”

= “1” （∵上記 5 ビット中の“1”の数が奇数）

●(パリティ・チェック A_5)

= SB(2) xor SB(3) xor SB(8) xor SB(10) xor SB(20)

= “1” xor “1” xor “0” xor “1” xor “1”

= “0” （∵上記５ビット中の“1”の数が偶数）

以上の結果より、パリティ・チェック「A_1～A_5」の5ビットの内、反転している SB(0)=”0” を含む、排他的論理和を取っている、「A_4」のみが“1”を出力しているので、エラーが発生していることを検出できる。（ただし、この場合、エラーが「A_4」で起きていると限定することはできない。あくまで、20 ビットの送信ビットの中の「何処か」のビットに、「エラーが生じた」ということのみわかる。具体的に、どの送信ビットにエラーが生じたかを知

る方法は 3.2.4.4 節で示す。）

しかも、SB(20)以外のビットは、SB(20)のようにパリティ・チェック式「A_1～A_5」に、全て“1”が与えられている物は無く、「A_1～A_5」のうち、どれか一つのみが、“1”を出力している。

3.2.4.4　SB(20)のエラー検知と訂正方法

したがって、3.2.4.2 節に示したように、各パリティ・チェック式「A_1～A_5」について、順に対応する送信ビットの排他的論理和の計算で結果の「多数決」をとれば、必然的に、“1”を多数出力している、SB(20)でエラーを発生していることを検知できる。

（∵パリティ・チェック式「A_1～A_5」において、同時にパリティ・ビット“1”が与えられているのは SB(20)だけであるから。）

例えば、多数決の結果「A」が「3」以上の場合にその送信ビットで、エラーが発生したと判断するように設定しているならば、SB（20）の場合、

$$\begin{aligned} 「A」 &= A_1 + A_2 + A_3 + A_4 + A_5 \\ &= “1” + “1” + “1” + “1” + “1” \\ &= “5” \geq “3” \end{aligned}$$

となり、エラーが発生していることが検出できる。

よって、エラー個所が SB(20)と検知できたので、SB(20)を反転させれば、エラーを訂正できる。

3.2.4.5　それ以外のビットで発生したエラーの検知と訂正

表 3.4 の中の送信ビット SB(19)にエラーが起きた場合を例にあげて示す。

表 3.4 における、送信ビット SB(20)を基準に、表 3.8 の様に、全体に左に 1 ビットだけ巡回（シフト）し、左端であふれた“1”を、右端にシフトさせる。

表 3.8　左に 1 ビット巡回した例

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	

以上の様に、シフトしたパリティ・チェックを用いて、3.2.4.2 節のように、排他的論理和をとり、3.2.4.4 節の場合と同様に多数決をとれば、SB(20)の場合と同様に、SB(19)のエラーの検出と訂正ができる。

また、SB(19)の場合と同様に、パリティ・チェック式をシフトすることで、他のビット位置のエラーも訂正できる。

3.3　送信ビットの生成方法

前節までに説明した「パリティ・チェック」や「差集合巡回符号」をもとに、11 ビットの「情報ビット」に対して、10 ビットの「エラー訂正用ビット」を付加した、21 ビットの「送信ビット」を生成するための方法を、概念的に多項式を用いて順に示す。

3.3.1 送信ビットの生成方法

送信ビットの生成方法を、概念的な手順を図 3.11 に示す。

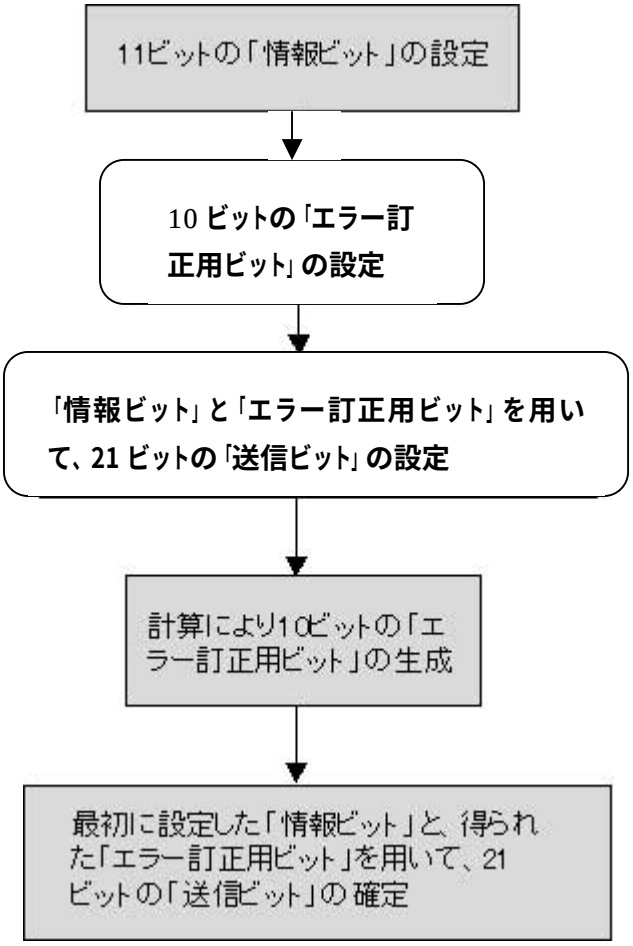

図 3.11　送信ビットの生成方法

11ビットの「情報ビット」は、自分の送信したい 11 ビットのデータである。10ビットの「エラー訂正ビット」は、11ビットの「情報ビット」を用いて、ある一定の計算により、求めたものである。この結果より、任意の値の11ビットの「情報ビット」と、それにより生成さ

れた、10ビットの「エラー訂正用ビット」を組み合わせたものが 21 ビットの「送信ビット」を構成する。

3.3.2 送信ビット生成の計算

送信ビットの生成方法、および、送信回路や受信回路のシドローム計算では多項式を割り算し、余りを求める回路が必要になるので、多項式を用いた計算方法・計算過程を次に示す。

- まず、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、次のような、多項式 U(x)で表す。

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

- 次に、10 ビットのエラー訂正用ビットを $\{r_0,r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_8,r_9\}$ とし、次のような、多項式 R(x)で表す。

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

- この結果、上記の情報ビット 11 ビットにエラー訂正用 10 ビットを付加した、21 ビットの送信ビット SB は表 3.9 のようになる。

表 3.9 送信ビットの設定

	エラー訂正用(10 ビット)										情報ビット(11 ビット)										
情報ビット (11bit)											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 (21bit)	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}

- 表 3.9 より、送信ビット SB(x)を次に表す。

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

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

● よって式(3)より、情報ビット 11 ビットを新たに U'(x)として次のように表す。

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

● この時、11 ビットの情報ビットが与えられているので、10 ビットのエラー訂正用ビット $\{r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}$ を、求めるために次のような計算をする。

$$U'(x) \div G(x) = D(x) \ldots R(x) \qquad \text{となる}$$

ただし、

$$U'(x) = u_0 \cdot x^{10} + u_1 \cdot x^{11} + u_2 \cdot x^{12} + u_3 \cdot x^{13} + u_4 \cdot x^{14} + u_5 \cdot x^{15}$$
$$+ u_6 \cdot x^{16} + u_7 \cdot x^{17} + u_8 \cdot x^{18} + u_9 \cdot x^{19} + u_{10} \cdot x^{20} \qquad \cdots\cdots(5)$$

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

D(x)＝（商）

R(x)＝（余り）（上式(2)） である。

● 実際に上記の計算により、R(x)を求めると

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

$+ (u_0 - u_3 - u_4 - 2\cdot u_7 + u_9 - 2\cdot u_{10})$　　となる。

表3.9の例では、「情報ビット」の値は全て“1”なので、上記の式の$\{u_0 \sim u_{10}\}$の中に、それぞれ“1”の値を入れると、uの個数が偶数の場合は“0”、奇数の場合は“1”となる。（ただし、“1”の個数の偶･奇を求めるだけなので、“-1”も、“1”とみなすことができる。）

$$R(x)= (-1-1+2\cdot1+1-2\cdot1-1)\cdot x^9$$
$$+ (-1-1+2\cdot1+1-2\cdot1-1)\cdot x^8$$
$$+ (-1-1+2\cdot1+1-2\cdot1-1)\cdot x^7$$
$$+ (-1-1+1+1-1+1-2\cdot1+3\cdot1)\cdot x^6$$
$$+ (-1-1+1+1+1-1+1-3\cdot1)\cdot x^5$$
$$+ (-1-1+1+1+1-1+1-3\cdot1+2\cdot1)\cdot x^4$$
$$+ (-1-1+1+1+1+1-2\cdot1-3\cdot1)\cdot x^3$$
$$+ (-1-1+1+1+1+1-2\cdot1-3\cdot1+2\cdot1)\cdot x^2$$
$$+ (1-1-1+2\cdot1+1)\cdot x$$
$$+ (1-1-1-2\cdot1+1-2\cdot1)$$

となり、

$$x^9=0,\ x^8=0,\ x^7=0,\ x^6=1,\ x^5=0,$$
$$x^4=0,\ x^3=1,\ x^2=1,\ x^1=0,\ x^0=0$$

となる。したがってR(x)は、

$$『\ R(x) = 2 \geq 1』$$

となり、「表」の様に、10ビットのエラー訂正用ビット$\{r_0,r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_8,r_9\}$の値が求まる。

以上の操作を同様に行うことによって、その他の任意の11ビットの情報ビットの場合においても、21ビットの送信ビットSBがられる。

3.4　受信ビットのエラー訂正方法

「情報ビット」と、それに付加された「エラー訂正ビット」を送信することによって、どのような仕組みで受信ビットのエラーを訂できるのか本節で説明する。

3.4.1 なぜエラー訂正ビットが必要か？

そもそも、なぜ、エラー訂正ビットが必要なのかというと、前述したように、文字多重放送におけるデータの送受信には、伝送中に何らかの原因により、高確率でエラーが発生する。

このエラーは、その情報ビットから計算された「固有のデータ（エラー訂正用ビット）」を付加しておくことによって、受信側で訂正できる。

例えれば、ある「A」という人物がいたとする。この人物「A」は、現在地より、ある目的地に移動しようとするが、道中何らかの理由により、記憶があいまいになり、突如何処に行こうとしていたのか忘れてしまうとする。そこで、移動前に目的地を記したメモを持っていたことに気づく。ただし、そのメモは、移動中に汚れてしまったらしく、移動開始時とは異なったものになっている。しかし、それを見たことにより、再び、きちんと目的地を記憶の中から思い出し、目的地に既に到着したことに気づく。

言うなれば、この「目的地を記したメモ」が「エラー訂正用ビット」であり、「記憶」が「情報ビット」である。目的地に向かう途中に、「記憶」があいまいになり、「目的地を記したメモ」も移動開始時とは異なったものになっていたとしても、それを見ることにより、「記憶」が再び復元され、そこが目的地だとわかる。

この一連の流れは、少し的がズレてはいるが、「情報ビット」と固有の「エラー訂正用ビット」を付加した、「送信ビット」を送信してから、受信し、エラーの訂正が行われるまでの工程と同等と考えて良い。

よって、ある「情報データ（情報ビット）」を送る際には、その情報ビット固有の、「エラー訂正用ビット」を付加して送信する必要がある。

3.4.2　なぜエラーが訂正できるのか

「送信ビット」が情報ビットとエラー訂正用ビットから構成されることから、送信側から情報ビットとエラー訂正用ビットを送信することで、受信側で伝送路で発生したエラーを訂正できる理由は、3.1 節の「パリティ･チェック」によるエラーの訂正方法と、3.2 節の「差集合巡回符号」によるエラーの訂正方法を参照にしていただきたい。

ただし、パリティ・チェックの場合と差集合巡回符号の場合、ある「情報ビット」に対し、固有な計算により生成される「ビット」（パリティ・ビットやエラー訂正用ビット等）を付加することにより、付加しない場合よりも、受信側でエラーを検出し、訂正する効率、確実性の向上に繋がることになることは言うまでもない。

以上により、本章の、テレビの「文字多重放送」における情報データの送受信中に生じるエラーに対して、一般的に用いられる訂正方法であり、かつ今回の研究テーマである「差集合巡回符号　エラー訂正回路設計」に用いる方式「パリティ・チェック」と「差集合巡回符号」についての説明を終える。

第 4 章

差集合巡回符号エラー訂正回路の設計

本章では、今回の卒業研究テーマである、テレビの「差集合巡回符号エラー訂正回路」のブロック図によるシステムの設計に始まり、システムを構成する各回路（エラー訂正に関わる部分として「送信機」、「受信機」を、それらのエラー訂正能力を評価するための「エラーカウンタ」）の VHDL や回路図による設計・シミュレーションを経て、最終的に全回路を組み合わせたシステムのシミュレーションに至るまでの手順ついて順に説明する。

さらに、今回は設計した回路を論理合成ツール「LeonardoSpectrum LS2001_1a_31」を用いて合成し、回路のクリティカル・パスの速度と合成後の回路規模について求めた。ただし、今回の設計の主旨が差集合巡回符号エラー訂正回路の設計であり、あくまで「エラーカウンタ」は設計した回路のエラー訂正能力の評価に使用するだけが目的なので、エラーカウンタについては、VHDL による設計および全回路のシミュレーションのみ行い、論理合成や回路規模については求めていない。

なお、今回差集合巡回符号エラー訂正回路のシステム設計をしていくうえで、用いる信号名や情報データのビット長などは、今回応募した「Design Wave 設計コンテスト 2002」の仕様にて設定されたものである。したがって、信号は各回路ごとに生成したので、回路が異なっても信号名が等しい場合がある。また、実際に一般にテレビの文字多重放送にて使われているものと異なることを予め了承頂きたい。

4.1　エラー訂正回路の全体的なシステム設計

4.1.1　システムの構成説明

「エラー訂正回路」の全体的なシステムは、ブロック図 4.1 に示すように、大きく分けて「送信機 Transmitter」と「受信機 RECEIVER」と「エラーカウンタ ERRORCNT」の 3 つに分類される。送信機と受信機は「差集合巡回符号エラー訂正回路」において核となる部分であり、かつ、実際にデータの送受信を行い、エラー訂正を行う回路である。エラーカウンタはそれらのエラー訂正能力を評価するための回路である。

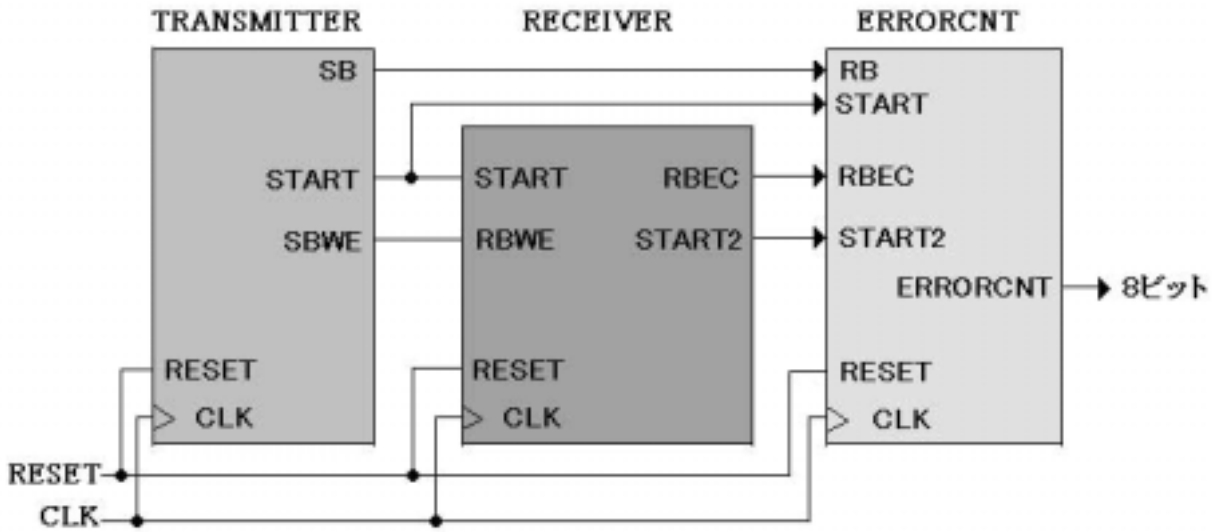

図４.１　エラー訂正回路のシステムブロック図

4.1.2　システムの動作説明

エラー訂正回路の全体的なシステムの動作説明を図 4.1 にしたがって行う。

（１）送信機は、送信信号「SB」とエラー付き送信信号「SBWE」とスタート用信号「START」生成し、受信機とエラーカウンタに送る。

（２）受信機は、送信機より受け取った、信号 START により、エラー付き受信信号「RBWE」のエラー訂正をスタートする。

（３）（２）に続いて、受信機は、エラー訂正の完了した信号を順に「RBEC」

としてエラーカウンタに送る。

（4）（3）と同時に、受信機は RBEC の先頭部分を表すための信号「START2」を生成し、エラーカウンタに送る。

（5）エラーカウンタは、START と START2 により、送信機から受信した信号「RB」と受信機より受信した信号 RBEC を比較して、エラーの数のカウントをスタートし、結果を 8 ビット信号で出力する。

なお、各回路の具体的な構成や機能、動作などは、本章の該当する各節にて詳しく説明する。

4.2 「TRANSMITTER（送信機）」の設計

本節では、エラー訂正の一連の動作において、データを送信する部分である「TRANSMITTER（送信機）」の設計について説明する。

4.2.1 TRANSMITTER の役割

本節で説明する「TRANSMITTER（送信機）」は、その名前からも推測できるように、差集合巡回符号エラー訂正回路において、送信したい情報データを生成し、受信側へ向けて送信する役割を果たしている。この TRANSMITTER は、図 2.1.では「TV 主整室」や「多重化装置」および「TV 送信機」等が含まれる送信側にあたる。

ただし、今回の研究では、この TRANSMITTER は、11 ビットの「情報ビット」を生成した後、3.3 節で紹介した方法により、その 11 ビット信号から、10 ビットの「エラー訂正用ビット」を生成し、更にそれらの情報ビットとエラー訂正用ビットを合わせて、21 ビットの「送信ビット」を生成するよう設計されている。

なお、この TRANSMITTER は、2.1.5 節で説明したように実際の送信中には、データにエラーが生じることを想定して、先ほどの送信ビットに、強制的にある一定の確率でエラーが生じるように設計されている。

4.2.2 回路ブロックの構成説明

図４.2 に、TRANSMITTER の路ブロックを示す。また、ブロック図４.2 にしたがって、主要なブロックの機能について説明をする。なお、回路名と番号はブロック図に対応している。

（１）情報ビット生成回路　①：
　　フェイズが“ 20 ”の場合のみ、11 ビットの情報ビットを生成する。

（２）エラー信号生成回路　②：
　　エラー信号生成回路用内部カウンタが“ 0 ”の場合のみ、‘ 1 ’を生成し信号「errsig」として出力する。

（３）リニア・フィードバック・シフトレジスタ　③：

フェイズが“22”からフェイズが“32”の間のみ、11 ビットの情報ビットを用いて 10 ビットのエラー訂正用ビットを生成し出力する。

（4）START 信号発生回路　④：

フェイズが“21”の時のみ、「RECEIVER」が動作開始をするための信号‘1’を生成し出力する。

（5）42 フェイズ生成回路　⑤：

リセットが完了し“0”なり、クロックが‘1’になったとき、フェイズ“0”からフェイズ“41”までカウントアップを開始する。

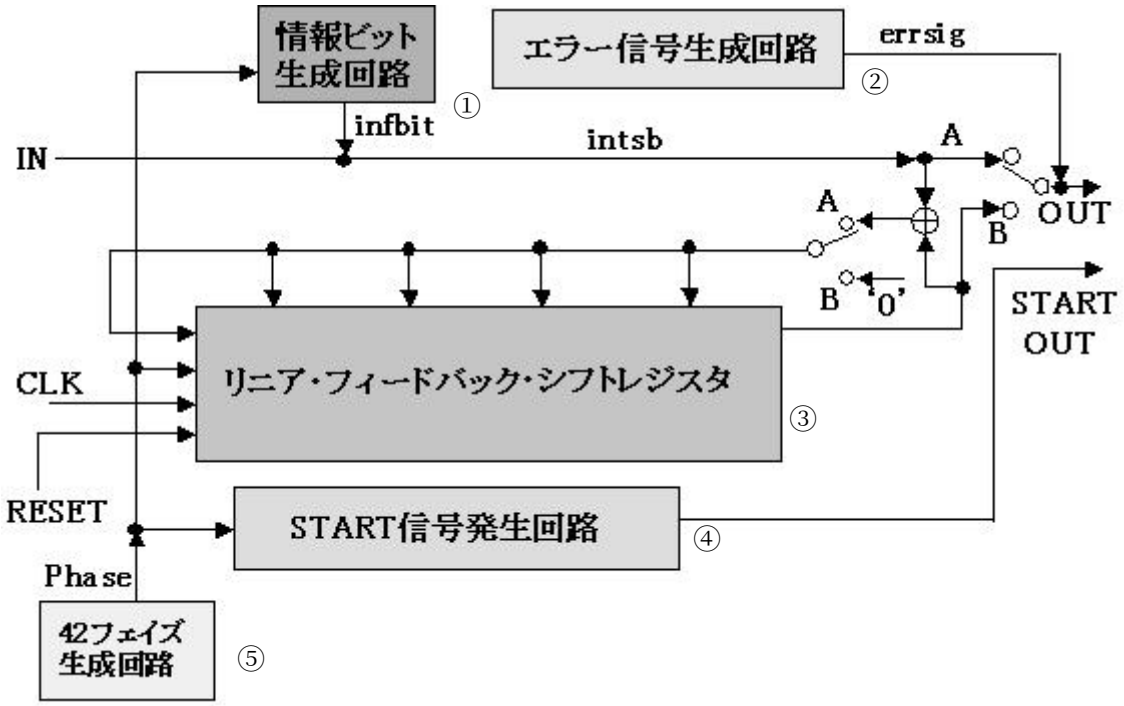

図 4.2　TRANSMITTER のブロック図

4.2.3　回路ブロックの動作説明

（1）42 フィズ生成回路⑤は、リセットが完了し“ 0 ”なり、クロックが‘ 1 ’になったとき、フェイズ“ 0 ”からフェイズ“ 41 ”までカウントアップし、「Phase」を生成する。

（2）情報ビット生成回路①は、フェイズ“ 20 ”になると、11 ビットの情報ビット「11111111111」を生成し出力する。その後フェイズが“ 20 ”になるたびに、「11111111110」から「00000000000」に至るまで、繰り返し生成し信号「infbit」として出力する。

（3）入力端子「IN」は、フェイズ“ 0 ”からフェイズ“ 20 ”の間、21 ビットの一定の同期信号を内部信号「intsb」に入力する。また、フェイズ“ 21 ”からフェイズ“ 31 ”の間は、intsb に infbit が代入される。

（4）リニア・フィードバック・シフトレジスタ③は、フェイズが“ 22 ”からフェイズが“ 32 ”の間のみ、infbit を用いて、10 ビットのエラー訂正用ビット「parity」を生成し出力する。

（5）エラー信号生成回路②は、エラー信号生成回路用内部カウンタが“ 0 ”の場合のみ、‘ 1 ’を生成し「errsig」として出力する。

（6）フェイズ“ 33 ”もしくはフェイズ“ 0 ”の時、「parity(9)」が信号「intsb2」として出力される。

（7）信号 intsb2 は、送信ビット「SB」としてそのまま送信される。それと同時に、errsig と排他的論理和をとり、エラー付き送信ビット「SBWE」として出力される。

（8）START 信号発生回路④は、フェイズが“ 21 ”のとき、信号「START」を出力する。

4.2.4　回路作成ソフトを用いた設計

本節では、ALTERA 社製「MAX+plus Ⅱ10.1 BASELINE」を用いて、回路作成し、設計した回路を図 4.3 に、またシミュレーションした結果を図 4.4 に示す。[2],[3],[4]

なお、回路のシミュレーションは、TRANSMITTER の動作イメージを掴むことと、次の節での VHDL による設計の正確性を計るための目安としての設計が目的であるので、図 4.2 中の幾つかの回路については省略し、簡略化した。

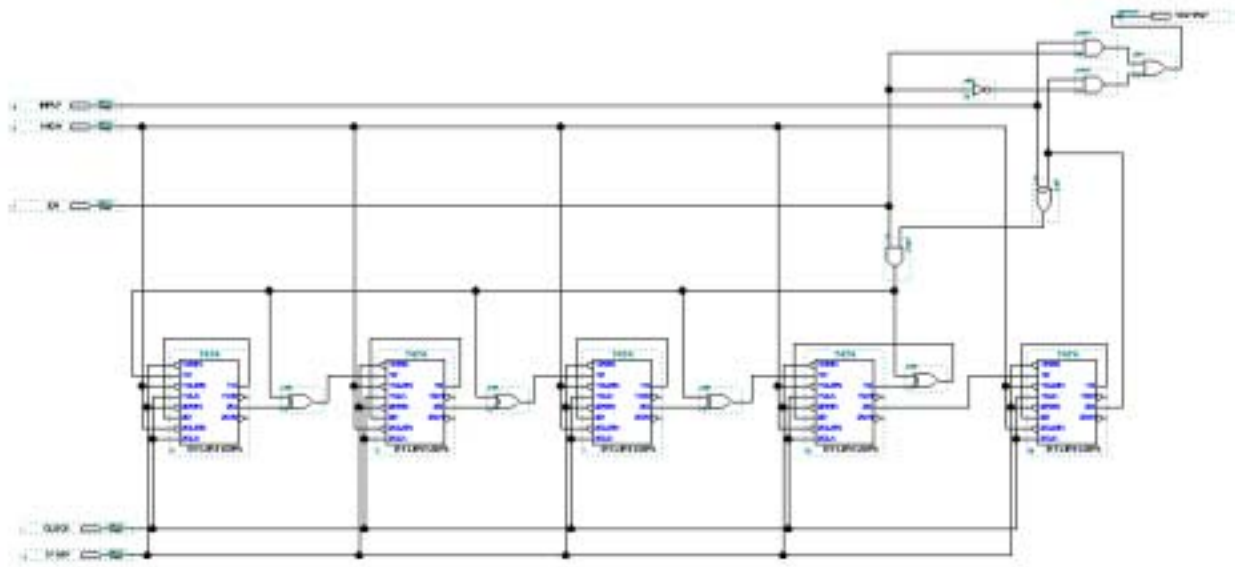

図 4.3　TRANSMITTER 回路図

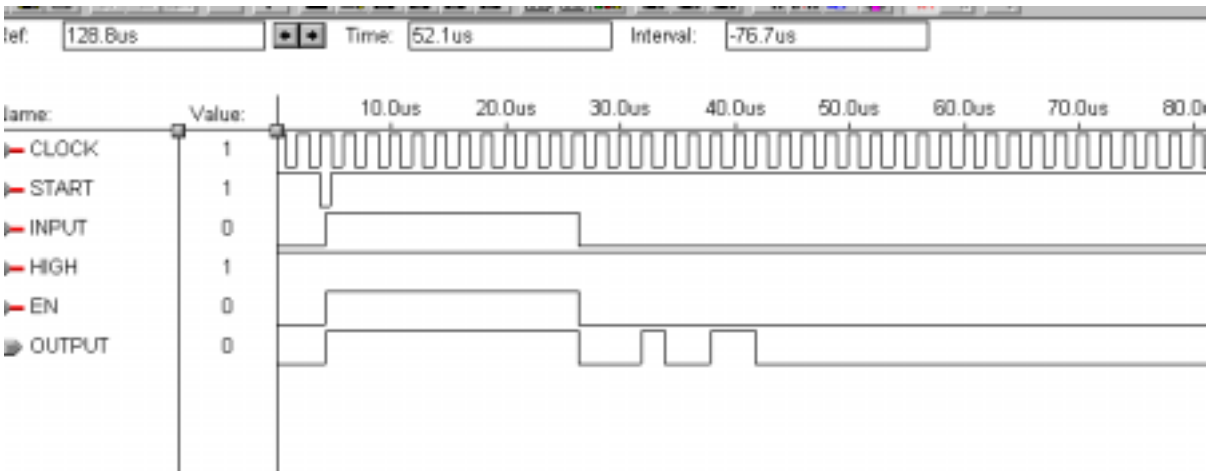

図 4.4　シミュレーション波形

4.2.5　VHDL ソフトを用いた設計

本節では、Xilinx 社製「WebPACK Project Navigator」を用いて、VHDL による回路設計を行い、同社製「Xilinx WebPACK 4.1 ModelSim XE Starter」を用いてシミュレーションした結果を表す。ソースコードをリスト 4.1 にテストベンチをリスト 4.2 に示す。また、シミュレーション結果を図 4.4 に示す。[6],[7],[8]

リスト 4.1　TRANSMITTER ソースコード 1/6

```
library IEEE;
use IEEE.STD_LOGIC_1164.all, IEEE.STD_LOGIC_unsigned.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 TRANSMITTER;

architecture RTL of TRANSMITTER is
    signal phase : std_logic_vector   (5 downto 0);          -- 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 infbit : std_logic_vector   (10 downto 0);          -- 11bits information bits
    signal intstart : std_logic;                                 -- internal start signal
    signal intsb,intsb2 : std_logic;                                 -- internal sb signal
    signal parity : std_logic_vector   (9 downto 0);                 -- 10 parity bits
    signal errsig : std_logic;                                   -- error signal
    signal errcnt : std_logic_vector   (9 downto 0);                 -- error counter
begin

-----------------------------------------------
-- 42 phase (0 to 41 counter) generation unit
-----------------------------------------------
  PHASE_CNT: process(CLK,RESET)
```

リスト 4.1　TRANSMITTER ソースコード 2/6

```
 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 <= "00000000000";
        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 4.1　TRANSMITTER ソースコード 3/6

```
        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
```

リスト 4.1　TRANSMITTER ソースコード 4/6

```
            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);
```

リスト 4.1　TRANSMITTER ソースコード 5/6

```
                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
           else errsig <= '0';
         end if;
        end if;
      end process ERROR_GEN;
-----------------------------------------------
-- output signals
-----------------------------------------------
```

リスト 4.1　TRANSMITTER ソースコード 6/6

```
   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
-------------------------------------------------
           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 RTL;
```

リスト 4.2　TRANSMITTER テストベンチ 1/2

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

	COMPONENT transmitter
	PORT(
		RESET : IN std_logic;
		CLK : IN std_logic;
		SBWE : OUT std_logic;
		SB : OUT std_logic;
		START : OUT std_logic
		);
	END COMPONENT;

	SIGNAL cycle : integer := 0;
	SIGNAL SBWE :   std_logic;
	SIGNAL SB :   std_logic;
	SIGNAL START :   std_logic;
	SIGNAL RESET :   std_logic:= '1';
	SIGNAL CLK :   std_logic:= '0';

BEGIN

	uut: transmitter PORT MAP(
		SBWE => SBWE,
		SB => SB,
		START => START,
		RESET => RESET,
		CLK => CLK
	);
```

リスト 4.2　TRANSMITTER テストベンチ 2/2

```
process
    begin
                if (cycle < 300)then
                        cycle <= cycle +1;
                        wait for 10 ns;
                        CLK <= not CLK;
              else wait;
       end if;
    end process;

   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;
   end process;

end architecture behavior;
```

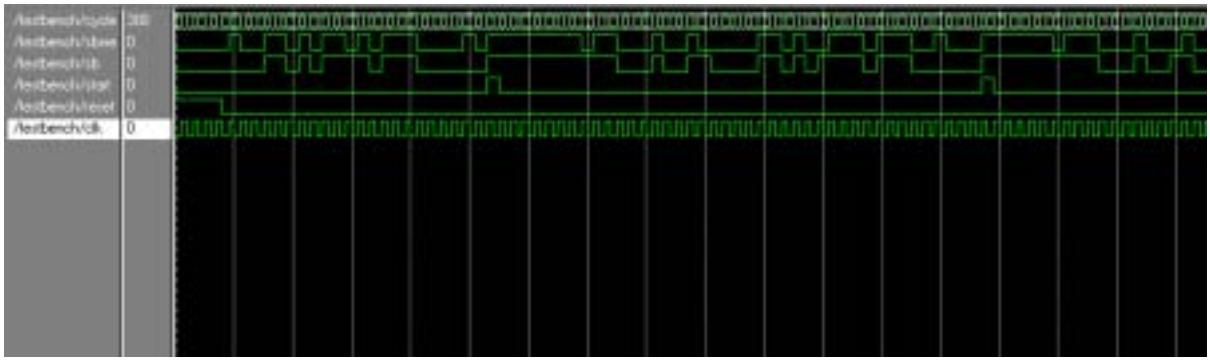

図 4.4　TRANSMITTER シミュレーション波形

4.2.6　クリティカル・パスの速度、論理合成後の回路規模

本節では、論理合成ツール「LeonardoSpectrum LS2001_1a_31」を用いて実際に論理合成を行い、クリティカルパスの速度および、論理合成後の回路規模を求めた。

4.2.6.1　クリティカル・パスの速度

論理合成されたTRANSMITTERのクリティカル･パスの速度を求めるために、クリティカルパス遅延の基準として、コンテストの仕様で設定された「50 入力のXOR」を用いた。

50 入力のXORを実際に論理合成するとクリティカル・パス遅延は、使用した「LeonardoSpectrum LS2001_1a_31」では､「2.99」である（リスト 4.3 参照)。この 50 入力の XOR を論理合成すると 6 段で合成された。したがって一段あたりの遅延を求めると「2.99／6　　0.49」となり、１段あたりのクリティカル・パス遅延時間として「0.49」を基準にする。

設計した TRANSMITTER を、同論理合成ツールにて論理合成すると、遅延が「6.08」である（リスト 4.4 参照)。したがってクリティカル・パスを求めると「6.08／0.49　　12.4」 となるので､クリティカル･パスの速度は「12.4 UNIT 遅延」となった。

4.2.6.2　論理合成後の回路規模

論理合成後の回路規模はリスト 4.3 に示すように､「ゲート数 981」となった。また、論理合成後の回路図を図 4.5 に示す。

リスト 4.3　50 入力 XOR の論理合成結果

```
Critical path #1, (unconstrained path)
NAME                                    GATE                    ARRIVAL
LOAD
---------------------------------------------------------------------------
A(36)/                                  0.00   0.00 up          0.08
ix69/X                          XR2T0   0.23   0.23 up          0.08
ix255/X                         XN3R0   0.47   0.70 up          0.14
ix75/X                          XN2R0   0.48   1.22 dn          0.08
ix91/X                          XN3R0   0.33   1.61 up          0.09
ix95/X                          XR3T0   0.51   2.12 up          0.21
ix97/X                          XR3T0   0.87   2.99 up          0.04
Y/                                      0.00   2.99 up          0.00
data arrival time                               2.99
data required time                              not specified
---------------------------------------------------------------------------
data required time                              not specified
data arrival time                               2.99
                                                ---------
                                                unconstrained path
****************************************************************
Cell: xor50inputs       View: RTL       Library: work
****************************************************************
              Cell          Library   References      Total Area
             XN2R0           scl05u      3 x       5      15 gates
             XN3R0           scl05u      9 x       7      61 gates
             XR2T0           scl05u     12 x       5      59 gates
             XR3T0           scl05u      8 x       7      54 gates
  Number of ports :                     51
  Number of nets :                      82
  Number of instances :                 32
  Number of references to this view :   0
  Total accumulated area :
  Number of gates :                     189
```

リスト 4.4　TRANSMITTER 論理合成結果 1/3

```
                        Critical Path Report
Critical path #1, (unconstrained path)
NAME                                          GATE                  ARRIVAL
LOAD
-------------------------------------------------------------------------------
clock information not specified
delay thru clock network                              0.00 (ideal)


infbit(1)/Q                    FD1I0        0.00    0.43 dn              0.19
ix856/X                        NR2R0        0.51    0.95 up              0.19
ix195/X                        ND2N0        0.55    1.49 dn              0.19
ix877/X                        NR2R0        0.56    2.06 up              0.19
ix219/X                        ND2N0        0.55    2.61 dn              0.19
ix874/X                        NR2R0        0.56    3.17 up              0.19
ix243/X                        ND2N0        0.55    3.72 dn              0.19
ix907/X                        NR2R0        0.56    4.28 up              0.19
ix267/X                        ND2N0        0.55    4.83 dn              0.19
ix904/X                        NR2R0        0.49    5.33 up              0.14
ix283/X                        XR2T0        0.40    6.08 dn              0.06
infbit(10)/D                   FD1I0        0.00    6.08 dn              0.00
data arrival time                                     6.08
data required time                                    not specified
-------------------------------------------------------------------------------
data required time                                    not specified
data arrival time                                     6.08
                                                      ----------
                                                      unconstrained path
-------------------------------------------------------------------------------
**************************************************
Cell: TRANSMITTER       View: RTL       Library: work
**************************************************
                Cell            Library    References        Total Area

               AN2T0             scl05u      15 x          5        71 gates
```

リスト 4.4　TRANSMITTER 論理合成結果 2/3

AN4T0	scl05u	1 x	8	8 gates
AN5T0	scl05u	1 x	9	9 gates
AO1I1	scl05u	1 x	6	6 gates
AO2A0	scl05u	6 x	8	47 gates
AO2I0	scl05u	1 x	8	8 gates
AO3I0	scl05u	1 x	8	8 gates
AOA4I0	scl05u	1 x	8	8 gates
FD1B0	scl05u	27 x	9	232 gates
FD1B1	scl05u	3 x	9	26 gates
FD1B2	scl05u	1 x	10	10 gates
FD1I0	scl05u	11 x	11	117 gates
HA1A0	scl05u	12 x	7	79 gates
IV1N0	scl05u	6 x	3	19 gates
IV1NP	scl05u	2 x	4	8 gates
MX2L0	scl05u	1 x	6	6 gates
MX2T0	scl05u	2 x	6	13 gates
ND2N0	scl05u	4 x	5	18 gates
ND3N0	scl05u	2 x	6	12 gates
ND4N1	scl05u	1 x	8	8 gates
NR2R0	scl05u	12 x	5	54 gates
NR2R1	scl05u	1 x	5	5 gates
NR3R0	scl05u	1 x	6	6 gates
NR4R0	scl05u	2 x	8	15 gates
NR4R1	scl05u	1 x	8	8 gates
NR4R2	scl05u	1 x	8	8 gates
NR5R2	scl05u	1 x	10	10 gates
OAI2N0	scl05u	1 x	8	8 gates
OAI3N0	scl05u	1 x	8	8 gates
OAI3R0	scl05u	1 x	8	8 gates
OAI4R0	scl05u	1 x	14	14 gates
OAOI0	scl05u	2 x	8	15 gates
OR2T0	scl05u	2 x	5	9 gates
XN2R0	scl05u	8 x	5	39 gates
XR2T0	scl05u	7 x	5	34 gates
XR3T0	scl05u	4 x	7	27 gates

リスト 4.4　TRANSMITTER 論理合成結果 3/3

```
Number of ports :                          5
  Number of nets :                         171
  Number of instances :                    145
  Number of references to this view :      0
Total accumulated area :
  Number of gates :                        981
```

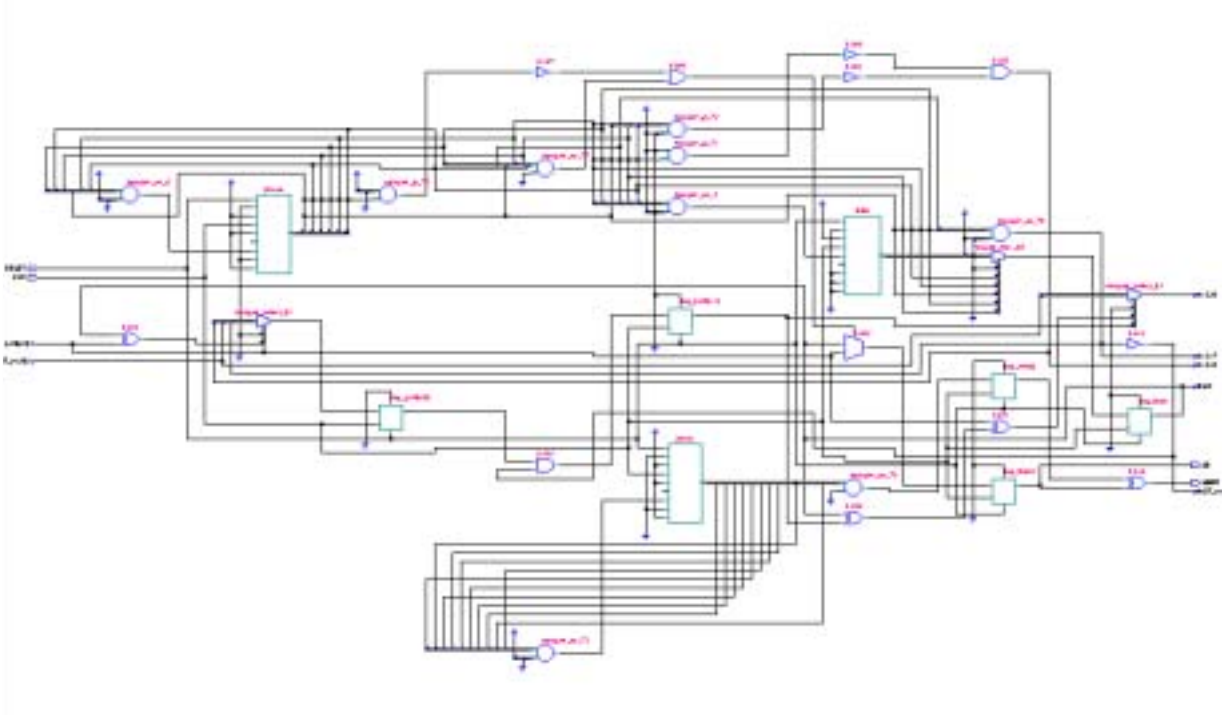

図 4.5　TRNSMITTER 論理合成回路図

4.3 「RCEIVER（受信機）」の設計

本節では、エラー訂正の一連の動作において、最も重要な部分であり、送信機よりデータを受信し受信ビットのエラーを検出・訂正する部分である RECEIVER の設計について説明する。

4.3.1 RCEIVER の役割

本節で説明する「RECEIVER（受信機）」は、最も重要な部分であり、その名前からも推測できるように、差集合巡回符号エラー訂正回路において、送信機より送られたエラー付きの受信ビットのエラーを検出・訂正する役割を果たしている。この RECEIVER は、図 2.1.では「映像復調部」や「信号処理部」および「TV/文字切り換え器」から成る「デコーダ」等が含まれる受信側にあたる。

ただし、今回の研究では、この RECEIVER は、21 ビットの「受信ビット」を受信した後、3.2.4 節で紹介した方法により、その 21 ビット信号から、５種類の「パリティ・チェック」を生成しする。更にそのパリティ・チェックから「エラー訂正信号」を生成し、エラー訂正を行うよう設計されている。

4.3.2 回路ブロックの構成説明

主要なブロックの機能について、図４.6 を用いて説明する。なお、回路名と番号はブロック図に対応している。

（１）入力信号制御回路　①：
「Ensig」が‘１’の間のみ、RBWE を出力するように制限する。

（２）４２フェイズ生成回路　⑦：
「START」に‘１’が立つと、42 フェイズカウントダウンを始める。

（３）EN 信号生成回路　⑥：
フェイズ“41”からフェイズ“２２”の間のみ、Ensig を‘１’にする。

（４）START2 信号発生回路　⑧：
エラー訂正された「RBEC」が出力される時、その先頭ビットを表すために「START2」を生成する。

（5）エラー検出回路③：

「パリティ・チェック生成回路（リニア・フィードバック・シフトレジスタ,パリティ生成回路)」④および「多数決回路」⑤から構成され、フェイズ“ 41 ”からフェイズ“ 0 ”の間に、21 ビットの「受信ビット」を用いて、エラー訂正信号を生成する。

（6）21 ビットシフトレジスタ②：

21 ビットの受信ビットを順に格納し、フェイズ“ 20 ”からフェイズ“ 0 ”の間に 1 ビットずつ出力する。

なお、ブロック図中の「IN」には TRANSMITTER から送られてきた信号 SBWE（= RBWE）が、「CLK」には「CLOCK」信号が「RESET」には「RESET」信号が入力されている。

また、「OUT」からは RECEIVER 回路内でエラー訂正された RBWE 信号が、「START2 OUT」からは新たに生成された START2 信号が、出力されている。

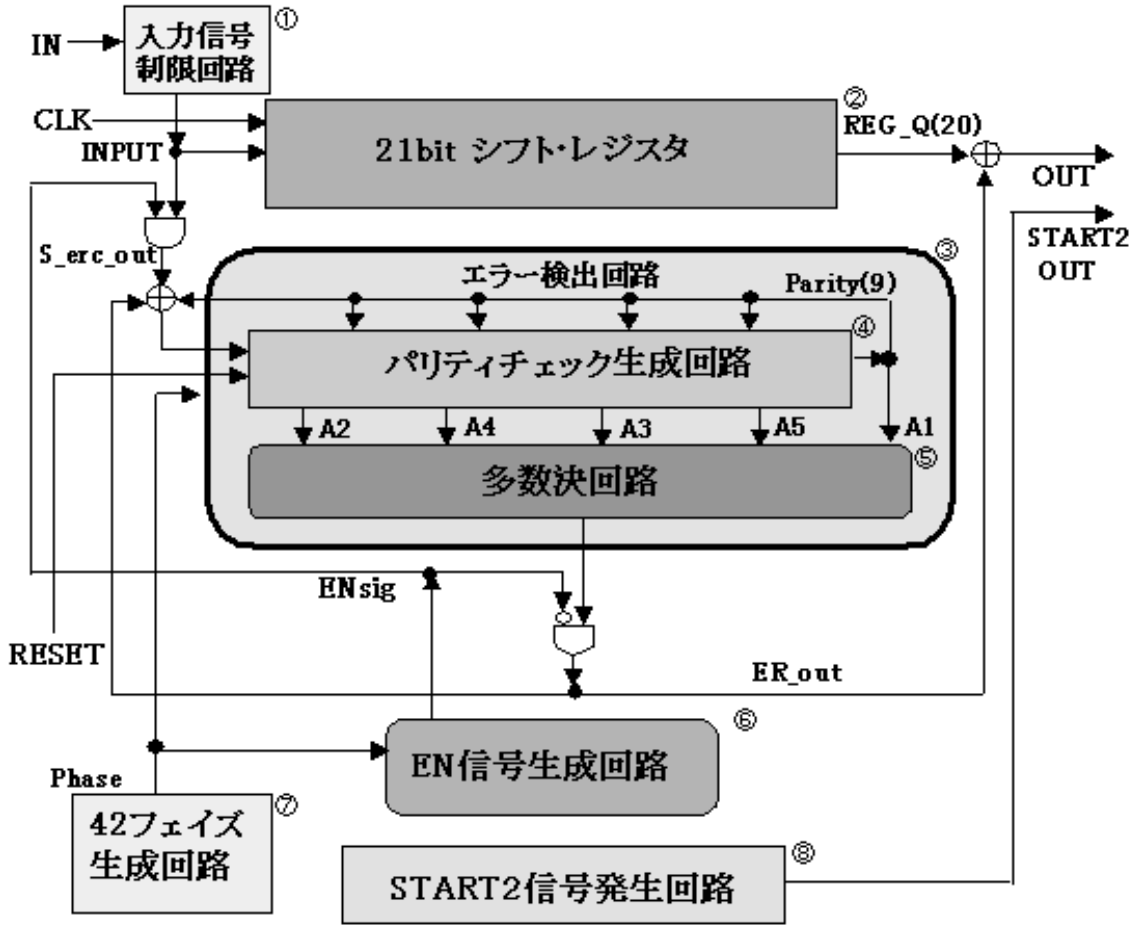

図 4.6　RECEIVER の回路ブロック

4.3.3 回路ブロックの動作説明

設計した回路の動作説明を図 4.6 に従って行う。続いて図 4.7 に動作を示す。

（1）42 フェイズ生成回路④は、フェイズ“ 41 ”からフェイズ“ 0 ”までカウントし、「Phase」を生成する。

（2）EN 信号生成回路⑤は、「ENsig」を、フェイズ“ 42 ”からフェイズ“ 22 ”の間‘ 1 ’にする。

（3）（2）で生成された ENsig が‘ １ ’の間、入力信号制御回路①は受信信号「RBWE」を入力し、同期信号を除いた「21 ビット」のみを「INPUT」として出力する。

（4）（3）で出力された信号 INPUT は、ENsig と論理積がとられ、その後「Parity(9)」と「ER_out」との排他的論理和がとられて、「エラー検出回路」③の入力「S_erc_out」となる。

（5）S_erc_out を入力して、エラー検出回路③は、フェイズが“ 41 ”からフェイズ“ 0 ”まで、の間、「パリティ・チェック」を生成するための計算を行う。

（6）（5）と同時に、エラー検出回路③内の多数決回路⑤は、パリティ・チェック結果の多数決「$A_1+A_2+A_3+A_4+A_5 \geq 3$」をとり、エラー訂正信号「P_out」を出力する。

（7）ENsig が‘ 0 ’の間、（6）で生成した P_out から、「ER_out」を作る。

（8）（7）と同時に、21 ビットシフトレジスタ②は、入力された INPUT をシフトして、「REG_Q(20)」として順に出力する。

（9）信号 ER_out と信号 REG_Q(20)の排他的論理和をとって、エラーを訂正した、出力信号 RBEC を出力する。

（10）フェイズ“ 22 ”のとき、START2 信号発生回路⑥は「START2」を出力する。

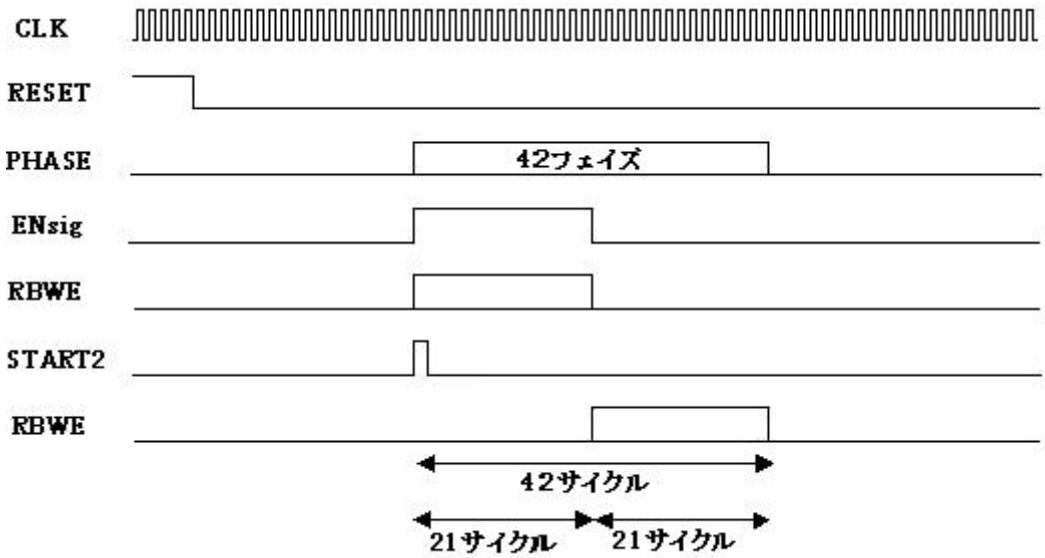
CLK
RESET
PHASE
42フェイズ
ENsig
RBWE
START2
RBWE
42サイクル
21サイクル
21サイクル

図 4.7　動作波形

4.3.4　回路作成ソフトを用いた設計

本節では、TRANSMITTER の場合と同様、ALTERA 社製「MAX+plus Ⅱ 10.1 BASELINE」を用いて、回路作成した。設計した回路を図 4.8 に、またシミュレーション結果を図 4.9 に示す。

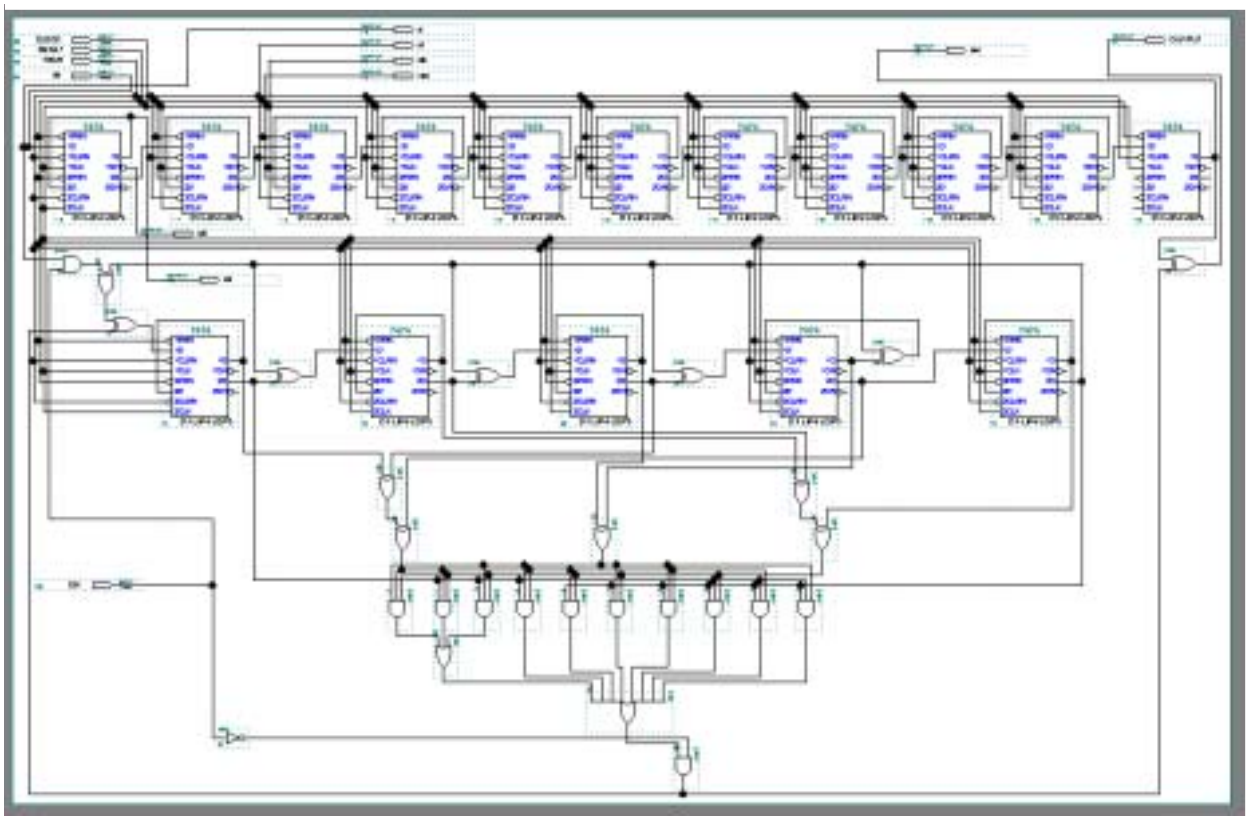

図 4.8　RECEIVER のブロック図

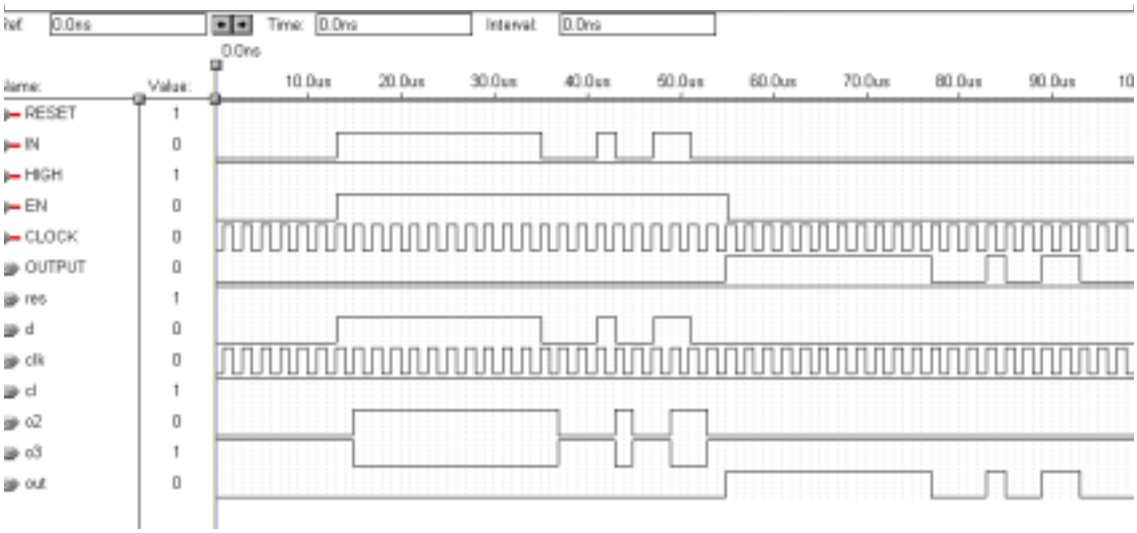

図 4.9　シミュレーション波形

4.3.5　VHDL ソフトを用いた設計

本節では、Xilinx 社製「WebPACK Project Navigator」を用いて、VHDL による回路設計を行い、同社製「Xilinx WebPACK 4.1 ModelSim XE Starter」を用いてシミュレーションした結果を表す。ソースコードをリスト 4.5 に、テストベンチをリスト 4.6 に示す。また、シミュレーション結果を図 4.10 に示す。

リスト 4.5　RECEIVER のソースコード 1/5

```
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 RECEIVER;

architecture RTL of RECEIVER is
      signal INPUT : std_logic;                          -- RBWE input to the receiver
      signal ENsig : std_logic;                          -- ENABLE signal
      signal phase : unsigned (5 downto 0);              -- phase signals
      signal intstart : std_logic;                       -- internal start signal
      signal parity : unsigned (9 downto 0);             -- 10 parity bits
      signal REG_Q : unsigned (20 downto 0 );            -- 21bits sifht register signals
      signal P_out : std_logic;                          -- output signal of majority circuit
      signal ER_out : std_logic;                         -- Error correction signal
      signal S_erc_out : std_logic;                      -- Input signal to the error check circuit
      signal A1,A2,A3,A4,A5 : std_logic;                 -- parity check signals
 begin
```

リスト 4.5　RECEIVER のソースコード 2/5

```
-------------------------------------------------------------------
-- 42 phase (41 downto 0 counter) generation unit
-------------------------------------------------------------------
   PHASE_CNT1: process(CLK,RESET)
    begin
            if (RESET='1') then
            phase <= "000000";
      elsif rising_edge(CLK) then
                    if (START='1')then
                        phase <= "101001";
         elsif (phase= "000000")then
                        phase <= "000000";
                    else
                        phase <= phase -1;
         end if;
      end if;
    end process PHASE_CNT1;
-------------------------------------------------------------------
-- ENABLE signal1 generation
-------------------------------------------------------------------
   EN1_GEN: process (START,phase)
    begin
     if (phase>="010110" and phase<="101001")then
                    ENsig <='1';
           else
                    ENsig <='0';
           end if;
    end process EN1_GEN;
-------------------------------------------------------------------
-- Controlled RBWE siganl
-------------------------------------------------------------------
   RB_n: process(CLK,ENsig)
    begin
        if(ENsig='1')then
            INPUT <= RBWE;
```

リスト 4.5　RECEIVER のソースコード 3/5

```
          else
             INPUT <= '0';
             end if;
  end process RB_n;
-------------------------------------------------------------------
--21bit SHIFT_REGISTER
-------------------------------------------------------------------
  SFIT_R21:      process(CLK, RESET)
   begin
     if ( RESET = '1' ) then
             REG_Q <= ( others => '0' );
      elsif rising_edge(CLK) then
              REG_Q(0) <= INPUT;
             REG_Q (20 downto 1) <= REG_Q (19 downto 0);
     end if;
         end process;
-------------------------------------------------------------------
-- Internal start2 (intstart) generation
-------------------------------------------------------------------
  START_GEN: process(CLK, RESET)
    begin
      if (RESET='1') then
              intstart <= '0';
       elsif rising_edge(CLK) then
       if (phase="010111") then
              intstart <= '1';
       else
               intstart <= '0';
       end if;
     end if;
   end process START_GEN;
-------------------------------------------------------------------
-- Input to the error_checker
-------------------------------------------------------------------
  S_erc_out <= (INPUT and ENsig) xor parity(9) xor ER_out;
```

リスト 4.5　RECEIVER のソースコード 4/5

```
-------------------------------------------------------------------
-- Parity_calculation
-------------------------------------------------------------------
   PARITY_CAL: process(CLK,RESET,phase)
     begin
        if (RESET='1') then
              parity <= "0000000000";
        elsif rising_edge(CLK) then
          if (phase>="000000" and phase<="101001")then
                    parity(9) <= parity(8);
                    parity(8) <= parity(7);
                    parity(7) <= parity(6) xor parity(9);
                    parity(6) <= parity(5) xor parity(9);
                    parity(5) <= parity(4);
                    parity(4) <= parity(3) xor parity(9);
                    parity(3) <= parity(2);
                    parity(2) <= parity(1) xor parity(9);
                    parity(1) <= parity(0);
                    parity(0) <= S_erc_out;
          end if;
        end if;
    end process PARITY_CAL;
-------------------------------------------------------------------
-- Parity input to the parity_checker
-------------------------------------------------------------------
              A1 <= parity(9);
              A2 <= parity(1);
              A3 <= parity(4) xor parity(6);
              A4 <= parity(0) xor parity(5) xor parity(7);
              A5 <= parity(2) xor parity(3) xor parity(8);
-------------------------------------------------------------------
-- Output from the majority circuit
-------------------------------------------------------------------
   PROCESS (A1,A2,A3,A4,A5)
          BEGIN
```

リスト 4.5　RECEIVER のソースコード 5/5

```
                    if (A1='1' and A2='1' and A3='1')then
                                P_out <= '1';
                     elsif (A4='1' and A5='1')then
                                P_out <= A1 or A2 or A3;
                     elsif (A1='1' and A2='1') then
                                P_out <= A4 or A5;
                     elsif (A3='1' and A5='1')then
                                P_out <= A1 or A2;
                     elsif (A3='1' and A4='1')then
                                P_out <= A1 or A2;
                     else
                                P_out <= '0';
        end if;
    end process;
-------------------------------------------------------------------
-- Output from signal of ERROR checker
-------------------------------------------------------------------
    ER_out <= P_out and not ENsig;
-------------------------------------------------------------------
-- Output signal
-------------------------------------------------------------------
   RBEC <= (REG_Q(20) xor ER_out);
-------------------------------------------------------------------
--Startout2
-------------------------------------------------------------------
          STARTOUT2: process (CLK, RESET)
      begin
          if (RESET='1') then
            START2 <= '0';
          elsif rising_edge(CLK) then
            START2 <= intstart;
        end if;
    end process STARTOUT2;

end RTL;
```

リスト 4.6　RECEIVER テストベンチ 1/5

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

    COMPONENT receiver
        PORT(
                START : IN std_logic;
                RBWE : IN std_logic;
                RESET : IN std_logic;
                CLK : IN std_logic;
                START2 : OUT std_logic;
                RBEC : OUT std_logic
                );
    END COMPONENT;
        SIGNAL cycle : integer := 0;
        SIGNAL START :   std_logic;
        SIGNAL RBWE :   std_logic;
        SIGNAL START2 :   std_logic;
        SIGNAL RBEC :   std_logic;
        SIGNAL RESET :   std_logic:= '1';
        SIGNAL CLK :   std_logic:= '0';
BEGIN
        uut: receiver PORT MAP(
                START => START,
                RBWE => RBWE,
                START2 => START2,
                RBEC => RBEC,
                RESET => RESET,
                CLK => CLK
        );
```

リスト 4.6　RECEIVER テストベンチ 2/5

```
process
 begin
            if (cycle < 300)then
                  cycle <= cycle +1;
                  wait for 10 ns;
                  CLK <= not CLK;
          else wait;
    end if;
 end process;

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;
end process;

process
    begin
      RBWE <= '0';
       wait for 80 ns; RBWE <= '1';

        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '1';
        wait for 20 ns; RBWE <= '1';
        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '1';
        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '1';
        wait for 20 ns; RBWE <= '1';
```

リスト 4.6　RECEIVER テストベンチ 3/5

```
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';

wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';

wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';

wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
```

リスト 4.6　RECEIVER テストベンチ 4/5

```
wait for 20 ns; RBWE <= '0';

wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';

wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';

wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '0';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
wait for 20 ns; RBWE <= '1';
```

リスト 4.6　RECEIVER テストベンチ 5/5

```
        wait for 20 ns; RBWE <= '0';

        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '1';
        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '1';
        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '0';
        wait for 20 ns; RBWE <= '1';

        wait for 20 ns; RBWE <= '0';
        wait;
    end process;

  process
     begin
            START <= '0';
             wait for 520 ns; START <= '1';
                    wait for 20 ns; START <= '0';
             wait for 820 ns; START <= '1';
                    wait for 20 ns; START <= '0';
        wait;
     end process;
end architecture behavior;
```

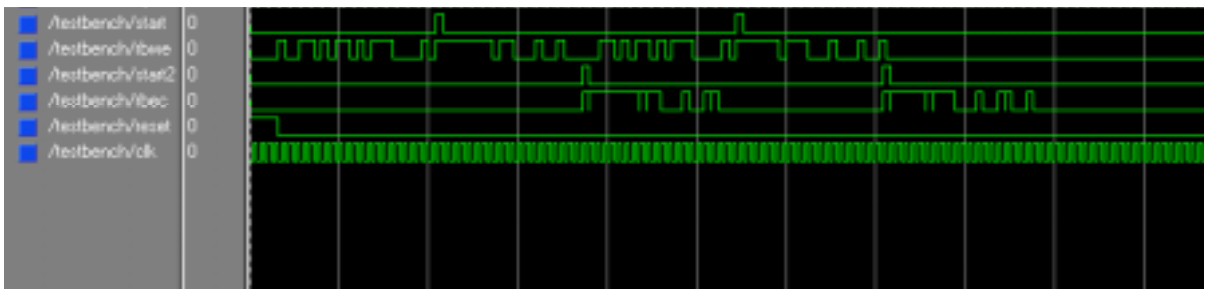

図 4.10　RECEIVER シミュレーション波形

4.3.6　クリティカル・パスの速度、論理合成後の回路規模

本節では、「4.2.5」節同様、論理合成ツール「LeonardoSpectrum LS2001_1a_31」を用いて実際に論理合成を行い、クリティカルパスの速度および、論理合成後の回路規模を求めた。

4.3.6.1　クリティカル・パスの速度

１段あたりのクリティカルパス遅延の基準は、論理合成された多段 XOR のクリティカル・パス遅延をもとに、4.2.5 節で求めた「0.49」を用いた。

設計した RECEIVER を、同論理合成ツールにて論理合成すると、遅延が「3.93」である（リスト 4.7 参照）。したがって「3.93／0.49　　7.97」　となるので、クリティカル・パスの速度は「7.97 UNIT 遅延」となった。

4.3.6.2　論理合成後の回路規模

論理合成後の回路規模はリスト 4. 7 に示すように、「ゲート数 585」となった。また、論理合成後の回路図を図 4.11 に示す。

リスト 4.7　RECEIVER の論理合成結果 1/3

```
                    Critical Path Report
Critical path #1, (unconstrained path)
NAME                                    GATE                ARRIVAL
LOAD
------------------------------------------------------------------------
clock information not specified
delay thru clock network                          0.00 (ideal)


reg_phase(1)/Q                  FD1B0       0.00   0.44 dn      0.20
ix579/X                         NR2R0       0.52   0.96 up      0.19
ix47/X                          ND2N0       0.62   1.59 dn      0.27
ix576/X                         NR2R0       0.65   2.24 up      0.19
ix79/X                          ND2N0       0.47   2.71 dn      0.11
```

リスト 4.7　RECEIVER の論理合成結果 2/3

```
ix573/X                          NR2R0          0.44    3.15 up          0.16
ix571/X                          NR2R0          0.41    3.56 dn          0.17
ix11/X                           AN2T0          0.36    3.93 dn          0.11
reg_phase(0)/D                   FD1B0          0.00    3.93 dn          0.00
data arrival time                                          3.93


data required time                                     not specified
--------------------------------------------------------------------------------
data required time                                     not specified
data arrival time                                          3.93
                                                       ----------
                                                   unconstrained path
--------------------------------------------------------------------------------
****************************************************
Cell: RECEIVER      View: RTL      Library: work
****************************************************
              Cell            Library    References       Total Area

              AN2T0            scl05u      5 x       5       24 gates
              AO1A0            scl05u      1 x       6        6 gates
              AO2L0            scl05u      2 x       8       15 gates
              AOA4I0           scl05u      1 x       8        8 gates
              AOA4I2           scl05u      1 x       8        8 gates
              FD1B0            scl05u     29 x       9      249 gates
              FD1I0            scl05u      9 x      11       95 gates
              FD1I1            scl05u      1 x      11       11 gates
              IV1N0            scl05u      3 x       3        9 gates
              IV1NP            scl05u      2 x       4        8 gates
              ND2N0            scl05u      5 x       5       23 gates
              NR2R0            scl05u      4 x       5       18 gates
              NR3R0            scl05u      1 x       6        6 gates
              NR4R1            scl05u      1 x       8        8 gates
              OAI1A0           scl05u      3 x       6       19 gates
              OAI3R2           scl05u      1 x       8        8 gates
```

リスト 4.7　RECEIVER の論理合成結果 3/3

```
        XN2R0          scl05u     10 x      5      49 gates
        XN3R0          scl05u      3 x      7      20 gates
 Number of ports :                    6
 Number of nets :                    89
 Number of instances :               82
 Number of references to this view :  0
Total accumulated area :
 Number of gates :                  585
```

RECEIVER の論合成後の回路図は、以下の 2 つの回路図を接合した形で構成される。

図 4.11　RECEIVER 論理合成回路図

4.4　VHDL ソフトを用いた ERRORCNT の設計

本節では、差集合巡回符号エラー訂正回路における、エラー訂正の一連の動作において、TRANSMITTER と RECEINVER による、エラー訂正能力の評価をするために用いる ERRORCNT（エラーカウント）の設計について説明する。

本節では、TRASIMITTER と RECEIVER の設計の際に用いた、Xilinx 社製「WebPACK Project Navigator」を用いて、VHDL による回路設計を行った。ソースコードをリスト 4.8 に示す。

リスト 4.8　ERRORCNT ソースコード 1/4

```
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;
            RESET : in std_logic;
            CLK : in std_logic;
                        ERRCNT : out unsigned(7 downto 0));
end ERRORCNT;

architecture RTL of ERRORCNT is

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

リスト 4.8　ERRORCNT ソースコード 2/4

```
-------------------------------------------------------------------
-- 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');
```

リスト 4.8　ERRORCNT ソースコード 3/4

```
        elsif rising_edge(CLK) then
         if (en='1') then
              data <= data(19 downto 0) & RB;    --RB_in;
         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;
```

リスト 4.8　ERRORCNT ソースコード 4/4

```
   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;    --RBec_in;
      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 RTL;
```

4.5　システム全体のシミュレーション

本節では、これまでに設計してきた TRANSMITTER、RECEIVER および ERRORCNT から構成される、「エラー訂正回路」のシステム全体のシミュレーションを VHDL にて前節までと同様に、Xilinx 社製「WebPACK Project Navigator」を用いて、VHDL によるテストベンチの設計を行い、同社製「Xilinx WebPACK 4.1 ModelSim XE Starter」を用いてシミュレーションした結果を表す。テストベンチはリスト 4.9 に、シミュレーション結果は図 4.12 および図 4.13 に示す。

リスト 4.9　全体システム用テストベンチ 1/3

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench 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;
```

リスト 4.9　全体システム用テストベンチ 2/3

```
    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 cycle : integer := 0;
   SIGNAL CLK :   std_logic := '0';
        SIGNAL RESET :   std_logic := '1';
        SIGNAL ERRCNT :   unsigned (7 downto 0);
        signal SBWE : std_logic;
        signal SB : std_logic;
        signal START : std_logic;
        signal RBEC : std_logic;
        signal START2 : std_logic;

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
            if (cycle < 1500)then
```

リスト 4.9　全体システム用テストベンチ 3/3

```
                cycle <= cycle +1;
            wait for 10 ns;
                    CLK <= not CLK;
             else wait;
      end if;
  end process;


  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 behavior;

configuration CFG_DCC21 of testbench is       for behavior
          end for;
end configuration CFG_DCC21;
```

図 4.12　全体システムのシミュレーション波形

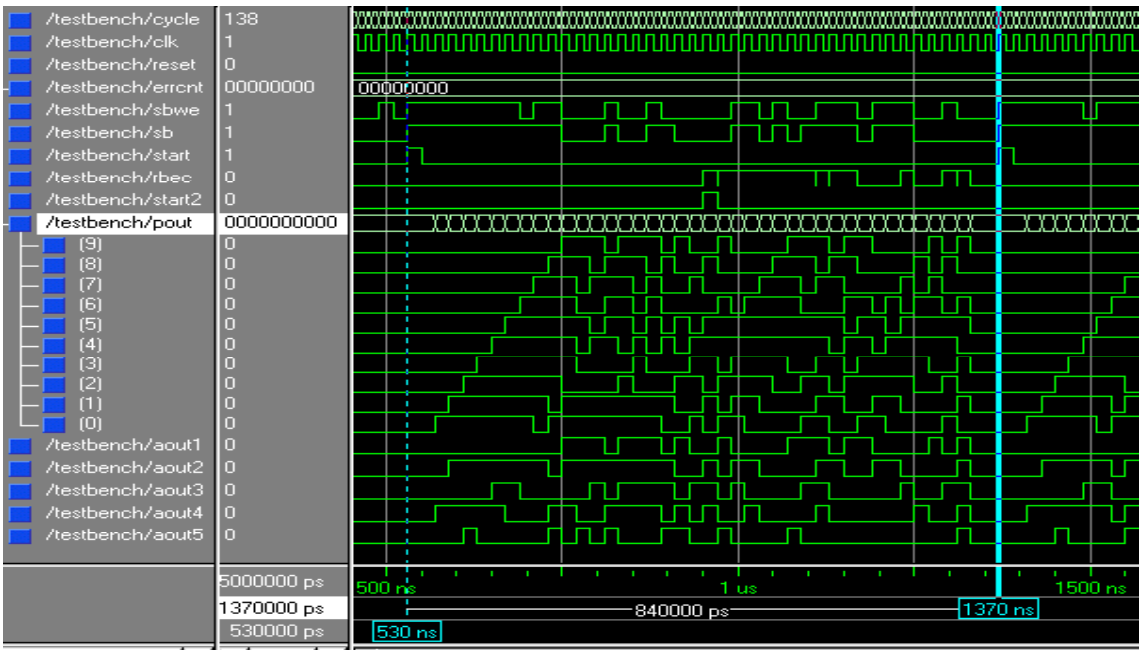

図 4.13　全体システムシミュレーション波形（拡大）

以上により、本章のテーマであり、本卒業研究テーマである「差集合巡回符号エラー訂正回路」の設計ブロック図によるシステム設計に始まり、システムを構成する各回路（エラー訂正に関わる部分として「送信機」、「受信機」および「エラーカウンタ」）の VHDL や回路図による設計・シミュレーション・論理合成等の過程を経て、最終的に全回路を組み合わせたシステムのシミュレーションに至るまでの手順ついての説明を終える。

第５章　おわりに

卒業研究を通して、文字多重放送に用いられる「差集合巡回符号エラー訂正回路」を設計するための技術を習得することができた。送信機で送りたいデータを作って送り出し、受信機で受け取ってエラーを発見し訂正する一連の動作を行うための概要や回路設計の技術について理解できた。また、エラー訂正が行われる背景となるテレビ等の文字多重放送についても知識を深めることができた。回路設計する上で用いた論理回路の基礎知識や VHDL 設計の技術（回路設計のソースコードの作成およびテストベンチを作成しシミュレーションする技術）についても学ぶことができた。さらに、論理合成ツールによるクリティカル･パスの速度や論理合成後の回路規模の評価についても理解できた。

今後の課題としては、FPGA で設計した回路を実装してみること、実装するために必要な知識を習得することが挙げられる。また、今回設計した回路のクリティカル・パスの速度や論理合成後の回路規模についても改善の余地があるとみられるので、より速度が速く、より規模が小さいものになるように検討したい。さらに、本研究は「DesignWave」誌の「設計コンテスト 2002」の中級レベルに相当するので、上級レベルの設計にも取り組みたい。

謝辞

本卒業研究を行うに際し、終始、懇切丁寧な御指導、御鞭撻を賜りました高知工科大学工学部電子・光システム工学科矢野政顕教授に心から感謝いたします。

研究中、懇切丁寧な御指導を賜りました高知工科大学工学部電子・光システム工学科学科長、原　央教授ならびに河津　哲教授、橘　昌良助教授に厚くお礼申し上げます。

また、終始、適切なご助言、ご助力をいただきました高知工科大学工学研究科基盤工学専攻情報システムコース前期修士課程、坂下雄一氏、高知工科大学大学院工学研究科基盤工学専攻電子･光エレクトロニクスコース前期修士課程、木村知史氏、中村基継氏、高知工科大学工学部電子・光システム工学科学部生、松村暢也氏、山崎慎太郎氏、松見隆之氏、ほか皆様に心から感謝し、お礼申し上げます。

参考文献

[1] 長坂進夫･小野義一･渡辺詔二･小山賢二 共著
“よくわかるテレビ･放送技術”オーム社 (1999 年)

[2] 荒谷孝夫 著
“通信ネットワーク” 東京電機大学出版局 (1997 年)

[3] 田丸啓吉 著
“論理回路の基礎” 工学図書株式会社 (1988 年)

[4] 中村次男 著
“ディジタル回路設計法” 日本理工出版会 (1990年)

[5] 高橋寛･関根好文･作田幸憲 共著
“ディジタル回路” コロナ社 (1996 年)

[6] 並木秀明･永井亘道 共著
“VHDL によるディジタル回路入門” 技術評論社 (2001 年)

[7] 長谷川裕恭 著
“VHDL によるハードウェア設計入門” CQ 出版社 (1995 年)

[8] “ Design Wave Magagin 2000 ” 12 月号 CQ 出版社

[9] “ Design Wave Magagin 2001 ” 11 月号 CQ 出版社

[10] “ NHK information ” http://www.nhk.or.jp/pr/bangumi/moji/moji-4.htm

[11] “ FM 文字多重放送 ” http://www.hht.co.jp/ass/fm/1.html

[12] “ 読売 ON-LINE Bit Bit Bit ”
http://www.yomiuri.co.jp/bitbybit/bbb12/882101.htm