卒 業 研 究 報 告

題　　目

VHDL による CRC 誤り検出回路の設計

指 導 教 員

矢野　政顕　教授

報 告 者

学籍番号：1030196
氏名：島崎　曜

平成 15 年　 2 月 10 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

近年は新しいネットワークとして、携帯電話を使う移動体通信網と、伝達情報に　電話以外の多様な情報も対象とするサービス総合ディジタル網（ISDN）が出現し、ユーザは急増しともに急成長するにいたっている。移動体通信が“ いつでも ”“ どこでも ”“ だれとでも ”を実現するサービスを提供するものであり、ISDN は“ なんでも ”で表現されるマルチメディア通信の要望に応える新しい統合化されたサービスを提供するものであり両者とも現在の情報化社会を支える基幹となっている。

後者の ISDN においては様々な情報通信サービスの統合化を図っているため、送受信において詳細な取り決めを厳密に行っておく必要がある。また伝送の途中で混入する雑音により符号誤りが生じたときに、チェック符号を用いてこれを検出し送信側からデータを再送してもらうことも不可欠である。本研究は、これらの各種の制御を行う手順として規格化されている HDLC 手順について論述し、その中で実際に使用されている CRC 誤り検出回路を VHDL を使って設計することを目的とする。

本報告は６章から構成されている。第１章では研究の目的について述べる。第２章では ISDN についてと HDLC 手順が使用されている ISDN の LAPD について説明する。第３章では HDLC 手順の仕組みや構成について説明する。第４章では誤り第３章では HDLC 手順の仕組みや構成について説明する。第４章では誤り制御符号として使われている CRC 方式について説明する。第５章では CRC 回路の VHDL による設計について説明する。第６章では本研究で得られた成果と今後の課題について述べる。

第２章 ISDN

2.1 ISDN について

2.1.1 ISDN とは

ISDN は Integrated Services Digital Network の略称で、日本語では、サービス総合ディジタル網とよばれる。その呼称の示すように、ISDN では情報も信号（制御のための情報）もディジタル化され、各種サービスが１つの通信網で総合的に提供される。

これまでの通信網は、電話，データ，ファクシミリなどのサービスごとにそれぞれ別々のネットワークで運営されてきたため、各メディアを利用する場合メディアごとに電話網、ディジタルデータ回線交換網、ディジタルデータパケット交換網などのように設備を別々に設けて作られてきた。ところが情報化社会が進んでくると通信の量も膨大になり、その度に新しくユーザー回線を引いたり、切り替えていたのではお金もかかり、効率的ではない。そこで、図２．１のように、異なった情報を１本の回線で同時に扱えるネットワークが必要となってきた。[4]

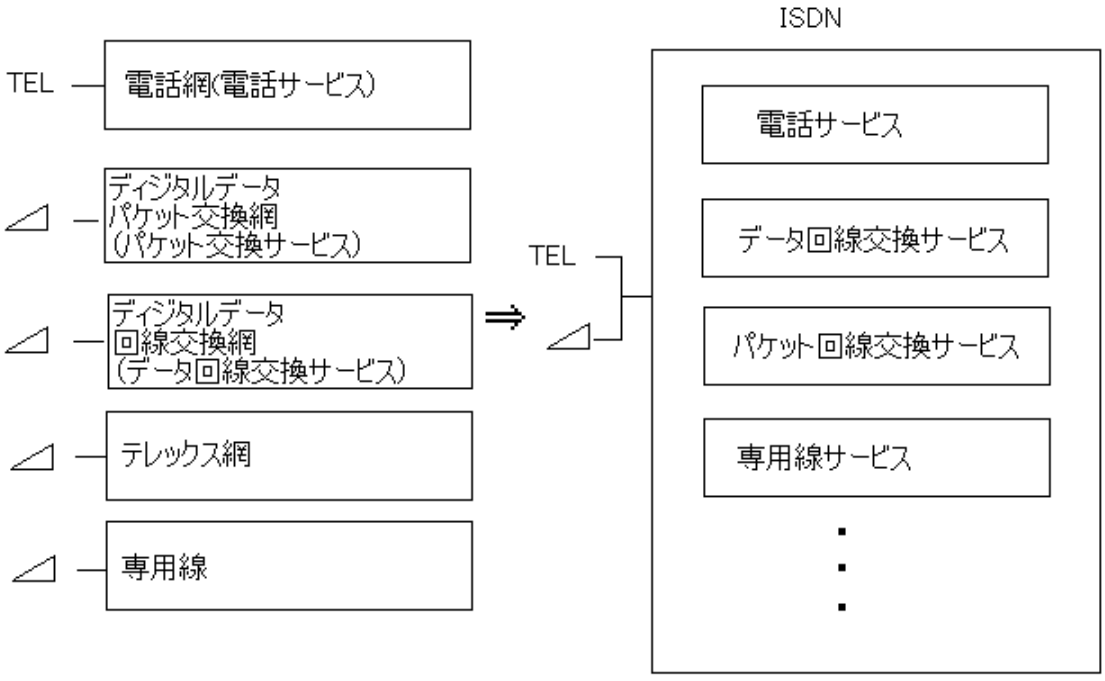


図 2.1 ISDN の概念

2.1.2 ISDN の標準化活動

電気通信に関する標準化活動は CCITT(国際電子電話諮問委員会の略で ITU：国際電気通信連合の下部組織)において古くから行われ、研究結果は勧告としてまとめられているが、これは実質上の国際標準化規格となっている。国際間の電気通信を対象とする新技術は、これまですべて CCITT により勧告され、これをもとに機器を製造し、通信システムを構築し、運用している。ISDN のようにネットワークを改革する影響の大きな新技術については早くから国際的に討議を重ねておく必要があり、１９７０年代に入ってから PCM 方式を取り扱っている委員会（Sp.D）で取り上げられ、１９７６年に ISDN を研究する委員会（SGX８）が発足し標準化に向けての本格的な活動が行われるようになった。１９８０年の総会では最初の ISDN の基本概念の勧告が制定され、１９８４年には ISDN の基本勧告が、１９８８年には詳細勧告が制定され、１９８８年から世界の先進国で一斉に ISDN のサービスが開始された。その後の CCITT における ISDN の研究は高速広帯域に拡張し本格的なものを目指したものに重点が置かれ、１９９１年には B-ISDN の基本勧告が制定され今日に至っている。これとは別に ITU の組織は１９９３年に効率を上げるために大幅に改革され、標準化活動は新しくできた電気通信標準化セクタ(略称は ITU-T：Telecommunication Standardization Sector)が CCITT に代わって行うこととなった。ISDN の勧告は I.XXX のように I と３桁の番号で系列化して表すので I シリーズ勧告と呼ばれている。[4]

2.2 プロトコル

2.2.1 プロトコルとは

人と人がたがいに意思の疎通を図ろうとする場合、そこにはさまざまな約束事が存在する。例えば、筆談をするときの文字はアルファベッド、言葉は英語というような取り決めがそれに当たる。コンピュータ端末間で通信を行う場合にも同様な約束事が必要となる。具体的には、要求された機能を実現するためにプログラムなどの機能実行主体（エンティティとよぶ）間で取り決められた情報交換の様式をさしている。

コンピュータ利用の発展に伴い、それぞれ独自に構築されたシステムやコンピュータネットワークを相互に接続して、資源の共用･分散化および処理の共同・分散化を図り、より一体化したネットワークに統合していく必要が出

てきた。そのためには、従来のようなシステム間での単なるデータ転送を可能とするレベルから、他システムのファイルやプログラムなどの資源へアクセス可能にするレベルまでを含んだ、プロトコルの総合化・体系化、およびその標準化が重要になる。

この約束事がプロトコル（protocol：通信規約）であり、ギリシャ語のProtokollon（ローマ帝国政府がパピルスの先頭に付けた見返しで、公式に認められた貿易であることを示す）を語源にもっている。[4]

2.2.2 OSI 参照モデル

コンピュータネットワークをより有用なものにしていくためには、同一メーカ内に閉じたコンピュータネットワークから、異なるメーカの製品を組み合わせた、より総合的なコンピュータネットワークへと発展させていく必要がある。OSI(Open Systems Interconnection：開放型システム相互接続)は、そのための一般的なシステム間相互接続を狙いとしたネットワークアーキテクチャであり、ISO(国際標準化機構)、および CCITT(国際電信電話諮問委員会)で国際標準化が進められている。ここで、open（開放型）とはどのシステムとも通信が可能であることを意味する。

OSI 参照モデルは、OSI のプロトコルを標準化していくために設定された一般的なシステム間通信モデルであり、開放型システムの通信機能を通信過程に対応させた７つの機能階層に分割している。各機能階層では、必要なプロトコルを定め、１つ下の層が提供するサービスを利用して、その層に割り当てられたサービスを実現する。７階層のうち、低位層（１～３層）はシステム間の透過的な効率の良いデータ転送を分担し、高位層（４～５層）は、応用プロセスが相互に会話しながら処理を進めていく上での各種の通信処理機能を実現している。[4]

以下に各層ごとの機能を示す。

（１）物理層

上位層から渡されるデータを通信回線を通して、隣接する装置へ忠実にビット伝送するために、接続用コネクタのピン位置規定などの機械的整合条件、電気的整合条件、および装置間の機械的・手続的な仕組みを実現する。

（2）データリンク層

連続するビットデータ列を伝送単位に区切る機能、伝送単位ごとに順序制御機能、誤り検出、回復機能、およびフロー制御などにより、装置間（リンク・バイ・リンク）のビット伝送を実現する。ISO が標準化した HDLC がこの層のプロトコルに対応する。

（3）ネットワーク層

データリンク層の提供するリンク・バイ・リンクのデータ転送機能を利用して、1つまたは複数の通信網を経由し、応用プロセスの存在する開放システム間のエンド・ツー・エンドなデータ転送を実現する。

（4）トランス―ポート層

伝送媒体の種類や品質、ネットワーク構成の違いを吸収して、エンド・ツー・エンド間における高信頼で経済的な両方向同時転送機能を提供します。

（5）セッション層

応用プロセス間で合意された一定のルールに従って、秩序正しくデータを送受信するための機能を提供する。

（6）プレゼンテーション層

データの表現形式の折衝・識別・解釈などを行い，必要に応じて表現形式を交換するための機能を提供する。

（7）アプリケーション（応用）層

データベースやアクセスなどの資源利用機能、およびコンピュータネットワークの運転制御に必要なネットワーク管理機能を提供する。

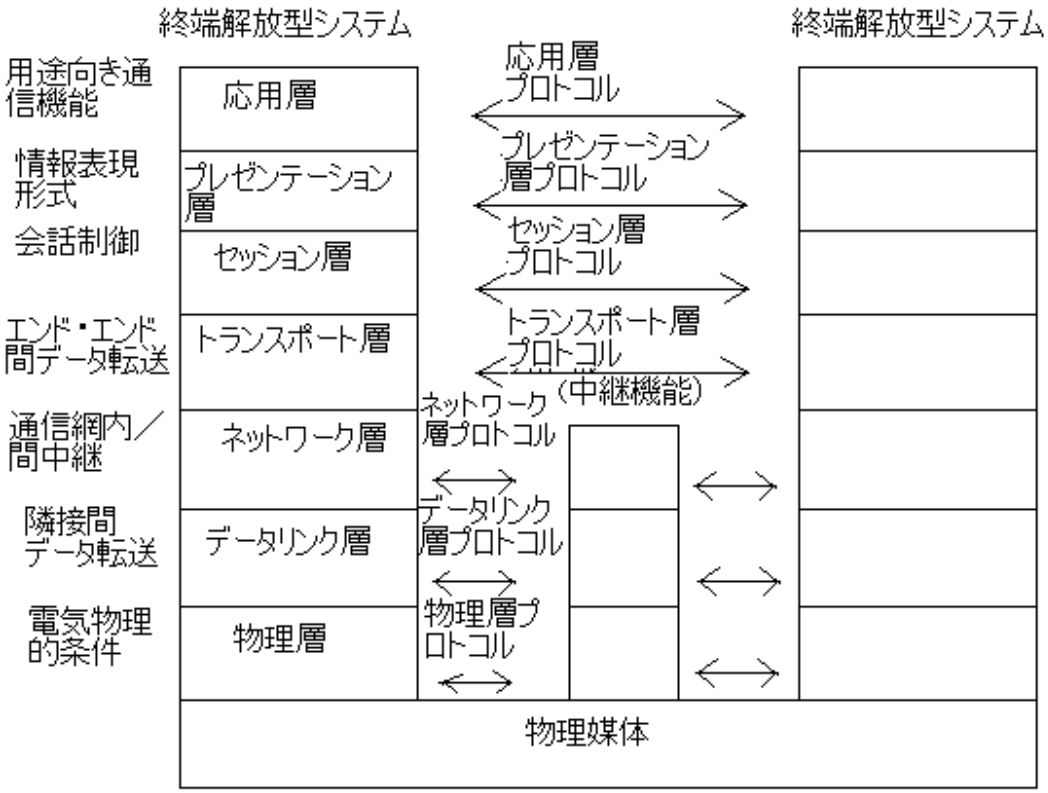


図 2.2 OSI プロトコル

ISDN ユーザ・網インタフェースでも、OSI 参照モデルの考え方に基づいて後に述べる D チャネルのプロトコルが階層化されている。実際には通信網を介してユーザ間に回線設定をするのに必要なレイヤ１からレイヤ３までが設定される。レイヤ４以上の階層は, 回線が設定された後ユーザ間で直接情報をやりとりするためのもので、ユーザ・網インタフェースとしては規定されない。［4］

2.3 チャネルについて

情報を運ぶためのパイプのようなものをチャネルとよび、運ぶ情報の種類と速度によって、大きく分けて B チャネルと D チャネルの２種類のタイプがきめられている。

B チャネルは音声やデータなどを送るためのチャネルで、複数あるチャネルは、それぞれ独立した回線として使用できる。したがって、１つの B チャネルを電話・ファックス兼用に、別の B チャネルをインターネットにというように接続することによって同時に別々に使うことができる。

D チャネルは B チャネルの接続に使うための共通の信号線となる。アナログ回線では、音声などの情報信号を送るのもダイヤル信号や呼び出し信号などの制御信号を送るのも同じ回線を使っているので、話している最中に制御信号を送ることはできない。

これに対して、ISDN では情報を送る B チャネルと独立して D チャネルが設けられているので、相手の通信機器やサービスに応じていろいろな制御信号を、速く正確に送ることができ、多彩なサービスが可能になる。[4]

2.4 ISDN における OSI 参照モデルの規定

2.4.1　OSI 参照モデルと ISDN ユーザ･網インタフェースの関係

OSI 参照モデルでは、通信機能を７つの階層（レイヤ）に分けて分類し、規定することによって各階層ごとの機能を互いに独立に扱えるようにしている。これにより、１つの階層に対応した機能の変更が通信システム全体の変更を伴わずに済み、拡張性のあるシステムを構築していくことができます。ISDN ユーザ・網インタフェースでも、これらの考え方に従って、OSI 参照モデルに準拠した階層モデルによる規定を採用している。

具体的な規定は、チャネルタイプ（D、B）サービス形態（回線交換、パケ

ット交換に対応して、レイヤ１からレイヤ３までの規定が行われています。例えば、D チャネルに対しては、D チャネルが呼制御用のチャネルあるいはパケット交換用のチャネルとして用いられるため、レイヤ１からレイヤ３までの規定が行われているが、B チャネルの回線交換サービスに対しては、レイヤ１の規定のみが行われている。[4]

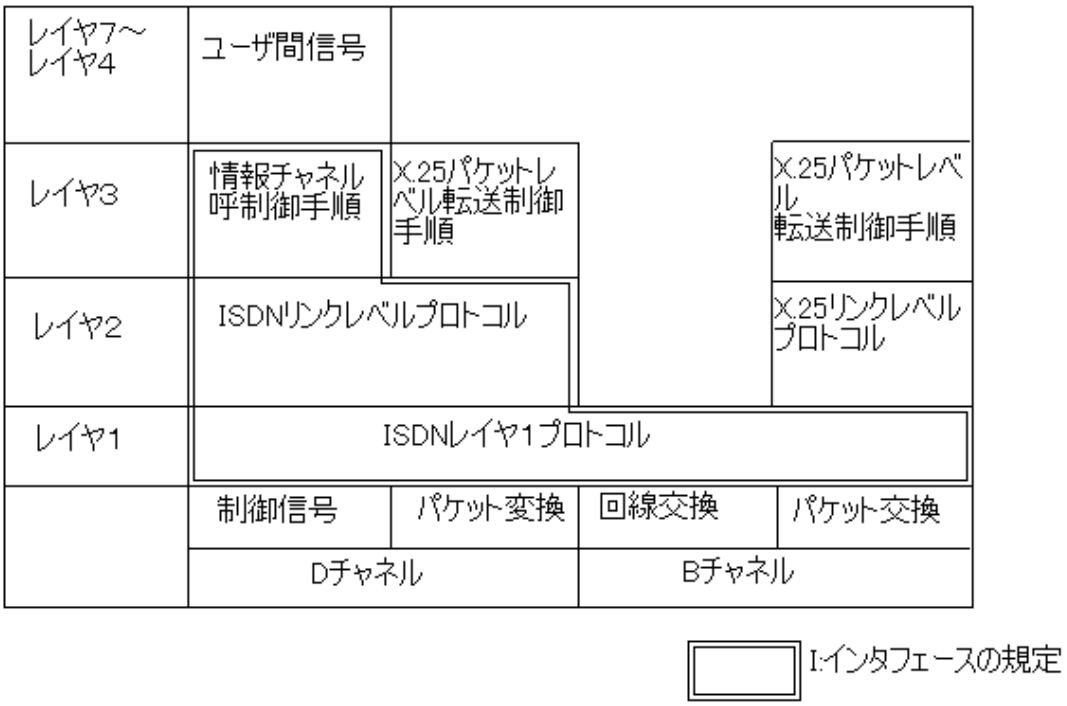


図 2.3　OSI 参照モデルと I インタフェースの関係

2.4.2 ユーザ・網インタフェースのレイヤ２の構成

ユーザ・網インタフェースのレイヤ２（データリンク層）は、通信網と端末の間で信号やパケットのやり取りをするための手順規定となる。信号やパケットはフレームと呼ばれる単位（最大２６８オクテット）で送受されます。ISDN の D チャネルでは、従来のパケット網で用いられてきた高水準データ伝送手順 HDLC の平行モード（通信をする装置がどちらも対等な制御機能をもつモード）を基本に、ISDN 特有のつぎのフレームごとに番号を付け、自局が送信

するフレームの中にすでに相手から受信している最終フレームのつぎの番号を送り返すことにより、フレームの受信確認を行う。これにより複数のフレームの一括送信が可能で、効率のよいデータ転送ができる。

- １つのインタフェースに対する複数端末の接続
- 複数サービスの同時提供（回線交換呼制御用情報の転送、パケット交換のパケットの転送など）

具体的には、フレームフォーマット中に信号、パケットなどのサービスを識別するサービスアクセスポイント識別子（SAPI：Service Access Point Identifier)、端末を識別する端末終端点識別子（TEI： Terminal Endpoint Identifier）を設けてこれらを実現している。また、全端末が受信することを指示する TEI を設定することにより、バス接続の特性を生かした放送型リンクによる情報転送も提供できます。TEI は手動により人間が設定することもできるが、ソケットに端末を差し込んだ時点で通信網が自動的に設定する方法も決められており、ユーザが TEI の管理をしなくてもよい。

ISDN のレイヤ２は、D チャネル上のリンクアクセスプロトコルであることから LAPD と総称され、CCITT 勧告 Q.９２１/I.４４１で詳細な仕様が規定されている。[4]

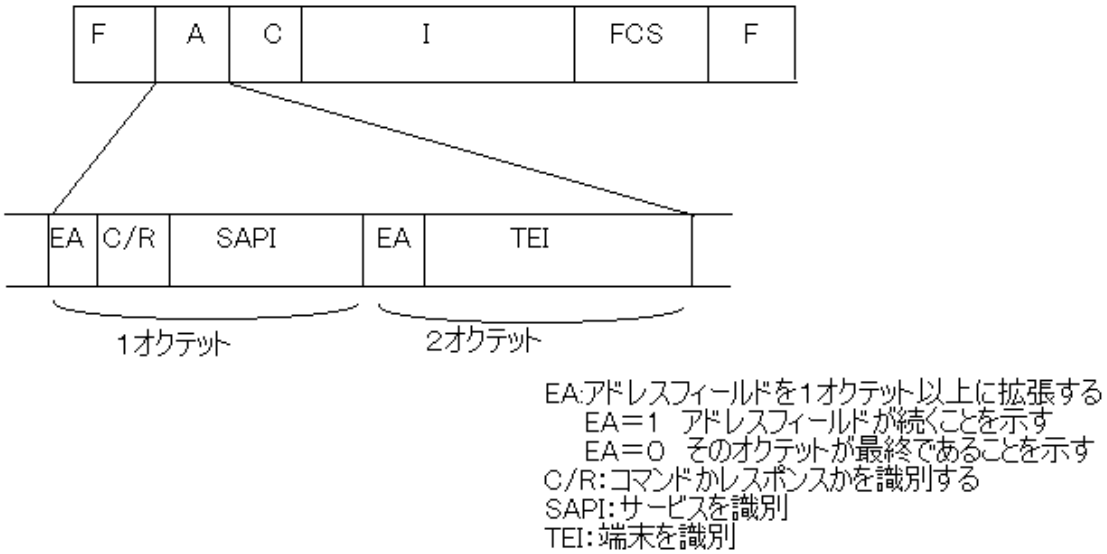


図 2.4　LAPD のアドレスフィールド

第 3 章 伝送制御

3.1 伝送制御の種類

端末とセンター間、または端末相互間でデータ伝送を行うときには、送信相手に確実に接続することと情報を正確に伝送することが重要である。電話による会話と異なり多様な機械対機械通信の場合には、送受信装置間でハードウェアとソフトウェアについて詳細な取り決め（プロトコル）を厳密に行っておく必要がある。また伝送の途中で混入する雑音により符号誤りが生じたときに、チェック符号を用いてこれを検出し送信側からデータを再送してもらう仕組みを作っておくことも不可欠である。これらの各種の制御を総称して伝送制御と呼び、プロトコルを伝送制御手順という。図 3.1 に伝送制御手順の種類を示す。

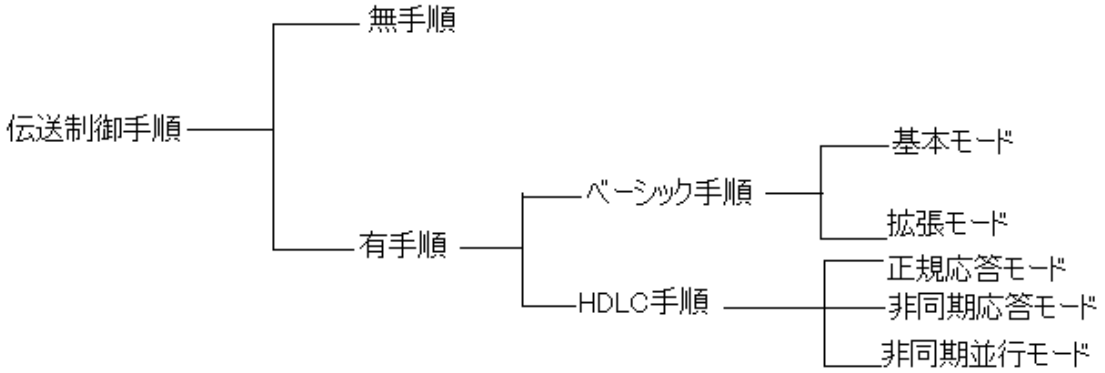


図 3.1 伝送制御手順の種類

無手順はスタート／ストップビットをつけただけでほかに何も取り決めのない簡単なものをいう。基本形データ伝送手順は低速から高速に至るまで幅広く使用されているもので基本モードのほかに拡張モードがあり、ベーシック手順と略称されているい。HDLC 手順は比較的新しく高速で高度の機能をもつもので個々の動作モードがある。図 3.2 にベーシック手順と HDLC 手順についての違いを示す。[3]

	ベーシック手順	HDLC手順
伝送単位	情報メッセージ	フレーム単位
転送データの内容及び長さ	キャラクタ単位	任意長のビット列
データ伝送方法	情報メッセージごとに確認応答を行う 伝送速度は、それほど速くない	連続してフレーム送信できる 高速伝送可能
同期制御方式	キャラクタ同期	フラグ同期
誤り制御方式	水平垂直パリティ方式 データの信頼性は、それほど高くない	CRC方式 データの信頼性が高い

図 3.2 ベーシック手順と HDLC 手順の比較

3.2 HDLC 手順

ベーシック手順はコンピュータと端末装置間のデータ伝送を前提としており、コンピュータ間の高速データ伝送に対しては伝送効率が悪く不十分な面が多かった。そこで、その後 ISO が中心となって検討され新しく考えられた制御手順が HDLC 手順であり、わが国では１９７８年に JIS の規格が制定された。この手順の考え方はパケット交換、ISDN、LAN などの制御手順の基本となるものであり、非常に重要な制御手順である。[3]

3.2.1 フレーム構成

HDLC 手順を特徴づけるものがブロックに代わるフレームである。フレームは転送の単位で情報メッセージも監視制御情報もすべてフレームの形で送受信される。フレームの構成を図 3.3 に示す。次に各フィールドについて説明する。[3]

（1）フラッグシーケンス

これはフレームの始めと終わりにつけられる８ビットの特定パターン“０１１１１１１０”であり、受信側にフレームの開始と終了を知らせるとともに同期（フラッグ同期）の役割をもっている。もしもこれと同じビットパターンが

ほかのところで現れると受信側でフレーム終了と誤って判断することになるので、送信側が１が５個連続したら強制的に“０”を６ビット目に挿入し受信側でこれを除去する方法がとられている。

（２）アドレスフィールド

フレームの宛先または送信元を（レスポンスの場合）のアドレスを８ビット使って表現するので２５６局まで指定できるが、それ以上のときは１６ビットまで拡張することができる。

（３）制御フィールド

相手局に対する動作の指令や応答を示す部分で最も複雑な取り決めがある。これはベーシック手順の伝送制御キャラクタの役割を、より高度にして８ビットで対応させる。

（４）情報フィールド

利用者にとって実際に必要な情報メッセージと制御情報など転送すべきデータが入っている部分である。ユーザ情報の長さには特に制限はなく、任意のビットパターンで伝送できす。

（５）フレームチェックシーケンス

これはフラッグを除くすべてのフィールドの内容が正確に転送されたかどうかをチェックする誤り制御のためのフィールドであり、ベーシック手順の BCC に相当する。HDLC では１６ビットを使い、CRC（Cycric Redundancy Check）方式が使われている。

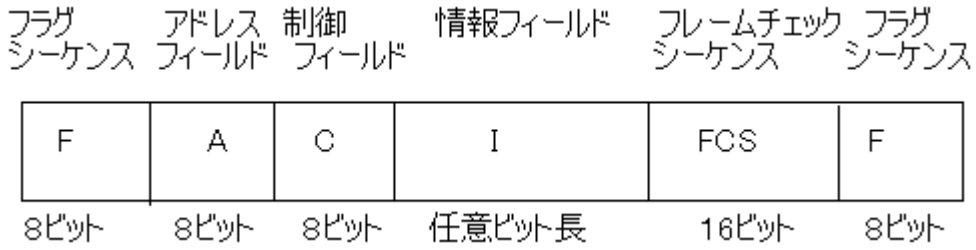


図 3.3 HDLC のフレーム構成

第４章 CRC 方式

4.1 誤りとは

ディジタル通信とアナログ通信との決定的なちがいは、ディジタル通信には誤り検出があるが、アナログ通信ではそのような概念が存在しないということである。それはアナログ通信では雑音や周波数分散による信号の歪みが元の信号との違いになり、これが誤りになるが最終的には人間の五感によって読み取ることができるからである。伝送途中で雑音などにより“１”が“０”に、もしくは“０”が“１”にビット誤りを起こすことがあるディジタル通信では、最終的にアナログ信号に戻される場合を除き、正確性が厳しく要求されることから誤りは重大な結果につながることになる。このような誤りを定量的に表す尺度としてビット誤り率（受信した全ビットに対する誤りビット数の割合）があるが、ビット誤り率はデータ通信の場合、回線交換サービスで１０^{-5}、パケット交換網で１０^{-10}程度と言われている。前者では１Mb のデータ中に１０b の誤りが、後者では１０Gb 中に１b の誤りデータがまぎれこむことがある。そのために、データ通信ではいたるところに誤り検出や誤り訂正が含まれている。

誤り検出は、送信側のデータに一定の演算を施し送信データと共に誤り検出用のチェックコードを付加して伝送し、受信側でチェックコードを検査し誤りの有無を判定する。誤り検出方法は、演算方法とチェックコードの違いによって大きくパリティチェック方式と CRC 方式に分けることができる。

$^{-10}$

4.2 パリティチェック方式

パリティチェック方式は、ASCII や JIS などの標準符号の伝送時に、伝送するキャラクタ（文字）ごとに１ビットを付加し、付加ビット（パリティビット）を含めた“１”の合計数が偶数（または奇数）とし、受信側で偶数（または奇数）になっているかどうかを調べ誤りを判定するものである。この方式は容易に誤りを検出できる利点があるが、反面、偶数個の誤りを検出できない欠点がある。したがって、誤りが連続的に発生しないランダム誤りは検出できるが２ビット以上がかたまって誤りとなるバースト誤りを検出できない。[3]

偶数パリティの例

送信データ　「A」＝＜１０００００１＞＋＜０＞

↓

受信データ　　＜１００１００１０＞

（1ビット誤りがあるが1の数が奇数になっているため誤りが検出できる。）

送信データ　「L」＝＜１００１１００＞＋＜１＞

↓

受信データ　　＜１０１０１００１＞

（2ビット誤りがあると1の数が偶数になっているため誤りがないと判断してしまう。）

4.3 CRC 方式

CRC 方式は、HCLD 手順における誤り検出方式で、アドレスフィールド、制御フィールド、情報フィールドを１つのデータ（情報ビット）として扱い、このデータに CRC 演算を施して計算されたチェックコードを FCS として付加したものを送信データとする。以下に CRC 演算によるチェックコードの生成方法を述べる。[3]

4.3.1 送信データの生成

CRC 演算を行うためにまずデータを変数 X の多項式と考える。例えばデータ“１１００１”において MSB を X^4、MSB より２番目の１を X^3、LSB の１を定数で表すと

$$M(x)=x^4+x^3+1$$

となる。次に定義されている生成多項式 G（ｘ）の最上位の係数をｋとして M(X)をｋビットシフトした $X^k M(X)$を G（ｘ）で割る。その際に次のような mod ２加算として、

$$1X^a+1X^a=0X^a$$

$$1X^a+0X^a=0X^a+1X^a=1X^a$$

$$0X^a+0X^a=0X^a$$

$$-1X^a=1X^a$$

と定義して商を Q(X),余りを R(X)とすると

$$X^k M(X)=Q(X)G(X)+R(X)$$

となる関係が得られる。FCS とはこの R（X）で表されるビット列のことを言う。CRC 方式では、実際に転送したいデータ M(X)に続けて FCS を送る。これを多項式で表すと

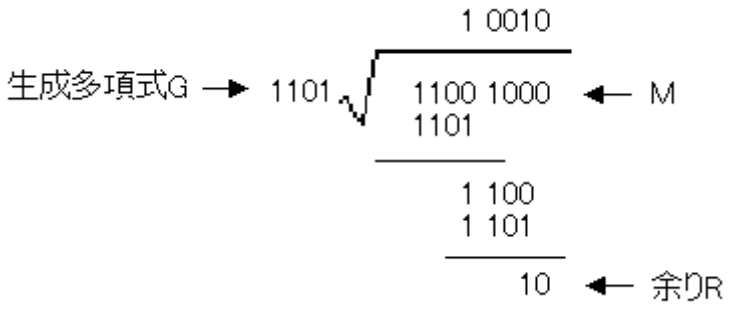
$$U(X)＝X^k M(X)+R(X)=Q(X)G(X)+R(X)+R(X)＝Q(X)G(X)$$

となる。mod２加算では１X^a+1X^a=0 なので一番右端の R(X)+R(X)は打ち消され、Q(X)G(X)が受信側に送信されることになる。

例として送信すべきデータ（FCS を除く）として M=１１００１、生成多項式 G＝１１０１とすると

X^kM=１１００１０００

X^kM/G＝１１００１０００/１１０１



X^kM+R=１１００１０１０

が送信データとなる。[5]

4.3.2 CRC 方式の誤り検出

受信側では、送信側から送られてきた U(X)＝Q(X)G(X)を生成多項式 G(X)で割り Q(X)を求め、余りがなかった場合に送信側からデータが正確に送られたことになる。

送信データ U(X)が、伝送路の雑音により誤りが生じた場合、受信データ V(X)は次のようになる。

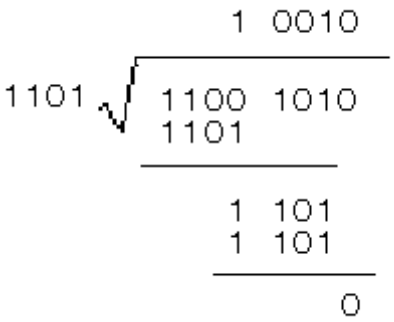
V(X)=U(X)＋E(X)

ここで、E(X)は、誤りの U(X)のデータの誤りが生じた項の集まりとする。

送信データ U(X)は G(X)で割り切ることができるので、受信データ V(X)が G(X)によって割り切れないならば、誤りが生じていることを示している。4.2.2 節の例を用いて計算を行ってみると

誤りがない場合

U(X)/G(X)＝１１００１０１０/１１０１



となり余りが０なので X^k M(X)の M(X)の部分、つまり１１００１が実際に送りたかったデータであったことが分かる。

誤り E(X)＝１１０が U(X)に加わったとすると

V(X)=U(X)+E(X)=１１００１０１０＋１１０＝１１００１１００

V(X)/G(X)＝１１００１１００/１１０１＝１００１余り１１０

となり、V(X)は G(X)により割り切ることができなかったため、送信中に誤りが発生したとみなすことができる。[5]

4.3.3 生成多項式

先程の例より V(X)が G(X)により割り切れないならば、それは E(X)により割り切れなくなったためであり、誤りが生じたことを示している。しかし V(X)が G(X)で割り切れるならば V(X)が誤りデータであってもそれを正しい受信データ

として受け取ってしまい、この場合は見逃し誤りとなってしまう。V(X)=U(X)＋E(X)の U(X)は G(X)で割り切れることが分かっているので、もし E(X)が G(X)で割り切れてしまうような雑音だった場合、V(X)が G(X)で割り切れてしまう。したがってデータ誤りの効果的な検出を保証するためには、検出したい誤りパターン E(X)が G(X)によって割り切れないような生成多項式 G(X)を選ばなければならない。そのためにデータ伝送を行っている実際の回線について、データ誤りの生のデータを収集し、それらの誤りパターンをすべて検出するような G(X)を選ぶようにしている。図 4.1 に生成多項式の種類を示す。[5]

名前	多項式	用途
CRC－CCITT	G(X)=X^16+X^12+X^5+1	文字８ビットデータ
CRC－１６	G(X)=X^16+X^15+X^2+1	文字８ビットデータ
CRC－１２	G(X)=X^12+X^11+X^3+X^2+X+1	６ビットデータ
CRC－３２	G(X)=X^32+X^26+X^23+X^22+X^16+X^12+X^11+X^10+X^8+X^7+X^5+X^4+X^2+X+1	LAN

図 4.1 生成多項式の種類

4.3.4 誤り検出の定理

4.3.4.1 奇数個の誤りの検出

図 4.1 中のそれぞれの生成多項式は全て（X＋１）を因数として含んでいることが分かる。（X＋１）で割り切れるデータ U(X)は必ず偶数個の項をもち誤りが生じて項の数が奇数になった場合に割り切れなくなる。そのために生成多項式に（X＋１）を因数としてもたせて偶数パリティチェックを付加したことに相当させ、奇数個の誤りを検出させることができる。[5]

4.3.4.2 バースト誤りの検出

MSB の係数をｋとする生成多項式 G(X)を用いて、送信データ U(X)を構成したとき、連続するｋビット以下の誤りを検出することができる。4.3.2 節の例を用いるとｋビット連続のバースト誤りは以下の６種類になる。

$E(X)=X^7+X^6+X^5=X^5\ (X^2+X+1)$

$E(X)=X^6+X^5+X^4=X^4\ (X^2+X+1)$

$E(X)=X^5+X^4+X^3=X^3\ (X^2+X+1)$

$E(X)=X^4+X^3+X^2=X^2\ (X^2+X+1)$

$E(X)=X^3+X^2+X=X(X^2+X+1)$

$E(X)=X^2+X+1$

これらは G(X)=１１０１では割り切れないために誤りを検出できる。CRC 方式がデータ伝送における誤り検出符号として多用されるようになったのはこのようなバースト誤りを検出することができるからである。[5]

4.3.5 演算回路の作成

今まで述べたように CRC 符号を作成するには乗算と除算（和、差に EXOR を用いた）を行う必要がある。次項から乗算回路、除算回路、及び乗算、除算を組み合わせた CRC 符号作成回路の作成方法を述べる。[5]

4.3.5.1 乗算回路

多項式の乗算回路として、多項式 A(X)と任意の多項式 B(X)を乗ずる回路を考える。

A＝１１００１　B＝１１０１とすると乗算回路は図 4.2 のようになる。図 4.2 の a,b,c,は多項式 B の定数 X^3、X^2、１の係数を示し、X^1の項の係数は０であるので、この項に対応する回路の接続はない。図中の□はシフトレジスタを表し＋は EXOR を表す。

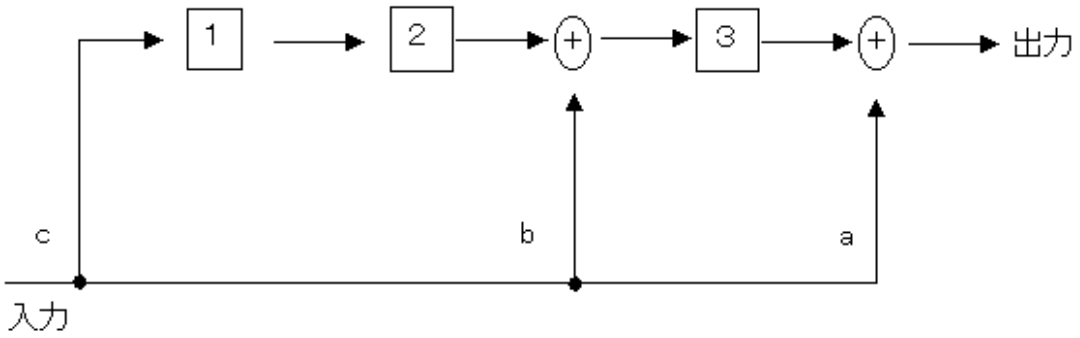


図 4.2 乗算回路

この動作を追ってみる。まずシフトレジスタ１～３には初期値として０が入っているものとする。入力線からは A＝１１００１が上位ビットから入力されていく。A の最上位ビットの値は１、シフトレジスタの値は全て０なので出力、レジスタ３、レジスタ２、レジスタ１には（１１０１）が入る。次に A の２ビット目の１が入力されると、出力、レジスタ３、レジスタ２、レジスタ１には（０１１１）が入る。この回路の入力、出力、シフトレジスタの内容を図 4.3 に示す。

シフト数	入力	シフトレジスタの内容	接続線	出力
		１２３	a b c	
0	-	０００	—	0
1	1	１０１	1 1 1	1
2	1	１１１	1 1 1	0
3	0	０１１	0 0 0	1
4	0	００１	0 0 0	1
5	1	１０１	1 1 1	0
6	0	０１０	0 0 0	1
7	0	００１	0 0 0	0
8	0	０００	0 0 0	1

図４．３　乗算テーブル

演算結果

この回路では、普通の乗算方法（下位から乗算を行い最後に全ての総和を算出する。）と違い上位から乗算を行いその都度そこまでの総和を算出している。計算方法と図 4.3 に対応する値を図 4.4 に示す。[5]

```
        １１０１
    ×１１００１
―――――――――――
１１０１
  １１０１
―――――――――――
  ０１１１  ←― 出力＆３＆２＆１（２シフト目）
   ００００
―――――――――――
    １１１０  ←―出力＆３＆２＆１（３シフト目）
     ００００
―――――――――――
     １１００  ←― 出力＆３＆２＆１（４シフト目）
       １１０１
―――――――――――
１０１１０１０１  ←― 出力＆３＆２＆１(５シフト目)
```

※＆はビットの連結
３，２，１はシフトレジスタの内容を示す

図 4.4 乗算回路計算手順

4.3.5.2 除算回路

多項式の割り算回路として多項式 U(X)を生成多項式 G(X)で除算する回路を作成する。

U(X)＝１１００１０００　G(X)＝１１１０１として、U(X)を G(X)で除算する回路を図 4.5 に、入出力・シフトレジスタの関係を図 4.6 に除算の計算を図 4.7 に示す。$G(X)=X^4+X^3+1$ の係数に対応するレジスタの後ろに加算記号を付加し、出力をフィードバック接続線を通して加算する回路となる。

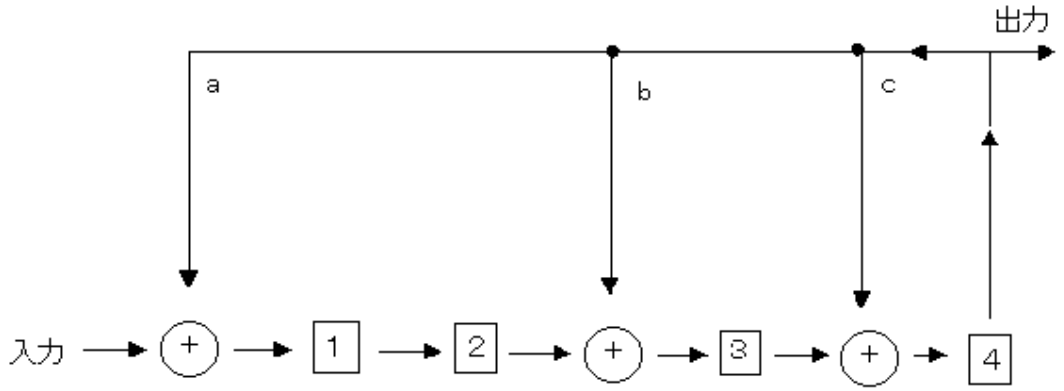


図 4.5 除算回路

シフト数	入力	フィードバック	シフトレジスタ	出力
		a b c	1 2 3 4	
0	—	0 0 0	0 0 0 0	0
1	1	0 0 0	1 0 0 0	0
2	1	0 0 0	1 1 0 0	0
3	0	0 0 0	0 1 1 0	0
4	0	0 0 0	0 0 1 1 ←A	1
5	1 ←B	1 1 1	0 0 1 0 ←C	0 ←商
6	0	0 0 0	0 0 0 1 ←D	1
7	0	1 1 1	1 0 1 1	1
8	0	1 1 1	1 1 1 0 ←余り	0

図 4.6 除算テーブル

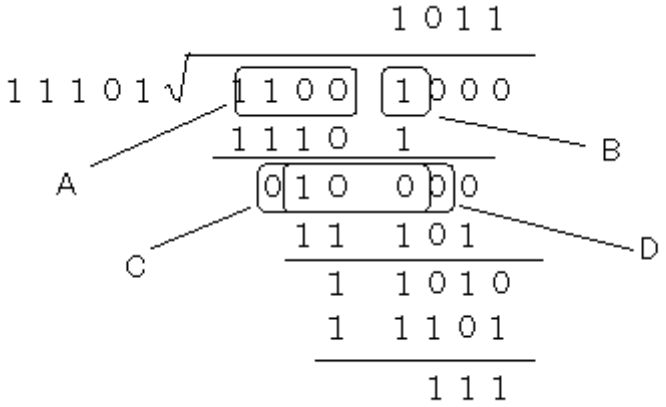


図 4.7 除算の計算

図 4.6 を見ると最初の４シフトの間の出力は０であり、４シフト後のシフトレジスタ１～４の値は図 4.7 の除算の計算で示す A のようになっており、その値に５シフトの時の入力 B を付加したものが最初の被除数となる。その時の除数はどうなるかというと４シフト後の出力がフィードバック接続線を通して（１１０１）が入る。この時の加算として被除数＋除数の最上位同士は共に１なので計算結果は０になることが分かっているので計算を行わない。回路図のレジスタ４の後に加算記号がつけられていないのはそのためである。(A&B）＋（フィードバック信号）の計算結果は図 4.6 の C となる。C の最上位の値（出力）は０なのでフィードバック接続線を通して加算を行っても影響がないので６シ

フト目は計算を行う必要がなくシフトするだけである。このように加算とシフトを繰り返すことにより最終的に出力は商であり、シフトレジスタの内容に余りが残ることになる。[5]

4.3.5.3 CRC 符号生成回路

4.3.1 節で述べたように余り R(X)を求めるためには X^k M/G を計算する必要がある。そこで 4.3.5.1 節の乗算回路と 4.3.5.2 節の除算回路を組み合わせた CRC 回路を作成する。

入力データ M＝１００１０

生成多項式 G＝１　０００１　０００００　００１０　０００１

とすると X^k M つまり M を生成多項式の最上位の係数ビットシフトさせる乗算回路は図 4.8 のようになる。図 4.2 の回路では値が１になっているところが X^3、X^2, 1 と３箇所あったが今回は X^{16} だけなので加算を行う場所も一つになる。

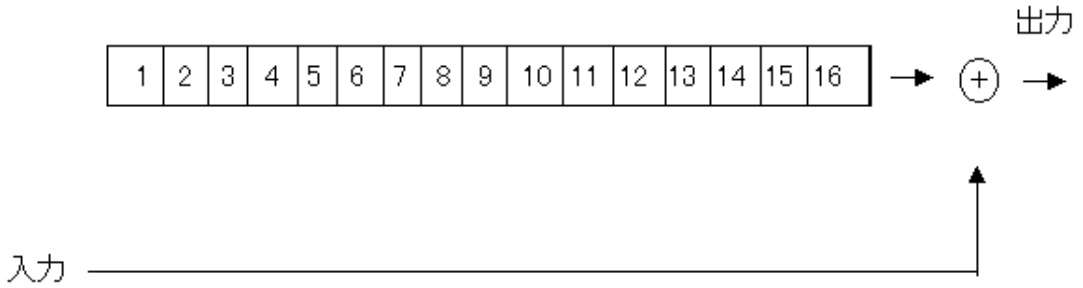


図 4.8　X^{16} 乗算回路

また生成多項式 G で除算する回路は図 4.9 のようになる。

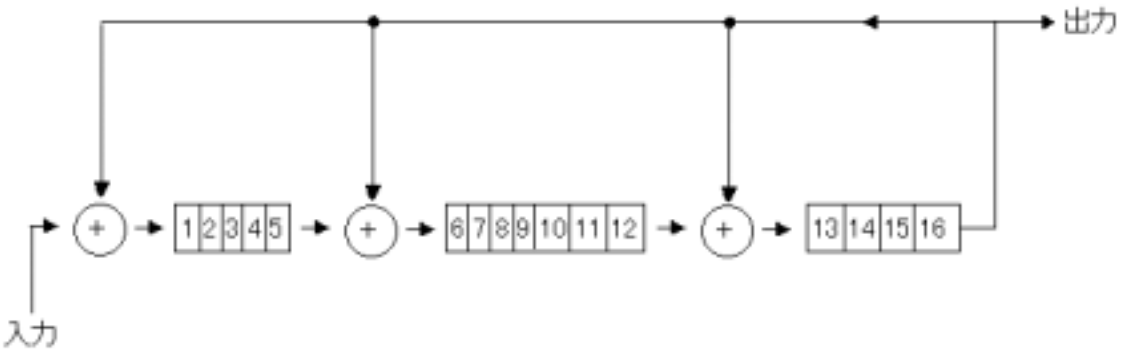


図 4.9 除算回路（生成多項式）

乗算と除算を同時に実行する回路は、図 4.8 と図 4.9 の回路を組み合わせた図 4.10 の回路によって $X^k M(X)/G(X)$の計算を実現できる。図 4.10 において入力データが入力し終わると、シフトレジスタには余り R（X）（＝００１１００１００１１１００１１）が残る。入力とシフトレジスタの内容を図 4.11 に示す。［5］

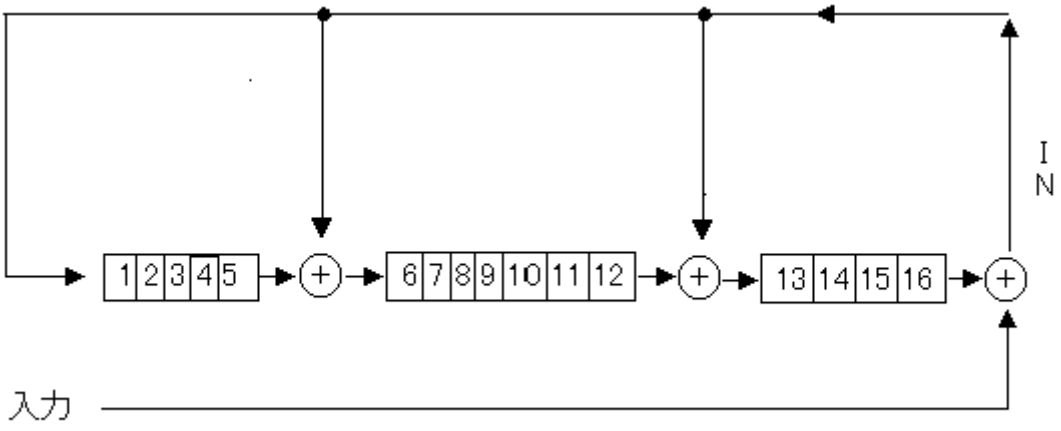


図 4.10 CRC 演算回路

入力データ	IN	レジスタの内容															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0
1	1	1	0	0	1	0	1	0	0	1	0	0	0	1	0	0	1
0	1	1	1	0	0	1	1	1	0	0	1	0	0	1	1	0	0

図 4.11 CRC 演算テーブ

第５章 VHDL による CRC 誤り検出回路の設計

5.1 直列入力送信機の設計

5.1.1 直列入力送信機の構成

4.3.5.3 節では送信データに付加する R(X)を求める回路を作成したが実際に送信するデータは入力データに R(X)を付加したものである。そのためにデータを出力する前の段階で入力データを出力するか R(X)を出力するかを選択する必要がある。また、R(X)が求まり出力する時に入力データが CRC 演算回路に入ってくると R(X)の値が変化する場合があるので入力データを切り離す必要がある。任意長ビットを送信する場合の送信機の構成を図 5.1 に示す。

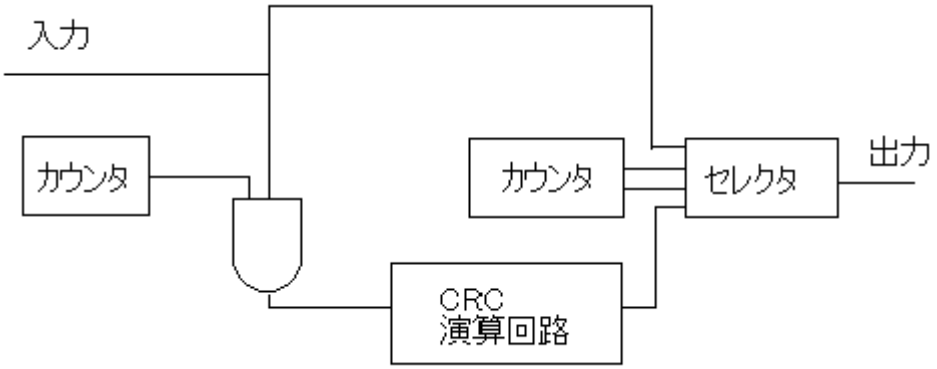


図 5.1 直列入力送信機の構成

任意長ビットを直列データで入力し、このビットシリアルのデータを CRC 演算回路（図 4.10）より R(X)を求める。この時に R(X)を求めるのには図 4.11 より入力ビット数分のクロックが必要となるのでこの間に入力データをそのまま出力する。入力データを出力し終わると同時に CRC 演算回路によって R(X)が求まるので今度は R(X)を出力する。この入力データから R(X)への切り替えをカウンタ回路によって行う。また、この時に入力データの CRC 演算回路への入力を切り離す。［1］［5］

5.1.2 VHDL による直列入力送信機の設計

図 5.1 の構成を元にした VHDL 記述を図 5.2 に、テストベンチを図 5.3 に示す。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity CRCSHIFT is
   port (
          a,clk,reset: in std_logic;
       outa: out std_logic
       );
end CRCSHIFT;
architecture BEHAVIOR of CRCSHIFT is
signal foo0,foo1,foo2,foo3,foo4,foo5,foo6,
foo7,foo8,foo9,fooa,foob,fooc,food,fooe,foof:std_logic;
signal count_in:std_logic_vector(4 downto 0);
signal outb:std_logic;
signal a_in:std_logic;
signal b_in:std_logic;
signal x:std_logic;

begin
```

図5.2 直列送信機のVHDL 1 / 5

```
process(a_in,a,outb,b_in)begin
   outa <= (a_in and a)or(outb and b_in);
end process;
process(clk,reset)begin
     if(reset='1')then
          foo0<='0';
          foo1<='0';
          foo2<='0';
          foo3<='0';
          foo4<='0';
          foo5<='0';
          foo6<='0';
          foo7<='0';
          foo8<='0';
          foo9<='0';
          fooa<='0';
          foob<='0';
          fooc<='0';
          food<='0';
          fooe<='0';
          foof<='0';
     elsif(clk'event and clk = '1')then
          foo0 <= (a  xor foof)and x;
```

図5.2 直列送信機のVHDL 2 / 5

```
          foo1 <= foo0;
          foo2 <= foo1;
          foo3 <= foo2;
          foo4 <= foo3;
          foo5 <= foo4 xor ((a  xor foof)and x);
          foo6 <= foo5;
          foo7 <= foo6;
          foo8 <= foo7;
          foo9 <= foo8;
          fooa <= foo9;
          foob <= fooa;
          fooc <= foob xor ((a   xor foof)and x);
          food <= fooc;
          fooe <= food;
          foof <= fooe;
          outb <= foof;

     end if;
end process;

process(clk,reset )begin
     if (reset = '1') then
          count_in<="00000";
```

図5.2 直列送信機のVHDL３/５

```
    elsif (clk'event and clk='1') then
          count_in <= count_in+'1';
     end if;
end process;

process(count_in)begin
     if(count_in<="00100"and count_in>="00000")then
          x <= '1';
     else x<= '0';
     end if;
end process;

process(count_in)begin
     if(count_in<="00101" and count_in>="00000")then
          a_in <= '1';
     else a_in <= '0';
     end if;
end process;

process(count_in)begin
     if(count_in<="11111" and count_in>="00110")then
          b_in <= '1';
     else b_in <= '0';
```

図5.2 直列送信機のVHDL４/５

```
     end if;
end process ;


end behavior;
```

図5.2 直列送信機のVHDL５/５

foo0,foo1,foo2,foo3,foo4,foo5,foo6,foo7,foo8,foo9,fooa,foob,fooc,food,fooe,foof：図4.10におけるCRC演算を行うシフトレジスタ

count_in:出力の選択と入力の切り離しのタイミングをカウントする信号。

a_in:入力データを出力するための選択線。

b_in:CRC演算データを出力するための選択線。

x:CRC演算回路への入力を切り離すための選択線。

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.CRCSHIFT;


entity TESTBNCH is
end TESTBNCH;
```

図5.3直列送信機のテストベンチ1/3

```
architecture stimulus of TESTBNCH is
component CRCSHIFT is
    port (
        a,clk,reset: in std_logic;
        outa: out std_logic

    );

end component;


signal a,clk,reset: std_logic;
signal outa: std_logic;

begin
    DUT: CRCSHIFT port map (
           a,clk,reset,
           outa
         );

    stimulus1: process
           begin
        a<='0';wait for 20ns;
```

図5.3直列送信機のテストベンチ2/3

```
      a<='1';wait for 40ns;
       a<='0';wait for 40ns;
       a<='0';wait for 40ns;
       a<='1';wait for 40ns;
       a<='0';wait for 40ns;
       end process stimulus1;
   stimulus2:process
   begin
clk<='0';wait for 20ns;

clk<='1';wait for 20ns;

end process stimulus2;


stimulus3:process
begin
reset<='0';wait for 3ns;
reset<='1';wait for 6ns;
reset<='0';wait for 1800ns;
end process stimulus3;
end stimulus;
```

図5.3直列送信機のテストベンチ3/3

5.1.3 シミュレーション結果

シミュレーション結果を図 5.11 に示す。a=“１００１０”が入力されている間は a_in が立ち上がり a がそのまま出力される。入力数ビットのクロック時間が経過した時の foo の値に R(X)が求められているので x を立ち下げることにより foo への入力を切り離す。次のクロックから b_in を立ち上げることにより

R(X)を出力する。出力としては“１００１０００１１００１００１１１００１１”となる。

5.2 直列入出力受信機の設計

5.2.1 直列入出力受信機の構成

4.3.2 節より受信側では、送信側から送られてきた U(X)＝Q(X)G(X)を生成多項式 G(X)で割り余りがなかった場合に送信側からデータが正確に送られたことになる。逆に送信データに誤りが起こったとすると、G(X)で割った時に余りが残ることになる。この計算を行うために図 4.9 の生成多項式による除算回路を用いた受信回路の構成を図 5.4 に示す。

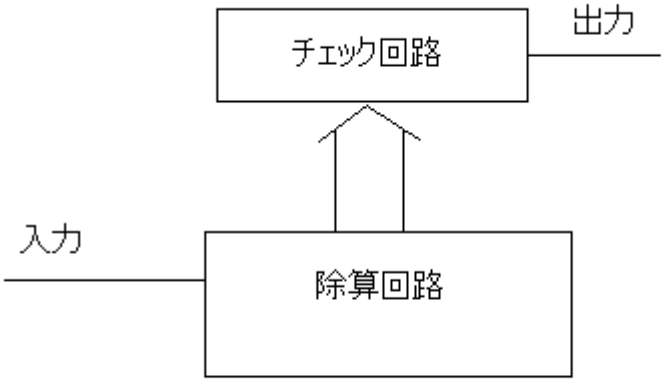


図 5.4 並列入出力受信機の構成

図 4.6 より除算の余りは入力クロック数の時間が経過した時のシフトレジスタの中身になる。チェック回路は受信データが R(X)まで全て入力されたときに、シフトレジスタの中身が全て０であるかどうかをチェックするための OR ゲートである。[1][5]

5.2.2 直列入出力受信機の設計

図 5.4 の構成を元にした VHDL 記述を図 5.5 に、テストベンチを図 5.6 に示す。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity CRCDEC is
   port (
         a: in std_logic;
      clk: in std_logic;
      reset: in std_logic;
      outa: out std_logic
      );
end CRCDEC;

architecture BEHAVIOR of CRCDEC is

signal foo0,foo1,foo2,foo3,foo4,foo5,foo6,outb,
foo7,foo8,foo9,fooa,foob,fooc,food,fooe,foof:std_logic;
signal count_in:std_logic_vector(4 downto 0);

begin

    process(clk,reset)begin
       if(reset='1')then
          foo0<='0';
```

図5.5 直列受信機のVHDL1/4

```
            foo1<='0';
            foo2<='0';
            foo3<='0';
            foo4<='0';
            foo5<='0';
            foo6<='0';
            foo7<='0';
            foo8<='0';
            foo9<='0';
            fooa<='0';
            foob<='0';
            fooc<='0';
            food<='0';
            fooe<='0';
            foof<='0';
        elsif(clk'event and clk = '1')then
            foo0 <= a  xor foof;
            foo1 <= foo0;
            foo2 <= foo1;
            foo3 <= foo2;
            foo4 <= foo3;
            foo5 <= foo4 xor foof;
            foo6 <= foo5;
```

図5.5 直列受信機のVHDL2/4

```
           foo7 <= foo6;
           foo8 <= foo7;
           foo9 <= foo8;
           fooa <= foo9;
           foob <= fooa;
           fooc <= foob xor  foof;
           food <= fooc;
           fooe <= food;
           foof <= fooe;


    end if;
  end process;
process(clk,reset )begin

       if (reset = '1') then
          count_in<="00000";
       elsif (clk'event and clk='1') then
          count_in <= count_in+'1';
       end if;
end process;
process(count_in)begin
    if(count_in ="10101")then
```

図5.5 直列受信機のVHDL3/4

```
   outa<=foo0 or foo1 or foo2 or foo3 or foo4 or foo5 or
          foo6 or foo7 or foo8 or foo9 or fooa or foob or
          fooc or food or fooe or foof;
    else null;
    end if;
end process;
end BEHAVIOR;
```

図5.5 直列受信機のVHDL4/4

foo0,foo1,foo2,foo3,foo4,foo5,foo6,foo7,foo8,foo9,fooa,foob
fooc,food,fooe,foof:図4.9における除算を行うためのシフトレジスタ

count_in:R(X)まで除算回路に入力された時にシフトレジスタの内容をチェックするためのタイミングをカウントする信号。

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.CRCDEC;


entity TESTBNCH is
end TESTBNCH;
architecture stimulus of TESTBNCH is
component CRCDEC is
```

図5.6 直列受信機のテストベンチ1/4

```
        port (
        a: in std_logic;
        clk: in std_logic;
        reset: in std_logic;
        outa: out std_logic
    );

end component;



signal a: std_logic;
signal clk: std_logic;
signal reset: std_logic;
signal outa: std_logic;



begin
    DUT: CRCDEC port map (
            a,
            clk,
            reset,
            outa
          );
```

図5.6 直列受信機のテストベンチ2/4

```
stimulus1: process
   begin
a<='0';wait for 10ns;
a<='1';wait for 20ns;
a<='0';wait for 20ns;
a<='0';wait for 20ns;
a<='1';wait for 20ns;
a<='0';wait for 20ns;
a<='0';wait for 20ns;
a<='0';wait for 20ns;
a<='1';wait for 20ns;
a<='1';wait for 20ns;
a<='0';wait for 20ns;
a<='0';wait for 20ns;
a<='1';wait for 20ns;
a<='0';wait for 20ns;
a<='0';wait for 20ns;
a<='1';wait for 20ns;
a<='1';wait for 20ns;
a<='1';wait for 20ns;
a<='0';wait for 20ns;
a<='0';wait for 20ns;
a<='1';wait for 20ns;
```

図5.6 直列受信機のテストベンチ3/4

```
        a<='1';wait for 20ns;


        end process stimulus1;
    stimulus2:process
    begin
clk<='0';wait for 10ns;

clk<='1';wait for 10ns;
end process stimulus2;

stimulus3:process
begin
  reset<='0';wait for 3ns;
  reset<='1';wait for 6ns;
  reset<='0';wait for 1800ns;
end process stimulus3;



   end stimulus;
```

図5.6 直列受信機のテストベンチ4/4

5.2.3 シミュレーション結果

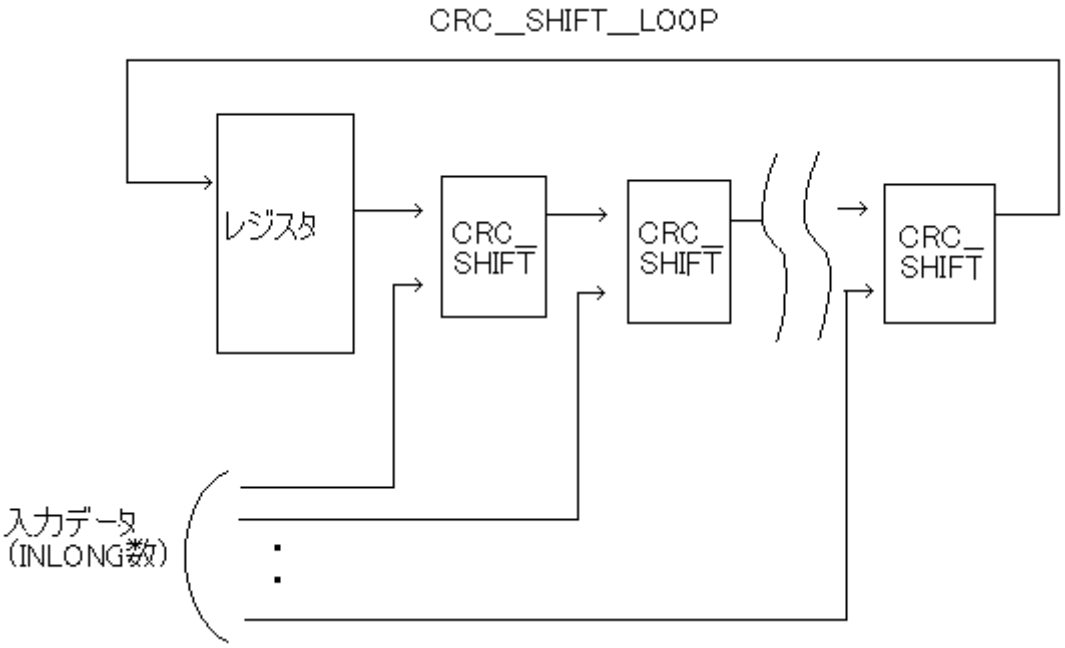
図 5.14 にシミュレーション結果を示す。

入力ビット数のクロックが経過したと同時にシフトレジスタの論理和を計算し再送要求が必要なら出力に‘１’、必要がない場合は‘０’を出力する。

5.3 並列入出力送信機の設計

5.3.1 並列入出力送信機の構成

シフトレジスタのクロック周波数が固定だとすると、処理能力を向上させるためには処理にかかるクロック数を減らす必要が出てくる。5.1 節 5.2 節の直列入力の送受信機では、回路規模は小さくて済むが、１ビットずつシフトを行っていくのでデータを出力するまでのクロック数が増えることになる。そこでデータの入出力幅を増やして、２ビットや４ビットのデータを１クロックで１度に処理できる回路を作成する。図 5.7 に並列入力の場合の CRC 回路を示す。他の送信機回路の部分は、直列入力の場合と同じとする。[1][2]



（CRC__SHIFT は入力１ビット分の処理を行う）

図 5.7 並列入力 CRC 演算回路

5.3.2 並列入出力送信機の設計

CRC 演算の並列入力を行うには、図 4.10 の入力１ビット分の処理を、組み合わせ回路として直列に入力ビット幅の数だけ呼び出す必要がある。プログラム中では入力１ビット分の処理の切り出しをサブプログラム CRC__SHIFT で行い、それを CRC_SHIFT_LOOP により複数回呼び出す回路を作成している。また、処理クロック数、入出力幅、生成多項式などのパラメータ化を行い、ワンタッチで自在に変更できるように generic 文により汎用性を上がるようにしている。

VHDL 記述を図 5.8 に、テストベンチを図 5.9 に示す。尚、生成することができる回路は INLONG 数が入力ビット数の 1/（2^N）（N は整数）のものとする。［1］［2］

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
entity CRC is
   generic(inlong:integer:=16;
          Ulong:integer:=32;
          Glong:integer:=16;

        g:std_logic_vector:="0000100000100001"
        --g:std_logic_vector:="0100000000000101"
        );
   port (
       a: in std_logic_vector(inlong-1 downto 0);
       clk: in std_logic;
       reset: in std_logic;
       outa: out std_logic_vector(inlong-1 downto 0)
       );

end CRC;

architecture BEHAVIOR of CRC is

function crc_shift(prev_fcs:std_logic_vector(16 downto 0);
```

図5.7 並列送信機のVHDL1/5

```
           aa:std_logic;
          x:std_logic)
        return std_logic_vector is
       variable adda:std_logic;
       variable next_fcs:std_logic_vector(16 downto 0);
     begin
       adda:=(prev_fcs(15) xor aa)and x;
       for I in 16 downto 1 loop
         if(g(16-I)='1')then
          next_fcs(I):=prev_fcs(I-1) xor adda;
         else
          next_fcs(I):=prev_fcs(I-1);
         end if;
       end loop;
        next_fcs(0):=adda;
       return next_fcs;
 end crc_shift;



function crc_shift_loop(prev_fcs:std_logic_vector(16
downto 0);
         data:std_logic_vector(inlong-1 downto 0);
         x:std_logic)
```

図5.7 並列送信機のVHDL2/5

```
   return std_logic_vector is
        variable next_fcs:std_logic_vector(16 downto 0);
        begin
          next_fcs:=prev_fcs;
        for I in 0 to inlong-1 loop

next_fcs:=crc_shift(next_fcs,data(inlong-1-I),x);
       end loop;
     return next_fcs;
 end crc_shift_loop;
 signal foo:std_logic_vector(16 downto 0);
signal count_in:integer range 0 to ulong/inlong;
signal a_in:std_logic;
signal b_in:std_logic;
signal x:std_logic;
signal out_a:std_logic_vector(inlong-1 downto 0);
signal fcs:std_logic_vector(inlong-1 downto 0);
begin
     process(RESET,clk)
       begin
       if(reset='1')then
     for I in 16 downto 0 loop
```

```
       foo(I) <= '0';
           end loop;

    elsif(clk'event and clk='1')then
       foo <= crc_shift_loop(foo,a,x);
       for I in inlong-1 downto 0 loop
fcs(I)<=foo(16-(inlong-I));
end loop;
                              end if;
 end process;

process(clk,reset)begin
   if(reset='1')then
      count_in<=0;
elsif(clk'event and clk='1')then
      count_in<=count_in+1;
 end if;
end process;
process(count_in)begin
if(count_in<=glong/inlong and count_in>=0)then
       a_in<='1';
    else a_in<='0';
```

図5.7 並列送信機のVHDL4/5

```
   end if;
end process;
process(count_in)begin
if(count_in<glong/inlong+8 and
count_in>=glong/inlong+1)then
   b_in<='1';
   else b_in<='0';
 end if;
end process;
process(count_in)begin
if(count_in<=glong/inlong-1 and count_in>=0)then
         x<='1';
        else x<='0';
  end if;
end process;
outa<=out_a;
process(a_in,b_in,a,foo)begin
for I in inlong-1 downto 0 loop
out_a(I)<=(a_in and a(I))or (b_in and fcs(I));


end loop;
end process;
 end BEHAVIOR;
```

図 5.7 並列送信機の VHDL5/5

inlong:データ入出力幅

Ulong:（データ＋FCS）出力データ長

Glong:FCS長

g:生成多項式を表す

out_a:ポート文のout信号は直接process文内で使用できないのでout_aに仮入力してから出力する信号。

fcs:サブプログラムの影響かfoo(15)～foo(0)を出力しようとすると１ビットシフトした値が出力されたので１ビット仮のレジスタ（foo(16)）を作成した。その時のfoo(16)から下位レジスタの格納場所　　＊生成多項式の最上位に‘０’が入っているのとサブプログラム内のループ数が１６downto０になっているのはレジスタを拡張したからである。

他の信号宣言は直列送信機の用途と同じとする。

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.CRC;

entity TESTBNCH is
end TESTBNCH;

architecture stimulus of TESTBNCH is
component CRC is
   port (
```

図5.8 並列送信機のテストベンチ1/3

```
       a:std_logic_vector(15 downto 0);
       clk,reset: in std_logic;
       outa: out std_logic_vector(15 downto 0)

   );

end component;
signal a:std_logic_vector(15 downto 0);
signal clk,reset: std_logic;
signal outa: std_logic_vector(15 downto 0);

begin
   DUT: CRC port map (
          a,clk,reset,
          outa
        );

   stimulus1: process
          begin
    a<="0100011000101110";wait for 70ns;
      end process stimulus1;
   stimulus2:process
   begin
```

図5.8 並列送信機のテストベンチ2/3

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


end process stimulus2;


stimulus3:process
begin
       reset<='0';wait for 3ns;
 reset<='1';wait for 6ns;
reset<='0';wait for 1800ns;
end process stimulus3;
end stimulus;
```

図 5.8 並列送信機のテストベンチ 3/3

5.3.3 シミュレーション結果

シミュレーション結果を図 5.12(１６ビット入出力)と図 5.13（４ビット入出力）に示す。INLONG を１６としてテストベンチで、"0100011000101110“を入力すると演算は１クロックで算出し、データ出力も２クロックで終了する。また、INLONG を４とすると８クロックで全ての処理が終了する。

5.4 並列入出力受信機

5.4.1 VHDL による並列入出力受信機の設計

並列入出力の受信機を設計する場合も 5.2 節の除算回路の入力１ビットごとの処理を切り出して、それを複数回呼び出すプログラム（並列入出力の時のサブプログラムの呼び出し方と同じ）とする。VHDL 記述を図 5.9 に、テストベンチを図 5.10 に示す。［1］［2］

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity CRCDECD is
         generic(inlong:integer:=2;
                ulong:integer:=32;

        g:std_logic_vector:="0001000000100001"
        --g:std_logic_vector:="1000000000000101"
   );
   port (

       a: in std_logic_vector(inlong-1 downto 0);
       clk,reset: in std_logic;
       outa: out std_logic

   );

end CRCDECD;

architecture BEHAVIOR of CRCDECD is
function jo(b:std_logic_vector(15 downto 0);
          da:std_logic)
```

図5.9 並列受信機のVHDL1/4

```
   return std_logic_vector is
     variable yojo:std_logic_vector(15 downto 0);
  begin
      for I in 15 downto 1 loop
        if(g(15-I)='1')then
          yojo(I):=b(I-1) xor b(15);
        else
          yojo(I):=b(I-1);
        end if;
      end loop;
      yojo(0) := (b(15)xor da);
     return yojo;
end jo;

function jozan (b:std_logic_vector(15 downto 0);
            data:std_logic_vector(inlong-1 downto 0))
     return std_logic_vector is

   variable next_b:std_logic_vector(15 downto 0);
  begin
        next_b:=b;
    for I in 0 to inlong-1 loop
          next_b:=jo(next_b,data(inlong-1-I));
```

図5.9 並列受信機のVHDL2/4

```
    end loop;
      return next_b;
end jozan;
signal foo:std_logic_vector(15 downto 0);
signal count_in:integer range 0 to ulong/inlong+1;
begin

     process(RESET,CLk )
        begin
        if (RESET = '1') then
          for I in 15 downto 0 loop
             foo(I) <='0';
          end loop;
      elsif (clk'event and clk='1') then
          foo<= jozan(foo,a);

                  end if;
    end process ;
    process(clk,reset)begin
        if(reset='1')then
         count_in<=0;
        elsif(clk'event and clk='1')then
         count_in<=count_in+1;
```

図5.9 並列受信機のVHDL3/4

```
      end if;
   end process;


   process(count_in)
  variable tmp:std_logic;


    begin
    tmp:='0';


   if(count_in=ulong/inlong+1)then
    for I in 0 to inlong-1 loop
     tmp:=tmp or foo(I);
    end loop;
     outa <= tmp;
    else null;
    end if;
   end process;
end BEHAVIOR;
```

図5.9 並列受信機のVHDL4/4

tmp:I ループによるシフトレジスタ foo の論理和の途中経過と結果を表す。

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
```

図5.10 並列受信機のテストベンチ1/4

```
use work.CRCDECD;


entity TESTBNCH is
end TESTBNCH;


architecture stimulus of TESTBNCH is
component CRCDECD is
    port (
        a: in std_logic_vector(1 downto 0);
        clk: in std_logic;
        reset: in std_logic;
        outa: out std_logic    );

end component;



signal a: std_logic_vector(1 downto 0);
signal clk: std_logic;
signal reset: std_logic;
signal outa: std_logic;signal done: boolean := false;


begin
    DUT: CRCDECD port map (
```

図5.10 並列受信機のテストベンチ2/4

```
          a,
          clk,
          reset,
          outa
        );
stimulus1: process
          begin
       a<="00";wait for 20ns;
       a<="11";wait for 40ns;
       a<="00";wait for 40ns;
       a<="10";wait for 40ns;
       a<="10";wait for 40ns;
       a<="11";wait for 40ns;
       a<="10";wait for 40ns;
       a<="01";wait for 40ns;
       a<="10";wait for 40ns;
       a<="01";wait for 40ns;
       a<="10";wait for 40ns;
       a<="01";wait for 40ns;
       a<="00";wait for 40ns;
       a<="01";wait for 40ns;
       a<="11";wait for 40ns;
       a<="01";wait for 40ns;
```

図5.10 並列受信機のテストベンチ3/4

```
        a<="11";wait for 40ns;
        end process stimulus1;
stimulus2:process
    begin
        clk<='0';wait for 20ns;
        clk<='1';wait for 20ns;
end process stimulus2;

stimulus3:process
begin
        reset<='0';wait for 3ns;
        reset<='1';wait for 6ns;
        reset<='0';wait for 1800ns;
end process stimulus3;
   end stimulus;
```

図5.10 並列受信機のテストベンチ4/4

5.4.2 シミュレーション

シミュレーション結果を図5.15に示す。INLONGを２としてテストベンチにより２ビットのデータを除算回路に入力していくと、１６クロック目にシフトレジスタの中が全て０になる。このタイミングでシフトレジスタの論理和を取ると、１５クロック目のデータの論理和を取ってしまうので、１７クロック目に論理和の計算を行い再送要求の有無を確かめる。

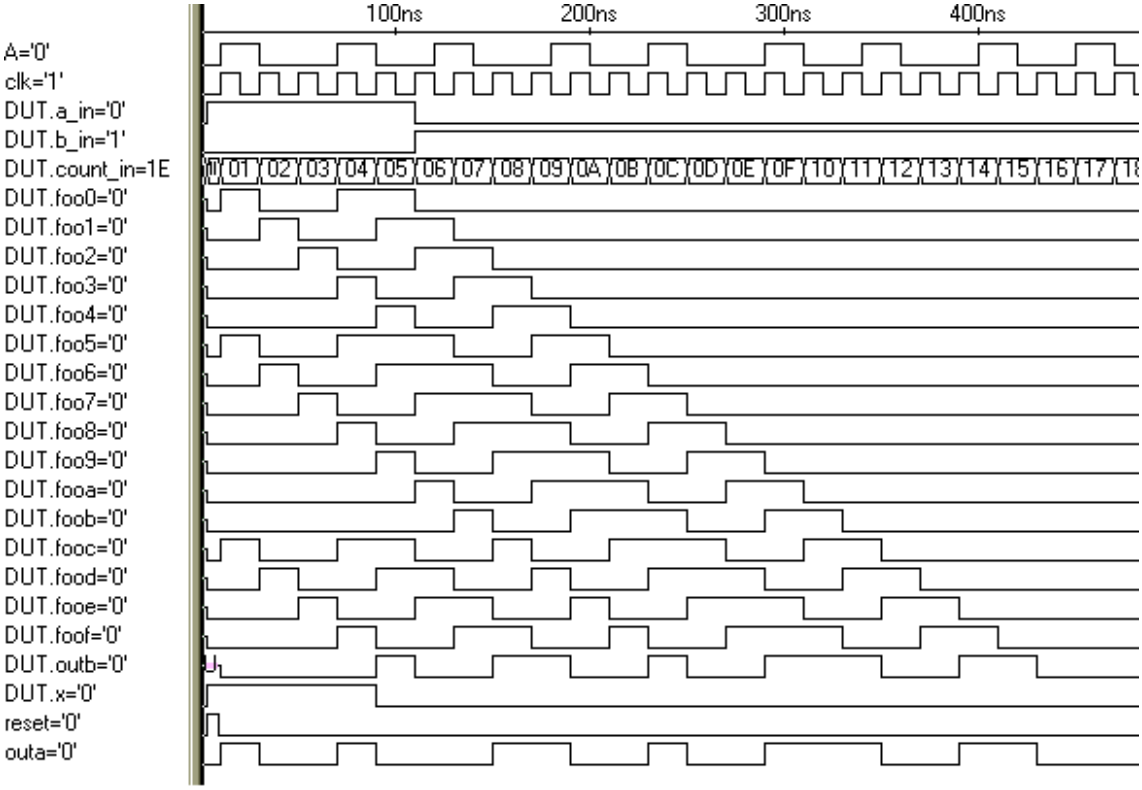


図5.11 直列送信機シミュレーション波形

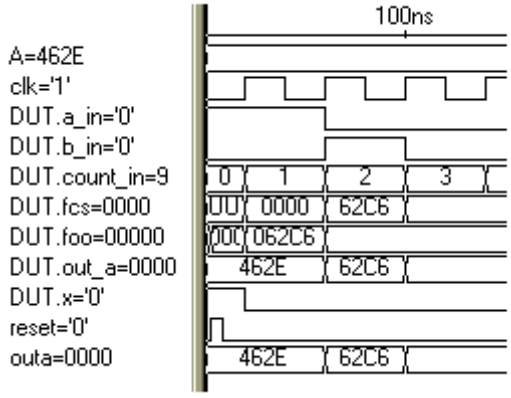


図5.12 並列送信機シミュレーション波形（１６ビット入出力）

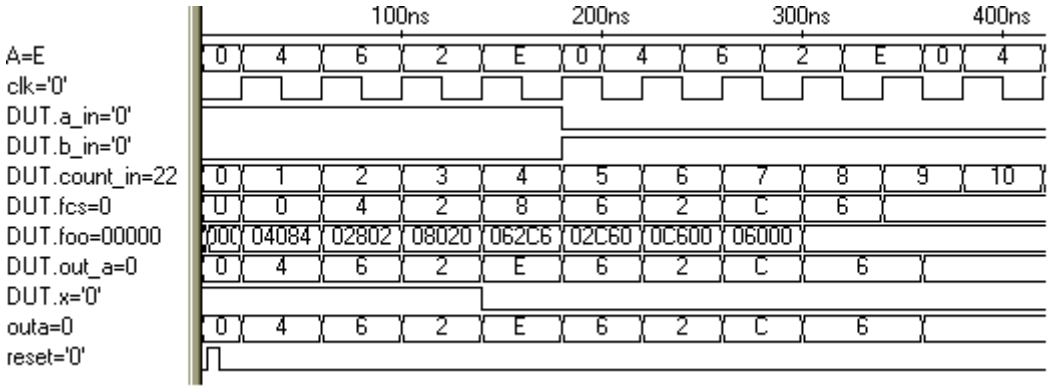


図5.13 並列送信機シミュレーション波形（４ビット入出力）

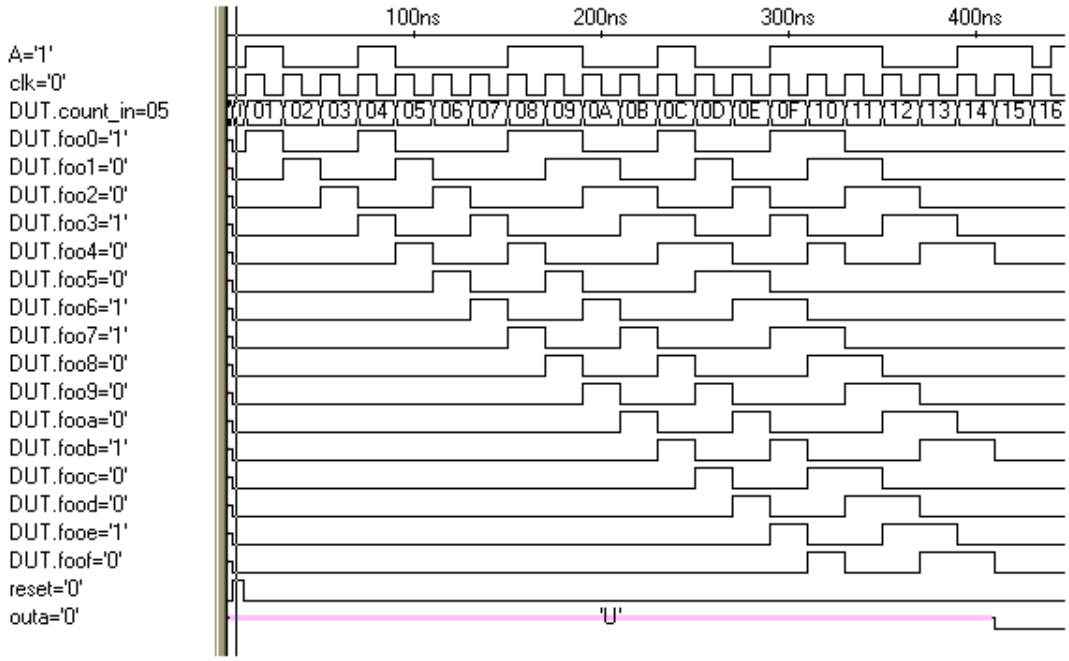


図5.14 直列受信機のシミュレーション波形

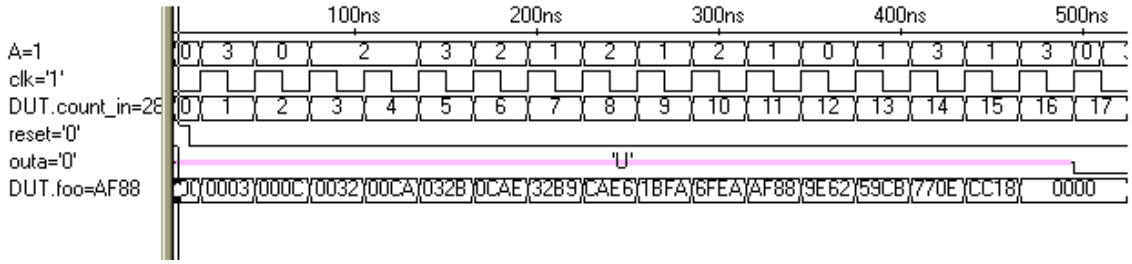


図5.15 並列受信機のシミュレーション波形

第６章　おわりに

卒業研究を通してCRC回路に使われている乗算回路や除算回路の簡易的な作り方を学ぶことができた。また今回の課題であったVHDLの記述も直列回路から並列回路へと序所にステップアップしていくことによって、回路設計の記述方法を学ぶことができた。また、所々に自分の思っていたシミュレーション結果と違う波形を生成したが（信号宣言の計算の時にDフリップフロップを生成してしまっていた）間違いの理由を意識することによってプログラムの修正も適切に行うことができた。今後も過程と結果が一致するようにもっとVHDLの記述に関する理解を深めていきたい。また、今回作成した回路はテストベンチにより入力とクロックのタイミングを同期させていたが、実際にFPGAに実装するとなると入力とクロックを同期させる回路が新たに必要となるので、この作成も今後の課題としたい。

謝辞

本研究を進めるにあたり、日頃より懇切丁寧なご指導してくださいました、矢野政顕教授に心から感謝致します。また、日頃から多くの助言を頂きお世話になりました原央教授、橘昌良助教授、他各先生方に厚くお礼申し上げます。最後に本研究を進めるにあたり助言を頂いた大学院生の石川純平氏と松村暢也氏他、同研究室の先輩方に心から感謝致します。

参考文献

［1］「VHDL によるハードウェア設計」 長谷川裕恭　CQ 出版

［2］「HDL による高性能ディジタル回路設計」 盛岡澄夫　CQ 出版

［3］「通信ネットワーク」 荒谷孝夫　東京電機大学出版局

［4］「ISDN　Q&A」 池田博昌　花澤隆　コロナ社

［5］「データ通信」 猪瀬博　山本厳　産報