平成 12 年度

学士学位論文

データ駆動方式による
並列処理形 CRC 処理に関する研究

A Study on Parallel CRC Processing Based on the Data-Driven Scheme

1010455　　森安　亮

指導教員　　岩田　誠

2001 年 2 月 5 日

高知工科大学 情報システム工学科

要 旨

データ駆動方式による
並列処理形 CRC 処理に関する研究

森安　亮

内容概要

巡回冗長符号化/復号化処理はルータの処理で高性能化のボトルネックの１つである。本論文では特に高速処理性能が要求される誤り訂正処理を対象として研究を行った。CRC はシフトレジスタにより簡単なハードウェアで実現可能なため、ATM と Ethernet などに広く利用されている。しかしながら、シフトレジスタはその逐次的動作が原因で高速化が望めない。そこで、本研究では CRC 処理の並列化に着目しデータ駆動型プロセッサ DDMP(Data-Driven Multimedia Processor) 上でソフトウェア的に高速実現する方式を提案した。また、ATM と Ethernet で共通して使用できるような複合命令を DDMP に追加して更なる高速化を図った。提案手法の実現可能性を確認するために、作成したプログラムのデータ駆動プロセッサ上での動作性能を確認した。これは ATM:OC-3 用の CRC-8 をデータ駆動型プロセッサコア１個で十分処理可能な性能であり、その結果プロセッサコアを増設すればより高速なリンクにも対応可能になることが明らかになった。

キーワード　CRC，データ駆動、ATM、Ethernet

Abstract

A Study on Parallel CRC Processing Based on the Data-Driven Scheme

Ryo Moriyasu

Abstract

CRC(Cyclic Redundancy Check)encoding/decoding process is one of the bottlenecks of high speed routers. The research in this paper, is enhancing the performance of error correction which requires high throughput, since CRC can be realizable by small hardware of a shift register, that is widely used for ATM, Ethernet, etc. However, a shift register is difficult to improve its performance in the speed, because of its native successive operations. This research is focused on a high-speed realization in software on DDMP(Data-Driven Multimedia Processor) utilizing parallelisms of CRC. I propose a new compound instruction of DDMP which can be used in common in ATM and Ethernet. And further improvement in speed was achieved. In order to find the possibility of the proposed method, I wrote a pilot program to perform CRC functions on the data-driven processor. The performance of the trial about CRC-8 on only one processor core of DDMP is competitive with ATM:OC-3(155Mbps). So, it became clear that nearly coming high-speed links, are meet to DDMP systems.

key words CRC, DDMP,ATM,Ethernet

目次

第 1 章　序論 **1**

第 2 章　パケット通信向け CRC 処理 **5**

2.1　緒言 5

2.2　CRC 処理について 5

2.2.1　CRC 処理内容 6

2.3　現行の CRC 処理の問題点 6

2.3.1　基本的な CRC 演算 (逐次処理形 CRC) 7

2.4　CRC 高速演算をねらいとする各構成 8

2.4.1　全展開処理型 CRC 処理 8

2.4.2　パイプライン処理型 CRC 処理 9

2.5　各 CRC 処理法を利用したデータ駆動型プロセッサ上での実現法 10

2.5.1　逐次処理形 CRC 処理を DDMP 上で実装 11

2.5.2　全展開処理形を DDMP により実装した場合 13

2.5.3　パイプライン処理形 CRC を DDMP により実装した場合 14

2.6　結言 15

第 3 章　並列処理形 CRC 処理 **16**

3.1　緒言 16

3.2　並列処理形 CRC 処理 16

3.2.1　剰余転送並列処理形の原理 16

3.3　DDMP 上で並列 CRC 処理の実現 17

3.3.1　データ駆動型並列 CRC 処理プログラムの要件 17

3.3.2　並列処理形 CRC のシフトレジスタ 18

3.3.3 DDMP 並列処理形 CRC 処理プログラム 20
3.4 並列 CRC 処理向きの複合命令 22
3.4.1 Ethernet での CRC 処理について 22
3.4.2 ATM と Ethernet での共通命令 23
3.5 シミュレーション手法 26
3.6 DDMP 上での並列 CRC 処理の評価 26
3.7 結言 29

第 4 章 結論 **30**

謝辞 **32**

参考文献 **33**

付録 A 並列処理形 CRC 算出法 **34**

図目次

2.1 ATM のセルフォーマット 7

2.2 CRC 演算回路の基本形 (線形帰還形シフトレジスタ) 8

2.3 全展開形 9

2.4 パイプライン処理形 10

2.5 基本的 CRC 処理プログラム 12

2.6 全展開処理型 CRC プログラム 13

2.7 パイプライン処理型 CRC プログラム 14

3.1 剰余転送処理形式の動作フロー 17

3.2 剰余転送処理形式のバイト処理形並列 CRC 剰余算出回路 19

3.3 1 ビット入力並列 CRC 処理プログラム 20

3.4 8 ビット入力並列 CRC 処理プログラム 21

3.5 イーサネットフレームフォーマット 23

3.6 Ethernet の DDMP 並列処理形 CRC プログラムイメージ 25

3.7 DDMP 新命令　SPS(Set Previous State) 26

表目次

3.1 CRC 処理性能比較表 27
3.2 ATM 性能比較表 28
3.3 複合命令を使用した場合の CRC 処理性能比較表 28
3.4 CRC 処理性能向上率 28

第 1 章

序論

情報通信は今、大きな変革期を迎えてきている。その主役は、いわゆる音声、画像などのマルチメディアである。従来の電話を主体にしたネットワークから、我々を取り巻くさまざまな情報を、時間や空間を越えて、自由に伝達できるネットワークに、大きく変わろうとしている。ネットワークが広帯域化し、高品質音声、高精彩映像のための超高速データ転送など制約のない通信を実現するためには、光ファイバ伝送、高速大容量スイッチング、符号化技術やトラフィック制御技術、ネットワーキング技術などが重要な位置を占める。

インターネットは新しい通信インフラを支える技術として地位を確立したといえる。しかしながら、次世代のマルチメディアネットワークの中核技術となるには、依然残された技術課題は多い、インターネットでのトラフィックの増大に伴い、大容量データを高速処理することが可能な高速ルータの開発が急がれる。情報伝送路の転送速度が高速になってもルータの処理速度が遅ければ意味がない。ルータの処理速度を速くするためにルータ処理のいくつかのボトルネックを改善する必要がある。例えば、IP アドレス検索を速くする方法、また、データ転送誤り訂正処理を高速処理する方法などがある。我々の研究室全体の目標は、DDMP（データ駆動型マルチメディアプロセッサ）を使用して、高速ルータを開発することである。IP アドレス検索の高速化は我々の研究室で進展しつつある。そこで、本研究では情報伝送でのデータ伝送誤り処理を取り上げる。ATM、Ethernet で使われている CRC チェックを取り上げ、CRC 処理を並列処理化することで高速化を促すことにした。並列形 CRC 処理プログラムを作成するにあたり、並列化に適した、DDMP プログラム開発ツールによりプログラムを作成する。この研究では CRC 処理をソフトウェアで実装することを最終目的とする。将来的にはソフトウェアで実装したものをハードウェアで実装し、CRC 処

理専用のチップを開発しルータに搭載する。この高速 CRC 処理チップを搭載することで今までルータのボトルネックであった転送データ誤り処理速度が飛躍的に向上する。

CRC(Cyclic Redundancy Check) とは巡回冗長検査のことで、データ伝送やディスク、テープなどへの読み書きにおいて、データが正しく伝送 (読み書き) できたかどうかを検査するための方法の１つである。パリティやチェックサムでは、２つ以上のデータが同時に変化を起こすと、それらが相殺されて正常状態とみなされてしまうことがあるが、CRC では、この確率をできるだけ少なくするために、単純な合計ではなく、CRC 生成多項式と呼ばれる巡回符号を用いて、シフトや加算などを組み合わせた方式で計算を行なう。CRC の値は通常は 16bit か 32bit のものが使われ、伝送する一定のビット列を１つの数字とみなし、ある数で割った剰余を計算し、その剰余をデータに付加して伝送する。 受信側では同じ数で割って、剰余が０なら誤り無し、剰余があれば誤り検出とする。CRC の計算は単純な加算だけではないので、ソフトウェアで行なうと負荷がチェックサムよりも大きいが、ハードウェアで実装すると簡単なため、ディスクコントローラや通信用 LSI などでは CRC 方式がよく使われている。

CRC の特徴

1. 任意長のデータ転送で利用される
2. ハードウェアで実現しやすく、高速である
3. 連続した誤りに強いこと

本研究で採用するデータ駆動方式とはデータが揃い次第計算が行われる方式である。現在広く使われているプロセッサは、いわゆるノイマン型処理方式にもとづき、データをひとつひとつ順番に処理していく逐次処理である。これに対し、DDMP（Data-Driven Multimedia Processors）は、一度に多数のデータを高速処理する、新しい非ノイマン型のプロセッサである。非ノイマン型データ駆動を実現した DDMP は、メイン CPU を補完して、音声、動画、静止画像など、大量のデータを低消費電力かつ高速処理を可能にするプロセッサである。

DDMP の特徴を挙げると次のようになる。

1. 低消費電力化

一般的にプロセッサやＬＳＩの内部回路を動作させる場合、データの受け渡しやメモリへの書き込みの同期を取る必要がある。これらの同期を取る方法の１つとして、すべての回路を共通の信号で結合する方法がある。これをクロック信号と呼ぶ。ＣＰＵなどの速度を計る場合に 200 MHｚ、133 MHｚなど周波数の単位を使う。周波数が高ければ高いほど処理速度もあがる。しかし、内部の大半の回路がこのクロックでつながれているため、周波数が上がれば当然内部回路全体の消費電力も上がってゆく。ノイマン型のプロセッサは、クロックが常時動作しており、データの入出力制御を行うとともに、プログラムを１ステップ毎に実行している。データ処理を高速化するには、クロック周波数を高める必要があり、そして、クロック周波数に比例して消費電力が増加してゆく。しかし、DDMP ではデータに制御機能をもたせて、演算部にデータが揃った時だけ作動するクロックレス構造により、低消費電力でプロセッサを動作させることができる。

2. 高速処理（600 ～ 16000 MOPS）MOPS=Million Operation Per Second

複数のプログラムが読み出される並列処理方式により、高速演算性能を実現する。画像認識など高度な処理もリアルタイムで実行することができる。

3. 複数の処理を同時実行可能

並列処理方式採用により、複数のマルチメディア処理を同時に実行が可能である。

4. ソフトウェアによる機能拡張／性能向上が可能

DDMP を用いれば、ソフトウェア処理だけで専用 LSI と同等の機能を実現できるため、容易に機能拡張や性能向上が図れ、大幅なコスト削減が可能である。また専用 LSI の開発に比べ短期間での開発が可能なため、モデルチェンジや仕様変更を容易に行うことができる。

データ駆動型処理方式は、ある処理に必要なデータが全て揃い次第処理が実行される方式であるため、同時並行処理性、パイプライン並列性を活かした細粒度の並列処理が可能であ

る。また、メモリからプロセッサへの命令とデータの供給を独立に行うことができる特徴がある。これらの特徴により逐次型処理方式で問題となっていたパイプラインハザードや副作用の問題はデータ駆動処理方式においては原理的に発生しない。しかし、データ駆動型処理を用いれば、どんなアプリケーションに対しても効率的な並列処理ができるわけではない。データ駆動型処理方式ではそれぞれの処理でその実行に必要なデータを調べる必要があるため、待ち合わせの部分でデータが揃うのを待っている処理が増えると、実行可能なデータセットを見つけるために遅延が発生し、プロセッサの能力を低下させる。そこでデータ駆動プログラムではこの遅延を避けるためにプロセッサに処理ごとのデータセット、処理の並列性を考慮してプログラムを複数のプロセッサに分割配置する必要がある。

第2章では、パケット通信向けのCRC処理についての概要およびATMでのCRC演算方法を具体的な手法をいくつか挙げて説明し、これらの高速化のポイントを明らかにする。第3章ではCRC処理の並列化したものをDDMP上で実装することを提案し、並列CRC処理プログラムをDDMP-4Gにより開発する。ATMのCRC処理とEthernetのCRC処理を比較検討しDDMP上で共通して使えるような複合DDMP命令を検討し、新命令として並列処理形CRC処理プログラムに追加する。そして、開発したプログラムのスループットを計測し逐次処理形CRCおよび複合命令を追加した場合の並列処理形CRCの処理性能などを比較する。最後の4章ではまとめ、および今後の課題、展望について述べる。

第 2 章

パケット通信向け CRC 処理

2.1 緒言

本章では ATM における CRC 処理の概要について述べ、CRC 処理の高速化のポイントを明らかにする。そして、各種高速化をねらいとする方式について説明し、問題点を明らかにする。各種 CRC 処理を DDMP-4G 上でプログラムの実装を行った場合、DDMP 上で CRC 処理を高速に行うために問題点を明らかにし、どのように改善してゆくかを述べる。

2.2 CRC 処理について

巡回冗長検査符号 CRC はシフトレジスタによって簡単なハードウェアで実現可能なため、ATM や Ethernet などに広く利用されている誤り訂正符号である。しかしながら、シフトレジスタはその逐次的動作が原因で高速化が望めない。レジスタとは 8 ビット、16 ビット、32 ビット等のデータを 一時的に記憶する一種のメモリのことであり、シフトレジスタは、記憶しているデータの桁を左右にシフトさせることができるレジスタである。

ATM では CRC-8 という CRC 処理が行われる。この方法は伝送路での符号誤りによってヘッダの値が変化する。これを検出するために CRC-8 ビットを使って受信セルのヘッダ内に生じた符号誤りを検出、訂正するものである。CRC-8 は、ヘッダ中の 1 ～ 3 ビットの誤りと 8 ビット以下のバースト誤りの検出ができる。また 1 ビットの誤りを訂正できる能力

を持つ。剰余 (シンドローム) が零でない場合に、符号誤りと認識し、1 ビット誤りの訂正は剰余の値によって行われる。

2.2.1 CRC 処理内容

1. 送信側では、伝送するセルごとに、ヘッダ先頭 4 バイトのビット列を生成多項式 G (X)* で割り、剰余多項式 R(X) を求める。(CRC 演算)ATM のセルフォーマットを図 2.1 に示す。
2. 剰余多項式 R(X) の係数に"01010101"を mod2 で加算し (排他的論理和をとる) 、結果を HEC フィールドに収容する。
3. 受信側では、受信したビット列のある 1 バイトに"01010101"を mod2 で加算する。
4. 加算後の 1 バイトとそれ以前の 4 バイトを合わせた 5 バイトのビット列を生成多項式で割り、剰余を求める。

＊ 生成多項式としては、3 ビット以下の誤り検出能力があること、1 ビット誤り訂 正能力があること、および奇数個の誤りを検出できることから

$G(X) = X^8 + X^2 + X + 1 \cdots (1)$

が用いられる。また、送信側、受信側でビットパターン"01010101"を mod2 で加算する処理はコセット (coset) と呼ばれるもので同期位置の誤判定を避けるために用いられる。

2.3 現行の CRC 処理の問題点

前節で述べたように、巡回冗長検査符号 CRC はシフトレジスタによって簡単なハードウェアで実現可能なため、ATM や Ethernet などに広く利用されている。しかし、CRC 処理を行うには、入力ビット列に対する演算範囲を 1 ビットずつシフトしながらクロックごと

に剰余を求めることのできる、40 ビット入力の高速 CRC 演算回路が必要になる。生成多項式に基づく基本構成を図 2.2 に示す。この回路は 1 演算結果を得るために毎回、8 個のフリップフロップの零リセットと、40 ビットのビット列の入力が必要である。そのため逐次的な処理が 1 ビットごと行われる現行のシフトレジスタを用いた CRC 処理は高速化に適さない。

2.3.1　基本的な CRC 演算 (逐次処理形 CRC)

図 2.2 は基本的な CRC 演算回路 (線形帰還形シフトレジスタ) を示す。この CRC 演算回路は EXOR の特徴を活かしたもので同じ内部状態に 2 度同一の入力データを加えると、その入力データは加えなかったのと同じ結果となることに基づいたものである。すなわち、最初にある入力データ列に対する剰余が得られた後、それ以後は CRC 演算回路に入力された列の内、最も古いデータを打ち消し、新たなデータを入力することによりスタート点が 1 ビットシフトした入力データ列に対する生成多項式による剰余を得るものである。

図 2.2 は生成多項式 $G(X) = X^8 + X^2 + X + 1$ の場合の複号化回路であり、動作説明をすると、受信信号（40 ビット）は 1 ビットずつ CRC 演算回路に入力される。受信信号が入力され終わった段階（40 回シフト＋演算の処理が終わった段階）でシフトレジスタに格納されているデータを割り算の剰余（シンドローム）として取り出すことをあらわしている。

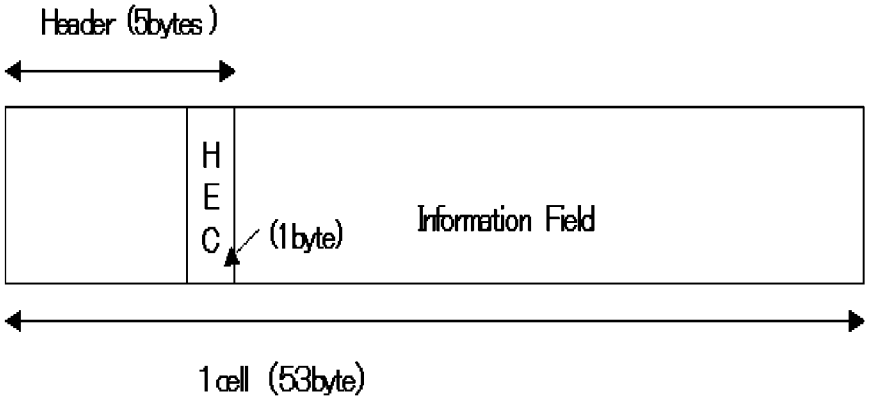

図 2.1　ATM のセルフォーマット

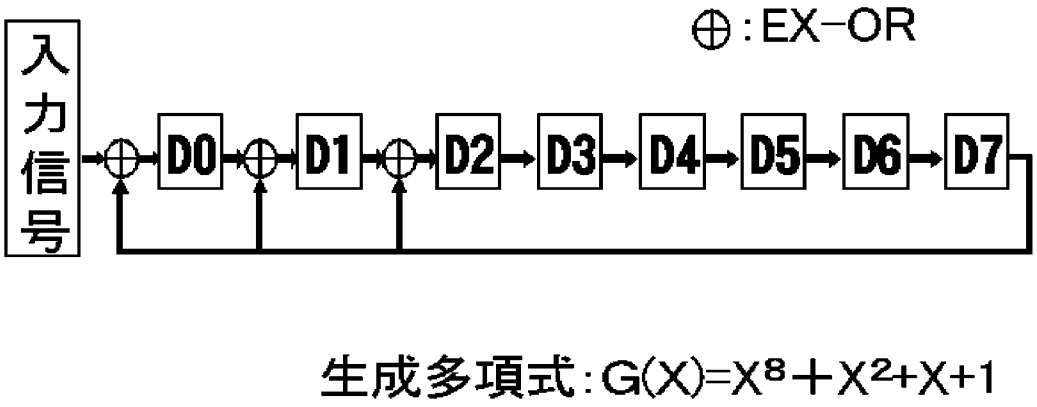

図 2.2　CRC 演算回路の基本形 (線形帰還形シフトレジスタ)

2.4　CRC 高速演算をねらいとする各構成

現在 CRC 処理を高速に処理する線形帰還形シフトレジスタを基にした各種構成が提案されている。ここでは、現在提案されている全展開処理形、パイプライン処理形の CRC 処理の説明と DDMP 上で実装した場合、どれくらいの性能が期待できるのかを検証し高速化の問題点について述べてゆく。

2.4.1　全展開処理型 CRC 処理

図 2.2 での回路の計算で D0 ～ D7 のフリップフロップの状態を展開したものが全展開処理型の処理である。D0 ～ D7 のフリップフロップの状態を展開したものを図 2.3 に示す。1 クロックごとに演算範囲をシフトした符号長に等しい長さの入力データ列を入力し、生成多項式による剰余を出力する EXOR を用いたもので縦続接続される EXOR の段数が多くなる。ATM の場合、符号長は長くないのである程度の処理速度は期待できるが、Ethernet のような符号長が長い場合には、高速処理に適さない。全展開処理形を DDMP 上で実装した場合、命令数が増加し CRC 処理プログラムの性能を低下させてしまうことは明らかである。そのことから判るように DDMP 上で CRC 処理を高速に行うためには、まず命令数を

なるべく少なくしてナノプロセッサー 1 つで CRC 処理を実現することと、全展開処理の場合は最初のデータが入ってから最後のデータとの排他的論理和を計算するまでノードで、待ち合わせの遅延が発生する危険性がある。

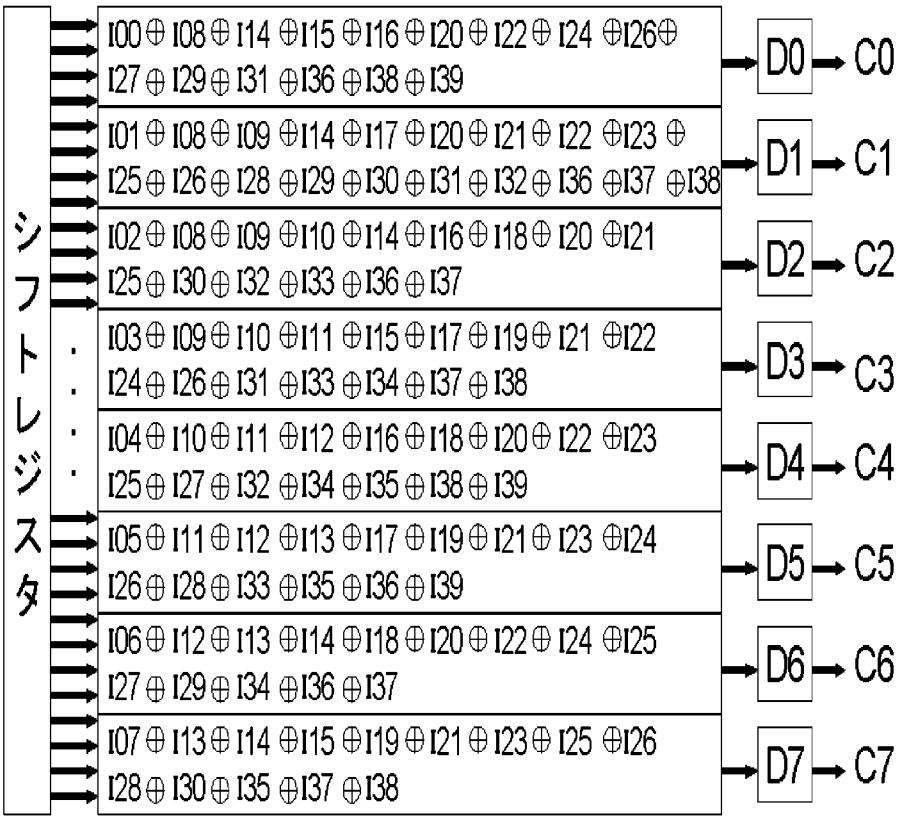

図 2.3　全展開形

2.4.2　パイプライン処理型 CRC 処理

パイプライン処理形の CRC 処理を図 2.4 に示す。図 2.2 の CRC 処理の基本形である線形帰還形シフトレジスタを用いて、シフトレジスタ内をフィードバックループで値を戻すのではなく次の段階のフリップフロップに演算結果を送る構造である。このパイプライン方式の場合には、1 ビット次の検査位置で演算中の結果が、1 段前のフリップフロップに格納されている。パイプライン処理形の場合、高速処理には適しているが、線形帰還形シフトレジスタを多段に接続して構成されているで、DDMP で実装した場合、命令数が明らかに 1 つ

のプロセッサコアではまかないきることができず複数のプロセッサコアを必要とするため、DDMP で実装する場合には 8 段のパイプライン構造をフィードバックループにより接続して半パイプライン処理形として設計すれば 1 つのプロセッサコアで CRC 処理プログラムを動作させることが可能である。

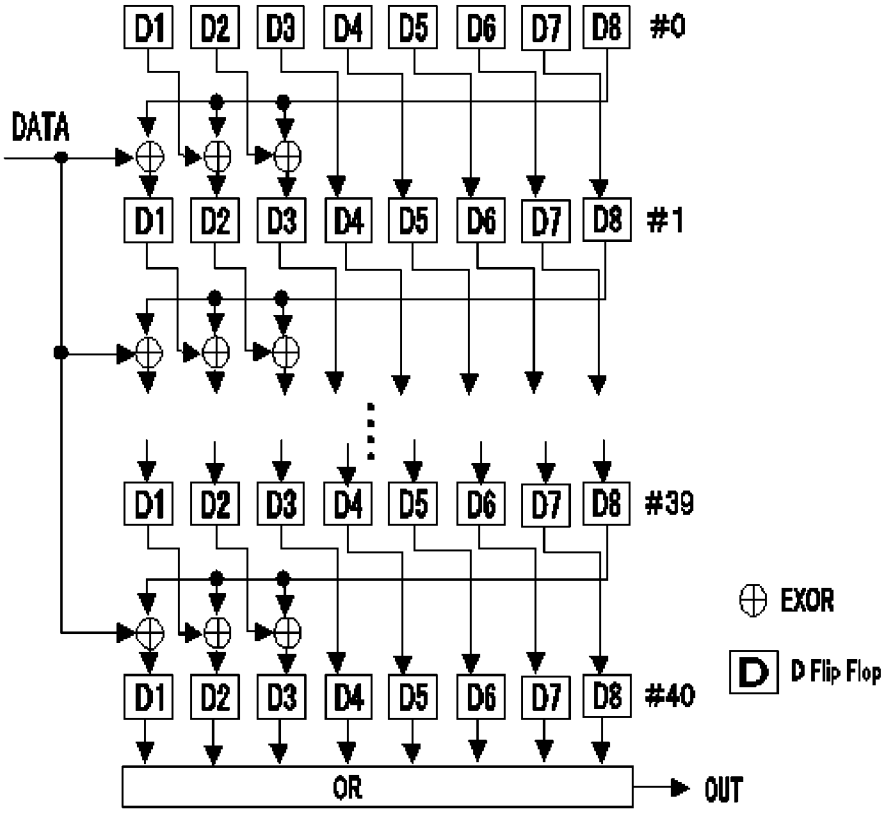

図 2.4　パイプライン処理形

2.5　各 CRC 処理法を利用したデータ駆動型プロセッサ上での実現法

前節で説明した線形帰還形シフトレジスタを用いた逐次処理形 CRC 処理、線形帰還形シフトレジスタを用いたパイプライン処理形、また、線形帰還形シフトレジスタのフリップフロップの内部状態を利用した全展開処理形について DDMP 上で実装した場合のフローグラフの構成を説明する。

2.5.1　逐次処理形 CRC 処理を DDMP 上で実装

2.3.1 で説明した回路をそのまま動作させることができるプログラムについて説明をする。入力データ 40 ビットと生成多項式 G(X) を基本とする図 2.2 の回路を DDMP-4G で実装する。このプログラムは 40 ビットをシフトしながら入力し剰余を求めて出力できるものとする。フローグラフを図 2.5 に示す。図 2.5 は図 2.2 で示した基本的な CRC 演算の 1 ビットシフトの状態をフローグラフに表したものである。つまり、このフローグラフをフィードバックループにより 40 個接続することにより 1 回の CRC 処理が行われることになる。この DDMP-4G では最大 12bit の計算であるので、CRC 処理に適した 8bit 入力で処理を行うプログラムである。更に 1 bit ごと入力のプログラムをそれぞれ作成した。本研究では並列 CRC 処理のプログラムを DDMP で実装したとき、性能向上のため DDMP 複合命令の追加を行った。並列 CRC 処理の性能を比較するため、逐次処理形 CRC 処理の場合にも DDMP の命令を復合化し性能を測定した。図 2.5 に示すフローグラフはプログラムの CRC 演算モジュールを表してある。逐次処理形 CRC 処理プログラムの復合化は 1 ビットシフトを計算できる図 2.5 のモジュールを 1 つの複合命令とし CRC 処理の高速化を図っている。しかしながら、このプログラムを DDMP 新命令として追加したとしても汎用性が失われてしまう。なぜならば、ここで複合化した命令は ATM の生成多項式に基づいてシフトレジスタが作られていて ATM の CRC 処理の中でしか活用することができない。そのためこの方法で DDMP 高速ルータを開発するためには Ethernet の生成多項式に基づいたシフトレジスタの DDMP 複合命令を作らなければならない。そのため、余分なプロセッサを使い逆にルータの質を下げてしまう。ここでの考えなければならないのは、並列処理形 CRC 処理を DDMP で実現する場合には ATM と Ethernet で十分汎用性を持たせることが大切である。

2.5　各 CRC 処理法を利用したデータ駆動型プロセッサ上での実現法

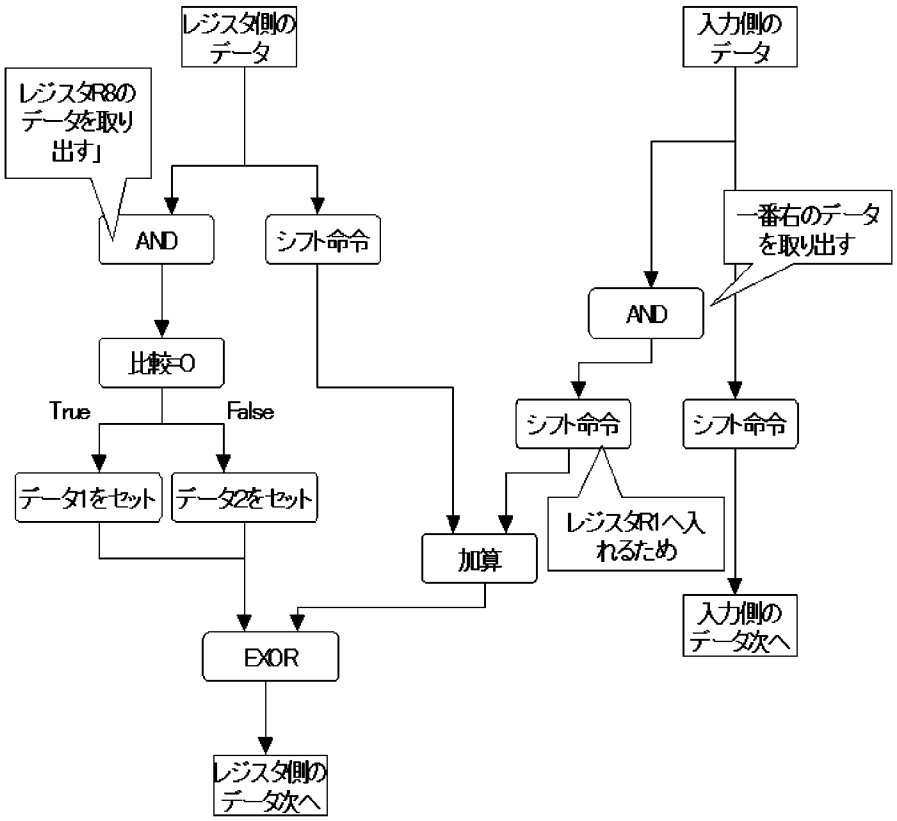

図 2.5　基本的 CRC 処理プログラム

線形帰還形シフトレジスタを用いた逐次処理形 CRC 処理プログラムの性能 (スループット) はそれぞれ

1 ビット入力プログラムで 43KCRC/sec

8 ビット入力プログラムで 100KCRC/sec

の CRC 処理性能を確認した。逐次処理形 CRC 処理プログラムの場合現行の ATM:OC-3 に相当する処理性能を DDMP 上で得ることはできなかった。次に高速処理を可能にするためには逐次処理形 CRC 処理の DDMP 複合命令を追加した場合の性能は 392kCRC/sec であった。これは ATM:OC-3 でセル同期を無視した場合の CRC 処理性能に匹敵する。またプロセッサコアを追加することで更に高速化が期待できる。

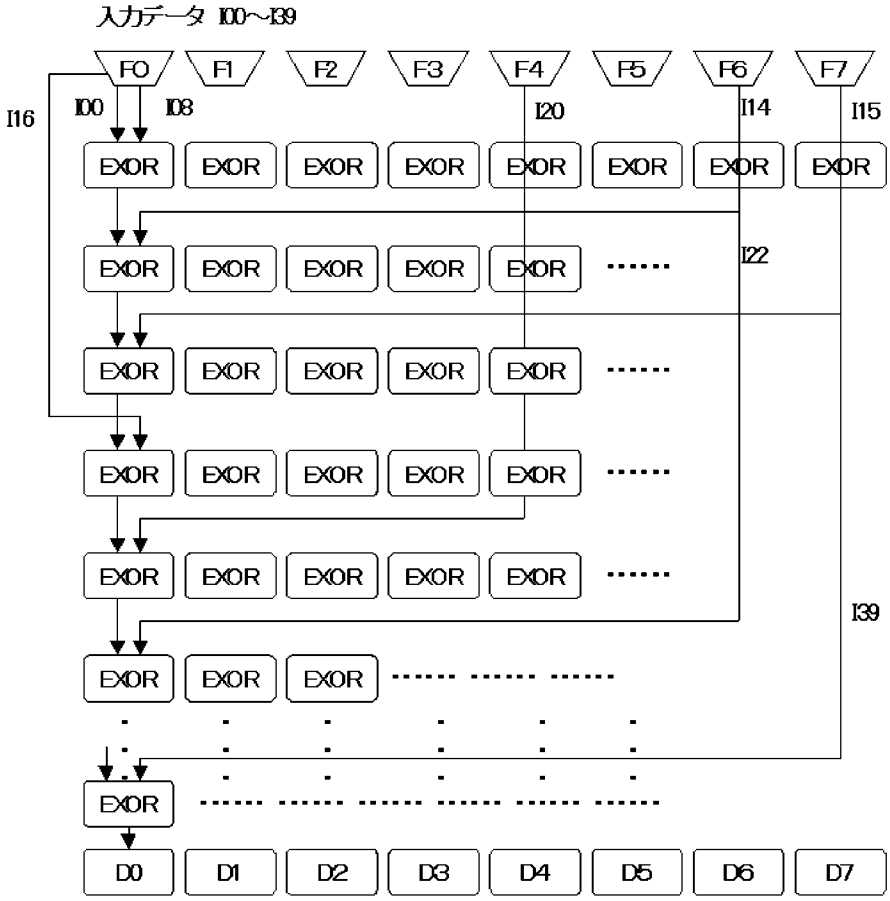

図 2.6　全展開処理型 CRC プログラム

2.5.2　全展開処理形を DDMP により実装した場合

2.4.1 の全展開処理形 CRC 処理を図 2.3 より各フリップフロップの状態を表したプログラムを DDMP-4G 上でフローグラフを検討すると図 2.6 のようになる。この図では 1 つのフリップフロップ F0 の内部状態についての処理を表したものである。このプログラムでは縦続接続される EXOR の数が増え、符号長が長い Ethernet の CRC 処理には適さないことが解かる。また DDMP-4G によりこのプログラムを実装した場合、待ち合わせの遅延時間が増加しプログラムの性能を低下させている。また命令数が多いため DDMP-4G では実装するのが困難である。プロセッサコアを複数にした場合全展開処理形の CRC プログラムの作成は可能になるが、ルータのプロセッサ容量が増加するためあまり良い方法ではない事がわかる。

2.5　各 CRC 処理法を利用したデータ駆動型プロセッサ上での実現法

DDMP 上で並列処理形 CRC 処理を実現すためには、この全展開処理形の例から解かるように命令数を少なくフローグラフを考慮しなければならない。

2.5.3　パイプライン処理形 CRC を DDMP により実装した場合

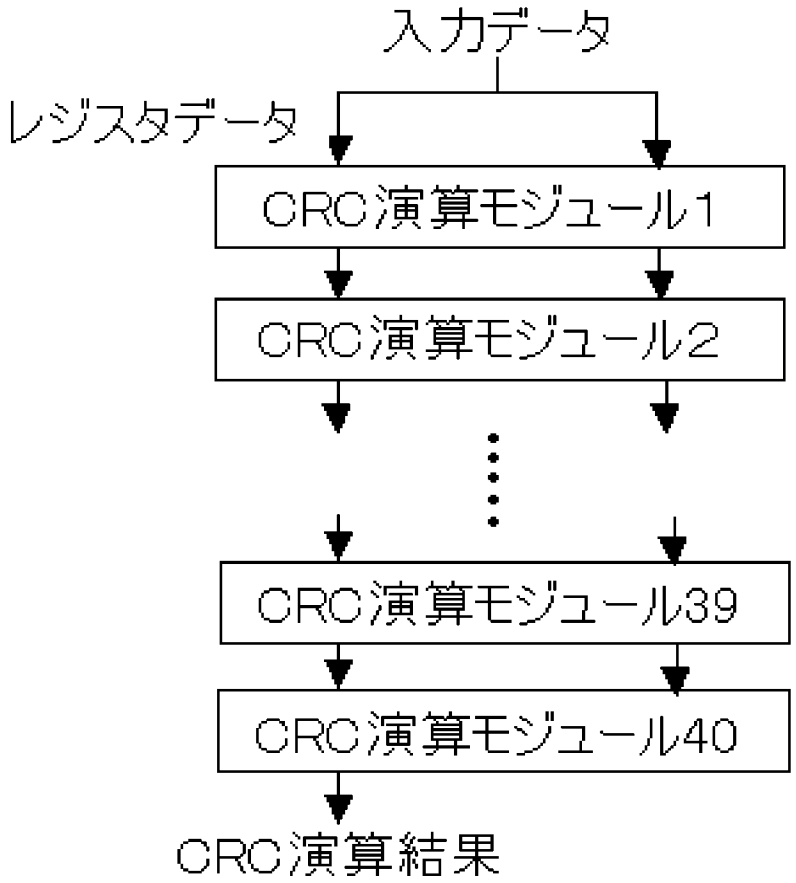

図 2.7　パイプライン処理型 CRC プログラム

図 2.7 はパイプライ処理形 CRC 処理を DDMP フローグラフで表したものである。図中の CRC 演算モジュールは図 2.5 の線形帰還形シフトレジスタを再利用したものである。このモジュールを再利用できるところが DDMP の優れた点でもある。このプログラムは線形帰還形シフトレジスタを４０段のパイプラインとして設計したものである。この CRC 処理プログラムは高速に行うことができるが、DDMP-4G で実装した場合、複数のプロセッサコアにまたがってしまう。ハードウェアで実装した場合には前展開形の処理と同様にハード

ウェアの容量が増えてしまうという欠点が生じる。DDMP 上で実装するにあたり、この欠点を解消するためには 2.4.2 で述べたように、8 段ずつのパイプラインに分けてループ処理によって命令数の削減を行えば、1 つのナノプロセッサー上で実装が可能である。

2.6 結言

本章では ATM における CRC 処理の概要について述べ、並列処理形 CRC 処理を DDMP 上で実現するにあたり CRC 処理の高速化のポイントを明らかにした、まず大切なことは命令数をナノプロセッサ 1 個分に抑えることと、汎用性を持たせたプログラムにすることである。また、本章では各種高速化をねらいとする方式について説明し問題点を明らかにし、DDMP 上で実装した場合のプロセッサの使用率や高速化の問題点を述べた。

第 3 章

並列処理形 CRC 処理

3.1 緒言

本章では並列処理形 CRC アルゴリズムに着目し、これをデータ駆動型プロセッサ上で高速実現する方式を検討し、DDMP-4G を使用して並列処理形 CRC 処理を実装する。実装する際には 2 章で述べたことを念頭においてプログラム開発を行う。また Ethernet での CRC 処理に共通して使える汎用性を持った DDMP の命令を検討し、新命令を使用して並列 CRC 処理プログラムの実装を行う。そして、実現可能性を確認するため DDMP プログラムの CRC 処理性能を評価する。

3.2 並列処理形 CRC 処理

3.2.1 剰余転送並列処理形の原理

本研究では文献 [1] による並列化の手法を用いて DDMP 上でソフトウェア的に並列処理形 CRC 処理を実現する。

剰余転送並列処理形での CRC 処理はハード量が少なくかつ符号長が長い場合にもハード量が増加しないという特徴がある。このことより Ethernet での CRC 処理の場合に対しても容易に対処することが可能である。剰余転送並列処理は図 2.2 で示した線形帰還形シフトレジスタを利用している。その動作フローを図 3.1 に示す。

n_pを生成多項式の周期、n を符号長、l を並列処理するビット数とすると、図 2.2 の線形帰還形シフトレジスタで CRC 処理を並列に行うためには打ち消す並列入力データと、新た

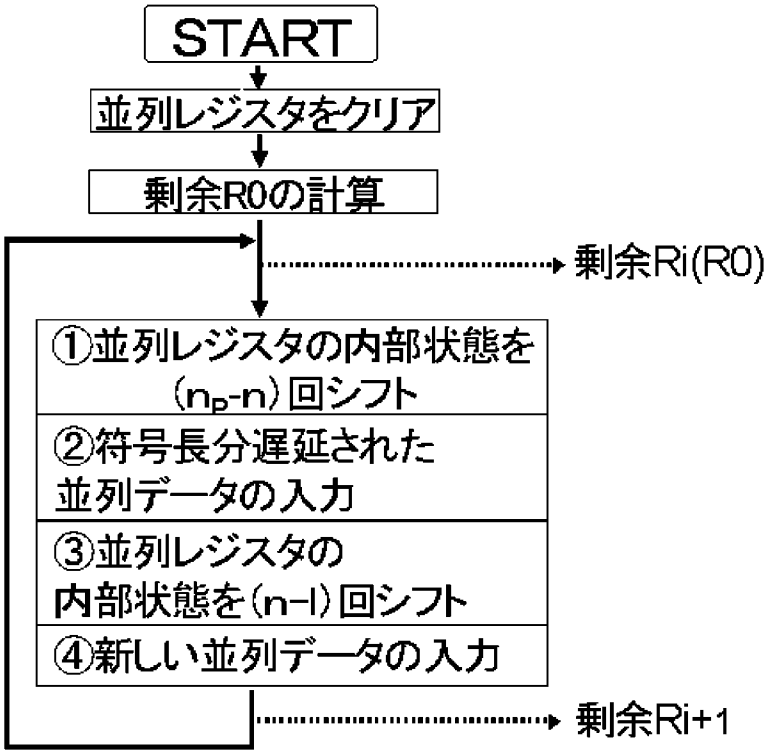

図 3.1　剰余転送処理形式の動作フロー

に入力する並列入力データに関して、入力すべき時刻におけるレジスタの内部状態が異なるため、最初に剰余 R0 が得られた後、次の剰余 R1 を得るためにはレジスタの内部状態を $(n_p - n)$ 回シフトすることで初期状態に戻す。そして、遅延データを入力する、さらに、$(n - l)$ 回シフトすることにより元の状態にレジスタの内部状態をもどし新しいデータを入力し新たな剰余を求めてゆく。処理するデータは ATM なので 40 ビットである。そして、8 ビットずつ処理されるので、このシフトを繰り返す一連の操作を 5 回繰り返し行い 1 回の CRC 演算は終了となる。

3.3　DDMP 上で並列 CRC 処理の実現

3.3.1　データ駆動型並列 CRC 処理プログラムの要件

- 転送速度が 155Mbps 以上であること

  普通 ATM 網のデータ転送速度は OC-3(155Mbps) また、最近では OC-12(620Mbps),OC-48(2.4Gbps) というものが登場してきた。この研究では並列 CRC 処理をよりソフトウェア的に動作させ 155Mbps 以上の転送速度に対応できるのか検証する。最終的な目標としては次世代の ATM に対応することである。

- Ethernet の CRC 処理を視野に入れたプログラムであること

  ATM での CRC 処理をプログラムで実証するのだが ATM の転送プロトコルしか扱えないプログラムではほかの転送プロトコルに対応する別のデータ転送誤り処理回路を必要とするためルータの容量が増えてしまうという欠点が生じる。そこで、同じ CRC 処理方式を採用している Ethernet の CRC 処理についても視野に入れてアルゴリズムを考える必要がある。また、ATM,Ethernet での CRC 処理プログラムで共通命令を複合化することにより CRC 処理の高速化が期待できる。そのために Ethernet の CRC 処理も合わせて検討する必要がある。

3.3.2 並列処理形 CRC のシフトレジスタ

3.2.1 で述べた剰余転送処理形式の並列 CRC 処理の原理は、図 2.2 の線形帰還形 CRC 演算回路の場合と同じく np(生成多項式の周期) 個の内部状態の内、同じ内部状態に 2 度同一の入力データを加えると、その入力データは加えなかったのと同じ結果となることに基づいたものである。すなわち、最初にある入力データ列に対する剰余が得られた後、それ以後は CRC 演算回路に入力された列の内、最も古いデータを打ち消し、新たなデータを入力することによりスタート点が 1 ビットシフトした入力データ列に対する生成多項式による剰余を得るものである。

文献 [1] による演算法により f_1 ～ f_8 のフリップフロップの内部状態 $f_{1,i}$ ～ $f_{8,i}$ および時刻 $i+1$ におけるフリップフロップの内部状態 $f_{1,i+1}$ ～ $f_{8,i+1}$ の関係を以下の式 (2) に示す。

付録参照

$$
\begin{aligned}
f_{1,i+1} &= (f_{1,i} \oplus f_{7,i} \oplus f_{8,i} \oplus D_{8i+8}) \\
f_{2,i+1} &= (f_{1,i} \oplus f_{2,i} \oplus f_{7,i} \oplus D_{8i+7}) \\
f_{3,i+1} &= (f_{1,i} \oplus f_{2,i} \oplus f_{3,i} \oplus f_{7,i} \oplus D_{8i+6}) \\
f_{4,i+1} &= (f_{2,i} \oplus f_{3,i} \oplus f_{4,i} \oplus f_{8,i} \oplus D_{8i+5}) \\
f_{5,i+1} &= (f_{3,i} \oplus f_{4,i} \oplus f_{5,i} \oplus D_{8i+4}) \\
f_{6,i+1} &= (f_{4,i} \oplus f_{5,i} \oplus f_{6,i} \oplus D_{8i+3}) \\
f_{7,i+1} &= (f_{5,i} \oplus f_{6,i} \oplus f_{7,i} \oplus D_{8i+2}) \\
f_{8,i+1} &= (f_{6,i} \oplus f_{7,i} \oplus f_{8,i} \oplus D_{8i+1}) \cdots (2)
\end{aligned}
$$

ただし、$\oplus$ は EXOR を表す。

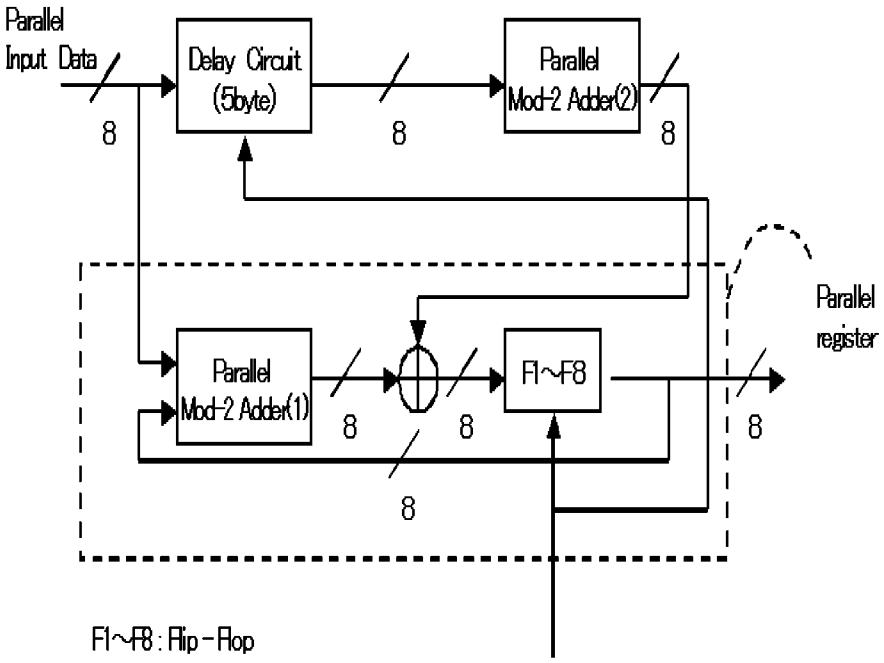

図 3.2　剰余転送処理形式のバイト処理形並列 CRC 剰余算出回路

式 (2) に示される演算回路は図 3.2 のようになる。ATM セルが STM 伝送路上にバイトごとセル単位で多重化されることを前提としているため、セル同期回路にはバイト同期の取られた並列データとそのクロックが入力されるものとする。

剰余転送処理形式のバイト処理形 CRC 剰余算出回路は、セル同期はずれ検出により並列線形帰還形シフトレジスタおよび遅延回路のメモリ素子がリセットされ、その後 5 クロック目以降クロックごとに 1 バイトずれた符号長に等しい入力データ列に対する剰余を出力する。同期確立後は、毎回、並列線形帰還形シフトレジスタをリセットして並列線形帰還形シフトレジスタのみにより剰余を求め、それ以外の時間は並列線形帰還形シフトレジスタへのデータの入力を禁止することにより、並列線形帰還形シフトレジスタを誤り訂正位置検出用回路としても、用いることができる。

このような剰余転送処理形式の剰余算出回路を用いた場合、剰余算出回路を初期設定後から同期確立に至るまでにノイズ等によりエラーが生じた場合には、永久に同期復帰ができないという欠点がある。したがって、周期タイマをもうけ、同期はずれ状態においてある時間を経過しても同期確立状態に移行しない場合には、剰余算出回路にエラーが生じたものとして、剰余算出回路をもう一度、初期設定に戻すことが必要である。

3.3.3　DDMP 並列処理形 CRC 処理プログラム

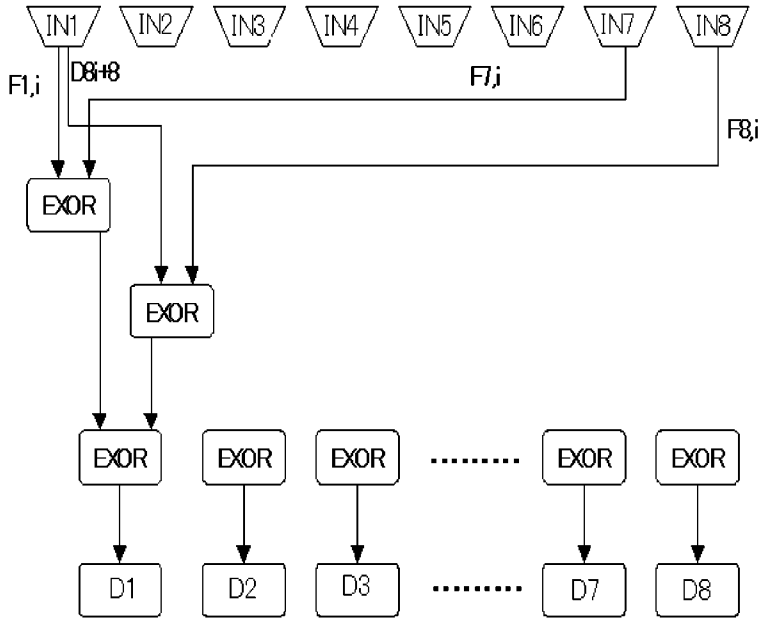

図 3.3　1 ビット入力並列 CRC 処理プログラム

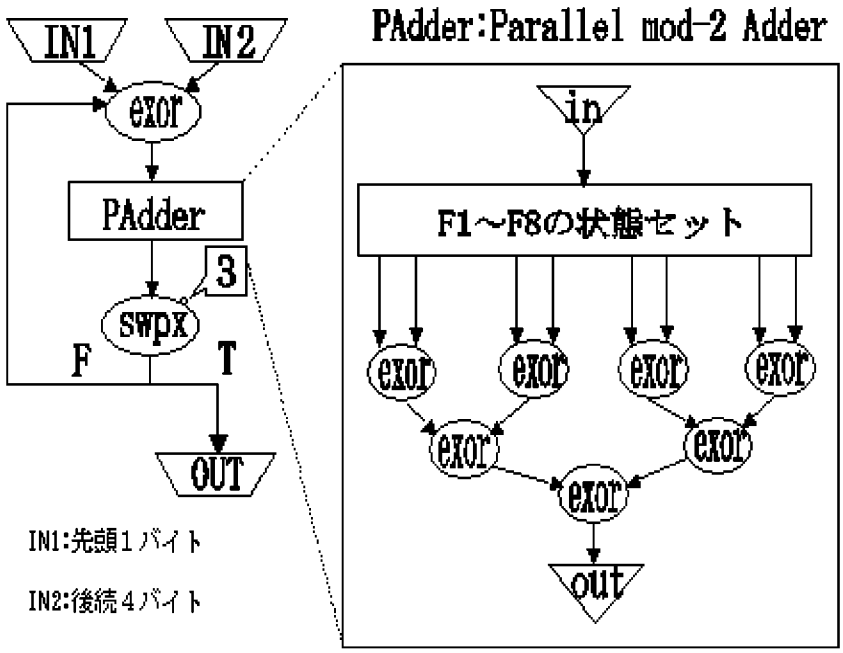

図 3.4　8 ビット入力並列 CRC 処理プログラム

式 (2) よりフリップフロップの各状態をフローグラフに表した 1 ビット入力バージョンのプログラムを図 3.3 に示す。また図 3.4 には 8 ビット入力バージョンのプログラムを示す。ただし図 3.3 のフローグラフはフリップフロップ F1 の動作の概要を示したものである。後者の方はフローグラフ内で 8 ビット単位による処理が可能である為、明らかに 1 ビット単位で CRC 処理を行うよりも高速である。8 ビット入力バージョンの CRC 処理プログラムは図の PAdder の部分が 1 つのモジュールになっており、モジュールの中では一つ前のフリップフロップの内部状態に応じて 8 ビットの内部状態の数値をセットできるような構造になっている。そして、状態ごとに排他的論理和をとることにより剰余を求めている。並列処理形 CRC 処理の優れている点は 1 ビット入力でプログラム内でも 1 ビットずつ処理されている前者プログラムでは命令数が多くナノプロセッサ 3 つで構成されているが、8 ビット入力 8 ビット処理の後者のプログラムは DDMP 上でナノプロセッサ 1 つにより構成されている点である。また、8 ビット入力の並列 CRC 処理プログラムは Parallel mod-2 Adder モジュールの内部に同じ命令セットが複数あり DDMP 複合命令に適した構造になっている。またこ

のプログラムは待ち合わせによる遅延が少ないためスムーズに CRC 処理が行われる。

このプログラムでの CRC 処理能力は、1 ビットずつ処理するプログラムで 152kCRC/sec、8 ビット単位で処理するプログラムで 377kCRC/sec の CRC 処理性能を確認した。

3.4　並列 CRC 処理向きの複合命令

DDMP でプログラムされた並列 CRC 処理プログラムと現行の ATM : OC-3（155Mbps）CRC 処理能力 365kCRC/sec(セル同期を無視して考えた場合) と比較したとき、上記 3.3.3 の両プログラムでは性能が同程度ため DDMP で CRC 処理を複合化した新命令を追加し高速化を図ることを検討した。複合命令を導入する際に ATM および Ethernet の CRC 処理に共通して利用できるよう、CRC の生成多項式に応じてパラメタを設定することができるような複合命令を検討する。共通命令を複合化した命令を DDMP 上でプログラミングすることにより複号命令を使用する前より命令数、パケットの投入間隔が減少し CRC 処理が高速化される。

3.4.1　Ethernet での CRC 処理について

CRC 処理プログラムの命令の複合化を行うにはまず Ethernet の CRC 処理についても ATM と同様に並列化の手法を用いて検討する必要がある。Ethernet の場合も CRC 処理内容は基本的には ATM の場合と同じである。Ethernet の場合は CRC-32 が使用されている。伝送する一定のビット列を 1 つの数字とみなし、ある数で割った剰余を計算し、その剰余をデータに付加して伝送する。 受信側では同じ数で割って、剰余が 0 なら誤り無し、剰余があれば誤り検出とする。Ethernet の場合の生成多項式は

$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X$　　（3）

となる。またイーサネットフレームフォーマットも ATM の場合とは異なっている。

図 2.10 にイーサネットフレームフォーマットを示す。

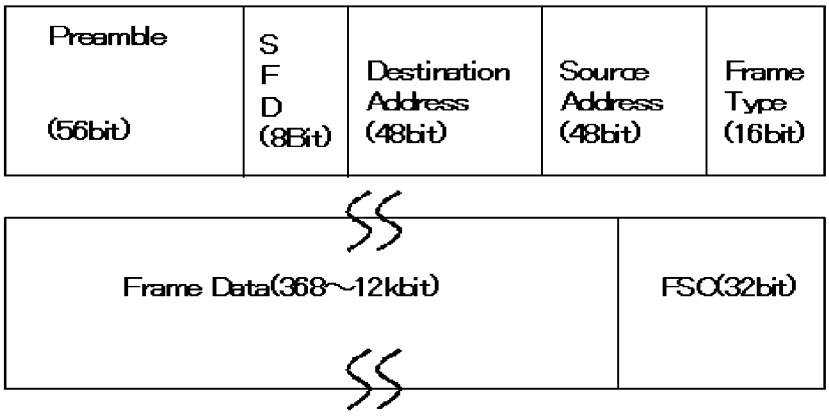

図 3.5　イーサネットフレームフォーマット

3.4.2　ATM と Ethernet での共通命令

ATM の場合と同様に並列処理型 CRC 処理剰余算出法, 文献 [1] を用いて Ethernet の f $_1$ ～ f $_{32}$ のフリップフロップの内部状態 $f_{1,i}$ ～ $f_{32,i}$ および時刻 $i+1$ におけるフリップフロップの内部状態 $f_{1,i+1}$ ～ $f_{32,i+1}$ の関係を以下に求める。

付録参照

$$f_{1,i+1} = (f_{25,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+32})$$

$$f_{2,i+1} = (f_{25,i} \oplus f_{32,i} \oplus D_{32i+31})$$

$$f_{3,i+1} = (f_{26,i} \oplus D_{32i+30})$$

$$f_{4,i+1} = (f_{25,i} \oplus f_{26,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{31,i} \oplus D_{32i+29})$$

$$f_{5,i+1} = (f_{25,i} \oplus f_{28,i} \oplus f_{32,i} \oplus D_{32i+28})$$

$$f_{6,i+1} = (f_{26,i} \oplus f_{29,i} \oplus D_{32i+27})$$

$$f_{7,i+1} = (f_{25,i} \oplus f_{26,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus D_{32i+26})$$

$$f_{8,i+1} = (f_{25,i} \oplus f_{28,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+25})$$

$f_{9,i+1} \sim f_{23,i+1}$まで省略

$$f_{24,i+1} = (f_{16,i} \oplus f_{26,i} \oplus D_{32i+9})$$

$$f_{25,i+1} = (f_{17,i} \oplus f_{27,i} \oplus D_{32i+8})$$

$$f_{26,i+1} = (f_{18,i} \oplus f_{25,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{31,i} \oplus D_{32i+7})$$

$$
\begin{aligned}
f_{27,i+1} &= (f_{19,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{30,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+6}) \\
f_{28,i+1} &= (f_{20,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+5}) \\
f_{29,i+1} &= (f_{21,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{32,i} \oplus D_{32i+4}) \\
f_{30,i+1} &= (f_{22,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{31,i} \oplus D_{32i+3}) \\
f_{31,i+1} &= (f_{23,i} \oplus \oplus f_{30,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+2}) \\
f_{32,i+1} &= (f_{24,i} \oplus f_{25,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{32,i} \oplus D_{32i+1}) \quad \cdots (4)
\end{aligned}
$$

ただし、$\oplus$ は EXOR を表す。

導出された式 (4) より Ethernet の場合の並列 CRC 処理を DDMP フローグラフにより考える。Ethernet の CRC 処理 DDMP プログラムを考える場合 12 ビット以内の単位で CRC 処理をするため、処理するフリップフロップを 8 ビットずつに分配してフローグラフの設計を行う。Ethernet の DDMP プログラムイメージを図 3.6 に示す。

ATM の並列線形帰還形シフトレジスタを用いた 8 ビット入力並列処理形 CRC 処理プログラムのフローグラフと Ethernet の並列処理形 CRC 処理プログラムのフローグラフを比較するとほぼ同じ形をした命令セット図 3.7 を確認することができる。ATM の並列処理形 CRC 処理プログラムの中には 1 クロック前のフリップフロップの状態をセットする命令が 8 組必要である必要である。また Ethernet の場合も同様に 1 クロック前のフリップフロップの状態をセットする命令セットが必要になる。そこで、1 クロック前のフリップフロップの状態を CRC の生成多項式に応じてパラメタを設定可能な複合命令を考案した。この図 3.4 と図 3.6 を見ると Parallel mod-2 Adder の部分を複合命令にしたらもっと高速に CRC 処理が可能になるが、内部状態をセットするためのパラメタの入力が必要なためこのような複合命令にした。内部状態のセットのところも全てプログラミングして DDMP の新命令に Parallel mod-2 Adder を追加できるが ATM 専用の命令、また、Ethernet 専用の命令となり一切の汎用性を失ってしまう。そのため本研究では図 3.7 に示す複合命令を提案した。

DDMP での命令の複合化には制約があり、1 命令で 33 サイクルを越えない範囲で複合命令を設定しなければならない。共通命令を複合化した命令を DDMP 上でプログラミングすることにより複合命令を使用する前より命令数、パケットの投入間隔が減少し CRC 処理の

3.4　並列 CRC 処理向きの複合命令

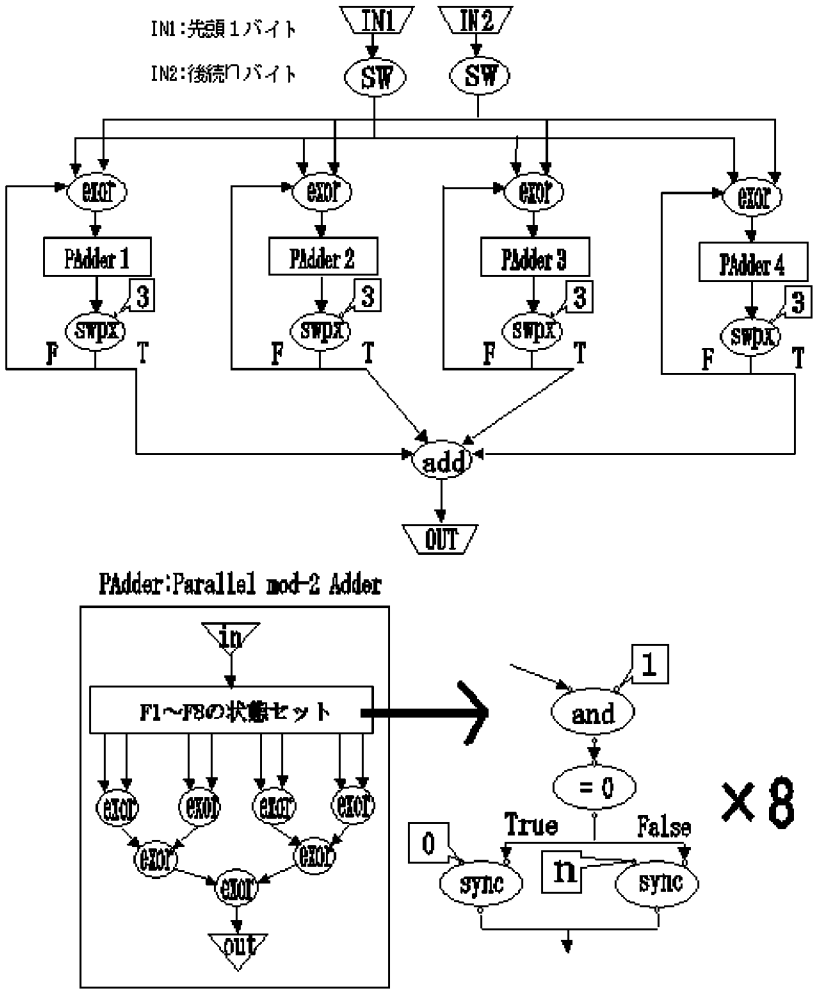

図 3.6　Ethernet の DDMP 並列処理形 CRC プログラムイメージ

高速化が期待できる。

複合命令を作成するのは並列線形帰還形シフトレジスタを用いた 8 ビット入力並列 CRC プログラムであるが性能を比較するために線形帰還形シフトレジスタを用いた逐次処理形 CRC プログラムでも同様に複合命令を導入し比較検証を行う。

本研究では複合命令を使用したプログラムのシミュレーションに別の DDMP の命令を複合命令として共通な命令としたフローグラフの位置に配置し仮想的にプログラムのシミュレーションを行う。例えば、いくつかの命令を DDMP-4G の ADD 命令に置き換えるなどして複合命令を使用したときの CRC 処理プログラムについてのシミュレーションを行う。厳密に言うと他の DDMP 命令と実際に DDMP 新命令として使う場合、若干の違いはある

が、ほぼ同等の処理性能を測定することは可能である。

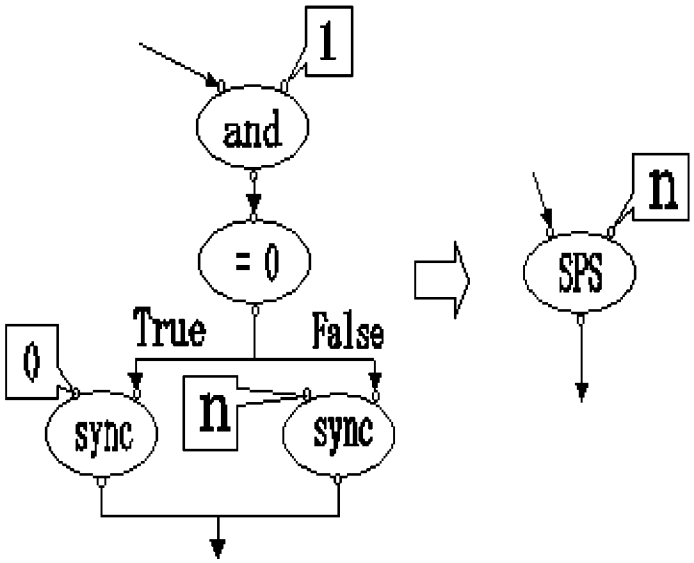

図 3.7　DDMP 新命令　SPS(Set Previous State)

3.5　シミュレーション手法

シミュレーション手法としては 50 パケット（8 ビットごと入力の場合、合計 10 回の CRC 処理をする）を入力して、DDMP-4G フローグラフエディタより 1 回の CRC 処理が行われる時の平均サイクル数などのデータを計測する。

1 ビットごと入力する場合のプログラムは 400 パケットを連続して入力し 10 回の CRC 処理のデータを 8 ビットの場合と同様にデータを採取する。

3.6　DDMP 上での並列 CRC 処理の評価

スループットは 1 秒間あたりに何回 CRC 処理をすることができるかを計測する。
計測式は、以下に示す。

$$\text{スループット} = \frac{1[sec]}{CRC\text{ 出力から次の }CRC\text{ 出力までの差の平均サイクル数}\times 7.5[nsec] \times 10^{-9}}$$

	スループット
1bit 入力逐次処理形 CRC	42k CRC 処理/sec
8bit 入力逐次処理形 CRC	100k CRC 処理/sec
1bit 入力並列処理形 CRC	152k CRC 処理/sec
8bit 入力並列処理形 CRC	374k CRC 処理/sec

表 3.1 CRC 処理性能比較表

この表を見ると逐次処理形の線形帰還形シフトレジスタを基に作成した 1 ビット、8 ビット入力の両プログラムより明らかに並列処理形の CRC 処理プログラム方が CRC 処理性能において優れている。しかしながら、1 ビット入力並列処理形 CRC プログラムはプロセッサコアを 3 つ使用しているが、そのほかの 3 つの CRC プログラムではプロセッサコアは 1 つしか使っていないのでその性能は一概には言えない。プロセッサコアを 2 個,3 個と配置することにより性能も 2 倍,3 倍となる。8 ビット入力並列処理形プログラムについて考察すると、セル同期を無しと考えると ATM:OC-3 用の CRC-8 の処理をデータ駆動型プロセッサコア 1 個で処理可能な性能である。プロセッサコアを増設すれば、より高速なリンクにも対応可能になる。次に普通の ATM の通信速度と通信速度より算出される CRC 処理速度をまとめた表である。ATM の受信側では 1 ビットごとに CRC を使用してセル同期が行われている。セル同期は、セルの流れからセルの境界を識別する機能である。本研究で開発したプログラムは入力パケットが 1 ビット、又は、8 ビットであるのでセル同期を 1 ビット単位で行う場合と、8 ビットごとにセル同期を行う場合、セル同期を無視した場合を考える。この表は ATM の世代別の伝送速度とセル同期を考える場合についての CRC 処理速度についてまとめたものである。この表を見ると表 3.1 で示したプログラム性能では到底表 3.2 に示す性能はカバーすることはできない。そこで前章で述べた複合命令を使用した場合の CRC 処理についての性能を表 3.3 に示す。

本研究で並列 CRC 処理を DDMP 上で実装した場合、共通命令を使用して、初めて ATM の転送速度に対応することができる。

ATM 階級	セル同期なし	1 ビットごと セル同期した場合	8 ビットごと セル同期した場合
OC-3(155Mbps)	365k CRC 処理/sec	155M CRC 処理/sec	19.4M CRC 処理/sec
OC-12(620Mbps)	1.4M CRC 処理/sec	620M CRC 処理/sec	77.5M CRC 処理/sec
OC-48(2.4Gbps)	5.66M CRC 処理/sec	2.4G CRC 処理/sec	0.3G CRC 処理/sec

表 3.2 ATM 性能比較表

	スループット
8bit 逐次処理形 CRC	392 k CRC 処理/sec
8bit 並列処理形 CRC	499k CRC 処理/sec

表 3.3 複合命令を使用した場合の CRC 処理性能比較表

CRC 処理の並列化と複合命令による性能向上率を評価した結果を表 3.4 に示す。逐次処理形の CRC プログラムの場合も DDMP 複合命令を使用した場合、既存命令を使用した場合の並列処理形プログラムの性能を超えているが、逐次処理形の複合命令は ATM の CRC 処理にしか対応しておらず並列処理形の複合命令のように Ethernet の CRC 処理に共通して使えるように作られていないため、並列処理形の CRC プログラムの方が優位といえる。この結果から、線形帰還形シフトレジスタを既存の DDMP で動作させたときより、並列化と複合命令を使用した場合を比較すると、約 5 倍の性能向上が可能になることが判った。これは、セル同期を無視して考えた場合、ATM : OC-3 用 CRC-8 の処理をデータ駆動型プロ

	既存命令使用時	複合命令使用時
8bit 逐次処理形 CRC	100 k CRC 処理/sec	392 k CRC 処理/sec
8bit 並列処理形 CRC	374k CRC 処理/sec	499 k CRC 処理/sec

表 3.4 CRC 処理性能向上率

セッサコア 1 個で十分処理可能な性能である。また、プロセッサコアを増設すれば、より高速なリンクにも対応可能になる。DDMP の並列 CRC 処理に 3 つプロセッサを割り振った場合、ATM : OC-12 用の CRC-8 の処理にも対応することは可能になる。

3.7 結言

本章では CRC 処理の並列化に着目し、DDMP 上で並列処理形 CRC 処理を実装し、更に、Ethernet での CRC 処理に共通して使える DDMP の命令を提案した。また、DDMP-4G で作成したプログラムの性能評価を行い、現状の ATM ならば問題なく本研究の DDMP プログラムにより実装できること確認した。

第 4 章

結論

本研究ではルータの処理のボトルネックであるデータ伝送誤り処理を具体的な問題として、データ駆動型プロセッサ DDMP 上でソフトウェア的に並列処理形 CRC 処理を実現する方法を提案した。そして、パケット通信向けの DDMP 新命令を提案した。また、本研究はデータ駆動原理を基礎とする研究であり、データ駆動型の原理、データ駆動型マルチメディアプロセッサの特徴なども合わせて説明を行った。

第 2 章では CRC 処理の高速化をねらいとした手法を取り上げ各 CRC 処理手法における利点と問題点について検討した。ATM での CRC 処理をパイプライン的に行う逐次処理形 CRC を DDMP 上で実装した場合の問題点を網羅した。CRC 処理性能評価を行い 100kCRC/sec の性能を確認した。3 章で述べる複合命令を用いた並列処理形 CRC プログラムと性能を比較するために逐次処理形 CRC プログラムでも同様に複合命令を検討し、複合命令を導入した場合についてもシミュレーションを行った。

第 3 章では CRC 処理を高速化するために並列線形帰還形シフトレジスタを用いて DDMP 上で並列処理形 CRC 処理を実現した。しかしながら、DDMP で実装された並列 CRC 処理プログラムと現行の ATM を比較した場合、並列処理形 CRC プログラムの処理性能が同程度ため DDMP で CRC 処理を複合化した新命令を追加し CRC 処理の高速化を行った。この複合命令を導入する際には ATM および Ethernet の CRC 処理に共通して使用できるように CRC 生成多項式に応じてパラメタを自由に設定可能な複合命令とした。その複合命令を使用した場合で並列処理形 CRC プログラム性能評価のシミュレーションを行った結果、約 500kCRC/sec の性能を確認した。これにより、逐次処理形 CRC プログラムの約 5 倍の性能向上が可能になることが判った。これは、ATM : OC-3 用の CRC-8 の処

理をデータ駆動型プロセッサコア 1 個で十分処理可能な性能である。また、プロセッサコアを増設することにより更に高速な ATM や Ethernet にも対応可能になる。

本研究では CRC の並列化手法を用いてデータ駆動型プロセッサ上で並列 CRC 処理を実装した。ATM:OC-3 の場合は複合命令を使用した並列 CRC 処理プログラムをプロセッサコア 1 つで動作させてその性能をカバーできるが、OC-12、OC-48 の処理能力にはプロセッサコアを増設したとしても届かないためアルゴリズムの改良、新たな新命令の検討などが必要である。今後の課題としては今回作成した複合命令は実際には DDMP の命令には追加されないので複合命令を最適化し実際に DDMP 新命令としてプログラミングし追加した場合、CRC 処理速度の測定を行わなければならない。また、この複合命令を使用してのハードウェア設計による詳細な評価が残されている。

今回の研究では主に ATM での CRC 処理を取り上げたが、Ethernet の CRC 処理について厳密に検討し複合命令を考慮し、より一層の CRC 処理において汎用性が求められる DDMP 命令を考える必要がある。

今後の展望としては、本研究において DDMP で開発した CRC プログラム、本研究室で研究の進んでいる高速 IP アドレス検索プログラムなどを搭載した、ルータ専用のチップを開発し、高速ルータをデータ駆動型マルチメディアプロセッサにより開発することが我々研究室全体の目標である。

謝辞

本研究において、懇切なる御指導、御鞭撻を賜った岩田 誠 助教授に心より感謝の意を表します。

本研究を進めるにあたり、懇切なる御指導、御鞭撻を賜った大森 洋一 助手に心より感謝の意を表します。

論文執筆にあたり、適切な御助言、ご指導を賜った岩田研究室の細美 俊彦氏に心より感謝の意を表します。

本研究を進めるにあたり、適切な御助言、ご指導を賜った岩田研究室の方々
橋本 正和氏、古家 俊之氏、別役 宣奉氏、前田 庸亮氏、森川 大智氏に感謝の意を表します。

参考文献

[1] 竜野秀雄、戸倉信之 :“ 即時シフト形セル同期回路に用いる並列処理形 CRC 剰余算出法 ” 、電子情報通信学会論文誌 (B1)、Vol.J73-B- 1 No.11、pp.873-876(1990-11).

[2] 豊島 鑑、龍野 秀雄 :“ ヘッダ誤り制御によるセル同期回路構成法の検討 ” 信学技法、CS89-70(1989-11).

[3] 青木利晴、青山友紀 監修、濃沼 健夫 : 広帯域 ISDN と ATM 技術、電子情報通信学会 (1995-2)

[4] H.Terada,S.Miyata,and M.Iwata,“DDMP's: self-timed super-pipelined data-driven multimedia processors, ” Proc. of IEEE,87(2),282-296(1999).

[5] `http://www.sharp.co.jp`

付録 A

並列処理形 CRC 算出法

Ethernet のフリップフロップ内部状態の導出法を以下に示す。

次数mの生成多項式に対応する線形帰還形シフトレジスタの 1 回のシフト動作を (m×m) の行列 T で表すと、並列処理形線形帰還シフトレジスタの 1 回のシフト動作は次式で示される $((l+m) \times (l+m))$ の行列 T_p を用いて T_p^l で表される。

$$T = \begin{bmatrix}
00000000000000000000000000000001 \\
10000000000000000000000000000001 \\
01000000000000000000000000000000 \\
00100000000000000000000000000001 \\
00010000000000000000000000000001 \\
00001000000000000000000000000000 \\
00000100000000000000000000000001 \\
00000010000000000000000000000001 \\
00000001000000000000000000000000 \\
00000000100000000000000000000001 \\
00000000010000000000000000000001 \\
00000000001000000000000000000001 \\
00000000000100000000000000000000 \\
00000000000010000000000000000000 \\
00000000000001000000000000000000 \\
00000000000000100000000000000001 \\
00000000000000010000000000000000 \\
00000000000000001000000000000000 \\
00000000000000000100000000000000 \\
00000000000000000010000000000000 \\
00000000000000000001000000000000 \\
00000000000000000000100000000001 \\
00000000000000000000010000000001 \\
00000000000000000000001000000000 \\
00000000000000000000000100000000 \\
00000000000000000000000010000001 \\
00000000000000000000000001000000 \\
00000000000000000000000000100000 \\
00000000000000000000000000010000 \\
00000000000000000000000000001000 \\
00000000000000000000000000000100 \\
00000000000000000000000000000011
\end{bmatrix} \cdots (1)$$

$$T_p = \left[\begin{array}{c|c} 00\ldots00 & \\ 10\ldots00 & \\ 01\ldots00 & \phi \\ \vdots\quad\vdots & \\ 00\ldots10 & \\ \hline 00\ldots01 & \\ & \mathrm{T} \\ \phi & \end{array}\right] \cdots (2)$$

ただし、シリアル入力データを D_1～ D_lとすると式 (2) の第 l 列は D_1を、第 1 列は D_lを示し、ϕ は零行列をを示すものとする。また、行列演算は $mod-2$ の加算により行われるものとする。T_p^lは並列入力データ部分を行列 T_D で表せば、次式のようになる。

$$T_p^l = \left[\begin{array}{c|c} \phi & \phi \\ \hline T_D & T^l \end{array}\right] \cdots (3)$$

ここで、入力データに関する行列演算を可能とするため、以下のように定義した行列 T'、$T_p^{l\prime}$ を導入する。

$$T' = \left[\begin{array}{c|c} \mathrm{E} & \phi \\ \hline \phi & \mathrm{T} \end{array}\right] \cdots (4)$$

$$T_p^{l'} = \left[\begin{array}{c|c} E^l & \phi \\ \hline D_D & T^l \end{array}\right] \cdots (5)$$

ただし、E は時間演算子を示す $(l \times l)$ の単位行列であり、E、E^lはシリアルデータをそれぞれ 1 ビットシフト、l ビットシフトさせることを示すものとする。また、$E^{np} = E^0$であり、E^0は通常の単位行列の性質を持つものとする。

次に、並列データのクロック時刻 i における並列線形帰還形シフトレジスタの内部状態と並列入力データをそれぞれ列ベクトル Si を次式のように定義する。

$tS_i = (tD_i, tF_i) \cdots (6)$ ただし、

$tD_i = (D_{li+l}, D_{li+l-1}, \cdots, D_{li+1}) \cdots (7)$

$$tF_i = (f_{1,i}, f_{2,i}, \cdots, f_{m,i}) \cdots (8)$$

D_{li+l}は時刻 i における並列入力データの第 1 ビット目、$f_{m,i}$は時刻 i における並列線形帰還形シフトレジスタの第 m 番目のフリップフロップの内部状態を示すものとする。式 (4) ～ (6) を用いることにより, 時刻 $i+1$ における並列線形帰還形シフトレジスタの内部状態と並列入力データを表す列ベクトル S_{i+1}は図 2.4 の動作フローに従って、列ベクトル S_iに図 2.4 の①～④の状態遷移を表す行列を順次左側からかけることにより次式のように表せる。

$$S_{i+1} = T_p^{l'} \cdot T'^{(n-l)} \cdot T_p^{l'} \cdot T'^{(np-n)} \cdot S_i$$

$$= \left[\begin{array}{c|c} E^l & \phi \\ \hline D_D & T^l \end{array}\right] \left[\begin{array}{c|c} E^{(n-l)} & \phi \\ \hline \phi & T^{(n-l)} \end{array}\right] \left[\begin{array}{c|c} E^l & \phi \\ \hline D_D & T^l \end{array}\right] \left[\begin{array}{c|c} E^{(np-n)} & \phi \\ \hline \phi & T^{(np-n)} \end{array}\right] S_i$$

$$= \left[\begin{array}{c|c} E^{(np+l)} & \phi \\ \hline T_D \cdot E^{np} \oplus T^n \cdot T_D \cdot E^{(np-n)} & T^{(np-l)} \end{array}\right] S_i \cdots (9)$$

ただし、$\oplus$ は mod-2 による加算を示す。式 (9) は更に (6) ～ (8) を用いて表すと次式のようになる。

$$F_{i+1} = T_D \cdot D_i \oplus T^n \cdot T_D \cdot E^{-n} \cdot D_i \oplus T^l \cdot F_i \cdots (10)$$

ただし

$$T^{np+l} = E^0 \cdot T^l = T^l, E^{np} = E^0 \cdots (11)$$

式 (9) は、剰余転送処理形式の並列形剰余算出回路の動作を示してあり、右辺の第 1 項は新たに入力される並列データを表し、第 2 項は打ち消すべき n ビット前の並列データをそれぞれ表している。なお、並列線形帰還形シフトレジスタの動作は式 (12) の右辺の第 1 項と第 3 項の mod-2 による加算により表される。これらの式 (2)(3)(10) により時刻 i における *Ethernet* の並列データ D_{li+1}～ D_{li+8}, f $_1$ ～ f $_{32}$ のフリップフロップの内部状態 $f_{1,i}$ ～ $f_{32,i}$ および時刻 $i+1$ におけるフリップフロップの内部状態 $f_{1,i+1}$ ～ $f_{32,i+1}$ の関係の関係を以下に求める。

$$f_{1,i+1} = (f_{25,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+32})$$

$$f_{2,i+1} = (f_{25,i} \oplus f_{32,i} \oplus D_{32i+31})$$

$$f_{3,i+1} = (f_{26,i} \oplus D_{32i+30})$$

$$f_{4,i+1} = (f_{25,i} \oplus f_{26,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{31,i} \oplus D_{32i+29})$$

$$f_{5,i+1} = (f_{25,i} \oplus f_{28,i} \oplus f_{32,i} \oplus D_{32i+28})$$

$$f_{6,i+1} = (f_{26,i} \oplus f_{29,i} \oplus D_{32i+27})$$

$$f_{7,i+1} = (f_{25,i} \oplus f_{26,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus D_{32i+26})$$

$$f_{8,i+1} = (f_{25,i} \oplus f_{28,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+25})$$

$$f_{9,i+1} = (f_{1,i} \oplus f_{26,i} \oplus f_{29,i} \oplus f_{31,i} \oplus D_{32i+24})$$

$$f_{10,i+1} = (f_{2,i} \oplus f_{25,i} \oplus f_{26,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{31,i} \oplus D_{32i+23})$$

$$f_{11,i+1} = (f_{3,i} \oplus f_{25,i} \oplus f_{28,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+22})$$

$$f_{12,i+1} = (f_{4,i} \oplus f_{25,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{30,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+21})$$

$$f_{13,i+1} = (f_{5,i} \oplus f_{26,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+20})$$

$$f_{14,i+1} = (f_{6,i} \oplus f_{27,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{32,i} \oplus D_{32i+19})$$

$$f_{15,i+1} = (f_{7,i} \oplus f_{28,i} \oplus f_{30,i} \oplus f_{31,i} \oplus D_{32i+18})$$

$$f_{16,i+1} = (f_{8,i} \oplus f_{25,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{30,i} \oplus f_{32,i} \oplus D_{32i+17})$$

$$f_{17,i+1} = (f_{9,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+16})$$

$$f_{18,i+1} = (f_{10,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{32,i} \oplus D_{32i+15})$$

$$f_{19,i+1} = (f_{11,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{31,i} \oplus D_{32i+14})$$

$$f_{20,i+1} = (f_{12,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+13})$$

$$f_{21,i+1} = (f_{13,i} \oplus f_{30,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+12})$$

$$f_{22,i+1} = (f_{14,i} \oplus f_{25,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{32,i} \oplus D_{32i+11})$$

$$f_{23,i+1} = (f_{15,i} \oplus f_{25,i} \oplus D_{32i+10})$$

$$f_{24,i+1} = (f_{16,i} \oplus f_{26,i} \oplus D_{32i+9})$$

$$f_{25,i+1} = (f_{17,i} \oplus f_{27,i} \oplus D_{32i+8})$$

$$f_{26,i+1} = (f_{18,i} \oplus f_{25,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{31,i} \oplus D_{32i+7})$$

$$f_{27,i+1} = (f_{19,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{30,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+6})$$

$$f_{28,i+1} = (f_{20,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+5})$$

$$f_{29,i+1} = (f_{21,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{32,i} \oplus D_{32i+4})$$

$$f_{30,i+1} = (f_{22,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{31,i} \oplus D_{32i+3})$$

$$f_{31,i+1} = (f_{23,i} \oplus \oplus f_{30,i} \oplus f_{31,i} \oplus f_{32,i} \oplus D_{32i+2})$$

$$f_{32,i+1} = (f_{24,i} \oplus f_{25,i} \oplus f_{26,i} \oplus f_{27,i} \oplus f_{28,i} \oplus f_{29,i} \oplus f_{30,i} \oplus f_{32,i} \oplus D_{32i+1}) \ldots (11)$$

ただし、$\oplus$ は EXOR を表す。