計算機アーキテクチャ　99－13
(1993. 3. 12)

実時間並列処理計算機 CODA －プロセッサ－

西田健次†、戸田賢二、島田俊夫‡、山口喜教
電子技術総合研究所
‡名古屋大学
〒305 茨城県つくば市梅園 1-1-4
† E-mail nishida@etl.go.jp

あらまし

CODA はセンサフュージョンシステムに用いる実時間用並列処理計算機として設計されている。CODA では 32bit の優先度フィールドを持つことで、デッドライン時刻を直接優先度として用いることができる。それにより、タスクスケジューリングからパケット転送、割り込み制御などの基本操作に至るまで、一貫した優先度制御が可能である。CODA プロセッサは、タスク実行用に RISC 型のパイプラインアーキテクチャ、パケット処理および割り込み処理用にマルチスレッド型のパイプラインアーキテクチャの二本の処理パイプラインを持つ。これにより、タスクの実行時間の予測性を高めるとともに、パケットや割り込みに対する応答性を高めることを目指している。

和文キーワード　実時間処理 並列計算機アーキテクチャ RISC アーキテクチャ

The Real-Time Paralell Processor CODA – Processor Architecture–

Kenji NISHIDA, Kenji TODA, Toshio SHIMADA†, Yoshinori YAMAGUCHI
Electrotechnical Laboratory
† Nagoya University

1-1-4 Umezono Tsukuba-shi, IBARAKI 305, JAPAN

Abstract

This paper presents a hardware organization of CODA, a parallel processor for realtime applications. CODA consists of multiple identical processors connected via a packet communication network. A processor element for CODA integrates RISC pipeline for task execution and multi-thread circular pipeline for packet handling and interrupt handling. A 32-bit priority field is provided not only for task scheduling but also for packet communication and interruption, thereby every essential operation is handled by coherent priority in CODA processor.

英文 key words　Real-Time Processing, Parallel Architecture, RISC Architecture

1 はじめに

将来の高度情報処理システムにおいては、外界の情報を認識 / 判断等の高度な処理を限られた時間の中で行ない、外界に対して適切な応答を行なう自律性が要求されると考えられる。すなわち、従来、実時間で処理することが考慮されていなかった、大規模な計算を必要とする分野に対しても、実時間処理が必要とされてくることを示す。認識 / 判断を適切なものとするために、外界からの情報を入力するセンサの数および種類が、従来よりも遥かに多くなることが考えられる。センサフュージョンシステムは、このような応用の一分野として提案されており、膨大な数の異種センサからの情報を統合し、従来にない新たなセンシング機能を付与し、認識 / 判断等の高度な処理に対する支援を行なおうとするものである [2]。

CODA はセンサフュージョンシステムを支える計算機システムとして設計されており、外界の情報を得るため多数のセンサからの情報を処理するための高い I/O 能力と実時間性、そして認識 / 判断などを高速に行なうための高い計算能力の両立を目指している [3]。CODA は、並列計算機アーキテクチャとして必須の高速な同期機構と高速なプロセッサ間通信機能に加え、、ハードウェア化された優先度キューと高速なコンテクスト切替え能力、命令実行の他に通信および I/O 専用のパイプラインを持つことにより、実時間処理に必要な機能、特に割り込みや I/O 機器への応答性、を高めている。

本稿では、CODA のプロセッサアーキテクチャ、特にパイプライン構成について述べる。

2 実時間用並列アーキテクチャ

実時間処理においては、実行時間の予測性を確保することが重要な課題であるため、プロセッサ間の通信や同期に必要な時間が予測できないという理由で並列処理を行なうことは不利であると考えられてきた。しかし、センサやアクチュエータに対する応答性を確保したままで I/O バンド幅を大きくしていくためには、並列処理が必要な段階に来ていると考えられる。実時間処理を並列に実行するためには、命令レベルからタスクレベルに至るまで、実行時間の予測性を確保しなくてはならない。そのためには、プロセッサアーキテクチャのレベルから実行時間の予測性を考慮した設計を行なわなくてはならない。

我々は、実時間処理を支援する機構を持つプロセッサアーキテクチャを実時間用並列アーキテクチャとして提案してきた [1]。本節では実時間用並列アーキテクチャの備えるべき機能について考察する。

2.1 優先度 (デッドライン) による制御

逐次計算機による実時間処理においては、タスク毎に終了しなくてはならない時刻 (デッドライン) を定めて、それにしたがったスケジューリングを行なうことにより、処理全体が定められた時間内に終了することを保証しようとしている。この手法は、タスクを実行するのに必要な時間の上限が定まっていれば有効な方法だが、並列計算機ではプロセッサ間の通信や同期、共有資源に対するアクセス競合などの影響によりタスクの実行時間の上限を定めることが困難になっている。従って、並列計算機においては、ここに挙げた通信や同期。共有資源に対するアクセスなどの基本操作に対して処理時間の上限を定められるようにしなくてはならない。即ち、タスクよりも小さな単位でのデッドライン制御あるいは優先度制御が必要である。

2.2 タスクスケジューリングの援助

並列計算機においては、一度あるプロセッサに配置してしまったタスクを再配置することは、大きなオーバヘッドを伴うため実時間処理に適用するのは困難である。従って、並列計算機の計算能力を最大限に生かすためには、タスク生成時に最適な配置を行なわなくてはならない。しかし、これは NP 困難な問題であり実際的には不可能である。そのため、個々のプロセッサ上で最適とされるスケジューリングを行ない、(heuristics を用いた) 準最適なタスク配置を行なうことで対応することになる。このようなタスク配置方法が妥当なものであるためには、各プロセッサ上でのタスクスケジューリングが、(局所的には) 正しいものである必要があり、また、新たなタスクを配置可能であるかどうかの判定が充分に高速に行なえなくてはならない。

2.3 プロセッサ間通信の予測性の確保

並列計算機においてプロセッサ間通信は必須のものである。しかし、通信に必要な時間の上限は、通信路上での競合などにより、必ずしも一定に定められるものではない。そこで、プロセッサ間通信に用いられるデータ一つ一つについて、データが目的地に到着しなければならない時刻を与え、それにしたがって通信路上での制御を行なう。これにより、より早く到着しなければならないデータが優先して通信路を通過することになり、プロセッサ間通信での予測性を向上することができる。

2.4 外部事象に対する応答

外部事象に対する応答性は、割り込みに対する応答性として評価されることが多い。しかし、既存のプロセッサアーキテクチャでは、割り込みに与えられる優先度は、必ずしも割り込みで行なわれる処理のデッドラインと一致するものではない。そのため、実行中のタスクの実行時間の予測性を保つためには、割り込みの受け付け許可 / 禁止を注意深く行なわなくてはならない。並列計算機において、このような制御を一貫性を保って行なうことは、全体の同期を必要とするためコストの大きなものとなる。割り込み処理に対してもデッドラインを指定できるようにすれば、実行中のタスクと割り込み処理を一貫して管理することが可能になる。

3 CODAのアーキテクチャ

CODAは、通信や割り込みなど全ての処理を一貫した優先度(デッドライン)で管理することを目指して設計されている。そのため優先度フィールドとして32bitを設け、全ての処理に対して優先度による制御を行なう。プロセッサ間のネットワークは、パケット一つ一つに対して32bitの優先度判定を行ない、優先度逆転による通信の遅れを解消する機構を備えている[4]。そして、個々のプロセッサでは、それぞれの持つ最高優先度のタスクをつねに実行できるように、タスクキューの機能を内蔵している。CODAプロセッサの特徴をまとめると以下のようになり、これらによって前節で述べた実時間用並列計算機としての要件を満たそうとしている。

- 32bitの優先度フィールドによって、デッドラインを直接優先度とすることができる
- タスクキューの機能を内蔵している
- 最高優先度のタスクを常に実行できるように、高速なコンテクストスイッチを可能にしている
- プロセッサ間通信や割り込みによりタスク実行が妨げられないように、専用のパイプラインを持つ
- 高速、低オーバヘッドの通信、同期機構を持つ

CODAは、このプロセッサを優先度制御を行なうネットワークで接続するもので(図1)、現在64台プロセッサによるプロトタイプの試作を行なっている。

3.1 CODAの命令セット

3.1.1 分岐命令

CODAは、レジスタ上でプロセッサ間の同期を行なう機構を持つ。これは、レジスタに対して、外部からのパケットが直接書き込みを行なうことを意味する。そのため、命令パイプラインで実行中のコンテクストでの演算結果、および、演算によって立ったフラグと、レジスタの内容が必ずしも一致しないと考えられる。即ち、演算の履歴をコンディションコードレジスタ(CCR)に残し、それを利用して条件分岐を行なうことは、プログラムの挙動をわかりにくいものとしてしまう。

CCRを廃止して、レジスタの内容によって分岐を行なうことにより、上記の問題は避けられるが、分岐命令の実行にレジスタ内容のテストが必要となるため、分岐命令実行の効率低下を招く。そこで、レジスタフラグとして、レジスタの内容がゼロかどうかを示すフラグを設け、条件分岐に利用する。そのためには、単なるレジスタ間転送命令でも、ALUを通してフラグを立てなくてはならない。

3.1.2 パケットパイプラインとの命令の共用

パケットパイプラインの機能が、単にパケットの送受のみであるとすると、外部からのサブルーチン呼び出しあるいはタスク配置要求が来た場合、命令パイプラインが実行中のタスクを中断してそれらの処理を行なわなくてはならない。そこで、パケットパイプラインでも命令パイプラインと同様の命令を実行できるようにして、命令パイプラインをハウスキーピング的な処理から解放する。同時に、割り込み受け付け等もパケットパイプラインで行なうことにより、命令パイプラインはタスク実行に専念することができるようになる。

パケットパイプラインは、パケットの処理スループットを向上することを主眼としているため、メモリから読み出した命令をそのまま実行するには適さない。そこで、メモリから命令を読み出した後、レジスタ読み出しを行ない、プロセッサ内部でパケット形式を合成してパケットパイプラインに実行させるようにする。同様に、割り込み要求も、プロセッサ内部でパケット形式を合成して、パケットパイプラインに与えるものとする。これにより、命令実行、パケット処理、割り込み処理の調停を単純化することができ、特に割り込みに対する応答性を高めることができる。

命令形式とパケット形式の変換を簡単化するために、命令フォーマット、パケットフォーマットともに、単純な形式を用いるものとする。命令形式とパケット形式の対応を図2に示す。

3.2 CODAプロセッサの概要

CODAプロセッサの構成を図3に示す。CODAプロセッサは、二本の実行パイプラインを持ち、これらは基本的に同一の命令セットを実行することが可能である。Instruction Pipeline (IP)は、四段のRISC構成をとっており、最高優先度のタスクを常に実行することを目的としている。そのため、パケット受信、割り込み受け付けなどは行なわない。タスクテーブルは、IPで実行中のタスクの優先度を常に監視しており、実行中のタスクより高い優先度を持ったタスクが割り当てられた場合はIPに対してプリエンプションを要求する。

パケットパイプライン(PP)は、マルチスレッド循環パイプライン構成をとっており、毎クロックパケットを処理することを目指している。PF, PRステージは、IPと同一の命令形式をパケットパイプラインでの処理形式に変換する。これにより、パケット受信や割り込み処理などがない場合には、優先度が最高でないタスクの実行を行なうことができ、プロセッサの稼働率を向上することができる。

レジスタは、IP, PPに対して独立したアクセスポートを持つために6ポートのメモリを用いる。また、タスクテーブルの内容と一致させた8コンテクストを保持させることにより、プリエンプションや割り込みによるコンテクスト切替えの高速化を図っている。レジスタの各語は、同期フラグを持ちレジスタにデータが書き込まれているか否かを示す。パケットから直接レジスタにデータを書き込むこととあわせ、プロセッサ間の同期をレジスタアクセス時に行なうことができる。

パケット出力ステージ、データメモリバスは、IPとPP

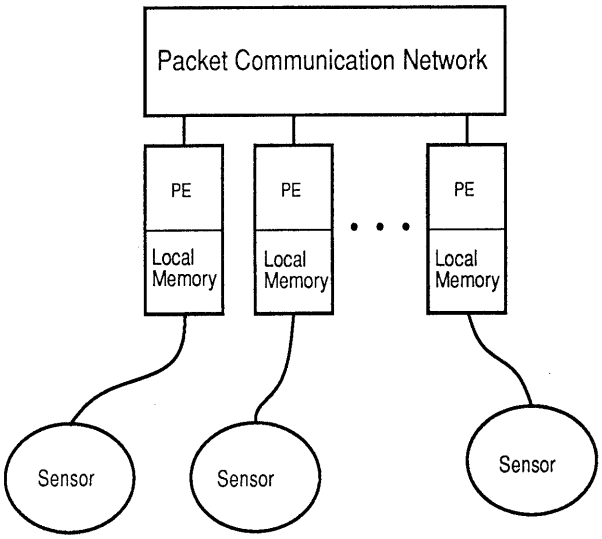
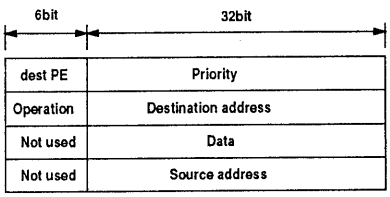
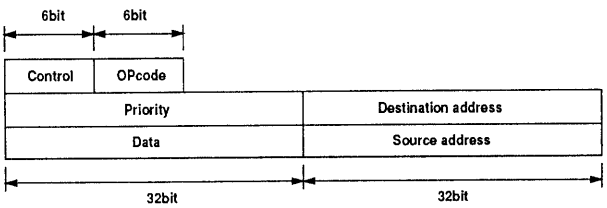


図 1: CODA の全体構成



a. Packet External Form



b. Packet Internal Form

図 2: パケットの外部形式と内部形式

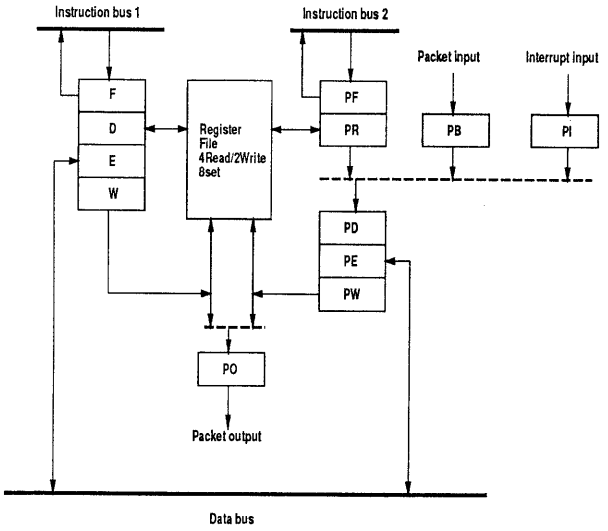


図 3: CODA プロセッサの構成

に共有される。アクセスが競合した場合は、IP を優先するものとする。

3.3 命令パイプライン (IP) の構成

図 4に命令パイプラインの構成を示す。

NXTPC は、フェッチする命令アドレスを示し、NXTPC から命令メモリを介して P1-FETCH-REG までが F(命令フェッチ) ステージである。PC, EXPC は、それぞれデコード中の命令アドレス、実行中の命令アドレスを保持しており、コンテクスト切替えの際の戻り番地として使われる。

D(命令デコード) ステージでは、レジスタの読みだし、次命令アドレスの計算、命令デコードを行なう。レジスタ読みだし時にレジスタの同期フラグのチェックを行なう。レジスタにデータが書き込まれていなければ、次命令のアドレスをタスクテーブルから選びだし、PC, EXPC を続く 2 クロックでタスクテーブルに退避することで、コンテクストの切替えを行なう。また、タスクテーブルからのプリエンプション要求を受け付けてコンテクスト切替えを行なうのも、同手順である。命令のデコード結果、読みだしたレジスタデータは P1-DECODE-REG に書き込まれ、E(実行) ステージに送られる。

E(実行) ステージは、メモリアクセスおよび演算を行なう。

実行結果、および、メモリ読みだしデータは P1-RESULT-REG に書き込まれ、W(レジスタ書き込み) ステージに送られる。

W ステージでは、通常の命令では結果をレジスタファイルに書き込む。この際パケットパイプライン (PP) 側との調停は不要である。パケット送出命令の場合、PO(パケット送出) ステージに要求を送る。PO ステージでは、W ステージと PW(パケットパイプラインのレジスタ書き込みステージ) から一方を選択し、パケット出力とする。

3.4 パケットパイプライン (PP) の構成

図 5にパケットパイプラインの構成を示す。

パケットパイプラインは、PD(パケットデコード) ステージの前に、PR (レジスタ読みだし) ステージ、PB(パケット受信) ステージ、PI(割り込み受け付け) ステージの三つが横に並ぶ構成となっている。PD ステージでは、割り込み処理、パケット処理、メモリ命令実行の順に優先して処理を行なう。

PI ステージは、割り込みを受け付けると、割り込みベクタ (5bit) にしたがって Interrupt Code Table から対応する命令を取り出す。一命令で実行が終了するものであれば、一命令分のパケット形式を生成し、P2-PI-REG に書き込む。複数の命令が必要な処理であれば、TRAP パケットを生成することにより、必要な処理を起動する。

PB ステージは、入力されたパケットのデスティネーションを Task Table と照合し、Task Table に登録されているタスクに対するパケットであれば、デスティネーションを書き換える。その他の場合は、パケットの持つデスティネーションをパケットデータの書き込まれるアドレスとする。

PF(命令フェッチ) ステージは P2-NPC(パケットパイプラインのプログラムカウンタ) で示されたアドレスから命

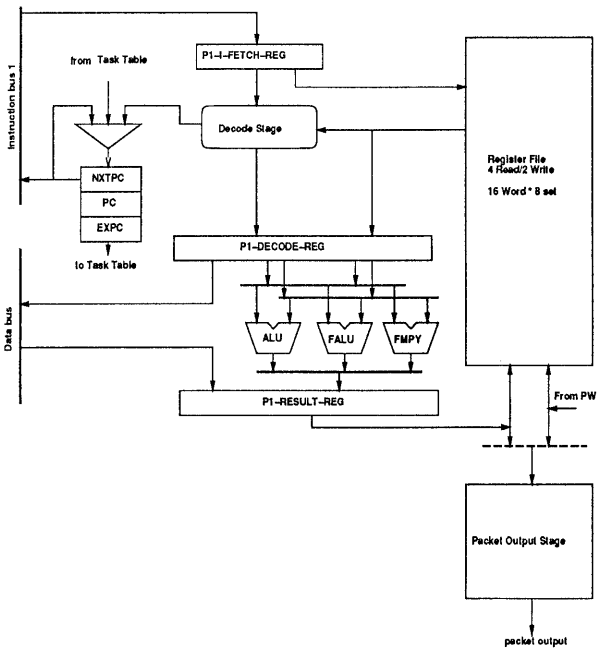


図 4: 命令パイプラインの構成

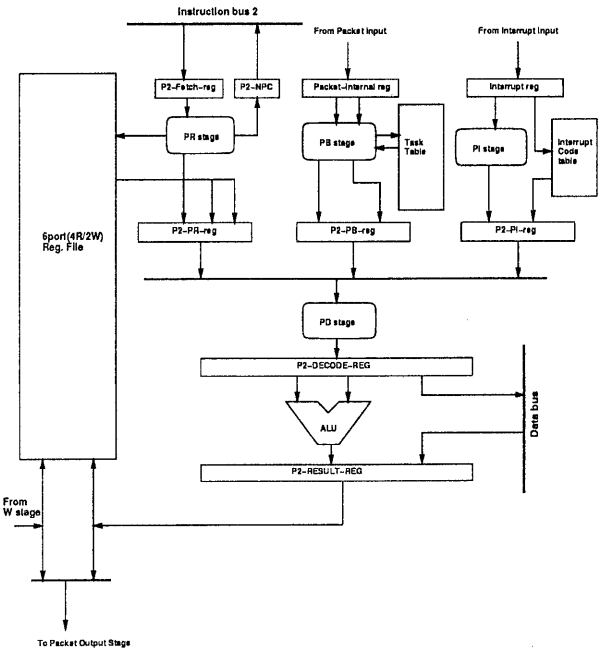


図 5: パケットパイプラインの構成

令をフェッチし、P2-FETCH-REG に書き込む。PR ステージでは、レジスタを読みだすとともに、次命令アドレスの計算を行なう。分岐命令に対するデコードのみ PR ステージで行なうことになる。レジスタ読みだし時の同期処理は IP と同様に行なう。この際、退避するのは P2-NPC(フェッチした命令のアドレス) と P2-RPC (レジスタ読みだし中の命令アドレス) の二語である。

PD ステージでは、パケットのデコードを行ない、PE(実行) ステージでの処理を定める。PE ステージは IP 側の E ステージと同様に、演算とメモリアクセスを行なうが、浮動小数点演算回路は持たない。PW(レジスタ書き込み) ステージも、IP の W ステージと同様の動作をする。

PO ステージ (図 6) は、外部へのパケット形式を合成するとともに、ネットワークとのハンドシェイクを行なう。

3.5 レジスタファイル

CODA では、CCR を持たない代わりにレジスタの各語に、ZERO フラグとキャリィビットを持たせる。この他に同期ビット (データの定義 / 未定義を示す) を持ち、一語の構成は、データ 32bit、フラグ 3bit となる。一つのコンテクストからアクセスされるレジスタは 16 語であり、16 語を 1 セットとして 8 コンテクスト分のレジスタセットを持つ。プリエンプション、同期、割り込みなどの際には、レジスタの退避を行なうことなくコンテクストを切替えることができる。

レジスタファイルには、読みだし 4 ポート / 書き込み 2 ポートのメモリセルを用いることにより、命令パイプラインとパケットパイプライン間の調停無しにレジスタにアクセスすることができる。パケットデータが直接レジスタに書き込まれることになるので、遠隔アクセスの手間を感じることなくプログラミングすることが可能である。

3.6 タスクテーブル

タスクテーブルは、タスク ID、タスク優先度 (デッドライン)、想定処理時間、レジスタセットアドレス、スタートアドレス (2 語) の七つ組で一つのタスクエントリを作り、8 タスク分のエントリを持つ。各タスクエントリには、同期待ち状態を示すフラグがあり、当該タスクが実行可能かどうかを示す。

タスクテーブル中のタスクは、タスク優先度にしたがって並べられており、先頭のタスクの優先度 (テーブル中の最高優先度が命令パイプラインで実行中のタスクの優先度よりも高くなった場合は、プリエンプション要求を発生する。また、タスク ID から使用中のレジスタセットアドレスを検索するための連想機能を有しており、外部からのパケットのデスティネーションをレジスタの物理アドレスに変換する時に用いられる。

3.7 割り込みコードテーブル

割り込みコードテーブルは、割り込みベクタ (5bit) から、操作、デスティネーション (値の書き込み / 読みだし先)、データを持ったパケット形式を生成するために用いられる。具体的には、32 パケット分のメモリであり、操作部には命令を、デスティネーション部にはアドレス (レジスタアドレス、もしくは、メモリアドレス) が、あらかじめ設定されている。データ部としては、即値を書き込んでおくことも可能であるが、センサ I/O ポートを指定しておくことにより外部機器からの一語入力が可能である。操作部にトラップ命令を設定しておくことにより、タスクテーブル内のタスクを起動することも可能である。

4 おわりに

CODA は、タスクスケジューリング、パケット転送、割り込み処理など全ての基本操作において、一貫した優先度 (デッドライン) 制御を行なうことにより、並列計算機でも実行時間の予測性を確保することを目指している。一方、CODA は並列計算機アーキテクチャとしてみた場合、高速な同期機構とコンテクスト切替え機構を持った RISC パイプラインと、効率良くパケット等を処理するマルチスレッド型のパイプラインの二本を有しており、それぞれで共通の命令セットを実行することができる。これにより、パケット処理や割り込み処理等によって、タスクの実行が妨げられないようにすることができ、RISC パイプライン上でのタスク実行時間の予測性を高めることができる。

現在、CODA はプロセッサ、ネットワークともに、ゲートアレイ LSI の設計中であり、1994 年にプロトタイプが完成する予定である。

謝辞

本研究を遂行するにあたり御指導、御討論いただいた弓場敏嗣情報アーキテクチャ部長ならびに計算機方式研究室の同僚諸氏に感謝致します。

参考文献

[1] 西田健次, 戸田賢二, 内堀義信, 坂井修一, 島田俊夫. 並列実時間計算機 coda の概要. In 電子情報通信学会技術研究報告, volume 90, pages 45–50, Oct. 1990.

[2] M. Ishikawa. Sensor Fusion System -Mechanism for Integration of Sensory Information-. *Journal of the Robotics Society of Japan*, 6(3):46–53, June 1989. (in Japanese).

[3] T. Shimada, K. Toda, and K. Nishida. Real-Time Parallel Architecture for Sensor Fusion. *Journal of Prallel and Distributed Computing*, 15(2):143–152, June 1992.

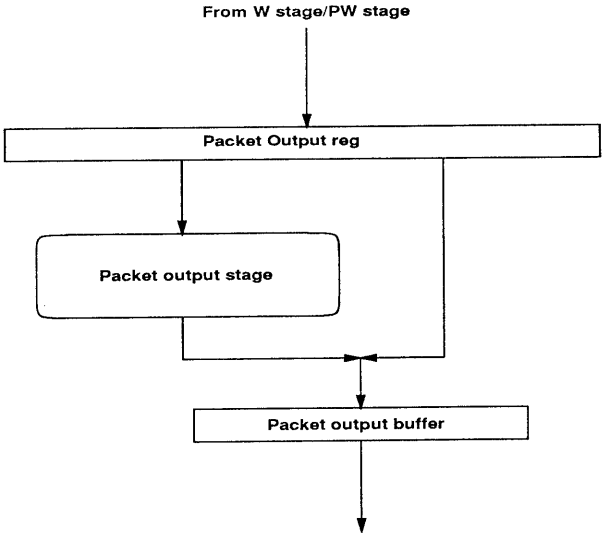


図 6: パケット出力部の構成

[4] 戸田賢二, 西田健次, 島田俊夫, 山口喜教. 実時間用並列計算機 CODA - 相互結合網 -. In 電子情報通信学会技術研究報告 *RTP 93*, Mar. 1993.